Міністерство освіти та науки України
Національний університет водного господарства та природокористування
Кафедра автоматизації, електротехнічних та комп'ютерно-інтегрованих технологій

04-03-358М

МЕТОДИЧНІ ВКАЗІВКИ

до виконання практичних робіт з навчальної дисципліни

«Штучний інтелект в робототехніці»

для здобувачів вищої освіти першого (бакалаврського) рівня за освітньо-професійною програмою «Робототехніка та штучний інтелект» спеціальності 151 «Автоматизація та комп'ютерно-інтегровані технології» денної та заочної форм навчання

Рекомендовано науково-методичною радою з якості ННІ АКОТ
Протокол № 10 від 20.09.2022 р.

Рівне – 2022

Методичні вказівки до виконання практичних робіт з навчальної дисципліни «Штучний інтелект в робототехніці» для здобувачів вищої освіти першого (бакалаврського) рівня за освітньо-професійною програмою «Робототехніка та штучний інтелект» спеціальності 151 «Автоматизація та комп'ютерно-інтегровані технології» денної та заочної форм навчання [Електронне видання] / Сафоник А. П. – Рівне : НУВГП, 2022. – 86 с.

Укладач: Сафоник А. П., д.т.н., професор, професор кафедри автоматизації, електротехнічних та комп'ютерно-інтегрованих технологій.

Відповідальний за випуск: Древецький В. В., д.т.н., професор, завідувач кафедри автоматизації, електротехнічних та комп'ютерно-інтегрованих технологій.

Керівник освітньої програми «Робототехніка та штучний інтелект»: Сафоник А. П., д.т.н., професор кафедри автоматизації, електротехнічних та комп'ютерно-інтегрованих технологій.

Зміст

Практична робота №1. Встановлення Python

1.1. Мета роботи

Навчитися встановлювати Python.

1.2. Теоретичні відомості

Python (найчастіше вживане прочитання — «Па́йтон», запозичено назву з британського шоу Монті Пайтон) — інтерпретована об'єктно-орієнтована мова програмування високого рівня зі строгою динамічною типізацією. Розроблена в 1990 році Гвідо ван Россумом. Структури даних високого рівня разом із динамічною семантикою та динамічним зв'язуванням роблять її привабливою для швидкої розробки програм, а також як засіб поєднування наявних компонентів. Python підтримує модулі та пакети модулів, що сприяє модульності та повторному використанню коду. Інтерпретатор Python та стандартні бібліотеки доступні як у скомпільованій, так і у вихідній формі на всіх основних платформах. В мові програмування Python підтримується кілька парадигм програмування, зокрема: об'єктно-орієнтована, процедурна, функціональна та аспектно-орієнтована.

1.3. Порядок виконання роботи

1. Версії python

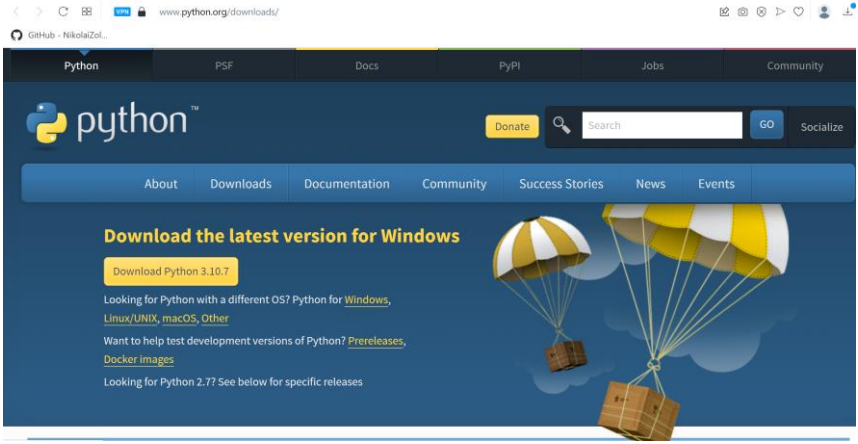
На сьогоднішній день існує дві версії Python - Python 2 і Python 3, вони не мають повної сумісності один з одним. На даний момент друга версія Python все ще широко використовується, але судячи зі змін, що відбуваються, з часом залишиться тільки запустити старий код. У своїй роботі ми будемо використовувати Python 3, і, в майбутньому, якщо десь знайдеться слово Python, то під ним слід розуміти Python 3. Випадки використання Python 2 будуть спеціально вказані.

2. Встановлення Python

Щоб встановити інтерпретатор Python на свій комп'ютер, перше, що потрібно зробити - це завантажити дистрибутив. Завантажити його можна з офіційного сайту, перейшовши за посиланням https://www.python.org/downloads/

2.1 Встановлення Python на Windows

Для операційної системи Windows дистрибутив поширюється або у вигляді виконуваного файлу (з розширенням exe), або у вигляді файлу архіву (з розширенням zip).

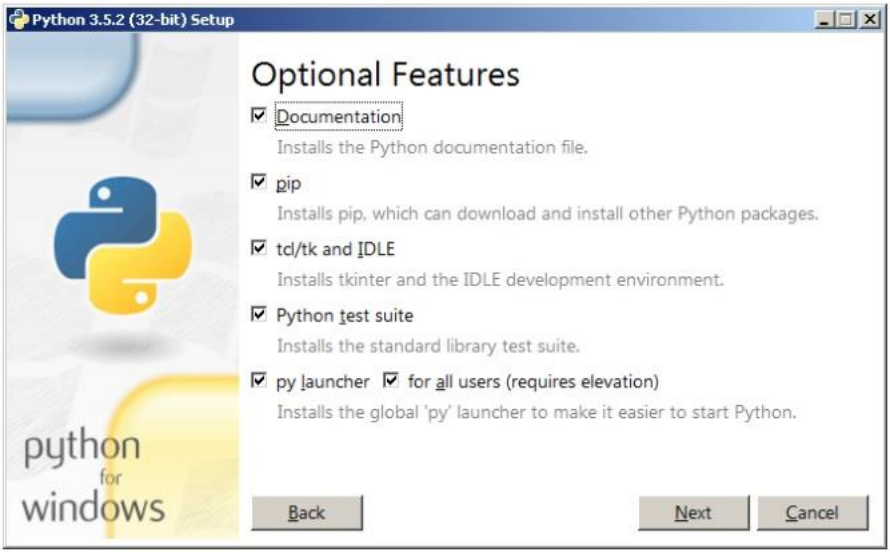


Порядок установки.

1. Запустіть завантажений інсталяційний файл.
2. Виберіть спосіб установки.

Це вікно пропонує два варіанти: «Встановити зараз» і «Налаштувати установку». Якщо ви виберете Install Now, Python буде встановлений в теці за вказаним шляхом. Крім самого інтерпретатора, буде встановлено IDLE (інтегроване середовище розробки), pip (менеджер пакетів) і документація, а також будуть створені відповідні ярлики і встановлені посилання на файли, які мають розширення .py з інтерпретатором Python. Налаштувати установку - це нестандартний варіант установки. Параметр Add python 3.5 to PATH необхідний для того, щоб дозволити інтерпретатору запускатися без вказівки повного шляху до виконуваного файлу при роботі в командному рядку.

3. Перевірте необхідні параметри встановлення (доступні, коли ви вибираєте Налаштувати встановлення)



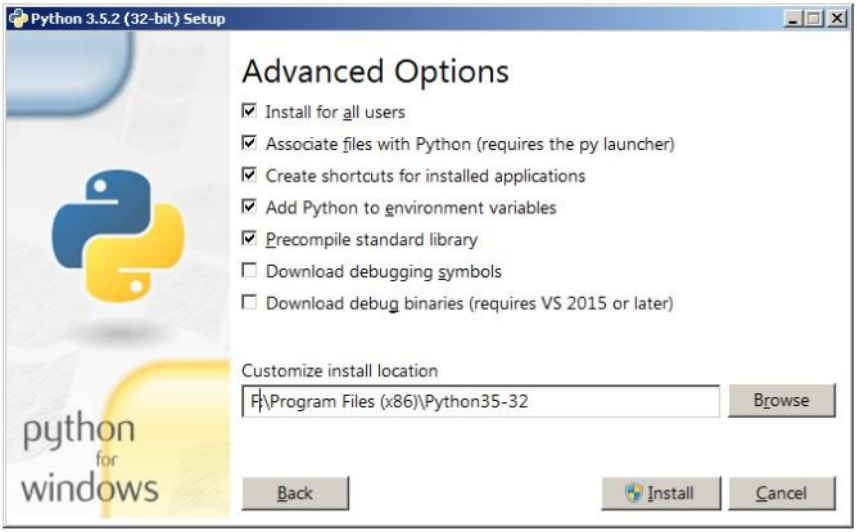
На цьому кроці нам пропонується позначити доповнення, які встановлені за допомогою інтерпретатора Python. Рекомендуємо вибрати всі варіанти.

Документація – установка документації.

pip – встановіть менеджер пакетів pip.

tcl/tk і IDLE – установка інтегрованого середовища розробки (IDLE) і бібліотеки для побудови графічного інтерфейсу (tkinter).

4. Виберіть місце встановлення (доступне, коли ви виберете Налаштувати встановлення)



Крім вказівки шляху, це вікно дозволяє внести додаткові зміни в процес установки за допомогою наступних опцій:

Встановити для всіх користувачів – Встановити для всіх користувачів. Якщо ви не виберете цей варіант, вам буде запропоновано встановити в папку користувача, що встановлює інтерпретатор.

Пов'язувати файли з Python – пов'язувати файли з розширенням .py з Python. Вибір цього параметра внесе зміни до Windows, які дозволять запускати сценарії Python подвійним клацанням миші.

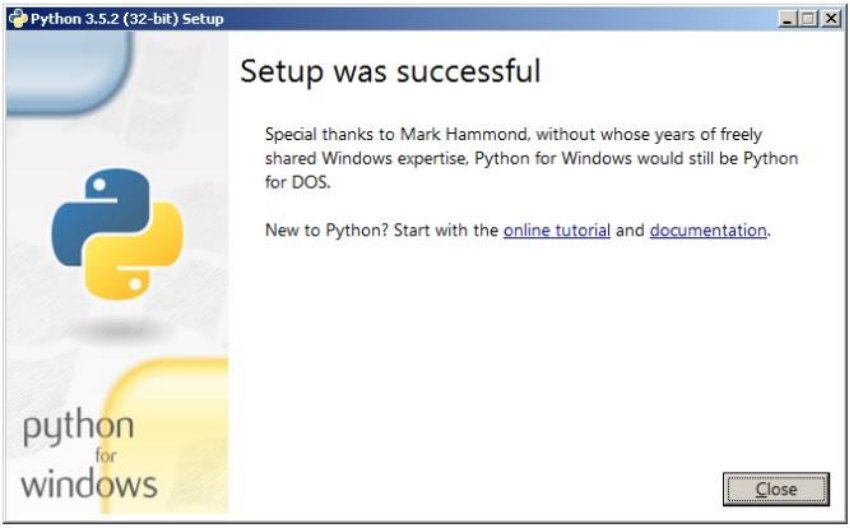
Створюйте ярлики для встановлених додатків .

Додати Python до змінних середовища — Додавання шляхів до інтерпретатора Python до змінної PATH.

Стандартна бібліотека попереднього складання – попередньо компіляція стандартної бібліотеки.

Останні два пункти пов'язані з завантаженням компонентів для налагодження, встановлювати їх ми не будемо.

5. Після успішної установки ви отримаєте наступне повідомлення.



2.2 Встановлення Python на Linux

Найчастіше інтерпретатор Python вже включений в дистрибутив. Це можна перевірити, набравши в терміналі

```
> python
```

або

```
> pytho3
```

У першому випадку ви запустите Python 2, у другому – Python 3. Надалі, швидше за все, всі дистрибутиви Linux, до складу яких входить Python, будуть включати тільки третю версію. Якщо при спробі запустити Python ви отримуєте повідомлення про те, що він не встановлений, або встановлений, але не те, що ви хочете, то у вас є два способи: а) побудувати Python з джерела; б) взяти зі сховища.

Для установки зі сховища в Ubuntu використовуйте команду

```
> sudo apt-get install python3
```

Збірку з джерел в цій статті розглядати не будемо.

3. Встановіть анаконду

Для зручності запуску зразків і вивчення мови Python рекомендуємо встановити пакет Anaconda на свій ПК. Цей пакет включає в себе інтерпретатор Python (є версії 2 і 3), набір найбільш часто використовуваних бібліотек, а також зручне середовище розробки і виконання, яка запускається в браузері.

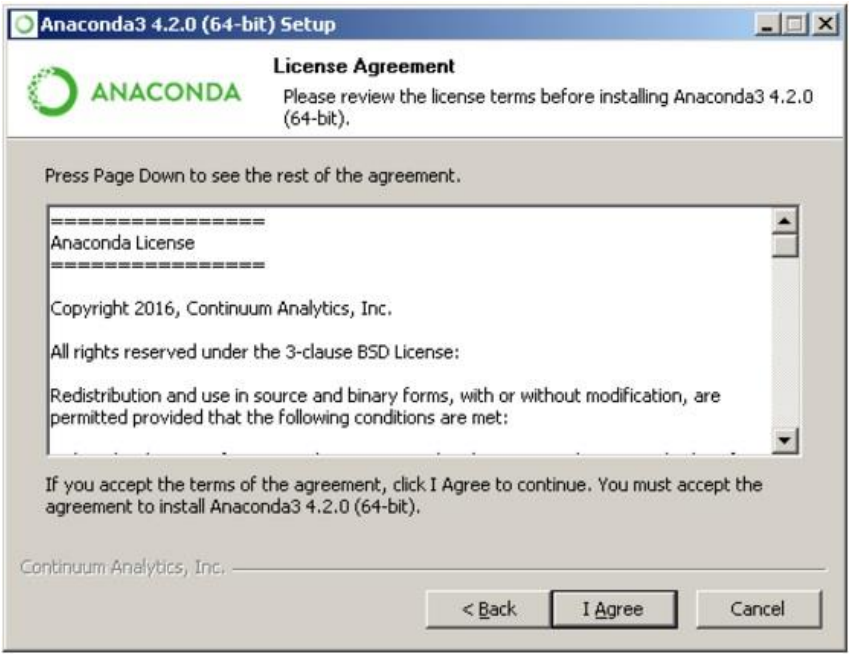
Щоб встановити цей пакет, спочатку потрібно завантажити дистрибутив https://www.continuum.io/downloads.

Є варіанти для Windows, Linux і macOS.

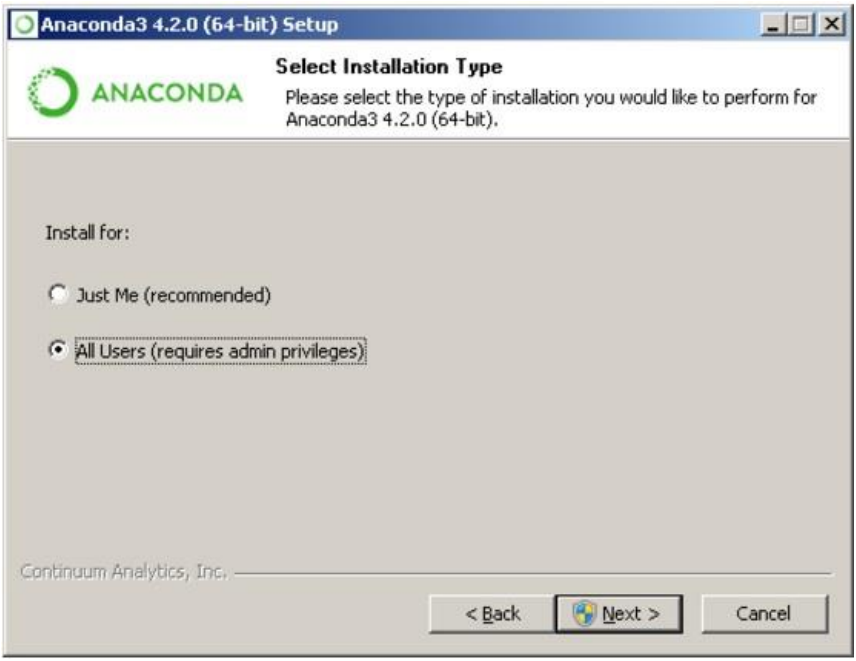
3.1 Установка Анаконди на Windows

1. Запустіть завантажений установник. У першому вікні натисніть "Далі".

