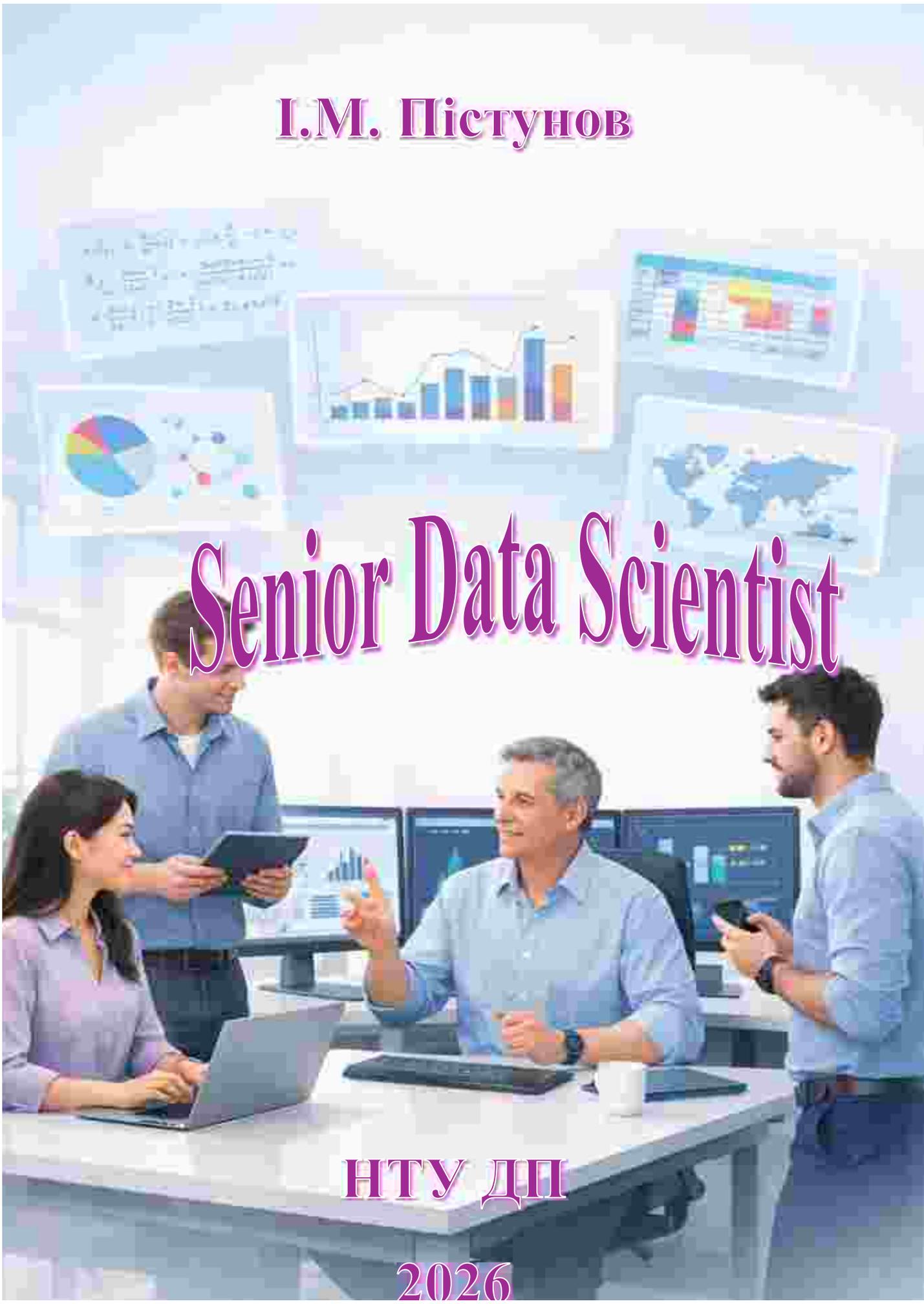


І.М. Пістунів

Senior Data Scientist



НТУ ДП

2026



МІНІСТЕРСТВО ОСВІТИ І
НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ
ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ДНІПРОВСЬКА
ПОЛІТЕХНІКА»

І.М. Пістунов

SENIOR DATA SCIENTIST

Навчальний посібник

Дніпро
НТУ «ДП»
2026

УДК 519.242:338.32.053.4
ПЗ4

*Затверджено вченою радою як навчальний посібник
для студентів спеціальності 051 Економіка
(протокол № __ від __.__.2025).*

Рецензенти:

Н.К. Васильєва, д-р екон. наук, проф., завідувач кафедри інформаційних систем і технологій Дніпровського державного аграрно-економічного університету;

К.Ф. Ковальчук, д-р екон. наук, проф., декан факультету економіки та менеджменту Інституту промислових та бізнес технологій Українського державного університету науки і технологій.

ПЗ4 Пістунов І.М. Senior data scientist: навч. посібник. Дніпро : НТУ «ДП», 2026. 259 с.

Подано поняття нового терміну senior data scientist, визначено напрям діяльності такого спеціаліста, визначено опис прийомів, яким володіє цей вчений. Викладення матеріалу зосереджено на методах і прийомах створення і впровадження генеративних рішень штучного інтелекту, з метою практичного застосування згенерованих розрахунків.

Показана можливість програмного підключення до різних моделей штучного інтелекту, таких як OpenAI, DeepSeek, Gemini та Grok з можливістю навчання генераційних моделей, таких як LLMs, GANs, VAEs, і Diffusion models.

Показано, як ці моделі можуть бути застосовані при вирішенні задач, що стоять перед економістами, таких як: прогнозування, статистичний аналіз даних, оптимізація, автоматизація прийняття рішень, тощо.

Посібник містить індивідуальні завдання, які дають поглиблене розуміння описаних методів та приймів.

Посібник базується на власних розробках автора, а також на літературних джерелах вітчизняних та зарубіжних авторів та на досвіді викладання дисциплін «Економіко-математичне моделювання», «Економічна кібернетика», «Методи прийняття управлінських рішень», «Мова програмування Python» за спеціальністю Економіка в Національному технічному університеті «Дніпровська політехніка».

УДК 519.242:338.32.053.4

© І.М. Пістунов, 2026

© НТУ« Дніпровська політехніка», 2026

ЗМІСТ

ВСТУП.....	6
Порядок засвоєння матеріалу.....	7
Набуті компетенції	11
Розділ 1. GİTHUB – БІБЛІОТЕКА ВЕРСІЙ ПРОГРАМ.....	12
1.1. Робота з GİTHUB.....	12
1.1.1. Встановлення Git.....	13
1.1.2. Список найкорисніших команд Git.....	14
1.1.3. Інтернет-сервіс GitHub	18
1.2. Файл менеджер Visual Studio Code (VS Code).....	26
1.2.1. Початок роботи з VS Code.....	31
1.2.2. Інтерфейс користувача.....	35
1.2.3. Доступ до терміналу з області панелі	38
1.2.4. Доступ до команд за допомогою панелі команд	39
1.2.5. Написання коду.....	41
1.2.6. Контроль джерел	42
1.2.7. Запуск і відладка коду програми.....	44
1.2.8. Онлайн інтерпретатори команд мови програмування.....	47
1.3. Приклад використання VS Code та GitHub	49
1.4. GitHub Actions.....	56
1.5. Індивідуальне завдання №1	61
Розділ 2. ПАЙТОН ЗІ ШТУЧНИМ ІНТЕЛЕКТОМ.....	63
2.1. Доступ до ШІ через Python	64
2.1.1. Отримання ключа API для ШІ.....	64
2.1.2. Обробка даних	75

2.1.3. Машинне навчання.....	80
2.1.4. Глибинне навчання.....	84
2.1.5. Готові, попередньо навчені моделі	89
2.1.5. Інтелектуальні сервіси	94
2.2. Збір, очищення та підготовка даних.....	96
2.3. Аналіз даних і прогнозування.....	100
2.4. Оптимізація рішень за допомогою ШІ.....	105
2.5. Основи теорії ігор та стратегій у ШІ.....	115
2.6. Використання Windows Task Sheduler для регулярного виконання програм	121
2.7. Індивідуальне завдання №2.....	123
Розділ 3. ПАКЕТ МАШИННОГО НАВЧАННЯ AZURE	128
3.1. Реєстрація в Microsoft Azure	128
3.2. Створення ресурсу	130
3.3. Деплоу, як функція AZURE.....	135
3.3.1. Деплой для веб-додатків, які працюють постійно і створені на Пайтоні або на HTML	135
3.3.2. Деплой для щоденного запуску скрипта.....	138
3.4. Створення середовища Azure Machine Learning	146
3.5. Створення власного агенту ШІ (Build an AI Agent)	149
3.5.1. ML-модель	150
3.5.2. Приклад використання Build an AI Agent	153
3.5.3. Готові для використання агенти.....	165
3.6. Інші сервіси Microsoft Azure	171
3.7. Індивідуальне завдання №3.....	175

Розділ 4. ПАКЕТИ ПРОГНОЗУВАННЯ AUTOML ТА MLFLOW	178
4.1. Порівняння можливостей двох систем прогнозування	178
4.2. Azure AutoML.....	181
4.3. MLflow (в Azure ML)	195
4.4. Індивідуальне завдання № 4.....	209
Розділ 5. ФРЕЙМВОРКИ PYTORCH ТА TENSORFLOW	214
5.1. Місце PyTorch та TensorFlow серед AI-фреймворків	217
5.2. Тензори у PyTorch і TensorFlow	224
5.3. PYTORCH	230
5.4. TENSORFLOW	238
5.5. Індивідуальне завдання № 4.....	248
ВИСНОВКИ.....	251
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	255
СЛОВНИК СПЕЦІАЛЬНИХ ТЕРМІНІВ	257

ВСТУП

Data Scientist – це спеціаліст, який є майстром обробки даних сфері машинного навчання Azure, Python (pandas, scikit-learn) і пакетів SDK Azure. У перше чергу це стосується прогнозування з постійним моніторингом зміни в часі економічних показників.

Senior Data Scientist – це, так би мовити, начальник над **Data Scientist`s**, що координує роботу групи спеціалістів з обробки даних, в галузі генеративного ШІ, визначаючи бізнес-сценарії, де генеративний ШІ можна ефективно застосовувати. Від нього вимагається також мати сильні навички програмування на Python, володіти принаймні одним фреймворком глибокого навчання, таким як PyTorch або TensorFlow –це найпопулярніші фреймворки з відкритим кодом для розробки та навчання моделей глибокого навчання та машинного навчання, кожен зі своїми сильними сторонами: TensorFlow від Google більш зрілий, масштабований та має потужні інструменти для продакшену та візуалізації, тоді як PyTorch від Meta (Facebook) відомий своєю гнучкістю, простотою використання, тісною інтеграцією з Python та перевагою в академічних дослідженнях.

Практично, це означає уміння будувати економіко-математичні моделі економічних процесів, знаходити їх оптимальні значення, прогнозувати хід економічних процесів і видавати результати розрахунків у простій зрозумілій для керівництва формі.

Тобто, все то, що студенти спеціальності «Економіка» вивчали у таких навчальних курсах як: «Економічна статистика», «Економіко-математичне моделювання», «Прогнозування соціально-економічних процесів», «Мова програмування Пайтон», «Аналіз даних з використанням Пайтон».

Всі ці науки описані у навчальних посібниках автора цього посібника, з якими ви можете ознайомитися за посиланнями:

[Пістунов І.М. Економіко-математичне моделювання. Дніпро: НТУ «ДП», 2025. 202 с](#)

Пістунов І.М. Пайтон для економістів. навч. посібник. Дніпро : НТУ «ДП», 2025. 231 с.

Тобто, все це вже уміють випускники спеціальності «Економіка».

Застосування штучного інтелекту (ШІ) полегшує пошук інформації, прискорює формування таблиць даних, забезпечує автоматичне поповнення цих таблиць новими даними по мірі їх зміни у часі, автоматизує розрахунки прогнозів, пропонує зручні форми представлення результатів обчислення.

Звертання до ШІ через Пайтон забезпечує можливість для дослідників доступ до потужних бібліотек цієї мови програмування та дозволить обробляти таблиці даних необмеженого розміру.

Набуття цих навичок збільшить ваш рейтинг на сайті [Glassdoor](#).

Гласдор – це вебсайт і спільнота для пошуку роботи та кар'єри, де працівники (теперішні й колишні) анонімно залишають відгуки про компанії, діляться інформацією про зарплати, процеси співбесіди та культуру на робочому місці, а також шукають вакансії, що допомагає іншим ухвалювати свідомі кар'єрні рішення.

Порядок засвоєння матеріалу

У тесті посібника подані індивідуальні завдання, які студенти виконують під час практичних занять та у вільний від навчання час.

Кожне завдання має на меті поглибити розуміння щодо порядку використання тих чи інших елементів штучного інтелекту, реалізованих на різних сайтах.

Завдання виконується у вигляді звіту.

Формат тексту: береги листа – кругом 2 см, шрифт – Times new Roman, кегль – 14, вирівнювання по ширині, абзац – 1.25 см, інтервал – 1,5 рядки, кожен розділ починається з нової сторінки, заголовки жирним і по центру.

Структура звіту:

ВСТУП
ЗМІСТ
ОСНОВНА ЧАСТИНА
ВИСНОВКИ
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ
ДОДАТКИ

Титульний лист має форму, подану нижче.

Звіт з індивідуального завдання має супроводжуватися елементами антивних вікон комп'ютера, які підтверджують успішність виконаного завдання.

До цих рисунків потрібно додати пояснення у вигляді виносок.

Документи у форматі *.DOC заповнюються і редагуються текстовим редактором Word.

Сайти зі штучним інтелектом майже завжди:

- блокують VPN / мобільні IP / IP з України;
- мають жорсткий антибот + антифрод;
- часто ламаються саме на сторінці реєстрації.

Якщо у вас браузер Google Chrom тоді тимчасово вимкнути:

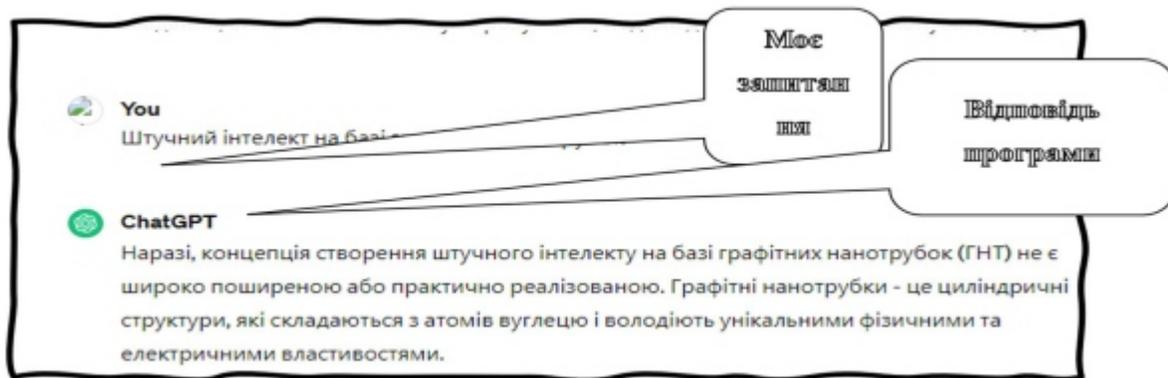
- Settings → Privacy and security: Block third-party cookies →
- Security: Enhanced protection → Standard
- Розширення: uBlock / AdBlock, Privacy Badger, NoScript

Якщо у вас браузер Microsoft Edge тоді тимчасово вимкнути:

- Settings → Privacy, search, services
- Tracking prevention → Basic
 - Cookies → увімкнути
- Allow third-party cookies: Microsoft Family Safety (якщо увімкнено – блокувати).

Потім увімкнете, коли отримаєте доступ до сайту після реєстрації.

У звіті використовуйте копії екрану, для пояснення ходу роботи, подавайте виноски, які дозволяють зрозуміти використання окремих елементів інтерфесу, наприклад



МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ДНІПРОВСЬКА ПОЛІТЕХНІКА»

Кафедра економіки та економічної кібернетики

Індивідуальне завдання №1
На тему: «Робота з ChatGPT»

Розробив(ла) ст. гр. 000-23-1

Косач-Квітка Л.П.

Якщо при виконанні завдання необхідно розробити програму на мові програмування Пайтон, то:

1. Студент має ознайомитися з текстом чергового пункту посібника, увімкнувши інтерпретатор Пайтон (IDLE).
2. Побачивши приклад, його потрібно скопіювати до інтерпретатора і перевірити порядок роботи.

3. Варто спробувати змінити текст прикладу, для отримання іншого результату, навіть, якщо це призведе до збою в програмі.

4. Кожна змістовна частина посібника супроводжується індивідуальним завданням розробки певної програми.

5. На початку тексту програми треба вмістити дані щодо номеру завдання, назви завдання, номеру групи та імені студента. Наприклад

```
# Лабораторна робота №2  
# студ. гр. 051-99-1  
# Косач-Лариси Петрівни
```

6. Текст кожної програми має супроводжуватися коментарями, що описують той чи інший етап розрахунку. Наприклад

```
print('Привіт усім', 12**14); print(12.45678**123.54677) # Тут пише коментар  
# Тут пише коментар
```

7. Файли з текстом програм треба розміщати у хмарному сховищі, вказаному викладачем, що проводить практичні заняття.

8. Файли з текстом програм у своїй назві мають містити номер завданні та ім'я студента і номер групи.

9. Сума оцінок по індивідуальним завдання складе результат вивчення цієї мови програмування.

10. Якщо практична робота після перевірки не зарахована, треба виправити помилки згідно з зауваженнями викладача. Доопрацьована практична робота надається для повторної перевірки разом з першим варіантом.

11. Студент, що не виконав практичні роботи, до іспиту не допускається.

12. Кожну програма потрібно розробляти з урахуванням некомпетентності оператора (захист від «дурня»), тому вона має містити підказки про момент введення даних, перевірку на правильність типу введених даних, повідомлення оператора про невірні дані, спробу знову ввести дані, опис результатів розрахунків.

13. Відсутність вимог з п. 12 автоматично зменшує оцінку до рівня не вище «добре».

Набуті компетенції

1. Уміти виконати економічний розрахунок будь якої складності.
2. Уміти проваджувати сучасні методи та моделі рішення економічних задач.
3. Уміти розраховувати ризики фінансових ринків та інвестиційних інструментів, на основі розумінні принципів інженерії та застосуванні методів математичного та статистичного аналізу.
4. Уміти отримувати дані з сайтів Інтернету, проводити їх статистичну розробку.
5. Уміти виводити дані у текстових форматах, а також у форматі Excel.
6. Уміти створювати автономні програми, які працюватимуть без підтримки інтерпретатора Пайтон.
7. Використовувати аналітичний та інструментарій мови програмування Python для обґрунтування економічних рішень.
8. Використовувати фреймворки PyTorch або TensorFlow для розробки та навчання моделей глибокого навчання та машинного навчання.
9. Використовувати пакети машинного навчання Azure (SDK Azure) та Python (pandas, scikit-learn).
10. Працювати з прогностичними пакетами AutoML та MLflow.

Розділ 1.

GITHub

БІБЛІОТЕКА ВЕРСІЙ ПРОГРАМ

У розділі описуються програма VS Code та Інтернет-сервіс GitHub, що працюють сумісно для прискорення процесу програмування.

GitHub – це платформа для управління версіями коду та спільної роботи над проєктами, побудована на основі системи контролю версій Git. Потрібна при розробці комплексів програм, які постійно модифікуються через зміну до їх вимог. Дозволяє не заплутатися у різних версіях і до того ж, мати постійний зв'язок з цими версіями через файл-менеджер на вашому комп'ютері.

Щоб зручно користатися цим сервісом, варто вивчити інші програми та сервіси, що полегшують процес програмування.

1.1. Робота з GitHub

Перед звертання до GitHub, необхідно на власному комп'ютері встановити програму Gi, яка зберігає історію змін, дозволяє повертатися до попередніх версій і працювати в команді без втрати коду.

Основні функції програми Git:

- зберігати всі зміни в проєкті;
- не боятися зламати код;
- працювати з гілками;
- об'єднувати роботу кількох людей;
- відстежувати, хто і що змінив.

1.1.1. Встановлення Git

1. Завантаження інсталятора з офіційного сайту Git – <https://git-scm.com/>
починається з вибору типу операційної системи



Скачайте файл і запустіть його на власному комп'ютері. Запустіть файл. Файл швидко інсталується на диск C: і відкриється вікно терміналу, через який і можна подавати команди цій програмі.

2. Налаштування Git. Після встановлення налаштуйте глобальні налаштування користувача:

```
git config --global user.name "Ваше ім'я"
```

```
git config --global user.email "ваша_почта@example.com"
```

3. Ініціалізація репозиторію. Створити новий репозиторій:

```
git init
```

4. Щоб клонувати існуючий репозиторій з GitHub:

```
git clone URL_репозиторію
```

5. Додавання файлів до індексу. Додавання файлів до індексу для підготовки до коміту:

```
git add <ім'я_файлу>
```

```
# або додайте всі файли
```

```
git add .
```

6. Коміт (фіксація) змін. Збереження змін в репозиторії:

```
git commit -m "Опис вашого коміту"
```

7. Переглянути статус вашого репозиторію:

git status

8. Перегляд історії комітів

git log

9. Відправка змін на GitHub

git push origin main

10. Отримати останні зміни з віддаленого репозиторію:

git pull origin main

11. Створення та перемикання гілок

Створення нової гілки

git branch нова_гілка

Перемикання на нову гілку

git checkout нова_гілка

Або створення і перемикання одразу

git checkout -b нова_гілка

12. Злиття змін з однієї гілки в іншу:

git checkout main

git merge нова_гілка

13. Видалення гілок

git branch -d нова_гілка

1.1.2. Список найкорисніших команд Git

Команди Git мають вирішальне значення для ефективної співпраці та управління проектами. У цій статті ми розглянемо список важливих команд Git, як-от команди git для натискання, команда git commit, команда git pull і команда git push тощо, які допоможуть покращити робочий процес і оптимізувати продуктивність. Це список команд Git, які можна часто використовувати в Git.

1. git help – Зверніться до розділу довідки Git щодо різних команд та інших помилок.

2. config git – Щоб налаштувати основні конфігурації Git, як-от ваше ім'я та електронна адреса.

3. git config --global user.name “ ” – Встановлює значення конфігурації для вашого імені користувача на git.

```
git config --global user.name “Ashish Madaan”
```

4. git config --global user.email ” “ – Встановлює значення конфігурації для електронної пошти користувача на git.

```
git config --global user.email ashishmadaan6@gmail.com
```

5. git config --global color.ui – Щоб побачити різні кольори в командному рядку для різних результатів. Наприклад,

```
git config --global color.ui true
```

6. mkdir – Створить каталог, якщо він не був створений спочатку.

7. cd – Щоб зайти всередину каталогу та попрацювати над його вмістом.

8. git init – Щоб створити для нас локальний репозиторій git у папці магазину. Це допоможе керувати командами git для цього конкретного сховища.

9. status git – Щоб побачити, що змінилося після останнього коміту. Він показує всі файли, які були додані та змінені та готові до фіксації, а також файли, які не відстежуються.

10. git add Readme.txt – Щоб додати файл, наприклад, Readme.txt до робочої області для відстеження його змін.

11. git commit -m “ ” – Щоб зафіксувати наші зміни (зробити знімок) і надати повідомлення, яке потрібно запам'ятати для подальшого використання. В лапках ваш текст з описом змін.

12. log git – Щоб перевірити історію комітів для нашої довідки.

13. git add – Щоб додати певний список файлів до робочої області.

14. git add --all – Щоб додати всі файли поточного каталогу до проміжної області.

15. git add *.txt – Щоб додати всі текстові файли поточного каталогу до проміжної області.

16. git add docs/*.txt – Щоб додати всі текстові файли певного каталогу (документів) до проміжної області.

17. git add docs/ – Щоб додати всі файли в певному каталозі (документи) до робочої області.

18. git додати «*.txt» – Щоб додати текстові файли всього проекту до проміжної області.

19. git diff – Щоб дізнатися, які зміни ви внесли з часу останнього коміту.

20. license git reset head – Щоб скасувати проміжне розміщення файлу, доданого в проміжну область.

21. git checkout – license – Видалити всі зміни з часу останнього коміту файлу.

22. git commit -a -m “ ” – Щоб додати будь-який із наших відстежуваних файлів до робочої області та зафіксувати їх, надавши повідомлення, яке потрібно запам'ятати.

23. git reset –soft HEAD^ – Щоб скасувати останню фіксацію та перенести файл у проміжну область.

24. git reset –hard HEAD^ – Щоб скасувати останній коміт і також вид

25. git reset –hard HEAD^^ – Щоб скасувати останні 2 коміти та всі зміни.

26. git remote add origin – Ці команди створюють закладку, яка вказує на те, що цей конкретний пульт посилається на цю URL-адресу. Цей віддалений використовуватиметься для отримання будь-якого вмісту з каталогу та надсилання нашого локального вмісту на глобальний сервер.

27. git remote add <адреса> – Щоб додати нові пульти до нашого локального сховища для певної адреси git.

28. git delete rm – Щоб видалити пульт дистанційного керування з нашого локального сховища.

29. git push -u origin master – Для надсилання всього вмісту нашого локального сховища, що належить до головної гілки, на сервер (Глобальний репозиторій).

30. clone git <https://github.com/madaan123/MyAlgorithms.git> – Щоб клонувати або створити локальну копію глобального репозиторію у вашій системі (команда `git clone` завантажує репозиторій і створює віддалений іменованій джерело, який можна перевірити за допомогою команди – `git remote -v`).

31. Testing branch git – Щоб створити нову гілку під назвою `Testing`.

32. branch git – Щоб побачити всі наявні та поточні гілки, над якими ми працюємо.

33. git checkout Testing – Щоб перейти до гілки з іменем, наприклад, `Testing` з головної гілки.

34. ls – Щоб переглянути каталоги та файли в поточному каталозі.

35. ls -la – Щоб побачити приховані каталоги та файли в поточному каталозі.

36. Testing git merge – Щоб об'єднати гілку з іменем, наприклад `Тестування`, з головною гілкою.

37. git branch -d Testing – Щоб видалити гілку, наприклад `Testing`.

38. git checkout -b admin – Щоб створити нового адміністратора гілки та встановити його як поточну гілку.

39. git гілка -r – Подивитися на всі віддалені гілки, наприклад `r`.

40. Гілка git -D Testing – Для примусового видалення гілки, наприклад `D`, без комітів.

41. tag git – Щоб переглянути список доступних тегів.

42. git checkout v0.0.1 – Щоб встановити поточний тег на `v0.0.1`.

43. tag git -a v0.0.3 -m «версія 0.0.3» – Щоб створити новий тег, наприклад «версія 0.0.3».

44. git push –теги – Щоб надсилати теги до віддаленого сховища.

45. git fetch – Щоб отримати будь-які зміни з глобального сховища до поточного сховища.

46. git stash – Щоб перемістити проміжні файли до сховища, яке є в проміжній області.

47. git stash pop – Щоб повернути файли, які присутні в схованці.

48. git stash clear – Щоб очистити сховану папку.

49. git rebase

Git rebase виконує три завдання

1. Перемістіть усі зміни до шаблону, які не знаходяться в джерелі/основі, до тимчасової області.

2. Виконайте всі головні коміти походження.

3. Запускайте всі коміти у тимчасовій області поверх нашого основного по одному, щоб уникнути злиття комітів.

50. git –версія – використовується для показу поточної версії Git.

1.1.3. Інтернет-сервіс GitHub

Це сайт на якому можна розміщати свої коди програм.

Адреса <https://github.com/new>

На цьому сайті легко зареєструватися через Google і одразу вам пропонується створити власний репозиторій (сховище) вашого проекту. Надаєте йому назву (Repository name) та опис вмісту (Description). Обираєте доступність інших до ваших кодів (Public) чи заборону доступу (Private).

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere?
[Import a repository.](#)

Required fields are marked with an asterisk (*).

Owner * Repository name *
 pistunovi /

Great repository names are short and memorable. Need inspiration? How about [supreme-octo-happiness](#) ?

Description (optional)

-  **Public**
Anyone on the internet can see this repository. You choose who can commit.
-  **Private**
You choose who can see and commit to this repository.

Initialize this repository with:

Add a README file

This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore

.gitignore template: None ▾

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license

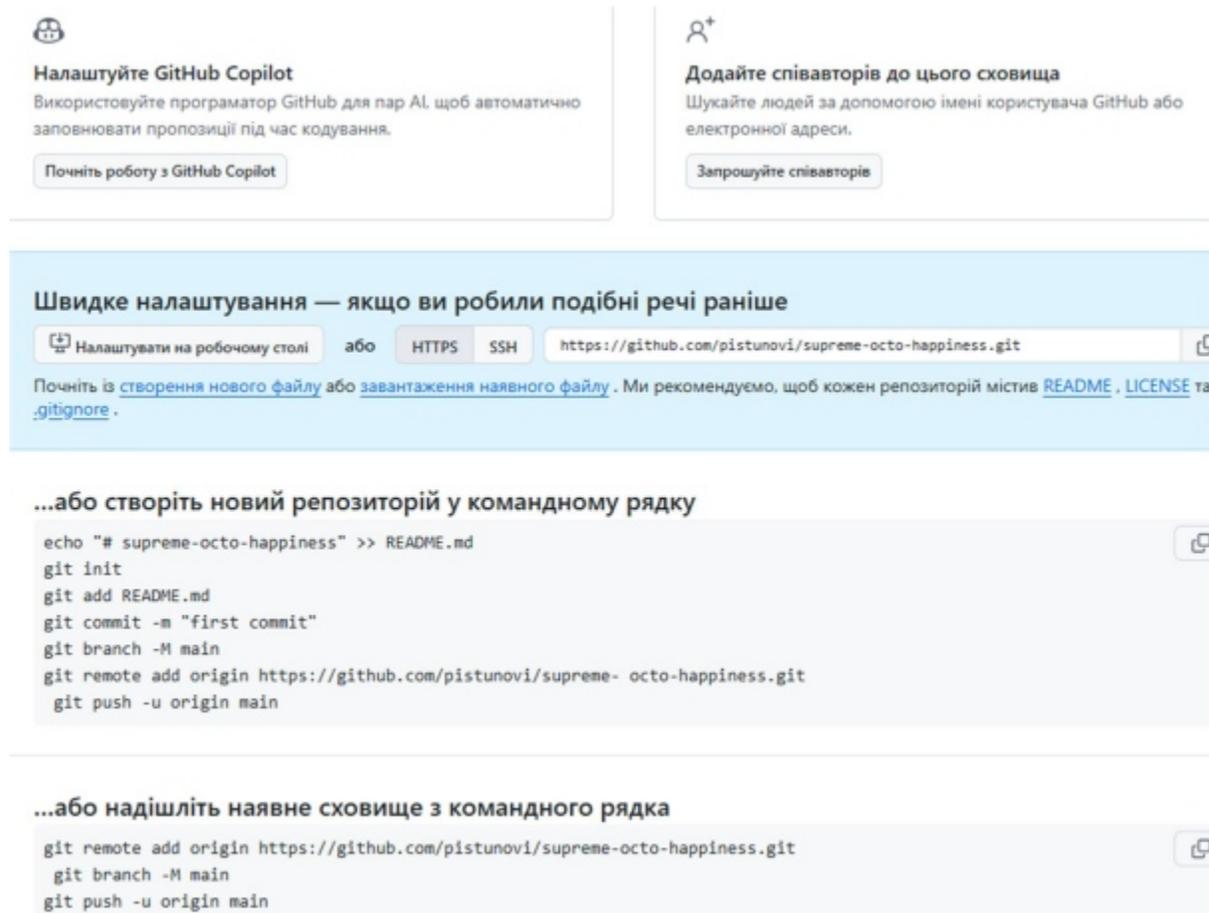
License: None ▾

A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

 You are creating a public repository in your personal account.

Все готово до прийняття ваших файлів з кодами програм.

Тепер треба створити папку з таким же ім'ям на вашому комп'ютері. Для цього треба виконати дії у командному рядку (Терміналі). Наступна інструкція з'являється одразу, як ви натиснете кнопку «Створити репозиторій».



Налаштуйте GitHub Copilot
Використовуйте програматор GitHub для пар AI, щоб автоматично заповнювати пропозиції під час кодування.
[Почніть роботу з GitHub Copilot](#)

Додайте співавторів до цього сховища
Шукайте людей за допомогою імені користувача GitHub або електронної адреси.
[Запрошуйте співавторів](#)

Швидке налаштування — якщо ви робили подібні речі раніше
Налаштувати на робочому столі або **HTTPS** **SSH** `https://github.com/pistunovi/supreme-octo-happiness.git`
Почніть із [створення нового файлу](#) або [завантаження наявного файлу](#). Ми рекомендуємо, щоб кожен репозиторій мистив [README](#), [LICENSE](#) та [.gitignore](#).

...або створіть новий репозиторій у командному рядку

```
echo "# supreme-octo-happiness" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/pistunovi/supreme-octo-happiness.git
git push -u origin main
```

...або надішліть наявне сховище з командного рядка

```
git remote add origin https://github.com/pistunovi/supreme-octo-happiness.git
git branch -M main
git push -u origin main
```

Тут у верхньому рядку є вже адреса вашого репозиторію, в який вказується логін власника та назва репозиторію.

У прикладі це pistunovi та supreme-octo-happiness.git.

Нижче подані основні команди операцій з вашим репозиторієм у GIT.

Тобто, паралельно можна у GIT тримати декілька варіантів однієї і тієї самої програми.

Співпраця та контроль версій важливі для розробки програмного забезпечення. GitHub став важливою платформою для розробників, що забезпечує безперебійну командну роботу та ефективне управління проектами. У цій статті ми познайомимо вас із GitHub, пояснимо, що це таке, як він працює та чому це обов'язковий інструмент для розробників.

Що таке контроль версій?

Система під назвою «Контроль версій», яку іноді називають «Контроль джерел» або «Контроль версій», відстежує зміни, внесені у файл або групу файлів з часом, щоб ви могли отримати певні версії пізніше. Хоча його можна застосовувати в будь-яких обставинах, коли створюється кілька версій чогось і, можливо, потрібно відстежувати та відкликати їх, найчастіше він використовується в розробці програмного забезпечення.

GitHub – це веб-платформа, яка використовує Git, систему контролю версій, щоб допомогти розробникам керувати та відстежувати зміни у своєму коді. Це дозволяє кільком людям співпрацювати над проектом, відстежувати зміни та робити внесок у код з будь-якої точки світу. GitHub пропонує як безкоштовні, так і платні плани, призначені як для окремих осіб, так і для великих організацій.

Ключові характеристики GitHub

1. Контроль версій

Основні функції GitHub базуються на Git, що дозволяє відстежувати зміни у вашому коді з часом. Це означає, що ви завжди можете повернутися до попередньої версії, якщо щось піде не так, порівняти різні версії та зрозуміти історію свого проекту.

2. Репозитарії

Репозиторій (або сховище) – це центральне місце, де зберігаються всі файли проєкту. Кожне сховище може містити кілька файлів і папок і відстежує історію кожної внесеної зміни. Репозиторії можуть бути публічними (доступними для всіх) або приватними (з обмеженим доступом).

3. Відділення

Гілки є важливою функцією GitHub, яка забезпечує паралельну розробку. Ви можете створити гілку для роботи над новою функцією або виправити помилку, не впливаючи на основну кодову базу. Коли ваші зміни будуть готові, ви можете об'єднати гілку назад у головну гілку.

4. Запити на отримання

Запити на витягування – це спосіб запропонувати зміни до сховища. Коли ви надсилаєте запит на отримання, ви просите розробників проєкту переглянути та об'єднати ваші зміни в основну кодову базу. Ця функція сприяє співпраці та гарантує якість коду через рецензування.

5. Проблеми та управління проєктами

GitHub надає інструменти для відстеження помилок, покращень та інших завдань за допомогою функції «Проблеми». Ви можете створювати завдання, призначати їх членам команди та відстежувати їхній прогрес. GitHub також пропонує дошки проєктів для розширеного керування проєктами.

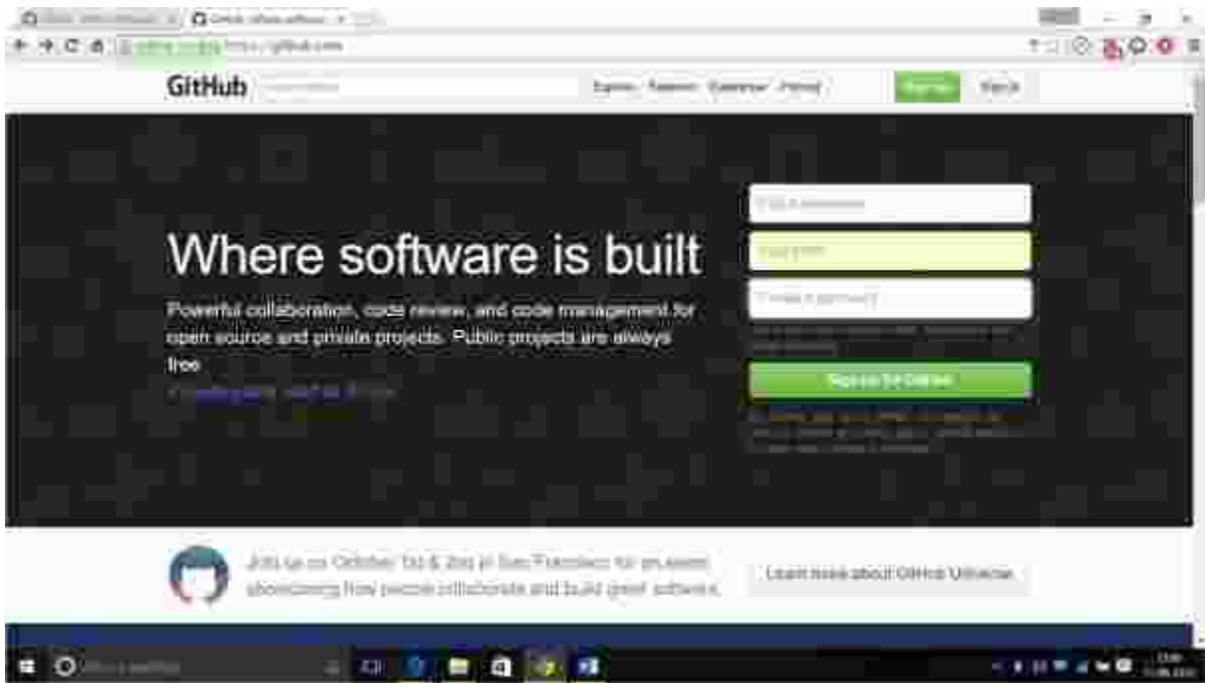
6. Дії та автоматизація

GitHub Actions дозволяє автоматизувати робочі процеси, такі як виконання тестів або розгортання коду, безпосередньо з вашого сховища. Ця функція підвищує продуктивність і забезпечує узгодженість процесів розробки.

Початок роботи з GitHub

1. Створення облікового запису GitHub

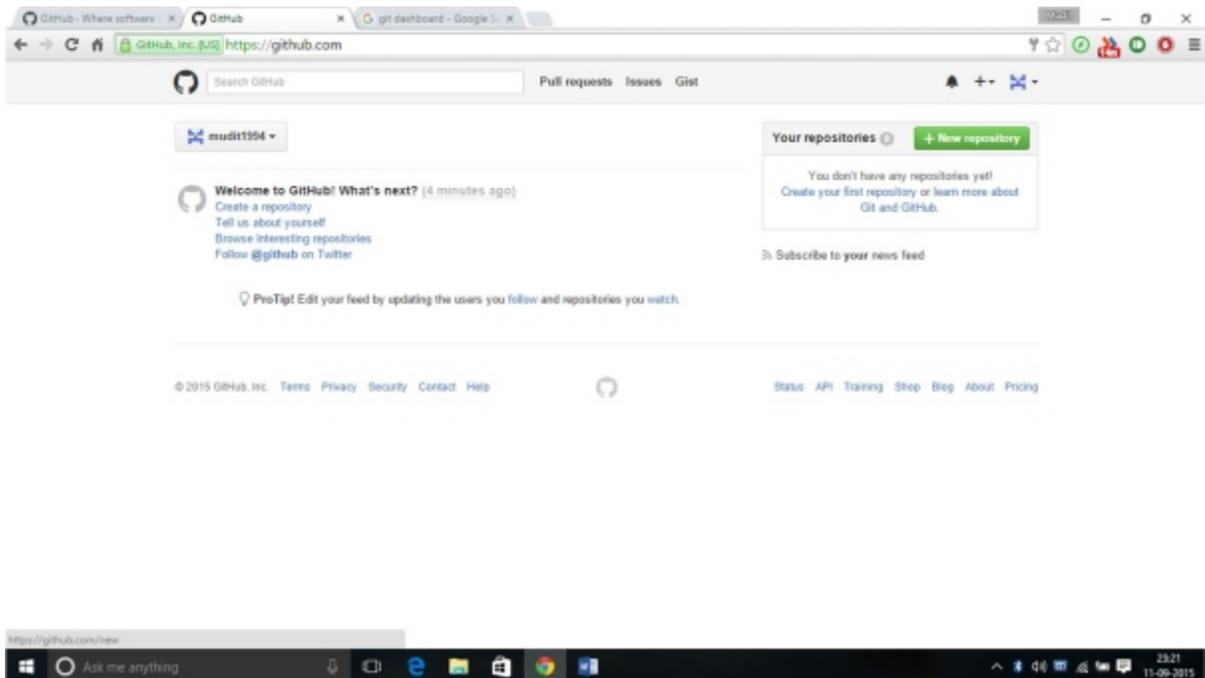
Крок 1. Перейдіть на github.com і введіть необхідні облікові дані користувача, які запитуються на сайті, а потім натисніть кнопку «Зареєструватися для GitHub».



3. Створення репозиторію

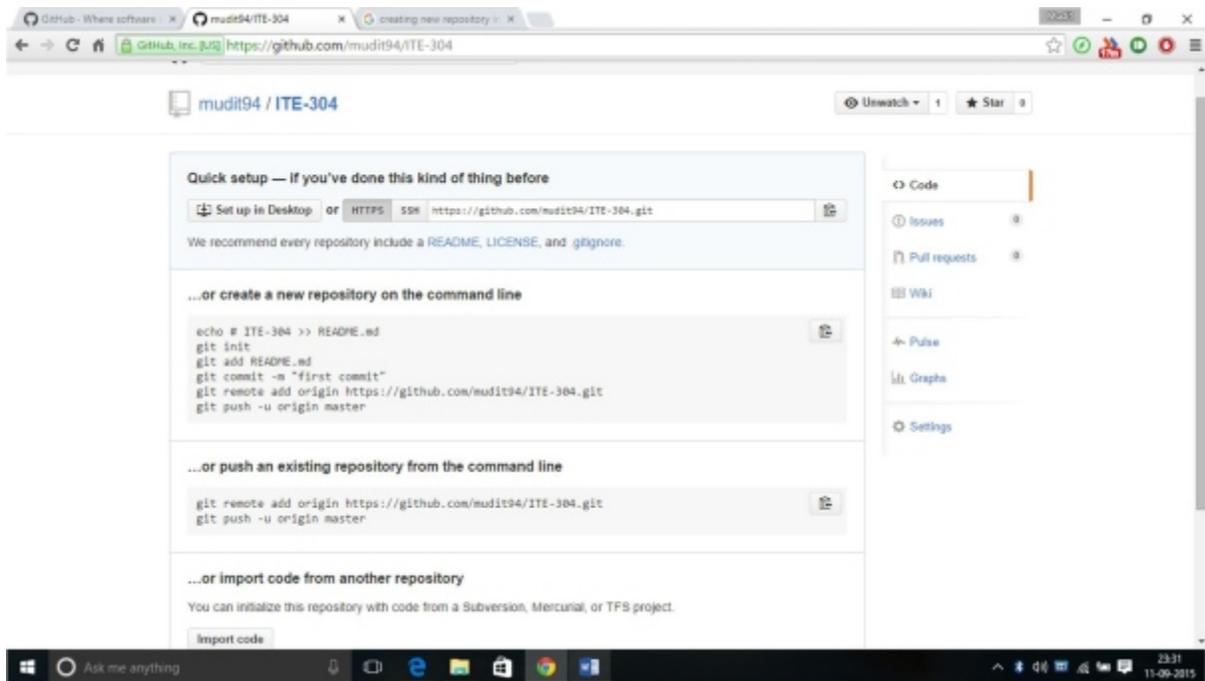
Щоб створити новий репозиторій на GitHub, виконайте такі дії:

Крок 1. Потім натисніть «Завершити реєстрацію». Обліковий запис створено. Користувач автоматично перенаправляється на вашу інформаційну панель.



Щоб створити новий репозиторій на GitHub, виконайте такі дії:

1. Перейдіть на [GitHub](#) і увійдіть.
2. Натисніть значок «+» у верхньому правому куті та виберіть «Новий репозиторій».
3. Введіть назву та опис сховища.
4. Виберіть, чи буде сховище загальнодоступним чи приватним.
5. Натисніть «Створити репозиторій».



Як завантажити існуючий репозиторій на GitHub

- У системі повинен бути встановлений git, якщо не встановлено [git](#) . Переконайтеся, що під час встановлення вибрано параметр «Запустити git із командного рядка Windows». В іншому випадку відкрийте git bash замість кроку 2.
- Відкрийте термінал (для користувачів Mac) або командний рядок (для користувачів Windows і Linux).
- Змініть поточний робочий каталог на свій локальний проєкт
- Ініціалізуйте локальний каталог як репозиторій git різними способами, як описано на зображенні.

git init

- У каталозі, який за замовчуванням приховано , створюється нова папка .git .

- Додайте файли в нове локальне сховище. Це готує їх для першого коміту.

git add

#Додає файли в локальне сховище та готує їх для фіксації. Щоб скасувати сцену файлу, скористайтеся

git reset HEAD YOUR_FILE

Зафіксуйте файли, які ви розмістили у своєму локальному сховищі.

git commit -m 'Перша фіксація'

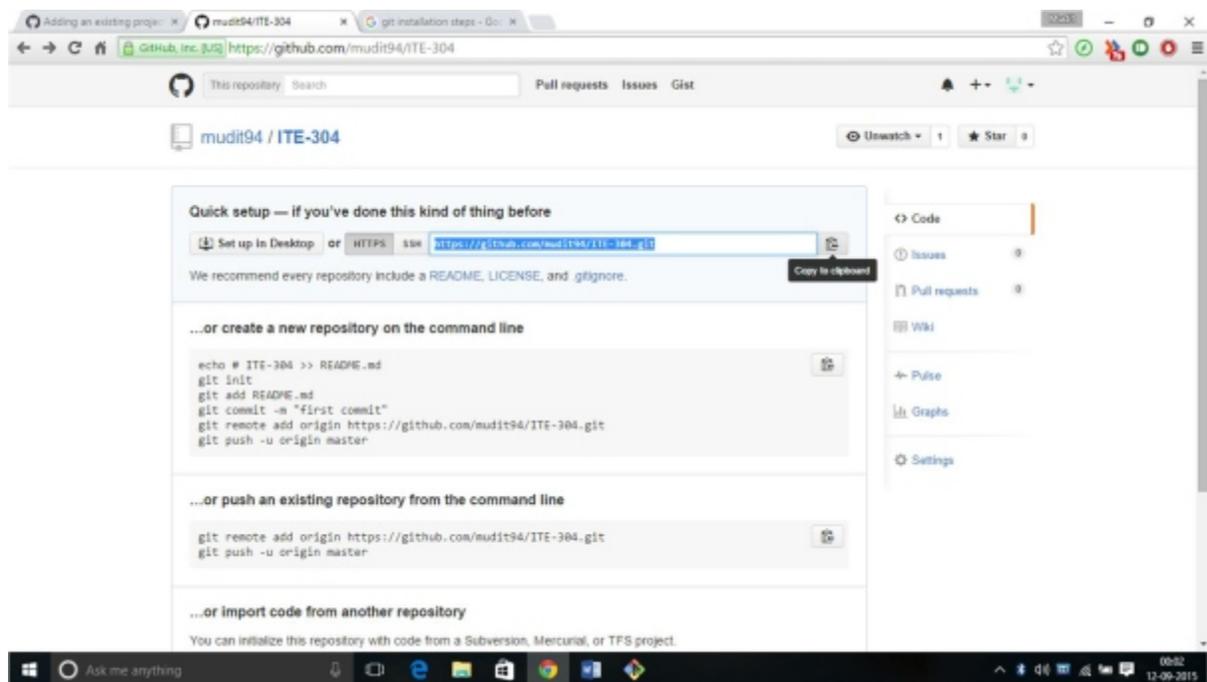
Фіксує відстежувані зміни та готує їх для надсилання у віддалений репозиторій.

Щоб видалити цю фіксацію та змінити файл, скористайтеся

git reset --soft HEAD~1

, зафіксуйте та додайте файл знову.

У верхній частині сторінки швидкого налаштування репозиторію GitHub клацніть зображену піктограму та скопіюйте URL-адресу віддаленого сховища.



У командному рядку додайте URL-адресу віддаленого сховища , куди буде надіслано ваше локальне сховище.

git віддалений URL-адреса віддаленого сховища джерела

Підключається до віддаленого сховища

git remote -v

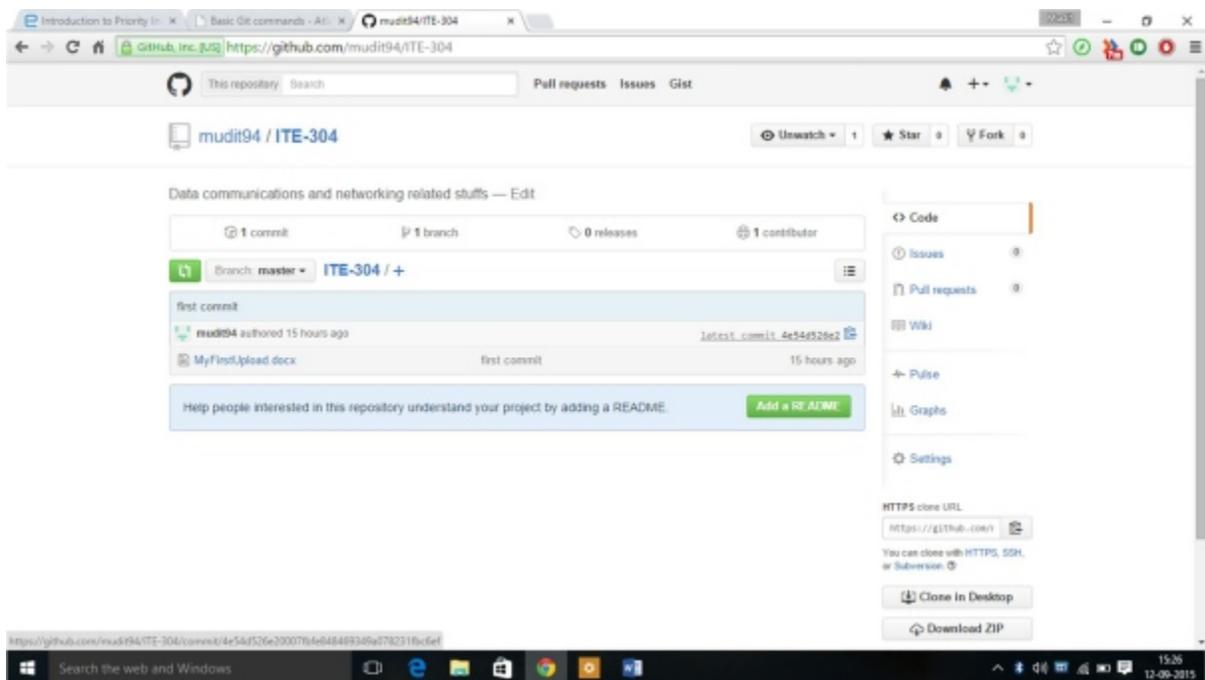
Перевіряє новий віддалений URL

Надішліть зміни у вашому локальному сховищі на [GitHub](#) .

git push origin master

Надсилає зміни у вашому локальному сховищі до віддаленого сховища, яке ви вказали як джерело.

І ось ви...



Ви можете завантажити зміни з віддаленого репозиторію в локальний за допомогою команди:

git pull

Різниця між Git і GitHub

Git	GitHub
Git — це розподілена система контролю версій (DVCS).	GitHub — це веб-платформа, яка допомагає git розміщувати свої репозиторії.
Git відстежуватиме зміни в проектах, внесені розробниками.	Github допоможе вам вирішити проблему, запит на отримання, перевірку коду тощо.
Git працює в командному рядку (CLI).	GitHub – це графічний веб-інтерфейс.
Git працює на локальній машині розробників, за допомогою якої ви можете вносити зміни, створювати гілки, об'єднувати код.	GitHub – це служба віддаленого хостингу, за допомогою якої інші розробники можуть отримати доступ і отримати код відповідно до потреб.

1.2. Файл-менеджер Visual Studio Code (VS Code)

Visual Studio Code (VS Code) – це безкоштовний, потужний та популярний редактор коду від Microsoft для розробки на різних платформах (Windows, macOS, Linux), який підтримує безліч мов програмування, має вбудовану відладку, інтеграцію з GitHub, підсвічування синтаксису та величезну екосистему розширень для додавання функціоналу. Він є «легким», але гнучким інструментом, що дозволяє налаштувати його під себе і працювати з веб- і хмарними застосунками, а також іншими типами проектів.

Основні можливості VS Code:

- **Підтримка мов:** Вбудована підтримка JavaScript, TypeScript, Node.js, а також розширення для C++, C#, Java, Python, PHP, Go тощо.
- **Розумний код (IntelliSense):** Автоматичне доповнення коду, підказки та рефакторинг.
- **Вбудований відладчик:** Дозволяє знаходити та виправляти помилки.
- **Інтеграція з Git:** Керування версіями прямо в редакторі (commit, push, pull).
- **Кастомізація:** Безліч тем, комбінацій клавіш та налаштувань.
- **Розширення (Extensions):** Можливість додавати нові функції для будь-яких потреб через магазин розширень.
- **Інтегрований термінал:** Виконання команд без виходу з редактора.
- **Кросплатформність:** Працює на Windows, macOS та Linux.

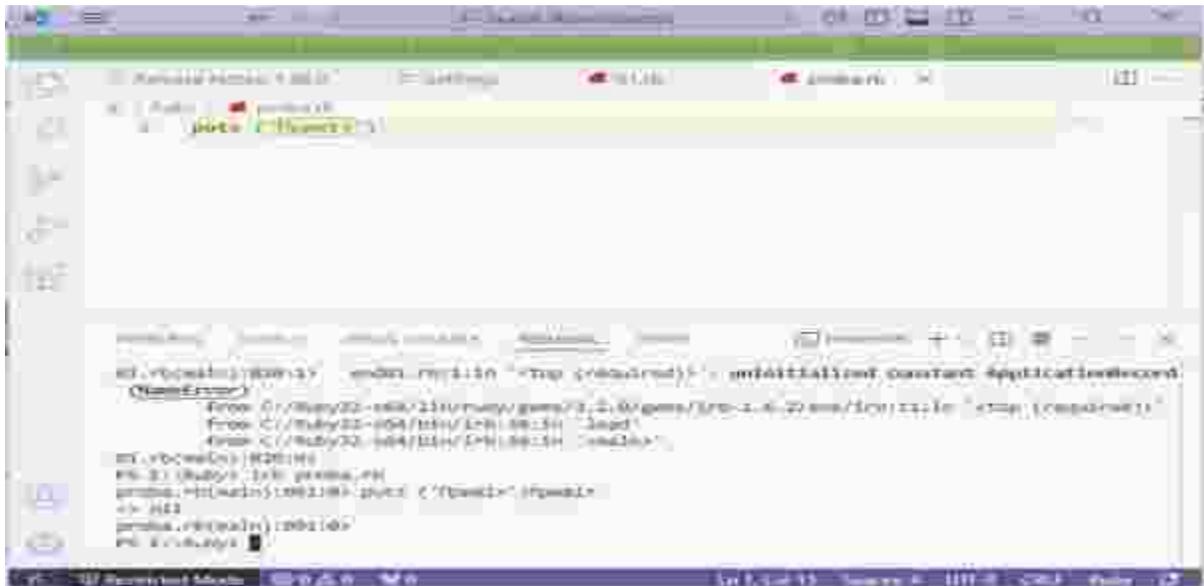
Чим відрізняється від Visual Studio (VS):

- **VS Code** – це редактор коду (code editor), який стає потужним завдяки розширенням, ідеально підходить для веб-розробки.
- **Visual Studio (VS)** – це повноцінне інтегроване середовище розробки (IDE) з усім необхідним "з коробки", часто використовується для розробки на C# та .NET.

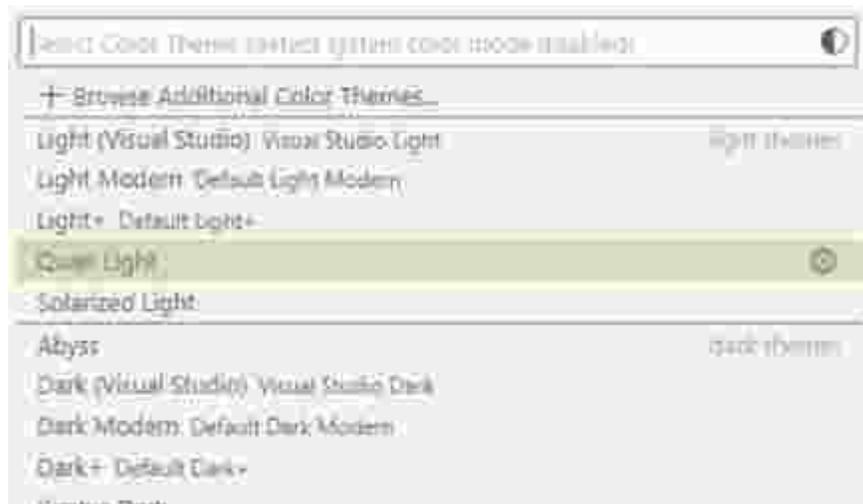
Файл програми можна безкоштовно скачати за адресою: <https://code.visualstudio.com/>

Документація VS Code міститься за адресою: <https://code.visualstudio.com/docs>

Після інсталяції варто спочатку обрати кольорову гаму, натиснувши кнопку * ліворуч внизу рамки програми. У меню, що відкривається, обрати Temes-Color-Temes.



