ТОВ «ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«МЕТІНВЕСТ ПОЛІТЕХНІКА»

# ПРОГРАМУВАННЯ НА PYTHON:

## методичні рекомендації
## до виконання практичних та індивідуальних робіт

Запоріжжя 2024

metinvest polytechnic

УДК 004:43(072)
П68

Рекомендовано Науково-методичною радою
ТОВ «ТЕХНІЧНИЙ УНІВЕРСИТЕТ «МЕТІНВЕСТ ПОЛІТЕХНІКА»
(протокол № 2 від 25.10.2024 р.)

***Укладач***

Жерліцин Д.М., доктор економічних наук, професор

П68 **Програмування на Python** : методичні рекомендації до виконання практичних та індивідуальних робіт / уклад. Д. М. Жерліцин. Запоріжжя : ТОВ «ТЕХНІЧНИЙ УНІВЕРСИТЕТ «МЕТІНВЕСТ ПОЛІТЕХНІКА», 2024. 61 с.

Методичні рекомендації включають алгоритм роботи з мовою програмування Python для аналізу та візуалізації даних з використанням сучасних бібліотек, таких як NumPy, Pandas, Matplotlib та Seaborn. Надані приклади виконання завдань охоплюють обробку структур даних, побудову графіків, а також автоматизацію аналітичних процесів. Рекомендовано для здобувачів вищої освіти за освітньо-професійною програмою першого (бакалаврського) рівня спеціальності 051 «Економіка».

УДК 004:43(072)

## ЗМІСТ

# Вступ

Методичні рекомендації з виконання практичних та індивідуальних робіт із дисципліни «Програмування на Python» створені для здобувачів вищої освіти першого (бакалаврського) рівня за спеціальністю 051 «Економіка». Метою рекомендацій є допомога студентам у засвоєнні основних концепцій програмування на мові Python та їх застосуванні для аналізу й обробки даних.

Python є потужним інструментом, який поєднує простоту освоєння з широкими можливостями. Завдяки своїй універсальності, ця мова програмування знаходить застосування у численних сферах, таких як аналіз даних, машинне навчання, візуалізація даних, автоматизація процесів та розробка програмних продуктів. Особливу увагу у методичних вказівках приділено практичним аспектам використання Python у бізнес-аналізі та економічних дослідженнях, що відповідає профілю спеціальності.

Методичні рекомендації охоплюють тематику від налаштування середовища програмування до опрацювання складних структур даних і створення графічних звітів. Матеріали представлені таким чином, щоб полегшити навчальний процес через структуровані завдання, покрокові інструкції та практичні економічні приклади.

Основними завданнями курсу є:

- ознайомлення з базовими концепціями та алгоритмами програмування;
- вивчення структур даних та основних методів їх обробки з Python;
- застосування інструментів Python для статистичного аналізу даних та їх візуалізації;
- розвиток аналітичного мислення через виконання завдань з реальними кейсами.

Рекомендації сприяють формуванню у здобувачів освіти професійних навичок, таких як:

- написання структурованого коду з дотриманням стандартів Python;
- використання сучасних бібліотек для аналізу та візуалізації даних (NumPy, Pandas, Matplotlib, Seaborn);
- застосування програмних методів для розв'язання економічних задач.

Після опанування дисципліни «Програмування на Python» здобувачі освіти набудуть таких програмних результатів навчання:

1. Застосовувати відповідні економіко-математичні методи та моделі для вирішення економічних задач.

2. Ідентифікувати джерела та розуміти методологію визначення і методи отримання соціально-економічних даних, збирати та аналізувати необхідну інформацію, розраховувати економічні та соціальні показники.
3. Вміти використовувати дані, надавати аргументацію, критично оцінювати логіку та формувати висновки з наукових та аналітичних текстів з економіки.
4. Використовувати інформаційні та комунікаційні технології для вирішення соціально-економічних завдань, підготовки та представлення аналітичних звітів.
5. Вміти абстрактно мислити, застосовувати аналіз та синтез для виявлення ключових характеристик економічних систем різного рівня, а також особливостей поведінки їх суб'єктів.
6. Демонструвати розуміння взаємозв'язку між перебігом технологічних, організаційних та інших процесів та економічними показниками під час аналітичного супроводу розробки і реалізації проєктів розвитку бізнес-діяльності.
7. Аналізувати та моделювати бізнес-процеси компанії на основі дослідження закономірностей у великих обсягах даних з використанням передових методологій та цифрових інструментів.

Очікується, що після завершення курсу студенти матимуть змогу ефективно використовувати Python для аналізу даних, створення інтерактивних звітів та автоматизації обробки великих масивів інформації.

**Критерії оцінювання**

Максимальні 5 балів за практичною роботою передбачають, що студент:

- підготував і завантажив звіт з виконання практичної роботи, що виконано у повній відповідності до поставлених завдань, у т.ч. індивідуального характеру (2 бали);

- дав пряму і релевантну відповідь на поставлене питання щодо виконаного завдання, у т.ч. у вигляді додаткових запитань / зміг стисло формалізувати вербально сутність проблеми за ситуацією, ідентифікувати ключові складові і пріоритети вирішення, запропонував логічне розв'язання (3 бали)

Виконання та захист індивідуального завдання: підготовлене аналітичне завдання у вигляді файлу *.docx, *.ipynb, *.pdf розміщується у відповідному розділі дисципліни в Moodle і перевіряється протягом тижня після завершення терміну подачі. Оскарження оцінки може бути здійснене на останньому практичному занятті модуля. Невчасно складене

Максимальна оцінка 15 балів за індивідуальну роботу виставляється студенту:

який підготував завдання (звіт) за ситуаційним завданням, в якому: правильно визначив проблеми, комплекс факторів, які могли вплинути на їх виникнення, обґрунтував своє бачення теоретичними концепціями або моделями, виконав необхідні розрахунки в разі потреби, представив висновок або власне бачення виходу з проблеми і окреслив можливі перспективи і обмеженість такого рішення; завдання структуровані, викладені діловим, науковим або публіцистичним стилем української (або часткового, англійської) мови з використанням вивчених методів підготовки аналітичних звіті (5 балів);

оформив аналітичний звіт містить комплексну, логічну і оригінальну розв'язку поставлених завдань аж до міждисциплінарного підходу; використання штучного інтелекту (ШІ) не забороняється, оскільки пропозиції відомих застосунків ШІ суттєво залежать від обміркованої постановки питання і уточнюючих питань; однак в разі, якщо відповідь, отримана з використанням ШІ, не є комплексною або не відповідає за стилем і викладеними позиціями іншим частинам завдання, містить очевидно неправдиву інформацію, то оцінка за цим критерієм знижується (5 балів)

під час презентації / захисту аналітичного звіту демонструє володіння термінологічним апаратом, відповідає на запитання, здатний швидко адаптувати позицію під зміни у вихідному ситуаційному завданні (5 бали)

# Практична робота №1.
# Встановлення та налаштування інтерпретаторів Python.
# Базові концепції мови програмування Python.

**Мета -** визначити ключові інтерпретатори Python та ознайомитися з основними концепціями використання мови програмування Python.

**Терміни виконання практичної роботи**

Практична робота виконується протягом двох занять:
**Практичне заняття 1** — виконання завдань 1–3.
**Практичне заняття 2** — виконання завдань 4–6, завантаження звіту та захист роботи.

**Завдання 1. Знайомство та огляд наповнення Anaconda**

Anaconda — це популярний дистрибутив Python з відкритим кодом, широко використовується в науці про дані та машинному навчанні. Він включає численні попередньо встановлені бібліотеки, такі як NumPy, Pandas, Matplotlib, а також зручний графічний інтерфейс Anaconda Navigator та середовище Spyder IDE.

**Зміст завдання**

Встановіть Anaconda на свій комп'ютер. Для цього перейдіть на сайт Anaconda і завантажте відповідний інсталятор для вашої операційної системи.

Створіть нове середовище в Anaconda. Середовища дозволяють ізолювати інсталяцію Python і окремо керувати залежностями для різних проектів. Щоб створити нове середовище, відкрийте Anaconda Navigator і натисніть «Середовища». Потім натисніть «Створити» та введіть назву для нового середовища.

Встановіть пакети у новому середовищі. Пакети — це додаткові бібліотеки або інструменти, які можна встановити для розширення функціональності Python. Щоб установити пакети, виберіть своє нове середовище в Anaconda Navigator і натисніть «Відкрити термінал». Потім введіть «`conda install [package_name]`.», щоб установити пакет.

Запустіть Spyder. Spyder — це популярне середовище розробки для аналізу даних Python, яке входить до складу Anaconda. Щоб запустити Spyder, виберіть нове середовище в Anaconda Navigator і натисніть «Відкрити за допомогою Spyder».

За відсутності можливості встановлення на локальний комп’ютер ознайомтеся з апаратними вимогами Anaconda та її складовими (Anaconda Navigator, Anaconda Prompt).

**Завдання 2. Налаштування Google Colab і обмін результатами**

Google Colab — це безкоштовна хмарна платформа, яка надає середовище Python для аналізу даних. Вона включає підтримку популярних бібліотек, таких як TensorFlow, PyTorch, scikit-learn.

**Зміст завдання**

Налаштування облікового запису Google Colab. Щоб використовувати Colab, вам потрібен обліковий запис Google. Якщо у вас його ще немає, ви можете створити його безкоштовно на сторінці реєстрації Google.

Створення нового блокнота Colab. Щоб створити новий блокнот Colab, перейдіть на домашню сторінку Google Colab і натисніть «Новий блокнот». Потім ви можете ввести свій код Python у блокнот і запустити його в хмарі.

Спільний доступ до блокнота Colab. Щоб поділитися блокнотом Colab з іншими, ви можете надати їм посилання на блокнот або поділитися блокнотом із певними співавторами. Щоб поділитися посиланням, натисніть «Поділитися» у верхньому правому куті блокнота та скопіюйте посилання. Щоб поділитися з певними співавторами, натисніть «Додати співавтора» та введіть їх електронну адресу.

Спільна робота над записником Colab. Colab пропонує прості способи спільної роботи над записниками з іншими, зокрема редагування в реальному часі, коментарі та спілкування в чаті. Щоб спільно працювати над блокнотом, надайте спільний доступ до блокнота своїм співавторам і дозвольте їм редагувати блокнот.

Загалом, Google Colab надає зручне та потужне середовище для аналізу даних Python і є чудовим інструментом для навчання. Це дозволяє легко ділитися та співпрацювати, а також надає потужний обчислювальний ресурс для аналізу даних

**Завдання 3. Markdown у Colab та Jupyter Notebook**

Ознайомитись з Markdown і його призначення в блокнотах Colab. Сформуйте короткий посібник із використання Markdown для форматування тексту та додавання зображень до блокнота Colab

Markdown — це мова розмітки для форматування тексту. Вона дозволяє створювати добре оформлені документи у блокнотах Jupyter.

Основні елементи Markdown

**Заголовки**. `# Заголовок першого рівня, ## Заголовок другого рівня.`

**Форматування тексту**. `**Жирний текст**, *Курсив*.`

**Списки**.

Марковані: `- Пункт`.

Нумеровані: `1. Пункт`.

**Додавання зображень**. `![Alt text](URL)`.

**Додавання посилань**. `[Текст посилання](URL)`.

**Зміст завдання.**

Додавання заголовка. Щоб додати заголовок у Markdown, використовуйте символ «#», а потім текст вашого заголовка. Наприклад, «# Це заголовок» створить заголовок верхнього рівня у вашому блокноті.