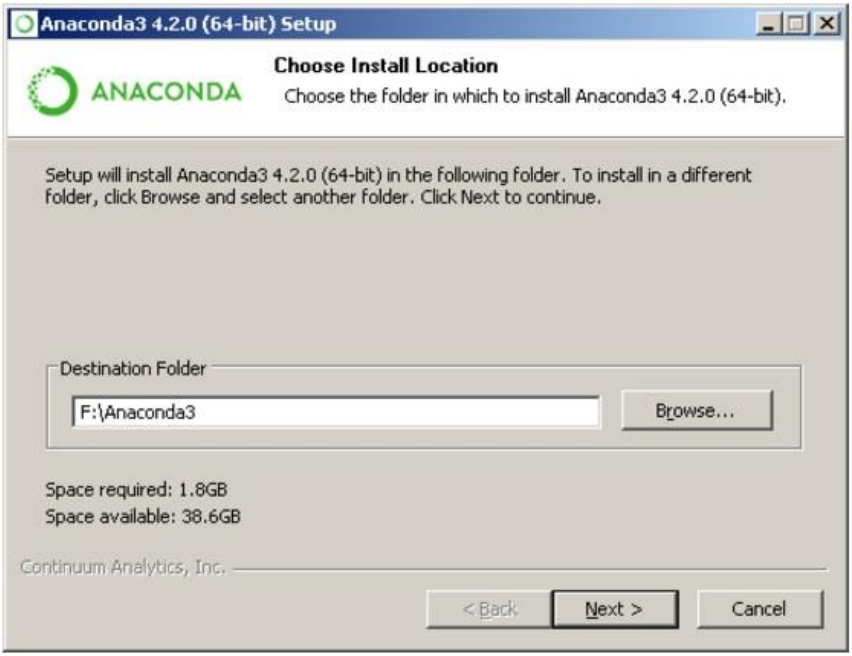
2. Далі слід прийняти ліцензійну угоду.



3. Виберіть один з варіантів установки:
Just Me – тільки для користувача, який почав установку;
Усі користувачі – для всіх користувачів.



4. Вкажіть шлях, куди буде встановлена Анаконда.

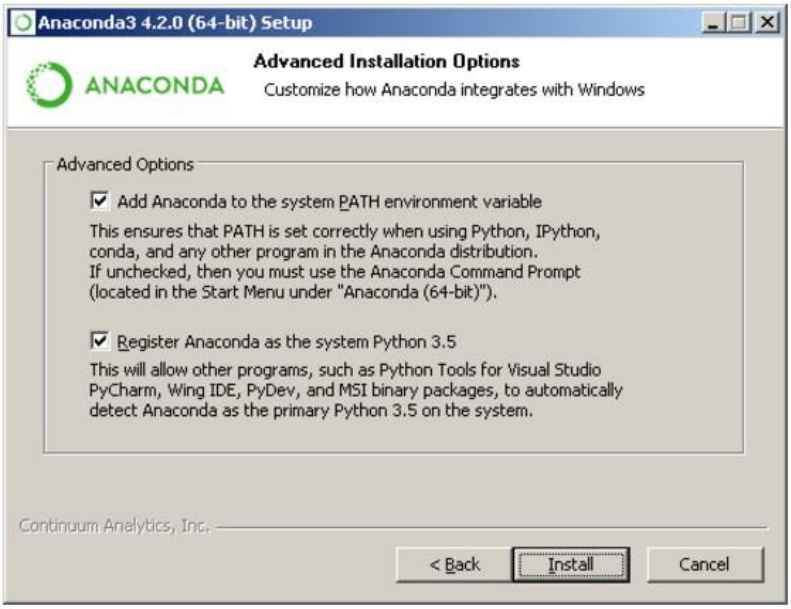


5. Вкажіть додаткові опції:

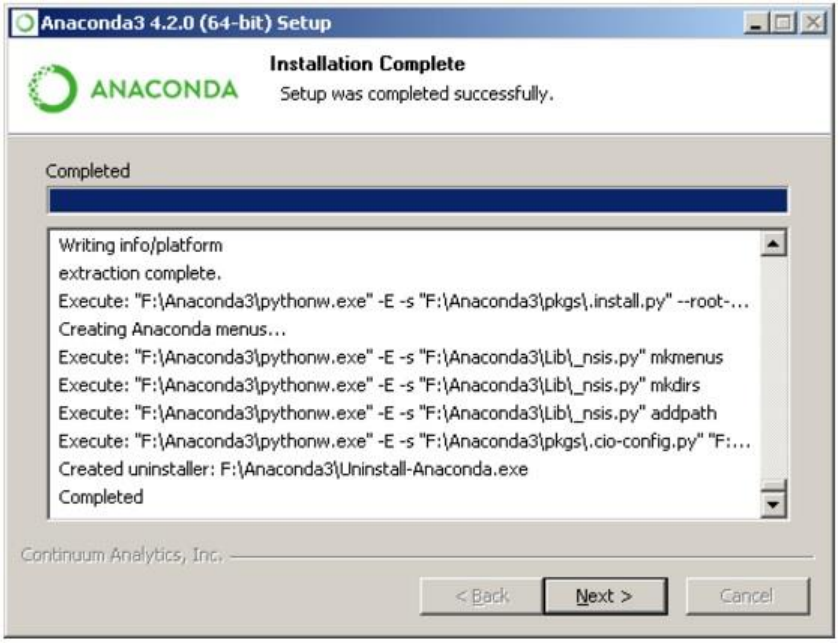
Add Anaconda в систему PATH environment variable – добавить Anaconda в системну переменную PATH

Зареєструйте Anaconda як систему Python 3.5 - Використовуйте Anaconda як інтерпретатор Python 3.5 за замовчуванням.

Щоб почати установку, натисніть на кнопку «Встановити».



5. Після цього на ваш комп'ютер встановиться Anaconda.

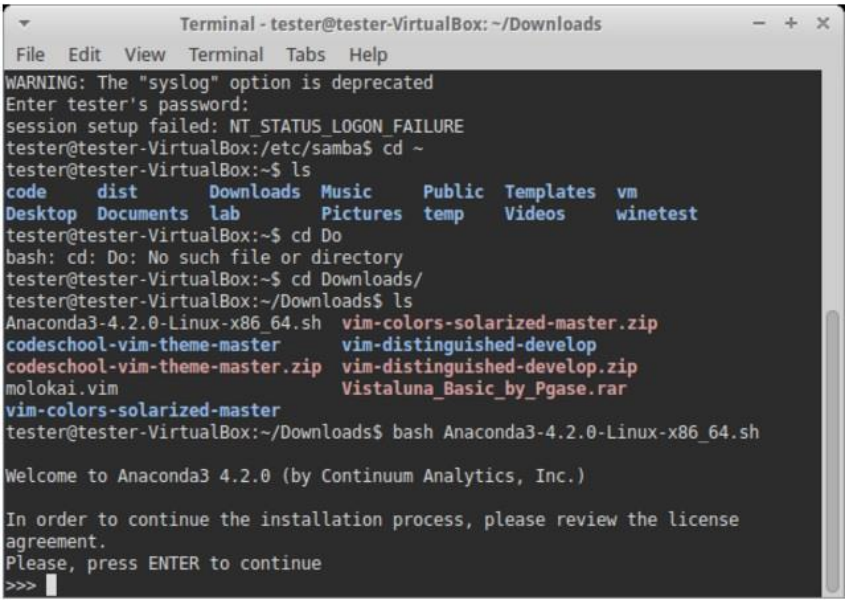


3.2 Установка Анаконди на Linux

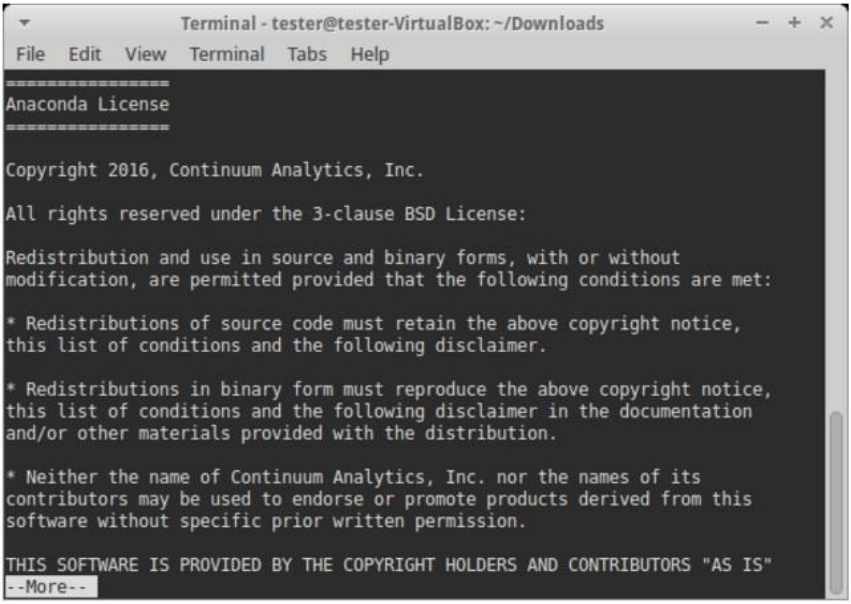
1. Скачайте дистрибутив Anaconda для Linux, в ньому буде розширення .sh, і запустіть команду установки:

```
> bash ім'я_distribution.sh
```

В результаті ви побачите підказку про встановлення. Для продовження процесу натисніть «Enter».

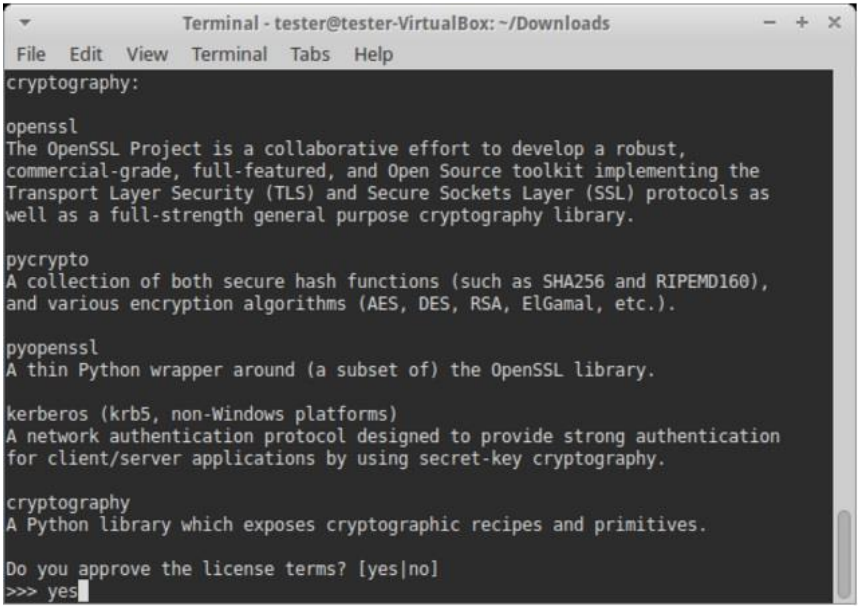


2. Прочитайте ліцензійну угоду, потрібно прокрутити до кінця.



Погодьтеся з ним, для цього потрібно набрати «так» в командному рядку, у відповіді на питання установника:

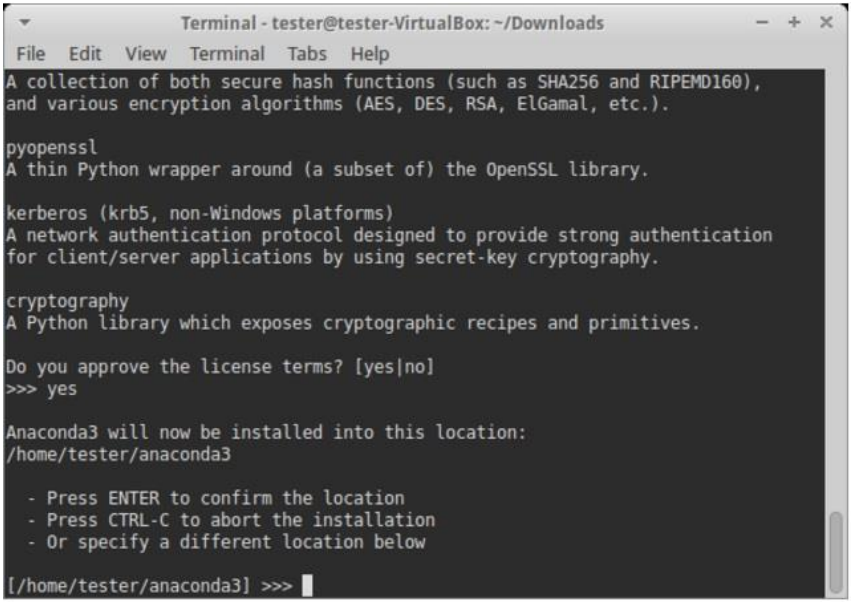
Чи затверджуєте ви умови ліцензії? [так|ні]



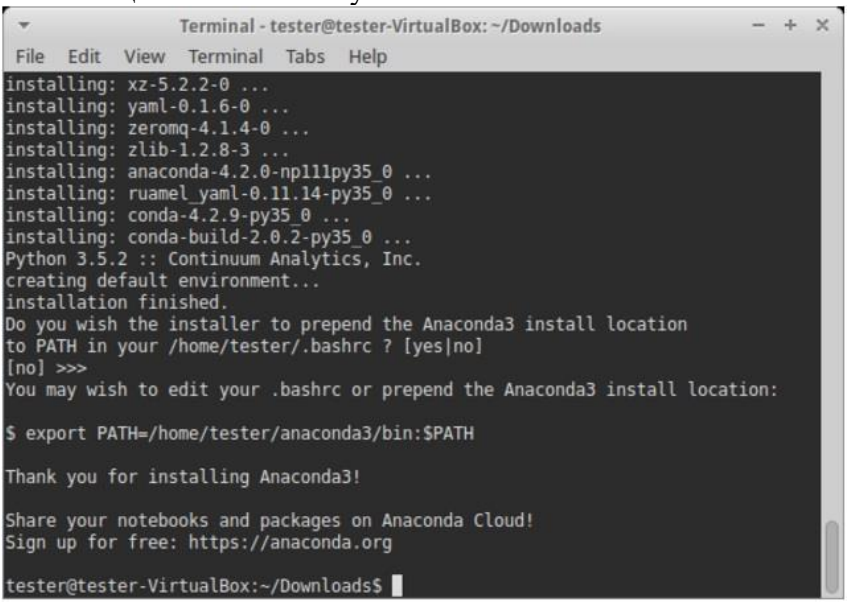
3. Виберіть місце установки. Можна вибрати один з наступних параметрів:

- *Натисніть ENTER, щоб підтвердити місцезнаходження* – Натисніть ENTER, щоб прийняти запропонований шлях встановлення. Шлях за замовчуванням для моєї машини - /home/tester/anaconda3, трохи вище цього меню.
- *Натисніть ctrl-C, щоб перервати інсталяцію* – натисніть CTRL-C, щоб скасувати інсталяцію.
- *Або вкажіть інше місце нижче* – або вкажіть інший шлях у рядку нижче.

Натисніть клавішу Enter.



4. Після цього почнеться установка.