Показане забарвлення відповідає темі



Щоб почати роботу, можна скористатися гарячими клавішами Ctl + N, або по меню File – New.

Щоб бути впевненим, що ваша програма буде сприйматися системою як програма на обраній мові програмування, разом із іменем файлу додайте обов'язкове розширення, наприклад proba.rb, або proba.py.

Для програм на обрані мові програмування краще створити власну папку.

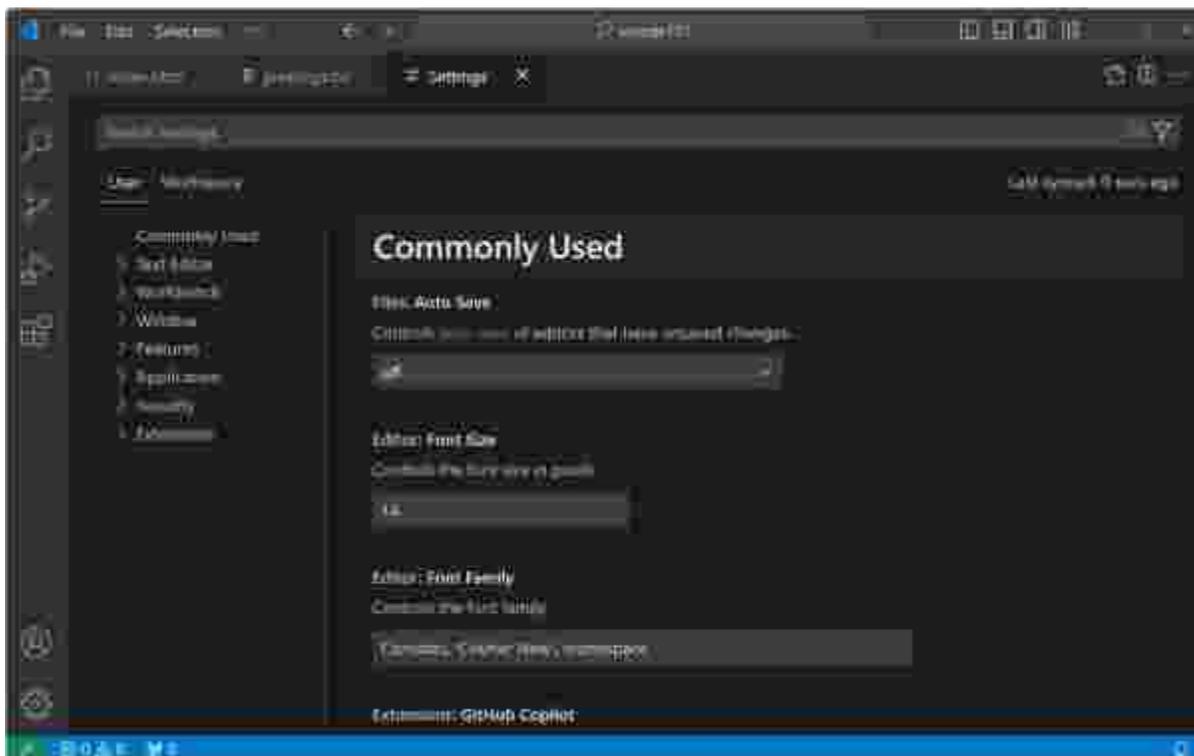
Цей редактор кодів має вбудований доступ до терміналу. Потрібно тільки обрати у меню пункт View-Open View-View -Terminal

Тоді екран розділиться на дві частини: угорі текст програми, а внизу – команди терміналу. Всі дії в терміналі редактора аналогічні діям у терміналі Вільдвз.



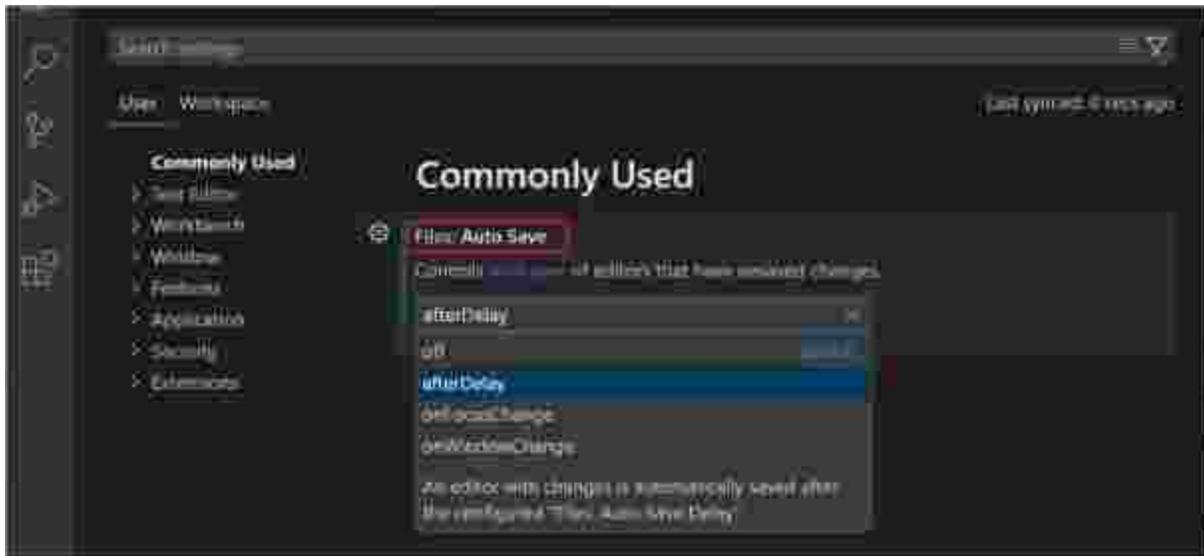
Ви можете налаштувати майже кожен частину VS Code, налаштувавши параметри. Ви можете використовувати редактор налаштувань , щоб змінити параметри у коді VS або безпосередньо змінити settings.jsonфайл.

1. Натисніть Ctrl+, щоб відкрити редактор налаштувань (або виберіть пункт меню «Файл» > «Параметри» > «Параметри»).



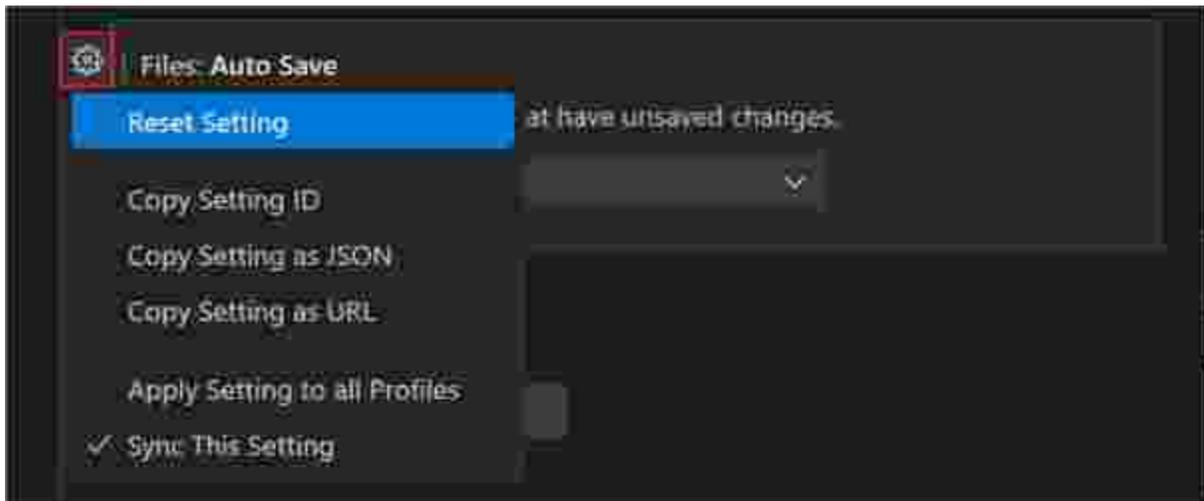
Використовуйте поле пошуку, щоб відфільтрувати список налаштувань, які відображаються.

2. За замовчуванням VS Code не зберігає автоматично змінені файли. Виберіть значення зі спадного списку автозбереження, щоб змінити цю поведінку.



VS Code автоматично застосовує зміни до налаштувань. Коли ви змінюєте файл у своїй робочій області, він має автоматично зберегатися.

3. Щоб повернути параметр до значення за замовчуванням, виберіть значок шестірні поруч із параметром і виберіть Скинути налаштування .

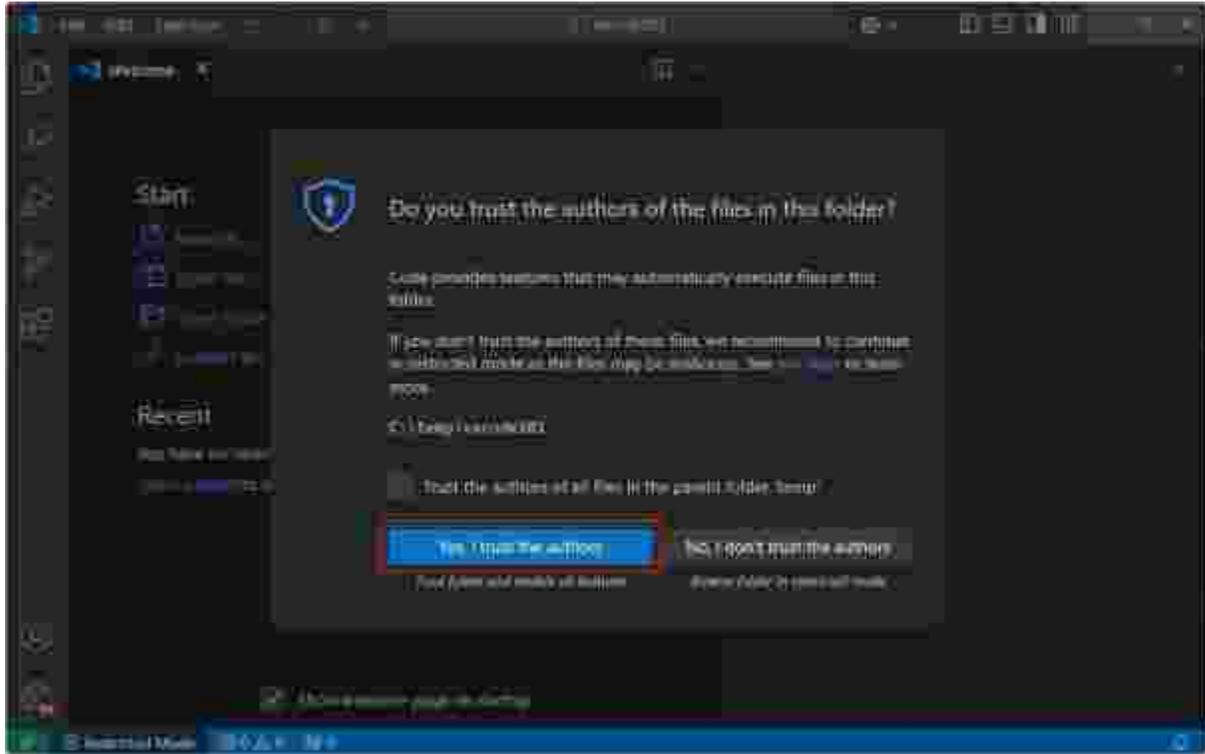


Ви можете швидко знайти всі змінені параметри, ввівши їх @modifiedy поле пошуку або вибравши змінений фільтр.

3. Виберіть «Нова папка», створіть нову папку під обраною вами назвою, наприклад назвою «vscode101Вибрати папку» (« Відкрити в macOS»).

Папка, яку ви створюєте, є коренем вашої робочої області.

4. У діалоговому вікні «Довіра робочої області» виберіть «Так, я довіряю авторам, щоб увімкнути всі функції в робочій області».



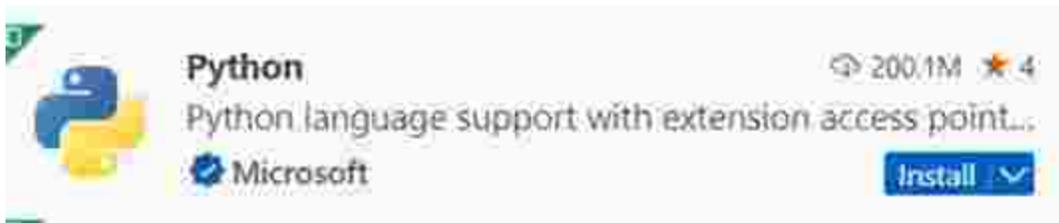
Оскільки ви створили папку на своєму комп'ютері, ви можете довіряти коду в папці.

Коли ви завантажуєте код з Інтернету, ви повинні спочатку переглянути його, щоб переконатися, що його безпечно запускати. Отримайте більше інформації про [Workspace Trust](#).

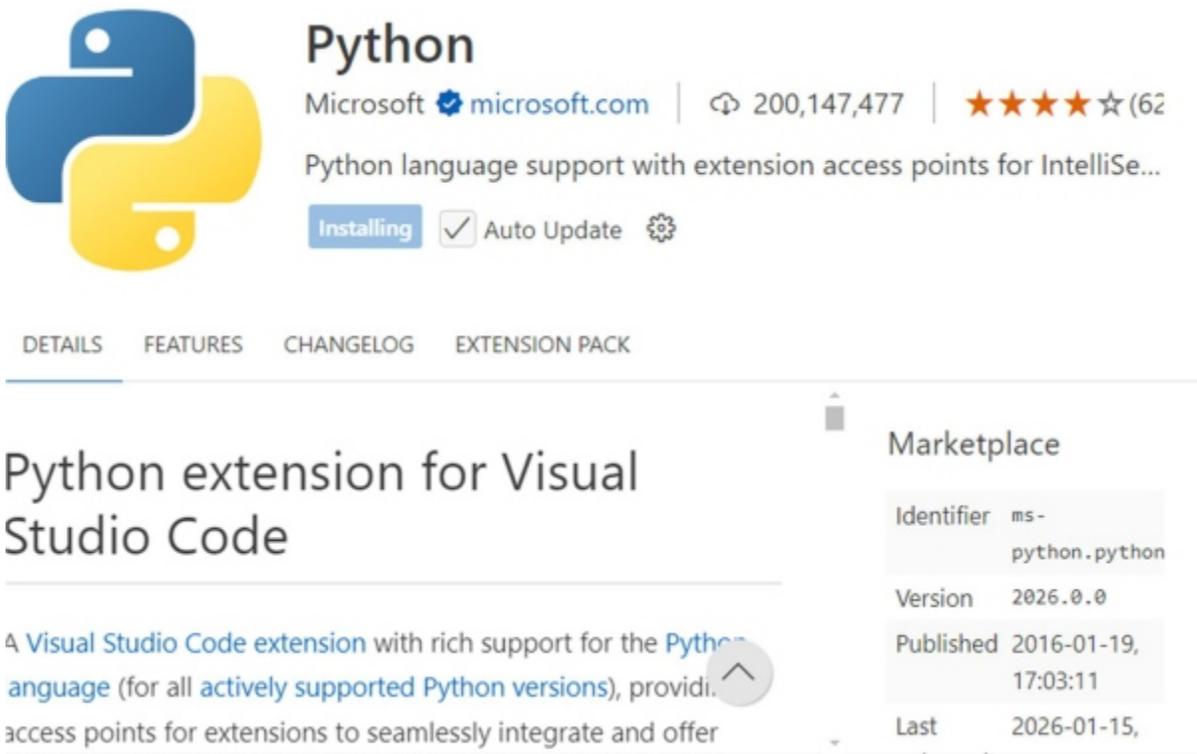
5. Тепер оберіть мову програмування на якій ви будете працювати. Для цього у вікні, що відкриється за закладкою Welcome натисніть кнопку



та зі списку мов обіріть необхідну, наприклад Python.



Тоді до Vs Code підключиться інтерпретатор мови і з'явиться наступне повідомлення



Тепер ви повинні побачити вікно «Провідник» ліворуч із назвою папки.

Для перегляду та керування файлами та папками у вашій робочій області ви використовуватимете подання «Провідник».

Коли ви відкриваєте папку у VS Code, VS Code може відновити стан інтерфейсу користувача для цієї папки, наприклад відкриті файли, активне подання та макет редактора. Ви також можете налаштувати параметри, які застосовуються лише до цієї папки, або визначити конфігурації налагодження.

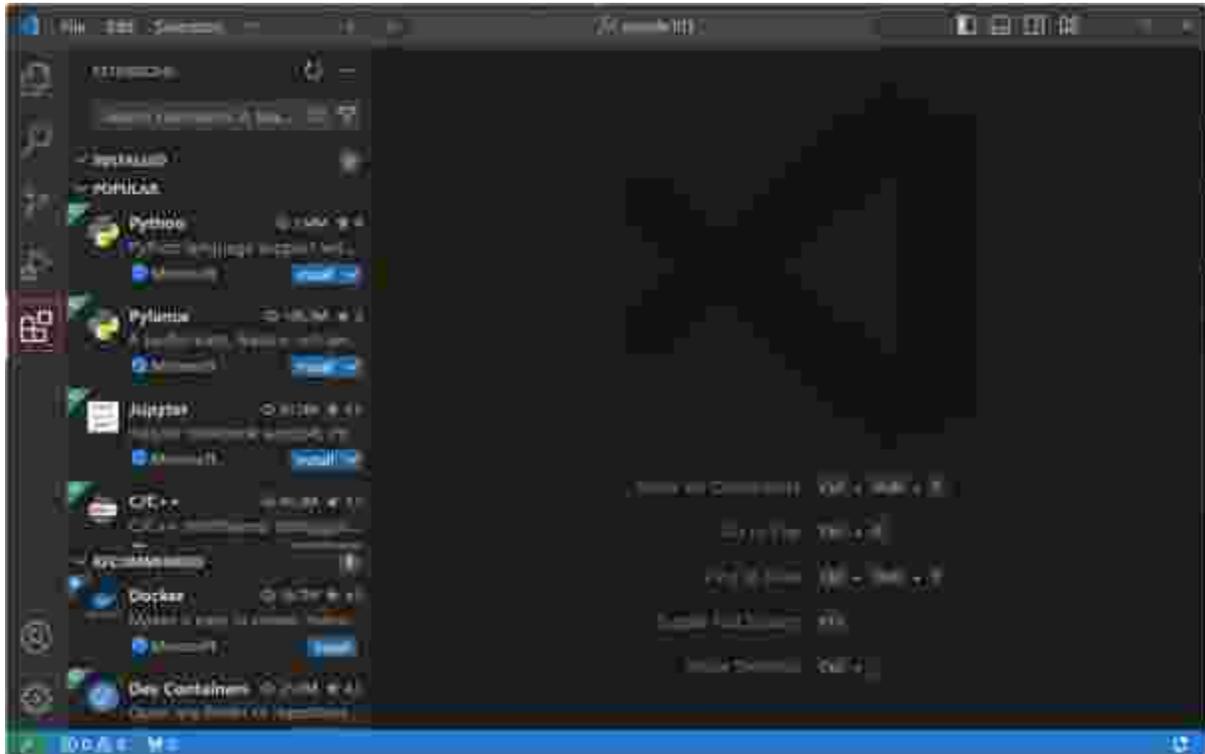
Існує інший метод підключати потрібну мову, у термінології VS Code це називається мовне розширення.

У [Visual Studio Marketplace](#) доступні тисячі розширень .

Давайте встановимо мовне розширення, щоб додати підтримку Python або будь-якої іншої мови програмування, яка вас цікавить.

1. Виберіть подання «Розширення» на панелі активності. Воно помічено червоною рамкою.

Перегляд розширень дає змогу переглядати та встановлювати розширення з VS Code.



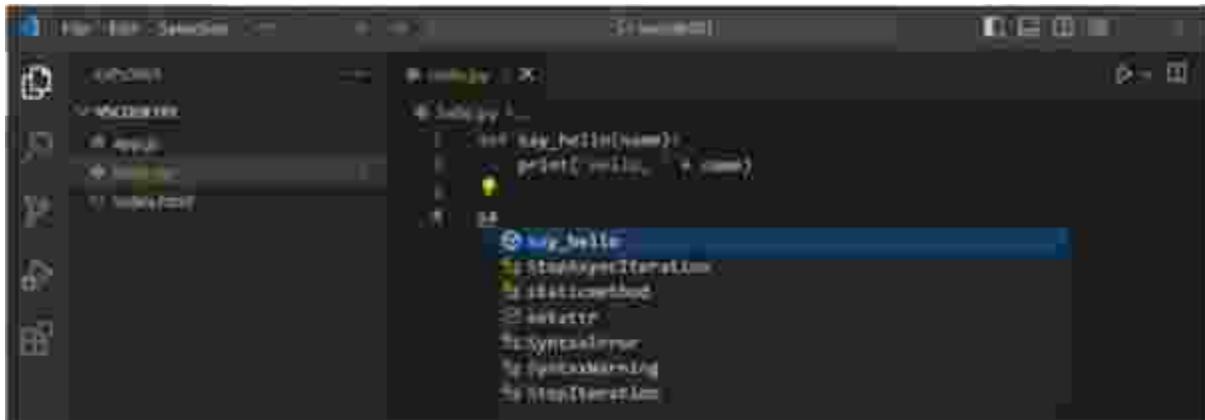
2. Введіть *Python* у вікні пошуку розширення, щоб переглянути пов'язані з Python розширення. Виберіть розширення Python , опубліковане Microsoft, а потім натисніть кнопку «Установити» .



3. Тепер створіть новий файл Python hello.py у своїй робочій області та почніть вводити наступний код Python:

```
def say_hello(name):  
    print("Hello, " + name)  
  
say_hello("VS Code")
```

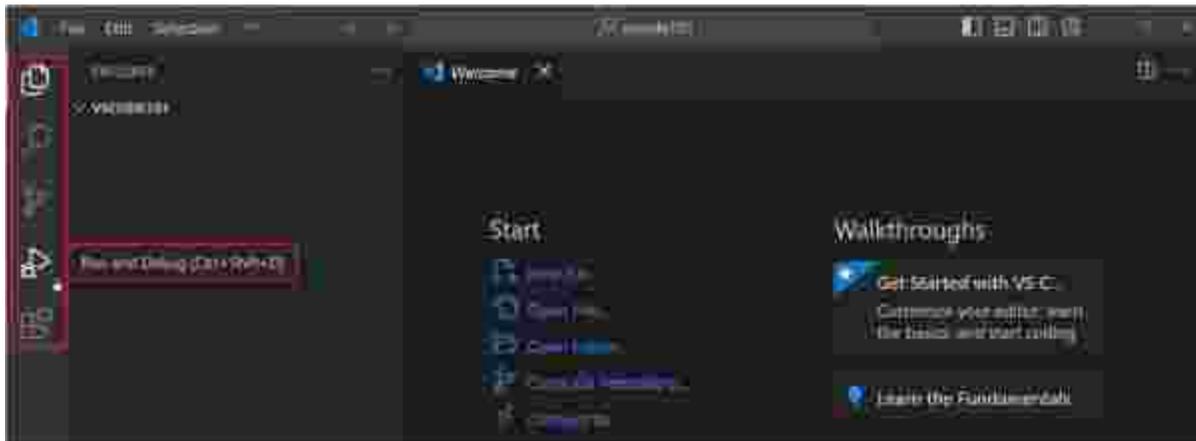
Зверніть увагу, що тепер ви також отримуєте пропозиції та IntelliSense для коду Python, що дають вам можливі варіанти коду.



1.2.2. Інтерфейс користувача

Тепер, коли у вас відкрита папка у VS Code, давайте зробимо короткий огляд інтерфейсу користувача.

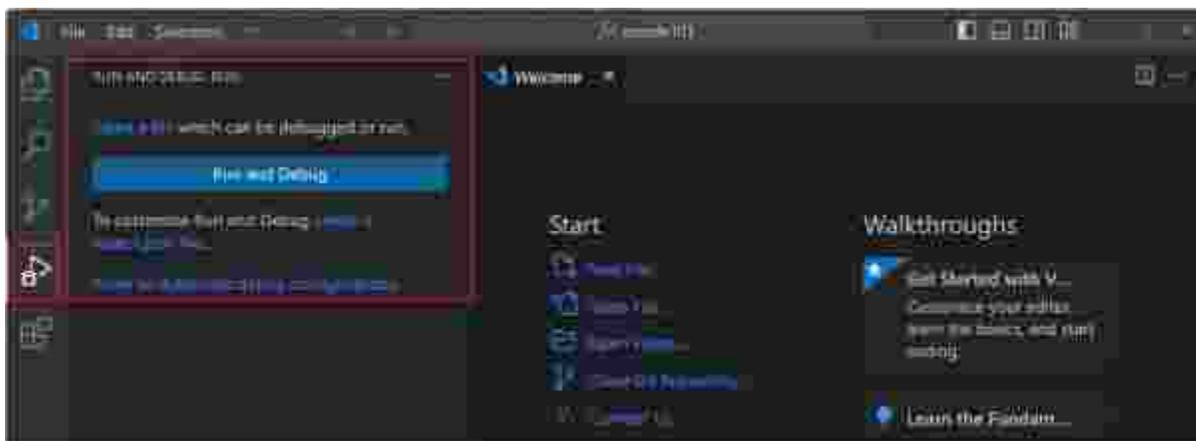
- Використовуйте панель активності для перемикання між різними видами.



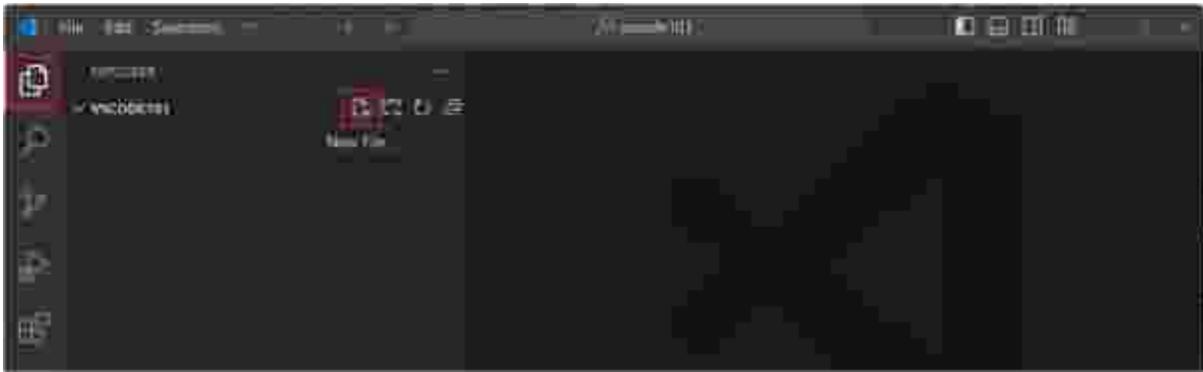
Наведіть курсор на панель активності, щоб побачити назву кожної іконки на панелі користувача та відповідну комбінацію клавіш. Ви можете перемикати подання, відкриваючи та закриваючи, знову вибравши подання або натиснувши комбінацію клавіш.

- Коли ви вибираєте подання на панелі активності, відкривається основна бічна панель, на якій відображаються відомості про подання.

Наприклад, перегляд «Запуск і налагодження» дає змогу налаштувати та розпочати сеанси налагодження.

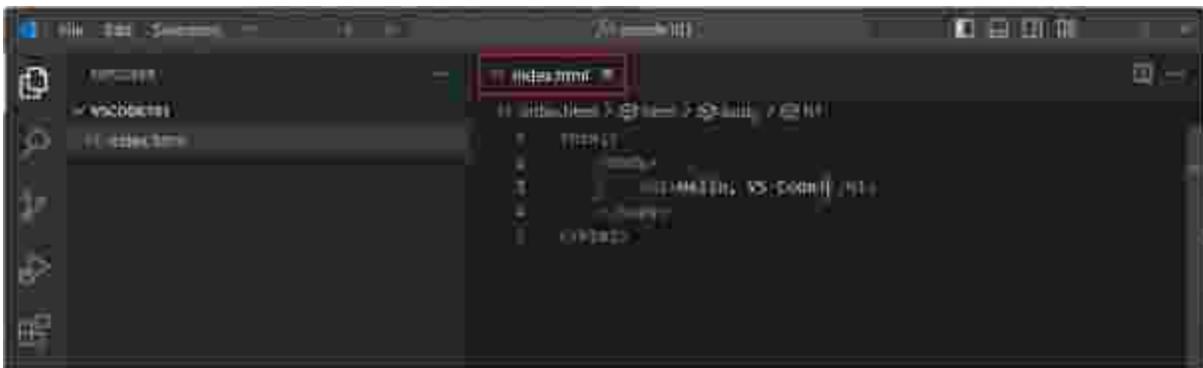


- Виберіть подання «Провідник» на панелі активності та натисніть кнопку «Новий файл...», щоб створити новий файл у своїй робочій області.



- Введіть назву, наприклад, index.html та натисніть Enter .

Файл буде додано до вашої робочої області, і в головній частині вікна відкриється редактор.



- Почніть вводити HTML-код у index.html файл.

Під час введення повинні з'являтися пропозиції, які допоможуть завершити код (*IntelliSense*). Ви можете використовувати клавіші «Вгору» та «Вниз», щоб переміщатися між пропозиціями, і Tab, щоб вставити вибрану пропозицію.

- Додайте більше файлів у свою робочу область і зверніть увагу, що кожен файл відкриває нову вкладку редактора.

Ви можете відкривати скільки завгодно редакторів і переглядати їх поруч по вертикалі чи горизонталі. Дізнайтеся більше про [паралельне редагування](#) .



1.2.3. Доступ до терміналу з області панелі

1. VS Code має вбудований термінал. Відкрийте його, натиснувши `Ctrl+``.

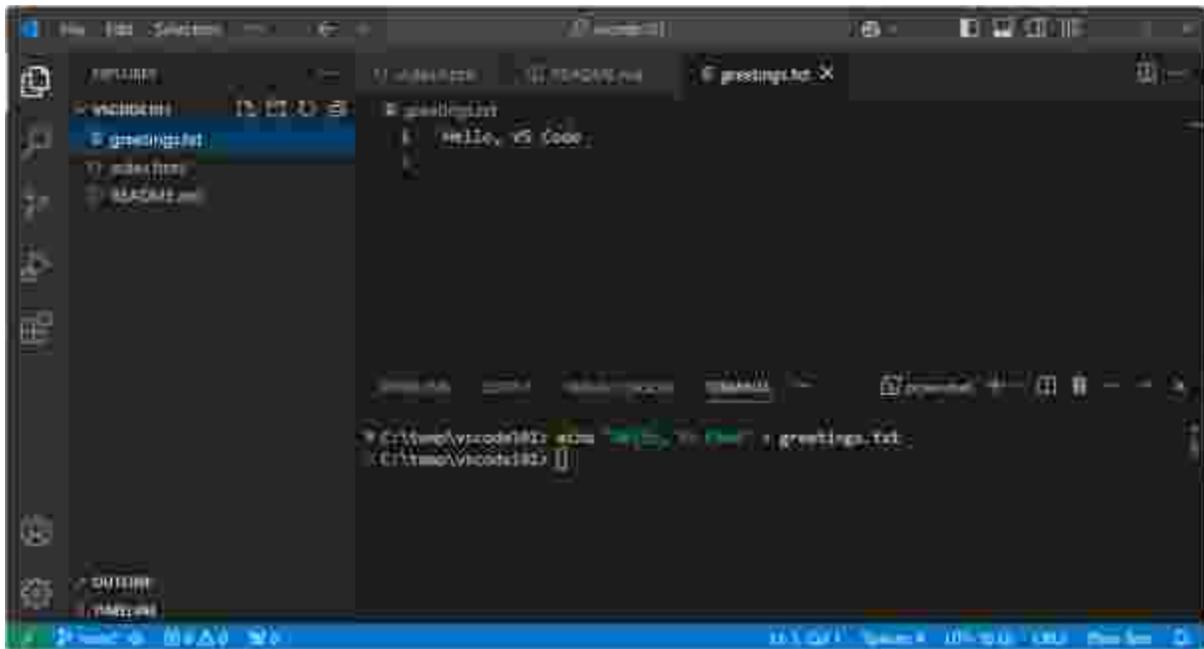
Ви можете вибрати між різними оболонками, такими як PowerShell, командний рядок або Bash, залежно від конфігурації вашої операційної системи.



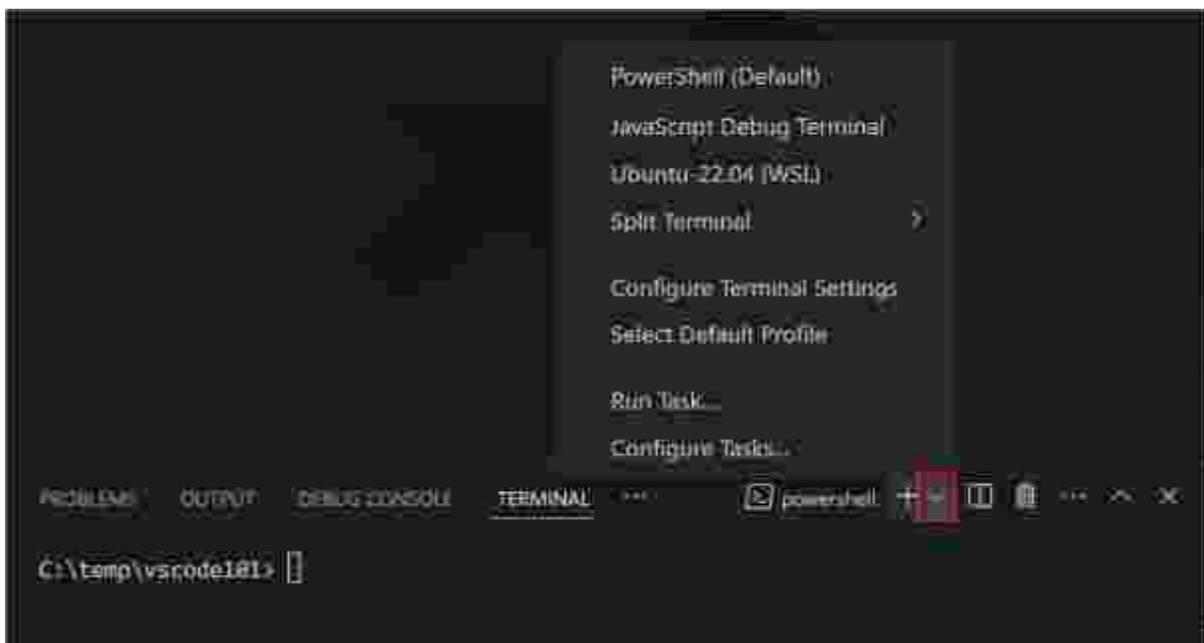
2. У терміналі введіть таку команду, щоб створити новий файл у вашій робочій області.

3. `echo "Hello, VS Code" > greetings.txt`

Робоча папка за умовчанням є коренем вашої робочої області. Зауважте, що перегляд Провідника автоматично підбирає та показує новий файл.



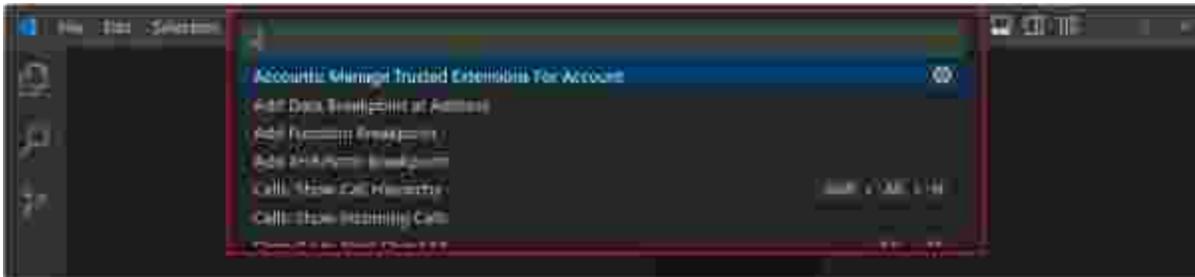
4. Ви можете відкрити кілька терміналів одночасно. Виберіть спадне меню Запустити профіль , щоб переглянути доступні оболонки та вибрати одну.



1.2.4. Доступ до команд за допомогою панелі команд

1. Відкрийте панель команд , натиснувши Ctrl+Shift+P . Ви також можете скористатися пунктом меню «Перегляд» > «Палітра команд...» .

Багато команд у VS Code доступні через палітру команд. Коли ви встановлюєте розширення, вони також можуть додавати команди до панелі команд.



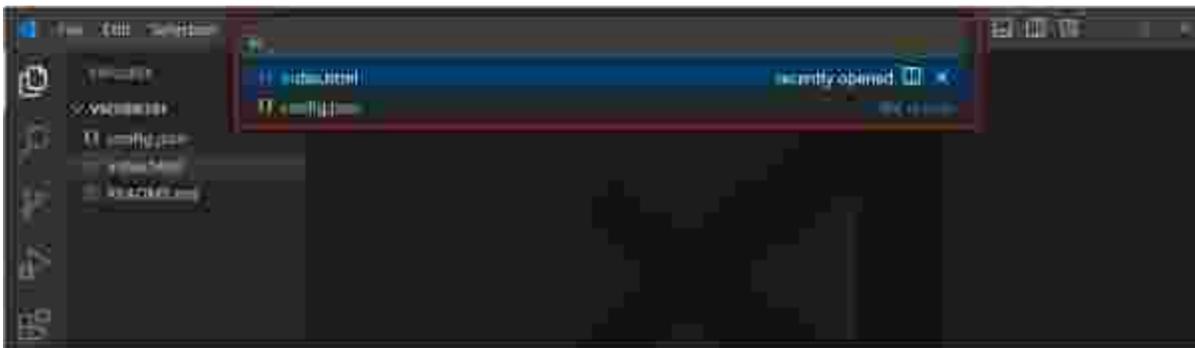
Зверніть увагу, що на панелі команд показано стандартне сполучення клавіш для команд, які його мають. Ви можете використовувати комбінацію клавіш, щоб запустити команду безпосередньо.

2. Палітра команд підтримує різні режими роботи:

- Після >символу почніть вводити, щоб відфільтрувати список команд. Наприклад, введіть, `move terminal` щоб знайти команди, щоб перемістити термінал у нове вікно.



- Видаліть >символ і почніть вводити текст, щоб шукати файли у вашій робочій області. Ви можете скористатися комбінацією клавіш `Ctrl+P`, щоб відкрити палітру команд і почати безпосередній пошук файлів.



VS Code використовує нечітку відповідність для пошуку файлів або команд. Наприклад, введення `odkspovertae Open Default Keyboard Shortcuts` команду.

1.2.5. Написання коду

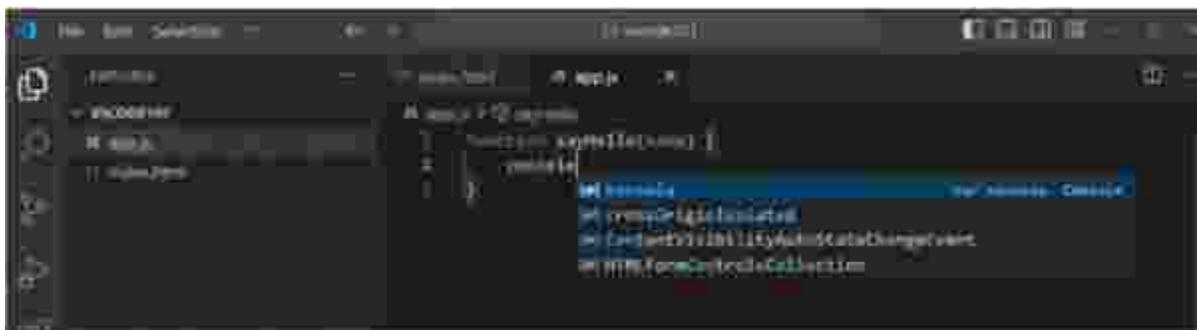
VS Code має вбудовану підтримку JavaScript, TypeScript, HTML, CSS тощо. У цьому прикладі створюється зразок файлу JavaScript і використовуються деякі функції редагування коду, які пропонує VS Code.

1. У вікні Провідника створіть новий файл `app.js` почніть вводити такий код JavaScript:

2. `function sayHello(name) {`
3. `console.log('Hello, ' + name);`
4. `}`
- 5.
6. `sayHello('VS Code');`

Під час введення повинні з'являтися пропозиції, які допоможуть завершити код (*IntelliSense*). Ви можете використовувати клавіші «Вгору» та «Вниз», щоб переміщатися між пропозиціями, і `Tab`, щоб вставити вибрану пропозицію.

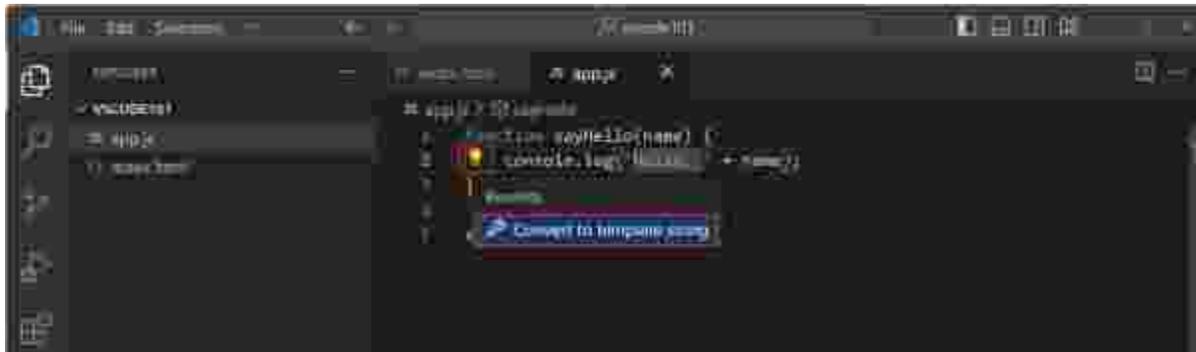
Зверніть також увагу на форматування коду (*підсвічування синтаксису*), щоб допомогти вам розрізнити різні частини коду.



7. Коли ви наведете курсор на рядок `Hello,`, ви побачите піктограму лампочки, яка вказує на дію коду.

Ви також можете скористатися комбінацією клавіш Ctrl+Пробіл, щоб відкрити меню лампочки.

8. Виберіть піктограму лампочки, а потім виберіть «Перетворити на рядок шаблону».



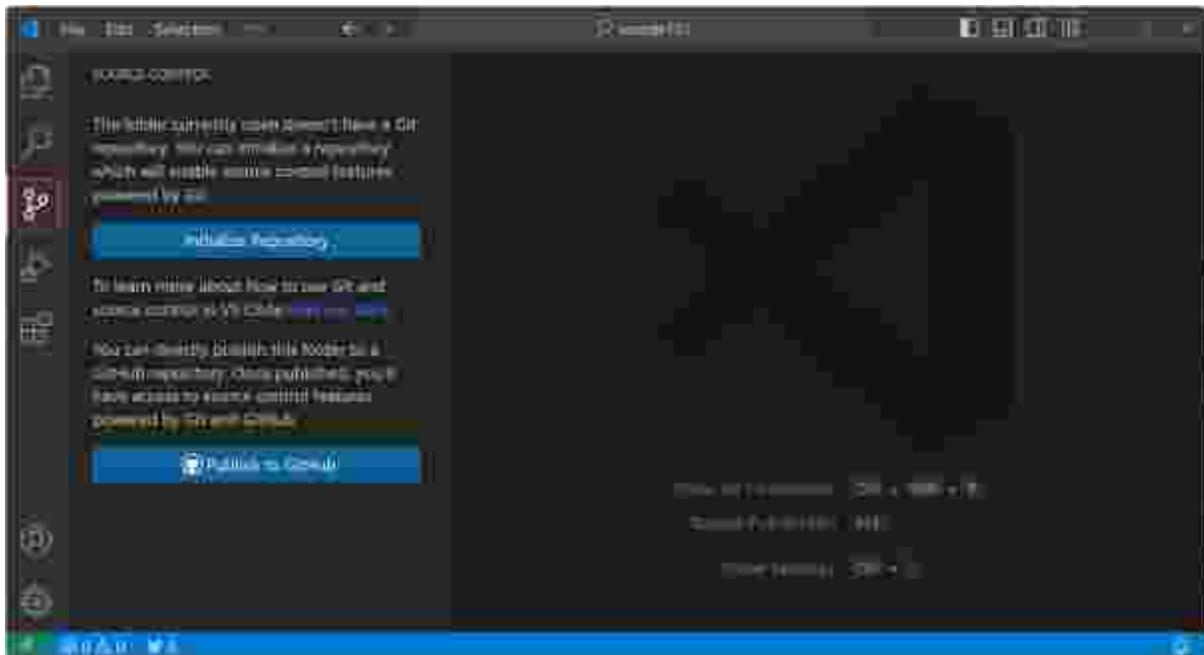
Дії коду – це пропозиції щодо застосування швидких виправлень до вашого коду. У цьому випадку дія коду перетворюється `""Hello, " + name` на рядок шаблону ``Hello, ${name}``, який є спеціальною конструкцією JavaScript для вбудовування виразів у рядки.

1.2.6. Контроль джерел

Visual Studio Code має інтегроване керування джерелами (SCM) і включає підтримку [Git](#) із коробки.

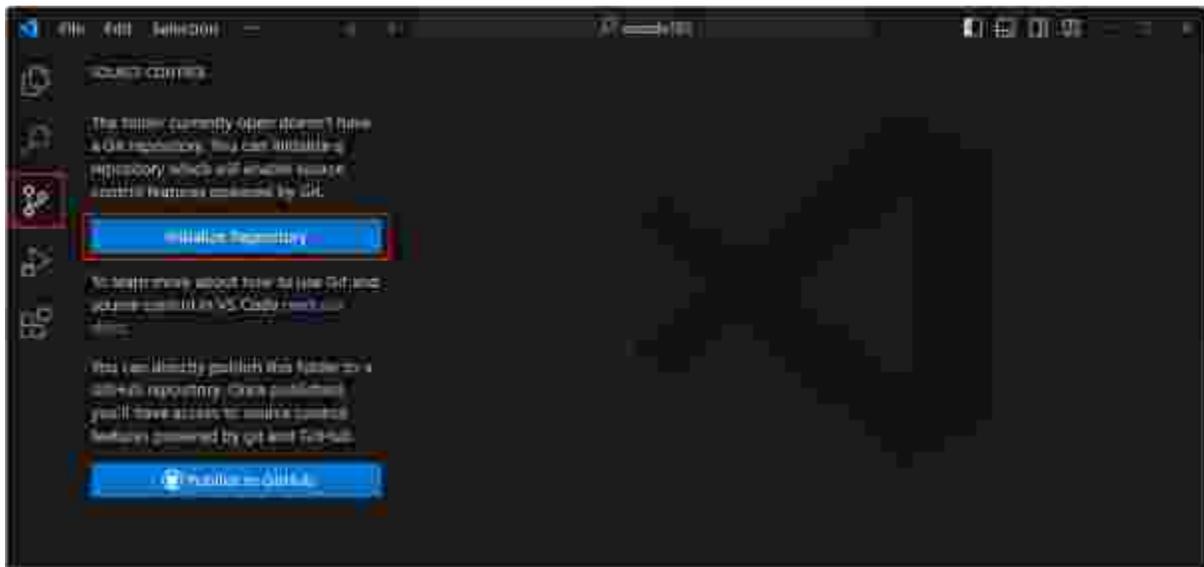
Давайте скористаємося вбудованою підтримкою Git, щоб прийняти зміни, які ви внесли раніше.

1. Виберіть подання Source Control на панелі активності, щоб відкрити подання Source Control.



2. Переконайтеся, що на вашому комп'ютері встановлено [Git](#) . Якщо у вас не встановлено Git, ви побачите кнопку в поданні Source Control, щоб установити його на вашій машині.

3. Виберіть «Ініціалізувати репозиторій» , щоб створити нове сховище Git для вашої робочої області.



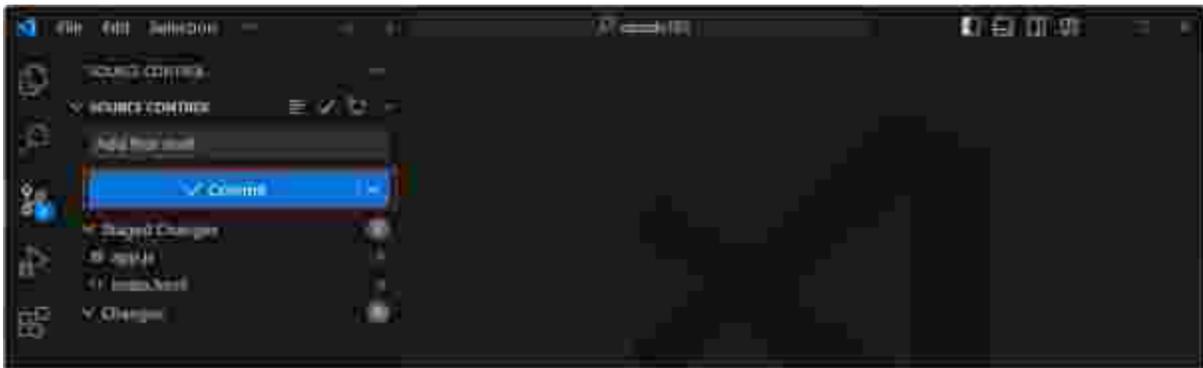
Після ініціалізації репозиторію в поданні Source Control відображаються зміни, які ви внесли у свою робочу область.

4. Ви можете внести окремі зміни, навівши курсор на файл і вибравши +поруч із файлом.



Щоб відредагувати всі зміни, наведіть вказівник миші на «Зміни» та виберіть кнопку « Відтворити всі зміни» .

5. Введіть повідомлення про фіксацію, наприклад Add hello function, а потім виберіть « Закріпити» , щоб зафіксувати зміни у вашому сховищі Git.



Ви можете використовувати Source Control Graph , щоб показати візуальне представлення історії комітів вашого сховища Git.

У VS Code можна дізнатися багато іншого про керування джерелами.

1.2.7. Запуск і відладка коду програми

VS Code має вбудовану підтримку для запуску та налагодження програм Node.js. У цьому посібнику ви використовуєте розширення Python, яке ви встановили на попередньому кроці, щоб налагодити програму Python.

Давайте налагодимо hello.рупрограму, яку ви створили на попередньому кроці.

1. Переконайтеся, що на вашому комп'ютері встановлено Python.

Якщо на вашому комп'ютері не встановлено інтерпретатор Python, ви побачите сповіщення в нижньому правому куті вікна. Виберіть «Вибрати інтерпретатор», щоб відкрити палітру команд і вибрати інтерпретатор Python, який ви бажаєте використати, або встановити його.

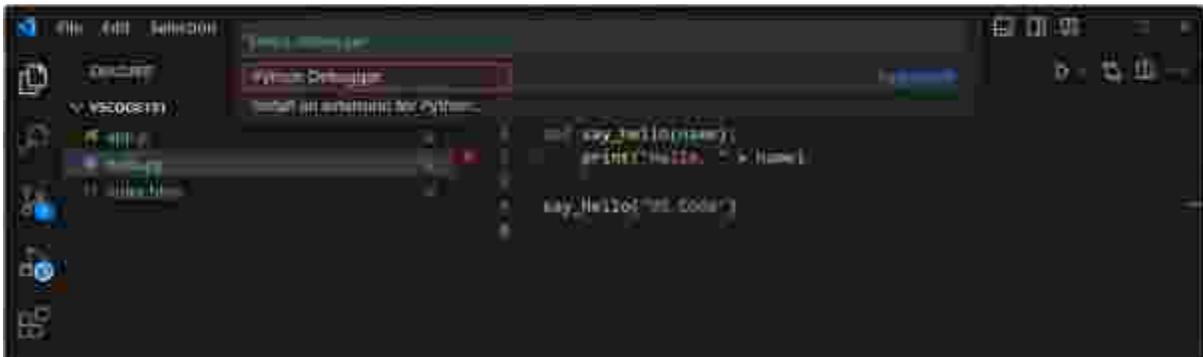
2. У hello.py файлі помістіть курсор на print рядок і натисніть F9, щоб встановити точку зупину.

На лівому полі редактора з'являється червона крапка, яка вказує на те, що встановлено точку зупину. За допомогою точки зупину ви можете призупинити виконання вашої програми на певному рядку коду.



3. Натисніть F5, щоб почати сеанс налагодження.

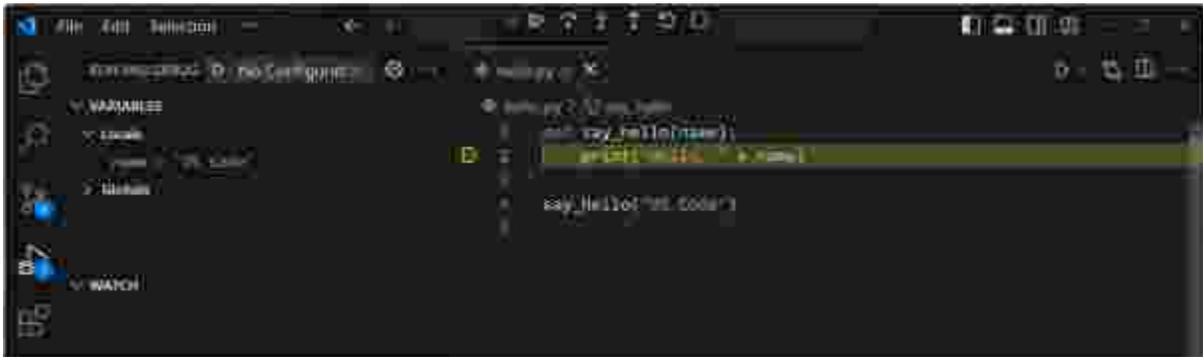
- Виберіть налагоджувач Python:



- Виберіть запуск поточного файлу Python:

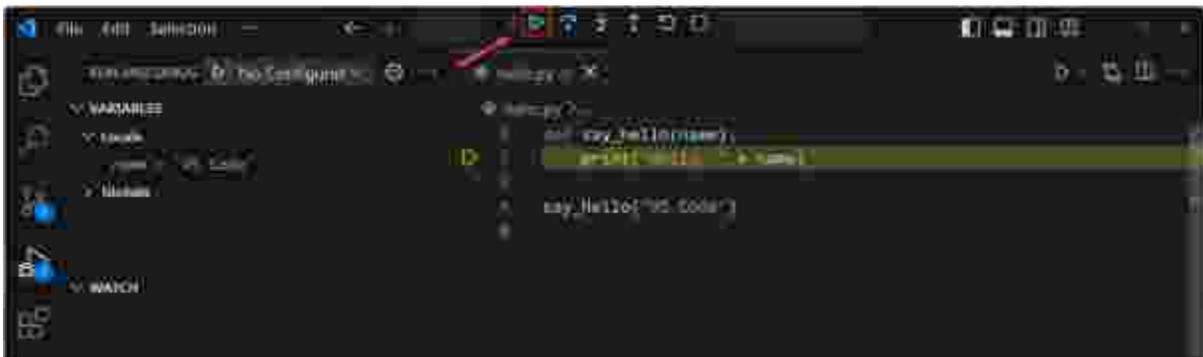


- Зверніть увагу, що програма запускається і виконання зупиняється на встановленій точці зупинки.



Перевірте значення змінної name, навівши на неї курсор у редакторі, коли виконання призупинено. Ви можете будь-коли переглянути значення змінних у поданні «Змінні» в поданні «Виконати та налагодити» .

- Натисніть кнопку «Продовжити» на панелі інструментів Debug або натисніть F5 , щоб продовжити виконання.



1.2.8. Онлайн інтерпретатори команд мови програмування

Окрім VS Code, що має бути встановленим на вашому комп'ютері, існують он-лайн інтерпретатори. Ними варто користатися, коли під рукою немає VS Code.

Ці сервіси легкі у використанні, але мають обмеження по можливостям роботи з GitHub, на деяких із них не працює режим залучення можливостей бібліотек, для обраної мови програмування, тощо.

Спочатку розглянемо Онлайн-інтерпретатор **OneCompiler**
<https://onecompiler.com/>



Натиснемо на іконку з Пайтоном і отримаємо готовий до роботи інтерпретатор



Тут у лівому вікні текст вашої програми, а у правому – результат її роботи.

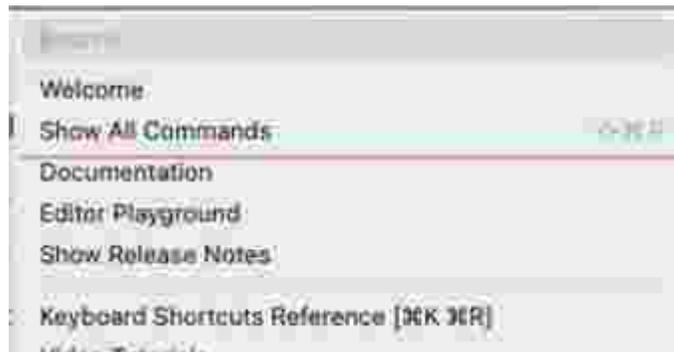
Онлайн-інтерпретатор **Code Runner** можна скачати за адресою

<https://marketplace.visualstudio.com/items?itemName=formulahendry.code-runner>

І після його установки у головному меню натиснути три крапки. А у списку, що з'явиться, обрати Help.