Додавання жирного та курсивного тексту. Щоб додати жирний текст у Markdown, використовуйте символ подвійної зірочки перед і після тексту, який потрібно виділити жирним. Наприклад, «Цей текст напівжирним» створить жирний текст. Щоб додати текст курсивом, використовуйте одну зірочку перед і після тексту.

Додавання списків. Щоб додати нумерований список у Markdown, використовуйте число, після якого крапка, потім пробіл, а потім текст вашого елемента списку. Наприклад, «1. Перший елемент» створить пронумерований пункт списку. Щоб створити маркований список, скористайтеся тире та пробілом, а потім текстом елемента списку.

Додавання зображень. Щоб додати зображення в Markdown, використовуйте "!" символ, за яким іде альтернативний текст у дужках, а потім URL-адреса зображення в дужках. Наприклад, «Alt text» створить зображення з альтернативним текстом «Alt text» і URL-адресою.

Додавання посилань. Щоб додати посилання в Markdown, використовуйте текст посилання в квадратних дужках, а потім URL-адресу в дужках. Наприклад, «Натисніть тут» створить посилання з текстом «Натисніть тут» і URL-адресою «https://example.com».

Загалом Markdown є корисним інструментом для форматування тексту в блокнотах Colab. Освоївши Markdown, ви зможуть створювати більш професійні та візуально привабливі зошити, які легше читати та розуміти.

***Приклад*** *представлення результатів виконання завдань за допомогою Markdown:*

# Python Programming for Data Analysis: Lab 1

## Introduction and Overview of Anaconda

In this lab, you will learn about Anaconda, a popular open-source distribution of Python that is widely used in data science and machine learning. You will explore its features and components, and learn how to install and use it on your own computer.

### Tasks:

- Learn what Anaconda is and its purpose in Python data analysis
- Describe the main components of Anaconda, including the Anaconda Navigator, Anaconda Prompt, and Spyder IDE
- Install Anaconda on your computer
- Create a new environment in Anaconda
- Install packages in your new environment
- Launch Spyder

### Authority

This lab was created by [Your Name], a Python expert and data scientist with over 10 years of experience in the field.

## Configuration of Google Colab and Results Sharing

In this section, you will learn about Google Colab, a free cloud-based platform that provides a Python environment for data analysis. You will explore its features and learn how to create, share, and collaborate on Colab notebooks.

**Завдання 4. Використання операторів Print, For, If**

Надайте огляд оператора друку та того, як його можна використовувати для відображення результату в Python. Ознайомтись з операторами for і if і тим, як їх можна використовувати для умовних і

ітеративних операцій у Python. Наведіть приклади простих операцій for і if, таких як ітерація в діапазоні чисел або перевірка парного чи непарного числа

**Зміст завдання.**

У цьому завданні треба ознайомитись з деякими основними концепціями програмування на Python, зокрема оператором друку, циклами for та операторами if. Ці концепції є основними будівельними блоками для більш просунутих методів аналізу даних Python, і оволодіння ними допоможе вашим учням навчитися програмувати на Python.

Використання оператора друку. Інструкція print використовується для відображення виведення в Python. Наприклад, наступний код відобразить рядок "Hello, world!" на екрані:

```
print("Привіт, світ!")
```

Використання циклів for. Цикл for використовується для повторення послідовності значень у Python. Наприклад, наведений нижче код буде виконувати ітерацію в діапазоні чисел від 0 до 4 і відображати кожне число на екрані:

```
for i in range(5):
    print(i)
```

Використання операторів if. Інструкція if використовується для виконання коду лише за умови виконання певної умови. Наприклад, наступний код перевірить, чи змінна x перевищує 10, і відобразить на екрані повідомлення, якщо це так:

```
x = 15
if x > 10:
    print("x більше 10")
```

Приклад: використання циклу for для перебору списку чисел з урахуванням певних умов (оператор if) є задача перевірки парності чи непарності кожного числа:

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
for num in numbers:
    if num % 2 == 0:
        print(num, "is even")
    else:
        print(num, "is odd")
```

Виконання інших базових функцій Python: перевірте всі блоки коду, що представлені у слайдах лекцій на практиці.

Загалом, оволодіння цими основними концепціями програмування допоможе навчитися зручніше працювати з Python і закладе основу для вдосконалених методів аналізу даних.

**Завдання 5. Робота з типами даних**

Знайомство з основними типами даних у Python, включаючи цілі числа, числа з плаваючою точкою, логічні значення та символи (рядки). Оголошення та присвоювання значень змінним кожного типу даних. риклади простих операцій з кожним типом даних, таких як арифметика з цілими числами та числами з плаваючою точкою та логічні операції з булевими значеннями

**Зміст завдання.**

У цьому проводиться практичне ознайомлення з основними типами даних у Python, включаючи цілі числа, числа з плаваючою крапкою, логічні значення та символи (рядки), а також щодо оголошення та присвоєння значень змінним кожного типу даних

Integers. Цілі числа — це цілі числа, які можуть бути додатними, від'ємними або нульовими. Щоб оголосити цілочисельну змінну в Python, просто призначте значення імені змінної. Наприклад, наступний код оголошує цілочисельну змінну x зі значенням 5:

```
x = 5
```

Floats. Числа з плаваючою крапкою — це десяткові числа, які можуть бути додатними, від'ємними або нульовими. Щоб оголосити змінну float у Python, використовуйте десяткову кому в числі. Наприклад, наступний код оголошує змінну y зі значенням 3.14:

```
y = 3.14
```

Booleans. Логічні значення – це двійкові значення, які можуть бути істинними або хибними. Щоб оголосити логічну змінну в Python, використовуйте ключові слова True або False. Наприклад, наступний код оголошує логічну змінну z зі значенням True:

```
z = True
```

Characters та Strings. Символи – одиничні нечислові значення, рядки – являють собою послідовності символів. Щоб оголосити рядкову (String) змінну в Python, використовуйте лапки навколо символів. Наприклад, наступний код оголошує рядкову змінну w зі значенням “Hello, world!”:

```
w = "Hello, world!"
```

або

```
w = f"Привіт, світ!"
```

Комбінуючи вказані типи даних і змінні можа виконувати складніші операції в Python. Наприклад, можна оголосити цілу змінну для кількості яблук у кошику, змінну з плаваючою речовиною для ціни за яблуко та використовувати арифметичні операції для обчислення загальної вартості яблук, як це:

```
apples = 10
price = 0.5
total_cost = apples * price
print("The total cost of the apples is:", total_cost)
```

або

```
total_cost = apples * price
text = f"The total cost of the apples is: {total_cost}"
```

Загалом, оволодіння цими елементарними типами даних і змінними дає можливість навчитися зручніше працювати з Python і підготуватися до більш просунутих методів аналізу даних, які включають складніші типи даних і структури.

**Завдання 6. Кодування символів і проблема з кирилицею**

Поясніть концепцію кодування символів і як воно впливає на відображення тексту в Python. Опишіть найпоширеніші схеми кодування символів, включаючи ASCII та Unicode. Наведіть приклади проблеми з кириличними символами в Python, включаючи помилки кодування та проблеми з відображенням.

У цьому завданні необхідно ще раз зануритись у проблеми кодування символів у Python, які можуть виникнути під час роботи з символами, що не належать до ASCII, наприклад символами кирилиці.

Зміст завдання.

У Python символи представлені за допомогою набору символів Unicode, який призначає унікальну кодову точку кожному символу. Найпоширенішим кодуванням для Unicode є UTF-8, яке використовується за замовчуванням у Python 3.

Оголошення рядка з символами, відмінними від ASCII. Щоб оголосити рядок із символами, відмінними від ASCII, наприклад

символами кирилиці, використовуйте відповідний код Unicode для кожного символу. Наприклад, наступний код оголошує рядкову змінну з кириличним словом "Привет, мир!", що означає "Привіт, світ!":

```
hello_world = "\u041f\u0440\u0438\u0432\u0456\u0442
\u0441\u0432\u0456\u0442"
print(hello_world)
```

Читання та запис файлів із символами, відмінними від ASCII. Під час роботи з файлами, які містять символи, відмінні від ASCII, важливо вказати правильне кодування символів. Наприклад, щоб прочитати файл із кодуванням UTF-8, використовуйте такий код:

```
with open("filename.txt", "r", encoding="utf-8") as f:
    contents = f.read()
```

Щоб написати файл із кодуванням UTF-8, використовуйте такий код:

```
with open("filename.txt", "w", encoding="utf-8") as f:
    f.write("Привет, мир!")
```

Обробка помилок із символами, відмінними від ASCII. Іноді під час роботи з символами, відмінними від ASCII, можуть виникати помилки, якщо використовується неправильне кодування. Для обробки цих помилок можна використовувати параметри обробки помилок, надані Python. Наприклад, щоб ігнорувати помилки під час читання файлу, використовуйте такий код:

```
with open("filename.txt", "r", encoding="utf-8",
errors="ignore") as f:
    contents = f.read()
```

Загалом, розуміння кодування символів і того, як працювати з символами, відмінними від ASCII, є важливою навичкою для аналітиків даних Python, особливо при роботі з багатомовними даними.

**Питання для самоаналізу**

1. У чому переваги використання Anaconda?
2. Як працювати з Google Colab для аналізу даних?
3. Які можливості надає Markdown у блокнотах Jupyter?
4. Як визначити та використовувати типи даних у Python?
5. Які поширені проблеми з кодуванням символів у Python?

# Практична робота №2.
# Структури даних та масиви у Python.

**Мета - навчитися** працювати з масивами даних за допомогою стандартних функцій Python.

### Терімни виконання практичної роботи

Практична робота виконується протягом двох занять:
**Практичне заняття 1** — виконання завдань 1–3.
**Практичне заняття 2** — виконання завдань 4–5, завантаження звіту та захист роботи.

### Завдання 1. Створення списків та обробка даних

Створіть списки зі значеннями, пов’язаними з економікою, та заповніть їх випадковими значеннями, що розподілені за нормальним законом розподілу. Використовуйте бібліотеку `random` та цикли `for` для генерації даних. Об'єднайте списки у вигляді пар ключ-значення, використовуючи кортежі, та виведіть кожен список як текстовий рядок.

**Зміст завдання.**

Для генерації випадкових значень скористайтеся функцією `random.normalvariate(mu, sigma)`, де `mu` — середнє значення, а `sigma` — стандартне відхилення. Наприклад, щоб створити список рівня інфляції, можна використати наступний код:

```
from random import normalvariate
inflation = [round(normalvariate(1.0, 0.05), 2) for _
in range(15)]
```

Аналогічно створіть інші списки, такі як **валовий внутрішній продукт (ВВП)** та **рівень безробіття**, за допомогою функції `normalvariate`. Для ВВП встановіть середнє значення `100` і стандартне відхилення `12`:

```
gdp = [round(normalvariate(100, 12), 2) for _ in
range(15)]
```

Для рівня безробіття середнє значення має дорівнювати `20`, а стандартне відхилення — `5`:

```
unemployment = [round(normalvariate(20, 5), 2) for _ in
range(15)]
```

Згенеровані дані об'єднайте у формат таблиці за допомогою функції `zip`. Кожен рядок таблиці міститиме значення рівня інфляції, ВВП та рівня безробіття. Приклад об'єднання даних:

```
data_table = list(zip(inflation, gdp, unemployment))
```

Для виведення отриманих даних у вигляді таблиці використовуйте наступний код:

```
print("| Inflation | GDP | Unemployment |")
print("| --- | --- | --- |")
for row in data_table:
    print(f"| {row[0]} | {row[1]} | {row[2]} |")
```

Це дозволить відобразити результати у зручному табличному форматі, де кожен стовпець відповідає окремому показнику.

**Завдання 2. Обчислення описової статистики**

Обчисліть параметри описової статистики (мінімум, максимум, сума, кількість, середнє, медіана, дисперсія) для кожного списку даних.

**Зміст завдання.**

Для обчислення основних статистичних характеристик списку, таких як мінімум, максимум, сума, середнє значення, медіана та дисперсія, використовуйте наступний підхід.

Для мінімального, максимального значень, суми та середнього значення:

```
min_val = min(inflation)
max_val = max(inflation)
total = sum(inflation)
mean = total / len(inflation)
```

Для медіани спочатку відсортуйте список, а потім визначте середнє значення для середнього елемента або двох середніх елементів (залежно від кількості елементів у списку):

```
sorted_list = sorted(inflation)
n = len(sorted_list)
median = (sorted_list[n // 2] if n % 2 != 0 else
(sorted_list[n // 2 - 1] + sorted_list[n // 2]) / 2)
```

Для обчислення дисперсії знайдіть квадрат відхилень від середнього значення та обчисліть їх середнє значення:

```
variance = sum((x - mean) ** 2 for x in inflation) /
(len(inflation) - 1)
```

Виведіть обчислені значення у форматі описової статистики:

```
print(f"Min: {min_val}, Max: {max_val}, Mean:
{mean:.2f}, Median: {median}, Variance:
{variance:.2f}")
```

Аналогічні дії можна виконати для інших списків, таких як **gdp** та **unemployment**. Це дозволить отримати мінімум, максимум, середнє значення, медіану та дисперсію для кожного з показників.

**Завдання 3. Робота з логічними значеннями**

Створіть списки логічних значень, що базуються на порівняннях між значеннями списків із Завдання 1. Наприклад, визначте, які значення перевищують середнє значення ВВП. Відфільтруйте списки на основі цих логічних значень.

**Зміст завдання.**

Для створення логічних списків на основі порівнянь значень зі списків, обчислених у Завданні 1, виконайте наступні дії.

Спочатку створіть логічний список, який визначає, чи перевищує кожне значення **ВВП** середнє значення:

```
gdp_above_mean = [value > mean for value in gdp]
```

Де `mean` — середнє значення зі списку `gdp`, обчислене раніше.

Далі використовуйте логічний список для фільтрації значень **ВВП**. Застосовуйте функцію `zip` для об'єднання значень зі списку `gdp` з відповідними логічними значеннями зі списку `gdp_above_mean`. Фільтрація відбувається лише для тих значень, де умова є **істинною**:

```
filtered_gdp = [value for value, condition in zip(gdp,
gdp_above_mean) if condition]
```

Нарешті, виведіть відфільтровані значення **ВВП**:

```
print(filtered_gdp)
```

Цей підхід можна аналогічно використати для інших списків, замінюючи значення `gdp` на потрібний список, наприклад, **інфляцію** або **рівень безробіття**, та використовуючи відповідне середнє значення.

**Завдання 4. Виконання завдань для словників**

Перетворіть списки з Завдань 1–3 на словники. Для цього використовуйте функцію `zip` для об’єднання назв показників (наприклад, "Інфляція", "ВВП", "Безробіття") зі списками значень. Виконайте всі розрахунки та фільтрацію для цих словників, супроводжуючи результати коментарями та висновками.

**Зміст завдання.**

Для перетворення списків зі **Завдань 1–3** на словники та подальших обчислень виконайте наступні кроки.

Створення словника

Спочатку об'єднайте списки у словник, де ключами будуть назви показників, а значеннями — відповідні списки:

```
data_dict = {
    'Inflation': inflation,
    'GDP': gdp,
    'Unemployment': unemployment
}
```

Обчислення описової статистики

Для кожного ключа у словнику обчисліть середнє значення та виведіть його:

```
for key, values in data_dict.items():
    mean = sum(values) / len(values)
    print(f"{key} - Mean: {mean:.2f}")
```

Фільтрація значень на основі середнього

Виконайте фільтрацію значень для кожного ключа, залишивши лише ті значення, які перевищують середнє значення:

```
filtered_dict = {key: [v for v in values if v >
(sum(values) / len(values))] for key, values in
data_dict.items()}
print(filtered_dict)
```

Результат та коментарі

Перший етап створює словник `data_dict`, який структурує дані у вигляді "назва показника: список значень".

Другий етап розраховує середнє значення для кожного списку та відображає його на екрані.

Третій етап формує новий словник `filtered_dict`, що містить лише ті значення, які перевищують середнє.

Приклад виводу

```
Inflation - Mean: 1.02
GDP - Mean: 100.12
Unemployment - Mean: 19.98
{'Inflation': [1.05, 1.08, 1.03], 'GDP': [110.5,
102.3], 'Unemployment': [21.4, 20.5]}
```

У підсумку, фільтрований словник міститиме значення, які є вищими за середнє для кожного показника.

Завдання 5. Робота з текстовими файлами

Збережіть словники у текстовий файл із розширенням CSV. Виконайте перетворення словника зі списками даних на список словників. Напишіть коментарі, які пояснюють формат створеного файлу, та поясніть, як його можна відкрити у текстовому редакторі або програмі для роботи з електронними таблицями.

Зміст завдання.

Для створення CSV-файлу та запису даних у нього виконайте наступні кроки.

Створення файлу `data.csv`

Використовуйте вбудовану функцію `open()` для створення файлу та запису заголовків стовпців разом із даними:

```
with open('data.csv', 'w') as file:
    file.write('Inflation,GDP,Unemployment\n')
    for i, g, u in zip(inflation, gdp, unemployment):
        file.write(f"{i},{g},{u}\n")
```

Пояснення

**`open('data.csv', 'w')`**: Відкриває файл для запису (режим `'w'` означає "write"). Якщо файл уже існує, його вміст буде перезаписано.

**`file.write()`**: Додає рядки у файл.

Перший рядок — заголовки стовпців: `Inflation`, `GDP`, `Unemployment`.

Подальші рядки містять дані, об'єднані функцією `zip` для синхронного ітераційного перебору значень зі списків `inflation`, `gdp` та `unemployment`.

Перевірка файлу

Після виконання коду:

**Відкрийте файл `data.csv`** у будь-якому текстовому редакторі або програмі, такій як **Excel**.

У Excel дані автоматично відобразяться у форматі таблиці зі стовпцями: `Inflation`, `GDP`, `Unemployment`.

**Приклад результату у файлі `data.csv`:**

```
Inflation,GDP,Unemployment
1.01,102.3,20.1
1.02,98.7,19.5
1.05,110.2,21.0
```

Файл готовий для подальшого аналізу та обробки.

**Питання для самоаналізу**

1. Як використовувати функції Python для генерації випадкових значень?
2. Що таке описова статистика, і як її обчислити?
3. Як використовувати логічні списки для фільтрації даних?
4. Які переваги надає робота зі словниками в Python?
5. Як зберігати дані у текстові файли та відкривати їх у електронних таблицях?

## Практична робота №3.
## Функції, модулі та основи ООП.

**Мета -** навчитися працювати з елементами аналітичного коду, що повторюється, та вивчити основи роботи з модулем NumPy.

**Терміни виконання практичної роботи**

Практична робота виконується протягом трьох занять:

- **Практичне заняття 1** — виконання завдань 1–2.
- **Практичне заняття 2** — виконання завдань 3–4, завантаження звіту та захист роботи.

**Завдання 1. Функції Python**

Напишіть функцію на Python, яка приймає список вхідних даних і обчислює середнє, медіану, дисперсію, стандартне відхилення та квартилі, використовуючи лише базові модулі Python. Функція повинна повертати значення відповідного показника описової статистики та друкувати результати.

**Зміст завдання.**

**Показники описової статистики** дозволяють аналізувати набір даних, характеризуючи його основні властивості. Для реалізації основних показників статистики створюються відповідні функції.

**Середнє значення (Mean)** — це середнє арифметичне значення списку даних. Для його обчислення необхідно знайти суму всіх елементів списку та поділити її на кількість елементів. Функція виглядає наступним чином:

```
def mean(val):
    return sum(val) / len(val)
```

**Медіана (Median)** — це середнє значення відсортованого списку даних. Для її обчислення список сортується, а потім знаходиться середнє значення для одного елемента (якщо кількість елементів непарна) або двох середніх елементів (якщо кількість елементів парна):

```
def median(val):
    n = len(val)
    sorted_val = sorted(val)
    midpoint = n // 2
    if n % 2:
        return sorted_val[midpoint]
    else:
        return (sorted_val[midpoint - 1] +
sorted_val[midpoint]) / 2
```

**Дисперсія (Variance)** — це показник, який обчислює середній квадрат відхилень кожного значення від середнього значення. Формула враховує середнє значення, після чого для кожного елемента обчислюється квадрат різниці від середнього:

```
def variance(val):
    m = mean(val)
    return sum((x - m) ** 2 for x in val) / (len(val) -
1)
```

**Стандартне відхилення (Standard Deviation)** — це корінь квадратний з дисперсії. Воно показує, наскільки значення у списку відхиляються від середнього значення:

```
def std_dev(val):
    return variance(val) ** 0.5
```

**Квартилі (Quartiles)** — це значення, що ділять відсортований список даних на чотири рівні частини. Для визначення значення квартиля слід знайти відповідний індекс у відсортованому списку:

```
def quartiles(val, n):
    sorted_val = sorted(val)
    q_index = int(n * len(sorted_val) / 4)
    return sorted_val[q_index]
```

**Приклад застосування функцій** для набору даних, який представляє рівень інфляції:

```
inflation = [1.03, 1.02, 1.05, 1.01, 1.07]

print("Середнє значення:", mean(inflation))
print("Медіана:", median(inflation))
print("Дисперсія:", variance(inflation))
print("Стандартне відхилення:", std_dev(inflation))
print("Квартиль Q1:", quartiles(inflation, 1))
```

**Пояснення результату**:

**Середнє значення** покаже середню величину серед усіх значень.

**Медіана** визначить центральне значення у відсортованому списку.

**Дисперсія** вкаже на ступінь розкиду значень відносно середнього.

**Стандартне відхилення** буде корисним для інтерпретації дисперсії у тих самих одиницях, що й початкові дані.

**Квартиль Q1** поділить дані таким чином, що 25% значень будуть меншими або дорівнюватимуть цьому значенню.

Результати застосування функцій дозволяють глибше проаналізувати дані та визначити їх характерні особливості.

### Завдання 2. Модулі Python

Створіть власний модуль, який містить функції для обчислення показників описової статистики. Імпортуйте цей модуль у основну програму та використовуйте його функціональність для аналізу мінімум трьох змінних.

**Зміст завдання.**

Для створення власного модуля **mystats.py**, що містить функції для обчислення показників описової статистики, а також його використання в основній програмі, виконайте наступні кроки.

Крок 1. Створення модуля mystats.py

Створіть новий файл `mystats.py` і додайте до нього функції для обчислення **середнього значення**, **дисперсії** та **стандартного відхилення**:

```
# mystats.py

def mean(val):
    return sum(val) / len(val)

def variance(val):
    m = mean(val)
    return sum((x - m) ** 2 for x in val) / (len(val) -
1)

def std_dev(val):
    return variance(val) ** 0.5
```

Цей модуль містить три функції:

**mean(val)** — обчислює середнє значення.

**variance(val)** — обчислює дисперсію на основі середнього значення.

**std_dev(val)** — обчислює стандартне відхилення як корінь з дисперсії.

Крок 2. Імпортування модуля в основну програму

Створіть основний файл програми, де ви імпортуєте модуль **mystats** та використовуєте його функції для аналізу даних.