4. Встановіть PyCharm

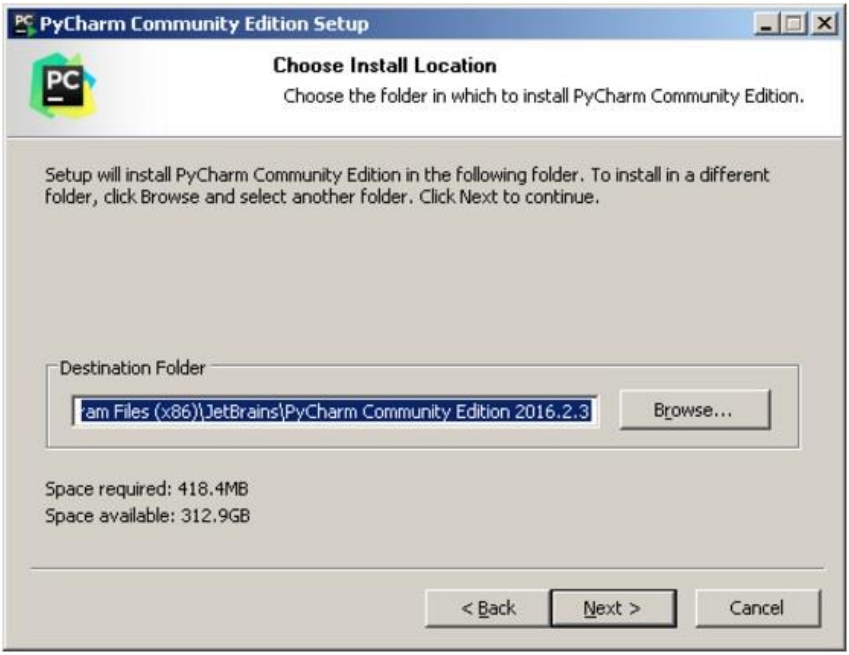
Якщо вам потрібен відлагоджувач в процесі розробки і взагалі ви звикли працювати в IDE, а не в текстовому редакторі, то одним з кращих варіантів стане PyCharm IDE від JetBrains. Щоб завантажити цей продукт потрібно перейти за посиланням https://www.jetbrains.com/pycharm/download/

IDE доступна для Windows, Linux і macOS. Існує два типи ліцензії PyCharm – Professional и Community. Ми будемо використовувати версію Community, оскільки вона безкоштовна і її функціоналу більш ніж достатньо для наших завдань.

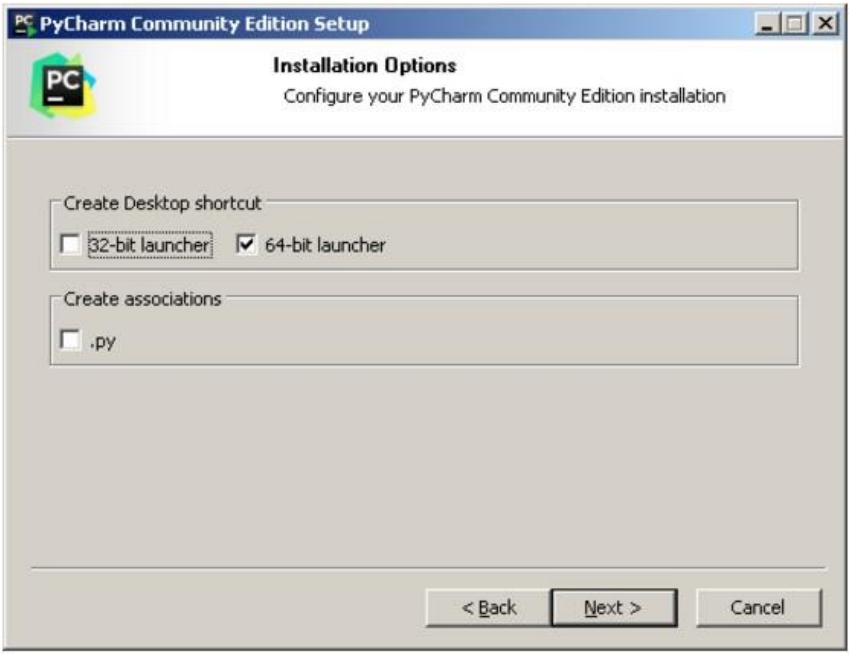
4.1 Установка PyCharm на Windows

1. Запустіть завантажений дистрибутив PyCharm.

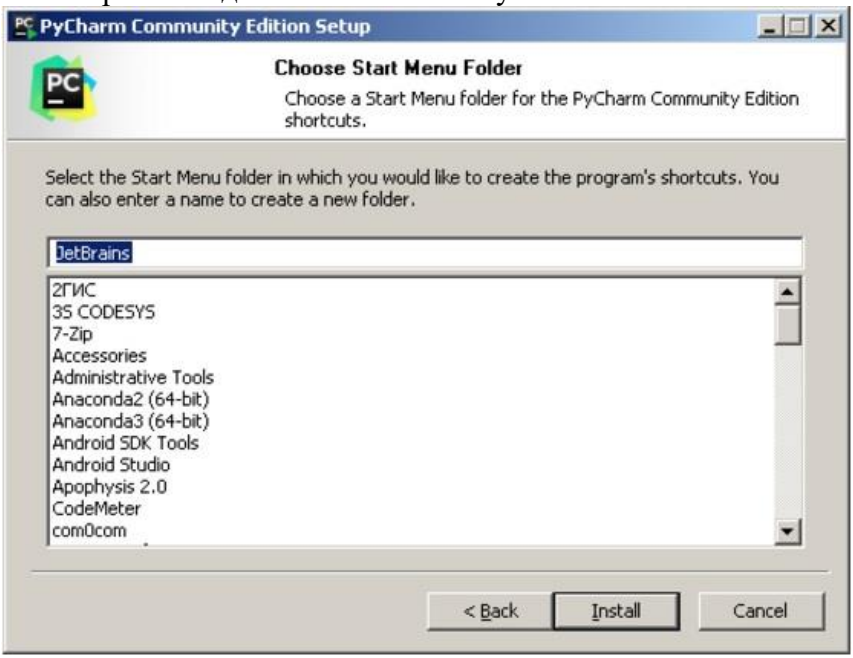
2. Виберіть шлях установки програми.



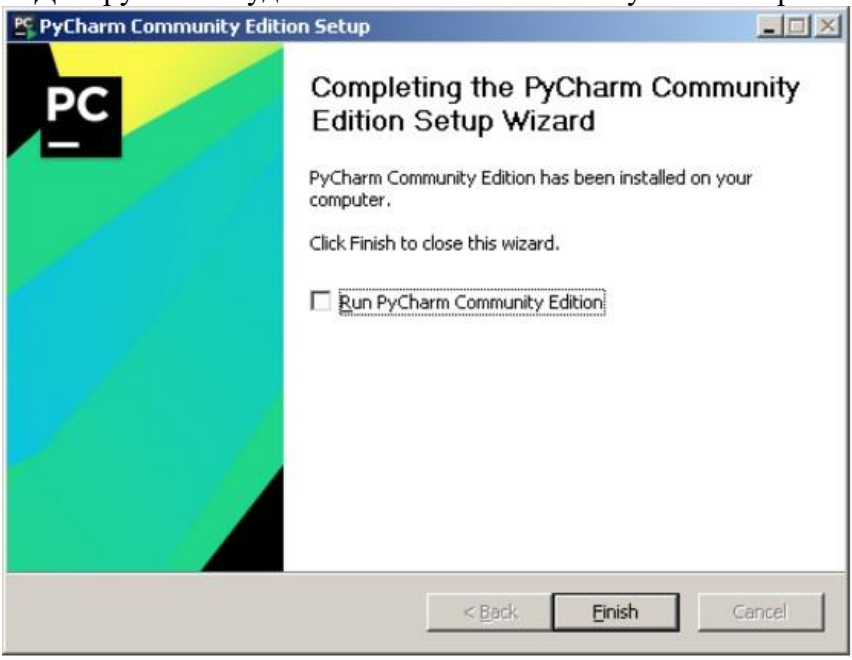
3. Вкажіть ярлики, які ви хочете створити на робочому столі (під управлінням 32-бітної і 64-бітної версій PyCharm) і поставте галочку на опції з блоку Створити асоціації, якщо ви хочете пов'язати файли з розширенням .py з PyCharm.



4. Виберіть ім'я для папки в меню Пуск.



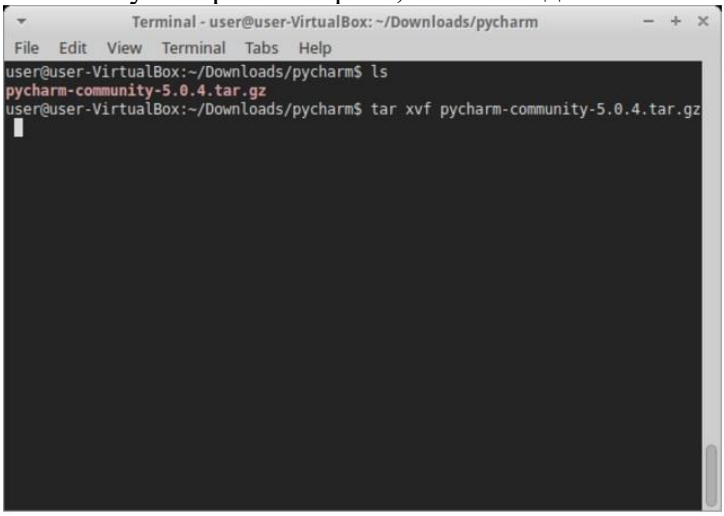
5. Далі pyCharm буде встановлений на вашому комп'ютері.



4.2 Встановлення PyCharm на Linux

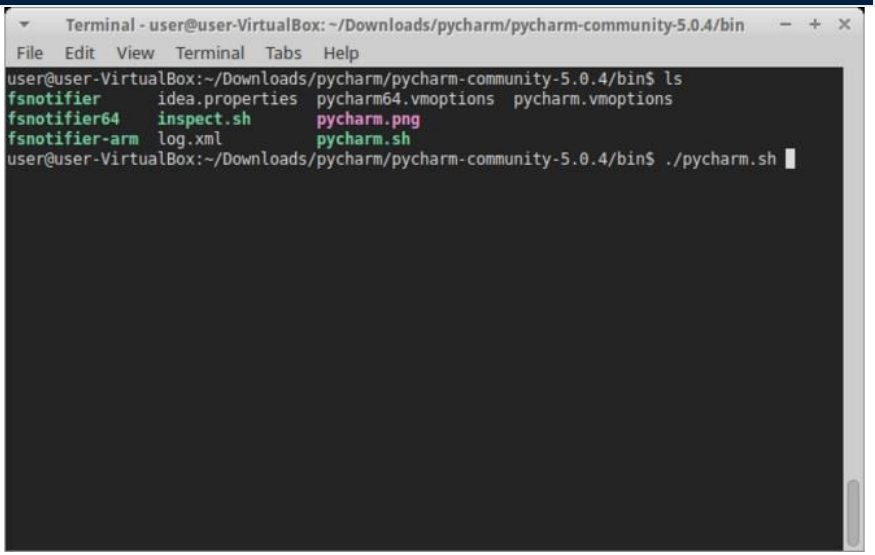
1. Скачайте дистрибутив з сайту на свій комп'ютер.

2. Розпакуйте архівний файл, можна за допомогою команди:



Зайдіть в директорію, яка була створена після розпакування дистрибутива, знайдіть в ній підкаталог bin і перейдіть в нього. Команда "Виконати pycharm.sh":

```
> ./pycharm.sh
```



З цього слід почати PyCharm.

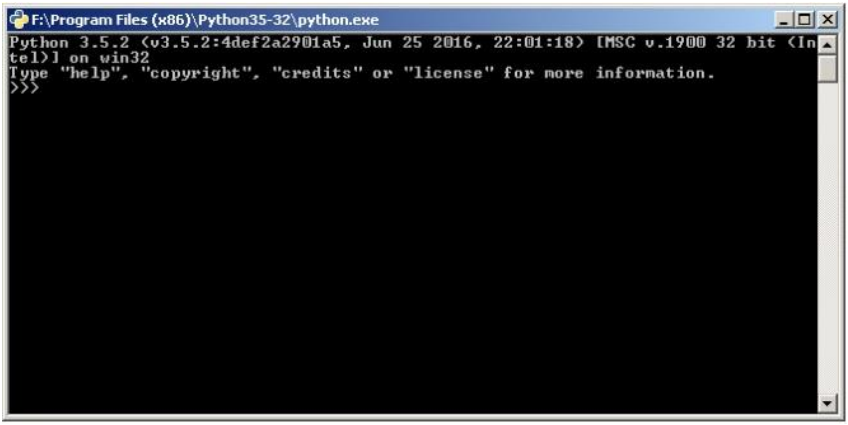
5. Перевірка стану роботи

Тепер давайте перевіримо працездатність всього, що ми встановили.

5.1 Тестування інтерпретатора Python

Для початку протестуємо інтерпретатор в командному режимі. Якщо ви працюєте в Windows, то натисніть комбінацію win + R і у вікні введіть python. У Linux відкрийте вікно терміналу і введіть в нього python3 (або python).

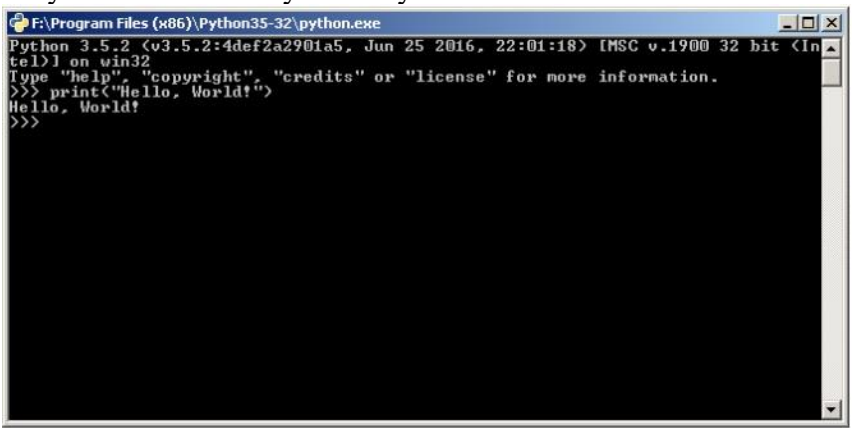
В результаті Python запуститься в командному режимі, це буде виглядати приблизно так (картинка дана для Windows, в Linux результат буде аналогічним):



У вікні введіть:

```
print("Hello, World!")
```

Результат повинен бути наступним:



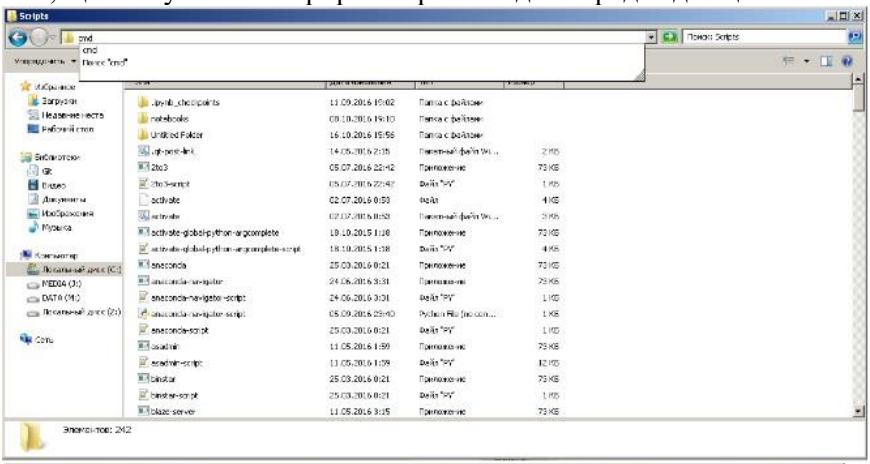
5.2 Перевірка анаконди

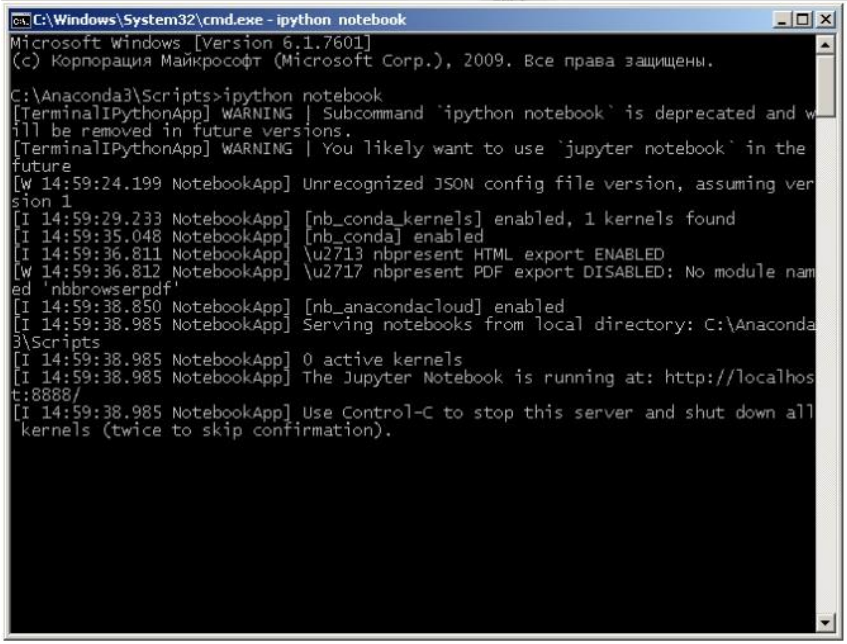
Тут і нижче будемо вважати, що пакет Anaconda встановлений в Windows, в папці C:\Anaconda3, в Linux його можна знайти в директорії, яку ви вибрали при установці.