Тоді з'явиться новий список, у якому треба обрати Show ALL Comasnds.



З'явиться ще один список, у якому треба обрати перший пункт Preferences: Open User Settings (JSON).



Відкриється вікно файлу з якимось текстом. Цей текст треба замінити на наступний зі вказанням мови програмування, на якій ви плануєте розробляти програму. У прикладі показана мова Ruby.

```
"code-runner.executorMap":{  
  "ruby": "ruby"
```

```
},  
"code-runner.defaultLanguage": "ruby",  
"code-runner.runInTerminal": true
```

Тоді колір стрілки, що викликає Runner зміниться з зеленого на сірий, але працювати буде бездоганно.



```
{Running} ruby "c:\Ruby\prueba1.rb"  
-24.0  
{Done} exited with code=0 in 0.191 seconds
```

Реєстрацію можна виконати через Facebook або через Google, далі натиснути New Project/ потім обрати мову, на якій будете програмувати, клацнувши по пункту Language. Щоб створити новий файл програми, треба натиснути на 

Угорі головного меню дати файлу ім'я та розширення, наприклад *.ру, і можна писати код.

1.3. Приклад використання VS Code та GitHub

Нехай маємо новий проєкт з назвою A1, в якому необхідно створити файл за допомогою VS Code з іменем hello.py двох версій, 1.0 та 1.1.

Перша версія файлу складається з однієї команди **print("Hello world")**, а друга **print("Hello Dolly")**. Тобто різниця тільки у словах **world** та **Dolly**.

Після їх створення як на комп'ютері так і на GitHub, обидві версії треба об'єднати в одну через дії на GitHub. Але ця зміна має відбутися і у виділеній для проєкту теці на вашому комп'ютері.

Почнемо з реєстрації. Перші дві команди, що визначають ім'я користувача та його електронну пошту мають вигляд



```
python@M-PC M1hGed4 ~$  
$ git config --global user.name "Igor Pistunov"
```

```
Microsoft Windows [Version 10.0.17134.0]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\pistunovi> git config --global user.email "pistunovi@gmail.com"
```

Git дуже жорстко ставиться до синтаксису команд, тому зверніть увагу, що перед двома мінусами має бути пробіл. Всі імена мають бути тільки латиницею.

Тепер створимо репозиторій на GitHub, бо саме так ми зможемо забезпечити поєднання файлі на GitHub з файлами на нашому комп'ютері. Для цього викликаємо за відомою адресою GitHub, реєструємося, якщо раніше не заходили туди, і отримаємо запрошення на створення нового репозиторію. Кожен проєкт повинен мати свій репозиторій.

Create a new repository
Repositories contain a project's files and version history. Have a project elsewhere? [Import a repository.](#)
Required fields are marked with an asterisk ().*

1 General

Owner * / Repository name * ✓ A1 is available.

Great repository names are short and memorable. How about [crispy-octo-train?](#)

Description 70 / 350 characters

2 Configuration

Choose visibility * 5

Add README 6

Add .gitignore 7

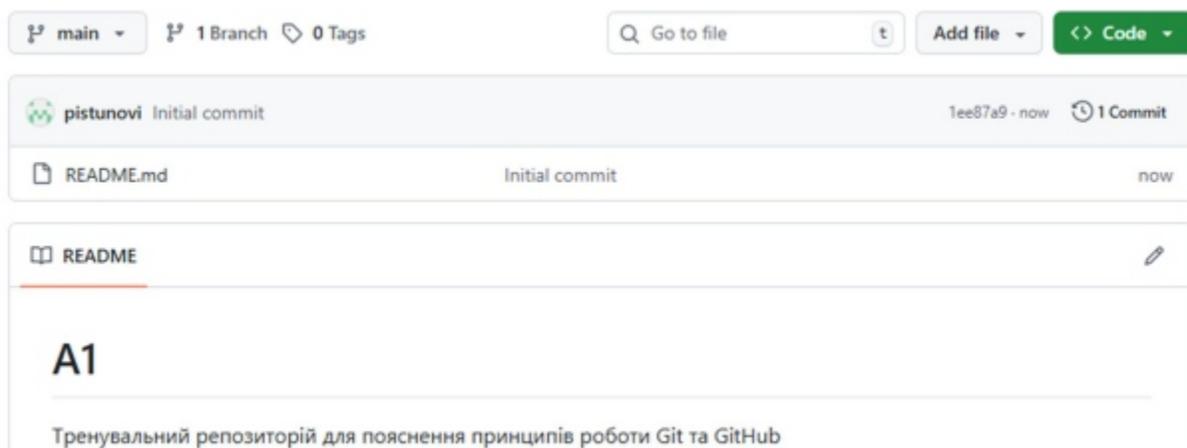
Add license

Create repository

Пояснення до вікна створення: 1 – Назва репозиторію; 2 – повідомлення про можливість створити такий репозиторій. Тобто, в системі GitHub репозиторію з таким іменем немає; 3 – Короткий опис проєкту; 4 – Чи можуть інші користувачі бачити ваші файли. Public означає, що можуть; 5 – Додавання спеціально файлу Readme.md, що містить будь яку інформацію про ваш проєкт; 6 – Чи існують у вашому проєкті файли, які GitHub має ігнорувати. За замовчуванням – ні; 7 – Чи потрібна ліцензія на використання ваших файлів сторонніми особами. За замовчуванням – ні;



Репозиторій створено. Тут подано ім'я репозиторію – A1, власника – pistunovi, та показано наявність файлу Readme.md



Символ авторучки біля імені файлу readme показує можливість його редагувати. Натиснувши кнопку маємо вікно текстового редактору, що вже містить ім'я репозиторію – A1, та короткий опис, що ми його надали спочатку.

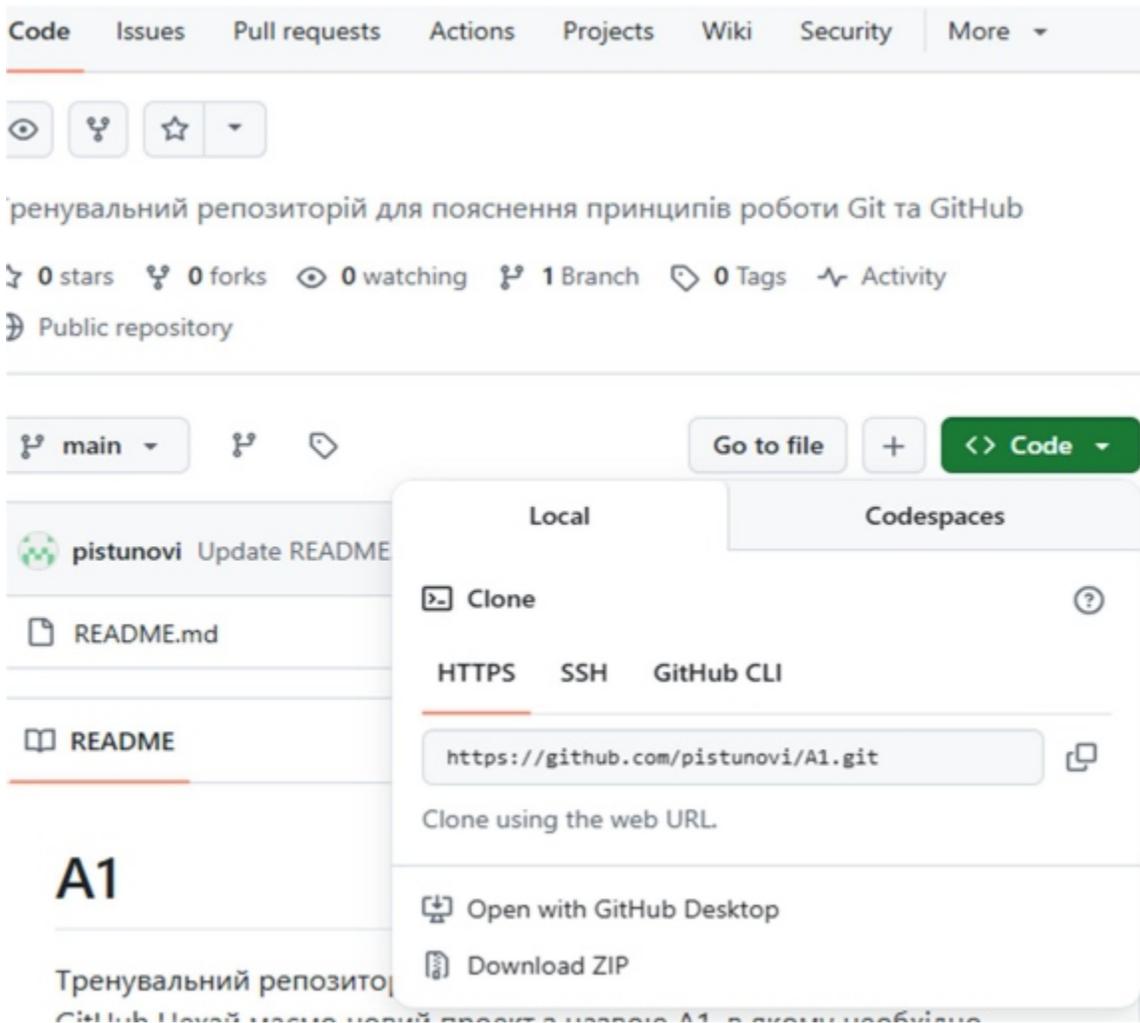
Додамо туди наше завдання і отримаємо

```
← Files A1 / README.md in main Cancel changes Commit changes...  
Edit Preview Spaces 2 Soft wrap  
1 # A1  
2 Тренувальний репозиторій для пояснення принципів роботи Git та GitHub  
3 Нехай маємо новий проект з назвою A1, в якому необхідно створити файл за допомогою  
4 VS Code з іменем hello.py двох версій, 1.0 та 1.1.  
5 Перша версія файлу складається з однієї команди print("Hello world"),  
6 а друга print("Hello Dolly"). Тобто різниця тільки у словах world та Dolly.  
7 Після їх створення як на комп'ютері так і на GitHub, обидві версії треба  
8 об'єднати в одну через дії на GitHub. Але ця зміна має відбутися і  
9 у виділеній для проекту теці на вашому комп'ютері.  
10
```

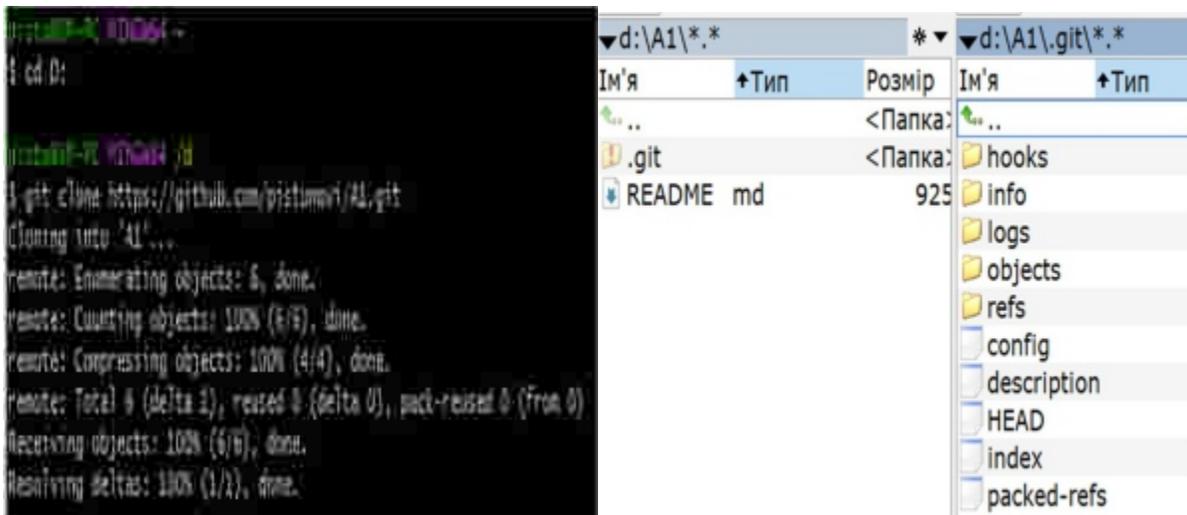
Кнопка `Commit changes` дозволяє запам'ятати все додане до тексту. Після цього натискаємо стрілочку вліво біля слова `Files` і повертаємося до попереднього вікна, у якому вже видно доповнення тексту файлу `readmy`.

Клонуємо репозиторій з GitHub на свій комп'ютер. Спочатку у вікні Git виконаємо перехід до потрібного диску і папки командою, наприклад `cd D:` щоб розмістити клон репозиторію в кореневому каталозі диску D. Якщо потрібно додати якусь папку, її ім'я треба вказати через зворотній слеш, наприклад `cd D:\przyklad\`, де `przyklad` – це ім'я папки.

Для клонування потрібно для Git надати команду типу `git clone <адреса на проєкту Github >`. Цю адресу можна отримати обравши пункт головного меню `Code`. Тоді на сайті з'явиться кнопка , натискання якої відкриває вікно, де і буде url-адреса вашого проєкту.



Ось приклад роботи Git по клонуванню. І ось такий зміст папки на цей момент



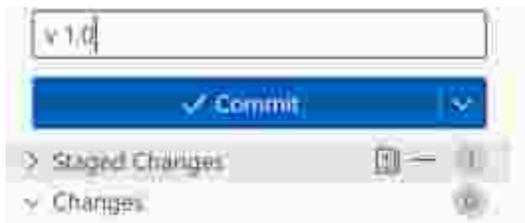
Відкриємо тепер через VS Code папку з репозиторієм. Для цього достатньо натиснути Ctrl + O і у файл менеджері, що відкриється, обрати папку і файл, який вам потрібен.

Відкриємо нове вікно, натиснувши Alt+Ctrl+Windows+N, і вставимо туди команду Пайтона. Збережемо цей файл в A1, натиснувши File-Save.

```
D: > A1 > Hello.py > ...
1 print ("Hello World")
2 a = input()
```

Тепер переходимо до нашої папки через меню File-New Folder. У списку, що відкрився, ми бачимо ті файли, що створені для нашого проекту. Натискаємо

кнопку  і потім знак «плюс» біля імені файлу. У вікні зверху треба прописати



версію файлу. Щоб цей текст запам'ятовувався, натисніть кнопку Commit. Команда **git log** у вікні **Git** вам покаже результат запам'ятовування.

```
pistunov-PC MINGW64 /d/A1 (main)
$ git log
commit 0cf7b10e7b7c644bc47e3f01ffa499b52f05ef7d (HEAD -> main)
Author: Igor Pistunov <pistunovi@gmail.com>
Date: Wed Jan 28 15:41:09 2026 +0200

v 1.0
```

Змінимо тепер текст у файлі згідно завдання і виконаємо ті самі дії, вказавши тепер версію 1.1. Тоді VS Code покаже вам два вікна з різними



файлами. Нова версія позначена приміткою (Working Tree). Щоб побачити, чи всі зміни зафіксовані, зручніше всього користуватися командою `git log --oneline`

Передамо тепер ці зміни на GitHub, скориставшись командою `git push`. Або у VS Code в режимі Source Control натиснемо PUSH.

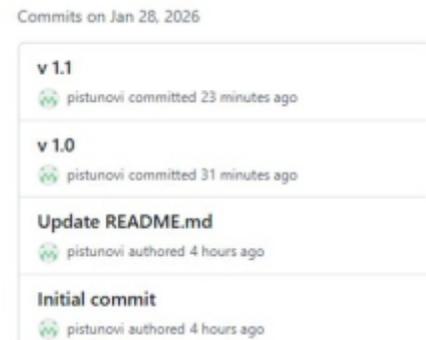
Тепер на GitHub видно файл останньої версії



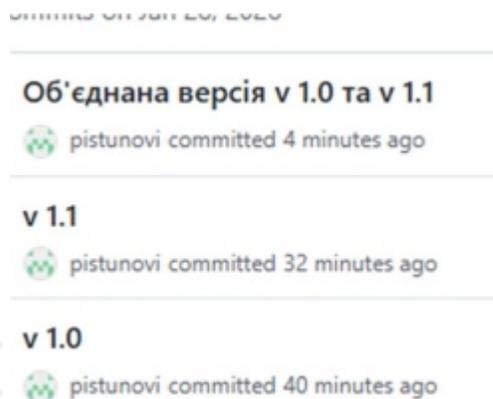
Та якщо натиснути піктограму , то можна побачити список усіх версій файлу.

Але на вашому комп'ютері буде відображено тільки останню версію файлу.

Об'єднання версій файлу може бути тільки «вручну». Після об'єднання кодів треба повторити всі дії і створити третю версію файлу. Результати такого об'єднання знову через PUSH можна передати до GitHub і у вікні Commits побачити всі версії.



```
1 print ("Hello Dolly")
2 print ("Hello Dolly")
3 a = input()
```



Кожну з цих версій можна переглянути, клацнувши по ній мишкою.

Коротко (для запам'ятовування)

- VS Code = редактор
- Git = інструмент
- GitHub = хмарне сховище
- Підключення = **вхід + репозиторій + push**

Тобто, всі роботи ми ведемо на комп'ютері, а потім перекидаємо наші дані на GitHub.

1.4. GitHub Actions

GitHub Actions – це вбудована у GitHub платформа автоматизації, яка дозволяє запускати скрипти та робочі процеси (workflows) у відповідь на події в репозиторії або за розкладом. GitHub Actions – це інструмент для автоматичного виконання коду, тестів або інших задач без участі людини.

Він дозволяє: Автоматично запускати код після push; Виконувати тести; Збирати проекти; Робити деплой; Запускати програми за розкладом (cron).

Для навчального прикладу використаємо репозиторій DS в якому є папка 2 а в ній програма rursingакс.ру. Ця програма і тільки вона має працювати в режимі деплоя кожного разу, як настане нова доба, тобто у 00:00. Результати цієї роботи мають потрапляти у мій же репозиторій, у файл у форматі Excel, хоча в самій програмі шлях до файлу прописаний на моєму комп'ютері як d:\A1\.

Спочатку потрібно у програмі змінити локальний шлях

`file_path = "D:\\A1\\result.xlsx"` на `file_path = "result.xlsx"`.

GitHub Actions працює у Linux-середовищі, тому файл створюється в корені робочої директорії, об то, у корені DS.

Далі необхідно Створити GitHub Personal Access Token. Для цього в налаштуваннях профіля (треба просто клацнути по аватару). Далі за пунктами меню: Settings - Developer settings - Personal access tokens - Tokens (classic).

І натисніть кнопку «Generate new token (classic)» і у вікні, що відкриється, поставте галочку і «геро».

Expiration

 30 days (Mar 18, 2026) ▾

The token will expire on the selected date

Select scopes

Scopes define the access for personal tokens. [Read more about OAuth scopes.](#)

<input checked="" type="checkbox"/> repo	Full control of private repositories
<input checked="" type="checkbox"/> repo:status	Access commit status
<input checked="" type="checkbox"/> repo_deployment	Access deployment status
<input checked="" type="checkbox"/> public_repo	Access public repositories
<input checked="" type="checkbox"/> repo:invite	Access repository invitations
<input checked="" type="checkbox"/> security_events	Read and write security events

І натиснуть кнопку . І ви отримаєте власний токен.

Personal access tokens (classic)

[Generate new token ▾](#)

Tokens you have generated that can be used to access the [GitHub API](#).

 Make sure to copy your personal access token now. You won't be able to see it again!

✓ `ghp_gUzxcg6Z1iJsi0NAJCnsyFVmyqEMqzi391odS` 

Delete

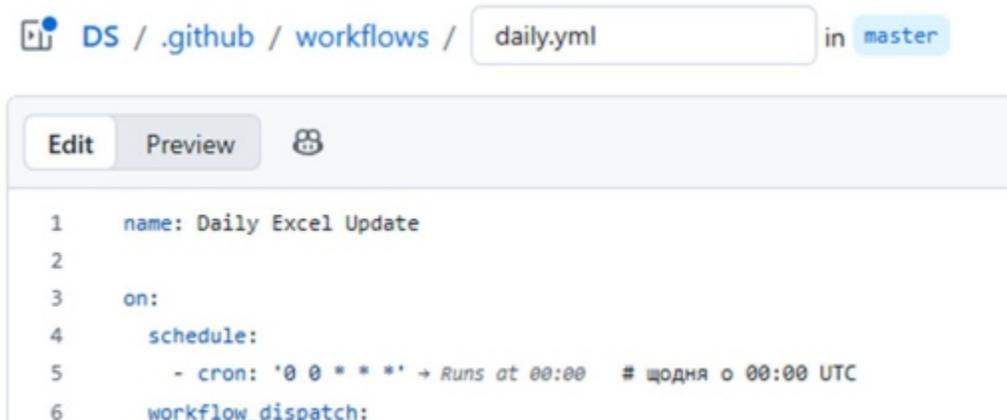
Personal access tokens (classic) function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to [authenticate to the API over Basic Authentication](#).

Скопіюйте його (негайно скопіюйте, бо варто закрити це вікно і прийдеться генерувати токен ще раз) і перейдіть до того репозиторію, де знаходиться ваша програма, у прикладі це DS.

Далі по меню: Settings - Secrets and variables - Actions - New repository secret. У вікнах що відкриється, написати Name: PAT_TOKEN, а там де Secret – вставити токен, а потім кнопку Create Secret.

Далі, у репозиторії **DS** необхідно перейти у вкладку **Code**, натиснути **Add file**, обрати **Create new file** і у полі назви файлу написати `.github/workflows/daily.yml`. Важливо написати саме так, з крапкою на початку

.github. Фактично назвою вашого файлу є daily.yml, а попередні символи є командою, що цей файл створено для виконання файлу на Пайтоні.



The screenshot shows a GitHub repository path: DS / .github / workflows / daily.yml in master. Below the path are buttons for 'Edit', 'Preview', and a GitHub logo. The code content is as follows:

```
1 name: Daily Excel Update
2
3 on:
4   schedule:
5     - cron: '0 0 * * *' → Runs at 00:00 # щодня о 00:00 UTC
6   workflow_dispatch:
```

Далі вставте програму

```
name: Daily Excel Update

on:
  schedule:
    - cron: '0 0 * * *' # щодня о 00:00 UTC
  workflow_dispatch:

jobs:
  run-script:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout DS repo
        uses: actions/checkout@v3

      - name: Set up Python
        uses: actions/setup-python@v4
        with:
          python-version: '3.10'
```

```
- name: Install dependencies
run: |
  pip install pandas openpyxl requests

- name: Run script
run: python 2/pursingakc.py

- name: Push Excel to A1 repository
env:
  TOKEN: ${{ secrets.PAT_TOKEN }}
run: |
  git clone https://$TOKEN@github.com/ТВИЙ_ЛОГІН/A1.git
  cp result.xlsx A1/
  cd A1
  git config user.name "github-actions"
  git config user.email "actions@github.com"
  git add result.xlsx
  git commit -m "Daily auto update"
  git push
```

У цій програмі є 2 команди `run`, перша має вигляд

```
- name: Run script
```

```
run: python 2/pursingakc.py
```

Виділений жирним текст є ознакою, з якої папки буде запускатися файл.

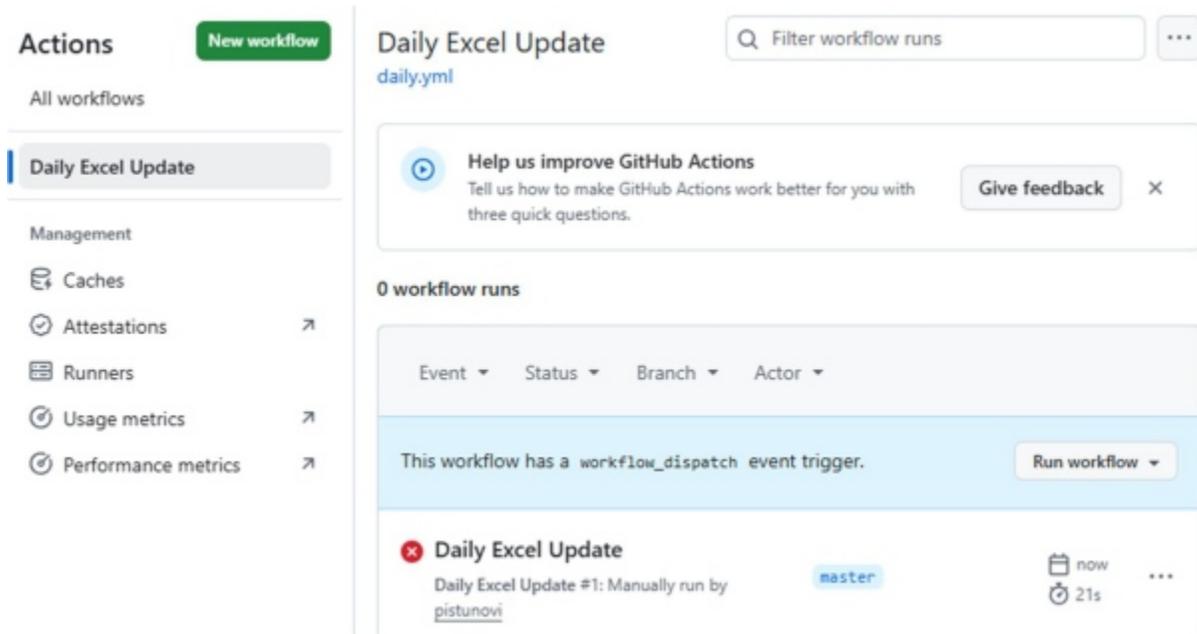
У кінці цієї програми є ще одна команда `run`: а нижче неї команда

```
git clone https://$TOKEN@github.com/ ТВИЙ_ЛОГІН/A1.git.
```

В ній `ТВИЙ_ЛОГІН` треба замінити на ваш логін у GitHub, наприклад

```
git clone https://\$TOKEN@github.com/pistunovi/A1.git
```

Далі необхідно натиснути кнопку **Commit changes** угорі ліворуч і у вікні, що відкриється, внизу вікна обрати **Commit directly to main** і натиснути **Commit changes**.



Запуск нашої програми виконується за схемою:

1. Відкрити вкладку **Actions** у репозиторії **DS**
2. Обрати workflow “Daily Excel Update”
3. Натиснути кнопку **Run workflow** (праворуч зверху)
4. Вибрати **branch** (зазвичай main)
5. Натиснути **Run workflow**

Після цього з’явиться перший запис про запуск workflow, і ви побачите логи виконання.

1.5. Індивідуальне завдання №1

Налаштування VS Code для роботи з GitHub

Критерії оцінювання: це завдання оцінюється у 5 балів за національною шкалою. За кожну помилку знімається 0,1 бали. Потім оцінка перераховується за 100-бальною системою згідно існуючого положення.

Мета роботи: Набути навичок у використанні програми VS Code разом з сервісом GitHub.

- Завдання:**
1. Встановити програму VS Code на власному комп'ютері.
 2. Зареєструватися на сервісі GitHub.
 3. Об'єднати бібліотеку програм на вашому комп'ютері з вашою бібліотекою на GitHub.
 4. Створити довільний текст, не обов'язково програмний.
 5. Зберегти його на GitHub.
 6. Зробити зміни у тексті і зберегти його на GitHub як іншу версію тексту.
 7. У звіті показати всі операції спільної роботи VS Code та GitHub з можливістю працювати з різними версіями тексту, який водночас є на вашому комп'ютері та у бібліотеці GitHub.

Контрольні запитання

1. Для чого ти використовуєш VS Code?
2. Які його переваги порівняно з іншими редакторами?
3. Які розширення (extensions) у VS Code ти використовуєш найчастіше і навіщо?

4. Що таке Git і чим він відрізняється від GitHub?
5. Опиши типовий робочий процес: як ти починаєш проєкт і завантажуєш його на GitHub з VS Code?
6. Що таке commit? Яку інформацію він повинен містити?
7. Як створити нову гілку (branch) і навіщо це робити?
8. Що таке pull request і коли його використовують?
9. Як ти вирішуєш конфлікти під час злиття (merge conflicts)?
10. Як підключити вже існуючий репозиторій GitHub до VS Code?
11. Як ти працюєш у команді через GitHub?
(Issues, pull requests, code review тощо)

Вивчивши матеріали цього розділу студент зможе ефективно створювати програми з можливістю оперувати з різними їх версіями.

Розділ 2.

ПАЙТОН ЗІ ШТУЧНИМ ІНТЕЛЕКТОМ

Цей розділ описує використання мови Python у прикладенні до можливостей штучного інтелекту.

Навіщо і яка практична цінність ШІ через Python? Адже можна це зробити просто через інтерфейс штучного інтелекту (ШІ). І чому саме ця моїа програмування?

Насправді Python – головна мова для ШІ, бо в нього є готові бібліотеки, які реально працюють:

- numpy, pandas – обробка даних;
- scikit-learn – машинне навчання;
- tensorflow, pytorch – нейромережі;
- opencv – обробка і створення зображення;

ШІ через Python може:

- автоматично перевіряти тексти
- сортувати файли
- аналізувати дані
- відповідати клієнтам (чат-боти)
- знаходить закономірності
- робить прогнози (ціни, попит, оцінки)
- допомагає приймати рішення
- переклад
- аналіз настрою
- чат-боти

- розпізнавання голосу
- розпізнавання облич
- номери авто
- об'єкти на фото/відео

При використанні Python можна почати з простого скрипта, потім перетворити в сервіс або продукт, з легкістю підключатися до сайтів, БД, API.

Отже, те, що людина робить годинами – код робить за секунди. ШІ через Python – це швидкий спосіб створювати розумні, корисні та автоматизовані рішення для реальних задач.

2.1. Доступ до ШІ через Python

Python є однією з основних мов програмування у сфері штучного інтелекту. Його популярність зумовлена простою синтаксичною структурою, широким набором спеціалізованих бібліотек та активною підтримкою спільноти розробників.

Використання Python у ШІ здійснюється такими основними шляхами:

- застосування бібліотек машинного навчання;
- доступ до віддалених сервісів через API;
- використання готових (попередньо навчених) моделей.

В наведеному нижче описі ви побачите приклади використання всіх методів, реалізовані на Python.

Застереження: Всі ці програми працюють за умови інсталяції відповідних бібліотек

2.1.1. Отримання ключа API для ШІ

При звертанні до можливостей **OpenAI** вимагається особистий ключ клієнта.

Щоб отримати одноразовий ключ необхідно:

- Зареєструйтесь на <https://platform.openai.com/api-keys>
- Перейдіть **Посилання API**.
- Натисніть **Create new secret key**
- У вікні, що відкриється, введіть довільний набір символів і/або чисел.
- Натисніть **Code Generate**, а потім кнопку **Copy**, що перенесе згенерований ключ в оперативну пам'ять.
- Відкрийте захищений файл і вставте у команду `client = OpenAI(api_key="YOUR_API_KEY")` замість `YOUR_API_KEY` свій ключ
- цей ключ, який має бути секретним, тільки для вас коли ви хочете комусь передати програму з ключем, ваш клієнт має згенерувати власний ключ, який ви і вставите у програму.

Цей ключ діятиме тільки для одноразового звертання. Щоб зробити його багаторазовим потрібно:

- Зайти знову на <https://platform.openai.com/api-keys>
- Натиснути кнопку **Start Building**
- Вказати тип вашого бізнес-проекту
- Обрати суму разової оплати
- Внести реквізити картки

Тепер кошти з вашої катки будуть зніматися автоматично тими, порціями, які ви вказали.

The image displays three sequential screenshots from the OpenAI Platform API keys management interface:

- Left Screenshot:** "Welcome to OpenAI Platform" form. It prompts the user to "Create an organization to generate API keys and start building." The "Organization name" field contains "Personell" and "What best describes you?" is set to "Engineer". A "Create organization" button is at the bottom.
- Middle Screenshot:** "Add some API credits" form. It asks the user to "Add a payment method to buy credits and unlock the full power of the API." Three radio button options are shown: "\$5 credits" (marked as "Recommended"), "\$10 credits", and "\$20 credits".
- Right Screenshot:** "Add a payment method" form. It requests credit card details: "Card number" (1234 1234 1234 1234), "Expiration date" (MM / YY), "Security code" (CVC), "Full name" (for platform), "Country or region" (dropdown), and "Address line 1". A checkbox for "I'm purchasing as a business" is present. "Cancel" and "Add payment method" buttons are at the bottom.

Деталізована таблиця типу інтерфейсу для кожного API / бібліотеки

Платформа / Бібліотека	Тип інтерфейсу	Пояснення	Приклади використання у Python
OpenAI API	Веб-API (REST)	Взаємодія через HTTP-запити; дані передаються у JSON	<pre>python\nfrom openai import OpenAI\nclient = OpenAI(api_key="YOUR_KEY")\nresp = client.chat.completions.create(model="gpt-4o-mini", messages=[{"role":"user","content":"Привіт"}])\nprint(resp.choices[0].message.content)\n</pre>
OpenAI API	Веб-API (REST)	Взаємодія через HTTP-запити; дані передаються у JSON	<pre>python\nfrom openai import OpenAI\nclient = OpenAI(api_key="YOUR_KEY")\nresp = client.chat.completions.create(model="gpt-4o-mini", messages=[{"role":"user","content":"Привіт"}])\nprint(resp.choices[0].message.content)\n</pre>
Azure AI	Веб-API (REST)	API від Microsoft для бізнес-аналітики	<pre>python\nimport requests\nurl = "https://<endpoint>.cognitiveservices.azure.com/text/analytics/v3.1/keyPhrases"\ndata = {"documents":[{"id":"1","text":"Привіт"}]}\nheaders = {"Ocp-Apim-Subscription-Key":"YOUR_KEY"}\nresp</pre>

		, NLP та CV; запити через HTTP	= requests.post(url,json=data,headers=headers)\nprint(resp.json())\n
спра Су (лок аль на NLP)	Лока льна Pyth оп- біблі отек а	Використовується прямо у кодї Python без серверів	python\nimport spacy\nnlp = spacy.load("en_core_web_sm")\ndoc = nlp("Apple купує стартап")\nprint([(ent.text, ent.label_) for ent in doc.ents])\n
Оре пCV (лок аль на обро бка зобр аже нь)	Лока льна Pyth оп- біблі отек а	Робота з зображеннями, відео та комп'ютерним зором без серверів	python\nimport cv2\nimage = cv2.imread("image.jpg")\ngray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)\ncv2.imshow('Gray', gray)\ncv2.waitKey(0)\ncv2.destroyAllWindows

Для **DeepSeek** ситуація аналогічна, звертання до його можливостей вимагає API-ключ.

Щоб отримати API ключ для DeepSeek:

1. Зареєструйтесь на platform.deepseek.com
2. Перейдіть в API Keys
3. Створіть новий ключ

Поточна політика DeepSeek API:

- Безкоштовний API ключ доступний після реєстрації.
- Щоденні ліміти на кількість запитів/токенів.
- Конкретні ліміти можуть змінюватись, але зазвичай достатньо для особистого використання та дрібних проєктів.

Реєстрація з якихось причин може не виконуватися через:

1. Геоблокування - деякі регіони можуть бути заблоковані.
2. Проблеми з акаунтом - облікові записи Google/Microsoft можуть не

підтримуватись.

3. VPN блокування - DeepSeek може блокувати VPN трафік.
4. Тимчасові технічні проблеми.

Тоді можливі наступні варіанти:

А) Через інші платформи, що підтримують DeepSeek:

- **OpenRouter** - openrouter.ai
- **Together AI** - together.ai
- **Fireworks AI** - fireworks.ai

В) Локальні альтернативи для навчання:

```
# 1. Використання локальних моделей через Ollama
# Встановіть Ollama: https://ollama.com/
# Запустіть модель:
# ollama run deepseek-coder
# 2. Безкоштовні альтернативи API:
# - OpenAI API (є безкоштовний кредит для нових акаунтів)
# - Google Gemini API (безкоштовний ліміт)
# 1. Зареєструйтесь на makersuite.google.com
# 2. Безкоштовний ліміт: 60 запитів/хв
import google.generativeai as genai
genai.configure(api_key="ваш_ключ")
model = genai.GenerativeModel('gemini-pro')
response = model.generate_content("Поясни цикли в Python")
# - Groq API (швидкий, безкоштовний ліміт) - підтримує Llama, Mixtral моделі.
```

Доступ до deepseek здійснюється так.

1. Базовий чат-бот:

```
# Простий приклад для студентів
import requests
def simple_chatbot(api_key, question):
    url = "https://api.deepseek.com/v1/chat/completions"
    headers = {
        "Authorization": f"Bearer {api_key}",
        "Content-Type": "application/json"
    }
    data = {
        "model": "deepseek-chat",
        "messages": [{"role": "user", "content": question}],
        "max_tokens": 500
    }
    response = requests.post(url, json=data, headers=headers)
    return response.json()["choices"][0]["message"]["content"]
# Використання
api_key = "ваш_ключ"
відповідь = simple_chatbot(api_key, "Поясни цикли в Python")
print(відповідь)
```

2. Помічник для перевірки коду:

```
def code_reviewer(api_key, student_code):  
    prompt = f"""  
    Перевір цей Python код студента та дай конструктивний відгук:  
    {student_code}  
    Знайди:  
    1. Синтаксичні помилки  
    2. Стилiстичні проблеми (PEP 8)  
    3. Потенційні покращення  
    """"  
    # Виклик API як у попередньому прикладі  
    # ...
```

Ось покрокова інструкція, як зареєструватися та увійти до **Grok** (grok.com) на комп'ютері під Windows через браузер, з реєстрацією по номеру телефону (де це можливо) або з підтвердженням телефону.

Увійти на сайт «grok.com» або «accounts.x.ai», а повноцінна реєстрація по чистому номеру телефону без email/X/Google/Apple трапляється рідко, але часто вимагають верифікацію телефону вже після створення акаунту.

Крок 1 – Перейди на сайт

Відкрий браузер → введи в адресний рядок:

«<https://grok.com>»

Натисни Enter.

Ти побачиш головну сторінку Grok з полем чату та кнопкою входу/реєстрації (зазвичай вгорі праворуч або посередині «Sign in» / «Try Grok» / «Start chatting»).

Крок 2 – Початок реєстрації

Натисни велику кнопку «Sign up» або «Create account» (якщо вже є акаунт – Sign in).

Крок 3 – Варіант з email + додавання телефону (найпоширеніший шлях)

1. Натисни «Sign up with email»

2. З'явиться вікно:

- Введи свою електронну пошту (наприклад, example@gmail.com)

- Натисни «Continue» або «Next».

3. Наступне вікно – створення пароля:

- Придумай надійний пароль (мінімум 8 символів, літери + цифри) або скористайся підказкою Grok.

- Натисни «Continue»

4. Перевір пошту – прийде лист від xAI з кодом (наприклад, 6 цифр)

- Введи цей код у вікно браузера → натисни «Verify» / «Confirm»

Якщо хочеш використовувати Grok без обмежень

- Безкоштовна версія має ліміти (кількість повідомлень на день).

- Для Grok 4 / Grok 4.1 / SuperGrok Heavy — потрібна підписка (SuperGrok ≈ \$30/міс).

Після входу на grok.com внизу або в налаштуваннях буде кнопка «Upgrade» → вибери план → оплати (карткою).

Що потрібно для старту:

- Зареєструйся на <https://x.ai> або <https://console.x.ai>

- Згенеруй API ключ (зазвичай платний доступ, є безкоштовні ліміти або trial)

- Додай ключ у змінну середовища `XAI_API_KEY` (або прямо в код, але не зберігай цей код у git!)

До Grok можна увійти (тобто використовувати його моделі) через Python – і це досить зручно робиться через офіційний сайт xAI API.

Станом на 2026 рік xAI має публічний API з підтримкою Grok 4, Grok 4.1 Fast та інших моделей. Є два основних способи підключення з Python:

1. Офіційний xAI Python SDK (рекомендовано, сучасний та зручний)

```
`pip install xai-sdk`
```

2. Сумісність з OpenAI SDK (дуже популярний варіант, бо майже копіює OpenAI формат)

Просто змінюєш `base_url` на ``https://api.x.ai/v1``

Ось простий приклад, як це виглядає (актуально на 2026 рік.):

```
# Спосіб 1: через офіційний xAI SDK
# pip install xai-sdk
import os
from xai_sdk import Client
from xai_sdk.chat import user, system
# Бери ключ з https://console.x.ai/team/default/api-keys
client = Client(api_key=os.getenv("XAI_API_KEY"))
response = client.chat.completions.create(
    model="grok-4",          # або grok-4-1-fast, grok-3 тощо
    messages=[
        system("Ти корисний український AI-помічник"),
        user("Привіт! Напиши короткий жарт про Python")
    ],
    temperature=0.7,
    max_tokens=200
)
print(response.choices[0].message.content)
```

Або через OpenAI-стиль (багато хто так робить):
```python
# pip install openai
from openai import OpenAI
import os
client = OpenAI(
    api_key=os.getenv("XAI_API_KEY"),
    base_url="https://api.x.ai/v1"
)
response = client.chat.completions.create(
    model="grok-4",
    messages=[
        {"role": "system", "content": "Ти крутий AI з почуттям гумору"},
        {"role": "user", "content": "Розкажи анекдот про програмістів"}
    ]
)
print(response.choices[0].message.content)
```
```

Найпростіше користуватися ШІ з назвою **Gemini**: окремо реєструватися "з нуля" зазвичай не потрібно, якщо у вас вже є звичайний **Google-акаунт** (Gmail). Напишіть <https://gemini.google.com/> і ви вже можете користуватися всіма можливостями цього ШІ.

Однак для того, щоб ваш Python-скрипт міг "спілкуватися" з моделлю, вам потрібно виконати кілька дій у спеціальному сервісі для розробників.

Що саме потрібно зробити:

1. Увійти в Google AI Studio: Перейдіть на сайт [aistudio.google.com](https://aistudio.google.com). Це основний робочий простір для розробників Gemini. Ви просто входите під своїм Google-акаунтом.
2. Прийняти умови використання: При першому вході сервіс попросить вас погодитися з правилами використання (Terms of Service).
3. Створити API-ключ: Натисніть кнопку "Get API key" у бічному меню. Це і є ваш "паспорт" для доступу до моделі з коду.

Важливі нюанси:

- Безкоштовний рівень (Free Tier): Google дозволяє користуватися Gemini безкоштовно, але з певними лімітами (наприклад, до 15 запитів на хвилину для моделі Flash).
- Конфіденційність: У безкоштовній версії ваші дані можуть використовуватися Google для покращення моделей. Якщо ви плануєте працювати з конфіденційною інформацією, варто розглянути платний рівень (Pay-as-you-go), де дані залишаються приватними.
- Доступність: Google AI Studio працює в Україні (крім тимчасово окупованих територій), тому проблем із доступом через звичайний інтернет бути не повинно.

Після цього треба виконати команду в терміналі:

```
pip install -U google-generativeai
```

Ось мінімальний приклад того, як надіслати запит до моделі:

```
import google.generativeai as genai
Налаштування API-ключа
genai.configure(api_key="ВАШ_API_KEY")
Вибір моделі (наприклад, gemini-1.5-flash)
model = genai.GenerativeModel('gemini-1.5-flash')
Генерація відповіді
response = model.generate_content("Привіт! Як справи?")
print(response.text)
```

Ви можете використовувати gemini-1.5-pro (для складних завдань) або gemini-1.5-flash (швидша та легша). Через Python можна передавати не лише текст, а й зображення, аудіо, відео та PDF-файли.

Але ніколи не "зашивайте" API-ключ прямо в код, якщо плануєте публікувати його (наприклад, на GitHub).

Краще використовуйте змінні оточення або .env файли. Для цього створіть **файл .env**

У кореневій папці вашого проєкту створіть файл з назвою .env (важливо: назва починається з крапки). Запишіть туди свій ключ:

```
Plaintext
```

```
GEMINI_API_KEY=ваш_справжній_ключ_тут
```

Якщо ви використовуєте Git, обов'язково створіть файл .gitignore і додайте туди рядок:

```
Plaintext
```

```
.env
```

Це дасть вказівку Git ігнорувати цей файл і не завантажувати його в репозиторій. Тепер у вашому скрипті ви використовуєте бібліотеку os та dotenv для отримання ключа.

Якщо ви працюєте в **PyCharm** або **VS Code**, вони зазвичай підсвічують .env файли та допомагають зручно ними керувати через плагіни.

```

import os
import google.generativeai as genai
from dotenv import load_dotenv
1. Завантажуємо змінні з файлу .env
load_dotenv()
2. Отримуємо значення ключа зі змінних оточення
api_key = os.getenv("GEMINI_API_KEY")
3. Налаштовуємо Gemini
if api_key:
 genai.configure(api_key=api_key)
 print("API ключ успішно завантажено!")
else:
 print("Помилка: API ключ не знайдено в файлі .env")
Далі робота з моделлю...
model = genai.GenerativeModel('gemini-1.5-flash')

```

### 2.1.2. Обробка даних

Обробка даних є необхідним етапом перед використанням алгоритмів штучного інтелекту.

Таблиця 2.1

#### Основні бібліотеки Python для обробки даних

| Назва бібліотеки | Призначення                 | Роль у ШІ          | Приклади застосування                                                                                                                                                                                                                                                      |
|------------------|-----------------------------|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NumPy</b>     | Робота з числовими масивами | Математична основа | <ul style="list-style-type: none"> <li>- Обчислення матриць для нейронних мереж</li> <li>- Лінійна алгебра для алгоритмів машинного навчання</li> <li>- Побудова числових моделей для прогнозування фінансових даних</li> <li>- Підготовка даних для ML-моделей</li> </ul> |

|                   |                          |                          |                                                                                                                                                                                                                                                                               |
|-------------------|--------------------------|--------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Pandas</b>     | Табличні дані            | Підготовка наборів даних | <ul style="list-style-type: none"> <li>- Збір та обробка великих наборів даних</li> <li>- Фільтрація, агрегування та трансформація даних</li> <li>- Підготовка датасетів для навчання моделей</li> <li>- Аналіз економічних і фінансових показників</li> </ul>                |
| <b>Matplotlib</b> | Побудова графіків        | Аналіз результатів       | <ul style="list-style-type: none"> <li>- Візуалізація результатів моделей машинного навчання</li> <li>- Побудова графіків прогнозів та трендів</li> <li>- Аналіз фінансових та статистичних показників</li> <li>- Візуалізація розподілу даних</li> </ul>                     |
| <b>Seaborn</b>    | Статистична візуалізація | Виявлення залежностей    | <ul style="list-style-type: none"> <li>- Виявлення кореляцій та залежностей між змінними</li> <li>- Побудова теплових карт та діаграм розсіювання</li> <li>- Аналіз сезонних та трендових патернів</li> <li>- Підготовка інтуїтивно зрозумілих графіків для звітів</li> </ul> |

Приклади програмного коду до Таблиці 2.1

**NumPy** використовується для швидких математичних обчислень, які лежать в основі алгоритмів машинного навчання.

Завдання: Обчислити середнє значення та стандартне відхилення (основа аналізу даних у ШІ).

Код:

```
import numpy as np # імпорт бібліотеки NumPy
створення числового масиву (наприклад, результати тестів)
scores = np.array([75, 82, 90, 68, 88])
обчислення середнього значення
mean_score = np.mean(scores)
обчислення стандартного відхилення
std_score = np.std(scores)
print("Середнє значення:", mean_score)
print("Стандартне відхилення:", std_score)
```

Результат:

```
Середнє значення: 80.6
Стандартне відхилення: 8.187795796183488
```

### **Pandas – формування таблиць спостережень.**

Завдання: Створити таблиці спостережень та аналіз даних. Pandas дозволяє працювати з таблицями спостережень — ключовим елементом у задачах прогнозування.

Код:

```
import pandas as pd # імпорт бібліотеки Pandas
створення таблиці спостережень
data = {
 "Ім'я": ["Анна", "Богдан", "Ірина"],
 "Відвідуваність (%)": [90, 75, 100],
 "Оцінка": [88, 74, 95]
}
df = pd.DataFrame(data) # створення DataFrame
виведення таблиці
print(df)
обчислення середньої оцінки
average_grade = df["Оцінка"].mean()
print("Середня оцінка:", average_grade)
```

## Результат

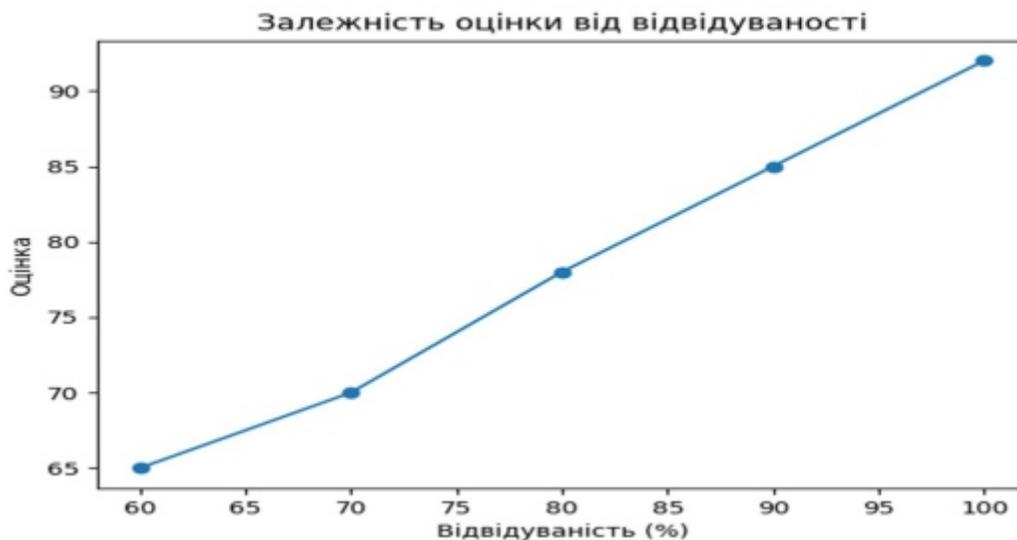
|                 | Ім'я   | Відвідуваність (%) | Оцінка            |
|-----------------|--------|--------------------|-------------------|
| 0               | Анна   | 90                 | 88                |
| 1               | Богдан | 75                 | 74                |
| 2               | Ірина  | 100                | 95                |
| Середня оцінка: |        |                    | 85.66666666666667 |

**Matplotlib** – візуалізація даних. Візуалізація допомагає виявляти закономірності перед застосуванням алгоритмів ШІ.

Завдання: Побудувати графік залежності оцінки від відвідуваності.