```
# main.py
import mystats

# Дані для аналізу: три змінні
inflation = [1.01, 1.02, 1.05, 1.04, 1.03]
gdp = [100, 102, 98, 101, 99]
unemployment = [20.1, 19.5, 20.0, 21.0, 19.8]

# Використання функцій модуля для кожної змінної
print("Показники для Inflation:")
```

```
print("Середнє значення:", mystats.mean(inflation))
print("Дисперсія:", mystats.variance(inflation))
print("Стандартне відхилення:",
mystats.std_dev(inflation))

print("\nПоказники для GDP:")
print("Середнє значення:", mystats.mean(gdp))
print("Дисперсія:", mystats.variance(gdp))
print("Стандартне відхилення:", mystats.std_dev(gdp))

print("\nПоказники для Unemployment:")
print("Середнє значення:", mystats.mean(unemployment))
print("Дисперсія:", mystats.variance(unemployment))
print("Стандартне відхилення:",
mystats.std_dev(unemployment))
```

Крок 3. Запуск програми

Запустіть файл **main.py**. Приклад результату для даних:

```
Показники для Inflation:
Середнє значення: 1.03
Дисперсія: 0.0003
Стандартне відхилення: 0.02

Показники для GDP:
Середнє значення: 100.0
Дисперсія: 2.5
Стандартне відхилення: 1.58

Показники для Unemployment:
Середнє значення: 20.08
Дисперсія: 0.36
Стандартне відхилення: 0.60
```

Пояснення

**Модуль `mystats.py`** містить функції для розрахунку основних показників описової статистики.

**Імпорт модуля** у програму здійснюється через `import mystats`.

Дані аналізуються для трьох змінних: **Inflation**, **GDP** та **Unemployment**.

Результати розрахунків наочно відображають середнє значення, дисперсію та стандартне відхилення для кожного набору даних.

Цей підхід дозволяє легко повторно використовувати модуль для інших наборів даних або програм.

**Завдання 3. Об'єктно-орієнтоване програмування**

Створіть клас `DescriptiveStatistics`, який реалізує функціональність модуля. Клас повинен мати методи для обчислення середнього, медіани, моди, дисперсії, стандартного відхилення, квартилів та коефіцієнта варіації. Включіть метод `.fit` для розрахунку всіх показників описової статистики та метод `.summary` для відображення результатів.

**Зміст завдання.**

**Реалізація класу `DescriptiveStatistics` та розширених функцій з використанням NumPy**

Крок 1: Створення класу `DescriptiveStatistics`

Клас реалізує функції для обчислення **середнього значення**, **медіани**, **моди**, **дисперсії**, **стандартного відхилення**, **квартилів** та **коефіцієнта варіації**.

```
import math # для швидкого розрахунку логарифму

class DescriptiveStatistics:
    def __init__(self, data):
        self.data = data

    def mean(self):
        return sum(self.data) / len(self.data)

    def median(self):
        sorted_data = sorted(self.data)
        n = len(sorted_data)
        if n % 2:
            return sorted_data[n // 2]
        else:
            return (sorted_data[n // 2 - 1] +
sorted_data[n // 2]) / 2

def mode(data):
    frequency = {}
    for value in data:
        if value in frequency:
            frequency[value] += 1
        else:
            frequency[value] = 1
    max_freq = max(frequency.values())
```

```
        modes = [key for key, val in frequency.items() if
val == max_freq]
        return modes

    def variance(self):
        m = self.mean()
        return sum((x - m) ** 2 for x in self.data) /
(len(self.data) - 1)

    def std_dev(self):
        return self.variance() ** 0.5

    def quartiles(self, q):
        sorted_data = sorted(self.data)
        index = int(q * len(sorted_data) / 4)
        return sorted_data[index]

    def coefficient_of_variation(self):
        return self.std_dev() / self.mean()

    def fit(self):
        self.stats = {
            "Mean": self.mean(),
            "Median": self.median(),
            "Mode": self.mode(),
            "Variance": self.variance(),
            "Standard Deviation": self.std_dev(),
            "Q1": self.quartiles(1),
            "Q3": self.quartiles(3),
            "Coefficient of Variation":
self.coefficient_of_variation()
        }

    def summary(self):
        if not hasattr(self, "stats"):
            self.fit()
        return self.stats
```

Крок 2. Створення підкласу `ExtendedStatistics`

Підклас реалізує додатковий метод для обчислення **середньої геометричної**:

```
class ExtendedStatistics(DescriptiveStatistics):
    def geometric_mean(self):
```

```
        return math.exp(sum(math.log(x) for x in
self.data) / len(self.data))
```

Крок 3. Реалізація з використанням NumPy

Додайте новий клас, який використовує NumPy для обчислень, що значно підвищує продуктивність:

```
import numpy as np

class NumPyStatistics(DescriptiveStatistics):
    def mean(self):
        return np.mean(self.data)

    def variance(self):
        return np.var(self.data, ddof=1)

    def std_dev(self):
        return np.std(self.data, ddof=1)

    def median(self):
        return np.median(self.data)

    def quartiles(self, q):
        return np.percentile(self.data, q * 25)

    def mode(self):
        counts = np.bincount(self.data)
        return np.flatnonzero(counts == counts.max())
```

Крок 4. Використання класів у програмному коді

```
# Основна програма
data = [1, 2, 3, 4, 4, 5, 5, 5]

# Використання базового класу
stats = DescriptiveStatistics(data)
print("Descriptive Statistics:", stats.summary())

# Використання розширеного класу
ext_stats = ExtendedStatistics(data)
print("Geometric Mean:", ext_stats.geometric_mean())

# Використання класу з NumPy
numpy_stats = NumPyStatistics(np.array(data))
print("NumPy Descriptive Statistics:",
numpy_stats.summary())
```

```
# Збереження результатів у файл
np.savetxt("data.csv", np.array(data), delimiter=",")
```

Пояснення коду

**Клас** **`DescriptiveStatistics`** містить методи для основних показників статистики.

**Підклас** **`ExtendedStatistics`** додає функціональність для обчислення середньої геометричної.

**Клас** **`NumPyStatistics`** реалізує всі обчислення з використанням бібліотеки NumPy.

**Результати** виводяться у структурованому форматі за допомогою методу `.summary`.

Дані зберігаються у CSV-файлі за допомогою `np.savetxt`.

**Результат реалізації програми**:

```
Descriptive Statistics: {'Mean': 3.625, 'Median': 4,
'Mode': [5], 'Variance': 1.696, 'Standard Deviation':
1.303, 'Q1': 2, 'Q3': 5, 'Coefficient of Variation':
0.3595}
Geometric Mean: 3.277
NumPy Descriptive Statistics: {'Mean': 3.625, 'Median':
4.0, 'Mode': [5], 'Variance': 1.696, 'Standard
Deviation': 1.303, 'Q1': 2.25, 'Q3': 4.75, 'Coefficient
of Variation': 0.3595}
```

**Питання для самоаналізу**

1. Як створити та використовувати функції Python для обчислення описової статистики?
2. Як створювати модулі та імпортувати їх у Python-програми?
3. Що таке об'єктно-орієнтоване програмування та які його основні концепції?
4. Як бібліотека NumPy полегшує аналіз даних?
5. Як зберігати дані у формат NumPy та завантажувати їх у текстовий файл?

# Практична робота №4.
# Структури даних та масиви з Pandas.

**Мета -** навчитися працювати з масивами даних за допомогою функцій та методів Pandas (DataFrame).

Теміни **виконання** практичної роботи

Практична робота виконується протягом двох занять:
**Практичне заняття 1** — виконання завдань 1–3.
**Практичне заняття 2** — виконання завдань 4–5, завантаження звіту та захист роботи.

**Завдання 1. Основи створення та завантаження даних у DataFrame та Series**

Виконайте завдання, що представлені у лекції, щодо створення DataFrame та Series з різних джерел даних.

**Зміст завдання.**

Для роботи з бібліотекою **Pandas** виконуються основні операції зі створення, обробки та завантаження даних у вигляді структур **Series** та **DataFrame**.

Спочатку для створення **Series** зі словника імпортується бібліотека Pandas. Дані у словнику представлені у форматі пар "ключ-значення", де ключі стають індексами, а значення — даними серії. У прикладі використовується словник `data = {'John': 25, 'Jane': 30, 'Bob': 35}`. Після створення об'єкта `pd.Series`, серія виводиться на екран:

```
import pandas as pd

data = {'John': 25, 'Jane': 30, 'Bob': 35}
s = pd.Series(data, name='Names and Ages')
print(s)
```

**Результат**:

```
John    25
Jane    30
Bob     35
Name: Names and Ages, dtype: int64
```

У цьому прикладі ключі словника `John`, `Jane` і `Bob` перетворюються на індекси, а значення `25`, `30` і `35` стають елементами серії.

Наступним кроком демонструється створення **DataFrame** зі списку списків. Кожен вкладений список у загальному списку стає окремим рядком у таблиці, а для стовпців передаються відповідні назви. Використовується список `data_list = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]`, а назви стовпців — `['col1', 'col2', 'col3']`. Створений DataFrame виводиться на екран:

```
data_list = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
df = pd.DataFrame(data_list, columns=['col1', 'col2',
'col3'])
print(df)
```

**Результат**:

```
   col1  col2  col3
0     1     2     3
1     4     5     6
2     7     8     9
```

Кожен список `[1, 2, 3]`, `[4, 5, 6]` і `[7, 8, 9]` відповідає рядку таблиці, а передані назви `col1`, `col2` та `col3` стають заголовками стовпців.

Далі показується завантаження даних з **CSV-файлу**. Для цього використовується метод `pd.read_csv()` для читання файлу `data.csv`. Метод `.head()` виводить перші п'ять рядків таблиці для швидкої перевірки структури даних:

```
df = pd.read_csv('data.csv')
print(df.head())
```

**Результат** залежить від вмісту файлу `data.csv`, але виведення містить перші 5 рядків таблиці з усіма доступними стовпцями.

Останній приклад демонструє створення **DataFrame** зі словника списків. У словнику кожен ключ стає заголовком стовпця, а список значень відповідає рядкам цього стовпця. У прикладі використовується словник `data_dict`, де є три ключі: `Name`, `Age` і `City`. Створений DataFrame виводиться на екран:

```
data_dict = {
    'Name': ['John', 'Jane', 'Bob'],
    'Age': [25, 30, 35],
    'City': ['New York', 'Paris', 'London']
}
df = pd.DataFrame(data_dict)
```

```
print(df)
```

**Результат**:

```
   Name  Age      City
0  John   25  New York
1  Jane   30     Paris
2   Bob   35    London
```

У цьому прикладі ключі `Name`, `Age` і `City` стають заголовками стовпців у таблиці, а значення списків `['John', 'Jane', 'Bob']`, `[25, 30, 35]` та `['New York', 'Paris', 'London']` розподіляються по рядках таблиці.

Загалом ці завдання показують, як гнучко працювати з даними у **Pandas**, створювати об'єкти Series і DataFrame з різних форматів, а також завантажувати дані з зовнішніх джерел, таких як CSV-файли.

### Завдання 2. Базові операції з масивами даних

Для виконання завдань з попередніх практичних робіт з використанням бібліотеки **Pandas** слід застосувати методи для обробки, фільтрації та аналізу даних. Для роботи в рамках цього завдання необхідно знайти або завантажити власний масив даних (*dataset*).

Для додавання нового стовпця у вже існуючий **DataFrame** використовується операція, де значення стовпця обчислюються на основі даних інших стовпців. Припустимо, що в таблиці є стовпці `GDP` (валовий внутрішній продукт) та `Population` (населення). Новий стовпець `GDP per capita` буде містити значення ВВП на душу населення:

```
df['GDP per capita'] = df['GDP'] / df['Population']
```

Це додає новий стовпець, де кожне значення розраховується шляхом ділення значень у стовпці `GDP` на відповідні значення у стовпці `Population`.

Для фільтрації даних використовується умовна перевірка. У цьому прикладі обираються лише ті рядки, у яких значення інфляції у стовпці `Inflation` перевищує середнє значення в цьому стовпці. Метод `.mean()` обчислює середнє значення для стовпця:

```
filtered_df = df[df['Inflation'] >
df['Inflation'].mean()]
```

Фільтрований **DataFrame** `filtered_df` містить лише ті рядки, що відповідають заданій умові.

Для вибору конкретних рядків за індексами використовується метод `loc`. Припустимо, що потрібно отримати рядки за індексами від `2000` до `2005` роки включно (якщо у масиві даних індекси рядків відображають роки спостережень). Метод `loc` забезпечує доступ до рядків на основі міток індексів:

```
subset = df.loc[2000:2005]
print(subset)
```

Виведений **subset** буде містити всі рядки з індексами у заданому діапазоні.

Метод `iloc` забезпечує доступ до рядків на основі **номерів індексів** починаючи з 0.

### Завдання 3. Завантаження даних та робота з пропущеними даними

Для роботи з пропущеними даними виконується завантаження датасету **Titanic** із вбудованих наборів даних бібліотеки Seaborn. Спочатку імпортується бібліотека Seaborn, після чого дані завантажуються у **DataFrame** і виводиться його структура за допомогою методу `.info()`:

```
import seaborn as sns

df = sns.load_dataset('titanic')
print(df.info())
```

Після завантаження даних можна перевірити наявність пропущених значень у кожному стовпці. Для цього використовується метод `.isna()` у поєднанні з `.sum()`:

```
print(df.isna().sum())
```

У випадку наявності пропущених значень у колонці `age`, вони замінюються медіанним значенням колонки. Спочатку обчислюється медіана для стовпця `age` методом `.median()`, після чого відбувається заповнення пропусків за допомогою `.fillna()`:

```
median_age = df['age'].median()
df['age'].fillna(median_age, inplace=True)
```

Це дозволяє замінити всі пропущені значення у стовпці `age` на медіану.

Для обробки даних, завантажених з **Excel-файлу**, використовується метод `pd.read_excel()`. Після завантаження даних пропущені значення у таблиці заповнюються нулями:

```
df_excel = pd.read_excel('data.xlsx')
df_excel.fillna(0, inplace=True)
```

Для запису даних з DataFrame у файл застосовують відповідні методи `df.to_excel()` та `df.to_csv()`.

Таким чином, виконуються базові операції з обробки даних, такі як додавання стовпців, фільтрація, робота з конкретними рядками, перевірка та заповнення пропущених значень. Ці операції є основою для аналізу даних за допомогою бібліотеки **Pandas**.

**Завдання 4. Створення підмножин та зрізів**

Для створення підмножин даних у Pandas використовуються різні техніки, такі як умовна фільтрація, індексація за допомогою методів `loc` та `iloc`, а також булева індексація.

**Створення підмножин за умовами** виконується за допомогою методу `loc`, який дозволяє фільтрувати дані на основі логічних умов. У цьому прикладі вибираються всі пасажири, які старші за 18 років і подорожували другим класом. Для об'єднання кількох умов використовується оператор `&` (логічне "і"):

```
subset = df.loc[(df['age'] > 18) & (df['pclass'] == 2)]
print(subset)
```

Цей код перевіряє, чи значення у стовпці `age` більше 18 та чи значення у стовпці `pclass` дорівнює 2. Метод `loc` повертає всі рядки, що відповідають обом умовам одночасно.

**Вибір даних за індексами** виконується за допомогою методу `iloc`, який дозволяє доступ до рядків та стовпців за числовими індексами. У цьому прикладі вибираються перші п'ять рядків та перші три стовпці з таблиці:

```
iloc_subset = df.iloc[0:5, 0:3]
print(iloc_subset)
```

`iloc[0:5, 0:3]` вказує діапазон індексів: перші п'ять рядків (індекси від 0 до 4) та перші три стовпці (індекси від 0 до 2).

Результатом є підмножина таблиці з відповідними рядками та стовпцями.

**Булеві індекси** застосовуються для вибору рядків, які відповідають певній умові. У цьому прикладі вибираються пасажири, які заплатили за квиток більше ніж 50 одиниць. Булева умова `df['fare'] > 50` повертає логічний масив, який використовується для фільтрації даних:

```
bool_subset = df[df['fare'] > 50]
print(bool_subset)
```

Таблиця `bool_subset` міститиме всі рядки, для яких значення у стовпці `fare` більше за 50.

Загалом ці методи дозволяють гнучко працювати з даними у Pandas: створювати підмножини на основі умов, здійснювати доступ до даних за числовими індексами та фільтрувати дані за булевими умовами.

**Завдання 5. Математичні розрахунки та створення нових стовпців**

Для виконання основних операцій аналізу даних у Pandas розглянемо кілька ключових завдань, пов'язаних з обчисленням статистичних показників, створенням нових стовпців і групуванням даних.

**Розрахунок середнього значення віку пасажирів** здійснюється за допомогою методу `.mean()` для стовпця `age`. Метод обчислює середнє арифметичне значення всіх числових елементів у стовпці:

```
mean_age = df['age'].mean()
print(mean_age)
```

Змінна `mean_age` містить середнє значення віку пасажирів у таблиці.

**Розрахунок кількості чоловіків та жінок** виконується за допомогою методу `.value_counts()`. Цей метод підраховує кількість унікальних значень у стовпці `sex`:

```
gender_counts = df['sex'].value_counts()
print(gender_counts)
```

Результатом є таблиця, де відображено кількість чоловіків та жінок у відповідних категоріях.

**Створення нового стовпця, що позначає дитину або дорослого** реалізується за допомогою методу `.apply()`. До значень стовпця `age` застосовується лямбда-функція, яка перевіряє умову: якщо вік менше 18, то значенням буде `'Дитина'`, інакше `'Дорослий'`:

```
df['age_category'] = df['age'].apply(lambda x: 'Дитина'
if x < 18 else 'Дорослий')
```

Цей код додає новий стовпець `age_category` до таблиці, де кожному пасажиру присвоєно категорію "Дитина" або "Дорослий" залежно від його віку.

**Розрахунок середнього віку чоловіків та жінок окремо** здійснюється шляхом фільтрації даних за значеннями у стовпці `sex`. Для кожної групи (чоловіків та жінок) обчислюється середнє значення віку:

```
male_mean_age = df[df['sex'] == 'male']['age'].mean()
female_mean_age = df[df['sex'] ==
'female']['age'].mean()
print(male_mean_age, female_mean_age)
```

Змінні `male_mean_age` та `female_mean_age` містять середній вік чоловіків і жінок відповідно.

**Розрахунок кількості пасажирів для кожного класу** виконується за допомогою методу `.value_counts()`. Цей метод підраховує кількість унікальних значень у стовпці `pclass`, що відповідає класам обслуговування пасажирів:

```
pclass_counts = df['pclass'].value_counts()
print(pclass_counts)
```

Результатом є таблиця з кількістю пасажирів для кожного класу (`1`, `2`, `3`).

Таким чином, виконуються основні операції для аналізу даних: обчислення середніх значень, підрахунок кількостей, створення нових категорій та групування даних. Ці методи дозволяють ефективно аналізувати та структурувати дані у Pandas для подальшої обробки та візуалізації.

**Питання для самоаналізу**

1. Як створювати DataFrame та Series у Pandas?
2. Як виконувати базові операції з масивами даних у Pandas?
3. Як обробляти пропущені дані у Pandas?

4. Як використовувати методи `loc` та `iloc` для роботи з DataFrame?
5. Як створювати нові стовпці на основі існуючих даних?

## Практична робота №5. Розширені функції Pandas для роботи зі структурою масивів та групування даних

**Мета -** навчитися працювати з функціями Pandas під час первинної обробки, групування даних та трансформації структури масивів.

### Термін виконання практичної роботи

Практична робота виконується протягом трьох занять.

**Практичне заняття 1** — виконання завдань 1–3.

**Практичне заняття 2** — виконання завдань 4–5, завантаження звіту та захист роботи.

### Завдання 1. Основи очищення даних у Pandas

Очищення даних є важливим етапом роботи з даними. Виконайте завдання з очищення та попередньої обробки даних.

**Зміст завдання.**

Для виконання завдань з очищення та підготовки даних використовується датасет **Titanic** із бібліотеки **Seaborn**. Спочатку потрібно завантажити датасет, а потім виконати основні кроки обробки даних: видалення або заповнення пропущених значень, обробку дублікатів та зміну типів даних.

**Імпорт датасету Titanic** здійснюється за допомогою методу `load_dataset()` з бібліотеки Seaborn. Спочатку виконується імпорт необхідних бібліотек:

```
import pandas as pd
import seaborn as sns

# Завантаження датасету Titanic
df = sns.load_dataset('titanic')
print(df.head())
```

Цей код завантажує датасет Titanic у **DataFrame** `df` та виводить перші 5 рядків таблиці для перевірки структури даних.

**Видалення пропущених значень** починається з перевірки наявності таких значень у стовпцях таблиці. Метод `.isna().sum()` підраховує кількість пропущених значень для кожного стовпця, а метод `.dropna()` видаляє рядки, де значення у певному стовпці є `NaN`:

```
print(df.isna().sum())
df.dropna(subset=['age'], inplace=True)
```

Цей код виводить кількість пропущених значень та видаляє всі рядки, у яких у стовпці `age` є пропуски.

**Заповнення пропущених значень** у стовпці `age` виконується за допомогою медіанного значення. Метод `.fillna()` заповнює пропуски розрахованим значенням:

```
df['age'].fillna(df['age'].median(), inplace=True)
```

Цей рядок обчислює медіану для стовпця `age` та замінює всі пропущені значення на це значення.

**Перевірка наявності дублікатів** здійснюється методом `.duplicated()`. Для підрахунку дублікатів використовується `.sum()`, а метод `.drop_duplicates()` видаляє дублікати з таблиці:

```
duplicates = df.duplicated().sum()
print("Дублікати:", duplicates)
df.drop_duplicates(inplace=True)
```

Цей код виводить кількість рядків-дублікатів у таблиці та видаляє їх.

**Зміна типів даних** використовується для приведення значень у стовпці `fare` до типу `float` та округлення чисел до цілих значень. Метод `.astype()` перетворює тип, а `.round()` округлює значення:

```
df['fare'] = df['fare'].astype(float)
df['fare'] = df['fare'].round(0)
```

Цей код забезпечує перетворення значень у стовпці `fare` у числовий формат із округленням до найближчого цілого числа.

**В результаті виконання коду з очищення даних**:

- Відображаються перші 5 рядків датасету Titanic.
- Виводиться кількість пропущених значень у стовпцях.
- Видаляються або заповнюються пропущені значення у стовпці `age`.
- Відбувається перевірка на дублікати та їх видалення.

- Стовпець `fare` приводиться до числового типу та округлюється до цілих значень.
- Виводиться оновлена структура таблиці за допомогою `info()`.

Цей код забезпечує повний цикл очищення та підготовки даних у Pandas для подальшого аналізу.

**Завдання 2. Базові операції з групування та фільтрації даних**

Навчіться виконувати групування даних, підрахунок статистичних показників та фільтрацію даних за умовами, наприклад для аналізу економічних показників різних регіонів чи соціальних груп.

**Зміст завдання.**

**Групування даних за однією змінною** дозволяє обчислити статистичні показники для груп, сформованих на основі значень одного стовпця. У цьому випадку обчислюється середній вік пасажирів для кожного класу квитків. Метод `groupby('pclass')` групує дані за значеннями стовпця `pclass`, а метод `.mean()` обчислює середнє значення для кожної групи у стовпці `age`:

```
grouped = df.groupby('pclass')['age'].mean()
print(grouped)
```

**Групування за кількома змінними** дозволяє розділити дані на групи, враховуючи значення кількох стовпців одночасно. У цьому прикладі дані групуються за класом квитків (`pclass`) та статтю пасажирів (`sex`). Для кожної групи обчислюється медіанне значення віку та тарифу за допомогою методу `.median()`:

```
grouped = df.groupby(['pclass', 'sex'])[['age',
'fare']].median()
print(grouped)
```

**Агрегування даних** виконується за допомогою методу `.agg()`, який дозволяє одночасно застосовувати різні функції до кількох стовпців. У цьому прикладі для стовпця `age` обчислюється середнє значення, а для стовпця `fare` обчислюється максимальне значення. Це дозволяє отримати кілька статистичних показників для кожної групи, сформованої за класом квитків:

```
aggregated = df.groupby('pclass').agg({'age': 'mean',
'fare': 'max'})
print(aggregated)
```

**Результати виконання завдань:**

**Середній вік пасажирів для кожного класу квитків** показує, наскільки різниться вік пасажирів у залежності від класу (`pclass`).

**Медіанне значення віку та тарифу для кожної комбінації класу квитка та статі** дозволяє проаналізувати відмінності між групами чоловіків і жінок у кожному класі.

**Агрегація даних** надає об'єднані статистичні показники для кожного класу, зокрема середній вік і максимальні витрати на перевезення.

Ці методи надають гнучкий інструментарій для групування та аналізу даних у різних категоріях і є основою для глибокого дослідження даних у Pandas.

### Завдання 3. Обробка та злиття масивів даних з Pandas

Використовуйте функції Pandas для обробки та злиття масивів даних. Наприклад, злиття інформації про пасажирів і фінансові показники.

**Зміст завдання.**

**Використання функцій Pandas для обробки та злиття масивів даних** дозволяє виконувати ефективний аналіз, очищення та комбінування інформації з різних джерел. У цьому завданні продемонстровано завантаження датасету **Titanic**, обробку пропущених значень, об'єднання масивів даних із використанням методів **`merge`** та **`join`**, а також збереження результатів у файл Excel.

**Об'єднання масивів даних** виконається за допомогою методів `merge` та `join`. Об'єднання виконується на основі спільної змінної (`id` для обєднання).

**Використання `merge`**: Два DataFrame `df1` і `df2`, які містять стовпець `id`, об'єднуються за допомогою методу `merge`. Тип об'єднання задається як "inner", тобто зберігаються лише спільні рядки:

```
df1 = pd.read_csv('data1.csv')
df2 = pd.read_csv('data2.csv')

combined = pd.merge(df1, df2, on='id', how='inner')
print(combined)
```

**Використання `join`**: Метод `join` використовується для об'єднання DataFrame за індексами або за стовпцями, якщо вказати параметр `on`. Для цього необхідно, щоб у DataFrame один зі стовпців був індексом, або вручну задати спільний ключ:

```
df1.set_index('id', inplace=True)
```

```
df2.set_index('id', inplace=True)

combined_join = df1.join(df2, how='inner')
print(combined_join)
```

У цьому прикладі спільним ключем є стовпець `id`, який попередньо встановлюється як індекс для обох DataFrame. Метод `join` об'єднує дані на основі індексів.

**Збереження даних у файл Excel** виконується за допомогою методу `to_excel()`. Оновлений DataFrame зберігається у файл `data.xlsx` без індексів рядків:

```
combined.to_excel('data.xlsx', index=False)
```

Це дозволяє експортувати результат обробки та об'єднання даних у формат Excel для подальшого аналізу.

**Підсумок виконання завдання:**

- Завантажено датасет Titanic, і виконано базове ознайомлення з даними.
- Пропущені значення у стовпці `age` заповнено медіаною для забезпечення повноти даних.
- Виконано об'єднання двох масивів даних за допомогою методів `merge` та `join`. Метод `merge` об'єднує дані за спільною змінною `id`, а метод `join` об'єднує на основі індексів.
- Результат обробки збережено у файл Excel для подальшого аналізу.

Ці методи є ефективними для роботи з великими наборами даних, об'єднання інформації з різних джерел і підготовки даних до аналітичної обробки.

**Завдання 4. Проведення розрахунків з даними, що представлені часовими рядами**

Аналізуйте часові ряди, використовуючи методи групування та ресемплінгу, наприклад для аналізу змін економічних показників за місяцями або кварталами.

**Зміст завдання.**

**Аналіз часових рядів** у Pandas дозволяє виконувати групування, ресемплінг, обчислення зміщень та приростів для дослідження змін показників у часі. Такі методи корисні для аналізу економічних тенденцій за місяцями, кварталами або іншими часовими інтервалами. Для прикладу ми завантажимо дані акцій **Apple (AAPL)** та **Tesla (TSLA)** за допомогою бібліотеки `yfinance`.

**Завантаження даних** виконується через бібліотеку `yfinance`, яка дозволяє отримати історичні дані фондового ринку. У прикладі завантажуються дані для акцій AAPL і TSLA:

```
import pandas as pd
import yfinance as yf

# Завантаження даних акцій AAPL та TSLA
aapl = yf.download('AAPL', start='2023-01-01',
end='2024-01-01')
tsla = yf.download('TSLA', start='2023-01-01',
end='2024-01-01')

# Додавання префіксів для стовпців обох таблиць
aapl = aapl.add_prefix('AAPL_')
tsla = tsla.add_prefix('TSLA_')

df = aapl.join(tsla, how='inner')

df.dropna(inplace=True)

print(df.head())
```

Цей код завантажує ціни закриття акцій **Apple** та **Tesla** за вказаний період та об'єднує їх у спільний DataFrame.

**Групування за місяцями** виконується за допомогою методу `.groupby()` у поєднанні з `pd.Grouper()`. Стовпець `Date` використовується як ключ для групування. Параметр `freq='M'` вказує на групування за місяцями, а `.mean()` обчислює середні значення для кожного місяця:

```
monthly = df.groupby(pd.Grouper(key='Date',
freq='M')).mean()
print("Середні значення за місяць:\n", monthly)
```

Цей код обчислює середні значення цін закриття **Apple** та **Tesla** для кожного місяця.

**Ресемплінг даних** дозволяє змінювати частоту часових даних. Для обчислення середніх значень за квартал використовується метод `.resample()` із параметром `freq='Q'`:

```
resampled = df.resample('Q', on='Date').mean()
print("Середні значення за квартал:\n", resampled)
```

Цей код обчислює середні значення цін закриття для кожного кварталу.

**Розрахунок зміщення** дозволяє оцінити затримку впливу змінних у часових рядах. Метод `.shift()` зміщує значення на вказану кількість періодів назад. У прикладі створюються нові стовпці з даними, зміщеними на один та два періоди назад:

```
df['AAPL_shifted_1'] = df['AAPL_Close'].shift()
df['TSLA_shifted_2'] = df['TSLA_Close'].shift(2)
print("Зміщені значення:\n", df[['Date', 'AAPL_Close',
'AAPL_shifted_1', 'TSLA_shifted_2']].head())
```

Стовпець `AAPL_shifted_1` містить значення `AAPL_Close`, зміщені на один період назад, а `TSLA_shifted_2` — значення `TSLA_Close`, зміщені на два періоди назад.

**Розрахунок приросту** дозволяє обчислити відсоткову зміну значень для аналізу темпів зростання. Метод `.pct_change()` автоматично обчислює приріст:

```
df['AAPL_growth'] = df['AAPL_Close'].pct_change()
df['TSLA_growth'] = df['TSLA_Close'].pct_change()
print("Темпи зростання цін акцій:\n", df[['Date',
'AAPL_growth', 'TSLA_growth']].head())
```

Стовпці `AAPL_growth` та `TSLA_growth` містять відсоткову зміну значень для акцій **Apple** та **Tesla** відповідно.

**Підсумок виконання завдання:**

- Завантажено дані фондового ринку для акцій AAPL і TSLA за допомогою бібліотеки `yfinance`.
- Виконано групування за місяцями для обчислення середніх значень цін закриття.
- Застосовано ресемплінг для обчислення середніх значень за квартал.
- Додано зміщення значень у часових рядах для оцінки затримки впливу.
- Обчислено приріст значень у часових рядах для аналізу темпів зростання.

Ці методи забезпечують детальний аналіз змін економічних показників за часовими інтервалами та дозволяють глибоко досліджувати тенденції у часових рядах.

**Завдання 5. Огляд функцій Pandas**

Ознайомтесь із розширеними можливостями Pandas для аналізу даних, наприклад з описовою статистикою для фінансових показників чи соціальних змінних.

**Зміст завдання.**

На сайті [pandas.pydata.org](pandas.pydata.org) можна знайти детальну документацію, навчальні ресурси та приклади використання бібліотеки Pandas. Цей сайт є офіційним джерелом інформації для вивчення можливостей **Pandas**, які включають обробку, аналіз та маніпуляції з даними.

Основні розділи, доступні на сайті:

**Документація (Documentation)**. Докладний опис функцій, методів та можливостей Pandas. Тут наводяться приклади коду, опис параметрів та способів застосування основних інструментів бібліотеки. Документація охоплює роботу з:

**DataFrame** та **Series**

Групуванням та агрегацією даних

Обробкою пропущених значень

Ресемплінгом часових рядів

Об'єднанням даних за допомогою `merge` і `join`

Роботою з різними форматами файлів (CSV, Excel, SQL, JSON тощо).

**User Guide (Керівництво користувача)**. Покрокові інструкції та пояснення для новачків та досвідчених користувачів. Тут розглядаються конкретні сценарії використання Pandas із прикладами, зокрема:

Читання та запис даних.

Обчислення статистичних показників.

Візуалізація результатів у поєднанні з іншими бібліотеками (Matplotlib, Seaborn).

**Tutorials (Підручники)**. Навчальні матеріали для освоєння основних можливостей Pandas. Підручники містять практичні завдання та приклади для виконання:

Початок роботи з Pandas.

Основи маніпуляції даними.

Аналітичний аналіз та візуалізація даних.

Аналіз часових рядів і фінансових показників.

**API Reference (Посилання на API)**. Повний список методів та функцій, доступних у бібліотеці Pandas. Тут можна знайти детальну інформацію про функції з поясненням усіх можливих параметрів та повернутих значень.

**Community (Спільнота)**. Ресурси для взаємодії з Pandas-спільнотою:

Обговорення на форумах та в соціальних мережах.

Підтримка користувачів через Stack Overflow.

Можливість долучитися до розробки Pandas.

**Release Notes (Оновлення)**. Опис нових функцій, змін та виправлень у кожній версії Pandas.

**Підтримка суміжних інструментів**. Інтеграція Pandas з іншими бібліотеками та інструментами для обробки та аналізу даних, такими як NumPy, Matplotlib, SQLAlchemy та Scikit-Learn.

На сайті також можна знайти **приклади коду** для розв'язання типових завдань у роботі з даними, що допомагає користувачам швидко освоїти функціонал бібліотеки Pandas та застосовувати його у власних проектах.

### Питання для самоаналізу

1. Як видаляти та обробляти пропущені дані у Pandas?
2. Як здійснювати групування даних у Pandas?
3. Як працювати з часовими рядами у Pandas?
4. Які основні методи об’єднання даних у Pandas?
5. Як застосовувати аналіз показників описової статистики з Pandas?

## Практична робота №6.
## Аналітична візуалізація даних з Python

**Мета -** навчитися працювати з масивами даних за допомогою стандартних функцій Python та інструментів візуалізації даних, таких як Matplotlib.

### Терімни виконання практичної роботи

Практична робота виконується протягом трьох занять:

**Практичне заняття 1** — виконання завдань 1–3.

**Практичне заняття 2** — виконання завдань 4–5, завантаження звіту та захист роботи.

### Завдання 1. Вивчення прикладів із лекції

Перегляньте та виконайте всі приклади, що були розглянуті на лекції. Зверніть увагу на роботу з бібліотеками Pandas, Matplotlib, а також на побудову основних типів графіків.

**Зміст завдання.**

**Вивчення прикладів із лекції** передбачає практичну роботу з прикладами, що демонструють використання **Pandas** для обробки

даних та **Matplotlib** для їх візуалізації. Основний акцент робиться на побудові графіків різного типу, роботі зі стилями графіків, підписами та додатковими налаштуваннями.

**Імпорт необхідних бібліотек** є першим кроком. Для побудови графіків і обробки даних потрібно імпортувати бібліотеки **Pandas** для роботи з табличними даними, **Matplotlib** для візуалізації даних та **Seaborn** для покращеного стилю графіків. Крім цього, використовується `%matplotlib inline` для відображення графіків безпосередньо у середовищі розробки:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

**Лінійний графік** демонструє зміну значень у часі або на основі певної послідовності даних. Для побудови графіка використовуються функція `plt.plot()` та параметри для підписів осей і заголовка. Приклад показує створення лінійного графіка на основі списку значень:

```
data = [1, 3, 5, 7, 9]
plt.plot(data)
plt.title("Лінійний графік")
plt.xlabel("Ось X")
plt.ylabel("Ось Y")
plt.show()
```

**Кругова діаграма** використовується для відображення пропорцій елементів у складі цілого. Значення відображаються у вигляді секторів кола, а відсотки кожного сектору додаються за допомогою параметра `autopct`:

```
values = [20, 30, 50]
labels = ['A', 'B', 'C']
plt.pie(values, labels=labels, autopct='%1.1f%%')
plt.title("Кругова діаграма")
plt.show()
```

**Гістограма** дозволяє проаналізувати розподіл даних у вибірці. Графік будується за допомогою функції `plt.hist()`, де параметр `bins` визначає кількість інтервалів для групування даних:

```
data = [1, 2, 2, 3, 3, 3, 4, 4, 4, 4]
plt.hist(data, bins=5)
```

```
plt.title("Гістограма розподілу")
plt.xlabel("Значення")
plt.ylabel("Частота")
plt.show()
```

**Boxplot (ящик з вусами)** використовується для візуалізації розподілу даних, включаючи медіану, квартилі та викиди. Функція `plt.boxplot()` будує графік на основі переданого набору даних:

```
data = [7, 8, 5, 6, 7, 9, 10, 5, 6, 8]
plt.boxplot(data)
plt.title("Boxplot")
plt.show()
```

**Scatter plot (діаграма розсіювання або залежності)** дозволяє показати залежність між двома змінними. Графік будується за допомогою функції `plt.scatter()`, де параметри `x` та `y` відповідають значенням на осях:

```
x = [1, 2, 3, 4, 5]
y = [5, 7, 6, 8, 7]
plt.scatter(x, y, color='red')
plt.title("Scatter plot")
plt.xlabel("X")
plt.ylabel("Y")
plt.show()
```

**Стовпчаста діаграма** використовується для порівняння категорій за певними значеннями. Функція `plt.bar()` будує стовпці на основі переданих категорій та їх значень:

```
labels = ['A', 'B', 'C', 'D']
values = [10, 15, 20, 25]
plt.bar(labels, values, color='blue')
plt.title("Стовпчаста діаграма")
plt.xlabel("Категорії")
plt.ylabel("Значення")
plt.show()
```

**Налаштування стилів та форматування графіків** включає додавання підписів, легенди та параметрів для кастомізації зовнішнього вигляду. Використання методу `plt.legend()` дозволяє додати пояснення для кожної лінії графіка:

```
plt.plot([1, 2, 3], [4, 5, 6], label="Лінія 1")
plt.plot([1, 2, 3], [6, 5, 4], label="Лінія 2")
plt.title("Графік з легендою")
plt.xlabel("Ось X")
plt.ylabel("Ось Y")
plt.legend()
plt.show()
```

**Створення декількох підграфіків** дозволяє відобразити кілька графіків на одній ділянці. Для цього використовується функція `plt.subplots()` із визначенням кількості рядків та стовпців:

```
fig, axs = plt.subplots(1, 2, figsize=(10, 5))
axs[0].plot([1, 2, 3], [4, 5, 6])
axs[1].bar(['A', 'B', 'C'], [7, 8, 9])
plt.show()
```

**Налаштування кольорів, маркерів та стилів ліній** дозволяє контролювати зовнішній вигляд графіків. У прикладі використовується параметри `color`, `linestyle` та `marker`:

```
plt.plot([1, 2, 3], [4, 5, 6], color='green',
linestyle='--', marker='o')
plt.title("Налаштований графік")
plt.show()
```

### Завдання 2. Побудова лінійчастих та стовпчикових діаграм

**Побудова лінійчастих та стовпчикових діаграм** передбачає завантаження фінансових даних за допомогою бібліотеки **yfinance**, побудову лінійчастих графіків для цін закриття акцій **Apple** та **Bitcoin**, а також створення стовпчикових діаграм для середньомісячних цін. Завдання охоплює роботу з часовими рядами, групуванням даних та їхньою візуалізацією.

**Завантаження масивів даних** виконується за допомогою бібліотеки **yfinance**, що є зручним інструментом для отримання фінансових даних із Yahoo Finance. Спочатку завантажуються дані про ціни акцій компанії **Apple (AAPL)** та ціни **Bitcoin** у доларах США. Функція `download()` використовується для отримання даних за вказаний проміжок часу:

```
import yfinance as yf

# Завантаження даних Apple
```

```
apple = yf.download('AAPL', start='2022-01-01',
end='2023-01-01')

# Завантаження даних Bitcoin
btc = yf.download('BTC-USD', start='2022-01-01',
end='2023-01-01')
```

Дані завантажуються у форматі **DataFrame**, де кожен рядок відповідає одній даті, а стовпці містять фінансові показники: відкриття, закриття, мінімальні та максимальні значення, а також обсяги торгів.

**Побудова лінійчастих графіків** виконується за допомогою бібліотеки **Matplotlib**, що дозволяє візуалізувати зміни цін закриття у часі. Графік будується для обох активів (Apple і Bitcoin), з використанням різних кольорів для їх відображення:

```
import matplotlib.pyplot as plt

plt.plot(apple['Close'], label='Apple', color='blue')
plt.plot(btc['Close'], label='Bitcoin', color='orange')
plt.title('Ціни акцій Apple та Bitcoin')
plt.xlabel('Дата')
plt.ylabel('Ціна')
plt.legend()
plt.show()
```

Функція `plt.plot()` будує лінійні графіки на основі значень у стовпці **`Close`** для обох наборів даних. Параметр **`label`** додає підпис для кожної лінії, а **`legend()`** відображає відповідну легенду на графіку. Кольори `blue` і `orange` використовуються для відрізнення даних про Apple та Bitcoin.

**Побудова стовпчикових діаграм** виконується на основі середньомісячних значень цін закриття **Bitcoin**. Спочатку дані групуються за місяцями за допомогою методу **`resample()`** з параметром `'M'`, що відповідає щомісячній частоті. Для кожної групи обчислюється середнє значення:

```
monthly_mean_btc = btc['Close'].resample('M').mean()
```

Після обчислення середніх значень використовується метод **`.plot()`** з параметром `kind='bar'` для побудови вертикальної стовпчикової діаграми:

```
monthly_mean_btc.plot(kind='bar', color='green')
plt.title('Середні місячні ціни на Bitcoin')
```

```
plt.xlabel('Місяць')
plt.ylabel('Середня ціна')
plt.show()
```

Стовпчикова діаграма відображає середні ціни **Bitcoin** для кожного місяця у вигляді вертикальних стовпців. Параметр **`color='green'`** задає колір стовпців. Титул та підписи осей надають додаткову інформацію для графіка.

**Побудова горизонтальних стовпчикових діаграм** може бути реалізована з використанням параметра `kind='barh'`, що будує горизонтальну діаграму на основі тих самих середньомісячних даних:

```
monthly_mean_btc.plot(kind='barh', color='purple')
plt.title('Середні місячні ціни на Bitcoin
(горизонтальна діаграма)')
plt.xlabel('Середня ціна')
plt.ylabel('Місяць')
plt.show()
```

Ця діаграма є альтернативою вертикальній стовпчиковій діаграмі та дозволяє краще візуалізувати дані у випадку великої кількості категорій або коли підписи мають великий розмір.

Результатом виконання завдання є графіки, що наочно показують динаміку змін цін та їх середні значення у часових рядах.

### Завдання 3. Групування даних та побудова кругової діаграми

**Групування покемонів за типом** виконується за допомогою функції `groupby()` бібліотеки Pandas.

База даних **Pokemon** є частиною бібліотеки Seaborn і містить інформацію про характеристики різних покемонів. Цей датасет корисний для аналізу розподілу характеристик покемонів, їхніх типів, потужності та інших показників, що дозволяють проводити глибокий статистичний аналіз.

Спочатку завантажується датасет покемонів з бібліотеки **Seaborn** через функцію `load_dataset()`. Далі за допомогою методу `.groupby()` дані групуються за стовпцем **`Type 1`**, а функція `.size()` підраховує кількість покемонів у кожній групі:

```
import seaborn as sns
import matplotlib.pyplot as plt

# Завантаження датасету покемонів
pokemon = sns.load_dataset('pokemon')
```

```
# Групування покемонів за типом (Type 1)
grouped = pokemon.groupby('Type 1').size()
```

**Побудова кругової діаграми** здійснюється методом `.plot()` із параметром **`kind='pie'`**, який будує кругову діаграму на основі групованих даних. Параметр **`autopct='%1.1f%%'`** дозволяє відображати відсоткові значення кожного сектору:

```
grouped.plot(kind='pie', autopct='%1.1f%%')
plt.title('Розподіл покемонів за типом')
plt.ylabel('')  # Видалення мітки для осі Y
plt.show()
```

Функція `plot()` будує діаграму, де кожен сектор відповідає типу покемонів, а відсоткова частка обчислюється автоматично.

### Задання 4. Побудова BoxPlot та гістограми розподілу

**BoxPlot для статистик покемонів** будується для аналізу розподілу значень рівня **атаки** покемонів.

Метод **`boxplot()`** у Pandas дозволяє створити діаграму, що відображає медіану, квартилі та викиди. Для групування даних за типом покемонів використовується параметр **`by`**, який групує значення на основі стовпця **`Type 1`**. Розмір графіка задається параметром `figsize`:

```
pokemon.boxplot(column='Attack', by='Type 1',
figsize=(10, 6))
plt.title('BoxPlot для атаки покемонів')
plt.suptitle('')  # Видалення автоматичного заголовка
plt.xlabel('Тип покемонів (Type 1)')
plt.ylabel('Значення атаки')
plt.show()
```

**BoxPlot** дозволяє оцінити розподіл значень атаки для кожного типу покемонів (`Type 1`), включаючи:

**Медіану**, що показує центральне значення розподілу.

**Квартилі**, які поділяють дані на чотири рівні частини (рідмасивив даних).

**Викиди**, що відображають значення, які значно відрізняються від решти даних.

Для покращення вигляду графіка видаляється автоматичний заголовок, що створюється Pandas, за допомогою параметра `plt.suptitle('')`.

**Гістограми розподілу** створюються для трьох основних статистик покемонів, зокрема **Attack**, **Defense** та **Speed**. Метод **`hist()`** у Pandas дозволяє побудувати гістограми для всіх зазначених стовпців одночасно. Параметр `bins` визначає кількість інтервалів для розподілу значень, а `figsize` налаштовує розмір діаграми:

```
# Побудова гістограм для значень атаки, захисту та
швидкості покемонів
pokemon[['Attack', 'Defense', 'Speed']].hist(bins=20,
figsize=(12, 6))
plt.suptitle('Розподіл статистик покемонів')
plt.show()
```

Метод **`.hist()`** будує три окремі гістограми для стовпців **Attack**, **Defense** та **Speed**. Кожна гістограма показує частоту значень у вибірці, розподілених по інтервалах. Параметр `bins=20` визначає кількість інтервалів для групування даних, що дозволяє детальніше побачити структуру розподілу.

На графіку для кожної гістограми додається спільний заголовок за допомогою функції **`plt.suptitle()`**, щоб пояснити, які саме дані відображаються.

### Завдання 5. Графіки залежностей та кореляційна матриця

**Scatterplot для залежності цін** використовується для візуалізації зв'язку між цінами акцій **Apple** та **Bitcoin**. Спочатку дані об'єднуються за допомогою методу **`join`** на основі індексів, щоб створити загальний DataFrame, який містить ціни закриття для обох активів.

**Об'єднання даних** виконується на основі стовпців **`Close`** із двох DataFrame. Параметри **`lsuffix`** і **`rsuffix`** додають суфікси для уникнення конфліктів у назвах стовпців:

```
import pandas as pd
import yfinance as yf
import matplotlib.pyplot as plt

# Завантаження даних Apple та Bitcoin
apple = yf.download('AAPL', start='2022-01-01',
end='2023-01-01')
```

```
btc = yf.download('BTC-USD', start='2022-01-01',
end='2023-01-01')

# Об'єднання даних за цінами закриття
merged = apple[['Close']].join(btc[['Close']],
lsuffix='_apple', rsuffix='_btc')
```

**Побудова scatterplot** демонструє залежність між цінами акцій **Apple** та цінами **Bitcoin**. Для побудови використовується функція **`plt.scatter()`**, а параметр **`alpha=0.5`** додає прозорість точок для кращої видимості у разі їх накладання:

```
plt.scatter(merged['Close_apple'], merged['Close_btc'],
alpha=0.5)
plt.xlabel('Ціна акцій Apple')
plt.ylabel('Ціна Bitcoin')
plt.title('Залежність цін Apple та Bitcoin')
plt.show()
```

На графіку кожна точка відповідає парі значень цін закриття для **Apple** та **Bitcoin** у конкретний день. Вісь **X** показує ціни **Apple**, а вісь **Y** – ціни **Bitcoin**. Прозорість дозволяє легше побачити густі ділянки, де точки накопичуються.

**Кореляційна матриця** дозволяє оцінити статистичний зв'язок між змінними. Метод **`.corr()`** обчислює коефіцієнти кореляції для всіх числових стовпців DataFrame. Для наочної візуалізації матриці використовується функція **`heatmap()`** з бібліотеки **Seaborn**:

```
import seaborn as sns

# Обчислення кореляційної матриці
correlation_matrix = merged.corr()

# Візуалізація кореляційної матриці за допомогою
heatmap
sns.heatmap(correlation_matrix, annot=True,
cmap='coolwarm')
plt.title('Кореляційна матриця')
plt.show()
```

Функція **`heatmap()`** будує теплову карту, де кольори відображають значення коефіцієнтів кореляції. Параметр **`annot=True`** додає числові значення на діаграму, а **`cmap='coolwarm'`** визначає колірну гаму для візуалізації. Значення кореляції **близьке до 1** вказує

на сильний позитивний зв'язок між змінними, а **близьке до -1** – на сильний негативний зв'язок.

Результати виконання коду:

**Scatterplot** показує залежність між цінами акцій **Apple** та **Bitcoin** у вигляді точкової діаграми.

**Кореляційна матриця** наочно відображає ступінь лінійного зв'язку між змінними, допомагаючи визначити, чи існує сильна кореляція між ними.

**Питання для самоаналізу**

1. Як будувати базові типи графіків у Matplotlib?
2. Як виконувати групування даних за категоріями?
3. Як створювати кругові діаграми для відображення пропорцій?
4. Як будувати кореляційні матриці та що вони показують?

## Індивідуальна робота 1.
## Робота зі структурами даних Python: списки, словники та кортежі

**Мета -** навчитися створювати та обробляти дані за допомогою базових структур Python.

**Завдання**

**Завдання 1. Робота зі структурами даних Python**

Створити список зі значеннями, що відповідають обраним економічним або соціальним показникам (наприклад, ВВП, інфляція, рівень безробіття).

Створити словник, у якому ключами будуть категорії (наприклад, роки або регіони), а значеннями списки даних.

Створити кортеж, що містить агреговані показники (середнє, медіана, дисперсія) для кожного списку даних.

**Завдання 2. Використання функцій для обробки даних**

Написати функцію, яка приймає список або словник та повертає обчислені показники (середнє, медіана, мінімум, максимум).

Застосувати функцію до створених даних та вивести результати у форматі таблиці або списку.

Додати до функції можливість фільтрації даних (наприклад, значення вище або нижче середнього).

**Завдання 3. Порівняння та обробка кількох структур даних**

Створити два словники з економічними показниками для різних періодів або регіонів.

Виконати об'єднання даних зі словників у новий словник з використанням циклів.

Перевірити, чи є спільні ключі в словниках, та виділити їх для подальшого аналізу.

Виконати сортування списків даних у словнику за значеннями.

**Варіанти вхідних даних**

Варіант 1. Економічні показники

Генерація випадкових даних для ВВП, рівня інфляції, безробіття за допомогою бібліотеки `random` або `numpy`.

```
import random

gdp = [round(random.uniform(1000, 5000), 2) for _ in
range(10)]
inflation = [round(random.uniform(1.0, 5.0), 2) for _
in range(10)]
unemployment = [round(random.uniform(5.0, 15.0), 2) for
_ in range(10)]
print("GDP:", gdp)
print("Inflation:", inflation)
print("Unemployment:", unemployment)
```

Варіант 2. Фінансові показники

Генерація даних для вартості акцій або криптовалют з урахуванням сезонності.

```
stock_prices = [round(random.gauss(100, 15), 2) for _
in range(20)]
crypto_prices = [round(random.uniform(20000, 60000), 2)
for _ in range(20)]
print("Stock Prices:", stock_prices)
print("Crypto Prices:", crypto_prices)
```

Варіант 3. Соціальні показники

Генерація даних про кількість населення у регіонах або вікових категоріях.

```
population = [random.randint(100000, 1000000) for _ in
range(5)]
regions = ['Region A', 'Region B', 'Region C', 'Region
D', 'Region E']
data = dict(zip(regions, population))
print("Population by Region:", data)
```

Варіант 4. Спортивна статистика

Генерація даних про очки, набрані командами у турнірах.

```
teams = ['Team A', 'Team B', 'Team C', 'Team D']
scores = {team: [random.randint(50, 150) for _ in
range(5)] for team in teams}
print("Scores by Team:", scores)
```

## Індивідуальна робота 2.
## Аналіз та обробка даних з використанням бібліотек Pandas та Matplotlib.

**Мета** – навчитися виконувати обробку даних за допомогою Pandas, аналізувати їх структуру та представляти результати у вигляді таблиць і візуалізацій.

**Завдання**

**Завдання 1. Створення та попередня обробка даних**

Створіть датасет за варіантом завдання.

Видалити пропущені дані або заповнити їх середнім чи медіанним значенням для відповідних колонок.

Виконати фільтрацію даних за умовами, наприклад, залишити лише записи з продажами понад заданий поріг.

Відсортувати дані за обраною колонкою, наприклад, за датою або числовим показником.

Зберегти оброблений датасет у новий файл CSV.

**Завдання 2. Групування даних та обчислення показників**

Згрупувати дані за категоріями, визначеними варіантом завдання (наприклад, за регіонами, статтю, класами).

Обчислити середнє, медіану, мінімум, максимум для кожної групи.

Створити новий DataFrame, що містить результати групування.

Зберегти результати групування у форматі Excel.

**Завдання 3. Візуалізація даних**

Використати Pandas та Matplotlib для побудови графіків на основі обраного варіанту завдання.

Побудувати лінійчастий графік залежності двох числових змінних.

Побудувати гістограму для розподілу числових значень.

Побудувати кругову діаграму для категоріальних даних.

Зберегти графіки у форматі PNG із підписами осей та заголовками.

**Варіанти вхідних даних**

Варіант 1. Дані про продажі

```
import pandas as pd
import random

categories = ['Electronics', 'Clothing', 'Groceries']
data = {
    'Category': [random.choice(categories) for _ in
range(50)],
    'Sales': [round(random.uniform(10, 1000), 2) for _
in range(50)],
    'Date': pd.date_range(start='2022-01-01',
periods=50)
}
df = pd.DataFrame(data)
print(df.head())
```

Варіант 2. Дані про пасажирів

```
import pandas as pd
import random

sex = ['male', 'female']
classes = [1, 2, 3]
data = {
    'Sex': [random.choice(sex) for _ in range(50)],
    'Class': [random.choice(classes) for _ in
range(50)],
    'Age': [random.randint(1, 80) for _ in range(50)],
    'Fare': [round(random.uniform(10, 100), 2) for _ in
range(50)]
}
df = pd.DataFrame(data)
print(df.head())
```

Варіант 3. Дані про економічні показники

```
import pandas as pd
import random

regions = ['Region A', 'Region B', 'Region C']
data = {
    'Region': [random.choice(regions) for _ in
range(30)],
```

```
    'GDP': [round(random.uniform(500, 5000), 2) for _
in range(30)],
    'Inflation': [round(random.uniform(0.5, 10.0), 2)
for _ in range(30)],
    'Unemployment': [round(random.uniform(1.0, 15.0),
2) for _ in range(30)]
}
df = pd.DataFrame(data)
print(df.head())
```

Варіант 4. Дані про акції

```
import pandas as pd
import random

data = {
    'Date': pd.date_range(start='2023-01-01',
periods=30),
    'Stock Price': [round(random.uniform(100, 200), 2)
for _ in range(30)],
    'Volume': [random.randint(1000, 10000) for _ in
range(30)]
}
df = pd.DataFrame(data)
print(df.head())
```

# ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Python Software Foundation : Python Documentation. URL: https://docs.python.org/3/ (дата звернення: 17.12.2024).
2. Python Software Foundation : Python Тьюторіал. URL: https://docs.python.org/uk/3/tutorial/ (дата звернення: 17.12.2024).
3. Pandas Documentation : Pandas Official Website. URL: https://pandas.pydata.org/ (дата звернення: 17.12.2024).
4. Юрченко І. В., Сікора В. С. Програмування мовою Python. Чернівці : ЧНУ імені Ю. Федьковича, 2022. 104 с.
5. Zherlitsyn D. Python for Finance: Data analysis, financial modeling, and portfolio management (English Edition). 1st edition. BPB Publications, 2024. 681 p.

Навчально-методичне видання

Жерліцин Дмитро Михайлович

# ПРОГРАМУВАННЯ НА PYTHON

**методичні рекомендації**
**до виконання практичних та індивідуальних робіт**

Самостійне електронне мережеве видання

Публікується в авторській редакції