Перейдіть до папки Scripts і введіть у командному рядку:

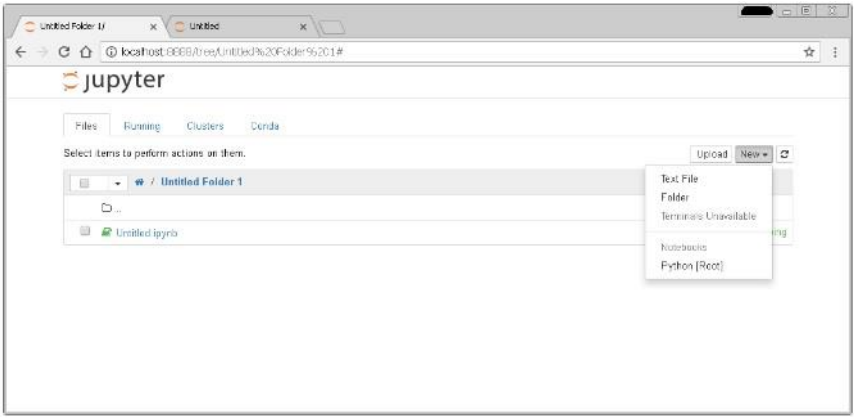
```
ipython notebook
```

Якщо ви перебуваєте у Windows і відкрили папку C:\Anaconda3\Scripts через Файловий провідник, введіть cmd у поле адреси, щоб запустити інтерпретатор командного рядка для цієї папки.





Це запускає веб-сервер і середовище розробки в браузері.

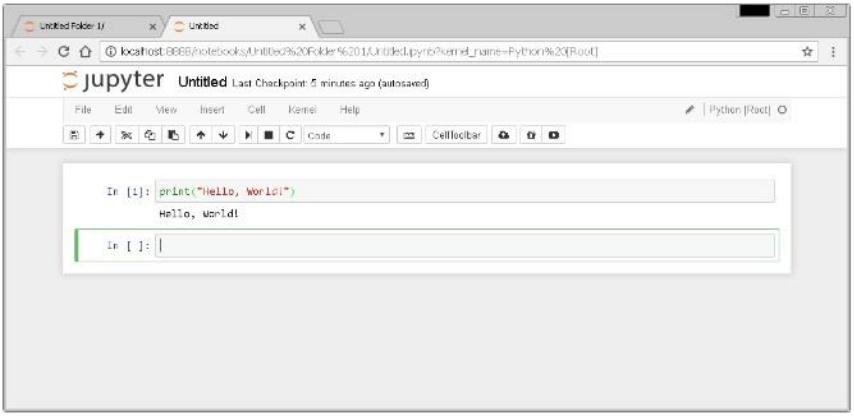


Створіть ноутбук для розробки, натисніть на кнопку New (в правому кутку вікна) і виберіть в списку Python.

Це створить нову сторінку в браузері вашого ноутбука. Введіть команду в першій клітинці

```
print("Hello, World!")
```

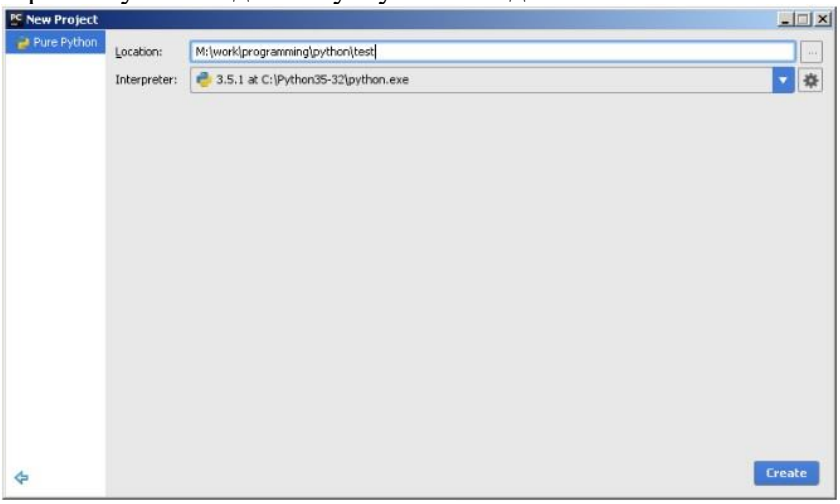
і натисніть клавіші Alt+Enter на клавіатурі. Під осередком повинен з'явитися відповідний напис.



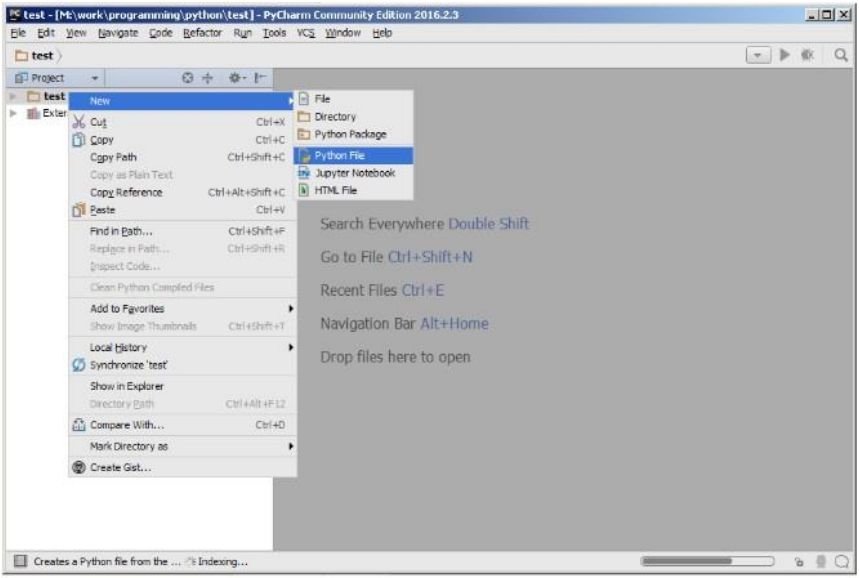
5.3 Перевірка PyCharm

Запустіть PyCharm і виберіть «Створити новий проект» у вікні.

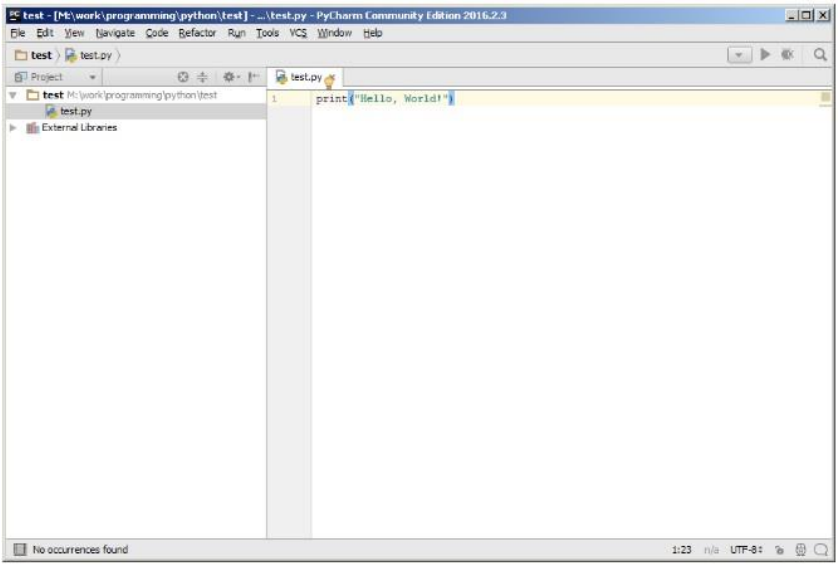
Вкажіть шлях до проекту Python і інтерпретатора, який буде використовуватися для запуску і налагодження.



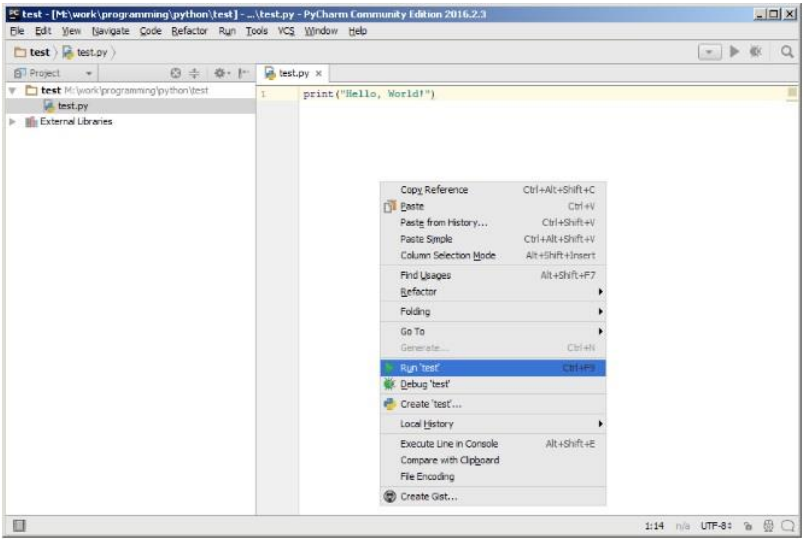
Додайте файл Python до проекту.



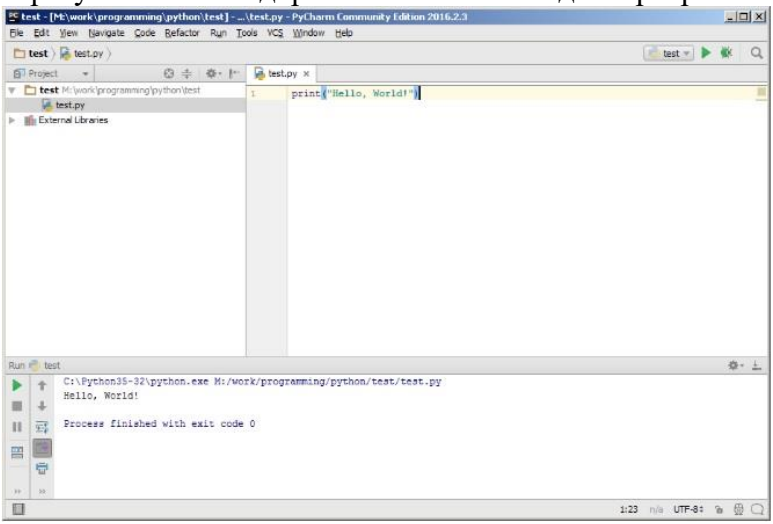
Введіть програмний код.



Запустіть програму.



В результаті має відкритися вікно з виходом програми.



1.5. Контрольні запитання

1. Що таке Python?
2. Які є версії Python?
3. Яка найбільш вживана версія Python?
4. Які є IDE для Python?
5. Що таке Anaconda?

Практична робота №2. Работа с IPython і Jupyter Notebook

2.1. Мета роботи

Навчитися встановлювати та працювати з IPython і Jupyter Notebook.

2.2. Теоретичні відомості

IPython - потужний інструмент для роботи з мовою Python. Базовими компонентами IPython є інтерактивна оболонка для багатого набору функцій і ядро для Юпітера. Ноутбук Jupyter - це графічна веб-обгортка для IPython, яка розширює ідею консольного підходу до інтерактивних обчислень.

Основними відмітними особливостями цієї платформи є складний самоаналіз об'єктів, збереження вхідної історії протягом усіх сеансів, кешування вихідних результатів, розширювана система «чарівних» команд, журналювання сеансів, додатковий синтаксис команд, виділення коду, доступ до системної оболонки, стикування з налагоджувачем pdb і профайлером Python.

IPython дозволяє кільком клієнтам підключатися до одного обчислювального ядра і, завдяки своїй архітектурі, може працювати в паралельному кластері.

У Jupyter notebook можна розробляти, документувати і виконувати додатки на Python, він складається з двох компонентів: веб-додатки, що працює в браузері, і ноутбуків - файлів, в яких можна працювати з вихідним кодом програми, запускати його, вводити і виводити дані і т.д.

Веб-додаток дозволяє:

- редагувати код Python в браузері, з підсвічуванням синтаксису, автопостами і автозаповненням;
- запускати код в браузері.
- відображення результатів розрахунків з медіа-представленням (схеми, графіки);
- працювати з мовою розмітки Markdown і LaTeX.

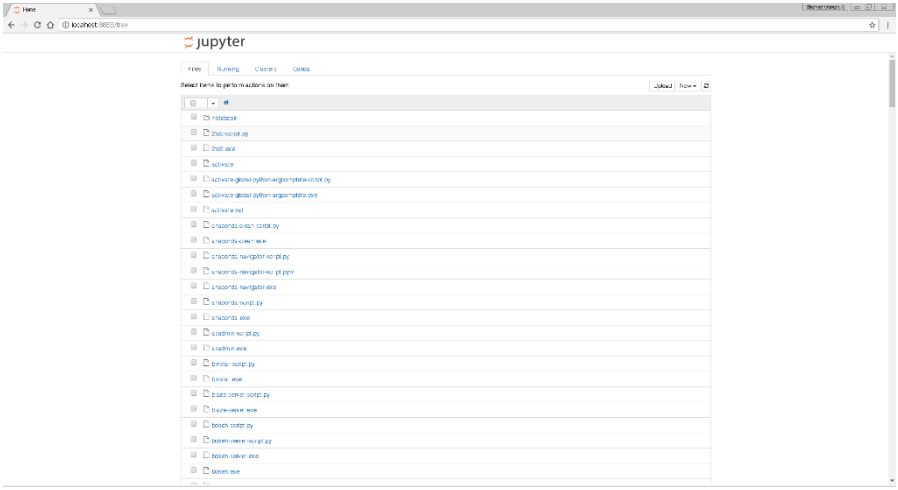
Ноутбуки - це файли, в яких зберігається вихідний код, входи і виходи, отримані в рамках сеансу. По суті, це запис вашої роботи, але вона дозволяє заново виконати код, присутній на ньому. Ноутбуки можна експортувати в формати PDF, HTML.