```
import matplotlib.pyplot as plt # імпорт бібліотеки Matplotlib
дані для побудови графіка
attendance = [60, 70, 80, 90, 100]
grades = [65, 70, 78, 85, 92]
побудова графіка
plt.plot(attendance, grades, marker='o')
підписи осей
plt.xlabel("Відвідуваність (%)")
plt.ylabel("Оцінка")
заголовок графіка
plt.title("Залежність оцінки від відвідуваності")
відображення графіка
plt.show()
```

Результат:



**Seaborn** – статистичний аналіз і кореляції. Seaborn дозволяє наочно побачити, які змінні найбільше впливають одна на одну — це важливо для вибору ознак у ШІ.

Завдання: Побудувати теплову карту кореляції між параметрами.

Код:

```
import seaborn as sns # імпорт бібліотеки Seaborn
import pandas as pd
import matplotlib.pyplot as plt
створення таблиці даних
data = {
 "Відвідуваність": [60, 70, 80, 90, 100],
 "Години підготовки": [2, 3, 5, 6, 8],
 "Оцінка": [65, 70, 78, 85, 92]
}
df = pd.DataFrame(data)
обчислення кореляційної матриці
correlation = df.corr()
побудова теплової карти
sns.heatmap(correlation, annot=True, cmap="coolwarm")
заголовок
plt.title("Кореляція між показниками успішності")
показ графіка
plt.show()
```

Результат:



### 2.1.3. Машинне навчання

Машинне навчання є підрозділом штучного інтелекту, що дозволяє системам навчатися на основі даних.

Таблиця 2.2

Бібліотеки машинного навчання Python

| Бібліотек а         | Харак терис тика    | Типові задачі          | Приклади застосування                                                                                                                                                                                                                                                                                             |
|---------------------|---------------------|------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>scikit-learn</b> | Класичні алгоритми  | Класифікація, регресія | <ul style="list-style-type: none"> <li>- Класифікація клієнтів за ризиком у фінансових установах</li> <li>- Регресія для прогнозування продажів та цін</li> <li>- Виявлення аномалій у фінансових або виробничих даних</li> <li>- Побудова простих моделей для навчальних або демонстраційних проектів</li> </ul> |
| <b>XGBoost</b>      | Підсилені алгоритми | Прогнозування          | <ul style="list-style-type: none"> <li>- Прогнозування попиту та продажів у ритейлі</li> <li>- Прогнозування кредитного скорингу клієнтів</li> <li>- Побудова моделей для оцінки ризиків і шахрайства</li> <li>- Участь у змаганнях з машинного навчання (Kaggle)</li> </ul>                                      |
| <b>LightGBM</b>     | Оптимізовані моделі | Великі набори даних    | <ul style="list-style-type: none"> <li>- Прогнозування поведінки клієнтів у великих базах даних</li> <li>- Аналітика великих фінансових потоків</li> <li>- Оптимізація рекомендаційних систем</li> <li>- Робота з Big Data для маркетингових і бізнес-аналітичних задач</li> </ul>                                |

|                         |                               |                             |                                                                                                                                                                                          |
|-------------------------|-------------------------------|-----------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Statsmode<br/>Is</b> | Стати<br>стичні<br>метод<br>и | Аналіз<br>часови<br>х рядів | - Аналіз економічних та фінансових часових рядів<br>- Прогнозування сезонних продажів<br>- Оцінка ефективності маркетингових кампаній<br>- Статистичне моделювання та тестування гіпотез |
|-------------------------|-------------------------------|-----------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

Приклади програмного коду до Таблиці 2.2. Усі приклади розв’язують одну логічну задачу – прогноз числового значення (регресія). Це полегшує порівняння бібліотек.

**scikit-learn** – Класична лінійна регресія. scikit-learn – базова бібліотека для вивчення ML, рекомендована як перша. При інсталяції цій пакет називається як scikit-learn. А у команді import – sklearn.

Метод: Machine Learning (ML)

Код:

```
from sklearn.linear_model import LinearRegression
import numpy as np
вхідні дані (наприклад, години навчання)
X = np.array([1, 2, 3, 4, 5]).reshape(-1, 1)
вихідні дані (оцінка)
y = np.array([60, 65, 70, 80, 85])
створення моделі
model = LinearRegression()
навчання моделі
model.fit(X, y)
прогноз для нового значення
prediction = model.predict([[6]])
print("Прогноз оцінки:", prediction[0])
```

Результат:

Прогноз оцінки: 91.5

**XGBoost** – Підсилене машинне навчання (Boosting). XGBoost широко використовується в аналітиці та змаганнях з ML через високу точність.

Метод: Ensemble Learning

## Застосування: точне прогнозування

```
import xgboost as xgb
import numpy as np
дані
X = np.array([[1], [2], [3], [4], [5]])
y = np.array([60, 65, 70, 80, 85])
створення моделі
model = xgb.XGBRegressor(
 objective="reg:squarederror",
 n_estimators=50
)
навчання
model.fit(X, y)
прогноз
prediction = model.predict([[6]])
print("Прогноз оцінки:", prediction[0])
```

Результат:

Прогноз оцінки: 84.99605

**LightGBM** – Швидке машинне навчання на великих даних. Оптимізація швидкості. LightGBM ефективний для великих наборів даних і використовується в промислових системах. Метод: Gradient Boosting

```
import lightgbm as lgb
import numpy as np
дані
X = np.array([[1], [2], [3], [4], [5]])
y = np.array([60, 65, 70, 80, 85])
створення моделі
model = lgb.LGBMRegressor(n_estimators=50)
навчання
model.fit(X, y)
прогноз
prediction = model.predict([[6]])
print("Прогноз оцінки:", prediction[0])
```

Результат:

Прогноз оцінки: 72.0

**Statsmodels** – Статистична регресія (OLS), Statsmodels використовується, коли важливі пояснення та статистичні показники, а не лише прогноз.

Метод: математична статистика

Код:

```
import statsmodels.api as sm
import numpy as np
дані
X = np.array([1, 2, 3, 4, 5])
y = np.array([60, 65, 70, 80, 85])
додавання константи (обов'язково)
X = sm.add_constant(X)
створення моделі
model = sm.OLS(y, X)
навчання
results = model.fit()
виведення статистичного звіту
print(results.summary())
```

Результат:

### OLS Regression Results

```
=====
Dep. Variable: y R-squared: 0.983
Model: OLS Adj. R-squared: 0.977
Method: Least Squares F-statistic: 169.0
Date: Thu, 29 Jan 2026 Prob (F-statistic): 0.000983
Time: 15:59:04 Log-Likelihood: -8.1084
No. Observations: 5 AIC: 20.22
Df Residuals: 3 BIC: 19.44
Df Model: 1
Covariance Type: nonrobust
=====
```

```
=====
 coef std err t P>|t| [0.025 0.975]

const 52.5000 1.658 31.659 0.000 47.223 57.777
x1 6.5000 0.500 13.000 0.001 4.909 8.091
=====
```

```

=====
Omnibus: nan Durbin-Watson: 2.533
Prob(Omnibus): nan Jarque-Bera (JB): 0.361
Skew: -0.408 Prob(JB): 0.835
Kurtosis: 1.967 Cond. No. 8.37
=====

```

Notes: Standard Errors assume that the covariance matrix of the errors is correctly specified

#### 2.1.4. Глибинне навчання

Глибинне навчання базується на нейронних мережах та використовується для складних інтелектуальних задач.

**Таблиця 2.3**

**Бібліотеки глибинного навчання**

| Бібліотека        | Розробник | Призначення                  | Приклади застосування                                                                                                                                                                                                                                                                                                                           |
|-------------------|-----------|------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>TensorFlow</b> | Google    | Побудова нейромереж          | <ul style="list-style-type: none"> <li>- Створення глибоких нейронних мереж для прогнозування фінансових показників</li> <li>- Розробка моделей комп'ютерного зору для аналізу зображень продуктів або документів</li> <li>- NLP-моделі для аналізу текстів, резюме або звітів</li> <li>- Прогнозування продажів та попиту на товари</li> </ul> |
| <b>Keras</b>      | Google    | Спрощена робота з нейромереж | - Швидке прототипування моделей машинного навчання                                                                                                                                                                                                                                                                                              |

|                |      |                                  |                                                                                                                                                                                                                                                                                                                          |
|----------------|------|----------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                |      |                                  | <ul style="list-style-type: none"> <li>- Створення моделей для класифікації тексту та зображень</li> <li>- Розробка навчальних моделей для студентів і дослідників</li> <li>- Розробка простих моделей для автоматизації бізнес-аналітики</li> </ul>                                                                     |
| <b>PyTorch</b> | Meta | Навчальні та наукові дослідження | <ul style="list-style-type: none"> <li>- Розробка дослідницьких моделей для наукових публікацій</li> <li>- Експериментальні нейронні мережі для NLP та CV</li> <li>- Навчальні моделі для університетських курсів та практичних завдань</li> <li>- Моделювання фінансових даних і побудова прогнозних моделей</li> </ul> |

Приклади програмного коду до Таблиці 2.3.

Для розуміння відмін у роботі різних методів створимо одну задачу для всіх них: потрібно **спрогнозувати ціну об'єкта комерційної нерухомості** на основі таких факторів:

| <b>Змінна</b> | <b>Зміст</b>                             |
|---------------|------------------------------------------|
| x1            | Площа об'єкта (м <sup>2</sup> )          |
| x2            | Відстань до центру міста (км)            |
| x3            | Вік будівлі (роки)                       |
| x4            | Клас об'єкта (A/B/C → числове кодування) |
| x5            | Рівень ділової активності району         |
| y             | Ціна об'єкта (тис. грн)                  |

С початку виконаємо генерацію економічних даних (спільна для всіх бібліотек) і запишемо ці дані на диск.

Код:

```
import numpy as np
np.random.seed(42)
n = 200 # кількість об'єктів нерухомості
Генерація факторів
X = np.column_stack((
 np.random.uniform(50, 500, n), # площа
 np.random.uniform(0.5, 15, n), # відстань до центру
 np.random.uniform(0, 40, n), # вік будівлі
 np.random.randint(1, 4, n), # клас об'єкта
 np.random.uniform(0.3, 1.0, n) # ділова активність
))
Ціна як економічна залежність + шум
y = (
 0.8 * X[:, 0] - 15 * X[:, 1] - 2 * X[:, 2]
 + 30 * X[:, 3] + 100 * X[:, 4]
 + np.random.normal(0, 20, n)
)
y = y.reshape(-1, 1)
Збереження на диск
np.savez("economic_data.npz", X=X, y=y)
print("Дані збережено у файлі 'economic_data.npz'")
```

Результат:

```
Дані збережено у файлі 'economic_data.npz'
```

Тепер запусимо різні методи, щоб обробити ці дані.

Метод **TensorFlow**

Код:

```

import numpy as np
import tensorflow as tf
from tensorflow.keras import models, layers
Завантаження даних
data = np.load("economic_data.npz")
X = data["X"]
y = data["y"]
Побудова моделі
model_tf = models.Sequential([
 layers.Dense(32, activation='relu', input_shape=(5,)),
 layers.Dense(16, activation='relu'),
 layers.Dense(1)
])
model_tf.compile(optimizer='adam', loss='mse', metrics=['mae'])
Навчання
model_tf.fit(X, y, epochs=50, batch_size=16, verbose=0)
Прогноз для перших 5 об'єктів
pred_tf = model_tf.predict(X[:5])
print("TensorFlow прогнози:\n", pred_tf)

```

Метод **Keras**.

Код:

```

import numpy as np
from keras.models import Sequential
from keras.layers import Dense
Завантаження даних
data = np.load("economic_data.npz")
X = data["X"]
y = data["y"]
Побудова моделі
model_keras = Sequential([
 Dense(32, activation='relu', input_dim=5),
 Dense(16, activation='relu'),
 Dense(1)
])
model_keras.compile(optimizer='adam', loss='mean_squared_error')
Навчання
model_keras.fit(X, y, epochs=50, batch_size=16, verbose=0)
Прогноз для перших 5 об'єктів
pred_keras = model_keras.predict(X[:5])
print("Keras прогнози:\n", pred_keras)

```

## Метод PyTorch.

Код:

```
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
Завантаження даних
data = np.load("economic_data.npz")
X = torch.tensor(data["X"], dtype=torch.float32)
y = torch.tensor(data["y"], dtype=torch.float32)
Модель
class PriceModel(nn.Module):
 def __init__(self):
 super().__init__()
 self.net = nn.Sequential(
 nn.Linear(5, 32),
 nn.ReLU(),
 nn.Linear(32, 16),
 nn.ReLU(),
 nn.Linear(16, 1)
)
 def forward(self, x):
 return self.net(x)
model_pt = PriceModel()
criterion = nn.MSELoss()
optimizer = optim.Adam(model_pt.parameters(), lr=0.01)
Навчання
for epoch in range(50):
 optimizer.zero_grad()
 outputs = model_pt(X)
 loss = criterion(outputs, y)
 loss.backward()
 optimizer.step()
Прогноз
pred_pt = model_pt(X[:5]).detach().numpy()
print("PyTorch прогнози:\n", pred_pt)
```

Результат роботи всіх трьох методів:

```

TensorFlow прогнози: Keras прогнози: PyTorch прогнози:
[[145.05296] [[131.77284] [[153.6247]
[359.7067] [378.2338] [339.6267]
[291.06564] [302.90472] [269.5792]
[177.34465] [158.95303] [224.05804]
[53.21799]] [37.125305]] [83.695]]

```

### 2.1.5. Готові, попередньо навчені моделі

Готові (попередньо навчені) моделі дозволяють використовувати ШІ без необхідності навчання моделей з нуля.

Таблиця 2.4

#### Платформи готових моделей

| Платформа              | Типи моделей   | Застосування             | Приклади застосування                                                                                                                                                                                                                                                                |
|------------------------|----------------|--------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Hugging Face</b>    | NLP,<br>Vision | Аналіз текстів           | <ul style="list-style-type: none"> <li>- Класифікація фінансових документів та звітів</li> <li>- Виявлення тональності відгуків клієнтів</li> <li>- Розпізнавання іменованих сутностей у текстах (NER)</li> <li>- Обробка юридичних або економічних текстів для аналітики</li> </ul> |
| <b>OpenAI DeepSeek</b> | Текст,<br>код  | Інтелектуальні асистенти | <ul style="list-style-type: none"> <li>- Автоматичні відповіді клієнтам у чат-ботах</li> <li>- Генерація аналітичних звітів і прогнозів</li> <li>- Написання та перевірка коду для фінансових моделей</li> <li>- Автоматичне створення навчальних матеріалів і пояснень</li> </ul>   |

|              |            |                         |                                                                                                                                                                                                                                                                                                                               |
|--------------|------------|-------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>spaCy</b> | <b>NLP</b> | Лінгвістичний<br>аналіз | <ul style="list-style-type: none"> <li>- Витяг іменованих сутностей (компанії, продукти, дати) з текстів</li> <li>- Класифікація та структуризація фінансових і юридичних документів</li> <li>- Пошук ключових слів і фраз у великих текстових масивах</li> <li>- Попередня обробка текстів для прогнозних моделей</li> </ul> |
|--------------|------------|-------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

Приклади програмного коду до Таблиці 2.4. Тут водночас продемонстровані можливості всіх бібліотек.

```
--- Підготовка даних ---
import numpy as np
import pandas as pd
import cv2
import matplotlib.pyplot as plt
Тексти для аналізу
texts = [
 "Економічні дані свідчать про зростання ВВП у наступному кварталі.",
 "Ціни на нерухомість знизилися через низький попит.",
 "Інфляція може вплинути на ставки оренди комерційної нерухомості."
]
Зображення для OpenCV
image_path = "office_building.jpg" # замініть на своє зображення
--- Hugging Face: аналіз тональності ---
from transformers import pipeline
hf_classifier = pipeline("sentiment-analysis")
hf_results = hf_classifier(texts)
hf_df = pd.DataFrame([{"Text": t, "Label": r["label"], "Score": r["score"]}
 for t, r in zip(texts, hf_results)])
print("Hugging Face Sentiment Analysis:\n", hf_df, "\n")
```

```

--- OpenAI: інтелектуальний асистент ---
from openai import OpenAI
client = OpenAI(api_key="YOUR_API_KEY") # замініть на свій ключ
openai_responses = []
for t in texts:
 resp = client.chat.completions.create(
 model="gpt-4o-mini",
 messages=[
 {"role": "system", "content": "Ти економічний аналітик."},
 {"role": "user", "content": t}
]
)
 openai_responses.append(resp.choices[0].message.content)
openai_df = pd.DataFrame({"Text": texts, "OpenAI_Response": openai_responses})
print("OpenAI Responses:\n", openai_df, "\n")
--- spaCy: лінгвістичний аналіз ---
import spacy
nlp = spacy.load("en_core_web_sm")
spacy_results = []
for t in texts:
 doc = nlp(t)
 entities = [(ent.text, ent.label_) for ent in doc.ents]
 spacy_results.append({"Text": t, "Entities": entities})
spacy_df = pd.DataFrame(spacy_results)
print("spaCy Named Entities:\n", spacy_df, "\n")
--- OpenCV: обробка зображень ---
image = cv2.imread(image_path)
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
edges = cv2.Canny(gray, 50, 150)
Візуалізація зображення та контурів
plt.figure(figsize=(10,5))
plt.subplot(1,2,1)
plt.title("Original Image")
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.axis("off")
plt.subplot(1,2,2)
plt.title("Edges Detected")
plt.imshow(edges, cmap='gray')
plt.axis("off")
plt.show()

```

Для цього прикладу, щоб його виконати, застосуємо прийом моку, тобто імітації роботи системи **openai**. Для цього блок попереднього коду

```
--- OpenAI: інтелектуальний асистент --- замінимо на наступний.
from unittest.mock import MagicMock
import pandas as pd
Припустимо, що у вас є список текстів
texts = ["Що таке рівноважна ціна?", "Як змінюється попит при зростанні
ціни?"]
Імітуємо OpenAI клієнт
class MockClient:
 class chat:
 class completions:
 @staticmethod
 def create(model, messages):
 # Повертаємо "фейкову" відповідь
 user_message = messages[-1]["content"]
 return MagicMock(
 choices=[
 MagicMock(
 message=MagicMock(content=f"Відповідь на: {user_message}")
)
]
)
Використовуємо замість реального OpenAI клієнта
client = MockClient()
Ваш код із замोकаним викликом
openai_responses = []
for t in texts:
 resp = client.chat.completions.create(
 model="gpt-4o-mini",
 messages=[
 {"role": "system", "content": "Ти економічний аналітик."},
 {"role": "user", "content": t}
]
)
 openai_responses.append(resp.choices[0].message.content)
openai_df = pd.DataFrame({"Text": texts, "OpenAI_Response": openai_responses})
print("OpenAI Responses:\n", openai_df, "\n")
```

Оскільки зображення теж не задано, то замість блоку програми

```
--- OpenCV: обробка зображень --- використаємо його мок у вигляді
import cv2
```

```

Замість реального файлу створимо випадкове зображення 200x300
image = np.random.randint(0, 256, (200, 300, 3), dtype=np.uint8)
Перетворимо у відтінки сірого та знайдемо контури
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
edges = cv2.Canny(gray, 50, 150)
Візуалізація
plt.figure(figsize=(10,5))
plt.subplot(1,2,1)
plt.title("Original Image (Fake)")
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.axis("off")
plt.subplot(1,2,2)
plt.title("Edges Detected")
plt.imshow(edges, cmap='gray')
plt.axis("off")
plt.show()

```

Результат:

No model was supplied, defaulted to distilbert/distilbert-base-uncased-finetuned-sst-2-english and revision 714eb0f.

Using a pipeline without specifying a model name and revision in production is not recommended.

Loading weights: 100%  104/104 [00:00<00:00, 3272.01it/s, Materializing param=pre\_cla

Hugging Face Sentiment Analysis:

|   | Text                                              | Label    | Score    |
|---|---------------------------------------------------|----------|----------|
| 0 | Економічні дані свідчать про зростання ВВП у н... | NEGATIVE | 0.566825 |
| 1 | Ціни на нерухомість знизилися через низький по... | POSITIVE | 0.706236 |
| 2 | Інфляція може вплинути на ставки оренди комерц... | POSITIVE | 0.715279 |

OpenAI Responses:

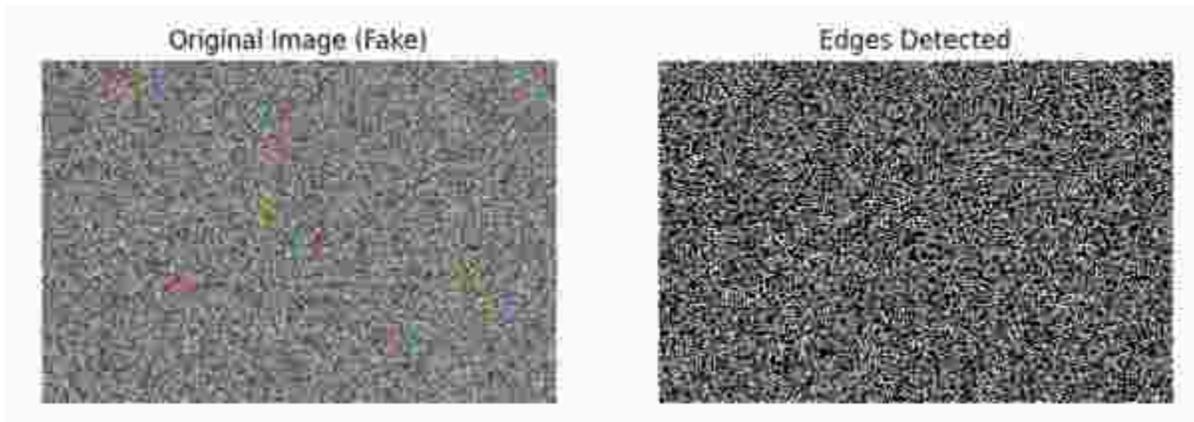
|   | Text                     | OpenAI_Response                        |
|---|--------------------------|----------------------------------------|
| 0 | Що таке рівноважна ціна? | Відповідь на: Що таке рівноважна ціна? |

1 Як змінюється попит при зростанні ціни? Відповідь на: Як змінюється попит при зростанн...

spaCy Named Entities:

|   | Text                                    | Entities                   |
|---|-----------------------------------------|----------------------------|
| 0 | Що таке рівноважна ціна?                | []                         |
| 1 | Як змінюється попит при зростанні ціни? | [(зростанні ціни, PERSON)] |

Також з'явилися зображення



### 2.1.6. Інтелектуальні сервіси

API дозволяє використовувати інтелектуальні сервіси через мережу Інтернет.

Таблиця 2.5

#### Приклади API для роботи з ШІ

| Тип інтерфейсу    | Можливості            | Сфера застосування   | Приклади застосування                                                                                                                                                               |
|-------------------|-----------------------|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>OpenAI API</b> | Генерація тексту, NLP | Чат-боти, аналітик а | - Автоматичне складання фінансових звітів та прогнозів<br>- Генерація пояснень економічних моделей для студентів або клієнтів<br>- Автоматичні відповіді у бізнес-чати та підтримку |

|                                            |                                     |                             |                                                                                                                                                                                                                                                                                         |
|--------------------------------------------|-------------------------------------|-----------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                            |                                     |                             | - Підготовка маркетингових текстів або презентацій                                                                                                                                                                                                                                      |
| <b>Azure AI</b>                            | Бізнес-аналітика, обробка даних     | Прогнозування, візуалізація | - Прогнозування продажів та попиту на товари/послуги<br>- Виявлення аномалій у фінансових даних і звітах<br>- Створення дашбордів для керівництва та аналітичних відділів<br>- Автоматичне узагальнення великих масивів бізнес-даних                                                    |
| <b>spracy (локальна NLP)</b>               | NER, POS, лінгвістичний аналіз      | Аналіз тексту               | - Витяг іменованих сутностей (компанії, продукти, дати) з економічних текстів<br>- Класифікація фінансових та юридичних документів<br>- Пошук ключових слів для аналітики ринку<br>- Структурування даних для моделювання прогнозів                                                     |
| <b>OpenCV (локальна обробка зображень)</b> | Комп'ютерний зір, обробка зображень | Візуалізація, детекція      | - Виявлення графіків, таблиць та діаграм на зображеннях фінансових документів<br>- Детекція контурів та об'єктів у виробничих або ринкових даних<br>- Підготовка даних для OCR та цифрового аналізу документів<br>- Автоматичний аналіз товарів на фотографіях для маркетингових звітів |

Порівняння способів доступу до ШІ наведено у табл. 2.6.

**Таблиця 2.6**

**Порівняльна характеристика підходів**

| <b>Підхід</b>       | <b>Переваги</b>       | <b>Недоліки</b>       |
|---------------------|-----------------------|-----------------------|
| Локальні бібліотеки | Повний контроль       | Потребують ресурсів   |
| API                 | Простота використання | Залежність від мережі |
| Готові моделі       | Швидкий результат     | Обмежена гнучкість    |

## 2.2. Збір, очищення та підготовка даних

Всі дії будемо виконувати на прикладі операцій з даними фондової біржі США для таких компаній як: Boeing, Chevron, Caterpillar, Microsoft, Walt Disney, Coca-Cola, McDonald's, Amazon.com, NVIDIA, Walmart, IBM, Nike.

Перша дія – парсинг даних на свій комп'ютер через можливості Python з 1.01.2023 по 2.02.2026 за розрахунком дохідності (різниці між поточним закриттям торгів та попередньою датою).

```
import yfinance as yf
import pandas as pd
from datetime import date
Список компаній та їх тикери на Yahoo Finance
tickers = {
 "Boeing": "BA",
 "Chevron": "CVX",
 "Caterpillar": "CAT",
 "Microsoft": "MSFT",
 "Walt Disney": "DIS",
 "Coca-Cola": "KO",
 "McDonald's": "MCD",
 "Amazon.com": "AMZN",
 "NVIDIA": "NVDA",
 "Walmart": "WMT",
 "IBM": "IBM",
 "Nike": "NKE"
}
Період збору даних
start_date = "2023-01-01"
end_date = date.today().strftime("%Y-%m-%d") # сьогодні
all_data = []
for name, ticker in tickers.items():
 df = yf.Ticker(ticker).history(start=start_date, end=end_date)
 df.reset_index(inplace=True)
 # Робимо datetime timezone-naive
 df["Date"] = df["Date"].dt.tz_localize(None)
 # Додаємо назву компанії і тикер
 df["Company"] = name
 df["Ticker"] = ticker
 # Денна дохідність
 df["Daily_Return"] = df["Close"].pct_change()
```

```

all_data.append(df)
Об'єднуємо всі компанії
final_df = pd.concat(all_data, ignore_index=True)
Сортування блоками
final_df = final_df.sort_values(by=["Company", "Date"])
Вибір стовпців
final_df = final_df[[
 "Company", "Ticker", "Date", "Open", "High", "Low", "Close", "Volume",
 "Daily_Return"
]]
Збереження у Excel
file_path = r"D:\A1\stocks_parsing_part1_with_returns.xlsx"
final_df.to_excel(file_path, index=False)
print("Файл з історичними цінами та денними дохідностями збережено:",
file_path)

```

Результат роботи:

| Company  | Ticker | Date                | Open  | High  | Low   | Close | Volume   | Daily_Return |
|----------|--------|---------------------|-------|-------|-------|-------|----------|--------------|
| Amazon.c | AMZN   | 2023-01-03 00:00:00 | 85,46 | 86,96 | 84,21 | 85,82 | 76706000 |              |
| Amazon.c | AMZN   | 2023-01-04 00:00:00 | 86,55 | 86,98 | 83,36 | 85,14 | 68885100 | -0,00792     |
| Amazon.c | AMZN   | 2023-01-05 00:00:00 | 85,33 | 85,42 | 83,07 | 83,12 | 67930800 | -0,02373     |

Тепер напишемо код для виконання тієї ж задачі через офіційний API Yahoo Finance (RapidAPI).

```

import requests
import pandas as pd
from datetime import datetime, date
1. Список компаній та тикерів
tickers = {
 "Boeing": "BA",
 "Chevron": "CVX",
 "Caterpillar": "CAT",
 "Microsoft": "MSFT",
 "Walt Disney": "DIS",
 "Coca-Cola": "KO",
 "McDonald's": "MCD",
 "Amazon.com": "AMZN",
 "NVIDIA": "NVDA",
 "Walmart": "WMT",
 "IBM": "IBM",
 "Nike": "NKE"
}

```

```

2. Період збору даних
start_date = int(datetime(2023, 1, 1).timestamp())
end_date = int(datetime.today().timestamp())

3. Параметри API (Yahoo Finance через RapidAPI)
url_template = "https://yahoo-
finance15.p.rapidapi.com/api/yahoo/historical/{ticker}/USD/1d"
headers = {
 "X-RapidAPI-Key": "YOUR_RAPIDAPI_KEY", # <- сюди встав свій ключ
 "X-RapidAPI-Host": "yahoo-finance15.p.rapidapi.com"
}
all_data = []
for name, ticker in tickers.items():
 url = url_template.format(ticker=ticker)
 response = requests.get(url, headers=headers)
 if response.status_code != 200:
 print(f"Помилка для {ticker}: {response.status_code}")
 continue
 data = response.json()["items"] # список щоденних спостережень
 # Перетворення у DataFrame
 df = pd.DataFrame(data)
 # Дата у datetime
 df["Date"] = pd.to_datetime(df["date"])
 # Фільтруємо по датах
 df = df[(df["Date"] >= pd.Timestamp("2023-01-01")) & (df["Date"] <=
pd.Timestamp(date.today()))]
 # Додаємо стовпці Company та Ticker
 df["Company"] = name
 df["Ticker"] = ticker
 # Денна дохідність
 df = df.sort_values("Date") # обов'язково сортуємо перед pct_change
 df["Daily_Return"] = df["close"].pct_change()
 # Вибір стовпців
 df = df[["Company", "Ticker", "Date", "open", "high", "low", "close", "volume",
"Daily_Return"]]
 all_data.append(df)
Об'єднання всіх компаній
final_df = pd.concat(all_data, ignore_index=True)
final_df = final_df.sort_values(by=["Company", "Date"])
Збереження в Excel
file_path = r"D:\A1\stocks_api_part1.xlsx"
final_df.to_excel(file_path, index=False)
print("Файл з історичними цінами та денними дохідностями через API
збережено:", file_path)

```

Завершенням операцій цього типу є очистка даних, що включає в себе:

1. Перевірити пропуски (NaN) у всіх стовпцях.
2. Видалити або замінити пропуски в Close та Daily\_Return.
3. Переконатися, що **ціни та обсяги**  $\geq 0$ .
4. Стовпець Date — datetime без часової зони.
5. Залишити лише **стовпці**: Company, Ticker, Date, Open, High, Low, Close, Volume, Daily\_Return.
6. Зберегти очищену таблицю в окремий файл (наприклад, з суфіксом \_cleaned).

```
import pandas as pd
from pandas.api.types import is_datetime64tz_dtype
1. Завантажуємо файл
file_path = r"D:\A1\stocks_parsing_part1_with_returns.xlsx"
df = pd.read_excel(file_path)
2. Перевірка типу Date і видалення часової зони
if is_datetime64tz_dtype(df['Date'].dtype):
 df['Date'] = df['Date'].dt.tz_localize(None)
3. Видалення рядків з пропусками у критичних стовпцях
df = df.dropna(subset=["Close", "Daily_Return"])
4. Замінюємо можливі пропуски у інших стовпцях на останнє відоме значення
df["Volume"] = df["Volume"].fillna(0)
df["Open"] = df["Open"].ffill()
df["High"] = df["High"].ffill()
df["Low"] = df["Low"].ffill()
5. Видаляємо некоректні значення (ціни та обсяги < 0)
df = df[(df["Open"] >= 0) & (df["High"] >= 0) & (df["Low"] >= 0) &
 (df["Close"] >= 0) & (df["Volume"] >= 0)]
6. Перевірка стовпців і порядок
df = df[["Company", "Ticker", "Date", "Open", "High", "Low", "Close",
 "Volume", "Daily_Return"]]
7. Сортування блоками
df = df.sort_values(by=["Company", "Date"])
8. Збереження очищеної таблиці у новий файл
clean_file_path = r"D:\A1\stocks_parsing_part1_with_returns_cleaned.xlsx"
df.to_excel(clean_file_path, index=False)
print("Очищена таблиця збережена у файл:", clean_file_path)
```

Очистка даних через API не має сенсу, оскільки це локальна задача, що виконується на вашому комп'ютері. Тому приклад очистки не надається.

### 2.3. Аналіз даних і прогнозування

#### Розрахуємо статистичні характеристики кожної акції

Ми обчислюємо для кожної компанії:

- середню ціну закриття (mean)
- стандартне відхилення (std)
- мінімум/максимум (min/max)
- середню денну дохідність (Daily\_Return)
- всі дані будуть братися з файлу Excel а результати розрахунків – записуватися в інші файли Excel.

```
import pandas as pd
Завантаження очищеної таблиці
file_path = r"D:\A1\stocks_parsing_part1_with_returns_cleaned.xlsx"
df = pd.read_excel(file_path)
Групування по компанії
stats_df = df.groupby("Company").agg({
 "Close": ["mean", "std", "min", "max"],
 "Daily_Return": ["mean", "std", "min", "max"]
}).reset_index()
Об'єднуємо мультиіндекс колонок у прості
stats_df.columns = ['_'.join(col).strip('_') for col in stats_df.columns.values]
Збереження у Excel
stats_file = r"D:\A1\stocks_statistics.xlsx"
stats_df.to_excel(stats_file, index=False)
print("Статистичні характеристики збережено:", stats_file)
```

Результат:

| Company     | Close_mean  | Close_std | Close_min  | Close_max   | Daily_Return_mean | Daily_Return_std | Daily_Return_min | Daily_Return_max |
|-------------|-------------|-----------|------------|-------------|-------------------|------------------|------------------|------------------|
| Amazon.c    | 176,3398191 | 44,330914 | 88,1200017 | 254         | 0,001523825       | 0,020048598      | -0,089791293     | 0,11977          |
| Boeing      | 197,3832472 | 25,94294  | 136,589956 | 264,289989  | 0,000448531       | 0,029900809      | -0,104710454     | 0,153741         |
| Caterpillar | 341,1832331 | 104,72413 | 187,025148 | 702,8900148 | 0,001833853       | 0,018429294      | -0,086158739     | 0,116346         |

Знайдемо матрицю кореляцій акцій (Close). Тут ми аналізуємо як змінюються ціни закриття акцій відносно одна одної.

```
import pandas as pd
Завантаження очищених даних
file_path = r"D:\A1\stocks_parsing_part1_with_returns_cleaned.xlsx"
df = pd.read_excel(file_path)
Переконаємося, що Date — datetime
df["Date"] = pd.to_datetime(df["Date"])
Формуємо таблицю: дати × акції
close_wide = df.pivot(index="Date", columns="Ticker", values="Close")
Кореляційна матриця
corr_matrix = close_wide.corr()
Збереження у Excel
corr_file = r"D:\A1\stocks_correlation_close.xlsx"
corr_matrix.to_excel(corr_file)
print("Матриця кореляцій цін закриття збережена:", corr_file)
```

Результат:

| Ticker | AMZN     | BA       | CAT      | CVX      | DIS      | IBM      | XO       | MCD      | MSFT     | NKE      | NVDA     | WMT      |
|--------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|
| AMZN   | 1        | -0,15168 | 0,840741 | 0,300672 | 0,487913 | 0,908781 | 0,756509 | 0,483873 | 0,930748 | -0,38925 | 0,951844 | 0,910254 |
| BA     | -0,15388 | 1        | 0,047228 | 0,130899 | 0,082947 | -0,05481 | -0,13616 | 0,072375 | -0,03238 | 0,300387 | -0,12488 | -0,09357 |
| CAT    | 0,840741 | 0,042208 | 1        | 0,011303 | 0,867926 | 0,881575 | 0,727839 | 0,649965 | 0,80157  | -0,72802 | 0,8948   | 0,862547 |

Тепер визначимо матрицю коваріації денних дохідності. Тут аналізуємо ризик (варіативність) та взаємозв'язок денних дохідності.

```
import pandas as pd
Завантаження очищених даних
file_path = r"D:\A1\stocks_parsing_part1_with_returns_cleaned.xlsx"
df = pd.read_excel(file_path)
Перетворення дати у datetime
df["Date"] = pd.to_datetime(df["Date"])
Формування таблиці: дати × тикери (денна дохідність)
returns_wide = df.pivot(index="Date", columns="Ticker", values="Daily_Return")
Видалення пропусків (обов'язково для коректної коваріації)
returns_wide = returns_wide.dropna()
Коваріаційна матриця
cov_matrix = returns_wide.cov()
```

```
Збереження у Excel
cov_file = r"D:\A1\stocks_covariance_returns.xlsx"
cov_matrix.to_excel(cov_file)
print("Матриця коваріацій денних дохідностей збережена:", cov_file)
```

Результат:

| Ticker | AMZN     | BA       | CAT      | CVX      | DHS      | HIM       | KO       | MCD      | MSFT     | NKE      | NVDA     | WMT      |
|--------|----------|----------|----------|----------|----------|-----------|----------|----------|----------|----------|----------|----------|
| AMZN   | 0.000402 | 0.000133 | 0.000117 | 8.16E-05 | 0.000118 | 8.21E-05  | -1.3E-05 | 1.33E-05 | 0.000179 | 0.000129 | 0.00029  | 1.44E-05 |
| BA     | 0.000133 | 0.000417 | 0.00011  | 7.25E-05 | 0.000104 | -5.44E-05 | 1.07E-05 | 2.74E-05 | 7.18E-05 | 0.000129 | 0.000147 | 5.94E-05 |
| CAT    | 0.000117 | 0.00011  | 0.000434 | 0.000103 | 0.000111 | 8.19E-05  | -6.1E-06 | 1.99E-05 | 6.47E-05 | 0.000109 | 0.000174 | 3.53E-05 |
| CVX    | 8.66E-05 | 7.25E-05 | 0.000103 | 0.000202 | 7.99E-05 | 9.94E-05  | 1.66E-05 | 2.14E-05 | 1.87E-05 | 6.83E-05 | 1.72E-05 | 2.06E-05 |

Розрахуємо Коефіцієнти  $\alpha$  (альфа) та  $\beta$  (бета) за моделлю CAPM. Ми розрахуємо альфа і бета для кожної акції відносно індексу ринку (для прикладу можна взяти S&P500, тикер ^GSPC).

- $\beta$  показує, наскільки акція реагує на ринок
- $\alpha$  — це додатковий “вироблений” дохід незалежно від ринку

```
import pandas as pd
import yfinance as yf
import statsmodels.api as sm

Завантаження очищених даних
file_path = r"D:\A1\stocks_parsing_part1_with_returns_cleaned.xlsx"
df = pd.read_excel(file_path)

Date → datetime (timezone-naive)
df["Date"] = pd.to_datetime(df["Date"]).dt.tz_localize(None)

Завантаження ринкового індексу (S&P 500)
sp500 = yf.Ticker("^GSPC").history(
 start="2023-01-01",
 end=pd.Timestamp.today()
)

РОЗВ'ЯЗАННЯ ПРОБЛЕМИ TZ
sp500.index = sp500.index.tz_localize(None)

Денна дохідність ринку
```

```

sp500["Daily_Return"] = sp500["Close"].pct_change()
market_returns = sp500["Daily_Return"]
alphas = []
betas = []
tickers = df["Ticker"].unique()
CAPM-регресія
for ticker in tickers:
 stock_returns = (
 df[df["Ticker"] == ticker][["Date", "Daily_Return"]]
 .set_index("Date")
)
 combined = (
 stock_returns
 .join(market_returns.rename("Market_Return"), how="inner")
 .dropna()
)
 # $r_i = \alpha + \beta r_m$
 X = sm.add_constant(combined["Market_Return"])
 y = combined["Daily_Return"]
 model = sm.OLS(y, X).fit()
 alphas.append(model.params["const"])
 betas.append(model.params["Market_Return"])
Результат
capm_df = pd.DataFrame({
 "Ticker": tickers,
 "Alpha": alphas,
 "Beta": betas
})
Збереження

```

```
capm_file = r"D:\A1\stocks_alpha_beta.xlsx"
capm_df.to_excel(capm_file, index=False)
print("Таблиця Alpha/Beta збережена:", capm_file)
```

| Ticker | Alpha    | Beta     |
|--------|----------|----------|
| AMZN   | 0,000337 | 1,463382 |
| BA     | -0,0004  | 1,04534  |
| CAT    | 0,00068  | 1,176684 |
| CVX    | -0,00016 | 0,560711 |
| KO     | 0,000323 | 0,119113 |

Результат:

Далі робимо прогнозування на 3 кроки уперед (3 дні) для кожної акції на основі оригінальної таблиці денних цін закриття. Ми використаємо просту модель ARIMA, яка добре підходить для короткострокових прогнозів фінансових часових рядів.

```
import pandas as pd
from statsmodels.tsa.arima.model import ARIMA
import warnings
warnings.filterwarnings("ignore")
Завантаження оригінальної таблиці
file_path = r"D:\A1\stocks_parsing_part1_with_returns_cleaned.xlsx"
df = pd.read_excel(file_path)
Підготовка для прогнозу
forecast_horizon = 3 # кількість днів уперед
results = []
tickers = df['Ticker'].unique()
for ticker in tickers:
 stock_data = df[df['Ticker']==ticker].sort_values("Date")

 # Використовуємо лише стовпець Close для прогнозу
 ts = stock_data.set_index('Date')['Close']

 # Якщо мало даних, пропускаємо
 if len(ts) < 10:
 continue
 # Простий ARIMA(1,1,1) — можна підібрати оптимальні параметри
 model = ARIMA(ts, order=(1,1,1))
 model_fit = model.fit()
 # Прогноз на 3 дні уперед
```

```

forecast = model_fit.forecast(steps=forecast_horizon)
Дати прогнозу
last_date = ts.index[-1]
forecast_dates = pd.date_range(start=last_date + pd.Timedelta(days=1),
periods=forecast_horizon)
Додаємо результати
for date_val, price in zip(forecast_dates, forecast):
 results.append({
 "Ticker": ticker,
 "Company": stock_data['Company'].iloc[0],
 "Forecast_Date": date_val,
 "Forecast_Close": price
 })
Таблиця прогнозів
forecast_df = pd.DataFrame(results)
Збереження у Excel
forecast_file = r"D:\A1\stocks_forecast_3days.xlsx"
forecast_df.to_excel(forecast_file, index=False)
print("Прогноз на 3 дні збережено:", forecast_file)

```

Результат:

| Ticker | Company     | Forecast_Date       | Forecast_Close |
|--------|-------------|---------------------|----------------|
| AMZN   | Amazon.com  | 2026-02-04 00:00:00 | 238,6713       |
| AMZN   | Amazon.com  | 2026-02-05 00:00:00 | 238,7127       |
| AMZN   | Amazon.com  | 2026-02-06 00:00:00 | 238,7461       |
| BA     | Boeing      | 2026-02-04 00:00:00 | 232,8636       |
| BA     | Boeing      | 2026-02-05 00:00:00 | 232,5798       |
| BA     | Boeing      | 2026-02-06 00:00:00 | 232,2987       |
| CAT    | Caterpillar | 2026-02-04 00:00:00 | 701,2874       |

## 2.4. Оптимізація рішень за допомогою ШІ

Підготуємо дані для оптимізаційних розрахунків складу портфелю цінних паперів.

Нижче – автономний скрипт, який:

- бере очищений файл
- залишає тільки денну дохідність
- формує таблицю виду Дата | Amazon | Boeing | Chevron | ...
- зберігає результат у новий Excel-файл.

```

import pandas as pd

Завантаження очищених даних

input_file = r"D:\A1\stocks_parsing_part1_with_returns_cleaned.xlsx"

df = pd.read_excel(input_file)

Формат дати
df["Date"] = pd.to_datetime(df["Date"])

Формування матриці дохідностей
Рядки — дати, колонки — компанії
returns_matrix = df.pivot(
 index="Date",
 columns="Company", # ← якщо хочеш тикери, заміни на "Ticker"
 values="Daily_Return"
)

Сортування за датою
returns_matrix = returns_matrix.sort_index()

Збереження у новий Excel-файл
output_file = r"D:\A1\stocks_daily_returns_matrix.xlsx"
returns_matrix.to_excel(output_file)

print("Файл з матрицею денних дохідностей створено:")
print(output_file)

```

Результат:

| Date                | Amazon   | Boeing   | Caterpillar | Charter  | Coca-Cola | IBM      | McDonald | Microsoft | NVIDIA   | Nike     | Walmart  | Wall Street |
|---------------------|----------|----------|-------------|----------|-----------|----------|----------|-----------|----------|----------|----------|-------------|
| 2023-01-04 00:00:00 | -0.00792 | 0.042224 | 0.010382    | -0.00761 | -0.00048  | 0.005418 | 0.000227 | -0.04374  | 0.000118 | 0.000714 | 0.001134 | 0.010382    |
| 2023-01-05 00:00:00 | -0.02173 | 0.006629 | -0.00447    | 0.018000 | -0.01144  | -0.01049 | -0.00843 | -0.02964  | -0.01282 | -0.00437 | 0.00341  | -0.00065    |
| 2023-01-06 00:00:00 | 0.018021 | 0.039079 | 0.023708    | 0.007533 | 0.019253  | 0.018334 | 0.027884 | 0.011703  | 0.04104  | 0.012014 | 0.004099 | 0.021758    |
| 2023-01-09 00:00:00 | 0.01487  | -0.0203  | -0.009      | -0.00781 | -0.01246  | -0.00104 | -0.00804 | 0.009736  | 0.051731 | 0.00237  | -0.01347 | 0.00908     |

Тепер розробимо 2 окремих незалежних програми, які знаходять оптимальний склад портфеля цінних паперів за Марковцем і за Шарпом за

критерієм максимізації доходності. Ризик ( $\sigma$ ) – задаємо як середнє стандартне відхилення доходності по вибірці.

#### Оптимальний портфель за Марковіцем

Максимізуємо середню очікувану дохідність портфеля при заданому ризику ( $\sigma$  = середнє стандартне відхилення по акціях). Якщо спочатку за заданим рівнем ризику портфель не знайдено, програма пропонує збільшити допустимий рівень ризику.

#### Алгоритм оптимізації портфеля за Марковіцем (Mean-Variance)

Мета: максимізувати очікувану дохідність портфеля при заданому рівні ризику ( $\sigma$ ).

##### 1. Вхідні дані:

- середні доходності акцій  $\mu_i$
- коваріаційна матриця  $\Sigma$

##### 2. Випадкове або чисельне формування портфелів:

- генеруємо набори ваг  $w_i$ , що задовольняють  $\sum w_i = 1$  і  $w_i \geq 0$

##### 3. Обчислення характеристик портфеля:

- доходність:  $R_p = \sum_i w_i \mu_i$
- ризик:  $\sigma_p = \sqrt{w^T \Sigma w}$

##### 4. Відбір портфелів з потрібним ризиком (у нашому випадку $\pm$ tolerance)

##### 5. Вибір портфеля з максимальною доходністю

##### 6. Результат: оптимальні ваги, доходність і ризик портфеля

В класичному підході Марковіца можна також розв'язувати аналітично через квадратичне програмування, але ми робили чисельний пошук через SLSQP або перебір.

```
import pandas as pd
import numpy as np
Завантаження матриці доходностей
file_path = r"D:\A1\stocks_daily_returns_matrix.xlsx"
returns = pd.read_excel(file_path, index_col=0)
expected_returns = returns.mean()
cov_matrix = returns.cov()
tickers = returns.columns
```

```

n_assets = len(tickers)
n_portfolios = 50000
tolerance = 0.01 # ±1% від цільового ризику
Початковий коефіцієнт ризику
risk_factor = 1.0
while True:
 risk_target = returns.std().mean() * risk_factor
 results = []
 # Генерація випадкових портфелів
 for _ in range(n_portfolios):
 weights = np.random.random(n_assets)
 weights /= np.sum(weights)
 port_return = np.dot(weights, expected_returns)
 port_volatility = np.sqrt(np.dot(weights.T, np.dot(cov_matrix, weights)))
 if abs(port_volatility - risk_target) < tolerance:
 results.append((weights, port_return, port_volatility))
 # Перевірка
 if results:
 best_portfolio = max(results, key=lambda x: x[1])
 best_weights, best_return, best_volatility = best_portfolio
 portfolio_df = pd.DataFrame({
 "Ticker": tickers,
 "Weight": best_weights
 })
 portfolio_df["Expected_Return"] = best_return
 portfolio_df["Volatility"] = best_volatility

 output_file = r"D:\A1\optimal_portfolio_markowitz.xlsx"
 portfolio_df.to_excel(output_file, index=False)
 print("Оптимальний портфель за Марковіцем збережено:", output_file)
 break
 else:
 print(f"Не знайдено портфеля з ризиком, рівним {risk_target:.6f}.")
 # запит на введення нового коефіцієнта
 risk_factor = float(input("Введіть коефіцієнт збільшення ризику
(наприклад, 1.5): "))

```

Алгоритм оптимізації портфеля за Шарпом (Maximum Sharpe Ratio)

Мета: максимізувати коефіцієнт Шарпа, тобто співвідношення доходності

до ризику:

$$S = \frac{R_p - R_f}{\sigma_p}$$

де  $R_f$  – безризикова ставка.

Кроки:

1. Вхідні дані:  $\mu_i, \Sigma, R_f$
2. Генерація можливих портфелів:
  - ваги  $w_i$ , які задовольняють  $\sum w_i = 1$  і  $w_i \geq 0$
3. Обчислення характеристик:
  - доходність  $R_p = \sum_i w_i \mu_i$
  - ризик  $\sigma_p = \sqrt{w^T \Sigma w}$
  - Шарп:  $S = (R_p - R_f) / \sigma_p$
4. Відбір портфелів з потрібним ризиком (опціонально)
5. Вибір портфеля з максимальним Шарпом
6. Результат: оптимальні ваги, доходність, ризик, Sharpe Ratio

```
import pandas as pd
import numpy as np
Завантаження матриці доходностей
file_path = r"D:\A1\stocks_daily_returns_matrix.xlsx"
returns = pd.read_excel(file_path, index_col=0)
expected_returns = returns.mean()
cov_matrix = returns.cov()
tickers = returns.columns
n_assets = len(tickers)
n_portfolios = 50000
tolerance = 0.01
rf = 0 # безризикова ставка для простоти
Початковий коефіцієнт ризику
risk_factor = 1.0
while True:
 risk_target = returns.std().mean() * risk_factor
 results = []
 # Генерація випадкових портфелів
 for _ in range(n_portfolios):
 weights = np.random.random(n_assets)
 weights /= np.sum(weights)
 port_return = np.dot(weights, expected_returns)
 port_volatility = np.sqrt(np.dot(weights.T, np.dot(cov_matrix, weights)))
 sharpe_ratio = (port_return - rf) / port_volatility
 if abs(port_volatility - risk_target) < tolerance:
```

```

 results.append((weights, port_return, port_volatility, sharpe_ratio))
Перевірка
if results:
 best_portfolio = max(results, key=lambda x: x[3]) # максимальний Sharpe
 best_weights, best_return, best_volatility, best_sharpe = best_portfolio
 portfolio_df = pd.DataFrame({
 "Ticker": tickers,
 "Weight": best_weights
 })
 portfolio_df["Expected_Return"] = best_return
 portfolio_df["Volatility"] = best_volatility
 portfolio_df["Sharpe_Ratio"] = best_sharpe
 output_file = r"D:\A1\optimal_portfolio_sharpe.xlsx"
 portfolio_df.to_excel(output_file, index=False)
 print("Оптимальний портфель за Шарпом збережено:", output_file)
 break
else:
 print(f"Не знайдено портфеля з ризиком, рівним {risk_target:.6f}.")
 risk_factor = float(input("Введіть коефіцієнт збільшення ризику
(наприклад, 1.5): "))

```

Результат за Марковіцем

| Ticker      | Weight   | Expected_Return | Volatility |
|-------------|----------|-----------------|------------|
| Amazon.c    | 0,154856 | 0,001919        | 0,014715   |
| Boeing      | 0,050107 | 0,001919        | 0,014715   |
| Caterpilla  | 0,15842  | 0,001919        | 0,014715   |
| Chevron     | 0,061695 | 0,001919        | 0,014715   |
| Coca-Cola   | 0,064181 | 0,001919        | 0,014715   |
| IBM         | 0,005137 | 0,001919        | 0,014715   |
| McDonald    | 0,005043 | 0,001919        | 0,014715   |
| Microsoft   | 0,012129 | 0,001919        | 0,014715   |
| NVIDIA      | 0,331791 | 0,001919        | 0,014715   |
| Nike        | 0,064203 | 0,001919        | 0,014715   |
| Walmart     | 0,08766  | 0,001919        | 0,014715   |
| Walt Disney | 0,004777 | 0,001919        | 0,014715   |

за Шарпом

| Ticker      | Weight   | Expected_Return | Volatility | Sharpe_Ratio |
|-------------|----------|-----------------|------------|--------------|
| Amazon.c    | 0,095549 | 0,001427        | 0,009133   | 0,156249     |
| Boeing      | 0,002572 | 0,001427        | 0,009133   | 0,156249     |
| Caterpilla  | 0,100278 | 0,001427        | 0,009133   | 0,156249     |
| Chevron     | 0,031209 | 0,001427        | 0,009133   | 0,156249     |
| Coca-Cola   | 0,139134 | 0,001427        | 0,009133   | 0,156249     |
| IBM         | 0,04889  | 0,001427        | 0,009133   | 0,156249     |
| McDonald    | 0,13871  | 0,001427        | 0,009133   | 0,156249     |
| Microsoft   | 0,047516 | 0,001427        | 0,009133   | 0,156249     |
| NVIDIA      | 0,149776 | 0,001427        | 0,009133   | 0,156249     |
| Nike        | 0,004315 | 0,001427        | 0,009133   | 0,156249     |
| Walmart     | 0,237663 | 0,001427        | 0,009133   | 0,156249     |
| Walt Disney | 0,001391 | 0,001427        | 0,009133   | 0,156249     |

Попередні задачі вирішувалися повним перебором із 50000 варіантів, але при необхідності можна вирішити нелінійну оптимізаційну задачу:

- Цільова функція із логарифмом ризику
- Обмеження: сума ваг = 1 та квадрат ризику  $\leq$  заданий поріг

---

Математична постановка

$$\text{maximize } f(w) = \sum_i w_i \mu_i - \lambda \cdot \log \left( 1 + \sum_{i,j} w_i w_j \sigma_{ij} \right)$$

Обмеження:

1.  $\sum_i w_i = 1$
2.  $\sum_{i,j} w_i w_j \sigma_{ij}^2 \leq \text{risk\_limit}$
3.  $0 \leq w_i \leq 1$

Тобто цільова функція нелінійна через логарифм, а обмеження квадратичне.

```
import pandas as pd
import numpy as np
from scipy.optimize import minimize
Завантаження матриці доходностей
file_path = r"D:\A1\stocks_daily_returns_matrix.xlsx"
returns = pd.read_excel(file_path, index_col=0)
expected_returns = returns.mean().values
cov_matrix = returns.cov().values
n_assets = len(expected_returns)
Параметр штрафу
lambda_penalty = 1.0
Поріг ризику (квадратна функція)
risk_limit = (returns.std().mean() * 3)**2
4 Цільова функція (максимізація логарифмічного штрафу ризику)
def objective(weights):
 port_return = np.dot(weights, expected_returns)
 port_variance = np.dot(weights.T, np.dot(cov_matrix, weights))
 return -(port_return - lambda_penalty * np.log(1 + port_variance))
Обмеження
constraints = [
 {'type': 'eq', 'fun': lambda w: np.sum(w) - 1}, # сума ваг = 1
 {'type': 'ineq', 'fun': lambda w: risk_limit - np.dot(w.T, np.dot(cov_matrix, w))**2}
]
квадрат ризику ≤ risk_limit
]
Межі для ваг
bounds = tuple((0, 1) for _ in range(n_assets))
Початкове наближення
x0 = np.ones(n_assets) / n_assets
Оптимізація
```

```

result = minimize(objective, x0, method='SLSQP', bounds=bounds,
constraints=constraints)
if result.success:
 optimal_weights = result.x
 port_return = np.dot(optimal_weights, expected_returns)
 port_variance = np.dot(optimal_weights.T, np.dot(cov_matrix, optimal_weights))
 portfolio_df = pd.DataFrame({
 'Ticker': returns.columns,
 'Weight': optimal_weights
 })
 portfolio_df['Expected_Return'] = port_return
 portfolio_df['Portfolio_Variance'] = port_variance
 output_file = r"D:\A1\nonlinear_log_square_portfolio.xlsx"
 portfolio_df.to_excel(output_file, index=False)
 print("Оптимальний нелінійний портфель з логарифмом та квадратом
обмеження збережено:", output_file)
else:
 print("Оптимізація не вдалася:", result.message)

```

Результат:

| Ticker      | Weight   | Expected_Return | Portfolio_Variance |
|-------------|----------|-----------------|--------------------|
| Amazon.c    | 0        | 0,003707        | 0,000937           |
| Boeing      | 0        | 0,003707        | 0,000937           |
| Caterpillar | 0,028045 | 0,003707        | 0,000937           |
| Chevron     | 0        | 0,003707        | 0,000937           |
| Coca-Cola   | 8,93E-18 | 0,003707        | 0,000937           |
| IBM         | 1,1E-17  | 0,003707        | 0,000937           |
| McDonald    | 0        | 0,003707        | 0,000937           |
| Microsoft   | 2,89E-17 | 0,003707        | 0,000937           |
| NVIDIA      | 0,971955 | 0,003707        | 0,000937           |
| Nike        | 2,5E-17  | 0,003707        | 0,000937           |
| Walmart     | 0        | 0,003707        | 0,000937           |
| Walt Disney | 0        | 0,003707        | 0,000937           |

Ось відповідність рядків коду до формул і їх фінансового змісту:

## 1. Цільова функція

Формула:

$$f(w) = \sum_{i=1}^n w_i \mu_i - \lambda \cdot \log(1 + w^T \Sigma w)$$

### Код:

```
def objective(weights):
 port_return = np.dot(weights, expected_returns) # $\sum w_i \mu_i$
 port_variance = np.dot(weights.T, np.dot(cov_matrix, weights)) # $w^T \Sigma w$
 return -(port_return - lambda_penalty * np.log(1 + port_variance))
```

### Пояснення:

- $\text{port\_return} = \sum w_i \mu_i \rightarrow$  очікувана доходність портфеля
- $\text{port\_variance} = w^T \Sigma w \rightarrow$  дисперсія портфеля (ризик)
- $\text{np.log}(1 + \text{port\_variance}) \rightarrow$  нелінійний штраф за ризик
- $\text{lambda\_penalty} \rightarrow$  коефіцієнт важливості ризику у функції
- Знак -  $\rightarrow$  тому що minimize шукає мінімум, а нам потрібен максимум

## 2. Обмеження

1. Сума всіх ваг = 1

$$\sum_i w_i = 1$$

**{'type': 'eq', 'fun': lambda w: np.sum(w) - 1}**

2. Квадрат ризику  $\leq$  допустимий поріг

$$(w^T \Sigma w)^2 \leq \text{risk\_limit}$$

**{'type': 'ineq', 'fun': lambda w: risk\_limit - (np.dot(w.T, np.dot(cov\_matrix, w)))\*\*2}**

### Пояснення:

- $\text{np.dot}(w.T, \text{np.dot}(\text{cov\_matrix}, w)) \rightarrow$  стандартна формула для дисперсії портфеля
- Піднесення у квадрат  $\rightarrow$  обмеження стає квадратичним
- $\text{risk\_limit} - (\dots)^2 \geq 0 \rightarrow$  нерівність для `scipy.optimize`

## 3. Межі ваги

$$0 \leq w_i \leq 1$$

```
bounds = tuple((0, 1) for _ in range(n_assets))
```

**Пояснення:**

- Неможливо продавати коротко (short selling), всі ваги додатні

#### 4. Виклик оптимізатора

```
result = minimize(objective, x0, method='SLSQP', bounds=bounds, constraints=constraints)
```

**Пояснення:**

- Метод **SLSQP** → обробляє **обмеження рівності та нерівності**
- objective → цільова функція з логарифмом
- constraints → сума ваг + квадрат ризику

#### 5. Результат

```
optimal_weights = result.x
```

```
port_return = np.dot(optimal_weights, expected_returns)
```

```
port_variance = np.dot(optimal_weights.T, np.dot(cov_matrix, optimal_weights))
```

- optimal\_weights →  $w^*$  (оптимальні ваги)
- port\_return → максимальна доходність з урахуванням логарифмічного штрафу
- port\_variance → ризик портфеля

Таким чином, **всі формули** чітко закодовані:

**Фінансова формула**

**Код**

$$R_p = \sum_i w_i \mu_i$$

```
port_return = np.dot(weights, expected_returns)
```

$$\sigma_p^2 = w^T \Sigma w$$

```
port_variance = np.dot(weights.T, np.dot(cov_matrix, weights))
```

**Фінансова формула****Код**

Нелінійний штраф за ризик  $\log(1 + \sigma_p^2)$

`np.log(1 + port_variance)`

Сума ваг = 1

`{'type': 'eq', 'fun': lambda w: np.sum(w)-1}`

Квадрат ризику  $\leq$  поріг

`{'type': 'ineq', 'fun': lambda w: risk_limit - (np.dot(w.T, np.dot(cov_matrix, w)))*2}`

**2.5. Основи теорії ігор та стратегій у ШІ**

Хороша робота! Тепер перейдемо до теорії ігор. У файлі `theory_of_gaim.xlsx` міститься таблиця, в якій перші 3 рядки визначають тип стратегії. 4 стовпці визначають стан природи. Четвертий рядок дає значення імовірності настання стану природи. Для коректної роботи програми треба прибрати назви стратегій (A1...A3) і назви станів природи (S1...S4). Також додано рядок імовірності ( $p$ ) настання станів природи. У робочому варіанті файлу (праворуч) додано точність ( $q$ ) визначення імовірності.

|         | S1        | S2        | S3        | S4        |           |           |           |           |
|---------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|
|         |           |           |           |           | 3,599E+11 | 3,599E+11 | 5,399E+11 | 5,399E+11 |
| A1      | 3,599E+11 | 3,599E+11 | 5,399E+11 | 5,399E+11 | 3,598E+11 | 3,597E+11 | 5,398E+11 | 5,397E+11 |
| A2      | 3,598E+11 | 3,597E+11 | 5,398E+11 | 5,397E+11 | 3,598E+11 | 3,597E+11 | 5,398E+11 | 5,397E+11 |
| A3      | 3,598E+11 | 3,597E+11 | 5,398E+11 | 5,397E+11 | 0,28      | 0,12      | 0,42      | 0,18      |
| robably | 0,28      | 0,12      | 0,42      | 0,18      | 0,85      | 0,77      | 0,91      | 0,44      |

ЯК використовується точність:

Коригуємо імовірності:  $p_i^* = p_i \cdot q_i$

Нормалізуємо:  $\tilde{p}_i = \frac{p_i^*}{\sum p_i^*}$

Саме ці  $\tilde{p}_i$  використовуємо для очікуваної вигоди

Створимо код, який визначить найкращу стратегію гри з природою у двох варіантах : 1. Коли імовірності невідомі 2. Коли імовірності відомі. За першим варіантом треба застосувати не менше 4-х методів знайдення найкращої стратегії, а потім вибрати найкращий за більшістю методів.