2.3. Порядок виконання роботи

Установка і запуск

Jupyter Notebook є частиною Anaconda. Опис процесу установки можна знайти в лаб. роб №1. Щоб запустити Jupyter Notebook, перейдіть в папку Scripts (вона знаходиться всередині директорії, куди встановлена Anaconda) і в типі командного рядка:

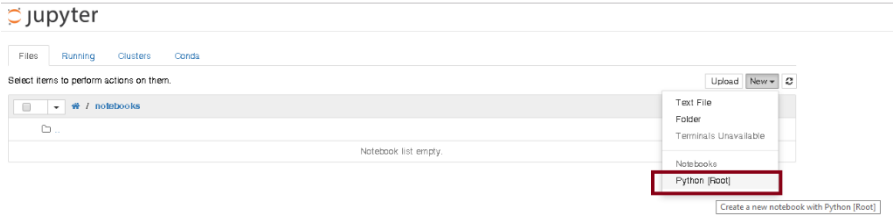
```
> ipython notebook
```



Запустіть блокнот Jupyter і створіть папку для наших прикладів, натисніть на New в правій частині екрана і виберіть Папка зі спадного списку.

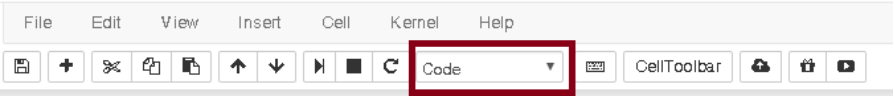
За замовчуванням папка носить ім'я "Untitled folder", перейменуйте її в "notebooks": перевірте ім'я папки і натисніть на кнопку "Перейменувати".

Зайдіть в цю папку і створіть в ній ноутбук за допомогою тієї ж кнопки New, тільки на цей раз потрібно вибрати "Python [Root]".

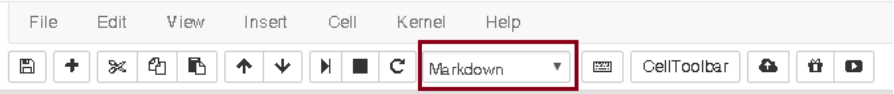


Код Python або текст в нотації Markdown необхідно вводити в осередку:

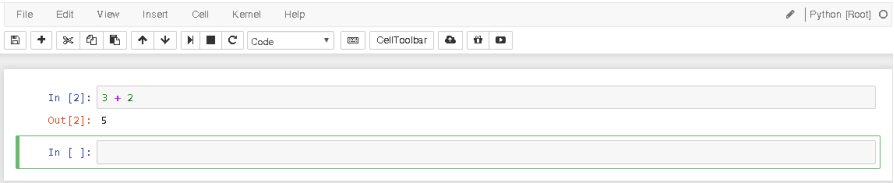
Якщо це код Python, то потрібно встановити властивість «Code» на панелі інструментів.



Якщо це текст Markdown, поставте "Markdown".



Для початку вирішимо просту арифметичну задачу: задаємо властивість «Code», вводите в осередку без лапок «2 + 3» і натискаємо Ctrl + Enter або Shift + Enter, в першому випадку введений вами код буде виконуватися інтерпретатором Python, у другому - буде виконаний код і створена нова осередок, яка буде розташовуватися на рівні нижче, як показано на рисунку.

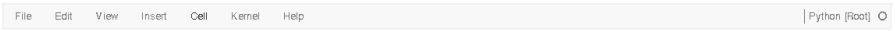


Якщо ви здатні це зробити, наведіть ще кілька прикладів.

Основні елементи інтерфейсу ноутбука Jupyter

Кожен ноутбук має назву, воно з'являється у верхній частині екрану. Щоб змінити ім'я, клікніть по його поточному імені і введіть нове.

З елементів інтерфейсу можна вибрати рядок меню:

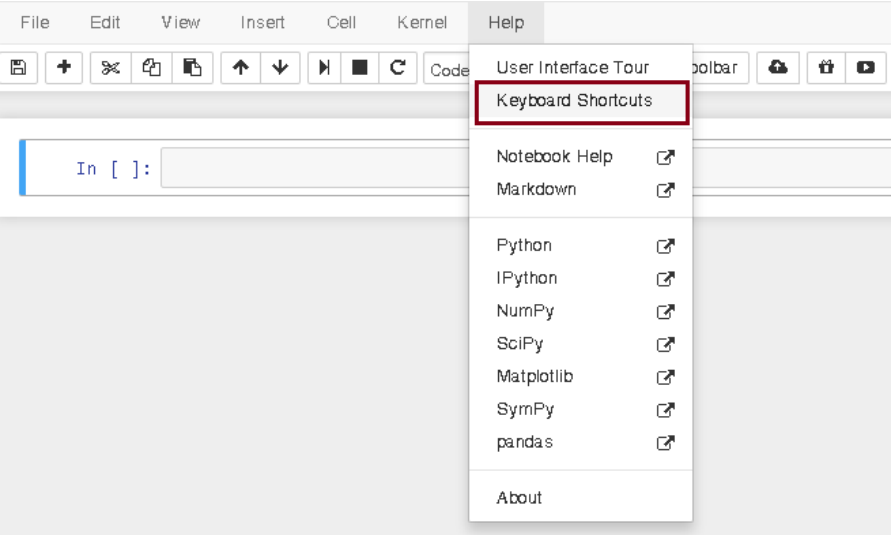


Панель інструментів:

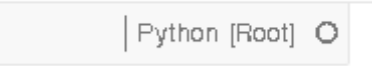
і робоче поле з осередками:

Ноутбук може перебувати в одному з двох режимів - це режим редагування (Edit mode) і режим команд (Command mode). Поточний режим відображається в рядку меню з правого боку, в режимі редагування з'являється зображення олівця, відсутність цього значка означає, що ноутбук знаходиться в командному режимі.

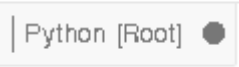
Щоб відкрити вікно «Довідка зі сполучень клавіш», натисніть кнопку «> клавіші клавіш»



У правій частині рядка меню знаходиться індикатор використання ядра python. Якщо ядро знаходиться в режимі очікування, індикатором є коло.



Якщо він виконає якесь завдання, то зображення зміниться на намальований коло.



Виконання коду запуску та переривання

Якщо ваша програма зависає, ви можете перервати її виконання, вибравши ядро -> Interrupt в рядку меню.

Щоб додати нову клітинку, скористайтеся командами "Вставити->Встановлена клітинка зверху" та "Вставити >Встановлена клітинка внизу".

Щоб запустити клітинку, скористайтеся командами з меню

"Клітинка" або скористайтеся такими сполученнями клавіш:

Ctrl+Enter – виконати вміст клітинки.

Shift+Enter – виконайте вміст комірки і перейдіть до клітинки нижче.

Alt+Enter – виконайте вміст комірки та вставте нову клітинку нижче.

Як зробити ноутбук доступним для інших людей?

Є кілька способів поділитися своїм ноутбуком з іншими людьми, щоб їм було комфортно з ним працювати:

Передайте безпосередньо файл ноутбука, який має розширення ".ipynb", а відкрити його можна тільки за допомогою Jupyter Notebook;

конвертувати ноутбук в html;

використовувати https://gist.github.com/;

використовувати http://nbviewer.jupyter.org/.

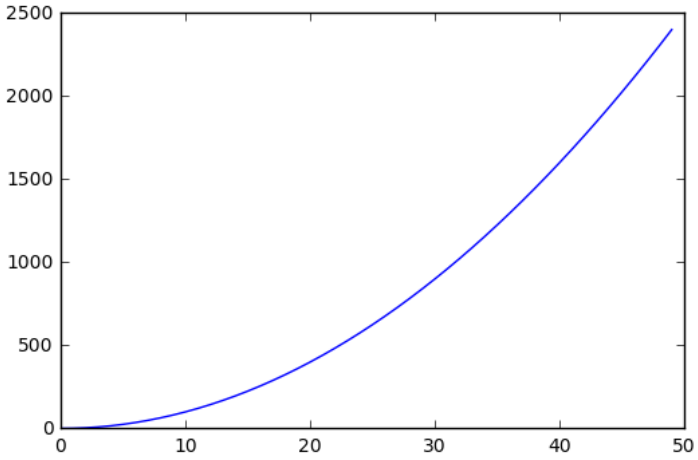
Відображення зображень у ноутбуці

Друк зображень може бути корисним, якщо ви використовуєте бібліотеку matplotlib для побудови графіків. За замовчуванням графіки в робочому полі ноутбука не відображаються. Для того щоб графіки відображалися, необхідно ввести і виконати наступну команду:

%matplotlib inline

Приклад виведення графіка показаний на малюнку нижче.

```
In [1]: from matplotlib import pylab as plt
        %matplotlib inline

In [2]: x = [i for i in range(50)]
        y = [i**2 for i in range(50)]
        plt.plot(x, y)

Out[2]: [<matplotlib.lines.Line2D at 0x7aa1b70>]
```



Магія

Важливою частиною функціоналу Jupyter Notebook є підтримка магії. Магія в IPython відноситься до додаткових команд оболонки, які спрощують процес розробки і розширюють ваші можливості. Список доступних магічних команд можна отримати за допомогою команди

%lsmagic

```
In [1]: %lsmagic

Out[1]: Available line magics:
        %alias  %alias_magic  %autocall  %automagic  %autosave  %bookmark  %cd  %clear  %cls  %colors  %config  %connect_info  %cop
        y  %ddir  %debug  %dhist  %dirs  %doctest_mode  %echo  %ed  %edit  %env  %gui  %hist  %history  %killbgscripts  %ldir  %les
        s  %load  %load_ext  %loadpy  %logoff  %logon  %logstart  %logstate  %logstop  %ls  %lsmagic  %macro  %magic  %matplotlib
         %mkdir  %more  %notebook  %page  %pastebin  %pdb  %pdef  %pdoc  %pfile  %pinfo  %pinfo2  %popd  %pprint  %precision  %prof
        ile  %prun  %psearch  %psource  %pushd  %pwd  %pycat  %pylab  %qtconsole  %quickref  %recall  %rehashx  %reload_ext  %ren
         %rep  %rerun  %reset  %reset_selective  %rmdir  %run  %save  %sc  %set_env  %store  %sx  %system  %tb  %time  %timeit  %un
        alias  %unload_ext  %who  %who_ls  %whos  %xdel  %xmode

        Available cell magics:
        %%!  %%HTML  %%SVG  %%bash  %%capture  %%cmd  %%debug  %%file  %%html  %%javascript  %%js  %%latex  %%perl  %%prun  %%pypy
         %%python  %%python2  %%python3  %%ruby  %%script  %%sh  %%svg  %%sx  %%system  %%time  %%timeit  %%writefile

        Automagic is ON, % prefix IS NOT needed for line magics.
```

Для роботи зі змінними середовища використовуйте команду

%env.

```
In [3]: %env TEST = 5
        env: TEST=5
```

Код Python запускається з файлів ".py", а також з інших ноутбуків - файлів з розширенням ".ipynb", здійснюється за допомогою команди %run.

```
In [5]: %run ./test.py
        Hello
        Hello
        Hello
        Hello
        Hello
```

Щоб виміряти час безвідмовної роботи коду, використовуйте %%time і %timeit.

%%time дозволяє отримати інформацію про час виконання коду в межах однієї комірки.

```
In [2]: %%time
        import time
        for i in range(50):
            time.sleep(0.1)
        Wall time: 5.45 s
```

2.5. Контрольні запитання

1. Що таке Jupyter?
2. Що дозволяє робити Веб-додаток Jupyter?
3. Опишіть процедуру встановлення Jupyter Notebook.
4. Ля чого використовується бібліотека matplotlib?
5. Що таке «магія» в Ipython?

Практична робота №3. Лінійна регресія

3.1. Мета роботи

Набути навичок використовувати лінійну регресію для практичних задач.

3.2. Теоретичні відомості

Уявіть, що у вас є набір об'єктів X, і ви хотіли б зіставити значення з кожним об'єктом. Наприклад, у вас є набір операцій на банківській карті, і ви хотіли б зрозуміти, які з цих операцій були здійснені шахраями. Якщо розділити всі операції на два класи і нуль позначити законні дії, а одиниця шахрайська, то перед вами найпростіше завдання класифікації. Уявіть іншу ситуацію: у вас є дані розвідки, з яких ви хотіли б оцінити перспективи різних родовищ. В цьому випадку, використовуючи набір геологічних даних, ваша модель дозволить, наприклад, оцінити потенційну річну рентабельність шахти. Це приклад проблеми регресії. Числа, яким ми хочемо зіставити об'єкти з нашого набору, іноді називаються цілями (від англійської **мішені**).

Таким чином, проблеми класифікації та регресії можна сформулювати як пошук відображення від сукупності об'єктів X до набору можливих цілей.

Математично проблеми можна описати так:

- **класифікація**: $X \to \{0,1,...,K\}$, де 0,...,K - номери класів,
- **Регресія**: $X \to R$.

Очевидно, що просто зіставляти деякі об'єкти з деякими числами - справа досить безглузда. Ми хочемо швидко виявити шахраїв або вирішити, де будувати шахту. Так що потрібен якийсь критерій якості. Ми хотіли б знайти відображення, яке найкраще апроксимує справжнє відображення між об'єктами та цілями. Що означає "кращий" - питання складне. Однак виникає і більш просте питання: серед яких карт ми будемо шукати кращі? Можливих відображень може бути багато, але ми можемо спростити собі завдання і погодитися з тим, що хочемо шукати рішення тільки в якомусь заданому параметризованому сімействі функцій. Вся ця глава буде присвячена найпростішим таким сімейства, лінійним функціям виду

$$y=w1x1+...+wDxD+w0,$$

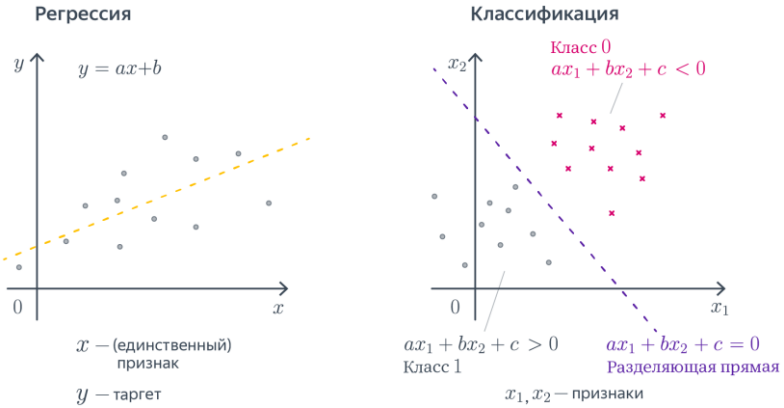
де y - цільова змінна (**ціль**), (x1,...,xD) - вектор, що відповідає об'єкту вибірки (**вектор** ознаки), а w1,...,wD,w0 є параметрами моделі. Особливості також називаються **ознаками**. Вектор w=(w1,...,wD) часто називають ваговим вектором, оскільки передбачення моделі можна розглядати як зважену суму ознак об'єкта, а число w0w0 є вільним коефіцієнтом, або **зміщенням**. Більш компактно лінійну модель можна записати як

$$y=\langle x,w\rangle+w0y=\langle x,w\rangle+w0$$

Тепер, коли ми вибрали сімейство функцій, в якому будемо шукати рішення, завдання стала набагато простіше. Ми вже не шукаємо якесь абстрактне відображення, а конкретний вектор $(w0,w1,...,wD)\in RD+1$.

Примітка: Для застосування лінійної моделі необхідно, щоб кожен об'єкт вже був представлений вектором числових ознак x1,...,xD. Звичайно, в лінійну модель просто не можна ставити текст або графік, спочатку доведеться придумати для нього числові ознаки. Модель називається лінійною, якщо вона лінійна за цими числовими ознаками.

Давайте подивимося, як буде працювати така модель, якщо D=1. Тобто наші об'єкти мають саме одну числову ознаку, за якою вони відрізняються. Тепер наша лінійна модель буде виглядати досить просто: y=w1x1+w0. Для задачі регресії ми зараз намагаємося апроксимувати значення гри якоюсь лінійною функцією від змінної x. А що означатиме лінійність для задачі класифікації? Згадаємо приклад пошуку шахрайських операцій по Картах. Скажімо, ми точно знаємо одну числову змінну, обсяг транзакції. Для бінарної класифікації угод на законні і потенційно шахрайські угоди будемо шукати так зване **правило поділу**: там, де значення функції позитивне, ми будемо прогнозувати один клас, де негативний - інший. У нашому прикладі найпростішим правилом буде якесь порогове значення обсягу операцій, після чого має сенс позначити транзакцію як підозрілу.