```
import pandas as pd
import numpy as np
from collections import Counter

Ввід r і s

while True:
 try:
 r = int(input("Введіть кількість стратегій r: "))
 s = int(input("Введіть кількість станів природи s: "))
 if r > 0 and s > 0:
 break
 else:
 print("r та s повинні бути додатні")
 except ValueError:
 print("Введіть цілі числа")

Зчитування Excel

file_path = r"D:\A1\theory_of_gaim.xlsx"
df = pd.read_excel(file_path, header=None)

Матриця виграшів

payoffs = df.iloc[:, :s].astype(float).values
payoffs = np.nan_to_num(payoffs, nan=0.0)

Ймовірності (рядок r)

probabilities = None
if df.shape[0] >= r + 1:
 probabilities = df.iloc[r, :s].astype(float).values
 probabilities = np.nan_to_num(probabilities, nan=0.0)
```

```

Точності (рядок r+1)

accuracy = None
if df.shape[0] >= r + 2:
 accuracy = df.iloc[r + 1, :s].astype(float).values
 accuracy = np.nan_to_num(accuracy, nan=0.0)

Ввід α

while True:
 try:
 alpha = float(input("Введіть коефіцієнт оптимізму α ($0 \leq \alpha \leq 1$): "))
 if 0 <= alpha <= 1:
 break
 else:
 print("α має бути між 0 та 1")
 except ValueError:
 print("Введіть число від 0 до 1")

Розрахунок по стратегіях

strategies = [f"A{i+1}" for i in range(r)]

min_vals = np.min(payoffs, axis=1)
max_vals = np.max(payoffs, axis=1)
avg_vals = np.mean(payoffs, axis=1)
hurwicz_vals = alpha * max_vals + (1 - alpha) * min_vals

Очікувана вигода (якщо є ймовірності)

expected_values = None
if probabilities is not None and len(probabilities) == s:

 # якщо точність ϵ — застосовуємо
 if accuracy is not None and len(accuracy) == s:
 probabilities = probabilities * accuracy

 # без нормалізації
 expected_values = payoffs.dot(probabilities)

```

```

Таблиця "результати по стратегіях"

results_by_strategy = pd.DataFrame({
 "Стратегія": strategies,
 "min": min_vals,
 "max": max_vals,
 "avg": avg_vals,
 "hurwicz": hurwicz_vals
})

if expected_values is not None:
 results_by_strategy["expected"] = expected_values

print("\n ✎ Результати по кожній стратегії:")
print(results_by_strategy)

Вибір кращих стратегій по методах

maximin_index = np.argmax(min_vals)
maximax_index = np.argmax(max_vals)
laplace_index = np.argmax(avg_vals)
hurwicz_index = np.argmax(hurwicz_vals)

maximin_strategy = strategies[maximin_index]
maximax_strategy = strategies[maximax_index]
laplace_strategy = strategies[laplace_index]
hurwicz_strategy = strategies[hurwicz_index]

votes = [maximin_strategy, maximax_strategy, laplace_strategy, hurwicz_strategy]
majority_strategy = Counter(votes).most_common(1)[0][0]

methods = [
 "Максимін",
 "Максимакс",
 "Лаплас",
 f"Гурвіца ($\alpha = \{alpha\}$)",
 "Більшістю методів"
]

strategies_result = [
 maximin_strategy,

```

```

 maximax_strategy,
 laplace_strategy,
 hurwicz_strategy,
 majority_strategy
]

sum_payoff_for_method = [
 float(min_vals[maximin_index]),
 float(max_vals[maximax_index]),
 float(avg_vals[laplace_index]),
 float(hurwicz_vals[hurwicz_index]),
 float(avg_vals[strategies.index(majority_strategy)])
]

Додаємо "Очікувана вигода"
if expected_values is not None:
 methods.append("Очікувана вигода")
 strategies_result.append(strategies[np.argmax(expected_values)])
 sum_payoff_for_method.append(float(expected_values.max()))

results_df = pd.DataFrame({
 "Метод": methods,
 "Стратегія": strategies_result,
 "Сума виграшу": sum_payoff_for_method
})

Збереження у файл

output_file = r"D:\A1\best_strategy_theory_of_game.xlsx"
results_df.to_excel(output_file, index=False)

print("\n☑ Результати вибору стратегій:")
print(results_df)
print(f"\n📁 Файл збережено: {output_file}")

```

### Пояснення коду

1. `payoffs` → матриця виграшів стратегій (рядки) для кожного стану природи (стовпці)
2. `probabilities` → ймовірності настання станів природи
3. Методи для невідомих ймовірностей:

- Максимін → максимізує мінімальний виграш
  - Максимакс → максимізує максимальний виграш
  - Лаплас → середнє виграшів для всіх станів
  - Гурвіца → компроміс між максимін та максимакс
4. Метод більшості → обираємо стратегію, яку обрали більшість методів
  5. Відомі ймовірності → обчислюємо очікувану вигоду для кожної стратегії та вибираємо максимальну.
  6. У програмі передбачено автоматичне перетворення десяткових ком у крапки для коректного зчитування числових даних з Excel.
  7. Програма працює з матрицею довільного розміру, а також враховує випадки, коли точності і/або імовірності невідомі.
  8. Програма ігнорує коми в десяткових числах, що стоять у файлі.

Результати роботи програми подано для трьох варіантів:

1. Відомі імовірності і їх точності
2. Відомі тільки імовірності
3. Імовірності невідомі.

| Метод | Стратегія         | Сума виграшу     |
|-------|-------------------|------------------|
| 0     | Максимін          | AI: 3.599247e+11 |
| 1     | Максимакс         | AI: 5.399432e+11 |
| 2     | Лаплас            | AI: 4.499337e+11 |
| 3     | Гурвіца (α = 0.6) | AI: 4.679356e+11 |
| 4     | Більшість методів | AI: 4.499337e+11 |
| 5     | Очікувана вигода  | AI: 3.299337e+11 |

| Метод | Стратегія         | Сума виграшу     |
|-------|-------------------|------------------|
| 0     | Максимін          | AI: 3.599247e+11 |
| 1     | Максимакс         | AI: 5.399432e+11 |
| 2     | Лаплас            | AI: 4.499337e+11 |
| 3     | Гурвіца (α = 0.6) | AI: 4.679356e+11 |
| 4     | Більшість методів | AI: 4.499337e+11 |
| 5     | Очікувана вигода  | AI: 4.379337e+11 |

| Метод | Стратегія         | Сума виграшу     |
|-------|-------------------|------------------|
| 0     | Максимін          | AI: 3.599247e+11 |
| 1     | Максимакс         | AI: 5.399432e+11 |
| 2     | Лаплас            | AI: 4.499337e+11 |
| 3     | Гурвіца (α = 0.6) | AI: 4.679356e+11 |
| 4     | Більшість методів | AI: 4.499337e+11 |

Як видно з результатів, числові значення очікуваної вигоди змінилися, а в останньому випадку, цей розрахунок зник, бо імовірності не визначені.

## 2.6. Використання Windows Task Sheduler для регулярного виконання програм

Якщо необхідно локально запускати програми, наприклад, парсинг акцій, збереження у Excel, щоденний запуск о 00:00, тоді ми використовуємо **Windows Task Sheduler**.

Для цього необхідно внести зміни у **pursingакс.py** – програму, що регулярно виконує парсинг даних щодо вартості акцій.

```
import pandas as pd

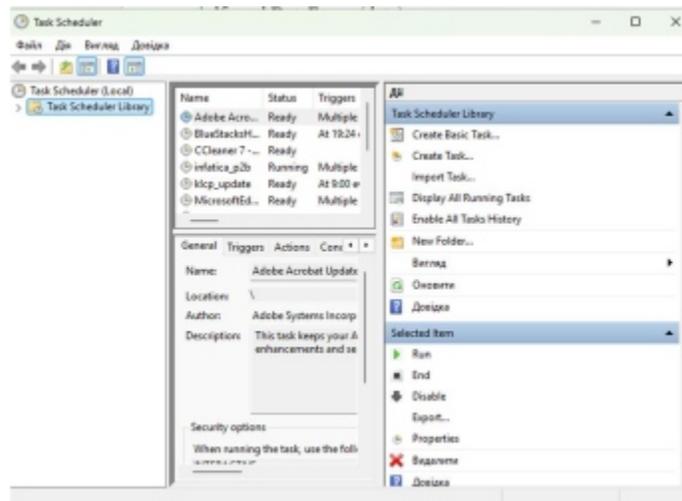
--- твій код парсингу акцій ---
Наприклад:
data = {"Ticker": ["AAPL", "MSFT"], "Price": [150, 320]}
df = pd.DataFrame(data)

Збереження Excel у тій же папці та з одним і тим самим іменем
output_path = "output.xlsx" # старий файл буде перезаписано
df.to_excel(output_path, index=False)

print(f"Файл Excel збережено: {output_path} (старий файл перезаписано)")
```

- **Важливо:** скрипт зберігає Excel локально, у директорії, де лежить **pursingакс.py**.

### Створення щоденного запуску у Windows Task Scheduler



1. Відкрій **Task Scheduler** (Пуск → Task Scheduler)

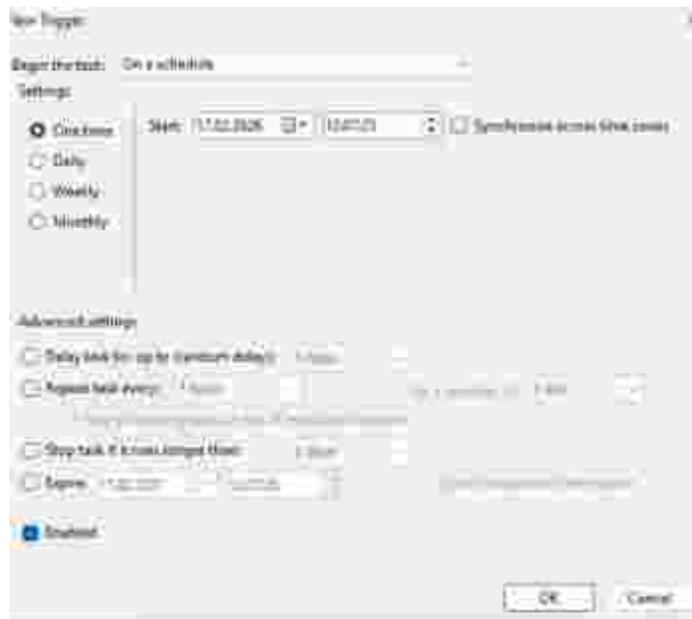
## 2. Натисни **Create Task**

### 3. Вкладка **General**:

- Name: DailyStockParser
- Location: пропускаємо.
- Run whether user is logged on or not
- Run with highest privileges

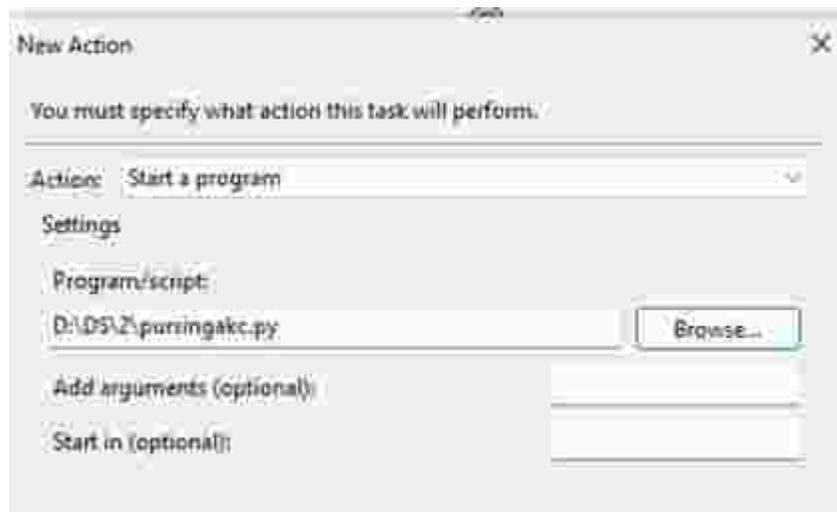
### 4. Вкладка **Triggers**:

- New → Begin the task: **On a schedule**
- Daily → Start: 00:00
- Repeat every: 1 day



### 5. Вкладка **Actions**:

- New → Action: **Start a program**
- Program/script: шлях до Python, наприклад:
- C:\Users\Ihor\AppData\Local\Programs\Python\Python311\python.exe
- Add arguments: шлях до твого скрипта
- "C:\path\to\DS\2\pursingакс.py"
- Start in: директорія скрипта
- C:\path\to\DS\2



6. Вкладки **Conditions** і **Settings** залиш стандартні або налаштуй під себе.
7. Натисни **ОК** та збережи завдання.

### Перевірка роботи

1. У Task Scheduler → правою кнопкою по завданню → **Run**
2. Перевір, що створився файл Excel у директорії скрипта, наприклад:  
output\_2026-02-17.xlsx
3. Наступного дня о 00:00 Windows автоматично запустить скрипт і збереже новий Excel.

### Поради

- Якщо скрипт використовує додаткові бібліотеки (pandas, requests, beautifulsoup4, openpyxl), переконатейся, що Python, який вказаний у Task Scheduler, має всі пакети. Це можна виконати командою у терміналі:

**pip install pandas requests beautifulsoup4 openpyxl lxml**

## 2.7. Індивідуальне завдання №2

### Збір даних через Python та ШІ

**Критерії оцінювання:** це завдання оцінюється у 5 балів за національною шкалою за декількома завданнями окремо. За кожну помилку знімається 0,1 бали. Потім оцінка перераховується за 100-бальною системою згідно існуючого положення.

**Мета роботи:** Набути навичок у використанні API для різних фондових бірж.

**Завдання:** 1. Обрати біржу та країну згідно наступної таблиці 2.1.

Таблиця 2.1

Країна та тип біржі

| № за списком групи | Країна                                 | Біржа / Платформа                             | Коментар щодо API доступу                                                      |
|--------------------|----------------------------------------|-----------------------------------------------|--------------------------------------------------------------------------------|
| 1                  | США                                    | <b>NASDAQ</b>                                 | Має офіційні API для ринкових даних (Nasdaq Data Link) та торговельних послуг. |
| 2                  | США                                    | <b>NYSE</b>                                   | Доступ до даних через Nasdaq Data Link та сторонні API-провайдери.             |
| 3                  | США                                    | <b>IEX (Investors Exchange)</b>               | Дані доступні через IEX Cloud API (реальні й історичні).                       |
| 4                  | США                                    | <b>FINRA / OTC ринки</b>                      | Через API-провайдерів, які агрегують дані з альтернативних ринків.             |
| 5                  | Велика Британія                        | <b>London Stock Exchange (LSE)</b>            | Доступ через API-сервіси даних (Marketstack, Finnhub, ін.).                    |
| 6                  | ЄС (Нідерланди, Франція, Бельгія тощо) | <b>Euronext</b>                               | Дані через API-провайдерів (Marketstack, Leeway тощо).                         |
| 7                  | Німеччина                              | <b>Frankfurt Stock Exchange (FRA / XETRA)</b> | Підтримується API-доступом через сторонні сервіси.                             |
| 8                  | Японія                                 | <b>Tokyo Stock Exchange (TSE)</b>             | Поширене у API-провайдерів (Marketstack, Finnhub).                             |
| 9                  | Індія                                  | <b>Bombay Stock Exchange (BSE)</b>            | Підтримується низкою API-платформ (Leeway, Twelve Data).                       |
| 10                 | Китай                                  | <b>Shanghai Stock Exchange (SSE)</b>          | API-доступ через універсальні API-платформи.                                   |

| № за списком групи | Країна    | Біржа / Платформа                           | Коментар щодо API доступу                     |
|--------------------|-----------|---------------------------------------------|-----------------------------------------------|
| 11                 | Канада    | <b>Toronto Stock Exchange (TSX)</b>         | API-дані через сторонні сервіси.              |
| 12                 | Австралія | <b>Australian Securities Exchange (ASX)</b> | Підтримка через глобальні API.                |
| 13                 | Сінгапур  | <b>Singapore Exchange (SGX)</b>             | API-підтримка через деяких провайдерів даних. |

2. Зайти на біржу та вибрати 8-10 типів акцій.
3. Скачати дані про перебір торгів цими акціями за період 2-3 роки до поточної дати виконання роботи.
4. Розмістити ці дані у файлі Excel.
5. Розрахувати поточну дохідність цих акцій та викласти ці дані в інший файл Excel.

**Завдання 2:** 1. За даними файлу з дохідністю вирішити задачу Марковіца і Шарпа, щоб визначити оптимальний склад портфеля цінних паперів.

2. Результати розрахунку розмістити в окремому файлі Excel.

**Завдання 3:** 1. За даними таблиці, що містить змінив часі вартості акцій, створити таблицю, в якій будуть перші 5 рядків цієї таблиці.

2. Використати цю нову таблицю для вирішення задачі з гри з природою в умовах повної невизначеності.

3. Результати вибору найкращої стратегії розмістити в окремому файлі.

Довідково: у табл. 2.2 знаходяться можливості різних API для доступу на сайти фондових бірж, а також наведені методи доступу до них.

## API та біржі

| Назва API                    | Біржі (приклади)                                               | Приклад звертання до 1 акції                                                                                                                                                                                                   | Як звертатися до декількох акцій                                                                                       |
|------------------------------|----------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------|
| <b>Global Market API</b>     | NASDAQ, NYSE, TSX, BSE, NSE, HKEX, JPX, LSE, Xetra, ASX та ін. | GET<br><a href="https://api.globalmarketapi.com/v1/quote?symbol=AAPL.NASDAQ">https://api.globalmarketapi.com/v1/quote?symbol=AAPL.NASDAQ</a>                                                                                   | Через список символів:<br><code>?symbols=AAPL.NASDAQ,TSLA.NASDAQ,7203.T</code>                                         |
| <b>Markets tack</b>          | NYSE, NASDAQ, LSE, Xetra, Euronext, ASX, HKEX, TSX та ін.      | GET<br><a href="https://api.marketstack.com/v2/eod?access_key=YOUR_KEY&amp;symbols=AAPL">https://api.marketstack.com/v2/eod?access_key=YOUR_KEY&amp;symbols=AAPL</a>                                                           | <code>&amp;symbols=AAPL,TSLA,7203.T</code>                                                                             |
| <b>FCS API</b>               | NYSE, NASDAQ, JPX, NSE, BSE, HKEX, LSE та ін.                  | GET <a href="https://api-v4.fcsapi.com/stock/latest?symbol=7203.T&amp;access_key=YOUR_KEY">https://api-v4.fcsapi.com/stock/latest?symbol=7203.T&amp;access_key=YOUR_KEY</a>                                                    | <b>По черзі:</b> <code>symbol=...</code> (FCS API не підтримує список у одному запиті, тому робиться цикл по символах) |
| <b>EODData API</b>           | NASDAQ, NYSE, LSE, ASX, Xetra, SSE та ін.                      | GET<br><a href="https://eodhistoricaldata.com/api/eod/AAPL.US?api_token=YOUR_KEY">https://eodhistoricaldata.com/api/eod/AAPL.US?api_token=YOUR_KEY</a>                                                                         | По декількох символах:<br><code>.../eod/AAPL.US,TSLA.US?api_token=...</code>                                           |
| <b>Alpha Vantage</b>         | США, Європа, Азія (через символи типу: VOD.LON)                | GET<br><a href="https://www.alphavantage.com/query?function=TIME_SERIES_DAILY&amp;symbol=VOD.LON&amp;apikey=YOUR_KEY">https://www.alphavantage.com/query?function=TIME_SERIES_DAILY&amp;symbol=VOD.LON&amp;apikey=YOUR_KEY</a> | <b>Не підтримує список</b> — робиться цикл по символах                                                                 |
| <b>Trading Economics API</b> | Інтегрує дані багатьох бірж і країн (акції, індекси)           | GET<br><a href="https://api.tradingeconomics.com/markets/stock?symbol=VOD:LN">https://api.tradingeconomics.com/markets/stock?symbol=VOD:LN</a>                                                                                 | Можна робити кілька символів через <code>symbol=...</code> або циклом                                                  |

## Контрольні запитання

1. Що означає “доступ до ШІ через Python” і які переваги має Python для роботи з ШІ?
2. Для чого потрібен API-ключ при роботі з ШІ-сервісами і як забезпечується його безпека?

3. Які типові етапи обробки даних виконуються перед подачею даних у модель ШІ?
4. У чому полягає різниця між машинним навчанням і традиційним програмуванням?
5. Які основні типи машинного навчання ви знаєте (з прикладами задач)?
6. Чим глибинне навчання відрізняється від класичного машинного навчання?
7. У яких випадках доцільно використовувати нейронні мережі глибинного навчання?
8. Що таке попередньо навчені (pre-trained) моделі і які їхні переваги для розробника?
9. Наведіть приклади інтелектуальних сервісів ШІ та поясніть, для чого вони використовуються.
10. Які основні проблеми можуть виникати під час збору даних і як їх можна мінімізувати?
11. Чому очищення та підготовка даних є критично важливими для якості прогнозування?
12. Які методи аналізу даних застосовуються перед побудовою прогнозних моделей?
13. Що означає прогнозування за допомогою ШІ і в яких сферах воно найчастіше застосовується?
14. Як ШІ може допомагати в оптимізації рішень (приклади з бізнесу або техніки)?
15. Як основи теорії ігор використовуються в ШІ для побудови стратегій і прийняття рішень?

*Вивчення цього розділу дає студентам можливість оперативного виконання наукових дій з даними.*

## Розділ 3.

# ПАКЕТ МАШИННОГО НАВЧАННЯ AZURE

*У цьому розділі студенти ознайомляться з хмарним сервісом Azure, який дозволяє зменшити рутинну частину роботи дослідника.*

Microsoft Azure – це комплексна хмарна платформа від Microsoft, що надає широкий спектр послуг для розробки, розгортання та управління додатками й сервісами через інтернет, дозволяючи компаніям використовувати віртуальні машини, бази даних, ІІІ, аналітику та інше, замість володіння власною фізичною інфраструктурою. Azure функціонує за моделями IaaS (інфраструктура як послуга), PaaS (платформа як послуга) і SaaS (програмне забезпечення як послуга), забезпечуючи гнучкість, масштабованість та надійність для бізнесу будь-якого розміру, від стартапів до великих корпорацій.

Azure Machine Learning – це хмарний сервіс, який дозволяє: створювати, навчати, тестувати, розгортати моделі машинного навчання за допомогою Python і готових інструментів.

*Його ідея: менше ручної інфраструктури – більше роботи з моделями.*

### 3.1. Реєстрація в Microsoft Azure

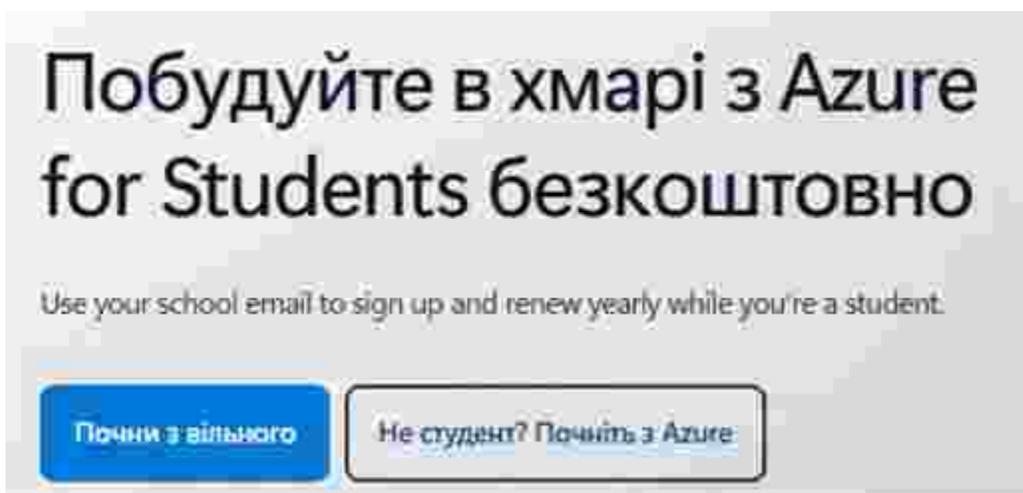
Щоб створити обліковий запис необхідно:

1. Перейти на сайт **portal.azure.com**
2. Увійдіть за допомогою студентської або особистої Microsoft-пошти.

Якщо ви вже маєте пошту Microsoft, вона підтягнеться автоматично.

3. Ведіть свій номер телефону і дочекайтеся смс з кодом доступу. Тепер код доступу буде вам відправлятися на ваш телефон перед кожним входом до Azure.

4. Оберіть пункт **Azure для студентів**, який має кредит у \$100, але цей кредит постійно оновлюється кожні пів року. Тобто, якщо у процесі навчання ви вичерпали всі ресурси кредиту, то через пів року знову зможете безкоштовно користуватися цим сервісом. Акаунт додається тільки при умові, що справді студент. Якщо ви спробуєте увійти як студент, не будучи ним, перевірка, яку робить Azure, вам відмовить в реєстрації.

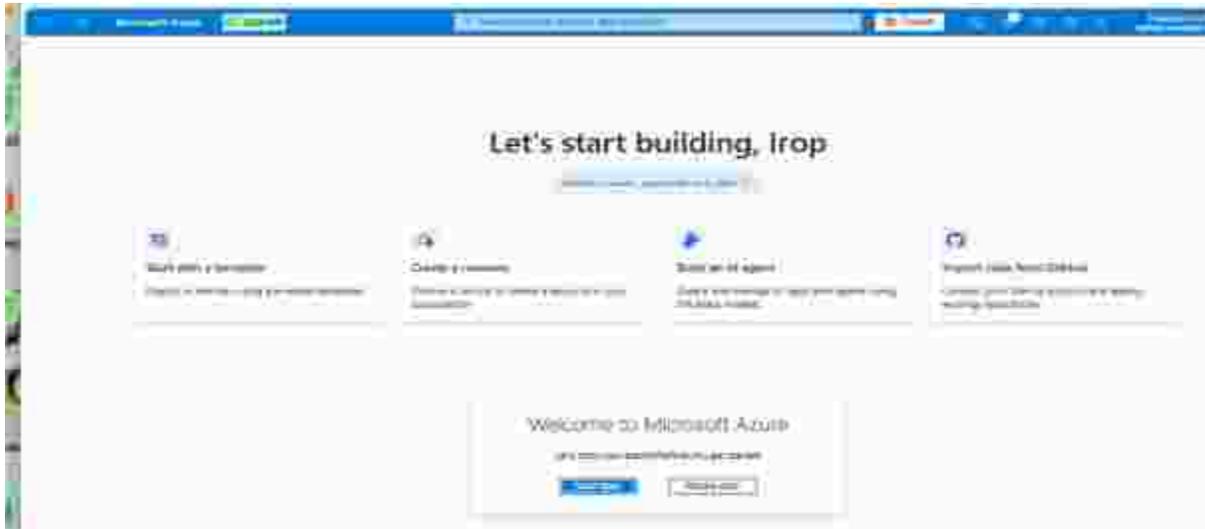


5. Варіант **Free Account** вимагатиме від вас введення реквізитів платіжної картки, вашої адреси, місця роботи, тощо. Цей варіант має кредит у \$200 на 30 днів. Коли кредит буде вичерпано, з вашої картки почнуть зніматися гроші. Зате цей варіант має більше можливостей для навчання можливостям системи Azure.

Після входу відкривається **Azure Portal** – центральне веб-середовище керування сервісами.

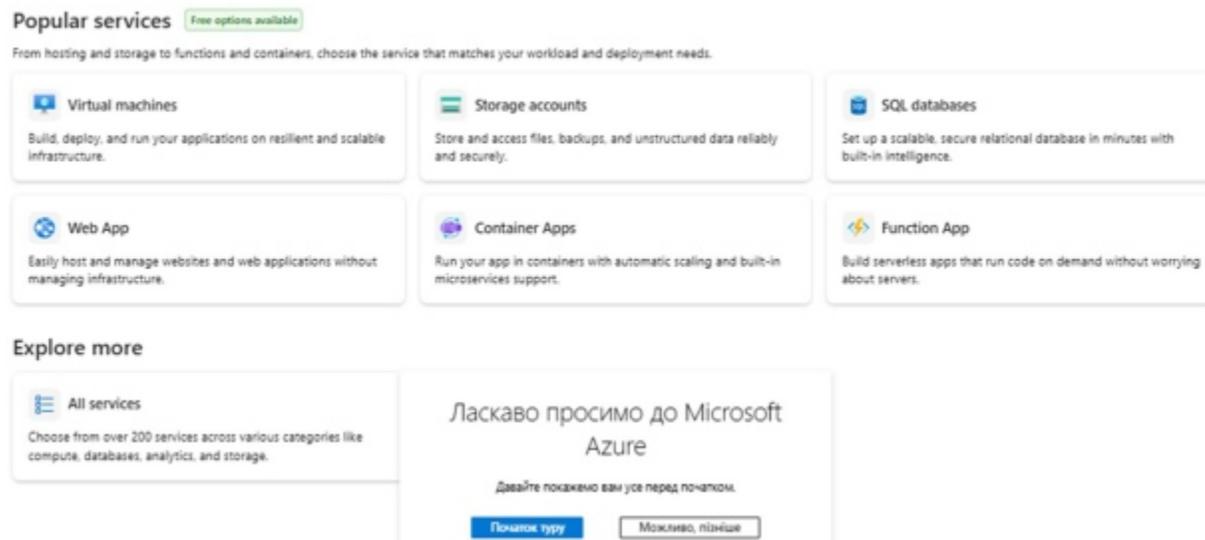
Основні елементи:

- ліве меню навігації;
- панель сервісів;
- пошуковий рядок;
- панель ресурсів.



### 3.2. Створення ресурсу

Оскільки після вивчення попередніх розділів ми всі маємо збережені програми на GitHub, то почнемо зі створення ресурсу, який забезпечить нам можливість деплоя та режим CI/CD без додаткових налаштувань. Для цього натиснемо відповідну кнопку на головній панелі сервісу.



З усіх можливостей спершу обираємо Web App і отримуємо анкету

Спробуйте Managed Instances (попередній перегляд) на Azure App Service: новий варіант, який надає переваги платформи, на які ви покладаетесь сьогодні, а також додаткові функції та гнучкість для безперешкодної модернізації [Дізнайтеся більше](#)

App Service Web Apps дозволяє швидко створювати, розгортати та масштабувати корпоративні веб-, мобільні та API додатки, які працюють на будь-якій платформі. Відповідати суворим вимогам продуктивності, масштабованості, безпеки та відповідності, використовуючи повністю керовану платформу для обслуговування інфраструктури. [Дізнайтеся більше](#)

#### Деталі проекту

Виберіть підписку для керування розгорнутими ресурсами та витратами. Використовуйте групи ресурсів, наприклад папки, щоб організувати та керувати всіма ресурсами.

Підписка \*

Ресурсна група \*   
[Створіть нове](#)

#### Деталі екземплярів

Ім'я  .azurewebsites.net

Увімкніть унікальне ім'я хоста за замовчуванням. [Детальніше про це оновлення](#)

Публікація \*  Код  Контейнер

Стек виконання \*

Операційна система  Linux  Windows

Регіон \*

**i** Не знаходите свій App Service Plan? Спробуйте інший регіон або виберіть середовище App Service Environment.

#### Ціноутворення

Тарифний рівень тарифного плану App Service визначає місцезнаходження, функції, вартість і обчислювальні ресурси, пов'язані з вашим додатком. [Дізнайтеся більше](#)

Linux Plan (Центральна Канада)   
**Виберіть групу ресурсів перед вибором плану.**

Тепер по пунктах анкети:

1. Підписка – залишаємо як є.
2. Ресурсна група – натискаємо Створити нове і у вікні, що відкриється вводимо назву нової ресурсної групи, наприклад ds-webapp-rg. Тут ds – назва проекту, webapp – тип ресурсу (Web App); rg – resource group (щоб одразу видно, що це група, а не сам додаток)
3. Ім'я веб-додатку (Name). Це буде URL користувача, що має структуру <назва>.azurewebsites.net. Правила: тільки англійські букви, цифри та дефіси; унікальне (Azure підкаже, якщо зайнято). Наприклад, ds-api-test. Після цього **URL** буде <https://ds-api-test.azurewebsites.net>.

4. Публікація (Publish). Оскільки плануємо деплоїти Python код, обираємо Код (Code). Контейнер потрібен тільки якщо хочеш Docker. Docker – це платформа для створення, доставки та запуску додатків у контейнерах. Контейнер – це легкий, ізольований пакет, який містить додаток і все необхідне для його роботи (код, бібліотеки, залежності, конфігурації). Головна перевага: додаток працює однаково на будь-якій системі, де встановлено Docker.

5. Стек виконання (Runtime stack). Обираємо Python, наприклад Python3.10, яку використовуємо локально, як на вашому своєму ноутбуку. Щоб узнати вашу версію, треба в терміналі подати команду `python –version`.

6. Операційна система (Operating system). Обираємо Linux, бо Linux дешевший та стабільніший для Python.

7. Регіон (Region). Варто обирати найближчий дата-центр, наприклад Canada Central (Центральна Канада). Чим ближче, тим швидше завантаження і менше затримка.

8. Ціноутворення. Linux Plan – придумайте назву типу `ds-webapp-plan`. App Service Plan – для початку варто обрати Basic B1 13,15 USD/Month (Estimated) ACU: 100, Memory: 1.75 GB, vCPU: 1. Пізніше можна все це змінити, якщо потрібно більше ресурсів.

Ім'я віртуальної мережі `ds-webapp-vnet-dev`.

Після цього натискаємо **Review + Create** (Перегляд + Створення) і коли з'являться параметри вашого ресурсу, натискаємо **Create (Створити)**, щоб завершити реєстрацію.

**Існує можливість реєстрації в Azure через термінал вашого комп'ютера.**

Для цього потрібно:

1. Завантажити MSI інсталятор для Azure CLI:  
<https://aka.ms/installazurecliwindows>
2. Запустити інсталятор і пройти всі кроки (Next → Install).
  - Під час інсталяції Azure CLI додасться в системний PATH.

- Після завершення інсталяції необхідно закрити і знову відкрити термінал. Це важливо, бо старий PowerShell ще не бачить az.

### 3. Перевірити установку командою

```
az --version
```

- Якщо бачите номер версії, установка пройшла успішно.
- ### 5. Виконати авторизацію у вашому акаунті Azure командою

```
az login
```

При цьому відкриється браузер для входу та на терміналі побачите список ваших підписок.

- ### 6. Тепер можна запускати усі команди для створення ресурсів, наприклад, використовуючи попередні дані:

```
az group create --name ds-webapp-rg --location eastus
```

```
az appservice plan create --name ds-webapp-plan --resource-group ds-webapp-rg --sku B1 --is-linux
```

```
az webapp create --resource-group ds-webapp-rg --plan ds-webapp-plan --name ds-api-test --runtime "PYTHON|3.11"
```

```
az resource tag --tags Purpose=Learning Environment=Test --name ds-api-test -resou
```

Підписка завершена.

**Третій варіант створення ресурсу на AZURE – через програму AS Code.**

Порядок дій:

1. Встановити розширення Azure у VS Code, для цього необхідно відкрити VS Code → Extensions, натиснувши Ctrl+Shift+X.

Потім знайти і встановити:

- Azure Account
- Azure Resource Manager (ARM) Tools
- Azure Virtual Machines (якщо плануєш VM)

### 2. Авторизація у VS Code

1. Після встановлення розширень натисніть **Azure** у бічній панелі.
2. Клікнути Sign in to Azure → авторизуватися через власний акаунт.

3. Після успішного входу ви побачите підписки, які доступні вашому акаунту.

- Якщо Subscription немає – доведеться або створити нову платну, або попросити адміністратора.

#### 4. Створення ресурсу через VS Code

- A. В Azure панелі VS Code натиснути + Create New Resource
- B. Вибрати ресурс, наприклад Virtual Machine
- C. Заповнити поля, наприклад:
  - **Resource Group:** ds-webapp-rg
  - **VM Name:** ds-webapp-vm1
  - **Region:** найближчий до тебе дата-центр
  - **Size / SKU:** B1
  - **Purpose / Tags:** для навчальних цілей (Learning / Test)
- D. Підтвердити і натиснути **Deploy**

Переваги цього способу над попередніми:

- Не треба чекати оновлення списку ресурсів у порталі.
- Можеш одразу вказати **Resource Group і ім'я ресурсу**.
- Зручно робити через **термінал VS Code**, якщо потрібні додаткові команди Azure CLI.

При створенні віртуальної машини (VM), Azure перевіряє, чи дозволяє поточна квота створювати ці ресурси. Якщо у підписці зараз немає дозволу створювати віртуальні машини цього типу, то для виправлення, потрібно запитати підвищення квоти.

Як правильно запитати нову квоту

1. Зайди в Azure Portal → Підтримка (Help + support) → Створити запит (Create a support request)
2. Тип запиту:
  - Issue type: Service and subscription limits (quotas)
  - Subscription: вибери ту, де потрібно створити VM

- Quota type: вибери ресурс, для якого потрібне підвищення, наприклад:
  - Virtual Machines
  - або конкретний тип VM (Standard Dv3, B-series тощо)
- 3. Вкажи нову кількість:
  - У полі “New limit” введи 1 (або більше, якщо плануєш декілька VM)
- 4. Обґрунтування (Justification):  
Напиши коротко, наприклад:  
"Потрібно розгорнути одну тестову віртуальну машину для експерименту/розробки."
- 5. Надіслати запит (Create)

### 3.3. Деплоу, як функція AZURE

Ми вже маємо створений App Service (наприклад, ds-api-test у групі ds-webapp-rg). Для того, щоб скористатися можливістю автономної роботи наших програм (тобто використати можливості деплою на AZURE), їх необхідно розмістити на GitHub, що ми вже вміємо робити.

Тепер найпростіший і найсучасніший спосіб зв'язати його з GitHub – це автоматичний CI/CD через GitHub Actions за допомогою Deployment Center в Azure Portal. Це робиться за 5–7 хвилин, і Azure сам створить потрібний workflow-файл у вашому репозиторії.

#### 3.3.1. Деплой для веб-додатків, які працюють постійно і створені на Пайтоні або на HTML

Крок 1: Перейдіть в Azure Portal

- Відкрийте <https://portal.azure.com>

- У пошуку зверху введи назву твого веб-додатку, наприклад ds-api-test

- Клікніть на нього, щоб відкрити огляд.

Крок 2: Відкритей Deployment Center. У лівому меню (ліворуч) знайдіть і клікніть Deployment Center (Центр розгортання). Там ви побачите вкладки: Settings (Налаштування) і Logs.

Крок 3: Оберіть джерело – GitHub.

- У розділі Source (Джерело) вибери GitHub.

- Якщо це перший раз – натисни Authorize (Авторизувати) або Continue.

- Відкриється вікно GitHub – увійдіть в свій GitHub-акаунт.

- Дозвольте Azure App Service доступ до репозиторіїв (можна обрати "All repositories" або тільки конкретні).

Після авторизації ви побачите список організацій і репозиторіїв.

Крок 4: Виберіть репозиторій і гілку

- Organization – ваша організація або ваш особистий акаунт (якщо репозиторій особистий).

- Repository – оберіть той репозиторій, де лежить ваш Python-код (наприклад, DS).

- Branch – оберіть гілку, з якої потрібно деплоїти (найчастіше main або master).

Крок 5: Виберіть спосіб аутентифікації:

У розділі Authentication type (Тип аутентифікації) оберіть:

- User-assigned identity (рекомендовано, безпечніше, використовує OpenID Connect). Azure сам створить managed identity і налаштує все. Потрібно, щоб у вас були права Contributor або Owner на ресурс (якщо ні – попросіть адміністратора або створіть новий акаунт з правами).

- Якщо цей варіант недоступний (через брак прав), оберіть Basic authentication (менш безпечно, але працює для тестів).

Крок 6: Збережіть налаштування.

- Прокрутіть вниз, натиснувши Save (Зберегти).
  - Azure автоматично створить файл `.github/workflows/azure-webapps-<твоя-назва>.yaml` у твоєму репозиторії; Додасть потрібні секрети в GitHub (`AZURE_CLIENT_ID`, `AZURE_TENANT_ID`, `AZURE_SUBSCRIPTION_ID` тощо); Почне перший деплой (може тривати 2–5 хвилин).

Крок 7: Перевірте, чи все запрацювало, для цього:

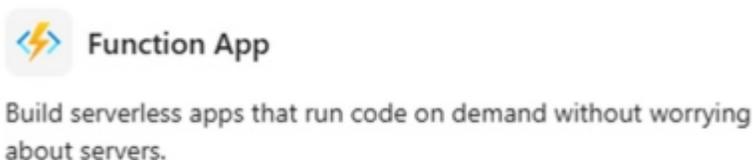
- Повернися в Deployment Center → вкладка Logs і Побачите статус останнього деплою (Succeeded / Failed).
- Перейдіть в GitHub → твій репозиторій → вкладка Actions і побачите workflow під назвою типу "Deploy to Azure" — клікни, щоб побачити логи.
- Відкрийте у браузері адресу твого додатку: <https://ds-api-test.azurewebsites.net> (якщо код правильний – має запуститися твій Python-додаток).
- Якщо перший деплой не спрацював – найпоширеніші причини (Python).
  - У репозиторії немає файлу `requirements.txt` у корені.
  - Немає головного файлу (наприклад, `app.py` або `main.py` або `index.html`).
  - Потрібен startup-команда (наприклад, для Flask: `gunicorn --bind=0.0.0.0 app:app`). → У Portal → Configuration → General settings → Startup Command — введіть її.

### 3.3.2. Деплой для щоденного запуску скрипта

Для навчального прикладу використано репозиторій DS в якому є папка 2 а в ній програма pursingакс.py. Ця програма і тільки вона має працювати в режимі деплоя кожного разу, як настане нова доба, тобто у 00:00. Результати цієї роботи мають потрапляти у мій же репозиторій A1, у файл у форматі Excel, хоча в самій програмі шлях до файлу прописаний на моєму комп'ютері як d:\A1\.

Крок 1: Створіть Azure Function App. Це основний "контейнер" для твоєї функції. Зробіть це в Azure Portal (portal.azure.com) — візуально простіше, ніж CLI.

1. Увійдіть в портал.
2. Натисніть "Create a resource" (Створити ресурс) – "Function App". У запиті Hosting Plan оберіть Flex Consumption і натисніть Select.



Створити функціональну програму

Select a hosting option  
These options determine how your app scales, resources available per instance, and pricing. Learn more about Functions hosting options.

| Hosting plans                            | Flex Consumption                                                                         | Functions Premium                                                         | App Service                                                                                                     | Container Apps environment                                                              | Consumption                                                              |
|------------------------------------------|------------------------------------------------------------------------------------------|---------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------|--------------------------------------------------------------------------|
|                                          | Get high scalability with compute choice, virtual networking, and pay-as-you-go billing. | Deploy multiple function apps on the same plan with event-driven scaling. | Run web apps and function apps on the same plan with more compute choice and pay for the instances of the plan. | Host function apps with other containerized microservices and pay for compute capacity. | Pay for compute resources when your functions are running pay-as-you-go. |
| Scale to zero                            | ✓                                                                                        | -                                                                         | -                                                                                                               | ✓                                                                                       | ✓                                                                        |
| Scale behavior                           | Not event-driven                                                                         | Event-driven                                                              | Metric based                                                                                                    | Event-driven with DCS                                                                   | Event-driven                                                             |
| Virtual networking                       | ✓                                                                                        | ✓                                                                         | ✓                                                                                                               | ✓                                                                                       | -                                                                        |
| Dedicated compute and prevent cold start | Optional with Always Ready                                                               | Minimum of 1 instance required                                            | Minimum of 1 instance required                                                                                  | Optional with minimum replicas                                                          | -                                                                        |
| Max waste out (instances)                | 1000                                                                                     | 100                                                                       | 10                                                                                                              | 100                                                                                     | 200                                                                      |

#### Порівняння варіантів плану

| План                    | Ціна (щоденний скрипт)       | Плюси для не частого виконання                       | Мінуси для виконання                                  | Рекомендація         |
|-------------------------|------------------------------|------------------------------------------------------|-------------------------------------------------------|----------------------|
| Consumption (класичний) | Майже безкоштовно (в гранті) | Найдешевше, автоматичне масштабування, scale to zero | Можливі "cold starts" (затримка 1–10 сек при першому) | Найкраще для початку |

| План                                                        | Ціна (щоденний скрипт)                               | Плюси для не частого виконання                                   | Мінуси для виконання                             | Рекомендація               |
|-------------------------------------------------------------|------------------------------------------------------|------------------------------------------------------------------|--------------------------------------------------|----------------------------|
|                                                             |                                                      |                                                                  | запуску після довгої паузи)                      |                            |
| <b>Flex Consumption</b> (новіший, рекомендований Microsoft) | Майже безкоштовно (трохи менший грант)               | Швидше масштабування, менше cold starts, краща інтеграція з VNet | Трохи менший безкоштовний грант                  | Добре, якщо є в списку     |
| <b>Premium</b> (Elastic Premium, EP1 тощо)                  | Фіксована плата ~\$70–200+/міс (залежно від розміру) | Немає cold starts, потужніше, VNet                               | Дорого для рідкісних запусків                    | Не варто брати – переплата |
| <b>Dedicated</b> (App Service Plan)                         | Фіксована ~\$13+/міс (B1) і більше                   | Повний контроль, завжди онлайн                                   | Платиш весь місяць, навіть якщо нічого не робить | Не варто брати – дорого    |

Якщо ви працюєте з України, краще увімкнути VPN і після цього заповніть форму

#### Деталі проекту

Виберіть підписку для керування розгорнутими ресурсами та витратами. Використовуйте групи ресурсів, такі як папки, для впорядкування та керування всіма вашими ресурсами.

Підписка \* 

Група ресурсів \*   [Створити нове](#)

#### Деталі екземпляра

Назва функціональної програми \*    
-dvhfeccec9frdahw.canadacentral-01.azurewebsites.net

Безпечне унікальне ім'я хоста за замовчуванням увімкнено.  
[Докладніше про це оновлення](#)

Регіон \*

Стек виконання \*

Версія \*

Розмір екземпляра \* 

#### Зонна резервування

У форму вкажіть:

- **Subscription:** Ваш (з кредитом).
- **Resource Group:** Використовуйте існуючу ds-webapp-rg (або створіть нову, наприклад ds-functions-rg).
- **Function App name:** Унікальне, наприклад ds-daily-script.
- **Publish:** Code (Код).
- **Runtime stack:** Python (обери версію, наприклад 3.13 або найновішу, як у твоєму App Service).
- **Region:** West Europe (ближче до України). Але Канада підійде.
- **Hosting:** Consumption (безкоштовний план для тестів, плати тільки за виконання – ідеально для щоденного скрипту).
- **Storage:** Створи новий (потрібен для тригерів).

2. Натисніть "Review + create" > "Create". Чекає 2–5 хвилин.

Якщо створення припинено, натисніть кнопку [< Попередній](#) і дійдіть до пункту Розгортання, де необхідно внести параметри вашого репозиторію на GitHub.

Безперервне розгортання  Вимкнути  Увімкнути

### Налаштування GitHub

Налаштуйте дії GitHub для надсилання контенту до вашої програми щоразу, коли у вашому репозиторії вносяться зміни коду. Примітка: ваш обліковий запис GitHub повинен мати доступ для запису до вибраного репозиторію, щоб додати файл робочого процесу, який керує розгортаннями у вашій програмі.

Обліковий запис GitHub

Організація \*

Репозиторій \*

Філія \*

Тепер, коли створена підписка, яка активна, права на створення ресурсів є, ресурс-група ds-webapp-rg існує, і ми можемо створювати ресурси.

Найкращий і найпростіший спосіб це зробити – Azure Functions з Timer trigger.

Для цього у терміналі потрібно виконати такі команди:

Для перевірок, що ми в потрібній підписці

```
az account show --output table
```

Для перевірки resource group

```
az group show --name ds-webapp-rg --output table
```

Для створення resource group, якщо ще не існує

```
az group create --name ds-webapp-rg --location westeurope
```

Для створення storage account (обов'язково для Functions)

```
az storage account create --name dsfuncstor20260210igor --resource-group ds-webapp-rg --location westeurope --sku Standard_LRS --kind StorageV2
```

Для створення Function App на Flex Consumption з Python 3.12

```
az functionapp create --resource-group ds-webapp-rg --name ds-daily-parser-20260210 --storage-account dsfuncstor20260210igor --runtime python --runtime-version 3.12 --functions-version 4 --flexconsumption-location westeurope
```

Не забувайте, що це тільки приклад тому оцю частину команди **dsfuncstor20260210igor** треба замінити на вашу.

Після виконання останньої команди потрібно зачекати 3–5 хвилин, поки ресурс повністю розгорнеться.

Тепер потрібно змінити шлях викладення файлу готових результатів. У програмі був код

```
file_path = r"D:\A1\stocks_parsing_part1_with_returns.xlsx"
```

Його потрібно змінити на

```
file_path = os.path.join("/tmp", "result.xlsx")
```

/tmp – це Linux-подібна директорія в Azure.

Тепер необхідно встановити Azure Functions на вашому AS Code.



Тепер в репозиторії DS створюємо папку FunctionApp в якій створюємо ще одну папку – MyFuncApp, у якій ще одну – DailyRunner. Назва останньої відповідає назві функції в AZURE.

В останню ми розміщаємо 2 файли. Перший має назву \_\_init\_\_.py.

```
import logging
import azure.functions as func
import sys
from pathlib import Path

Додаємо шлях до DS/2
sys.path.append(str(Path(__file__).parent.parent.parent / "2"))

from pursingakc import main as pursingakc_main

def main(mytimer: func.TimerRequest) -> None:
 logging.info("Azure Function timer trigger ran.")
 pursingakc_main()
```

Другий файл називається function.json.

```
{
 "bindings": [
 {
 "name": "mytimer",
 "type": "timerTrigger",
 "direction": "in",
 "schedule": "0 0 0 * * *"
 }
]
}
```

Далі, у папці MyFuncApp розміщаємо 3 файли. Перший має назву host.json.

```
{
 "version": "2.0",
 "logging": {
 "applicationInsights": {
 "samplingSettings": {
 "isEnabled": true,
 "maxTelemetryItemsPerSecond" : 5
 }
 }
 }
}
```

Другий файл називається requirements.txt.

```
pandas
openpyxl
requests
```

Третім файлом має бути ваша програма на Python.

Нижче показана структура цих папок і файлів.

```
DS/
├─ FunctionApp/
│ └─ MyFuncApp/
│ ├─ DailyRunner/
│ │ └─ __init__.py
│ │ └─ function.json
│ └─ host.json
│ └─ requirements.txt
│ └─ pusingaks.py ← твій скрипт тут фізично
```

---

Щоб запустити роботу створеного середовища через AZURE, необхідно виконати декілька процедур:

1 – Перевірка середовища

- Переконайся, що встановлений **Python 3.x**
- Переконайся, що встановлений **Azure Functions Core Tools**. Для цього у терміналі напишіть `func --version`
- Перевірте, що знаходишся у корені Function App, командою в терміналі **`DS/FunctionApp/MyFuncApp/`**

2 – Встановлення залежностей.

Перейти у корінь Function App через команду

```
cd d:\DS\ Function App
```

а потім подати команду

```
pip install -r requirements.txt
```

Це встановить pandas, openpyxl, requests, azure-functions

### 3 – Локальний запуск Function App

У терміналі у корені Function App подати команду

**func start**

Azure Function Core Tools підніме локальний сервер і ви побачите повідомлення типу:

**Hosting environment: Production**

**Content root path: ...**

**Now listening on: http://localhost:7071**

**Application started. Press Ctrl+C to shut down.**

Timer Trigger автоматично спрацює згідно CRON у function.json. Для тесту можна тимчасово поставити розклад на кожні 1-2 хвилини, наприклад:

```
"schedule": "*/2 * * * *"
```

### 4 – Перевірка виклику скрипта

- У виводі терміналу побачиш лог:

Azure Function timer trigger ran.

- І має виконатися `pursingaks.py` → створитися Excel → пуш в GitHub, якщо налаштований токен

5 – Зупинка локального запуску. Для цього достатньо натиснути Ctrl+C у терміналі, щоб зупинити сервер.

### 6 – Деплой на Azure через CLI.

- Створюємо ресурсну групу (якщо ще не створена):
- `az group create --name ds-webapp-rg --location eastus`
- Створюємо Function App:

```
az functionapp create \
--resource-group ds-webapp-rg \
--consumption-plan-location eastus \
--runtime python \
--functions-version 4 \
--name ds-webapp-func \

```

**--storage-account <унікальна\_назва\_storage>**

- Деплой локального Function App:

**func azure functionapp publish ds-webapp-func**

- Azure автоматично підхопить CRON і Timer Trigger буде виконуватися щодня о 00:00 UTC.

### **3.4. Створення середовища Azure Machine Learning**

Azure Machine Learning Environment – це ізольоване середовище для запуску машинного навчання.

Основна мета: гарантувати, що модель або скрипт працюватимуть однаково у будь-якому середовищі – локально, у хмарі або на compute target.

Основні цілі середовища AML:

1. Контроль версій Python та пакетів
  - Точно визначаєш, які бібліотеки (pandas, scikit-learn, tensorflow, torch) та версії використовуються.
  - Це виключає “у мене працює, а у хмарах ні” ефект.
2. Повторюваність
  - Один і той самий скрипт або експеримент можна запускати на різних compute ресурсах і отримувати однаковий результат.
3. Ізоляція
  - Кожне середовище ізольоване від інших проектів, тож не виникає конфліктів пакетів.
4. Використання з compute targets
  - Можна запускати експерименти на локальному комп’ютері, на віртуальних машинах Azure або на GPU/CPU кластерах, не змінюючи код.
5. Docker-підтримка

- Середовище може бути упаковане як Docker image, щоб його можна було деплоїти у Production.

#### Приклади використання

- Під час експериментів ML, щоб перевірити різні моделі і гіперпараметри
- Для деплою моделей у веб-сервісах або API
- Для спільної роботи команди, щоб всі використовували одне і те ж середовище

Розкладемо покроково створення середовища Azure Machine Learning (AML) на простому навчальному прикладі з урахуванням того, ваш акаунт уже є, і ми робимо все для Python скрипта, який тренує чи виконує просту модель або обчислення.

#### **Крок 0. Підготовка. Ми повинні мати:**

- Зареєстрований акаунт Azure
- Встановлений Python 3.x
- Встановлені Azure CLI і Azure ML SDK:
- `pip install azureml-core azureml-sdk`
- (Опційно) VS Code з Azure ML Extension

#### Крок 1. Створення Workspace у Azure ML.

Workspace – це ваш “проект” в AML, де зберігаються всі середовища, моделі, експерименти. Для цього переходимо по головному меню за схемою Azure Portal → Machine Learning → + Create.

При цьому потрібно вказати:

- Subscription → ваш акаунт
- Resource group → наприклад, ds-ml-rg
- Workspace name → наприклад, ds-ml-workspace
- Location → будь-який доступний регіон

Після цього потрібно натиснути **Review + Create** → **Create** і створюється Workspace – це як “папка-проект” для всіх експериментів ML.

## Крок 2. Створення середовища (Environment)

Environment – це Python + пакети + налаштування, щоби скрипт працював однаково у будь-якому compute target.

Для цього у Workspace → Environments → + Create → From conda specification потрібно вказати:

- Name → ds-learning-env
- Python version → 3.10
- Пакети: pandas, numpy, scikit-learn

Далі натискаєте Create.

Можна ці дії виконати через Python SDK (локально). Для цього необхідно створити файл environment.yml у твоєму проекті:

```
name: ds-learning-env
dependencies:
 - python=3.10
 - pip:
 - pandas
 - numpy
 - scikit-learn
```

І у Python скрипті:

```
from azureml.core import Workspace, Environment

ws = Workspace.from_config() # читає config.json з Workspace
myenv = Environment.from_conda_specification(name="ds-learning-env",
file_path="environment.yml")
myenv.register(workspace=ws)
```

Тепер середовище зареєстроване у Workspace і готове для експериментів.

## Крок 3. Використання середовища для запуску експерименту

Створюєте скрипт train.py (навіть простий, наприклад, робить прогноз на випадкових даних).