У разі більш високих габаритів замість прямої буде гіперплощина з аналогічним значенням.

Питання, над яким варто задуматися. Якщо ви подивитеся на зміст підручника, то не знайдете ні «поліноміальних» моделей, ні якихось «логарифмічних» моделей, хоча, здавалося б,

Це питання, над яким варто задуматися. А раптом одна з особливостей *категорична*, тобто бере значення з (зазвичай кінцевого числа) значень, які не є числами? Наприклад, це може бути пора року, рівень освіти, марка машини і так далі. Як правило, з такими значеннями неможливо виконувати арифметичні операції або результати їх застосування не мають сенсу.

	pet_type	color	weight
0	carrot	red	0.600000
1	cat	white	3.000000
2	hamster	brown	0.800000
3	cat	gray	5.000000
4	dog	black	7.000000

	weight	pet_type_carrot	pet_type_cat	pet_type_dog	pet_type_hamster	color_black	color_brown	color_gray	color_red	color_white
0	0.600000	1	0	0	0	0	0	0	1	0
1	3.000000	0	1	0	0	0	0	0	0	1
2	0.800000	0	0	0	1	0	1	0	0	0
3	5.000000	0	1	0	0	0	0	1	0	0
4	7.000000	0	0	1	0	1	0	0	0	0

	weight	pet_type_cat	pet_type_dog	pet_type_hamster	color_brown	color_gray	color_red	color_white
0	0.600000	0	0	0	0	0	1	0
1	3.000000	1	0	0	0	0	0	1
2	0.800000	0	0	1	1	0	0	0
3	5.000000	1	0	0	0	1	0	0
4	7.000000	0	1	0	0	0	0	0

Крім простоти, лінійні моделі мають ще ряд переваг. Наприклад, ми можемо досить легко судити про те, як ті чи інші ознаки впливають на результат. Наприклад, якщо вага wi позитивний, то з ростом ii-ї ознаки мета в разі регресії збільшиться, а в разі класифікації наш вибір зміститься на користь одного з класів. Значення ваг також має прозору інтерпретацію: чим більше вага wiwi, тим «важливіше» друга ознака для остаточного прогнозу. Тобто, якщо ви побудували лінійну модель, ви цілком можете пояснити замовнику деякі її результати. Якість моделей називається **інтерпретованістю**. Особливо вона цінується в промислових задачах, в яких ціна

помилки висока. Якщо життя людини може залежати від роботи вашої моделі, то дуже важливо зрозуміти, як модель приймає ті чи інші рішення і якими принципами керується. В той же час не всі методи машинного навчання добре інтерпретуються, наприклад, поведінка штучних нейронних мереж або градієнтний бустінг досить складно інтерпретувати.

При цьому сліпо довіряти масштабам лінійних моделей також не варто по ряду причин:

- Лінійні моделі - це ще досить вузький клас функцій, вони добре працюють для невеликих наборів даних і простих завдань. Однак якщо ви вирішуєте більш складну задачу з лінійною моделлю, то, швидше за все, доведеться придумувати додаткові можливості, які є складними функціями з вихідних. Пошук таких додаткових можливостей називається **feature engineering**. Але пошуком таких штучних ознак можна дуже захоплюватися, так що осмисленість інтерпретації буде сильно залежати від здорового глузду експерта, який побудував модель.
- При наявності приблизної лінійної залежності між ознаками коефіцієнти в лінійній моделі можуть повністю втратити свій фізичний сенс.
- Особливо уважно слід вірити твердженням виду «цей коефіцієнт невеликий, тому ця особливість не важлива». По-перше, все залежить від масштабу ознаки: раптом коефіцієнт малий, щоб компенсувати його. По-друге, залежність дійсно може бути слабкою, але хто знає, в якій ситуації це буде важливо. Такі рішення приймаються на основі даних, наприклад, шляхом перевірки статистичного критерію.
- Конкретні значення ваг можуть змінюватися в залежності від навчальної вибірки, хоча в міру збільшення її розміру вони будуть повільно сходитися до ваг «кращої» лінійної моделі, яка могла б бути побудована на універсальних даних в світі.

3.3. Порядок виконання роботи

```
import pandas as pd
#завантажуємо дані
```

```
data = pd.read_csv("https://gist.githubusercontent.com/nstokoe/7d4717e96c21b8ad04ec91f361b000cb/raw/bf95a2e30fceb9f2ae990eac8379fc7d844a0196/weight-height.csv")
data.head()
```

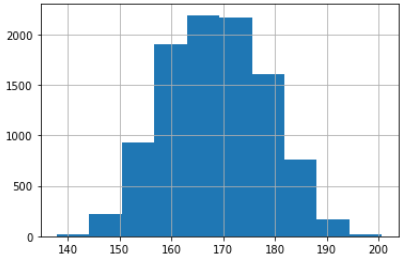
	Gender	Height	Weight
0	Male	73.847017	241.893563
1	Male	68.781904	162.310473
2	Male	74.110105	212.740856
3	Male	71.730978	220.042470
4	Male	69.881796	206.349801

```
#переводимj в см s кг
data['Height']*=2.54
data['Weight']*=0.4536
data.head()
```

	Gender	Height	Weight
0	Male	187.571423	109.722920
1	Male	174.706036	73.624030
2	Male	188.239668	96.499252
3	Male	182.196685	99.811265
4	Male	177.499761	93.600270

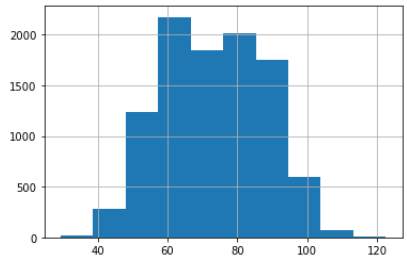
переглянемо розподіл даних
data['Height'].hist()

```
<AxesSubplot:>
```

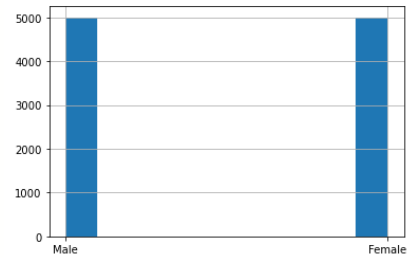


data['Weight'].hist()

```
<AxesSubplot:>
```




```
data['Gender'].hist()
<AxesSubplot:>
```



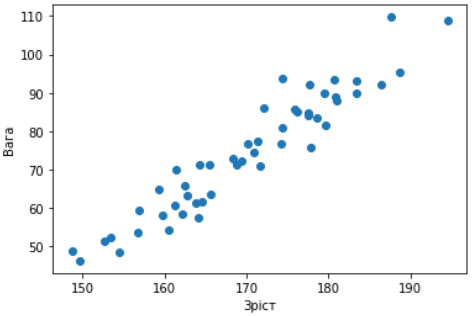

```
#скоротимо вибірку для наглядності N- номер варіанта
N=0
df=data[::N+200]
```

#виведемо структуру даних
df.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50 entries, 0 to 9800
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Gender  50 non-null     object
 1   Height  50 non-null     float64
 2   Weight  50 non-null     float64
dtypes: float64(2), object(1)
memory usage: 1.3+ KB
```

#задамо вхід і вихід моделі
X = df[['Height']]
y = df['Weight']

```
#побудуємо точковий графік
import matplotlib.pyplot as plt
plt.scatter(X, y)
plt.xlabel("Зріст")
plt.ylabel("Вага")
plt.show()
```



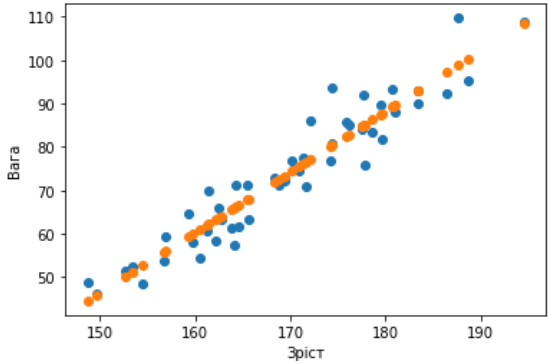

```
# побудуємо модель парної лінійної регресії виду y=w*x+b
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X, y) # навчання моделі
w=model.coef_
b=model.intercept_
w,b
```

```
(array([1.39252407]), -162.42219744720768)
```

y=1.39*x-162.4, де x-зріст в см, y-вага в кг

```
#знайдемо представлення моделі
y_pred=model.predict(X)

#нарисуємо графік вихідних даних і побудованої моделі
X = df[['Height']]
y = df['Weight']
plt.scatter(X, y)

plt.xlabel("Зріст")
plt.ylabel("Вага")
```

```
plt.scatter(X, y_pred)
plt.show()
```




```
# знайдемо точність побудованої моделі
model.score(X,y)
```

```
0.909534153356286
```

```
#закодуємо змінну Gender як 0 і 1
X1=df.drop(['Weight'], axis=1)
X1['Gender'] = pd.factorize(X1['Gender'])[0]
#побудуємо множинну регресію виду y=w1*x1+w2*x2+b
model1 = LinearRegression()
model1.fit(X1, y) # навчання моделі
w=model1.coef_
b=model1.intercept_
w,b
```

```
(array([-6.25690744,  1.18517264]), -
124.03451070427748)
```

y=-6.3 *x1+1.19 *x2 - 124, де x1-стать, x2-зріст в см, y-вага в кг

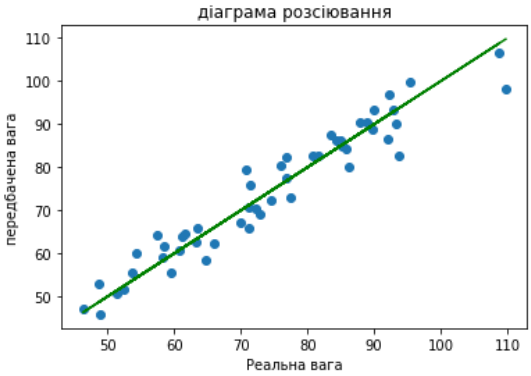
```
# знайдемо точність побудованої моделі
model1.score(X1,y)
```

```
0.9294561135335533
```

```
#знайдемо передбачення моделі
y_pred1=model1.predict(X1)
```

#нарисуємо діаграму розсіювання вихідних та передбачених значень ваги

```
plt.scatter(y, y_pred1)
plt.xlabel("Реальна вага")
plt.ylabel("передбачена вага")
plt.plot(y, y,'green')
plt.title('діаграма розсіювання')
plt.show()
```



3.5. Контрольні запитання

1. Що називають лінійною регресією?
2. Які задачі приводять до використання лінійної регресії?
3. Як визначеється похибка лінійної регресії?
4. Що називають інтерпретованістю?
5. Що потрібно для застосування лінійної моделі?

Практична робота №4. Аналіз даних

4.1 Мета роботи

Навчитися аналізувати дані (data mining) в *Python*.

4.2. Теоретичні відомості

Передбачення цін на нерухомість

Бізнес-постановка задачі

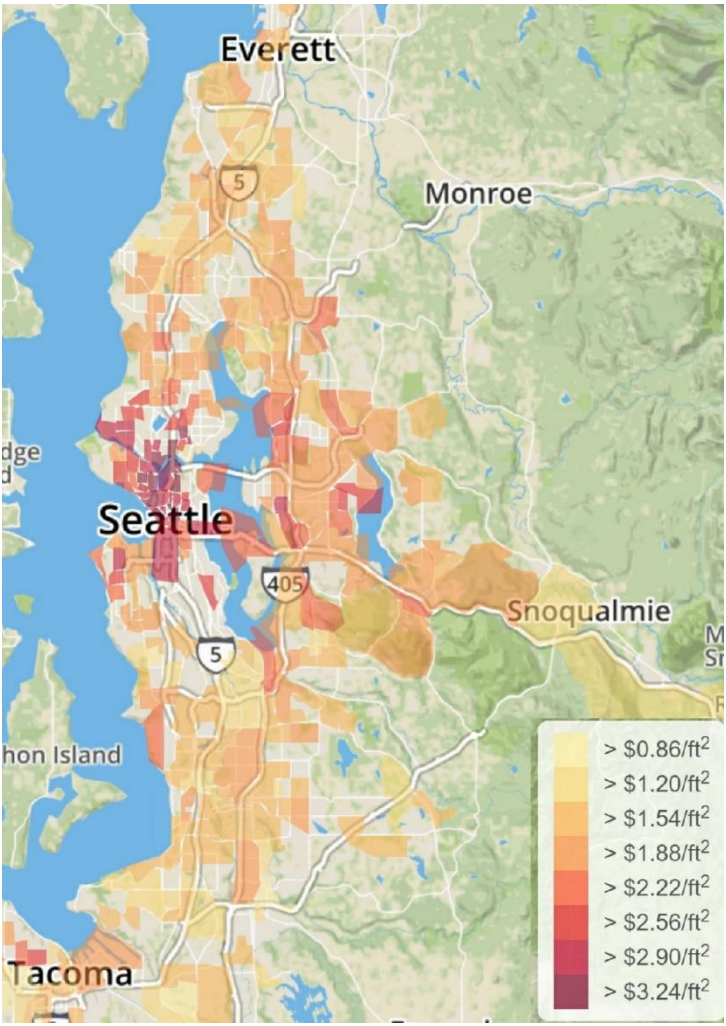
Компанії із продажу нерухомості оцінюють вартість, використовуючи методи машинного навчання. Завдання оцінки ціни на нерухомість також може бути необхідним для:

- виявлення аномально низьких цін на нерухомість;
- оцінки макроекономічних показників;
- уточнення ставок з іпотеки.

Постановка задачі аналізу даних

Метою даного завдання є прогнозування вартості будинків в окрузі Кінг (штат Вашингтон, США) за допомогою побудови регресійних моделей та їх аналізу. Набір даних складається з цін на будинки в окрузі Кінг, проданих у період із травня 2014 року по травень 2015 року. Дані опубліковані у відкритому доступі на платформі *Kaggle*.

Дані з сайту *renthub.com* за вартістю квартир для оренди в Сіетлі:



Огляд доступних даних

У вибірці 21613 спостережень та 21 змінна. Таким чином, про кожен з 21613 об'єктів нерухомості ми знаємо значення 21 їх характеристики (кількість спалень, оцінка стану рієлтором, наявність вигляду на воду і т.п.)

Вибірка була розбита на дві частини для навчання та для тестування моделі. Дані на навчання і тест зазвичай ділять випадково і рівномірно: потрібно, щоб і навчальна, і тестова вибірка були схожі на ті дані, на яких модель використовуватиметься в бойових умовах.

Більш складні розбиття можуть використовуватися якщо в даних є тимчасова компонента - наприклад, ми будуємо скорингову модель і для тесту відкладаємо дані за останні кілька місяців, а решту вибірку використовуємо для навчання. Таке розбиття імітує реальну роботу моделі: ми можемо навчатися тільки на даних минулих кредитів, а працювати модель вже буде на даних з майбутнього.

Частки загальної вибірки для навчання та тестування зазвичай 70% та 30% відповідно. Будь-які розумні числа підходять, якщо для навчання використовується достатньо даних (зазвичай понад 50%), але й для тестування щось залишається (10% і більше).

Дані містять два типи змінних:

• Цільова: **Цільова. Ціна**

• Інші змінні: **20 змінних можуть використовуватися для прогнозу цільової змінної.**

План аналізу даних (data mining):

1. Завантажити дані для навчання
2. Обробити дані перед навчанням моделі
3. Навчити модель на навчальній вибірці
4. Завантажити та обробити дані для тестування
5. Провалідувати модель на тестовій вибірці

4.3. Порядок виконання роботи

1. Завантажити дані для навчання

Крок 1.1. Завантажуємо бібліотеки

Бібліотека **warnings** відповідає за те, які попередження (warnings) про роботу виводитимуться користувачеві. FutureWarning – попередження про те, як зміниться робота бібліотек у майбутніх версіях. Тому такі попередження ми ігноруватимемо. Щоб увімкнути режим ігнорування, ми відбираємо всі попередження з категорії FutureWarning і вибираємо для них дії 'ignore'. Це робиться викликом функції simplefilter с завдання двох атрибутів: дії action та категорії попереджень category.

```
[ ]
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
```

Для коректної роботи з даними в python потрібно завантажити спеціальну бібліотеку **pandas**, програмну бібліотеку мовою python для обробки та аналізу даних.

```
[ ]
import pandas as pd # завантажуємо бібліотеку і для простоти звернення в коді називаємо її скорочено pd
```

Для коректної роботи з графіками в python потрібно завантажити спеціальну бібліотеку **matplotlib**, програмну бібліотеку мовою python для візуалізації даних двовимірною та тривимірною графікою.

Графіки використовуються для полегшення інтерпретації отриманих результатів, а також як ілюстрації у презентаціях та звітах.

Основні методи для побудови:

• *plot()* – графіки;

• *semilogy()* – логарифмічний графік ;

• *hist()* – гістограми.

```
[ ]
import matplotlib.pyplot as plt # завантажуємо
бібліотеку та для простоти звернення в коді називаємо
її скорочено plt
# вказуємо, щоб зображення виводилося прямо в ноутбуці
%matplotlib inline
```

Крок 1.2. Завантажимо дані

Для вирішення завдання ми будемо використовувати дані. Вони складаються з двох частин: частина для навчання та частина для тестування моделі. Завантажуємо дані за допомогою команди !wget. Для того, щоб ігнорувати повідомлення в процесі завантаження використовуємо магічну команду %%capture у першому рядку.

```
[ ]
%%capture
!wget https://www.dropbox.com/s/afwb0tnqm9izxha/predict_house_price_training_data.xlsx
!wget https://www.dropbox.com/s/sur2avqf4n5f4az/predict_house_price_test_data.xlsx
```

Оскільки дані у форматі xlsx (Excel), ми будемо використовувати спеціальну функцію бібліотеки pandas для завантаження таких даних *read_excel()*.

До функції передаємо один атрибут: назву таблиці з даними.

```
[ ]
training_data = pd.read_excel('predict_house_price_training_data.xlsx') #
завантажуємо таблицю в змінну training_data
```

Що важливо подивитися після того, як ми завантажили дані?

• перевірити, чи дані дійсно завантажилися;

• подивитися на дані, щоб переконатися, що вони правильні: колонки мають самі назви, що у таблиці тощо.

Щоб це зробити, потрібно викликати від змінної *training_data* метод **head**(), який виводить перші 5 рядків таблиці.

Для виклику методу об'єкта необхідно спочатку написати *ім'я об'єкта*, потім встановити *точку*, потім вже написати *назву методу*. Зверніть увагу, що в кінці обов'язково ставити дужки, тому що метод – це функція і в ній є аргументи, просто в такому випадку ми їх не передаємо, тому залишаємо поле порожнім

```
[ ]
training_data.head()
```

Крок 1.3. Подивимося на розміри завантаженої таблиці, у якій ми бачили лише перші 5 рядків.

Для цього викликаємо поле **shape** у нашої змінної *training_data*. Поле викликається як метод, але наприкінці дужки не ставляться, оскільки поля не передбачена передача аргументів.

[ ]
training_data.shape
(15129, 16)

Що означає перше та друге число?

Отже, таблиця містить 15129 рядків (об'єктів) та 16 стовпців (ознак), включаючи вихідну (цільову) ознаку. 15129 менше ніж 21613, тому що ми поки що завантажили тільки частину даних, яку будемо використовувати для навчання моделі.

Таблицю перевірили, тепер можна розпочинати обробку даних.

2. Обробити дані перед навчанням моделі

Крок 2.1. Перевіряємо дані на наявність пропусків і типів змінних

Почнемо з перевірки загальної інформації про дані. Щоб це зробити, потрібно звернутися викликати у змінної *training_data* метод **info()**.

Нагадаємо, що наприкінці необхідно поставити дужки.

[ ]
training_data.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15129 entries, 0 to 15128

```
Data columns (total 16 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
0 Цільова.Ціна 15129 non-null int64
1 Спальні 15129 non-null int64
2 Ванні 15129 non-null float64
3 Житлова площа 15129 non-null int64
4 Загальна площа 15129 non-null int64
5 Кількість поверхів 15129 non-null float64
6 Вода 15129 non-null int64
7 Переглянуто раніше 15129 non-null int64
8 Стан 15129 non-null int64
9 Оцінка рієлтора 15129 non-null int64
10 Площа без підвалу 15129 non-null int64
11 Площа підвалу 15129 non-null int64
12 Рік побудови 15129 non-null int64
13 Рік реновації 15129 non-null int64
14 Широта 15129 non-null float64
15 Довгота 15129 non-null float64
dtypes: float64(4), int64(12)
memory usage: 1.8 MB
```

Аналізуємо результати виконання команди:

• 15129 рядків (entries)

• 16 стовпців (Data columns)

У даних є всього два типи dtypes:

• int64 – ціле число (12 стовпців)

• float64 - дробове число (4 стовпці)

Цифри у кожному рядку позначають кількість заповнених (*non-null*) значень. Через те, що ці цифри в кожному рядку збігаються з числом рядків (15129), то в даних немає пропусків і можна рухатися далі.

Крок 2.2. Працюємо з цільовою змінною

Яка змінна цільова?

В такій ситуації за умовою завдання ми маємо прогнозувати вартість, тому цільова змінна – це ціна.

Двічі натисніть клітинку, щоб змінити її (або натисніть клавішу Enter)

[ ]

```
training_values = training_data['Цільова.Ціна']
```

[ ]

```
training_points = training_data.drop('Цільова.Ціна', axis=1)
```

Можна подивитися результати цих дій, викликавши метод **head()** та поле **shape**, якими ми користувалися раніше, але зараз потрібно викликати їх від нової змінної *training_points*.

[ ]

```
training_points.head()
```

[ ]

```
training_points.shape
```

```
(15129, 15)
```

Видно, що стовпця справді немає, а кількість рядків не змінилася. Дані в перших 5 рядках такі ж, як були раніше.

3. Навчити модель на навчальній вибірці

Крок 3.1. Вибираємо метод, який використовуватимемо

Найпростіше почати з найпростіших методів.

• Лінійна регресія *linear regression*

Для коректної роботи з методами побудови моделей у python потрібно завантажити спеціальну бібліотеку **sklearn**, програмну бібліотеку мовою python для машинного навчання та аналізу даних.

Ми імпортуємо модуль із цієї бібліотеки:

• *linear_model* – тут знаходяться всі лінійні моделі

[ ]

```
from sklearn import linear_model
```

Перш ніж розпочати ремонт, потрібно підготувати інструменти для роботи. Аналогічно в нашому випадку, перш ніж навчати моделі, потрібно створити прототипи.

Щоб створити модель лінійної регресії, пишемо ім'я модуля 'linear_model', потім точку, потім назву моделі.

[ ]

```
linear_regression_model = linear_model.LinearRegression() # створюємо модель
```

Крок 3.2. Навчити модель

Тепер, коли ми створили прототип моделі, можемо його навчити за допомогою навчальної вибірки.

Для цього викликаємо метод **fit()** у кожної моделі та передаємо йому на вхід два аргументи: таблицю вхідних ознак та стовпець значень цільової змінної - (training_points, training_values)

```
[ ]
linear_regression_model.fit(training_points, training_values)
LinearRegression()
```

• Ми отримали навчену модель;

• Тепер необхідно провалідувати модель на нові тестові дані.

4. Завантажити та обробити дані для тестування

Крок 4.1. Завантажимо та проаналізуємо тестові дані.

Оскільки дані у форматі xlsx (Excel), ми будемо використовувати спеціальну функцію бібліотеки pandas для завантаження таких даних **read_excel**.

У функції передаємо один атрибут: назву файлу, в якому міститься таблиця з даними.

```
[ ]
test_data = pd.read_excel('predict_house_price_test_data.xlsx')
```

Що важливо подивитися після того, як ми завантажили дані?

• перевірити, чи дані дійсно завантажилися;

• подивитися на дані, щоб переконатися, що вони правильні: колонки мають ті ж самі назви, що у таблиці тощо.

Щоб це зробити, потрібно викликати від змінної *test_data* метод **head()**, який виводить перші 5 рядків таблиці.

Для виклику методу об'єкта необхідно спочатку написати *ім'я об'єкта*, потім встановити *точку*, потім вже написати *назву*

методу. Зверніть увагу, що в кінці обов'язково ставити дужки, тому що метод – це функція і в ній є аргументи, просто в цьому випадку ми їх не передаємо, тому залишаємо поле порожнім

[ ]
test_data.head()

Подивимося на розміри завантаженої таблиці, оскільки ми бачили лише 5 рядків

Для цього викликаємо поле **shape** у нашої змінної *test_data*. Поле викликається як метод, але наприкінці дужки не ставляться (!), оскільки поля не передбачена передача аргументів.

[ ]
test_data.shape
(6484, 16)

Що означає перше та друге число?

Таблиця містить 6484 рядків (об'єктів) та 16 стовпців (ознак), включаючи вихідну (цільову) ознаку. Так само як у навчальних даних до навчання.

Таблицю перевірили, тепер можна приступати до обробки даних. Діємо аналогічно тому, як робили з даними для навчання

Перевіримо, чи є дані пропуски. Щоб це зробити, потрібно звернутися викликати у змінної *test_data* метод **info()**.

```
[ ]
test_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6484 entries, 0 to 6483
Data columns (total 16 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
0 Цільова.Ціна 6484 non-null int64
1 Спальні 6484 non-null int64
2 Ванні 6484 non-null float64
3 Житлова площа 6484 non-null int64
4 Загальна площа 6484 non-null int64
5 Кількість поверхів 6484 non-null float64
6 Вода 6484 non-null int64
7 Переглянуто раніше 6484 non-null int64
8 Стан 6484 non-null int64
9 Оцінка рієлтора 6484 non-null int64
10 Площа без підвалу 6484 non-null int64
11 Площа підвалу 6484 non-null int64
12 Рік побудови 6484 non-null int64
13 Рік реновації 6484 non-null int64
14 Широта 6484 non-null float64
15 Довгота 6484 non-null float64
dtypes: float64(4), int64(12)
memory usage: 810.6 KB
```

Цифри у кожному рядку позначають кількість заповнених (*non-null*) значень. Через те, що ці цифри в кожному рядку збігаються з числом рядків (6484), то в даних немає пропусків.

Крок 4.2. Відокремлюємо цільову змінну

Нам потрібно виділити в окрему змінну стовпець *test_values* з нашої таблиці, який відповідає визначеній вище цільовій змінній.

[ ]
```
test_values = test_data['Цільова.Ціна']
```

Відокремимо вхідні змінні від вихідних (цільових), щоб можна було побудувати модель передбачення цільової змінної по вхідних. Для цього необхідно у змінної *test_data* викликати спосіб **drop**(). Результат записуємо в нову змінну *test_points*. Після виконання запиту *test_points* міститиме вихідну таблицю без цільового стовпця.

[ ]
```
test_points = test_data.drop('Цільова.Ціна', axis=1)
```

І перевіряємо результат записаний у test_points

[ ]
```
test_points.head()
```

[ ]
```
test_points.shape
(6484, 15)
```

Очевидно, що кількість стовпців зменшилася на один. Дані в перших 5 рядках такі ж, як були раніше.

5. Провалідувати модель на тестовій вибірці

Отримаємо прогноз цільової змінної на тестових даних для моделі лінійної регресії.

Для цього викличемо метод **predict()**, як аргумент передамо *test_points*.

[ ]

```
test_predictions_linear = linear_regression_model.predict(test_points)
```

Якість регресійних моделей оцінимо двома способами:

1. Порівняємо візуально прогнози зі справжніми цінами (тестові з передбаченням)

2. Порівняємо метрики якості

Візуалізуємо прогноз лінійної моделі та реальні значення з тестової вибірки.

[ ]

```
%matplotlib inline
```

[ ]

```
plt.figure(figsize=(7, 7))
```

```
plt.scatter(test_values, test_predictions_linear) # Малюємо точки, що відповідають парам справжнє значення - прогноз
plt.plot([0, 6 * 10**6], [0, 6 * 10**6]) # малюємо пряму, на якій передбачення та справжні значення збігаються
plt.xlabel('Справжня ціна', fontsize=20)
plt.ylabel('Передбачена ціна', fontsize=20);
```

Для коректного підрахунку метрик якості моделі в python потрібно завантажити їх із бібліотеки **sklearn**.

Ми використовуємо дві метрики якості:

• *mean_absolute_error* – середня абсолютна помилка |yi−y^i|

• *mean_squared_error* – середня квадратична помилка (yi−y^i)2

```
[ ]
from sklearn.metrics import mean_absolute_error, mean_squared_error
```

Підрахуємо помилки для лінійної моделі.

Для цього викличемо методи **mean_absolute_error()** та **mean_squared_error()**. На вхід їм передається стовпець реальних значень *test_values* і стовпець значень, передбачених моделлю лінійної регресії *test_predictions_linear*.

```
[ ]
mean_absolute_error_linear_model = mean_absolute_error(test_values, test_predictions_linear)
mean_squared_error_linear_model = mean_squared_error(test_values, test_predictions_linear)
```

Тепер надрукуємо отримані помилки. Зазвичай дивляться на корінь із середньоквадратичної помилки, RMSE. Щоб

витягти корінь нам знадобиться бібліотека **Numpy**. За допомогою неї можна швидко проводити обчислення одразу над масивами чисел.