```
import pandas as pd
import numpy as np
print("Hello AML! Прогноз на простих даних:")
data = np.random.rand(5, 2)
df = pd.DataFrame(data, columns=["X1", "X2"])
print(df.sum(axis=1))
```

У Python створюєте ScriptRunConfig:

```
from azureml.core import ScriptRunConfig, Experiment

src = ScriptRunConfig(source_directory=".", script="train.py",
environment=myenv)

exp = Experiment(workspace=ws, name="ds-learning-experiment")
run = exp.submit(src)
run.wait_for_completion(show_output=True)
```

Тут `environment=myenv` – ваш скрипт запуститься у середовищі AML, яке було створене. `Experiment` – записує всі запуски та результати.

### 3.5. Створення власного агента III (Build an AI Agent)

AI Agent – це програма або об’єкт, який приймає рішення і виконує дії автономно, на основі даних або команд користувача.

В контексті сучасних платформ (наприклад, OpenAI, Azure, LangChain) “Build an AI Agent” означає створити інтелектуального агента, який може:

- обробляти запити на природній мові
- виконувати кроки у певній послідовності
- автоматично запускати скрипти, аналізувати дані, відправляти повідомлення

Простими словами: це “розумний помічник”, який виконує команди клієнта.

AI Agent використовується для наступних завдань:

- Автоматизація задач. Наприклад: збирати дані з вебсайтів, обробляти їх і зберігати у Excel
- Асистування користувачу. Наприклад, Чат-бот, який відповідає на запити або допомагає знайти інформацію
- Інтелектуальні робочі процеси. Наприклад: якщо нові дані з’являються у репозиторії, агент сам виконує аналіз і надсилає результат
- Модульність. Один агент може виконувати кілька кроків, комбінуючи різні модулі (API, скрипти, ML-моделі)

Як працювати з Build an AI Agent:

1. Визначаємо мету агента, що він має робити? (Наприклад, запуск Python скрипта щодня і пуш результату в репозиторій).
2. Налаштовуємо дії агента, Наприклад, отримати дані, Обробити / виконати скрипт, Зберегти результат
3. Обираємо середовище виконання, наприклад, виконувати обчислювальні дії локально, чи на сервері, чи у хмарі (Azure, AWS, GCP).
4. Додаємо інтеграції з іншими обчислювальними потужностями, наприклад, API, GitHub, база даних, Excel, Slack тощо.
5. Тестуємо агента. Для цього перевіряємо, чи виконує всі дії коректно. Можна запускати в “симуляції” на Azure або локально.

### 3.5.1. ML-модель

**ML-модель (Machine Learning model)** – це математична модель, яка навчається на даних і вчиться робити прогноз або приймати рішення без жорстко прописаних правил.

Тобто, це формула або алгоритм, який знаходить закономірності в даних і використовує їх для прогнозування.

Уявімо, що ми маємо такі дані:

| Площа квартири    | Ціна    |
|-------------------|---------|
| 40 м <sup>2</sup> | 50 000  |
| 60 м <sup>2</sup> | 75 000  |
| 80 м <sup>2</sup> | 100 000 |

ML-модель вивчає залежність між площею і ціною і потім може сказати:

Для 70 м<sup>2</sup> приблизна ціна буде 87 000.

Ми не прописували формулу вручну – модель її "вивела" сама з даних.

ML-модель складається з:

1. Дані (dataset) – на чому модель навчається
2. Алгоритм – спосіб знаходження закономірностей
3. Навчання (training) – процес підбору параметрів
4. Прогноз (prediction) – результат роботи моделі

## Основні типи ML-моделей

### 1 Регресія

Прогноз числового значення

- Прогноз цін
- Прогноз продажів
- Прогноз ВВП

Приклад:

Linear Regression

### 2 Класифікація

Визначення категорії

- Спам / не спам
- Кредит надавати / ні
- Дефолт / не дефолт

### 3 Кластеризація

Групування об'єктів без готових міток

- Сегментація клієнтів
- Групування товарів

### 4 Нейронні мережі

Складні моделі для:

- Розпізнавання зображень
- Обробки тексту
- Генерації мови

**Чим ML відрізняється від звичайного програмування**

**Звичайна програма:**

Правила + Дані → Результат

**Machine Learning:**

Дані + Результат → Модель (виводить правила)

Тобто модель сама знаходить правило.

**У твоєму випадку**

Якщо ти робиш прогноз у Python або Statistica:

- Якщо ти просто підставляєш формулу → це не ML
- Якщо ти тренуєш модель на даних → це вже ML

Наприклад, команда

```
from sklearn.linear_model import LinearRegression
```

імпортує реалізацію алгоритму лінійної регресії з бібліотеки scikit-learn для подальшого створення та навчання ML-моделі. Команда забезпечує доступ до алгоритму лінійної регресії, який використовується для оцінювання параметрів моделі та формування прогнозів на основі навчальної вибірки. Команда імпортує алгоритм, який є складовою методів машинного навчання (ML), що належать до інструментарію штучного інтелекту.

### 3.5.2. Приклад використання Build an AI Agent

Почнемо з реєстрації, для цього розглянемо сторінку, що відкривається після вибору кнопки Build an AI agent на основному вікні AZURE.

Це вікно вибору моделі (**Model deployment**). Те, що ви бачите – це каталог моделей, а не створення агента.

У вікні зверху написано:

#### **Model deployment → GPT-4o mini**

Це означає, що ви створюєте **deployment** моделі, а не агента.

#### **Чому так?**

В Microsoft Azure логіка така:

Спочатку ви **розгортаєте модель (Deploy model)**. Потім цю модель використовуєте або через API, або в AI Agent, або в Playground

У цьому вікні вам треба просто обрати **GPT-4o mini** (він уже вибраний і інші варіанти складніші, їх поки не варто використовувати).

Натиснути **Create an Agent**. Після цього створиться наступна таблиця.

В ній вже є деякі ваші параметри, але при створенні імені проєкту треба дотримувати наступних правил:

- Azure дозволяє тільки дефіси - і підкреслення \_ після першого символу.
- Замініть усі крапки на дефіси або підкреслення.
- Переконайтеся, що перший символ — буква або цифра.
- Довжина в межах 3–33 символів.

Після натискання кнопки Create, треба зачекати деякий час на створення проєкту.

Далі з'явиться вікно Deploy a Model, у якому можна обрати конкретну модель. Натиснувши на її назву, ви отримаєте її опис.

## Create a new project

To try out the generally available Agent experience and access the latest AI Foundry capabilities, you must create a project.

### Project \*

### Advanced options

We'll set up a new project for you with defaults selected for optimal functionality  
Your project will be located in the swedencentral region. ^

#### Project

(new) Pistunov\_I\_M\_2621

#### Microsoft Foundry resource \*

#### Subscription \*

 ▼

[Create a new subscription](#) 🔗

#### Public network access

Enabled

#### Resource group \*

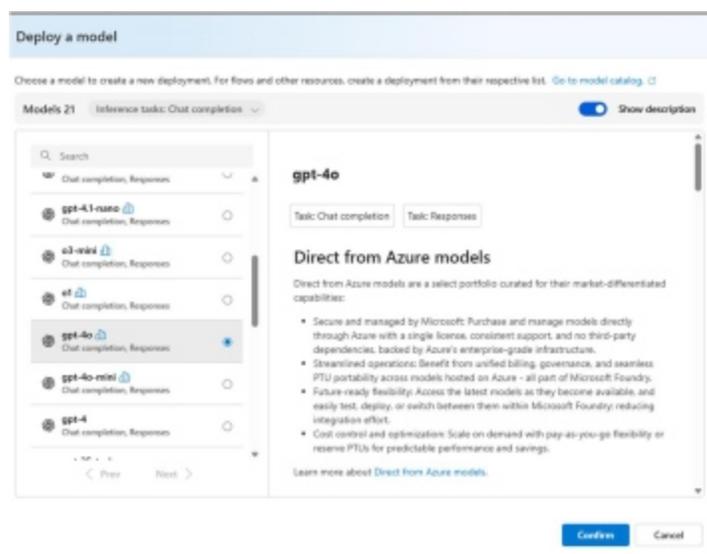
 ▼

[Create new resource group](#)

#### Region \*

 ▼

Got data, privacy or security policies to follow? [Configure in Azure Portal](#) 🔗



Яку обрати? Ось таблиця з коротким описом можливостей деяких агентів.

| Тип завдання агента                            | Рекомендована модель                           | Пояснення                                                                    |
|------------------------------------------------|------------------------------------------------|------------------------------------------------------------------------------|
| Загальний чат, відповіді на питання            | <b>gpt-4o-mini</b>                             | Швидка, недорога, добре справляється з типовими запитами та діалогами.       |
| Складний аналіз тексту, логіка, генерація коду | <b>gpt-4o</b>                                  | Потужніша, точніше обробляє складні завдання, більший контекст.              |
| Економія бюджету, базові консультації          | <b>gpt-3.5-turbo</b>                           | Дешевше, підходить для простих чатів, не так глибоко розуміє складні запити. |
| Генерація або аналіз коду                      | <b>code-davinci-002</b> або <b>gpt-4o-code</b> | Оптимізовані під програмування та код, можуть писати та перевіряти код.      |
| Пошук та обробка інформації у документах       | <b>text-embedding-3-small/large</b>            | Створюють векторні представлення тексту для пошуку та семантичного аналізу.  |

Якщо це перший агент і ви хочете просто спробувати чат/асистента – варто обрати **gpt-4o-mini**.

Після обрання моделі, з'являється вікно, у якому треба все вірно заповнити і зрозуміти опис вашого агента.

**Deploy gpt-4o-mini**

Deployment name \*  
PistunovIM\_Deploy

Deployment type  
Global Standard

Global Standard: Pay per API call with the highest rate limits. Learn more about [Global deployment types](#).

Data might be processed globally, outside of the resource's Azure geography, but data storage remains in the AI resource's Azure geography. Learn more about [data residency](#).

Deployment details Customize

|                                          |                                                                   |
|------------------------------------------|-------------------------------------------------------------------|
| Model version<br>2024-07-18              | Authentication type<br>Key                                        |
| Capacity<br>100K tokens per minute (TPM) | Resource location<br>Sweden Central                               |
| Content safety<br>DefaultV2              | Version upgrade policy<br>Once a new default version is available |

Deploy Cancel

### Deployment name. Поле обов'язкове (Required).

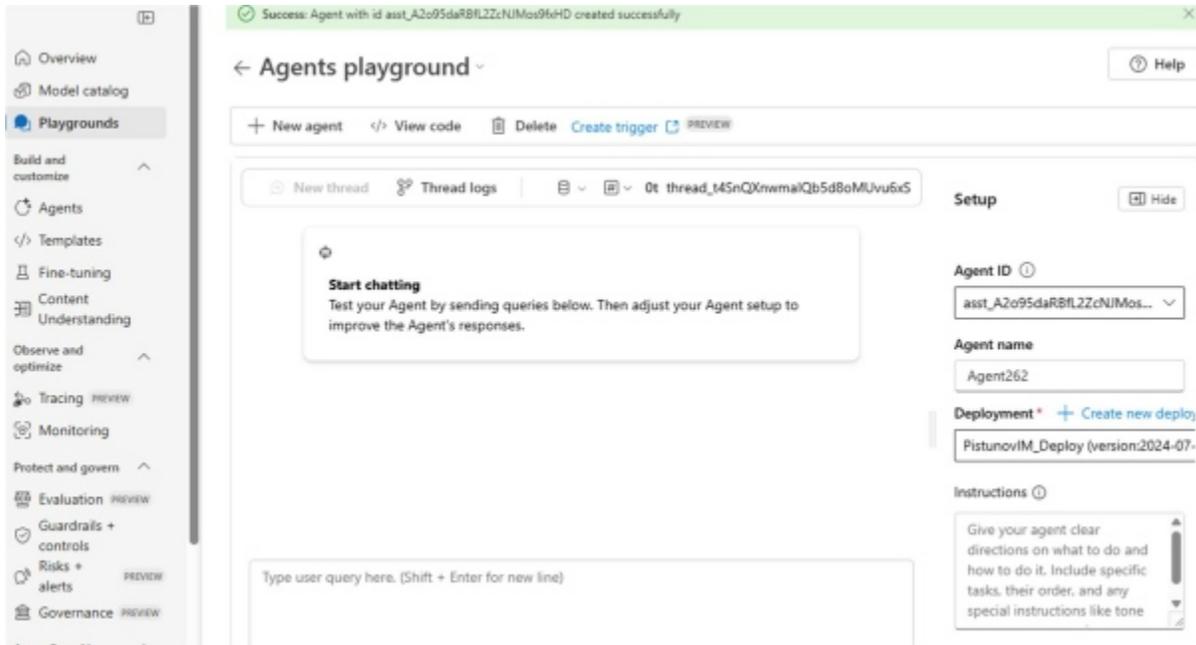
Це ім'я вашого деплойменту для цієї моделі. Правила ті ж, що для імені агента: 3–33 символи, перший символ – буква або цифра, дозволені лише дефіси - та підкреслення \_, пробіли та крапки заборонені. Наприклад, gpt4o\_mini\_deploy або PistunovIM\_Deploy.

**Deployment type.** Варіант **Global Standard** – найпоширеніший варіант. У ньому оплата береться за запит (pay per API call), має високі ліміти, дані можуть оброблятися глобально, але зберігатися у вашому регіоні.

### Deployment details.

- **Model version:** 2024-07-18 — це актуальна версія моделі
- **Capacity:** 100K tokens per minute (TPM) — скільки токенів можна обробити за хвилину
- **Content safety:** DefaultV2 — стандартні правила безпеки контенту
- **Authentication type:** Key — тобі знадобиться ключ для API
- **Resource location:** Sweden Central — де фізично зберігається ресурс
- **Version upgrade policy:** Once a new default version is available — автоматичне оновлення моделі, коли виходить нова версія

Натиснувши кнопку Deploy, ми отримаємо свого агента.



Те, що ви бачите ліворуч – це панель навігації **Azure AI Studio**, де зібрані всі доступні інструменти та сервіси для роботи з AI. Щоб зорієнтуватися, розберемо головні розділи:

### Основні робочі секції

- **Model catalog** – каталог всіх доступних моделей, які можна деплоїти.
- **Playgrounds** – інтерактивне тестування моделей у браузері (можеш одразу вводити запити й бачити відповіді).
- **Build and customize** – створення нових агентів, налаштування їх поведінки.

### Робота з агентами

- **Agents** – твої агенти (включно з PistunovIM).
- **Templates** – готові шаблони агентів під різні задачі.
- **Fine-tuning** – додаткове навчання моделі на своїх даних.
- **Content Understanding** – аналітика текстів, класифікація, розпізнавання намірів.
- **Observe and optimize** – моніторинг і оптимізація роботи агентів.

### Контроль і безпека

- **Tracing (PREVIEW)** – відстеження запитів до моделі.
- **Monitoring** – статистика та ліміти використання.

- **Protect and govern** – безпека, політики доступу.
- **Evaluation (PREVIEW)** – оцінка якості моделей.
- **Guardrails + controls, Risks + alerts, Governance (PREVIEW)** – управління ризиками, правилами, етичними обмеженнями.

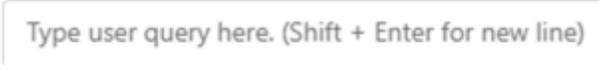
### Робота з даними та моделями

- **Azure OpenAI** – доступ до моделей OpenAI в Azure.
- **Stored completions** – збережені відповіді моделей.
- **Batch jobs** – пакетна обробка запитів.
- **My assets** – твої ресурси, наприклад, файли або моделі.
- **Data + indexes** – завантаження й індексація даних для пошуку або навчання.
- **Models + endpoints** – всі твої моделі та їх кінцеві точки для API.

### Адміністрування

- **More** → **Management center** – загальне управління ресурсами, білінг, користувачі.

Для початку варто відкрити **Playground**. Там можна надіслати перший запит до gpt-4o-mini і побачити, як агент відповідає. Скористайте вікно внизу

екрану , куди введіть ваш запит.

Наприклад, «Створи графік зміни населення України з 1990 року по наш час».

Відповідь така: «На жаль, я не можу створювати графіки безпосередньо у текстовому форматі. Однак я можу надати вам дані, що стануть в нагоді для створення графіка зміни населення України з 1990 року до сьогодні».

#### **Населення України (1990-2023):**

- **1990 рік:** ~52,2 мільйона
- **1995 рік:** ~51,0 мільйона
- **2000 рік:** ~48,2 мільйона
- **2005 рік:** ~47,4 мільйона

- **2010 рік:** ~45,9 мільйона
- **2015 рік:** ~42,9 мільйона
- **2020 рік:** ~41,4 мільйона
- **2023 рік:** ~40,0 мільйона (оціночно)

**Як створити графік:** Використайте програму для роботи з таблицями (наприклад, Microsoft Excel, Google Sheets).»

Як би я задав це запитання ChatGPT через його сайт, то отримав би і графік, але обрана модель має обмежені можливості.

Тепер спробуємо цю модель для збору складніших даних. Наприклад, «Збери дані про вартість акцій фірми Chevron з 1.01.2023 по наш час і розмісти їх у файлі Excel».

Відповідь «На жаль, я не можу безпосередньо створювати або надсилати файли, включаючи Excel. Але я можу дати вам вказівки, як самостійно зібрати дані про вартість акцій компанії Chevron (CVX) за потрібний період і зберегти їх у файлі Excel.

### **Кроки для збору даних про акції Chevron:**

#### **Відкрийте фінансовий веб-сайт:**

- Використовуйте сайти, такі як Yahoo Finance, Google Finance або Bloomberg.

#### **2. Знайдіть акції Chevron:**

- Введіть "Chevron" або символ "CVX" у пошукову строку на сайті.

#### **3. Перейдіть до графіку або історії цін:**

- Знайдіть розділ, який містить історичні дані (як правило, це називається "Historical Prices").

#### **4. Встановіть діапазон дат:**

- Виберіть діапазон з 1 січня 2023 року до даного моменту.

#### **5. Завантажте дані:**

- Багато сайтів пропонують можливість експортувати дані у форматі CSV або Excel.

## 6. Відкрийте дані в Excel:

- Якщо ви завантажили CSV файл, відкрийте його в Excel.
- Якщо дані ще не в Excel, скопіюйте та вставте їх у новий лист Excel.

## 7. Зберігайте файл:

- Збережіть файл у форматі Excel (.xlsx).

Загальна структура файлу Excel може виглядати так:

| Дата       | Вартість закриття | Вартість відкриття | Найвища ціна | Найнижча ціна | Обсяг |
|------------|-------------------|--------------------|--------------|---------------|-------|
| 2023-01-02 | XX.XX             | XX.XX              | XX.XX        | XX.XX         | XXXX  |

»

Тепер спробуємо використати цього агента для виконання задачі деплою, як у попередніх розділах. Для навчального прикладу використано репозиторій DS в якому є папка 2 а в ній програма `pursingакс.py`. Ця програма і тільки вона має працювати в режимі деплою кожного разу, як настане нова доба, тобто у 00:00. Результати цієї роботи мають потрапляти у мій же репозиторій A1, у файл у форматі Excel, хоча в самій програмі шлях до файлу прописаний на моєму комп'ютері як `d:\A1\`.

Для виконання цього завдання на вашому комп'ютері створіть наступну структуру файлів і папок у папці DS.

```
DS-Function-App/
├─ RunDailyScript/
│ ├── __init__.py
│ └─ function.json
├─ pursingакс.py + скрипт з DS/2
├─ requirements.txt
└─ host.json
```

DS-Function-App/ це окрема папка, окремо від вашого репозиторію DS.

Мета такої структури:

- DS – ваш оригінальний репозиторій з кодом та даними.
- DS-Function-App – проект спеціально для Azure Function, який містить твій скрипт `pursingакс.py` (копію з DS/2), конфігурацію функції і `requirements.txt`.

Тепер у файлі `pursingакс.py` необхідно внести зміни в адресації вихідного файлу на Excel наступним чином.

```
import pandas as pd

--- тут твій код парсингу акцій ---
Наприклад:
data = {"Ticker": ["AAPL", "MSFT"], "Price": [150, 320]}
df = pd.DataFrame(data)

Збереження Excel у робочу директорію (відносний шлях)
output_path = "output.xlsx" # раніше був d:\A1\output.xlsx
df.to_excel(output_path, index=False)

print(f"Файл Excel збережено: {output_path}")
```

Зміст файлу `init.py` має бути

```
import subprocess
import os
import requests
import json
import base64

Перехід у робочу директорію Function App (де лежить pursingакс.py)
os.chdir(os.path.join(os.path.dirname(__file__), ".."))

Запуск скрипта
subprocess.run(["python", "pursingакс.py"], check=True)

Завантаження output.xlsx у GitHub репозиторій A1
GITHUB_TOKEN = os.environ["GITHUB_TOKEN"]
REPO = "username/A1" # заміни на свій репозиторій
FILE_PATH = "output.xlsx"
BRANCH = "main"
COMMIT_MESSAGE = "Daily stock data update"
```

```

with open(FILE_PATH, "rb") as f:
 content = base64.b64encode(f.read()).decode()

url = f"https://api.github.com/repos/{REPO}/contents/{FILE_PATH}"

payload = {
 "message": COMMIT_MESSAGE,
 "content": content,
 "branch": BRANCH
}

headers = {
 "Authorization": f"token {GITHUB_TOKEN}",
 "Accept": "application/vnd.github.v3+json"
}

response = requests.put(url, headers=headers, data=json.dumps(payload))
print(response.json())

```

Зміст файлу **function.json** має бути

```

{
 "scriptFile": "__init__.py",
 "bindings": [
 {
 "name": "timer",
 "type": "timerTrigger",
 "direction": "in",
 "schedule": "0 0 * * *"
 }
]
}

```

Зміст файлу **requirements.txt** має бути

```

pandas
requests
beautifulsoup4
openpyxl
lxml

```

Зміст файлу **host.json** має бути

```
{
 "version": "2.0"
}
```

Структура файлів, розміщених на диску користувача, готова. Починаємо налаштування **Azure AI Agent**, щоб він щодня о 00:00 запускав скрипт і оновлював Excel у репозиторії.

### Крок 1. Підготувати Azure Function для запуску скрипта

1. У **Azure Portal** → **Create a resource** → **Function App**:
  - **Subscription**: твоя підписка
  - **Resource group**: можна взяти ту ж, що для агента
  - **Name**: stock-parser-func (або як тобі зручно)
  - **Runtime stack**: Python 3.11
  - **Region**: близько до тебе (наприклад, Sweden Central)
2. Після створення Function App:
  - Відкрий вкладку **Functions** → **Add** → **Timer Trigger**
  - **Name**: RunDailyScript
  - **Schedule**: 0 0 \* \* \* (щодня о 00:00 UTC)
3. Деплой локальної структури у Function App:
  - Через **VS Code** → **Azure Functions extension** або **ZIP deploy**
  - Всі файли (pursingакс.py, requirements.txt, host.json, папка RunDailyScript/) мають бути завантажені
4. У **Configuration** → **Application settings** додай змінну:
  - **GITHUB\_TOKEN** → токен GitHub з правом запису у репозиторій AI
5. Переконайся, що **requirements.txt** встановлено під час деплою (Azure автоматично підніме середовище Python і поставить пакети).

### Крок 2. Перевірка Azure Function

1. Відкрий **Functions** → **RunDailyScript** → **Code + Test**
2. Тимчасово змінити стоп на щохвилинний \* \* \* \* \* для тесту
3. Запусти вручну через **Test/Run**

#### 4. Перевірка:

- Чи скрипт виконався
- Чи створився output.xlsx
- Завантаження у GitHub (повинно з'явитися в репозиторії A1)

#### 5. Після тесту поверніть сон на 0 0 \* \* \*

### Крок 3. Створення skill у Azure AI Agent

#### 1. Відкрий Azure AI Studio → Agents → твій агент → Build and Customize

#### 2. Натисни Add new skill → HTTP endpoint

#### 3. Налаштування skill:

- **Name:** RunStockParser
- **URL:** URL твоєї Azure Function (можна знайти у Function App → Overview → Function URL)
- **Method:** POST

#### 4. Додайте опис для агента:

«Щодня о 00:00 запускай цей HTTP endpoint, щоб виконати скрипт pusingакс.py для парсингу акцій і оновлення файлу output.xlsx у репозиторії A1.»

### Крок 4. Тестування агента

#### 1. Відкрийте Playground → твій агент

2. Виконайте skill RunStockParser вручну. Перевірте, що скрипт виконався у Function App і output.xlsx з'явився у репозиторії A1

### Крок 5. Автоматичний запуск щодня

- Завдяки **Timer Trigger** у **Function App**, скрипт автоматично виконуватиметься щодня о 00:00.
- Твій агент **може лише ініціювати виклик**, або ти можеш залишити його як manual skill для контролю.

### Крок 6. Поради та перевірки

- Якщо сайт змінює структуру HTML, перевір парсер (pusingакс.py) на помилки
- Логи Azure Function можна переглядати у **Monitor** → **Logs**
- Переконайся, що токен GitHub має права **write** на репозиторій A1

Як бачимо з цього опису дій, якщо виконати кроки 1 і 2 (створивши **Azure Function** з Timer Trigger, яка щодня о 00:00 запускає `pursingакс.py` і заливає Excel у GitHub), то AI Agent тобі вже не потрібен для автоматичного запуску.

Якщо наша ціль – щоденний деплой даних у GitHub без ручного втручання, достатньо тільки Azure Function з Timer Trigger.

### 3.5.3. Готові для використання агенти

**Azure OpenAI** немає бібліотеки з наперед створеними агентами типу “Марковіц-агент” чи “Шарп-агент”, які можна просто вибрати зі списку і одразу використовувати.

Готовий агент в Azure OpenAI – це шаблон конфігурації AI-асистента, який включає модель GPT та механізм підключення інструментів, але не містить спеціалізованих галузевих алгоритмів (наприклад, Марковіца) з коробки.

Що означає “готовий агент” у Azure? У Azure OpenAI Studio є **Templates** (шаблони). Ось приклади конфігурацій агента:

- Customer support
- Data assistant
- Q&A over documents
- API-calling assistant

Також є **Playground Agent mode**, де можна швидко створити агента без коду. Але це шаблони, а не спеціалізовані фінансові алгоритми.

Як підключитися до готових агентів у Azure OpenAI

#### Крок 1. Увійти в Azure OpenAI Studio

1. Відкрий: <https://oai.azure.com/>
2. Обери свій ресурс **Azure OpenAI** (той, що ти створив, де розгортав GPT-4o-mini).

#### Крок 2. Перейти до Agents

1. У меню ліворуч знайди **Agents** → **Build and customize**
2. Натисни **Create new agent**

### Крок 3. Вибір шаблону або навичок

1. Можна обрати:
  - **From template** → готові шаблони агента
  - **Custom** → створити агента під свої потреби
2. Готові шаблони часто мають навички (skills), наприклад:
  - **Data Collector** → збір даних із веб-сайтів
  - **Optimization Agent** → обчислення оптимальних рішень
  - **Forecasting Agent** → прогнозування на основі даних

### Крок 4. Налаштування агента

1. Дай назву та опис агента
2. Вибери модель GPT (наприклад, **gpt-4o-mini**)
3. Додай **skills** (готові або кастомні)
4. Можна підключити **HTTP endpoint** або **API**, якщо агент має працювати з твоїми даними

### Крок 5. Тестування агента

1. Відкрий **Playground** → **Agents**
2. Вибери агента
3. Надішли запит (наприклад, URL сайту для збору даних, або таблицю для прогнозування)
4. Отримай результат у Playgrounds або через API

### Конкретні приклади готових агентів

| Agent Template        | Призначення         | Як працює                                        | Підходить для                           |
|-----------------------|---------------------|--------------------------------------------------|-----------------------------------------|
| <b>Data Collector</b> | Збір даних з сайтів | Агент робить HTTP-запити і парсить текст/таблиці | Збір цін акцій, курсів, текстових даних |

| <b>Agent Template</b>              | <b>Призначення</b>                  | <b>Як працює</b>                                                                                                | <b>Підходить для</b>                            |
|------------------------------------|-------------------------------------|-----------------------------------------------------------------------------------------------------------------|-------------------------------------------------|
| <b>Forecasting Agent</b>           | Прогнозування                       | Використовує GPT для аналізу числових даних, трендів                                                            | Прогнозування цін, попиту, ринку                |
| <b>Optimization Agent</b>          | Оптимізація                         | Підбирає оптимальні стратегії, ресурси, параметри                                                               | Теорія ігор, оптимізація виробництва, стратегій |
| <b>Simulation Agent</b>            | Моделювання сценаріїв               | Створює сценарії та прогнозує наслідки                                                                          | Теорія ігор, фінансові моделі                   |
| <b>Text Analyzer / Scraper</b>     | Збір і обробка тексту               | Перетворює веб-текст у структуровані дані                                                                       | Новини, аналітика, ринкові звіти                |
| <b>Portfolio Optimizer</b>         | Оптимізація портфеля Марковіца      | Агент отримує дані по акціях (ціни, очікувані доходи, ковариаційна матриця) і повертає оптимальні ваги портфеля | Максимізація доходу при заданому ризику         |
| <b>Sharpe Ratio Analyzer</b>       | Розрахунок коефіцієнта Шарпа        | Агент обчислює співвідношення доходність/ризик для заданого портфеля                                            | Порівняння ефективності різних портфелів        |
| <b>Risk &amp; Return Simulator</b> | Симуляція портфеля                  | Використовує історичні дані та моделює доходність і волатильність                                               | Стратегічне планування інвестицій               |
| <b>Financial Forecasting Agent</b> | Прогнозування фінансових показників | Аналізує історичні дані, повертає тренди, очікувану доходність                                                  | Прогноз доходів і ризиків                       |

Наведемо декілька прикладів.

### **Приклад 1. Архітектура агента для задачі Марковіца**

#### **Мета:**

Агент:

- приймає Excel або список активів
- рахує оптимальні ваги (Mean-Variance)
- рахує Sharpe Ratio

- пояснює результат

### Загальна архітектура



#### Пояснення архітектури Компон

##### 1. Агент (Azure OpenAI)

- Модель: gpt-4o або gpt-4o-mini
- System Prompt (завдання агенту):

You are a financial portfolio optimization assistant.

Use the provided function to compute Markowitz optimal weights and Sharpe ratio.

- Додається Tool:  
optimize\_portfolio

##### 2. Python-функція (Azure Function або API)

Вона реально рахує.

```

import numpy as np
import pandas as pd
from scipy.optimize import minimize

```

```

def optimize_portfolio(returns, risk_free_rate=0.02):

 mean_returns = returns.mean()
 cov_matrix = returns.cov()
 num_assets = len(mean_returns)

 def portfolio_volatility(weights):
 return np.sqrt(weights.T @ cov_matrix @ weights)

 def negative_sharpe(weights):
 portfolio_return = weights @ mean_returns
 portfolio_vol = portfolio_volatility(weights)
 sharpe = (portfolio_return - risk_free_rate) / portfolio_vol
 return -sharpe

 constraints = ({'type': 'eq', 'fun': lambda x: np.sum(x) - 1})
 bounds = tuple((0,1) for _ in range(num_assets))
 initial = num_assets * [1./num_assets]

 result = minimize(negative_sharpe, initial,
 method='SLSQP',
 bounds=bounds,
 constraints=constraints)

 return result.x

```

### 3. Як працює система (flow)

#### 1. Користувач:

Оптимізує портфель для AAPL, MSFT, GOOG

#### 2. Агент:

- запитує дані
- викликає функцію
- отримує ваги
- пояснює результат

#### 3. Відповідь:

- оптимальні ваги
- очікувана дохідність
- ризик

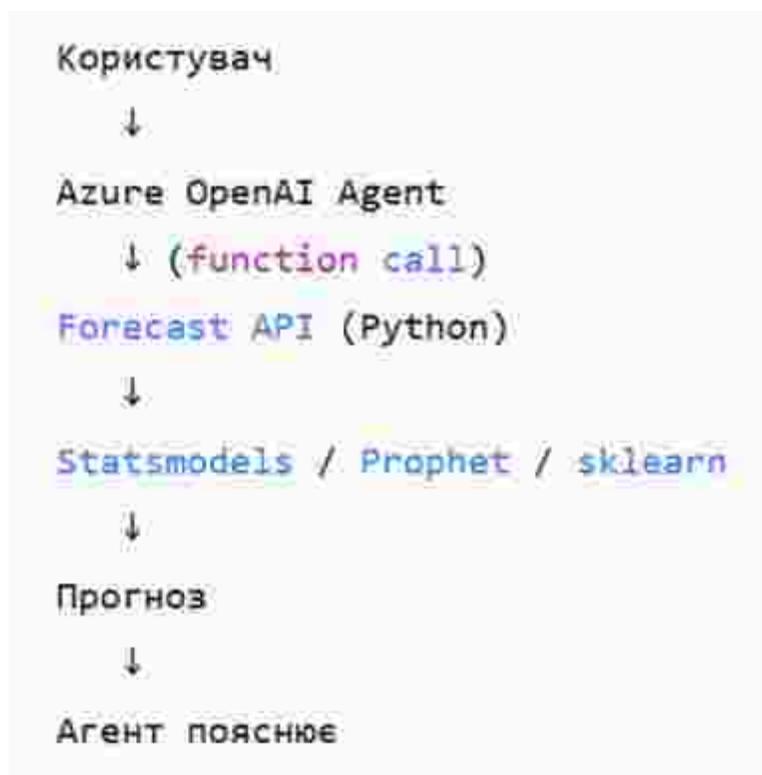
- Sharpe ratio
- текстове пояснення

## Приклад 2. Реальний Forecasting Agent з Python backend

Агент прогнозує часовий ряд:

- ARIMA
- Prophet
- LSTM
- або регресію

### Архітектура



### Python API приклад (ARIMA)

```
from statsmodels.tsa.arima.model import ARIMA
import pandas as pd

def forecast_series(data, steps=5):

 model = ARIMA(data, order=(1,1,1))
 model_fit = model.fit()

 forecast = model_fit.forecast(steps=steps)
```

```
return forecast.tolist()
```

### **Що робить агент**

System Prompt:

You are a financial forecasting assistant.

Use the forecast\_series tool when time series data is provided.

### **Приклад роботи**

Користувач:

Прогнозуй ціну AAPL на 10 днів вперед

Агент:

- викликає forecast\_series
- отримує масив прогнозу
- повертає:
  - таблицю
  - пояснення
  - ризики

### **Пояснення:**

- GPT не рахує оптимізацію
- GPT не тренує ARIMA
- GPT керує процесом
- Python рахує математику

## **3.6. Інші сервіси Microsoft Azure**

Всі ці сервіси можна знайти за кнопкою All Services.

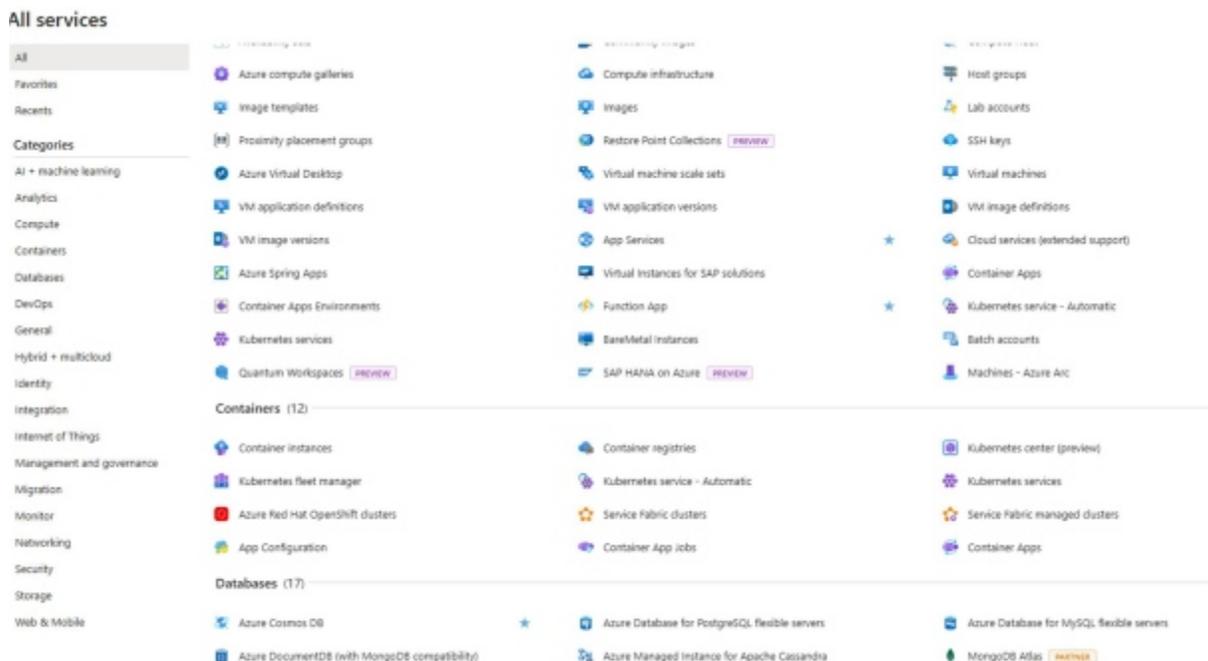
У All Services або в Azure OpenAI Studio під інструментами (Tools/Skills) маються на увазі готові можливості, які можна підключити до агента, наприклад

| Інструмент           | Що робить                                                     | Приклад використання                              |
|----------------------|---------------------------------------------------------------|---------------------------------------------------|
| HTTP REST API        | Дозволяє агенту робити запити до зовнішніх або внутрішніх API | Парсинг сайту, отримання цін акцій                |
| Azure Function       | Виклик Python-коду на сервері                                 | Оптимізація портфеля Марковіца, розрахунок Sharpe |
| Files / Blob Storage | Зберігання та читання файлів                                  | CSV або Excel з історичними даними                |
| Database             | Підключення до SQL/NoSQL                                      | Зберігання історії цін, фінансових даних          |

Чи “готові” вони до використання?

- Так, вони технічно готові – можна одразу підключити до агента
- Ні, вони не вирішують вашу конкретну задачу самі по собі:
  - HTTP API не буде сам збирати ціни акцій з сайту без правил
  - Python-функція не буде сама рахувати Марковіца без твого коду
  - GPT не буде прогнозувати точні числа без даних та алгоритму

Тобто інструменти — це будівельні блоки, а логіку додає користувач.



Щоб користуватися інструментами з агентом, спочатку необхідно створити дані у наступному порядку:

### 1. Відкрити Azure ML Studio. Для цього:

1. Зайти в **Azure Portal** → All Services → знайди **Machine Learning**.
2. Відкрити **Workspace**, який було створено.
3. У верхньому меню знайди і натисни **Launch Studio**.

### 2. Перейти в розділ Dataset

1. Ліворуч у меню знайти **Data** → натиснути **Datasets**.
2. У верхньому правому куті натиснути + **Create** → **Dataset** → **From local files**

### 3. Вибір файлу

1. З'явиться вікно **Create Dataset**.
2. Клікніть **Browse** → оберіть файл **Excel (.xlsx)** на своєму комп'ютері.
3. Натисніть **Next**.

### 4. Налаштування Dataset

1. Поле **Name** → введіть назву, наприклад: `stock_prices`
2. Поле **Description** → за бажанням, наприклад: "Daily prices for portfolio"
3. Тип даних: оберіть **Tabular**
4. Перевірте, що **First row as header** увімкнено, якщо перший рядок – заголовки стовпців
5. Роздільник: Excel зазвичай сам визначає
6. Натисніть **Next**

### 5. Перевірка даних (Preview)

1. ML Studio покаже **фрагмент таблиці** (перші 10 рядків)
  - Перевір, що колонки і типи даних правильні. Якщо щось не так – редагуйте у попередньому кроці
2. Натисніть **Create**

### 6. Dataset готовий

- Тепер Excel файл з'явився у **Datasets** → **My Datasets**
- Він зберігається у вашому **Workspace** як актив
- Ним можна використовувати у: **Designer** (перетягування блоків ML pipeline); **Notebooks** (Python код); **Automated ML**.

**Також існує готовий сервіс в Azure**, що дозволяє будувати різні моделі без коду. Такий сервіс називається Azure Automated Machine Learning (AutoML).

AutoML – це саме той інструмент, який можна використовувати, наприклад, для прогнозування без писання Python-коду.

Порядок доступу:

1. У **Azure Portal** → **All services**
2. Знайдіть **Machine Learning**
3. Запустіть **Azure Machine Learning Studio**
4. У меню ліворуч оберіть **Automated ML** – це і є AutoML інтерфейс.

Як завантажити дані для AutoML? Ось назва кнопок і порядок дій:

1. У ML Studio оберіть **Datasets** → + **Create**
2. Оберіть **From local files**
3. Завантажте свій Excel або CSV
4. Далі створіть Dataset (тип **Tabular**) і натисніть **Create**

Тепер у вас є Dataset, який можна використовувати для AutoML.

**Щоб зробити прогноз у AutoML без коду, виконайте наступні дії:**

Крок 1: Перейти до AutoML. У меню ліворуч знайдіть та натисніть Automated ML

Крок 2: Створити новий AutoML Experiment. Натисніть кнопку + New Automated ML run

Крок 3: Оберіть Dataset Excel датасет який створили раніше і де знаходяться дані для розрахунку.

Крок 4: Налаштування задачі

У вікні **Task type** Оберіть **Time series forecasting**, якщо дані часові;  
**Regression**, якщо хочете прогнозувати числові значення.

Крок 5: Налаштуйте параметри. Вкажіть:

| Параметр         | Що це                              |
|------------------|------------------------------------|
| Target Column    | Стовпець, який прогнозуємо         |
| Frequency        | Як часто дані (денні, щомісяця)    |
| Forecast horizon | Скільки кроків вперед прогнозувати |

Крок 6: Запустити AutoML

Після того як усе налаштовано – натисніть **Run. AutoML** сам обере найкращу модель, перевірить багато алгоритмів, згенерує графіки і метрики, і покаже прогнозовані значення

Після завершення AutoML ви побачите:

**Performance leaderboard** – список моделей з точністю

**Metrics** – оцінка прогнозу

**Forecast plot** – графік прогнозу

**Leader model** – найкраща модель

Ці результати доступні в **UI Machine Learning Studio**.

**Як вивантажити результат**

Після тренування ви можете зберегти прогнозовані значення як CSV, деплоїти модель на endpoint, потім викликати її через API.

## 2.7. Індивідуальне завдання №3

### Елементи роботи з Azure

**Критерії оцінювання:** це завдання оцінюється у 5 балів за національною шкалою за декількома завданнями окремо. За кожну помилку знімається 0,1 бали.

Потім оцінка перераховується за 100-бальною системою згідно існуючого положення.

**Мета роботи:** Набути навичок у налаштуванні Azure та використанні **Machine Learning Studio** для розрахунків.

**Завдання 1:** Налаштування Azure/

1. Увійти в Azure Portal
2. Створити Azure ML Workspace
3. Запустити Azure ML Studio
4. Створити Notebook
5. Виконати Python-код для завантаження та перегляду даних

**Завдання 2:** Збір, очищення та підготовка даних

1. Завантажити навчальний CSV-датасет
2. Дослідити структуру даних
3. Виявити пропущені значення
4. Виконати очищення та підготовку
5. Зберегти підготовлений датасет

**Завдання 3:** Машинне навчання та AutoML

1. Створити AutoML-експеримент
2. Обрати тип задачі (регресія або класифікація)
3. Налаштувати параметри навчання
4. Запустити експеримент
5. Проаналізувати результати

**Контрольні запитання**

1. У чому полягає роль Azure Machine Learning у сучасних системах ШІ?
2. Як Python використовується для доступу до сервісів ШІ в Azure?
3. Яке призначення API-ключів і як забезпечується їх безпека?

4. Чому якість даних є критичною для машинного навчання?
5. Опишіть повний цикл підготовки даних у Azure ML.
6. Порівняйте класичне машинне навчання та AutoML.
7. У яких випадках AutoML є доцільним, а в яких – ні?
8. Чим глибинне навчання відрізняється від традиційного ML?
9. Які переваги використання попередньо навчених моделей?
10. Що таке інтелектуальні сервіси Azure?
11. Як Azure ML застосовується для прогнозування?
12. Які метрики якості моделей використовуються в Azure?
13. Як ШІ може допомагати в оптимізації управлінських рішень?
14. Яку роль відіграє теорія ігор у штучному інтелекті?
15. Які обмеження безкоштовного акаунту Azure і як їх враховувати в навчанні?

*Ретельне освоєння системи Azure дозволить студентам використовувати практично необмежені обчислювальні потужності з управління розрахунками.*

## Розділ 4.

# ПАКЕТИ ПРОГНОЗУВАННЯ AUTOML ТА MLFLOW

*Вивчивши матеріали цього розділу студенти опанують методи роботи з пакетами Azure AutoML та MLflow (в Azure ML).*

З цього розділі мова про два різні інструменти в екосистемі Azure, яку ми вивчали у розділі 3 цього посібника:

- **Azure AutoML**
- **MLflow (в Azure ML)**

Вони не конкуренти. Вони вирішують різні задачі.

### 4.1. Порівняння можливостей двох систем прогнозування

**Azure AutoML (Automated Machine Learning)** – це готовий сервіс автоматичного підбору і тренування моделей без коду.

Принцип роботи наступний. Ви вводите дані, вказуєте, що потрібно прогнозувати, обираєте тип задачі (forecasting – прогнозування, regression – побудова регресійної моделі).

Після цього AutoML підбирає алгоритми, перебирає параметри, обирає найкращу модель, а потім показує метрики, тобто результати розрахунку.

Використання цієї підсистеми обмежується задачами прогнозу часових рядів, побудовою регресійних моделей, класифікацією даних, прогнозуванням продажів товару, прогнозуванням фінансових показників.

Роботу цієї підсистеми можна описати простими совами так:

1. Завантажуєте dataset.
2. Обираєте, наприклад Forecasting.
3. Вказуєте target column, тобто, за якою колонкою вхідної таблиці робити прогнозування.
4. Натискаєте Run.
5. Через деякий час бачите прогноз і графіки.

Перевагою цієї підсистеми є:

- ✓ Не потрібно програмування
- ✓ Підходить для бізнес-користувачів
- ✓ Все через UI
- ✓ Автоматично тестує багато моделей

Обмеженнями підсистеми є:

- ✗ Менше контролю.
- ✗ Не підходить для складних кастомних алгоритмів.
- ✗ Це “чорна скринька”, тобто, ви не знаєте, яким чином, за допомогою якого алгоритму були отримані ці результати.

**MLflow – це система керування ML-експериментами.**

Вона не будує модель, не підбирає алгоритм автоматично, не дає кнопку “зробити прогноз”.

MLflow – це інструмент для відстеження експериментів, збереження параметрів, збереження моделей, перевірки різних версій, порівняння результатів.

Тобто, якщо AutoML “робить модель”, то MLflow “записує і контролює процес створення моделі”.

Ця підсистема використовується у випадках, коли користувач сам пише ML код, тестує багато варіантів моделей, коли потрібно керувати версіями, коли працює команда дослідників.

### **Як працює підсистема:**

Дослідник запускає експеримент, а MLflow:

- записує параметри;
- записує метрики;
- зберігає модель;
- дозволяє порівнювати результати.

### **Головна різниця цих підсистем в AZURE**

| <b>Критерій</b>             | <b>AutoML</b>    | <b>MLflow</b>   |
|-----------------------------|------------------|-----------------|
| Потрібен код                | Ні               | Так             |
| Автоматичний підбір моделей | Так              | Ні              |
| Контроль над алгоритмом     | Обмежений        | Повний          |
| Для кого                    | Бізнес-аналітики | Data Scientists |
| Прогноз “з кнопки”          | Так              | Ні              |
| Управління експериментами   | Частково         | Повністю        |

Що в них спільного:

- Працюють всередині Azure Machine Learning
- Працюють з тими самими даними
- Підтримують розгортання моделей
- Можуть використовуватися разом

## 4.2. Azure AutoML

**Azure Automated Machine Learning (AutoML)** – це сервіс платформи **Azure Machine Learning**, що забезпечує автоматизований підбір, навчання та оцінювання моделей машинного навчання без необхідності програмування.

AutoML підтримує такі типи задач:

- регресія
- класифікація
- прогнозування часових рядів (Time Series Forecasting)

Система автоматично:

- виконує попередню обробку даних
- підбирає алгоритми
- оптимізує гіперпараметри
- оцінює моделі
- обирає найкращу модель

## 2. Доступ до AutoML

### 2.1 Необхідні умови

Для використання AutoML необхідно:

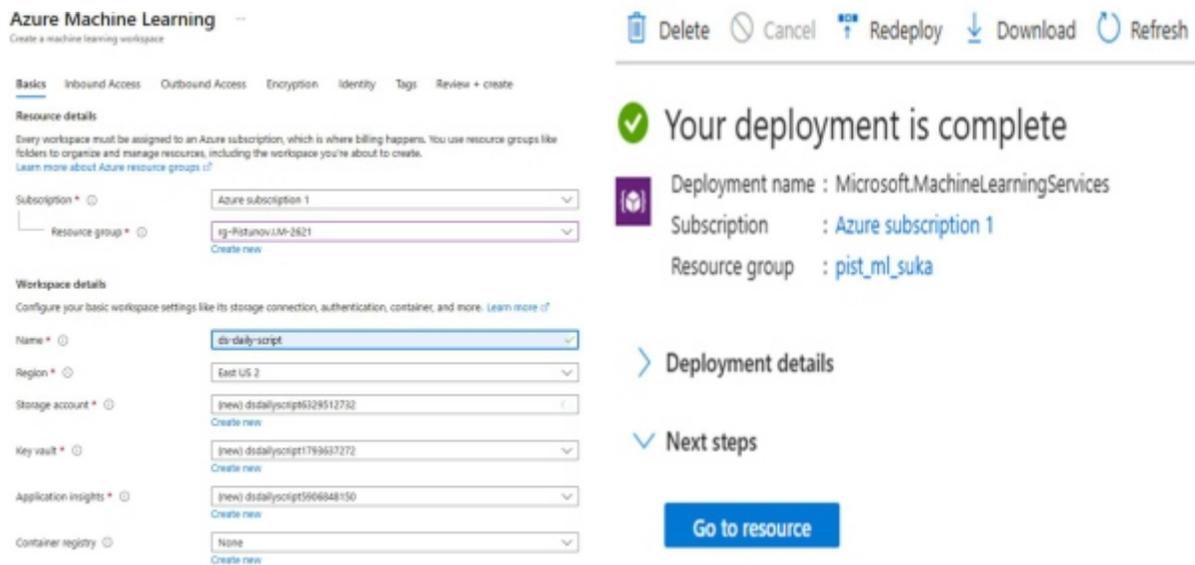
1. Обліковий запис Microsoft
2. Підписка Azure
3. Створений ресурс **Azure Machine Learning Workspace**

### 2.2 Шлях доступу

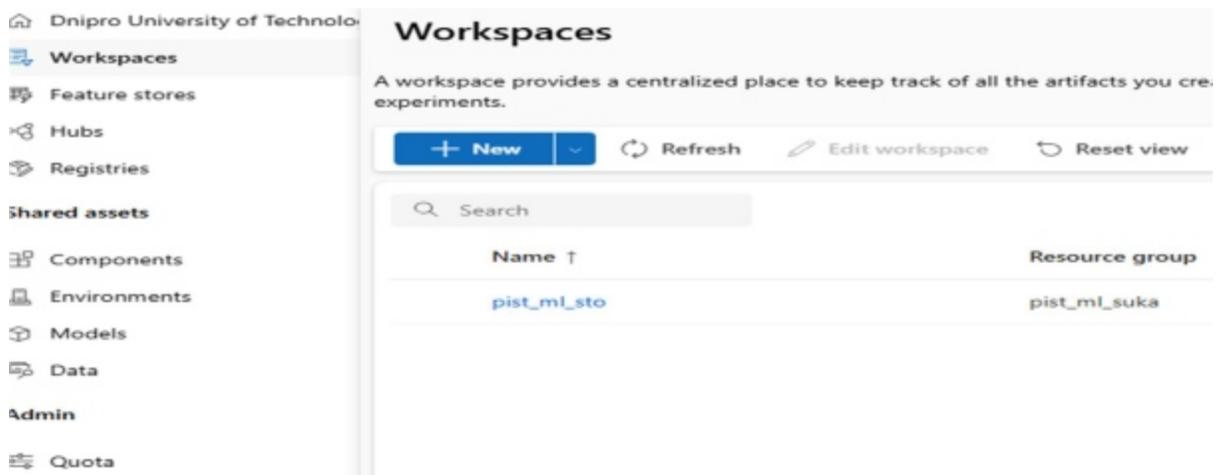
1. Увійти до Azure Portal <https://portal.azure.com>
2. Увімкніть VPN з адресою десь у Європі.
3. У верхньому пошуку введи Azure Machine Learning
4. Обери сервіс Azure Machine Learning
5. Створити Workspace можна через натискання кнопки в головному меню + Create. У пункті, що відкриється, оберіть Azure Machine

Learning і заповніть анкету. В ній запишіть нове ім'я проєкту та ваше теж нове, бо в Azure, ваша первинна реєстрація використовується тільки для входу в систему і перевірки, чи є на вашому рахунку гроші.

6. Далі у пунктах **Inbound Access** та **Outbound Access** та **Encryption** та **Identity** та **Tags** натисніть **Next**.
7. Коли з'явиться список всіх ваших налаштувань, натисніть **Create** і ви побачите, що система зареєструвала ваш проєкт.



8. Тепер відкрийте нове вікно браузера і введіть адресу <https://ml.azure.com/>
9. Натисніть синій напис **View all workspace**. Відкриється саме ваш робочий простір.



| Name ↑      | Resource group |
|-------------|----------------|
| pist_ml_sto | pist_ml_suka   |

## 10. Підготовка та створення бази даних

У контексті Azure AutoML база даних – це **Dataset (набір даних)**.

AutoML підтримує:

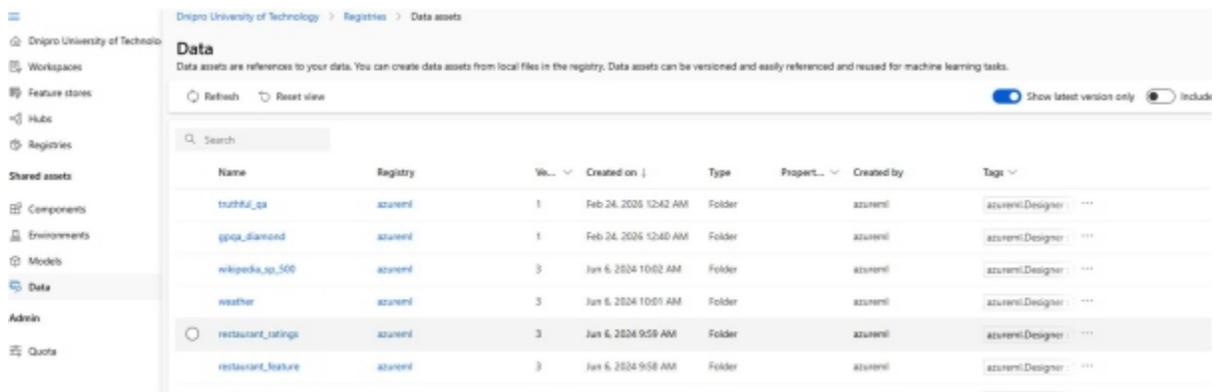
- локальні файли (CSV, Excel)
- Azure Blob Storage
- Azure Data Lake
- SQL-бази
- інші хмарні джерела

Послідовність дій створення бази даних:

- У Azure ML Studio обрати розділ **Data**
- Натиснути + **Create**
- Обрати **From local files** або інше джерело
- Завантажити файл
- Вказати:
  - назву набору даних
  - тип даних (Tabular)
  - наявність заголовків
- Натиснути **Create**

Після цього Dataset зберігається як актив Workspace.

11. Як ввести файл з даними формату CSV/Excel. Якщо ви зайшли в Azure за навчальною підпискою, дані ввести ви не можете, тому скористайтеся готовими даними, що подані у цьому вікні.



Якщо ви тільки навчаєтеся, варто скористатися можливостями готових наборів даних. У новому інтерфейсі Azure ML Studio серед наявних Dataset зазвичай є Dataset Sampler – це компонент, який дозволяє взяти частину наявного набору даних і підготувати її для подальшої роботи з ML pipeline.

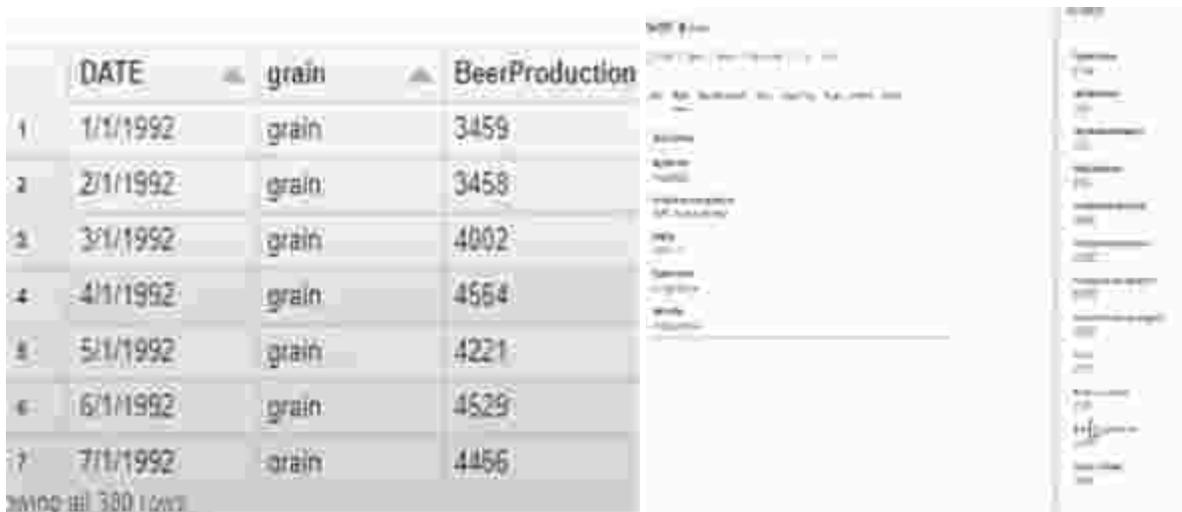
### Прогнозування через AutoML

Для створення нового запуску у розділі **Automated ML** натиснути + **New Automated ML run**. Потім обрати створений Dataset

Користувач визначає:

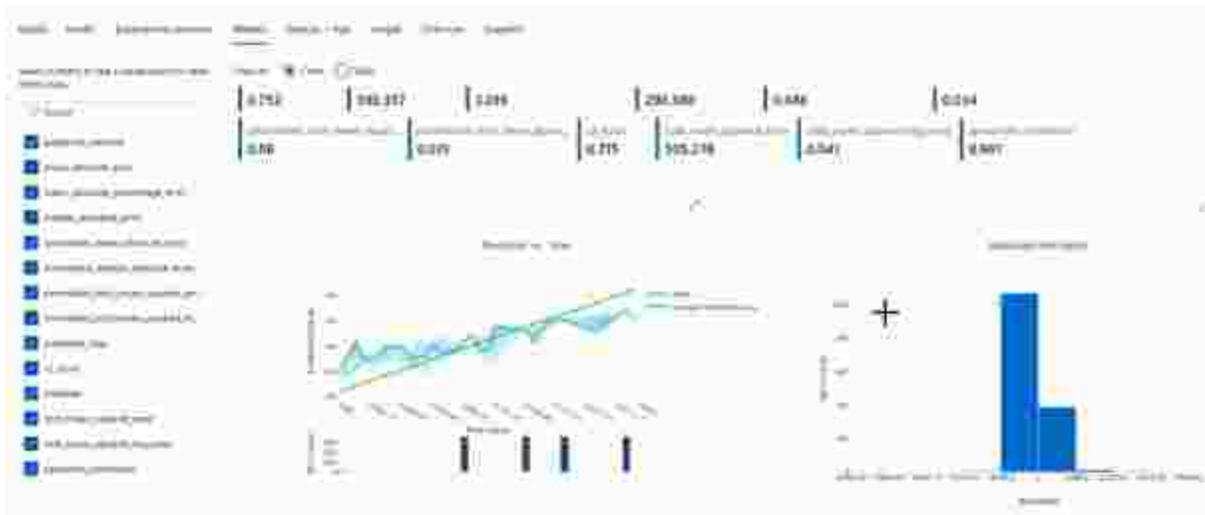
- **Task type:**
  - Regression
  - Classification
  - Time Series Forecasting
- **Target column** – змінна, яку потрібно прогнозувати
  - Для часових рядів:
    - колонку дати
    - частоту даних (денна, місячна тощо)
    - горизонт прогнозу

Наприклад, нехай маємо такі дані і створимо dataset



|   | DATE     | grain | BeerProduction |
|---|----------|-------|----------------|
| 1 | 1/1/1992 | grain | 3459           |
| 2 | 2/1/1992 | grain | 3458           |
| 3 | 3/1/1992 | grain | 4002           |
| 4 | 4/1/1992 | grain | 4554           |
| 5 | 5/1/1992 | grain | 4221           |
| 6 | 6/1/1992 | grain | 4529           |
| 7 | 7/1/1992 | grain | 4456           |

Натискання на кнопку Metrix, одразу дає всі характеристики цього часового ряду.



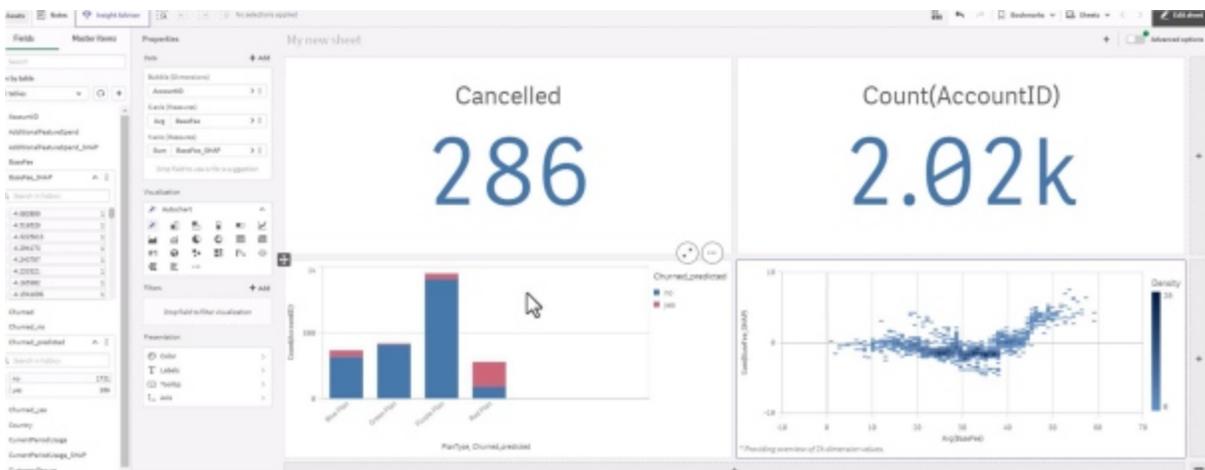
Потрібно обрати Existing compute cluster або Створити новий Compute Instance. Це середовище, в якому буде виконуватися навчання.

Після завершення налаштувань натискається **Run** і AutoML виконує:

- автоматичний перебір алгоритмів
- оптимізацію параметрів
- крос-валідацію
- оцінювання метрик

Після завершення виконання користувач отримує:

Таблицю з назвами алгоритмів, значеннями метрик (RMSE, MAE,  $R^2$  тощо), часом навчання, графік фактичних і прогнозованих значень, залишки, аналіз похибок.



AutoML автоматично визначає модель-лідера.

Можливі подальші дії: деплой як endpoint, завантаження результатів, експорт прогнозних значень

Результати зберігаються у:

- Workspace Azure ML
- розділі Experiments
- розділі Models (для найкращої моделі)
- Outputs (для прогнозних таблиць)

Прогнозні значення можна переглянути в UI, експортувати у CSV, використовувати через API після деплою.

**Регресія через AutoML** виконується, як і у попередньому випадку, через завантаження таблиці (CSV, Excel, база даних), в якій має бути один стовпець – цільова змінна (target), а інші стовпці автоматично сприймаються як фактори (features)

В таблиці не має бути пропусків, мають бути присутні числові або категоріальні змінні. Розмір вибірки бажано більше 100 спостережень.

Після створення нового проекту необхідно обрати тип задачі **Regression**, завантажити файл, вказати target column.

AutoML автоматично тестує такі алгоритми:

- Linear Regression;
- Random Forest;
- Gradient Boosting;
- XGBoost;
- Neural Networks.

а потім видає параметри найкращого результату.

Час навчання (може бути 30–60 хв). Метрика якості: RMSE, MAE,  $R^2$ .

Перший і останній тип регресії нам добре знайомий, розберемо інші типи.

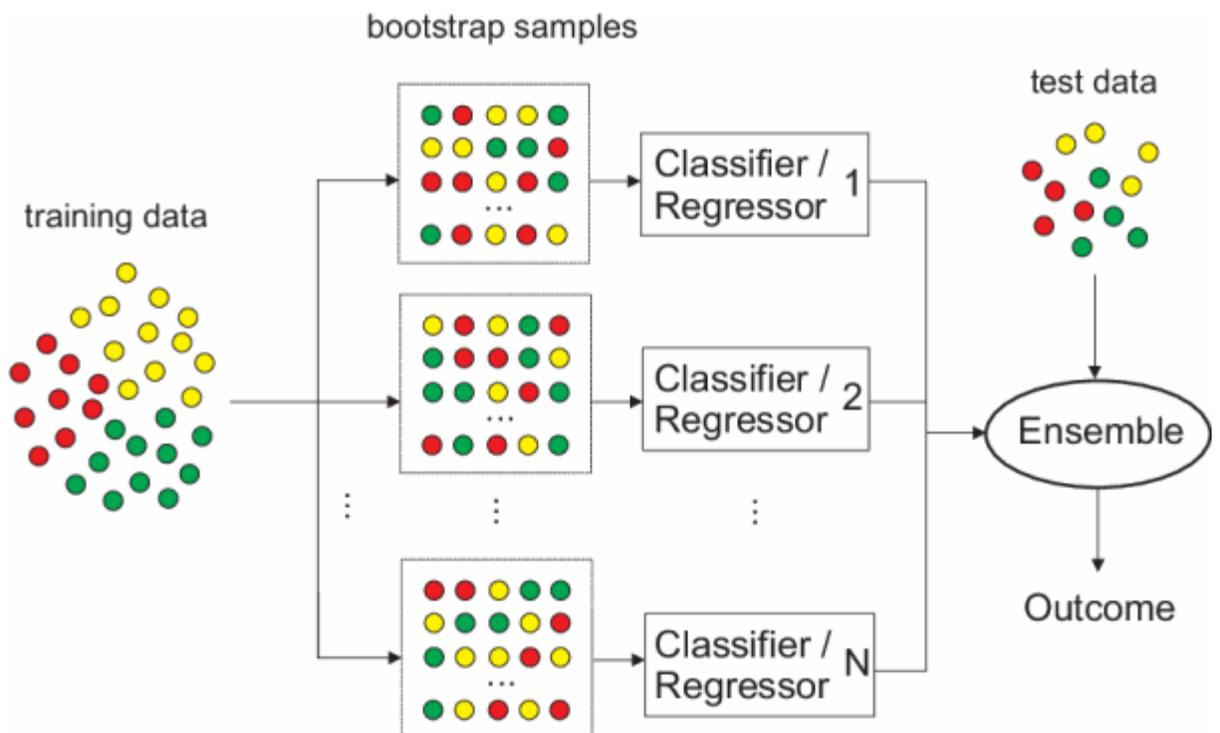
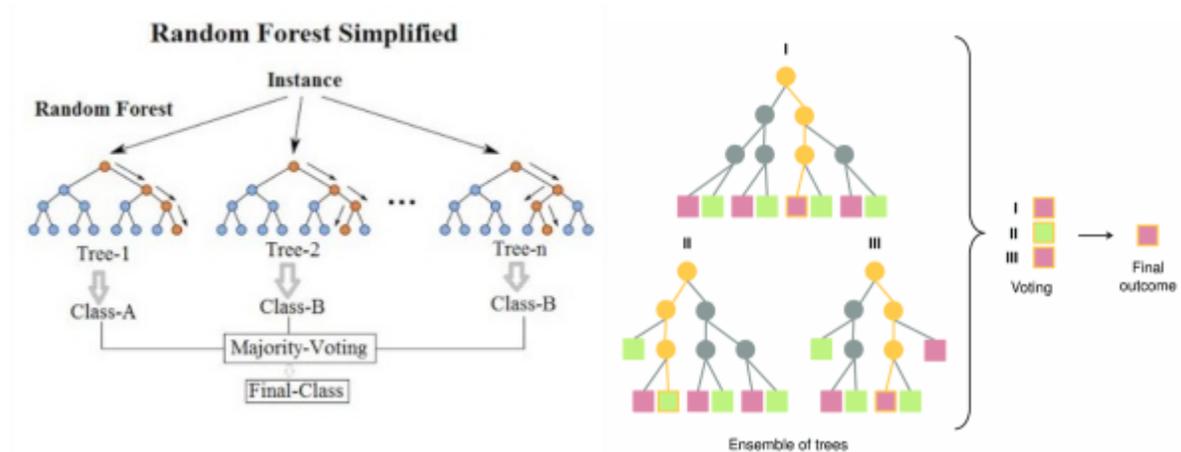
## Random Forest

Ідея алгоритму: ансамбль багатьох дерев рішень.

Алгоритм працює наступним чином, спочатку будується багато дерев на різних підвбірках даних (bootstrap). Потім кожне дерево робить свій прогноз. Для регресії береться середнє значення прогнозів.

**Плюси:** стабільність, простота, добре працює «з коробки».

**Мінуси:** велика кількість дерев, отже повільна робота.



## Gradient Boosting

**Ідея:** послідовне покращення помилок попередніх моделей.

**Як працює:**

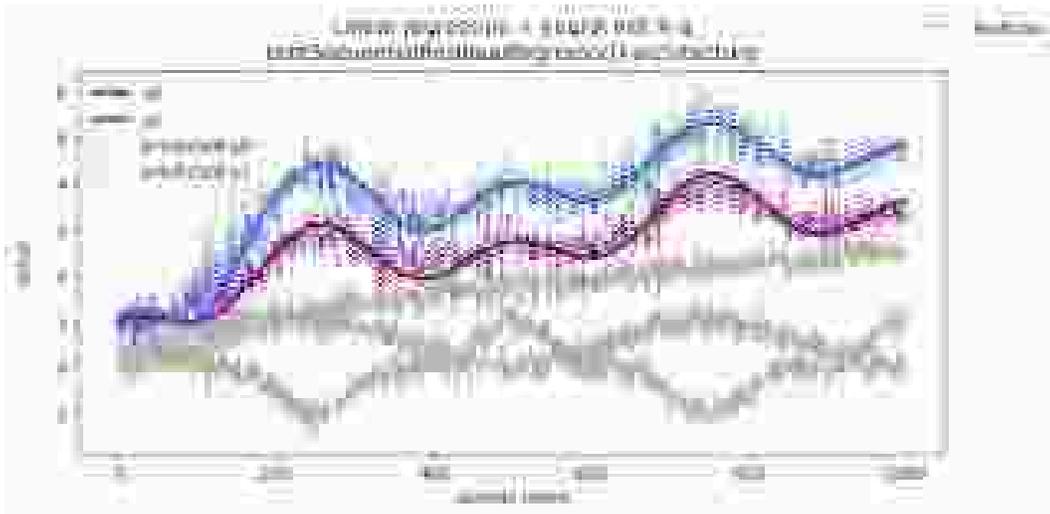
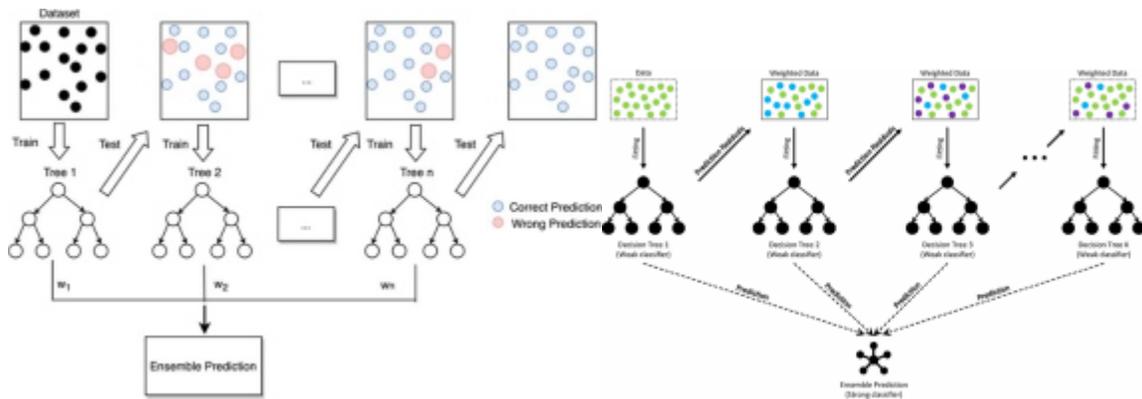
- Перше дерево робить прогноз.
- Друге дерево навчається на помилках першого.
- Кожне наступне дерево мінімізує залишкову помилку.
- Підсумковий прогноз — сума прогнозів усіх дерев.

**Особливості:**

- Моделі будуються послідовно.
- Менше зміщення (bias), висока точність.

**Плюси:** висока якість прогнозу.

**Мінуси:** чутливий до параметрів, може перенавчатися.



## XGBoost

Повна назва: **eXtreme Gradient Boosting**

Реалізація градієнтного бустингу з оптимізацією швидкості та регуляризації.

### Відмінності від класичного Gradient Boosting:

- Регуляризація (L1, L2)
- Паралельні обчислення
- Обробка пропущених значень
- Вбудований контроль перенавчання

**Плюси:** дуже висока точність, швидкість, популярний у Kaggle.

**Мінуси:** складніший у налаштуванні.

### Squared Error Loss:

$$loss = \sum_{i=0}^n (y_i - \hat{y})^2$$

$$loss = \sum_{i=0}^n (y_i - (p_i^0 + O_{value}))^2$$

where:

$p_i^0$  = sum of previous trees

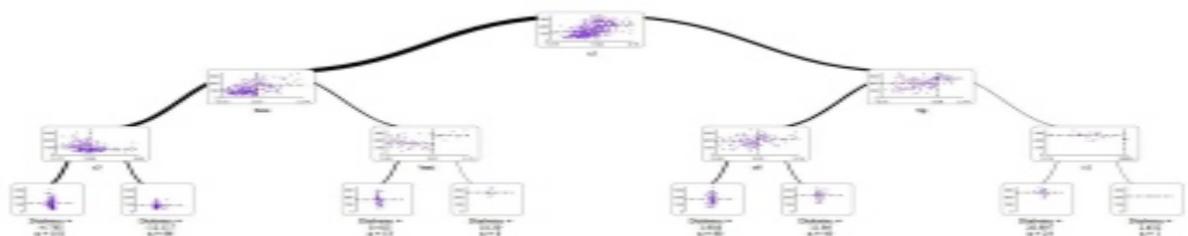
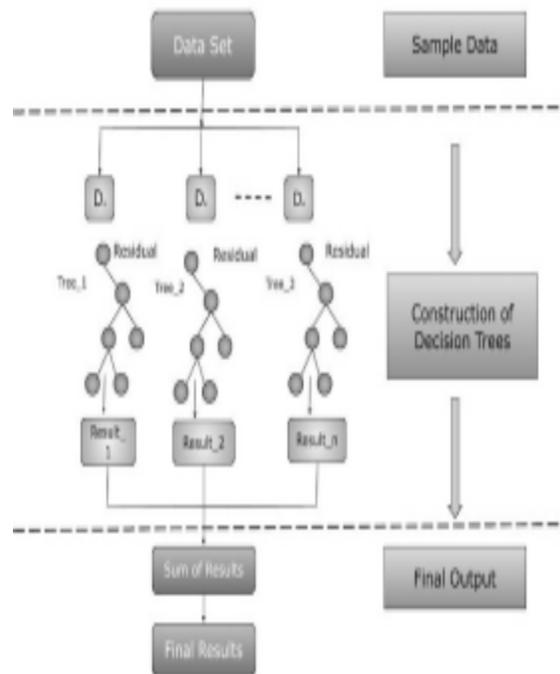
$O_{value}$  = leaf values

### L2 Regularization

$$loss = \left( \sum_{i=0}^n (y_i - (p_i^0 + O_{value}))^2 \right) + \lambda O_{value}^2$$

where:

$\lambda O_{value}^2$  = lambda, L2 Parameter



### Коротке порівняння

| Алгоритм          | Тип                    | Суть                               | Коли використовувати          |
|-------------------|------------------------|------------------------------------|-------------------------------|
| Random Forest     | Bagging                | Усереднення багатьох дерев         | Базова надійна модель         |
| Gradient Boosting | Boosting               | Послідовне виправлення помилок     | Коли потрібна висока точність |
| XGBoost           | Оптимізований Boosting | Швидкий та регуляризований бустинг | Складні задачі, великі дані   |

Стисло нагадаємо порядок побудови регресійної моделі в Azure Machine Learning:

1. Create → Automated ML
2. Select dataset
3. Task type → Regression
4. Choose target column
5. Start training
6. Compare models
7. Deploy best model

Результати створення моделі можна експортувати у REST API, Python-код, збереження як pickle / ONNX або розгортання у хмарі.

Експорт моделі потрібен для того, щоб не розбиратися зі складними формулами, а мати одразу готовий інструмент для подальшого використання.

AutoML в Azure автоматично перебирає десятки моделей регресії, обирає найкращу за метриками та дозволяє розгорнути її як веб-сервіс без написання складного коду.

## Класифікація даних через AutoML

Класифікація – це тип задачі машинного навчання, коли потрібно віднести об’єкт до однієї або кількох категорій (label).

Наприклад:

- Виявлення спаму → клас “spam” / “not spam”
- Діагностика хвороби → клас “здоровий” / “хворий”
- Сегментація клієнтів → “VIP”, “Regular”, “New”

**AutoML** автоматично:

1. Аналізує дані.
2. Розпізнає числові та категоріальні ознаки.
3. Вибирає алгоритми класифікації:
  - Logistic Regression.
  - Random Forest.
  - Gradient Boosting / XGBoost / LightGBM.
  - Neural Networks.
4. Перевіряє моделі через крос-валідацію.
5. Вибирає найкращу модель за обраною метрикою (accuracy, F1, AUC).

Для роботи цієї функції необхідні:

1. Таблиця (CSV, Excel, база даних)
2. Цільова змінна – категоріальна (label)
3. Фактори – числові або категоріальні
4. Перевірка: відсутність критичних пропусків, правильний тип змінних

Приклад таких даних:

**age income gender buys\_product**

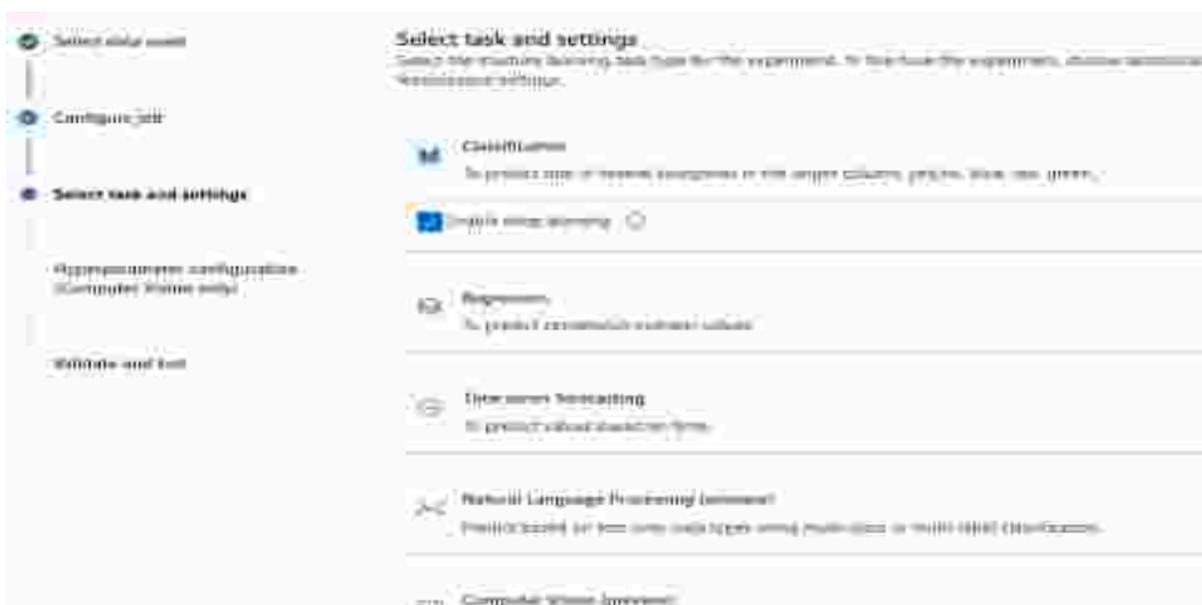
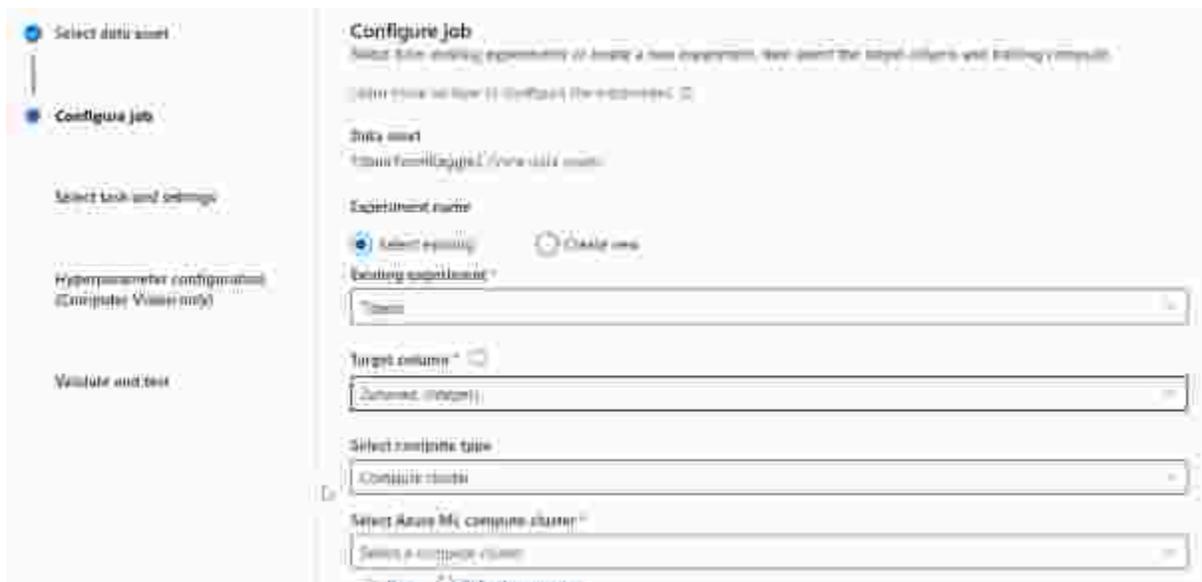
25 50000 M yes

40 80000 F no

Цільова змінна – buys\_product

## Як виконати класифікацію в Azure AutoML

1. Create → Automated ML
2. Select dataset
3. Task type → Classification
4. Target column → buys\_product
5. Налаштування:
  - Primary metric – Accuracy, AUC, F1
  - Training time – 30–60 хв
  - Validation type – k-fold cross validation



### Additional configurations

Primary metric

- Select data asset
- Configure job
- Select task and settings
- Hyperparameter configuration (Compute Vision only)

Select the validation and test type (You can choose a validation type and select a data asset to currently provide features)

Validation type

- Auto
- K-fold cross validation
- Monte Carlo cross validation
- Train-validation-test
- Dev-validation-test

| <input checked="" type="checkbox"/> | Column name      | Feature type | Impute with | Data example |
|-------------------------------------|------------------|--------------|-------------|--------------|
| <input checked="" type="checkbox"/> | zero_13          | Auto         | Auto        | 0, 0, 0      |
| <input checked="" type="checkbox"/> | zero_14          | Auto         | Auto        | 0, 0, 0      |
| <input checked="" type="checkbox"/> | Pclass           | Auto         | Auto        | 3, 1, 3      |
| <input checked="" type="checkbox"/> | zero_15          | Auto         | Auto        | 0, 0, 0      |
| <input checked="" type="checkbox"/> | zero_16          | Auto         | Auto        | 0, 0, 0      |
| <input checked="" type="checkbox"/> | Embarked         | Auto         | Auto        | 2, 0, 2      |
| <input checked="" type="checkbox"/> | zero_17          | Auto         | Auto        | 0, 0, 0      |
| <input checked="" type="checkbox"/> | zero_18, zero_19 | Auto         | Auto        | 0, 0, 0      |

Azure перебирає всі релевантні алгоритми, оптимізує гіперпараметри, ранжує моделі на leaderboard.

Leaderboard в AutoML – це таблиця, яка показує результати всіх моделей, які були автоматично згенеровані під час експерименту AutoML. Вона допомагає швидко порівняти ефективність моделей і вибрати найкращу для подальшого використання.

Типова таблиця leaderboard може включати:

- Назва моделі – наприклад, LightGBM, XGBoost, LogisticRegression.
- Основний показник якості (Primary metric) – той, що ви обрали для оцінки моделей (наприклад, Accuracy, AUC, RMSE).
- Додаткові метрики – Precision, Recall, F1-score, MSE,  $R^2$ , залежно від типу задачі.
- Конфігурація моделі – параметри, гіперпараметри, особливості обробки даних.
- Час тренування – скільки часу витрачено на навчання моделі.
- Rank – місце моделі в рейтингу за основним показником.

Після завершення AutoML експерименту відкриваєте **Leaderboard** у веб-порталі Azure ML. Ви можете Сортувати моделі за потрібною метрикою (наприклад, від найвищої до найнижчої точності). Або можете клікнути на конкретну модель, щоб завантажити її для деплою, проаналізувати її гіперпараметри, переглянути метрики та графіки.

AutoML показує:

- Метрики: Accuracy, Precision, Recall, F1-score, AUC
- Feature importance – які фактори найбільше впливають на класифікацію
- Confusion matrix – наскільки добре модель розрізняє класи

Деплой моделі:

- Обрати Best model.
- Deploy – REST API / Python scoring script.
- Можна робити real-time прогнозування або пакетне.

Категоріальні змінні можна залишати як текст – AutoML кодує їх автоматично

Якщо ви маєте велику кількість класів – краще перевіряти баланс класів.

Для інтерпретації важливих факторів – дивитися feature importance або SHAP values.

### 4.3. MLflow

**MLflow** – це відкрита платформа керування життєвим циклом моделей машинного навчання, інтегрована в середовище **Azure Machine Learning**.

На відміну від AutoML, MLflow **не створює модель автоматично**, а забезпечує:

- реєстрацію експериментів
- фіксацію параметрів
- збереження метрик
- версіонування моделей
- управління розгортанням

Таким чином, MLflow – це інструмент контролю та управління процесом побудови моделей, а не система автоматичного прогнозування.

MLflow використовується для:

- документування експериментів
- порівняння різних моделей
- відтворюваності результатів
- збереження артефактів (файлів, графіків, моделей)
- реєстрації моделей для подальшого деплою

Особливо актуальний для:

- наукових досліджень
- фінансового моделювання
- складних оптимізаційних задач
- командної роботи Data Science

Необхідні умови для доступу до MLflow в Azure:

1. Підписка Azure
2. Створений **Azure Machine Learning Workspace**
3. Наявність Compute Resource (для запуску експериментів)

Шлях доступу

1. Увійти в **Azure Portal**
2. Обрати **Machine Learning**
3. Відкрити відповідний Workspace
4. Натиснути **Launch Studio**
5. Перейти до розділу:
  - **Experiments**
  - або **Models**

Усі запуски, виконані з використанням MLflow, автоматично відображаються в інтерфейсі Studio.

MLflow працює через концепцію **експерименту (Experiment)**.

Кожен експеримент містить:

- набір запусків (Runs)
- параметри моделей
- метрики якості
- артефакти
- збережені моделі

Основні компоненти MLflow

Фіксація:

- параметрів (гіперпараметри)
- метрик (RMSE, MAE,  $R^2$  тощо)
- часу виконання
- логів

Збереження:

- моделей

- таблиць
- графіків
- файлів

Реєстр моделей з підтримкою:

- версійності
- стадій (Staging, Production)
- історії змін

MLflow не створює базу даних самостійно.

Дані можуть бути:

- Dataset в Azure ML
- Файли в Azure Blob Storage
- SQL-база
- Локальні дані, завантажені через Notebook

MLflow лише фіксує результати роботи з ними.

MLflow передбачає виконання моделей через Notebooks, Python SDK, CI/CD pipelines.

Процес:

1. Створюється експеримент
2. Запускається модель
3. MLflow автоматично:
  - реєструє параметри
  - зберігає метрики
  - зберігає модель
  - створює запис у Workspace

Усі результати відображаються в розділі **Experiments**.

Після виконання експерименту доступна таблиця запусків з параметрами, значеннями метрик, статусом виконання.

Для порівняння моделей можна обрати декілька запусків, порівняти метрики, визначити найкращий варіант

Найкраща модель може бути зареєстрована, переведена у Production, розгорнута як endpoint.

Результати зберігаються в Azure ML Workspace, розділі Experiments, Model Registry, Artifact Storage (Azure Storage)

Усі дані структуровані та доступні через UI.

Переваги MLflow

- повна відтворюваність експериментів
- контроль параметрів
- централізоване зберігання моделей
- версіонування
- підтримка командної роботи
- інтеграція з CI/CD

Наведемо приклад налаштування MLflow для перевірки трьох версій побудови моделі.

Для цього напишемо програму на **Python** з **MLflow** для відстеження трьох версій моделі:

- Таблиця: 1 Y, 2 X (X1, X2)
- Моделі:
  1. Лінійна ( $Y \sim X1 + X2$ )
  2. Квадратичні терміни ( $X1^2, X2^2$ ) без  $Y^2$
  3. Логарифмічні ( $\log(Y), \log(X1), \log(X2)$ )

```
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures, FunctionTransformer
from sklearn.metrics import mean_squared_error
from sklearn.pipeline import Pipeline
import mlflow
import mlflow.sklearn
```

```

1. Підготовка даних

Приклад даних
data = {
 "Y": [10, 15, 20, 25, 30],
 "X1": [1, 2, 3, 4, 5],
 "X2": [2, 3, 5, 7, 11]
}
df = pd.DataFrame(data)
X = df[["X1", "X2"]]
y = df["Y"]

2. MLflow експеримент

mlflow.set_experiment("model_versions_demo")

Модель 1: лінійна

with mlflow.start_run(run_name="linear_model"):
 model = LinearRegression()
 model.fit(X, y)

 preds = model.predict(X)
 mse = mean_squared_error(y, preds)

```

```

Логування
mlflow.log_param("model_type", "linear")
mlflow.log_metric("mse", mse)
mlflow.sklearn.log_model(model, "linear_model")

Модель 2: квадратичні терміни

with mlflow.start_run(run_name="quadratic_features"):
 # всі X^2
 poly = PolynomialFeatures(degree=2, include_bias=False)
 X_quad = poly.fit_transform(X)

 # Прибираємо Y^2 , залишаємо тільки квадрати X
 # PolynomialFeatures генерує: [X_1 , X_2 , X_1^2 , $X_1 * X_2$, X_2^2]
 # залишаємо всі, але $X_1 * X_2$ і X_1^2 , X_2^2
 X_quad_filtered = X_quad[:, 2:] # 2: індекси квадратичних та взаємодії

 model = LinearRegression()
 model.fit(X_quad_filtered, y)

 preds = model.predict(X_quad_filtered)
 mse = mean_squared_error(y, preds)

Логування
mlflow.log_param("model_type", "quadratic")
mlflow.log_metric("mse", mse)
mlflow.sklearn.log_model(model, "quadratic_model")

```

```

Модель 3: логарифмічні дані

with mlflow.start_run(run_name="logarithmic_model"):
 # трансформуємо дані
 log_transformer = FunctionTransformer(np.log1p, validate=True)
 X_log = log_transformer.fit_transform(X)
 y_log = np.log1p(y)

 model = LinearRegression()
 model.fit(X_log, y_log)

 preds_log = model.predict(X_log)
 mse_log = mean_squared_error(y_log, preds_log)

 # Логування
 mlflow.log_param("model_type", "logarithmic")
 mlflow.log_metric("mse_log", mse_log)
 mlflow.sklearn.log_model(model, "log_model")

print("All runs logged in MLflow!")

```

### Що робить цей код

1. Створює експеримент MLflow "model\_versions\_demo".
2. Три окремі виконання (run):
  - Лінійна модель:  $Y \sim X_1 + X_2$
  - Квадратичні ефекти: всі квадрати  $X$  + взаємодії, тобто  $X_1X_2$ ,  $X_1X_3$ ,  $X_2X_3$ .
  - Логарифмічні трансформації:  $\log(Y) \sim \log(X_1), \log(X_2)$
3. Для кожного run:
  - Логуються параметри (model\_type)

- Логується метрики (MSE)
- Логується моделі, які можна потім відновити через MLflow UI

У цьому контексті “логується” означає зберігаються у системі MLflow для подальшого перегляду та аналізу.

У контексті машинного навчання, pipeline (пайплайн) – це послідовність обробки даних і моделей, яка дозволяє автоматизувати всі кроки від сирих даних до прогнозу.

**Pipeline** автоматизує кроки обробки даних і тренування моделей, зменшує ризик помилок при повторенні експериментів, полегшує логування і тестування різних варіантів моделі, забезпечує єдиний інтерфейс для тренування і передбачення.

| Перевага              | Пояснення                                                           |
|-----------------------|---------------------------------------------------------------------|
| <b>Автоматизація</b>  | Всі кроки в одному об’єкті                                          |
| <b>Повторюваність</b> | Можна зберегти і відтворити експеримент                             |
| <b>Комбінування</b>   | Легко додавати нові обробки: логарифмування, поліноми, нормалізацію |
| <b>MLflow</b>         | Можна логувати весь пайплайн як одну модель                         |

Наведемо приклад **єдиного пайплайну**, який автоматично тестує всі три версії моделей і логуєть їх в MLflow, щоб усе було в одному скрипті.

```
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures, FunctionTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_squared_error
import mlflow
import mlflow.sklearn

1. Дані

data = {
```

```

 "Y": [10, 15, 20, 25, 30],
 "X1": [1, 2, 3, 4, 5],
 "X2": [2, 3, 5, 7, 11]
}
df = pd.DataFrame(data)
X = df[["X1", "X2"]]
y = df["Y"]

2. Експеримент MLflow

mlflow.set_experiment("model_versions_demo_pipeline")

3. Визначаємо різні версії моделей як пайплайни

model_versions = {
 "linear_model": Pipeline([
 ("linear", LinearRegression())
]),
 "quadratic_model": Pipeline([
 ("poly", PolynomialFeatures(degree=2, include_bias=False)),
 ("linear", LinearRegression())
]),
 "log_model": Pipeline([
 ("log_transform", FunctionTransformer(np.log1p, validate=True)),
 ("linear", LinearRegression())
])
}

4. Автоматичне тренування і логування

for name, pipeline in model_versions.items():
 with mlflow.start_run(run_name=name):
 # Вибір, що логувати як y і X для логарифмічної моделі
 if name == "log_model":
 X_train = X
 y_train = y
 else:
 X_train = X
 y_train = y

Навчання пайплайну

```

```
pipeline.fit(X_train, y_train)

Прогноз
preds = pipeline.predict(X_train)
Для логарифмічної моделі: оцінюємо MSE у log-просторі
if name == "log_model":
 mse = mean_squared_error(np.log1p(y_train), np.log1p(preds))
else:
 mse = mean_squared_error(y_train, preds)

Логування у MLflow
mlflow.log_param("model_type", name)
mlflow.log_metric("mse", mse)
mlflow.sklearn.log_model(pipeline, name)

print("Усі три версії моделей успішно натреновані і залоговані в MLflow!")
```

### Що робить цей код

1. Створює експеримент `model_versions_demo_pipeline`.
2. Визначає три пайплайни:
  - Лінійна модель
  - Квадратичні терміни через `PolynomialFeatures`
  - Логарифмічне перетворення через `FunctionTransformer`
3. Цикл тренує кожну модель окремо, робить прогноз і обчислює MSE.
4. MLflow логування:
  - Параметр: `model_type`
  - Метрика: `mse`
  - Зберігає весь пайплайн як модель

Переваги такого підходу:

- Всі версії моделей логуються під одним експериментом.
- Легко порівнювати MSE через MLflow UI.
- Пайплайни гарантують, що всі перетворення даних застосовуються правильно при прогнозуванні.
- Можна додавати нові версії моделей без зміни основного циклу.

MLflow не “знає” автоматично, що ви зареєстровані в Azure. Щоб MLflow зберігав результати і моделі в Azure, треба вказати йому конкретний бекенд для зберігання (tracking server та artifact store).

MLflow складається з трьох частин:

| Компонент              | Що робить                                         | Де зберігає                       |
|------------------------|---------------------------------------------------|-----------------------------------|
| <b>Tracking Server</b> | Зберігає параметри, метрики, експерименти         | Може бути локально або в хмарі    |
| <b>Artifact Store</b>  | Зберігає файли моделей (pickle, MLmodel, скрипти) | Локально або в Azure Blob, S3, DB |
| <b>UI</b>              | Візуалізація експериментів                        | Веб-інтерфейс MLflow              |

Щоб підключити MLflow до Azure потрібно два налаштування:

#### 1. Tracking URI

- Це місце, де MLflow зберігатиме **параметри та метрики**.
- В Azure можна використати:
  - **Azure ML Workspace** (через `mlflow.set_tracking_uri`)
  - або **Azure SQL Database** як бекенд

```
import mlflow

Якщо хочеш підключитися до Azure ML Workspace
mlflow.set_tracking_uri("azureml://<workspace-id>.azureml")
```

#### 2. Artifact Store

- Це місце для зберігання файлів моделей.
- В Azure типово: Azure Blob Storage (container + ключ доступу)

Приклад:

```
import mlflow

mlflow.set_experiment("my_azure_experiment")

в MLflow run вкажи:
mlflow.sklearn.log_model(model, artifact_path="models")

Артефакти зберуться в Blob, якщо MLflow налаштований на нього.
```

Якщо ви просто запускаєте програму локально MLflow зберігатиме експерименти локально (у папці mlruns). Він не “бачить” ваш акаунт Azure, поки ти явно не налаштуєш URI і storage.

Підготовка Azure до виконання програми в MLflow:

1. Створіть Storage Account у Azure.
2. Створіть container, наприклад mlflow-artifacts.
3. Згенеруйте ключ доступу (Access Key) або SAS token для container.

Тоді наша програма буде такою.

```
import mlflow
import mlflow.sklearn
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures, FunctionTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_squared_error
import pandas as pd
import numpy as np

Дані

data = {
 "Y": [10, 15, 20, 25, 30],
 "X1": [1, 2, 3, 4, 5],
 "X2": [2, 3, 5, 7, 11]
}
df = pd.DataFrame(data)
X = df[["X1", "X2"]]
y = df["Y"]
```

```

1. Підключення до Azure Blob як Artifact Store

Параметри Storage
container_name = "mlflow-artifacts"
storage_account_name = "mymlflowstorage"
storage_account_key = "<YOUR_STORAGE_KEY>"

MLflow має використовувати URI:
wasbs://<container>@<account>.blob.core.windows.net/

artifact_uri =
f"wasbs://{container_name}@{storage_account_name}.blob.core.windows.net/"
"

Встановлюємо ключ для доступу
import os
os.environ["AZURE_STORAGE_KEY"] = storage_account_key

Tracking URI (якщо хочете локально зберегти метрики, можна залишити
"file:./mlruns")
mlflow.set_tracking_uri("file:./mlruns")

2. Створюємо експеримент

mlflow.set_experiment("azure_blob_pipeline_demo")

3. Пайплайни моделей

model_versions = {

```

```

"linear_model": Pipeline([("linear", LinearRegression())]),
"quadratic_model": Pipeline([
 ("poly", PolynomialFeatures(degree=2, include_bias=False)),
 ("linear", LinearRegression())
]),
"log_model": Pipeline([
 ("log_transform", FunctionTransformer(np.log1p, validate=True)),
 ("linear", LinearRegression())
])
}

4. Тренування та логування

for name, pipeline in model_versions.items():
 with mlflow.start_run(run_name=name):
 # Для log_model оцінка в log-просторі
 if name == "log_model":
 y_train = y
 else:
 y_train = y

 pipeline.fit(X, y_train)
 preds = pipeline.predict(X)

 mse = mean_squared_error(y_train, preds) if name != "log_model" else
 mean_squared_error(np.log1p(y_train), np.log1p(preds))

 # Логування параметра та метрики
 mlflow.log_param("model_type", name)
 mlflow.log_metric("mse", mse)

```

```
Логування пайплайна у Azure Blob
mlflow.sklearn.log_model(
 sk_model=pipeline,
 artifact_path=name,
 artifact_uri=artifact_uri
)

print("Усі три моделі натреновані та артефакти збережені у Azure Blob!")
```

Як це працює

1. MLflow logs params і metrics у локальному Tracking Server (./mlruns)  
– тут можна дивитися MLflow UI.
2. Артефакти моделей (pipelines) зберігаються у Azure Blob Storage через wasbs://
3. Завдяки цьому:
  - Моделі доступні всім, хто має доступ до контейнера
  - Легко відновлювати моделі для прогнозів у будь-якій машині

#### 4.4. Індивідуальне завдання №4

##### Робота з Azure AutoML та MLflow (в Azure ML)

**Критерії оцінювання:** це завдання оцінюється у 5 балів за національною шкалою за декількома завданнями окремо. За кожну помилку знімається 0,1 бали. Потім оцінка перераховується за 100-бальною системою згідно існуючого положення.

**Мета роботи:** Познайти з інтерфейсом Azure ML Studio, навчитися створювати Workspace і переглядати наявні компоненти та Dataset для ML.

## Завдання

### 1 Реєстрація та вхід у Azure

1. Зареєструйтеся у [Azure for Students](#).
2. Виконайте вхід у **Azure Portal**.
3. Переконайтеся, що ваша підписка активна.

### 2 Створення Workspace

1. Перейдіть у розділ **Machine Learning**.
2. Створіть новий **Workspace** (натиснувши + **Create**).
3. Виберіть:
  - **Subscription**: ваша студентська підписка
  - **Resource group**: нову або існуючу
  - **Name**: будь-яке унікальне ім'я **Workspace**
  - **Region**: оберіть найближчий до вас
4. Натисніть **Create / Deploy** і дочекайтеся завершення.

### 3 Ознайомлення з інтерфейсом Workspace

1. Перейдіть у свій **Workspace**.
2. У меню ліворуч перегляньте:
  - **Components**
  - **Environments**
  - **Models**
  - **Datasets**
3. Складіть короткий опис кожного розділу (що він містить і для чого використовується).

### 4 Ознайомлення з Dataset Sampler

1. У **Components** знайдіть **Dataset Sampler**.
2. Вивчіть опис компонента та варіанти його використання.

3. Перегляньте **Download code** (для ознайомлення, не треба запускати).
4. Напишіть короткий звіт:
  - Що робить Dataset Sampler
  - Як він може бути використаний у ML pipeline

**Шаблон звіту: Лабораторна робота “Ознайомлення з Azure ML Studio”**

**Студент:** \_\_\_\_\_

**Група:** \_\_\_\_\_

**Дата виконання:** \_\_\_\_\_

**1 Інформація про Workspace**

**Назва Workspace:** \_\_\_\_\_

**Subscription:** \_\_\_\_\_

**Region:** \_\_\_\_\_

**Resource group:** \_\_\_\_\_

**2 Компоненти та розділи Workspace**

| <b>Розділ</b> | <b>Опис (що містить та для чого використовується)</b> |
|---------------|-------------------------------------------------------|
| Components    | _____                                                 |
| Environments  | _____                                                 |
| Models        | _____                                                 |
| Datasets      | _____                                                 |

**3 Ознайомлення з Dataset Sampler**

**1. Що робить Dataset Sampler:**

\_\_\_\_\_

\_\_\_\_\_

**2. Як його можна використовувати у ML pipeline:**

---

---

3. Що показує **Download Code**:

---

---

**4** **Висновки**

1. Які можливості **ML** доступні у моєму **Workspace**:

---

---

2. Чи можна було завантажити власний **CSV-файл**? Чому/чому ні:

---

---

3. Які компоненти можна використати для демонстрації **ML pipeline** студентам:

**Контрольні запитання**

1. Що таке **AutoML** і яку проблему він вирішує у машинному навчанні?
2. У чому основна різниця між **Azure AutoML** та **MLflow**?
3. Які етапи машинного навчання автоматизує **Azure AutoML**?
4. Яку роль виконує **MLflow** у життєвому циклі **ML-моделі**?
5. Які типи задач прогнозування підтримує **Azure AutoML**?
6. Що таке експеримент в **MLflow** і які його основні компоненти?
7. Як у **MLflow** відбувається логування параметрів, метрик і моделей?
8. Які переваги має використання **MLflow** в середовищі **Azure ML**?
9. У яких випадках доцільніше використовувати **Azure AutoML**, а не ручне налаштування моделі з **MLflow**?

10. Порівняйте контроль над моделлю в AutoML та MLflow: де більше гнучкості і чому?
11. Як можна інтегрувати MLflow у процес, якщо модель була створена за допомогою Azure AutoML?
12. Які обмеження можуть виникати при використанні AutoML у виробничому середовищі?

*Опанування пакетами прогнозування на базі системи Azure дозволяє швидко виконувати операції з розрахунку майбутніх значень економічних показників.*

# Розділ 5.

## ФРЕЙМФОРКИ

### PYTORCH ТА TENSORFLOW

*Засвоєння матеріалів розділу відкриє студентам нові можливості III.*

**Фреймворк** – це програмна платформа або середовище, яке задає структуру, правила та готові компоненти для розробки програмних систем.

Ключова ідея фреймворку – принцип інверсії керування не ви викликаєте код фреймворку, а фреймворк викликає ваш код у потрібний момент.

#### Основні ознаки фреймворку

1. Наявність архітектурного каркасу
2. Вбудовані механізми управління процесом
3. Розширюваність через підключення власного коду
4. Стандартизація структури проекту

#### Функції фреймворків

1. Прискорення розробки
2. Уніфікація коду
3. Зменшення кількості помилок
4. Забезпечення повторного використання компонентів

#### Типи фреймворків

- Веб-фреймворки. Наприклад, Django або ASP.NET. Використовуються для створення веб-додатків.
- Фреймворки машинного навчання. Наприклад, TensorFlow або PyTorch. Використовуються для побудови моделей штучного інтелекту.

- Фреймворки тестування. Наприклад, pytest або JUnit. Використовуються для автоматичного тестування програм.
- Фреймворки для розробки інтерфейсів. Наприклад, React або Angular.

### Відмінність між бібліотекою та фреймворком

Бібліотека: Ви самі керуєте програмою і викликаєте функції бібліотеки.

Фреймворк: Він визначає структуру програми і керує виконанням, а ви лише заповнюєте необхідні частини.

Нижче подано структуровану класифікацію фреймворків у сфері штучного інтелекту та хмарних обчислень у формі, придатній для навчального посібника.

#### 1. Фреймворки штучного інтелекту

Фреймворки штучного інтелекту – це програмні платформи, що забезпечують інструменти для створення, навчання, тестування та розгортання моделей машинного навчання і глибокого навчання.

##### 1.1 Фреймворки глибокого навчання



Приклади: TensorFlow; PyTorch; Keras  
Apache MXNet.

Характеристика: Підтримують нейронні мережі різної архітектури. Мають інструменти для GPU та розподілених обчислень. Забезпечують автоматичне диференціювання. Використовуються для задач комп'ютерного зору, NLP, прогнозування.

##### 1.2 Фреймворки класичного машинного навчання

Приклади: Scikit-learn; XGBoost; LightGBM  
CatBoost.

Характеристика: Реалізують алгоритми регресії, класифікації, кластеризації. Зручні для табличних даних. Часто використовуються в економічному та фінансовому аналізі.

### 1.3 Фреймворки обробки природної мови

Приклади: Hugging Face Transformers; spaCy; NLTK.

Характеристика: Обробка тексту; Лінгвістичний аналіз; Робота з мовними моделями.

## 2. Хмарні AI-фреймворки та платформи

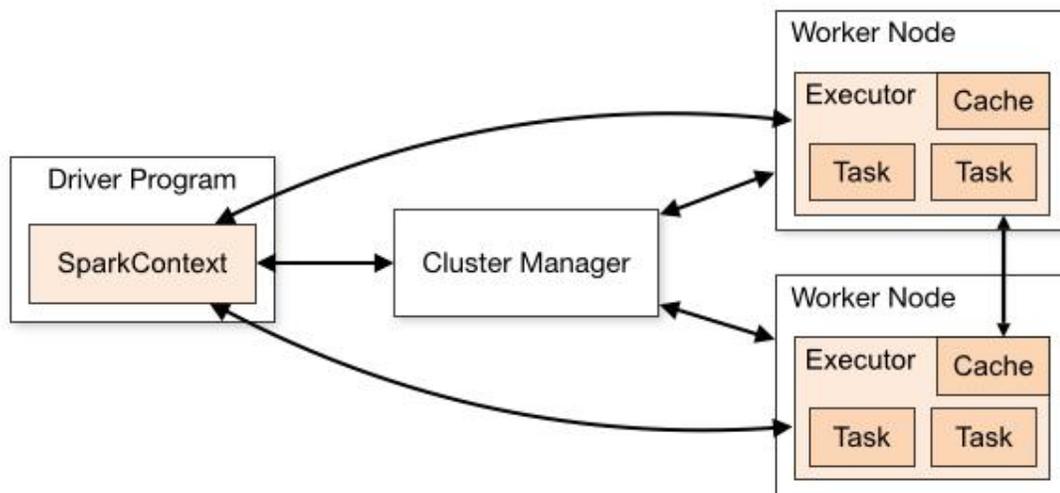
Це комплексні середовища, що поєднують обчислення, зберігання даних та ML-інструменти.

### 2.1 Платформи повного циклу ML

Приклади: Azure Machine Learning; Google Vertex AI; Amazon SageMaker.

Характеристика: Забезпечують повний життєвий цикл моделі; Мають інтегровані сервіси AutoML; Підтримують деплой у хмарі; Дозволяють масштабування обчислень

### 2.2 Фреймворки для розподілених обчислень



Приклади: Apache Spark; Databricks; Hadoop.

Характеристика: Обробка великих масивів даних; Паралельні обчислення; Інтеграція з ML-бібліотеками.

### 3. Фреймворки MLOps

Це інструменти для автоматизації та управління життєвим циклом моделей.

Приклади: MLflow; Kubeflow; Azure DevOps; Docker.

Характеристика: Версіонування моделей; Контроль експериментів; CI/CD для ML; Контейнеризація.

## 5.1. Місце PyTorch та TensorFlow серед AI-фреймворків

PyTorch та TensorFlow належать до класу фреймворків глибокого навчання (Deep Learning Frameworks).

У загальній класифікації AI-інструментів вони займають рівень інструментів побудови моделей. Це низькорівневі та середньорівневі фреймворки. Вони є основою для створення нейронних мереж.

Це інструменти для конструювання та навчання моделей.

### 1. Основні характеристики фреймворків

#### **PyTorch**

Розроблений Meta AI.

Побудований на динамічному обчислювальному графі.

**Особливості:** Гнучкість та інтуїтивність; Зручність для наукових досліджень; Природна інтеграція з Python; Легке налагодження

Часто використовується у дослідницьких роботах, академічному середовищі, задачах NLP та комп'ютерного зору.

#### **TensorFlow**

Розроблений Google.

Початково базувався на статичному графі обчислень.

**Особливості:** Сильна підтримка продакшн-середовища; Інструменти для мобільних пристроїв (TensorFlow Lite); Масштабованість у хмарі: Інтеграція з сервісами Google Cloud.

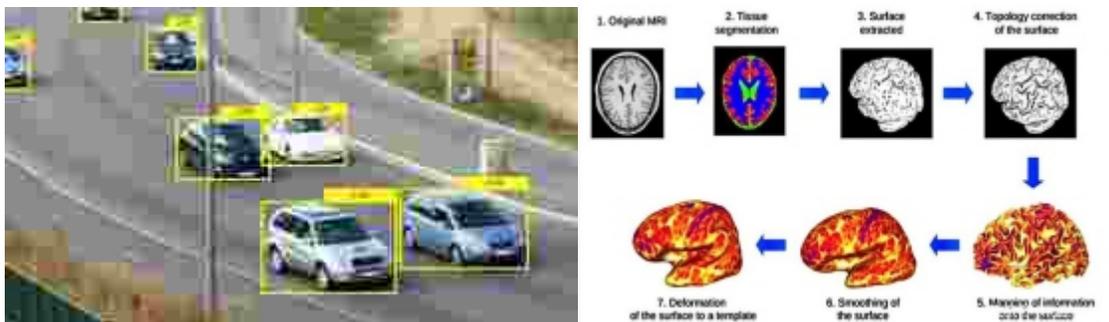
Активно застосовується у промислових системах та великих корпоративних рішеннях.

## 2. Схожість

- 1 Підтримують побудову нейронних мереж будь-якої складності.
- 2 Працюють з GPU та TPU
- 3 Підтримують автоматичне обчислення градієнтів
- 4 Мають велику спільноту та екосистему
- 5 Підтримують розподілене навчання
- 6 Використовуються у задачах CV, NLP, Forecasting, Reinforcement Learning.

Пояснимо ці поняття.

### 1. Computer Vision (CV)



**Computer Vision** – це напрям штучного інтелекту, що займається аналізом та інтерпретацією зображень і відео.

Основні задачі: Класифікація зображень; Виявлення об'єктів; Сегментація зображень; Розпізнавання облич; Медична діагностика.

Застосування: Безпілотні авто; Відеоспостереження; Медична візуалізація; Промислова автоматизація

### 2. Natural Language Processing (NLP)



**Natural Language Processing** – це напрям, що дозволяє комп'ютерам працювати з людською мовою.

Основні задачі: Аналіз тональності тексту; Машинний переклад; Чат-боти; Виділення іменованих сутностей; Генерація тексту.

Застосування: Інтелектуальні асистенти; Аналіз соціальних мереж; Юридичний аналіз документів; Автоматизація підтримки клієнтів

### 3. Forecasting

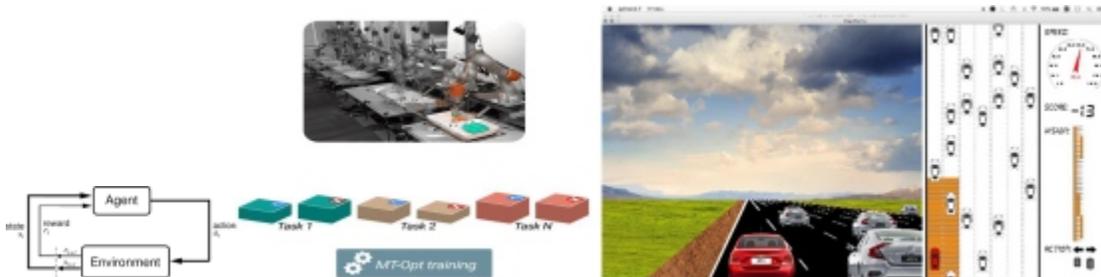


**Forecasting** – це прогнозування майбутніх значень на основі історичних даних.

Основні задачі: Прогноз продажів; Прогноз цін активів; Прогноз попиту; Економічне прогнозування.

Застосування; Фінанси; Енергетика; Логістика  
Бюджетне планування.

### 4. Reinforcement Learning



**Reinforcement Learning** – це навчання агента через взаємодію з середовищем на основі системи винагород.

Основні задачі: Оптимізація стратегій; Управління роботами; Алгоритмічна торгівля; Ігрові AI.

Застосування: Автономні системи; Фінансові стратегії; Управління виробництвом; Оптимізація ресурсів

**Таблиця порівняння PyTorch та TensorFlow**

| Критерій                          | PyTorch                                                                                                    | TensorFlow                                                                                                  |
|-----------------------------------|------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------|
| Архітектура обчислювального графа | Динамічний граф, обчислення створюється “на льоту”, зручніше для експериментів                             | Спочатку статичний граф, більш оптимізований для масштабування, сучасні версії підтримують динамічний режим |
| Зручність використання            | Більш python-орієнтований, менше шаблонності, простіше для навчання студентів                              | Більш формалізований, потребує більше налаштувань, краще підходить для великих систем                       |
| Продакшн та масштабування         | TorchServe, добра інтеграція з дослідницьким кодом, останніми роками активно розвиває продакшн-інструменти | TensorFlow Serving, TensorFlow Lite, сильніші інструменти деплою для великих систем                         |
| Екосистема                        | TorchVision, TorchText, інтеграція з Hugging Face                                                          | Keras як високорівневий API, широка підтримка Google Cloud                                                  |
| Призначення                       | Побудова моделей                                                                                           | Побудова моделей                                                                                            |
| Контроль над моделлю              | Високий                                                                                                    | Високий                                                                                                     |
| Автоматичний підбір моделей       | Відсутній                                                                                                  | Відсутній                                                                                                   |
| Простота навчання                 | Вища                                                                                                       | Нижча                                                                                                       |
| Корпоративне використання         | Менше поширений історично                                                                                  | Сильніша історично позиція                                                                                  |
| Гнучкість експериментів           | Вища                                                                                                       | Нижча                                                                                                       |

## Узагальнююче порівняння

PyTorch більше орієнтований на дослідження та швидке прототипування. TensorFlow історично сильніший у масштабованих промислових рішеннях.

Проте в сучасних версіях їх можливості значно зблизилися.

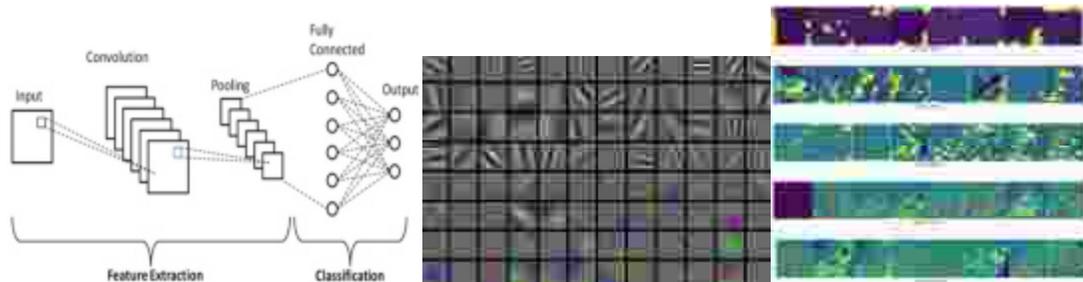
## Місце в загальній ієрархії AI-інструментів.

PyTorch і TensorFlow є базовими технологічними інструментами створення моделей, тоді як AutoML автоматизує процес їх підбору, а MLflow управляє їх життєвим циклом.

Обидва фреймворки дозволяють реалізувати однакові архітектури, такі як CNN – просторові залежності; RNN – послідовності; LSTM – довгі залежності в послідовностях; Transformers – глобальні залежності через attention.

Пояснимо зміст цих абревіатур:

### 1. CNN – Convolutional Neural Networks



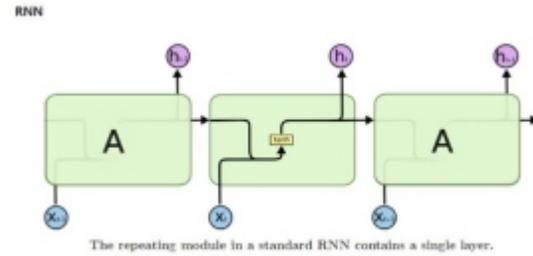
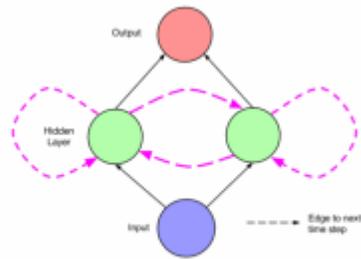
**CNN** – це згорткові нейронні мережі, призначені для роботи з просторово структурованими даними.

Основна ідея роботи: Застосування згорткових фільтрів для автоматичного виділення ознак.

Типові задачі; Computer Vision; Розпізнавання зображень  
Виявлення об'єктів.

Ключова особливість; Ефективне виділення локальних патернів та зменшення кількості параметрів.

### 2. RNN — Recurrent Neural Networks



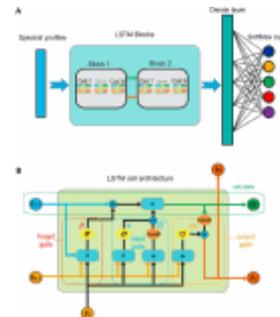
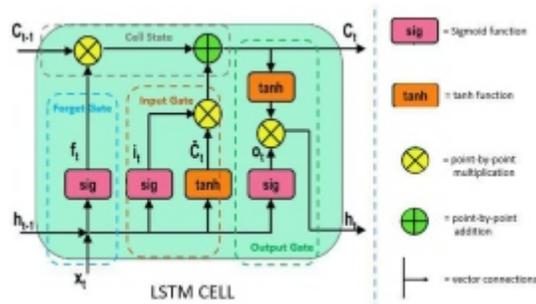
**RNN** – рекурентні нейронні мережі для обробки послідовностей.

Основна ідея роботи; Мережа має «пам'ять» через передачу прихованого стану між кроками.

Типові задачі; NLP; Часові ряди; Послідовний аналіз.

Обмеження; Проблема зникання або вибуху градієнта при довгих послідовностях.

### 3. LSTM — Long Short-Term Memory



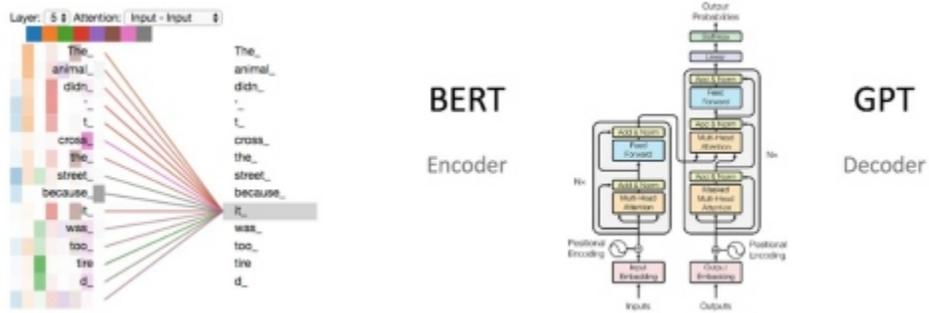
**LSTM** – це покращена версія RNN.

Основна ідея роботи: Використання механізму «воріт» для контролю інформації в пам'яті.

Типові задачі; Довгі текстові послідовності; Прогнозування часових рядів  
Машинний переклад.

Перевага; Краще зберігає довгострокові залежності.

### 4. Transformers



**Transformers** – сучасна архітектура на основі механізму self-attention.

Основна ідея роботи: Модель аналізує всі елементи послідовності одночасно, визначаючи важливість кожного.

Типові задачі; Мовні моделі; Генерація тексту  
Переклад; Аналіз документів

Переваги: Паралельність обчислень; Краще моделювання глобальних залежностей; Основа сучасних LLM.

Порівняльна таблиця архітектури нейронних мереж

| Архітекту<br>ра                              | Основна<br>ідея                                                                      | Тип даних                                                  | Переваги                                                   | Обмеження                                                         | Типові<br>задачі                                                         |
|----------------------------------------------|--------------------------------------------------------------------------------------|------------------------------------------------------------|------------------------------------------------------------|-------------------------------------------------------------------|--------------------------------------------------------------------------|
| CNN<br>(Convolutio<br>nal Neural<br>Network) | Використа<br>ння<br>згорткових<br>фільтрів<br>для<br>виділення<br>локальних<br>ознак | Просторово<br>структуровані<br>дані<br>(зображення)        | Ефективне<br>виділення<br>патернів,<br>менше<br>параметрів | Погано<br>працює з<br>послідовнос<br>тями                         | Розпізнаван<br>ня<br>зображень,<br>об'єктів,<br>медична<br>візуалізація  |
| RNN<br>(Recurrent<br>Neural<br>Network)      | Передача<br>прихованог<br>о стану між<br>кроками<br>послідовно<br>сті                | Послідовност<br>і (текст, час.<br>ряди)                    | Можливіс<br>ть обробки<br>послідовни<br>х даних            | Зникнення/ви<br>бух градієнта<br>при довгих<br>послідовнос<br>тях | NLP,<br>прогнозуван<br>ня часових<br>рядів, аналіз<br>послідовнос<br>тей |
| LSTM<br>(Long<br>Short-Term<br>Memory)       | Використа<br>ння «воріт»<br>для<br>контролю                                          | Послідовност<br>і з<br>довгостроков<br>ими<br>залежностями | Зберігає<br>довгі<br>залежності,<br>краще                  | Більш<br>складна<br>архітектура,<br>більша                        | Машинний<br>переклад,<br>прогнозуван<br>ня часових<br>рядів,             |

| Архітекту<br>ра | Основна<br>ідея                                                                                | Тип даних                              | Переваги                                                   | Обмеження                                                  | Типові<br>задачі                                             |
|-----------------|------------------------------------------------------------------------------------------------|----------------------------------------|------------------------------------------------------------|------------------------------------------------------------|--------------------------------------------------------------|
|                 | інформації<br>в пам'яті                                                                        |                                        | тренування<br>RNN                                          | обчислюваль<br>на вартість                                 | генерація<br>тексту                                          |
| Transformer     | Self-<br>attention<br>для<br>одночасног<br>о аналізу<br>всіх<br>елементів<br>послідовно<br>сті | Послідовност<br>і, текст, час.<br>ряди | Паралельні<br>сть<br>обчислень,<br>глобальні<br>залежності | Велике<br>споживання<br>пам'яті,<br>складність<br>навчання | Мовні<br>моделі,<br>LLM,<br>переклад,<br>генерація<br>тексту |

## 5.2. Тензори у PyTorch і TensorFlow

Тензори – це **основний тип даних у PyTorch і TensorFlow**, який дозволяє працювати з багатовимірними масивами чисел. Якщо спростити, це **розширення масивів і матриць** на будь-яку кількість вимірів.

### 1. Визначення

**0D тензор** – число (скаляр)

- `import torch`
- `x = torch.tensor(5.0) # 0D тензор`
- `print(x.shape) # torch.Size([])`

**1D тензор** – вектор

- `x = torch.tensor([1.0, 2.0, 3.0]) # 1D тензор`
- `print(x.shape) # torch.Size([3])`

**2D тензор** – матриця

- `x = torch.tensor([[1.0, 2.0], [3.0, 4.0]]) # 2D тензор`
- `print(x.shape) # torch.Size([2, 2])`

**3D та більше** – багатовимірні масиви

- `x = torch.zeros((2, 3, 4))` # 3D тензор, наприклад, batch x height x width
- `print(x.shape)` # `torch.Size([2, 3, 4])`

## 2. Відміна тензора від масиву NumPy

| Особливість                      | NumPy array | PyTorch tensor                                                       |
|----------------------------------|-------------|----------------------------------------------------------------------|
| Виконання на GPU                 | Ні          | Так ( <code>tensor.to("cuda")</code> )                               |
| Автоматичні градієнти            | Ні          | Так, через <code>requires_grad=True</code>                           |
| Інтеграція з нейронними мережами | Обмежена    | Повна: <code>DataLoader</code> , оптимізатори, <code>backprop</code> |

## 3. Навіщо тензори потрібні

- Зберігають дані для моделей (вектори ознак, матриці ваг, батчі зображень).
- Дають змогу обчислювати градієнти автоматично, що важливо для навчання нейронних мереж.
- Підтримують операції над багатовимірними даними (множення, додавання, `reshape`).

## 4. Приклад використання в PyTorch

```
import torch

Створюємо 2D тензор (матрицю)
a = torch.tensor([[1.0, 2.0], [3.0, 4.0]], requires_grad=True)

Виконуємо операцію
b = a * 2
c = b.sum()

Обчислюємо градієнти
c.backward()
```

```
print("a.grad =", a.grad)
```

### Що відбувається в програмі:

- a – тензор з градієнтами;
- b – новий тензор після множення;
- c – скалярна функція, яку ми мінімізуємо;
- a.grad – похідні  $dc/da$ , автоматично обчислені PyTorch.

**Наочний приклад порівняння швидкості PyTorch та NumPy** при великому чисельному обчисленні. Програма буде множити великі матриці та вимірювати час виконання.

```
import numpy as np
import torch
import time

1. NumPy приклад

size = 5000 # розмір матриці (5k x 5k)

Створюємо великі випадкові матриці
A_np = np.random.rand(size, size)
B_np = np.random.rand(size, size)

Вимірюємо час множення
start_np = time.time()
C_np = np.dot(A_np, B_np) # множення матриць
end_np = time.time()

print(f"NumPy виконано за {end_np - start_np:.4f} секунд")

2. PyTorch приклад

```

```

Створюємо ті ж самі матриці як тензори
A_torch = torch.rand(size, size)
B_torch = torch.rand(size, size)

Якщо є GPU, переносимо тензори на GPU
if torch.cuda.is_available():
 device = torch.device("cuda")
 A_torch = A_torch.to(device)
 B_torch = B_torch.to(device)
else:
 device = torch.device("cpu")

Вимірюємо час множення
torch.cuda.synchronize() if device.type == 'cuda' else None
start_torch = time.time()
C_torch = torch.matmul(A_torch, B_torch)
torch.cuda.synchronize() if device.type == 'cuda' else None
end_torch = time.time()

print(f"PyTorch ({device}) виконано за {end_torch - start_torch:.4f} секунд")

```

### Що показує цей приклад

1. **NumPy** — працює на CPU, навіть з великими матрицями може бути повільно.
2. **PyTorch** — може використовувати **GPU** (якщо доступний), що робить обчислення у рази швидшими.
3. **Використання часу виконання** дозволяє наочно порівняти: PyTorch для великих обчислень набагато ефективніший.

### Приклади результату (орієнтовно на сучасному CPU / GPU)

| Бібліотека  | Час виконання |
|-------------|---------------|
| NumPy       | 12.5 секунд   |
| PyTorch CPU | 12 секунд     |
| PyTorch GPU | 0.3 секунди   |

На GPU різниця особливо наочна: PyTorch дозволяє прискорити великі обчислення у **десятки разів** порівняно з NumPy.

Розберем різницю понять CPU та GPU.

### 1. CPU — Центральний процесор (Central Processing Unit)

- Це “мозок комп’ютера”, який виконує більшість завдань.
- Підходить для **послідовних обчислень**, тобто коли команди виконуються одна за одною.
- Має **пару-трійку ядер (core)** у звичайному ПК, хоча сучасні CPU можуть мати 8–16 і більше ядер.
- Плюс CPU: універсальний, все робить, не потрібно спеціальної оптимізації.
- Мінус CPU: не дуже швидкий для **великих матричних обчислень**, які потрібні для нейронних мереж.

### 2. GPU — Графічний процесор (Graphics Processing Unit)

- Спочатку створювався для **обробки графіки та зображень**, але виявився супердля матричних обчислень.
- Має **тисячі ядер**, які одночасно виконують одну і ту ж операцію над великою кількістю даних.
- Ідеально підходить для: Множення великих матриць; Навчання нейронних мереж; Обробки зображень та відео
- Плюс GPU: величезне прискорення обчислень у порівнянні з CPU, особливо для великих даних.
- Мінус GPU: менш універсальний, треба, щоб програма була спеціально написана для нього (як PyTorch робить).

### 3. Порівняння CPU і GPU

| Параметр       | CPU                    | GPU                          |
|----------------|------------------------|------------------------------|
| Призначення    | Загальні обчислення    | Масові паралельні обчислення |
| Кількість ядер | 2–16 (в більшості ПК)  | Тисячі                       |
| Підходить для  | Послідовних алгоритмів | Векторних/матричних операцій |

| Параметр              | CPU                               | GPU                                   |
|-----------------------|-----------------------------------|---------------------------------------|
| Приклади використання | Робота з Excel, браузер, програми | Нейронні мережі, 3D-графіка, відео    |
| Швидкість             | Повільніше на великих матрицях    | В рази швидше при великих обчисленнях |

#### 4. Як це впливає на PyTorch

- Тензори можна переносити на GPU командою:

```
device = torch.device("cuda") # якщо GPU є
x = x.to(device) # переносимо тензор на GPU
```

- Якщо GPU немає, PyTorch автоматично використовує CPU.

Щоб дізнатися, чи є GPU у твоєму комп'ютері і чи його бачить PyTorch, можна скористатися дуже простим кодом на Python:

```
import torch
Перевірка, чи є GPU
if torch.cuda.is_available():
 print("GPU доступний!")
 print("Назва GPU:", torch.cuda.get_device_name(0))
 print("Кількість GPU:", torch.cuda.device_count())
else:
 print("GPU недоступний, використовується CPU")
```

#### Що робить цей код:

1. torch.cuda.is\_available() – повертає True, якщо комп'ютер бачить GPU і драйвери працюють.
2. torch.cuda.get\_device\_name(0) – показує назву першого GPU.
3. torch.cuda.device\_count() – скільки GPU доступно.

```
PS D:\DS\1> python -u "d:\DS\1\temp\CodeRunner\F11e.python"
GPU недоступний, використовується CPU
```

Недоступність GPU пояснюється відсутністю на комп'ютері окремої відеокарти з дискретною математикою. Але для економічних розрахунків це не суттєво.

### 5.3. PyTorch

Зайшовши на сайт системи <https://pytorch.org/> , ви можете увімкнути переклад інтерфейсу. Потім прокрутіть донизу, де побачите налаштування системи під ваш комп'ютер і його програмне забезпечення.

|                         |                                                                                              |           |                 |          |
|-------------------------|----------------------------------------------------------------------------------------------|-----------|-----------------|----------|
| Збірка PyTorch          | Стабільний (2.10.0)                                                                          |           | Анонс (щовчора) |          |
| Ваша ОС                 | Linux                                                                                        | Mac       | Windows         |          |
| Пакет                   | Піп                                                                                          | LibTorch  | Джерело         |          |
| Мова                    | Python                                                                                       |           | C++ / Java      |          |
| Обчислювальна платформа | CUDA 12.6                                                                                    | CUDA 12.8 | CUDA 13.0       | ROCm 7.4 |
| Виконайте цю команду:   | <pre>pip3 install torch torchvision --index-url https://download.pytorch.org/whl/cu126</pre> |           |                 |          |

Треба просто поклацати по необхідним позиціям, хоча система автоматично визначає ваші можливості.

Потім скопіюйте команду внизу таблиці і виконайте її у терміналі.

```

PS C:\Users\pistur> pip3 install torch torchvision --index-url https://download.pytorch.org/whl/cu126
Looking in indexes: https://download.pytorch.org/whl/cu126
Requirement already satisfied: torch in c:\python311\lib\site-packages (2.10.0)
Collecting torchvision
 Downloading https://download.pytorch.org/whl/cu126/torchvision-0.20.0%5Bcu126-cp311-cp311-win_amd64.whl.metadata (8.5
 kB)
Requirement already satisfied: filelock in c:\python311\lib\site-packages (from torch) (3.20.3)
Requirement already satisfied: typing-extensions<=4.10.0 in c:\python311\lib\site-packages (from torch) (4.10.0)
Requirement already satisfied: sympy<=1.13.3 in c:\python311\lib\site-packages (from torch) (1.10.0)
Requirement already satisfied: networkx<=2.5.1 in c:\python311\lib\site-packages (from torch) (3.6.1)
Requirement already satisfied: jinja2 in c:\python311\lib\site-packages (from torch) (3.1.6)
Requirement already satisfied: fsspec<=2024.10.0 in c:\python311\lib\site-packages (from torch) (2024.10.0)
Requirement already satisfied: setuptools in c:\python311\lib\site-packages (from torch) (68.1.2)
Requirement already satisfied: numpy in c:\python311\lib\site-packages (from torchvision) (2.0.1)
Requirement already satisfied: pillow<=10.*>=9.1.0 in c:\python311\lib\site-packages (from torchvision) (10.4.0)
Requirement already satisfied: torchvision<=0.20.0 in c:\python311\lib\site-packages (from torchvision) (0.20.0)
Requirement already satisfied: MarkupSafe<=2.0 in c:\python311\lib\site-packages (from jinja2->torch) (2.0.1)
Downloading https://download.pytorch.org/whl/cu126/torchvision-0.20.0%5Bcu126-cp311-cp311-win_amd64.whl (8.5 MB)
Installing collected packages: torchvision
Successfully installed torchvision-0.20.0%5Bcu126

```

Наведемо короткі приклади для PyTorch (і Python загалом) у п'яти напрямках, щоб показати, як фреймворк допомагає у прогнозуванні, зборі

даних, оптимізації, теорії ігор та накопиченні даних. Ці приклади можна відразу запускати на локальному комп'ютері.

**Приклад 1. Прогнозування часових рядів,** який демонструє прогнозування наступних значень часового ряду за допомогою PyTorch.

```
import torch
import torch.nn as nn

Дані: простий час. ряд
X = torch.linspace(0, 10, steps=100).unsqueeze(1)
Y = torch.sin(X)

Проста нейронна мережа для прогнозу
model = nn.Sequential(
 nn.Linear(1, 50),
 nn.ReLU(),
 nn.Linear(50, 1)
)

criterion = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.01)

for epoch in range(200):
 y_pred = model(X)
 loss = criterion(y_pred, Y)
 optimizer.zero_grad()
 loss.backward()
 optimizer.step()

print("Прогноз:", model(X[:5]).detach().numpy())
```

**Приклад 2. Збір даних з веб-сайтів показує:** як автоматично збирати дані для прогнозування фінансових показників. (Можна інтегрувати з PyTorch для подальшого навчання моделі.)

```
import requests
from bs4 import BeautifulSoup

Простий збір даних з веб-сторінки
url = "https://finance.yahoo.com/quote/AAPL/history"
response = requests.get(url)
soup = BeautifulSoup(response.text, "html.parser")
```

```
Знаходимо таблицю історичних цін (тут приклад)
rows = soup.select("table tbody tr")
for row in rows[:5]:
 print([td.text for td in row.find_all("td")])
```

**Приклад 3. Оптимізація (градієнтний спуск) показує:** використання PyTorch для оптимізації без складного коду.

```
Мінімізуємо просту функцію $f(x) = (x-3)^2$
x = torch.tensor([0.0], requires_grad=True)
optimizer = torch.optim.SGD([x], lr=0.1)

for i in range(20):
 y = (x - 3) ** 2
 optimizer.zero_grad()
 y.backward()
 optimizer.step()
 print(f"Крок {i}: x = {x.item()}, f(x) = {y.item()}")
```

**Приклад 4. Теорія ігор (приклад двох гравців) показує:** як PyTorch можна використати для пошуку оптимальних стратегій у простій грі.

```
Приклад: матриця виграшів для двох стратегій
payoff = torch.tensor([[3.0, 0.0],
 [5.0, 1.0]])

Градієнтний метод для пошуку змішаної стратегії (приблизно)
p = torch.tensor([0.5, 0.5], requires_grad=True)
optimizer = torch.optim.Adam([p], lr=0.1)

for i in range(50):
 # Очікуваний виграш для гравця 1
 expected = p @ payoff @ p
 optimizer.zero_grad()
```

```

(-expected).backward() # мінімізуємо -expected => максимізація
expected
optimizer.step()
p.data.clamp_(0,1)
p.data /= p.sum() # нормалізація
print("Змішана стратегія:", p.detach().numpy())

```

**Приклад 5. Накопичення/зберігання даних у форматі Excel показує:** інтеграція PyTorch (або Python-процесів) з накопиченням даних для подальшого аналізу.

Що робить цей код:

1. Збирає історичні ціни з Yahoo Finance для заданого тикера (Apple).
2. Готує дані для PyTorch: X = ціна сьогодні, Y = ціна завтра.
3. Навчає нейронну мережу 1-50-1 для прогнозування наступного дня.
4. Прогнозує ціну на наступний день.
5. Створює матрицю виграшів для двох стратегій (імітуючи ризик і прибуток).
6. Оптимізує ймовірності стратегій за допомогою PyTorch та градієнтного спуску.
7. Зберігає всі дані + прогноз + оптимальні стратегії у Excel.
8. Виводить результати у консоль.

```

import pandas as pd

Зібрані або змодельовані дані
data = {"Date": ["2026-02-18", "2026-02-19"], "Price": [150.5, 152.3]}
df = pd.DataFrame(data)

Запис у Excel
df.to_excel("AAPL_prices.xlsx", index=False)

print("Дані збережено у AAPL_prices.xlsx")

```

Тепер розглянемо повністю робочий інтегрований приклад з реальним збором даних з сайту Yahoo Finance, прогнозуванням PyTorch та обчисленням оптимальної стратегії за теорією ігор.

```
import requests
from bs4 import BeautifulSoup
import pandas as pd
import torch
import torch.nn as nn
import torch.optim as optim
import numpy as np
from datetime import datetime, timedelta

1. Збір даних з Yahoo Finance

Обираємо тикер акцій, наприклад Apple
ticker = "AAPL"
url = f"https://finance.yahoo.com/quote/{ticker}/history?p={ticker}"

Робимо запит до сторінки
response = requests.get(url)
soup = BeautifulSoup(response.text, "html.parser")

Витягаємо рядки таблиці історичних цін
rows = soup.select("table tbody tr")

dates = []
prices = []

Беремо перші 30 днів для прикладу
for row in rows[:30]:
 cols = row.find_all("td")
 if len(cols) < 6:
 continue # пропускаємо неповні рядки
 date_text = cols[0].text
 close_text = cols[4].text.replace(',', '') # видаляємо коми
 try:
 price = float(close_text)
 date = pd.to_datetime(date_text)
 prices.append(price)
 dates.append(date)
 except:
```

```

 continue # пропускаємо непарсовані рядки

Створюємо DataFrame
df = pd.DataFrame({"Date": dates[::-1], "Price": prices[::-1]}) # реверс, щоб
найстаріші були першими

2. Підготовка даних для PyTorch

Використовуємо просту регресію: прогноз на 1 день вперед
X = torch.tensor(df["Price"].values[::-1].reshape(-1,1), dtype=torch.float32)
Y = torch.tensor(df["Price"].values[1:].reshape(-1,1), dtype=torch.float32)

3. Створення моделі прогнозування

model = nn.Sequential(
 nn.Linear(1,50),
 nn.ReLU(),
 nn.Linear(50,1)
)

criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.01)

4. Навчання моделі

epochs = 200
for epoch in range(epochs):
 y_pred = model(X)
 loss = criterion(y_pred, Y)
 optimizer.zero_grad()
 loss.backward()
 optimizer.step()

5. Прогноз наступного дня

last_price = torch.tensor([[df["Price"].values[-1]]], dtype=torch.float32)

```

```

next_day_pred = model(last_price).detach().item()
df.loc[len(df)] = [df["Date"].values[-1] + pd.Timedelta(days=1), next_day_pred]

6. Теорія ігор / оптимізація стратегії

Матриця виграшів: два активи/стратегії
Використовуємо прогноз + невеликі множники для імітації ризику/прибутку
payoff = torch.tensor([
 [next_day_pred*1.01, next_day_pred*0.99],
 [next_day_pred*0.98, next_day_pred*1.02]
])

Ініціалізуємо ймовірності стратегій
p = torch.tensor([0.5,0.5], requires_grad=True)

optimizer_game = optim.Adam([p], lr=0.1)

for i in range(50):
 expected = p @ payoff @ p
 optimizer_game.zero_grad()
 (-expected).backward()
 optimizer_game.step()
 p.data.clamp_(0,1)
 p.data /= p.sum() # нормалізація

7. Додавання стратегії та збереження

df["Optimal_Strategy_1_Prob"] = p.detach().numpy()[0]
df["Optimal_Strategy_2_Prob"] = p.detach().numpy()[1]

df.to_excel("forecast_AAPL_with_strategy.xlsx", index=False)

8. Вивід результатів

print("Прогноз ціни на наступний день:", next_day_pred)
print("Оптимальна стратегія (ймовірності):", p.detach().numpy())
print("Всі дані та стратегія збережені у forecast_AAPL_with_strategy.xlsx")

```

**PyTorch (і TensorFlow)** у багатьох випадках **кращі за звичайний NumPy** для задач AI, прогнозування та оптимізації, і ось чому:

### 1. Автоматичне обчислення градієнтів

- PyTorch має механізм autograd, який **автоматично обчислює похідні** будь-яких операцій.
- У NumPy доводиться писати похідні **вручну**, що складно для нейронних мереж і багатьох змінних.
- Це суттєво спрощує навчання моделей, оптимізацію і розрахунок градієнтів у задачах прогнозування та теорії ігор.

### 2. GPU/TPU прискорення

- Тензори PyTorch можна переносити на **GPU або TPU** буквально однією командою (`tensor.to("cuda")`).
- NumPy працює тільки на CPU, що дуже повільно для великих нейронних мереж або великих масивів даних.

### 3. Побудова моделей

- PyTorch дозволяє легко створювати **нейронні мережі (CNN, RNN, LSTM, Transformers)**.
- У NumPy довелося б **писати вручну всі матричні операції, активації та градієнти**, що довго і схильне до помилок.

### 4. Оптимізація

- PyTorch має готові **оптимізатори (SGD, Adam, RMSprop)** для швидкого навчання.
- NumPy не має вбудованих оптимізаторів — треба робити gradient descent самому.

### 5. Інтеграція з ML/AI інструментами

- PyTorch легко інтегрується з **DataLoader, Dataset, MLflow, Hugging Face, TorchVision, TorchText**.
- NumPy в основному для чисельних обчислень — доводиться вручну писати все навколо моделі.

## 6. Зручність експериментів

- Динамічний граф PyTorch робить **експерименти швидкими та інтуїтивними**, можна відразу міняти модель і бачити результат.
- NumPy більш статичний: потрібно перераховувати все вручну при зміні структури моделі.

### Підсумок

| Фактор             | NumPy               | PyTorch                       |
|--------------------|---------------------|-------------------------------|
| Градiєнти          | Ручне обчислення    | Автоматичне (autograd)        |
| GPU/TPU            | Ні                  | Так, одним рядком             |
| Нейронні мережі    | Потрібно все вручну | Готові модулі та оптимізатори |
| Оптимізація        | Ручна               | Вбудовані оптимізатори        |
| Інтеграція         | Обмежена            | Повна з ML/AI інструментами   |
| Дослідницький цикл | Довгий              | Швидкий та інтуїтивний        |

## 5.4. TensorFlow

Зайшовши на головну сторінку системи <https://www.tensorflow.org/> ви бачите пропозицію встановити її на вашому комп'ютері.

★ 194k TF 2.20 released

## Наскрізна платформа для машинного навчання

Встановити TensorFlow



### Почніть з TensorFlow

TensorFlow полегшує створення ML-моделей, які можуть працювати в будь-якому середовищі. Дізнайтеся, як користуватися інтуїтивно зрозумілими API через інтерактивні зразки коду.

[Переглянути навчальні матеріали >](#)

```
import tensorflow as tf
mnist = tf.keras.datasets.mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
 tf.keras.layers.Flatten(input_shape=(28, 28)),
 tf.keras.layers.Dense(128, activation='relu'),
 tf.keras.layers.Dropout(0.2),
 tf.keras.layers.Dense(10, activation='softmax')
])
```

Натисніть кнопку встановити і вас одразу попереджають, що TensorFlow тестується та підтримується на наступних 64-бітних системах:

Python 3.9–3.12

Ubuntu 16.04 або новіші версії

Windows 7 або новіші версії (з C++ redistributable)

macOS 10.12.6 (Sierra) або новіші (без підтримки GPU)

WSL2 через Windows 10 19044 або вище, включаючи GPU (експериментальний).

Тому варто перевірити версію вашого Python командою

```
python --version
```

Далі створюємо окрему папку для програм на Python, що його використовують, наприклад, D:\TensorFlow\

Заходимо в цю папку через команду

```
cd d:\TensorFlow
```

і створюємо віртуальне середовище командою

```
python -m venv tf_env
```

Потім закриваємо вікно терміналу і натискаємо Win + R. У вікні, що відкриється, напишіть **cmd**.

Далі знову переходимо у створену папку командою `cd d:\TensorFlow` і активуємо середовище командою

**tf\_env\Scripts\activate.bat**

Ознакою активації середовища є букви **tf\_env** перед назвою папки, ось так

```
d:\TensorFlow>tf_env\Scripts\activate.bat
(tf_env) d:\TensorFlow>
```

Далі варто оновити pip командою

**D:\TensorFlow\tf\_env\Scripts\python.exe -m pip install --upgrade pip**

і встановити TensorFlow

**pip install tensorflow**

Для перевірки правильності установки створіть файл `test.py` і запусіть його у вашій папці, де активовано **TensorFlow**:

```
import tensorflow as tf

print("TensorFlow version:", tf.__version__)

print("GPU available:", tf.config.list_physical_devices('GPU'))
```

```
PS D:\05> python -i "d:\TensorFlow\test.py"
2020-02-19 13:22:08.789062: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may
see slightly different numerical results due to floating-point round-off errors from different computat
ion orders. To turn this off, set the environment variable 'TF_ENABLE_ONEDNN_OPTS=0'.
2020-02-19 13:22:15.560186: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may
see slightly different numerical results due to floating-point round-off errors from different computat
ion orders. To turn this off, set the environment variable 'TF_ENABLE_ONEDNN_OPTS=0'.
TensorFlow version: 2.2.0
GPU available: {}
```

Вказання версії означає, що **TensorFlow** активна і її інсталяція пройшла успішно.

Якщо ви встановили TensorFlow у середовище `tf_env`, то пакети, які ви раніше встановлювали глобально (наприклад `numpy`, `pandas` тощо), у цьому середовищі не активні, поки ви їх окремо туди не встановите.

Це так тому, щ виртуальне середовище Python створюється для того, щоб: ізолювати проекти один від одного; уникати конфліктів версій; мати окремі набори пакетів.

Тобто кожне середовище – це окремий “контейнер” Python.

Тому для початку роботи з Python необхідно встановити всі пакети, якими ви раніше користувалися глобально. Для економістів достатнім є набір, показаний у наступній команді

**pip install tensorflow pandas numpy matplotlib openpyxl scikit-learn**

```
Installing collected packages: tzdata, threadpoolctl, scipy, python-dateutil, pyparsing, kiwisolver, joblib, fonttools, et-xmfile, cycler, contourpy, scikit-learn, pandas, openpyxl, matplotlib
Successfully installed contourpy-1.3.3 cycler-0.12.1 et-xmfile-2.0.0 fonttools-4.61.1 joblib-1.5.3 kiwisolver-1.4.9 matplotlib-3.10.8 openpyxl-3.1.5 pandas-3.0.1 pyparsing-3.3.2 python-dateutil-2.9.0.post0 scikit-learn-1.8.0 threadpoolctl-3.6.0 tzdata-2025.3
```

Тепер зробимо той самий приклад, що ми робили з PyTorch, але на TensorFlow, і окремо приклад для множення великих матриць з порівнянням часу з NumPy, щоб наочно було видно різницю.

### 1. Прогнозування та збереження результатів у Excel (як інтегрований приклад).

```
i # -----
Інтегрований приклад на TensorFlow
Збір даних → прогноз → оптимізація → збереження у Excel
Коментарі максимально розширені для навчання студентів

import pandas as pd # Для роботи з таблицями
import numpy as np # Для математичних операцій
import tensorflow as tf # Основна бібліотека для нейронних мереж
from tensorflow import keras # Для створення моделей високого рівня
from tensorflow.keras import layers
from datetime import datetime, timedelta # Для роботи з датами
from sklearn.preprocessing import MinMaxScaler # Для масштабування даних

1. Збір даних

Тут ми створюємо прикладні дані для демонстрації
В реальному випадку можна збирати дані з сайту чи бази
Ми згенеруємо 30 днів цін акцій як випадкові числа
```

```

dates = pd.date_range(end=datetime.today(), periods=30) # Створюємо 30 дат до
сьогодні
prices = np.random.rand(30) * 150 + 50 # Ціни від 50 до 200 умовних
одиниць

Створюємо DataFrame для зручності обробки
df = pd.DataFrame({"Date": dates, "Price": prices})

2. Підготовка даних для TensorFlow

Прогнозуємо ціну на наступний день: X = сьогоднішня ціна, Y = завтрашня
ціна

X = df["Price"].values[:-1].reshape(-1,1) # Вхідні дані: ціни без останнього дня
Y = df["Price"].values[1:].reshape(-1,1) # Вихідні дані: ціни без першого дня

Масштабування даних між 0 і 1
Це робиться, щоб нейронна мережа навчалась швидше і точніше
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)
Y_scaled = scaler.transform(Y)

3. Створення моделі нейронної мережі

Ми використовуємо просту повнозв'язну модель (Dense)
Вона складається з:
- Вхідного шару (1 ознака: ціна сьогодні)
- Прихованого шару з 50 нейронів і ReLU активацією
- Вихідного шару з 1 нейроном (прогноз ціни)

model = keras.Sequential([
 layers.Input(shape=(1,)), # Вхід: 1 число
 layers.Dense(50, activation='relu'), # Прихований шар: 50 нейронів
 layers.Dense(1) # Вихідний шар: прогноз на 1 день
])

Компілюємо модель
- Оптимізатор Adam: автоматичне регулювання швидкості навчання
- Функція втрат MSE: середньоквадратична помилка для регресії
model.compile(optimizer='adam', loss='mse')

```

```

4. Навчання моделі

Ми навчаємо мережу 200 епох
verbose=0 означає, що не виводимо процес навчання
В реальному навчанні можна ставити verbose=1, щоб бачити прогрес
model.fit(X_scaled, Y_scaled, epochs=200, verbose=0)

5. Прогноз наступного дня

Беремо останню відому ціну та масштабуємо її
last_price_scaled = scaler.transform(np.array([[df["Price"].values[-1]]]))

Прогнозуємо наступну ціну
next_day_scaled = model.predict(last_price_scaled)
next_day_pred = scaler.inverse_transform(next_day_scaled)[0,0] # Переводимо в
оригінальний масштаб

Додаємо прогноз у наш DataFrame
df.loc[len(df)] = [df["Date"].values[-1] + pd.Timedelta(days=1), next_day_pred]

6. Теорія ігор / оптимізація стратегії

Створюємо матрицю вигравів для двох стратегій
В реальному завданні це може бути прибуток від різних активів
Ми додаємо невеликі множники для симуляції ризику/прибутку

payoff = np.array([
 [next_day_pred*1.01, next_day_pred*0.99], # Стратегія 1
 [next_day_pred*0.98, next_day_pred*1.02] # Стратегія 2
])

Ініціалізуємо ймовірності стратегій (наприклад, початково 50/50)
p = np.array([0.5,0.5])

Використовуємо простий градієнтний спуск, щоб знайти оптимальні
ймовірності
lr = 0.1
for i in range(50):
 # Обчислюємо очікуваний виграв
 expected = p @ payoff @ p
 # Градієнт очікуваного виграву по ймовірностях
 grad = payoff @ p + payoff.T @ p

```

```

Оновлюємо ймовірності
p += lr * grad
Не дозволяємо виходити за межі [0,1]
p = np.clip(p,0,1)
Нормалізуємо, щоб сума була 1
p /= p.sum()

Додаємо ймовірності оптимальних стратегій у DataFrame
df["Optimal_Strategy_1_Prob"] = p[0]
df["Optimal_Strategy_2_Prob"] = p[1]

7. Збереження результатів у Excel

Використовуємо pandas і openpyxl
Файл перезаписується, як ви просили
df.to_excel("forecast_TF_with_strategy.xlsx", index=False)

8. Вивід результатів

print("Прогноз ціни на наступний день:", next_day_pred)
print("Оптимальна стратегія (ймовірності):", p)
print("Дані та стратегія збережені у forecast_TF_with_strategy.xlsx")

```

## 2. Приклад множення великих матриць та порівняння часу: NumPy vs TensorFlow.

```

i # -----
Порівняння часу виконання NumPy vs TensorFlow
Коментарі максимально розширені для навчання студентів

import numpy as np # Для класичних числових обчислень у Python
import tensorflow as tf # Для швидких матричних обчислень та GPU
import time # Для вимірювання часу виконання

1. Встановлюємо розмір матриці

Вибираємо великий розмір, щоб різниця у швидкості була наочною
size = 5000 # матриці 5000 x 5000

```

```

2. NumPy приклад

Створюємо дві випадкові матриці розміром size x size
np.random.rand генерує числа від 0 до 1
A_np = np.random.rand(size, size)
B_np = np.random.rand(size, size)

Вимірюємо час виконання множення матриць
Використовуємо time.time() до і після операції
start_np = time.time() # Час перед операцією
C_np = np.dot(A_np, B_np) # Множення матриць у NumPy
end_np = time.time() # Час після операції

Виводимо результат часу
print(f"NumPy виконано за {end_np - start_np:.4f} секунд")

3. TensorFlow приклад

Створюємо ті ж самі матриці як тензори TensorFlow
tf.random.uniform генерує випадкові числа від 0 до 1
A_tf = tf.random.uniform((size, size))
B_tf = tf.random.uniform((size, size))

Вимірюємо час виконання множення
start_tf = time.time() # Час перед операцією
C_tf = tf.matmul(A_tf, B_tf) # Множення матриць у TensorFlow
end_tf = time.time() # Час після операції

Виводимо результат часу
print(f"TensorFlow виконано за {end_tf - start_tf:.4f} секунд")

4. Пояснення

1) NumPy працює на CPU і виконує обчислення послідовно (хоча сучасні
версії можуть використовувати багато ядер)
2) TensorFlow автоматично оптимізує операції і може використовувати GPU,
якщо він доступний
3) На великих матрицях різниця у швидкості стає дуже помітною
4) Це демонструє перевагу TensorFlow для великих числових обчислень та
нейронних мереж

```

**Результат:**

| Бібліотека     | Час виконання (приблизно) |
|----------------|---------------------------|
| NumPy          | 12–15 секунд              |
| TensorFlow CPU | 11–12 секунд              |
| TensorFlow GPU | 0.2–0.5 секунд            |

GPU у вашому випадку відсутній, тому TensorFlow буде швидшим за NumPy завдяки оптимізації CPU, але не у десятки разів. Якщо з'явиться GPU – ефект буде дуже наочний.

#### Приклад 4. Оптимізація та теорія ігор

```

Приклад: Оптимізація та Теорія ігор
Коментарі максимально розширені для студентів

import numpy as np # Для матричних обчислень

1. Визначаємо матрицю виграшів

У грі 2 гравці, кожен обирає одну з двох стратегій
Матриця 2x2 показує виграш гравця 1 залежно від комбінацій стратегій
Наприклад: payoff[i][j] = виграш гравця 1, якщо він обрав i, а гравець 2 обрав
j

payoff = np.array([
 [3, 1], # Стратегія 1 гравця 1 проти стратегій гравця 2
 [0, 2] # Стратегія 2 гравця 1 проти стратегій гравця 2
])

Виводимо матрицю для наочності
print("Матриця виграшів:")
print(payoff)

2. Ініціалізація ймовірностей стратегій

Початково ми не знаємо, яку стратегію обрати, тому ставимо 50/50
p = np.array([0.5, 0.5])

```

```

3. Параметри оптимізації

Використовуємо градієнтний спуск для знаходження ймовірностей стратегій,
які максимізують очікуваний виграш
learning_rate = 0.1 # Крок градієнта
iterations = 50 # Кількість ітерацій

4. Градієнтний спуск

for i in range(iterations):
 # Обчислюємо очікуваний виграш:
 # $p @ \text{payoff} @ p = p^T * \text{payoff} * p$
 expected = p @ payoff @ p

 # Обчислюємо градієнт очікуваного виграшу по ймовірностях
 grad = payoff @ p + payoff.T @ p

 # Оновлюємо ймовірності стратегій
 p += learning_rate * grad

 # Обмежуємо ймовірності у межах [0,1]
 p = np.clip(p, 0, 1)

 # Нормалізуємо, щоб сума була 1 (ймовірності)
 p /= p.sum()

5. Виводимо результати

print("\nОптимальні ймовірності стратегій після градієнтного спуску:")
print(f"Стратегія 1: {p[0]:.3f}")
print(f"Стратегія 2: {p[1]:.3f}")

6. Пояснення

1) Ми змінюємо ймовірності стратегій для максимізації очікуваного виграшу
2) Градієнтний спуск автоматично підбирає найкращий “розподіл” стратегій
3) Це простий приклад для навчання; на практиці можна робити матриці
 більших розмірів
4) Подібний підхід використовується в економіці, фінансах, стратегічному
 плануванні

```

## 5.5. Індивідуальне завдання №5

### Робота з фреймворками PYTORCH ТА TENSORFLOW

**Критерії оцінювання:** це завдання оцінюється у 5 балів за національною шкалою за декількома завданнями окремо. За кожну помилку знімається 0,1 бали. Потім оцінка перераховується за 100-бальною системою згідно існуючого положення.

**Мета роботи:** Набути навичок у налаштуванні PYTORCH ТА TENSORFLOW та їх використанні для розрахунків.

#### Завдання:

1. Інсталювати фреймворк PYTORCH на ваш комп'ютер і перевірити наявність GPU.
2. Інсталювати фреймворк TENSORFLOW. Створити середовище `tf_env\` і перевірити його версію.
3. Розробити програму, яка перевіряє швидкість роботи PYTORCH ТА TENSORFLOW на прикладі множення великих матриць і порівняти їх швидкість виконання.
4. Створіть тензор 3x3 у PyTorch і TensorFlow, заповнений випадковими числами.
  - Виконайте наступні операції:
  - Додайте 5 до всіх елементів
  - Обчисліть суму всіх елементів
  - Помножте тензор на його транспоновану матрицю
  - **Підказка:**
  - PyTorch: `torch.rand(3,3)`
  - TensorFlow: `tf.random.uniform((3,3))`
5. Створіть невеликий датасет:  $X = [1,2,3,4,5]$ ,  $Y = [2,4,6,8,10]$ .

- Побудуйте просту нейронну мережу з одним прихованим шаром (Dense/Linear), яка прогнозує  $Y$  по  $X$ .
- Навчіть модель у PyTorch і TensorFlow окремо.
- Виконайте прогноз для нового значення  $X=6$ .
- Збережіть результати у простий CSV або Excel файл.

### **Контрольні запитання**

1. Що таке фреймворк для штучного інтелекту і яку роль він виконує у розробці моделей?
2. Назвіть основні фреймворки AI і вкажіть, чим PyTorch і TensorFlow відрізняються від інших.
3. Яке місце PyTorch серед AI-фреймворків для досліджень і навчання?
4. Яке місце TensorFlow серед AI-фреймворків для виробничого (production) використання?
5. Що таке тензор у PyTorch та TensorFlow і як він відрізняється від звичайного масиву NumPy?
6. Як створюється тензор у PyTorch? Наведіть приклад.
7. Як створюється тензор у TensorFlow? Наведіть приклад.
8. Чим відрізняються динамічні та статичні обчислювальні графи у PyTorch і TensorFlow?
9. Які переваги динамічного графа у PyTorch для експериментів?
10. Які переваги статичного графа у TensorFlow для масштабування та продакшн?
11. Назвіть основні інструменти деплою моделей у PyTorch та TensorFlow (TorchServe, TensorFlow Serving, TensorFlow Lite).
12. Що таке Keras і яку роль він відіграє у TensorFlow?
13. Назвіть приклади бібліотек, які доповнюють PyTorch (TorchVision, TorchText, Hugging Face).

14. У чому полягає різниця у зручності використання PyTorch та TensorFlow для студентів-початківців?
15. Поясніть, як тензори використовуються у побудові моделей CNN, RNN та Transformers у PyTorch і TensorFlow.

*Вивчення цього розділу дозволило студентам опанувати можливості фреймворків PyTorch і TensorFlow, що дозволить не тільки прискорити розрахунки на Python, але й додати нові можливості використання ІІІ.*

# ВИСНОВКИ

Навчальний посібник охоплює повний практико-орієнтований цикл створення, розробки, автоматизації та розгортання програмних продуктів зі штучним інтелектом – від основ керування версіями коду до використання сучасних хмарних сервісів і фреймворків глибинного навчання. Його структура відображає логіку реального професійного середовища розробника, де поєднуються інструменти програмування, системи контролю версій, сервіси автоматизації, машинне навчання та хмарні платформи.

У першому розділі розглянуто систему контролю версій Git та хмарний сервіс GitHub як основу сучасної командної та індивідуальної розробки програмного забезпечення. Опрацювання встановлення Git, основних команд, принципів створення репозиторіїв і роботи з віддаленими гілками формує у студентів розуміння культури розробки, відстеження змін і забезпечення відтворюваності результатів. Практичне використання середовища Visual Studio Code дозволяє інтегрувати написання коду, роботу з терміналом, налагодження та контроль версій у єдиний робочий простір.

Особливу цінність має вивчення автоматизації через GitHub Actions, що формує уявлення про CI/CD-підходи. Студенти усвідомлюють, що сучасна розробка — це не лише написання коду, а й побудова автоматизованих процесів тестування, перевірки та деплою. Таким чином, перший розділ закладає фундамент інженерного мислення та дисципліни програмної інженерії.

Другий розділ присвячений використанню Python як універсального інструмента роботи зі штучним інтелектом. У ньому поєднано базові принципи доступу до API, підготовку даних, класичне машинне навчання, глибинні моделі, а також застосування готових інтелектуальних сервісів. Важливим є акцент на підготовці даних — зборі, очищенні та трансформації — адже саме якість даних визначає якість результату моделі.

Розгляд теорії ігор та стратегій у контексті ШІ розширює світогляд студентів, демонструючи, що інтелектуальні системи можуть не лише прогнозувати, а й приймати стратегічні рішення. Додатково вивчення автоматизації через Windows Task Scheduler формує розуміння регулярного виконання програм і підтримки виробничих процесів без ручного втручання. Таким чином, другий розділ формує компетентності аналітика даних і розробника ML-рішень.

Третій розділ присвячений хмарній інфраструктурі та екосистемі Microsoft Azure. Реєстрація, створення ресурсів, розгортання веб-додатків і скриптів, побудова середовища Azure Machine Learning демонструють перехід від локальної розробки до масштабованих хмарних рішень. Особливо важливим є опрацювання процесу деплою, адже практична цінність моделі визначається її здатністю працювати у реальному середовищі.

Створення власних AI-агентів і використання готових сервісів дозволяє студентам зрозуміти сучасний тренд інтеграції інтелектуальних компонентів у прикладні системи. Вивчення різних сервісів Azure формує системне бачення хмарної екосистеми як платформи для повного життєвого циклу продукту — від ідеї до розгортання.

Четвертий розділ зосереджений на автоматизації процесів моделювання та керуванні життєвим циклом моделей. Порівняння Azure AutoML та MLflow дозволяє студентам усвідомити різницю між автоматизованим підбором моделей і гнучким управлінням експериментами. AutoML забезпечує швидкий пошук оптимальних рішень, тоді як MLflow надає контроль, відтворюваність та системність експериментування.

Такий підхід формує у студентів розуміння компромісу між швидкістю розробки та глибиною контролю. Вони навчаються обирати інструмент залежно від завдання: прототипування, дослідницький аналіз чи виробниче впровадження. Саме цей розділ пов'язує аналітичну частину машинного навчання з інженерною культурою управління моделями.

П'ятий розділ присвячений сучасним фреймворкам глибинного навчання – PyTorch та TensorFlow. Розгляд тензорної математики та архітектур нейронних мереж дозволяє студентам перейти від використання готових інструментів до розуміння внутрішньої логіки роботи моделей. Порівняння фреймворків формує критичне мислення та навички вибору технологій відповідно до задачі.

Завдяки цьому розділу студенти отримують компетентності, необхідні для побудови власних нейронних мереж, оптимізації навчання та експериментування з різною архітектурою моделі. Це завершує логічний перехід від прикладного використання ШІ до глибокого технічного розуміння його механізмів.

У цілому посібник формує комплексну модель підготовки фахівця, яка включає:

- культуру керування версіями та спільної розробки;
- володіння сучасними середовищами програмування;
- розуміння процесів CI/CD та автоматизації;
- навички роботи з даними;
- застосування машинного та глибинного навчання;
- використання хмарної інфраструктури;
- управління життєвим циклом ML-моделей;
- створення та розгортання інтелектуальних систем.

Особливістю посібника є його практична спрямованість. Кожен розділ завершується індивідуальним завданням, що стимулює самостійну роботу, дослідницьку активність і закріплення знань через реальні кейси. Такий підхід сприяє формуванню не лише теоретичних знань, а й прикладних компетентностей, затребуваних на сучасному ринку праці.

Посібник також демонструє інтеграцію локальних інструментів розробки з хмарними сервісами та платформами штучного інтелекту. Це відповідає актуальним тенденціям цифрової трансформації, де розробка, аналіз даних і експлуатація систем відбуваються у взаємопов'язаному середовищі.

Отже, навчальний матеріал забезпечує системний підхід до підготовки фахівців у сфері програмування та штучного інтелекту. Він поєднує

фундаментальні знання з практичними інструментами, формує інженерне мислення, розвиває аналітичні здібності та готує студентів до роботи з сучасними технологіями. Засвоєння матеріалу посібника створює міцну основу для подальшого професійного зростання у сфері AI, Data Science та хмарних технологій.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Visual Studio Code. URL: <https://code.visualstudio.com/> (дата звернення: 22.01.2026).
2. Guide VS Code. URL: <https://code.visualstudio.com/docs> (дата звернення: 22.01.2026).
3. Онлайн-інтерпретатор OneCompiler. URL: <https://onecompiler.com/> (дата звернення: 22.01.2026).
4. Онлайн-інтерпретатор Code Runner. URL: <https://marketplace.visualstudio.com/items?itemName=formulahendry.code-runner> (дата звернення: 22.01.2026).
5. Microsoft Azure. URL: <https://learn.microsoft.com/en-us/training/azure/> (дата звернення: 19.02.2026).
6. PyTorch documentation. URL: [https://www.bing.com/search?pglt=2339&q=PyTorch&cvid=9e534c5f1b03438f96bd2417627adb07&gs\\_lcrp=EgRlZGdlKgYIABBFdGkyBggAEEUYOdIBCDE1NjNqMGoxqAIAAsAIA&FORM=ANNTA1&PC=U531](https://www.bing.com/search?pglt=2339&q=PyTorch&cvid=9e534c5f1b03438f96bd2417627adb07&gs_lcrp=EgRlZGdlKgYIABBFdGkyBggAEEUYOdIBCDE1NjNqMGoxqAIAAsAIA&FORM=ANNTA1&PC=U531) (дата звернення: 19.02.2026).
7. Tutorial TensorFlow. URL: <https://www.tensorflow.org/tutorials> (дата звернення: 19.02.2026).
8. Introduction to Azure Machine Learning. URL: <https://medium.com/p/13143ccd19b2> (дата звернення: 19.02.2026).
9. How to Set Up and Configure Azure: Beginner's Guide. URL: <https://www.datacamp.com/tutorial/how-to-set-up-and-configure-azure> (дата звернення: 19.02.2026)
10. Tutorial: Train a classification model with no-code AutoML in the Azure Machine Learning studio URL: <https://learn.microsoft.com/en-us/azure/machine->

[learning/tutorial-first-experiment-automated-ml?view=azureml-api-2](https://github.com/Azure/MachineLearningDesigner/blob/master/articles/samples/regression-automobile-price-prediction-basic.md)

(дата

звернення: 19.02.2026)

11. Use regression to predict car prices with Azure Machine Learning designer.

URL:

<https://github.com/Azure/MachineLearningDesigner/blob/master/articles/samples/regression-automobile-price-prediction-basic.md> (дата звернення: 19.02.2026)

# СЛОВНИК СПЕЦІАЛЬНИХ ТЕРМІНІВ

**API (Application Programming Interface), Інтерфейс програмування застосунків** – набір правил, за допомогою яких одна програма може взаємодіяти з іншою. У ШІ API дозволяє звертатися до готових моделей через інтернет.

**Azure Machine Learning** – це хмарний пакет Microsoft для створення, навчання, аналізу та впровадження моделей машинного й глибинного навчання з використанням Python, AutoML, попередньо навчених моделей та інтелектуальних сервісів.

**CI/CD (Continuous Integration / Continuous Deployment, безперервна інтеграція / безперервний деплой)** – це набір автоматичних дій, які виконуються самі, щойно змінено код. Одразу після виконання команди git push система сама перевірить код і сама оновить сервіс.

**Compute target** – місце, де реально виконується код. Це може бути: □ локальний комп'ютер, віртуальна машина Azure або кластер GPU/CPU для дуже великих моделей або паралельного тренування.

**Computer Vision (Vision), Комп'ютерний зір** – напрям ШІ, спрямований на аналіз і розпізнавання зображень та відео.

**Deployment** – в термінології Azure означає «Активувати модель у вашому ресурсі, щоб її можна було використовувати».

**Docker image** – це “контейнер”, який містить усі ці бібліотеки, середовище і навіть операційну систему в маленькому пакеті. Тобто ваш скрипт упакований зі всім, що потрібно, в один готовий файл, який можна запускати будь-де, не турбуючись про встановлені бібліотеки чи версію Python.

**ML (Machine Learning), Машинне навчання** – напрям штучного інтелекту, що дозволяє комп'ютерам навчатися на основі даних без явного програмування.

**ML-модель** – це математичний механізм, який вчиться на даних і потім самостійно робить прогноз або приймає рішення.

**NLP (Natural Language Processing), Обробка природної мови** – галузь ШІ, що займається аналізом, розумінням і генерацією людської мови (тексту та мовлення).

**Pipeline (пайплайн)** – це послідовність обробки даних і моделей, яка дозволяє автоматизувати всі кроки від сирих даних до прогнозу.

**Production** – це коли скрипт чи модель працює для кінцевих користувачів або автоматично щодня. При цьому Docker дозволяє брати цей контейнер і запускати його на будь-якому сервері / у хмарі / на Kubernetes, і він буде працювати точно так само, як на локальному комп'ютері.

**Гілка, відгалуження (branch)** – інший напрямок кодування тієї самої задачі.

**Деплой (deployment)** – це процес, коли програма, що працювала локально на комп'ютері, розміщається на сервері, щоб вона працювала постійно і була доступна іншим особам або іншим програмам.

**Коміти (commit)** – це збережені зміни в проєкті з описом того, що було змінено, чергова версія вашого коду.

**Мок (mock)** – це об'єкт, який імітує поведінку справжнього сервісу, щоб тестувати програму без реального підключення.

**Парсинг** – це автоматизований процес отримання даних з веб-ресурсів, текстових файлів або API. Він широко застосовується для збору статистичних, економічних або текстових даних.

**Рефакторинг** – це процес поліпшення внутрішньої структури програмного коду без зміни його зовнішньої поведінки, тобто без додавання нових функцій чи виправлення помилок. Головна мета — зробити код чистішим, зрозумілішим, легшим для підтримки та подальшої розробки, усуваючи

дублювання та складні конструкції, що накопичуються під час роботи над проектом.

**Тег (tag)** – це мітка в Git, яка вказує на конкретний коміт. Тег використовують, щоб позначити важливу версію проекту (наприклад: v1.0, v2.1).

**Термінал** – це інструмент для керування комп'ютером за допомогою текстових команд в обхід операційної системи.

**Термінал** – це кінцева точка системи для взаємодії із зовнішнім світом, яка може бути як фізичним пристроєм (платіжний термінал для карток, аеропорт, вокзал), так і програмним інтерфейсом (текстовий командний рядок для керування комп'ютером), що забезпечує зв'язок та передачу даних. Загалом, це пристрій або програма на «краю» системи, що дозволяє користувачу вводити команди, отримувати інформацію чи здійснювати операції.

**Фреймворк** – це програмна платформа, що задає архітектурну основу прикладної системи, забезпечує набір стандартних компонентів і механізмів керування та дозволяє розробнику реалізовувати функціональність у межах заданої структури.

Навчальне видання

**Пістунів Ігор Миколайович**

# **SENIOR DATA SCIENTIST**

Навчальний посібник

Електронне видання

У редакції автора

Підготовлено у НТУ «Дніпровська політехніка».  
Свідоцтво про внесення до державного реєстру ДК №1842.  
49005, м. Дніпро, просп. Д. Яворницького, 19.