```
[ ]
import numpy as np
print("MAE: {0:7.2f}, RMSE: {1:7.2f} для моделі лінійної регресії".format(
        mean_absolute_error_linear_model,
        np.sqrt(mean_squared_error_linear_model)))
```

MAE: 126852.51, RMSE: 201883.24 для моделі лінійної регресії

Огляд результатів

У цьому ноутбуці ми навчилися

1. Завантажувати бібліотеки, необхідних для роботи.
2. Завантажити дані для навчання, представлені у форматі Excel таблиці.
3. Проводити попередню обробку даних перед побудовою та використанням моделі машинного навчання: дивитися на частини таблиці, розуміти, який розмір у вибірки даних, виділяти окремі стовпці таблиці у нові таблиці.
4. Навчати модель лінійної регресії на навчальній вибірці.
5. Валідувати модель на тестовій вибірці за допомогою крос-плота для моделі та реальних значень, стандартних помилок моделі.

4.5. Контрольні запитання

1. Назвіть приклади використання аналізу даних (data mining)?
2. За що відповідає бібліотека warnings?
3. Для чого використовується бібліотека pandas?
4. Як можна імпортувати дані з Excel?
5. Для чого використовується цільова функція?

Практична робота №5. Математична обробка даних експерименту. Поліноміальна регресія

5.1. Мета роботи

Навчитися обробляти експериментальні дані із застосуванням поліноміальної регресії.

5.2. Теоретичні відомості

У статистиці, поліноміальна регресія є однією з форм регресійного аналізу, в якому залежність між незалежною змінною x і залежною змінною y моделюється як поліном від x ступеню n. Поліноміальна регресія відповідає нелінійній залежності між значенням x та відповідним умовним математичним сподіванням y, що позначається E(y |x). Хоча поліноміальна регресія налаштовує нелінійній моделі даних, з боку теорії оцінювання ця задача є лінійною, в тому сенсі, що функція регресії E(y | x) є лінійною за невідомих параметрів які оцінюються за даними. З цього приводу поліноміальна регресія вважається приватним випадком множинної лінійної регресії.

Пояснювальні (незалежні) змінні, що є результатом поліноміального розширення «базових» змінних, відомі як терміни вищого ступеня. Такі змінні також використовуються в налаштуваннях класифікації.

5.3. Порядок виконання роботи

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

Намалюємо точки графіка y=x*x, додавши шум до y.

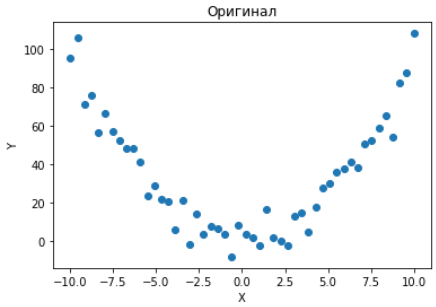
```
m = 50
X = np.linspace(-10,10,m).reshape(-1, 1)
X
```

```
array([[-10. ], [ -9.59183673], [ -9.18367347], [ -8.7755102 ], [ -
8.36734694], [ -7.95918367], [ -7.55102041], [ -7.14285714], [ -6.73469388],
[ -6.32653061], [ -5.91836735], [ -5.51020408], [ -5.10204082], [ -
4.69387755], [ -4.28571429], [ -3.87755102], [ -3.46938776], [ -3.06122449],
[ -2.65306122], [ -2.24489796], [ -1.83673469], [ -1.42857143], [ -
1.02040816], [ -0.6122449 ], [ -0.20408163], [ 0.20408163], [ 0.6122449 ], [
1.02040816], [ 1.42857143], [ 1.83673469], [ 2.24489796], [ 2.65306122], [
3.06122449], [ 3.46938776], [ 3.87755102], [ 4.28571429], [ 4.69387755], [
5.10204082], [ 5.51020408], [ 5.91836735], [ 6.32653061], [ 6.73469388], [
7.14285714], [ 7.55102041], [ 7.95918367], [ 8.36734694], [ 8.7755102 ], [
9.18367347], [ 9.59183673], [ 10. ]])
```

```
y =  X**2 + 8*np.random.randn(m, 1)
y
```

```
array([[ 95.15365417], [105.77135332], [ 71.2973269 ], [ 75.90642391], [
56.64462719], [ 66.34374264], [ 57.16514472], [ 52.37303094], [
48.38836603], [ 48.30941909], [ 41.03494834], [ 23.63630025], [
28.40442032], [ 21.83699622], [ 20.69449015], [ 6.0139107 ], [ 21.07465498],
[ -1.93334671], [ 13.73161624], [ 3.10339232], [ 7.32258898], [ 6.34800809],
[ 3.4532803 ], [ -8.36397773], [ 8.16771493], [ 3.42125936], [ 1.88847226],
[ -2.31217093], [ 16.29350832], [ 1.53000506], [ -0.10863659], [ -
2.64440228], [ 12.5805591 ], [ 14.33751356], [ 4.66457266], [ 17.63550729],
[ 27.26281995], [ 29.92950095], [ 35.74475812], [ 37.40550153], [
41.01311927], [ 38.36075736], [ 50.21773213], [ 52.19773789], [ 58.9812729
], [ 64.93267701], [ 53.91964266], [ 82.29047699], [ 87.76632297],
[108.0528467 ]])
```

```
plt.scatter(X, y);
plt.title('Оригинал');
plt.xlabel("X")
plt.ylabel("Y")
plt.show()
```



Побудуємо за цими точками лінійну регресію.

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

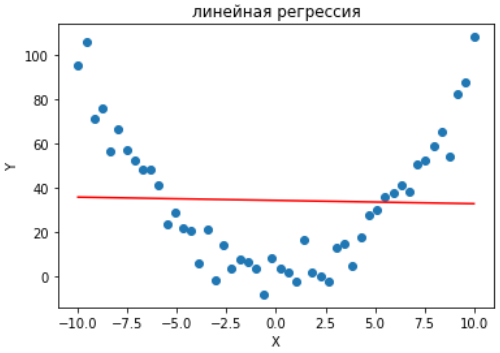
```
lr = LinearRegression()
lr.fit(X, y);
print('R² Value: ')
lr.score(X, y)
```

```
R² Value:
0.0008264947006746404
```

```
lr.intercept_, lr.coef_
```

```
(array([34.14478881]), array([[-0.14873214]]))
```

```
y_pred=lr.predict(X)
plt.scatter(X, y);
plt.title('лінійная регрессія');
plt.plot(X, y_pred, color='red');
plt.xlabel("X")
plt.ylabel("Y")
plt.show()
```



Додамо регуляризацію.

```
from sklearn.linear_model import Lasso, Ridge
rg = Ridge(fit_intercept=True, alpha=1,
random_state=0, normalize=True)
rg.fit(X, y)
print('R² Value:',rg.score(X, y))
```

```
R² Value: 0.0006198710255059803
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_base.py:145: FutureWarning: 'normalize' was deprecated in version 1.0 and will be removed in 1.2.
If you wish to scale the data, use Pipeline with a StandardScaler in a preprocessing stage. To reproduce the previous behavior:

from sklearn.pipeline import make_pipeline

model = make_pipeline(StandardScaler(with_mean=False), Ridge())

If you wish to pass a sample_weight parameter, you need to pass it as a fit parameter to each step of the pipeline as follows:

kwargs = {s[0] + '__sample_weight': sample_weight for s in model.steps}
model.fit(X, y, **kwargs)

Set parameter alpha to: original_alpha * n_samples.
  FutureWarning,
```

Додамо поліноміальні ознаки 2-го ступеня

```
from sklearn.preprocessing import PolynomialFeatures
pf = PolynomialFeatures(degree=2)
X_polynomial = pf.fit_transform(X)
X_polynomial
```

```
array([[ 1.00000000e+00, -1.00000000e+01, 1.00000000e+02], [ 1.00000000e+00,
-9.59183673e+00, 9.20033319e+01], [ 1.00000000e+00, -9.18367347e+00,
8.43398584e+01], [ 1.00000000e+00, -8.77551020e+00, 7.70095793e+01], [
1.00000000e+00, -8.36734694e+00, 7.00124948e+01], [ 1.00000000e+00, -
7.95918367e+00, 6.33486047e+01], [ 1.00000000e+00, -7.55102041e+00,
5.70179092e+01], [ 1.00000000e+00, -7.14285714e+00, 5.10204082e+01], [
1.00000000e+00, -6.73469388e+00, 4.53561016e+01], [ 1.00000000e+00, -
6.32653061e+00, 4.00249896e+01], [ 1.00000000e+00, -5.91836735e+00,
3.50270721e+01], [ 1.00000000e+00, -5.51020408e+00, 3.03623490e+01], [
1.00000000e+00, -5.10204082e+00, 2.60308205e+01], [ 1.00000000e+00, -
4.69387755e+00, 2.20324865e+01], [ 1.00000000e+00, -4.28571429e+00,
1.83673469e+01], [ 1.00000000e+00, -3.87755102e+00, 1.50354019e+01], [
1.00000000e+00, -3.46938776e+00, 1.20366514e+01], [ 1.00000000e+00, -
3.06122449e+00, 9.37109538e+00], [ 1.00000000e+00, -2.65306122e+00,
7.03873386e+00], [ 1.00000000e+00, -2.24489796e+00, 5.03956685e+00], [
1.00000000e+00, -1.83673469e+00, 3.37359434e+00], [ 1.00000000e+00, -
1.42857143e+00, 2.04081633e+00], [ 1.00000000e+00, -1.02040816e+00,
1.04123282e+00], [ 1.00000000e+00, -6.12244898e-01, 3.74843815e-01], [
1.00000000e+00, -2.04081633e-01, 4.16493128e-02], [ 1.00000000e+00,
2.04081633e-01, 4.16493128e-02], [ 1.00000000e+00, 6.12244898e-01,
3.74843815e-01], [ 1.00000000e+00, 1.02040816e+00, 1.04123282e+00], [
1.00000000e+00, 1.42857143e+00, 2.04081633e+00], [ 1.00000000e+00,
1.83673469e+00, 3.37359434e+00], [ 1.00000000e+00, 2.24489796e+00,
5.03956685e+00], [ 1.00000000e+00, 2.65306122e+00, 7.03873386e+00], [
1.00000000e+00, 3.06122449e+00, 9.37109538e+00], [ 1.00000000e+00,
3.46938776e+00, 1.20366514e+01], [ 1.00000000e+00, 3.87755102e+00,
1.50354019e+01], [ 1.00000000e+00, 4.28571429e+00, 1.83673469e+01], [
1.00000000e+00, 4.69387755e+00, 2.20324865e+01], [ 1.00000000e+00,
5.10204082e+00, 2.60308205e+01], [ 1.00000000e+00, 5.51020408e+00,
3.03623490e+01], [ 1.00000000e+00, 5.91836735e+00, 3.50270721e+01], [
1.00000000e+00, 6.32653061e+00, 4.00249896e+01], [ 1.00000000e+00,
6.73469388e+00, 4.53561016e+01], [ 1.00000000e+00, 7.14285714e+00,
5.10204082e+01], [ 1.00000000e+00, 7.55102041e+00, 5.70179092e+01], [
1.00000000e+00, 7.95918367e+00, 6.33486047e+01], [ 1.00000000e+00,
8.36734694e+00, 7.00124948e+01], [ 1.00000000e+00, 8.77551020e+00,
7.70095793e+01], [ 1.00000000e+00, 9.18367347e+00, 8.43398584e+01], [
1.00000000e+00, 9.59183673e+00, 9.20033319e+01], [ 1.00000000e+00,
1.00000000e+01, 1.00000000e+02]])
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test =
train_test_split(X_polynomial, y, train_size=0.7)

lr.fit(X_train, y_train) #навчаємо
lr.score(X_test,y_test) #перевіряємо точність
```

```
0.9368926772176968
```

```
y_pred=lr.predict(X_polynomial)

plt.scatter(X, y);
plt.title('Поліноміальна регрессія ступені 2');
plt.plot(X, y_pred, color='red');
```

```
plt.xlabel("X")
plt.ylabel("Y")
plt.show()

lr.coef_
```

```
array([[ 0. , -0.28591284, 0.91847656]])
```

```
print('R² Value: \n', lr.score(X_polynomial, y))
```

```
R² Value:
 0.945751903840927
```

Додамо поліноміальні ознаки 15-го ступеня

```
pf1 = PolynomialFeatures(degree=15)
X_polynomial1 = pf1.fit_transform(X)
X_train, X_test, y_train, y_test =
train_test_split(X_polynomial1, y, train_size=0.7)


lr.fit(X_train, y_train)
lr.score(X_test,y_test)
```

```
-0.07767345359166056
```

```
lr.score(X_train, y_train)
```

```
0.978614599104848
```

```
lr.coef_
```

```
array([[ 0.00000000e+00, -2.73055146e+00, -1.14458588e+00, 2.27723740e-01,
1.96070070e-01, 2.92331644e-02, -7.37866100e-03, -3.39814291e-03,
1.57212541e-04, 1.25900502e-04, -2.13869411e-06, -2.17544363e-06,
1.70683516e-08, 1.79390700e-08, -5.83786353e-11, -5.69468916e-11]])
```

```
y_pred1=lr.predict(X_polynomial1)
plt.scatter(X, y);
plt.title('Поліноміальна регрессія ступені 15');
plt.plot(X, y_pred1, color='red');
plt.xlabel("X")
```


```
plt.ylabel("Y")
plt.show()
```

Додамо регуляризацію.

```
ls = Lasso(fit_intercept=True, alpha=0.1,
random_state=0, normalize=True)
ls.fit(X_train, y_train)
ls.score(X_test,y_test)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_base.py:145: FutureWarning: 'normalize' was deprecated in version 1.0 and will be removed in 1.2.
If you wish to scale the data, use Pipeline with a StandardScaler in a preprocessing stage. To reproduce the previous behavior:

from sklearn.pipeline import make_pipeline

model = make_pipeline(StandardScaler(with_mean=False), Lasso())

If you wish to pass a sample_weight parameter, you need to pass it as a fit parameter to each step of the pipeline as follows:

kwargs = {s[0] + '__sample_weight': sample_weight for s in model.steps}
model.fit(X, y, **kwargs)

Set parameter alpha to: original_alpha * np.sqrt(n_samples).
  FutureWarning,
0.9246081700323339
```

```
y_pred2=ls.predict(X_polynomial1)
plt.scatter(X, y);
plt.title('Поліноміальна регрессія с
регуляризацією');
plt.plot(X, y_pred2, color='red');
plt.xlabel("X")
```

```
plt.ylabel("Y")
plt.show()

rg.coef_
```

Намалюємо діаграму розсіювання (співвідношення реальних та передбачених значень).

```
plt.scatter(y, y_pred);
plt.plot(y, y, color='red');  #график x=y для
сравнения
plt.title('Діаграмма розсіювання для останньої
моделі');
plt.xlabel("y")
plt.ylabel("y_pred")
plt.show()
```

5.4. Контрольні запитання

1. Що називають поліноміальною регресією?
2. Яка відмінність між лінійною та поліноміальною регресією?
3. Що показує діаграма розсіювання?
4. Як обчислити похибку в задачах регресії?

Практична робота № 6. Математична обробка даних експерименту. Парна регресія.

6.1. Мета роботи

Навчитися обробляти експериментальні дані із застосуванням лінеаризації деяких видів двопараметричних зв'язків.

6.2. Теоретичні відомості

Лінійні методи припускають, що між ознаками об'єкта та цільовою змінною існує лінійна залежність, тобто:

$$y = w_1 x_1 + w_2 x_2 + \ldots + w_k x_k + b,$$

де y – цільова змінна (що хочемо передбачити), x_i – ознака об'єкта x, w_i – вага i-ої ознаки, b – $bias$ (зміщення, вільний член).

Часто припускають, що об'єкт x містить у собі фіктивну ознаку, яка завжди дорівнює 1, тоді $bias$ це вага цієї ознаки. У цьому випадку формула набуває простого вигляду:

$$y = < w, x >.$$

У матричній формі, у випадку, коли ми маємо n об'єктів, формулу можна переписати наступним чином:

$$Y = Xw,$$

Y – вектор розміру n, X – матриця, що містить об'єкти-ознаки розміру n x k , w – вектор ваги розміру k.

Рішення за методом найменших квадратів дає

$$w = (X^T X)^{-1} X^T Y$$

6.3. Порядок виконання роботи

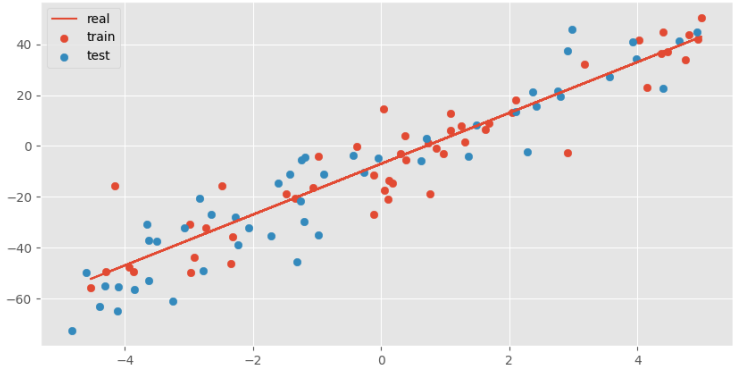
Згенеруємо штучні дані – 100 точок, розкиданих навколо прямої $y = 10x - 7$. З них 50 помістимо у навчальну вибірку та 50 у тестову.

```
X_train =  10 * np.random.rand(50, 1)-5

y_train = 10 * X_train - 7+np.random.randn(50,1)*10
```

```
y_real=10 * X_train - 7

X_test = 10 * np.random.rand(50, 1)-5

y_test = 10 * X_test - 7+np.random.randn(50,1)*10
```

Візуалізуємо дані, використовуючи бібліотеку *matplotlib*.

```
plt.figure(figsize=(10, 5))

plt.plot(X_train, y_real, label='real') #лінійний
графік

plt.scatter(X_train, y_train, label='train') #точковий
графік

plt.scatter(X_test, y_test, label='test') #точковий
графік

plt.legend(loc='best')

plt.show()
```



Попрацюємо з найпростішим одновимірним випадком регресії. Навчимо нашу модель на тренувальній вибірці та передбачимо значення на тестовій виборці. Вагу моделі збережемо в змінних w, b.

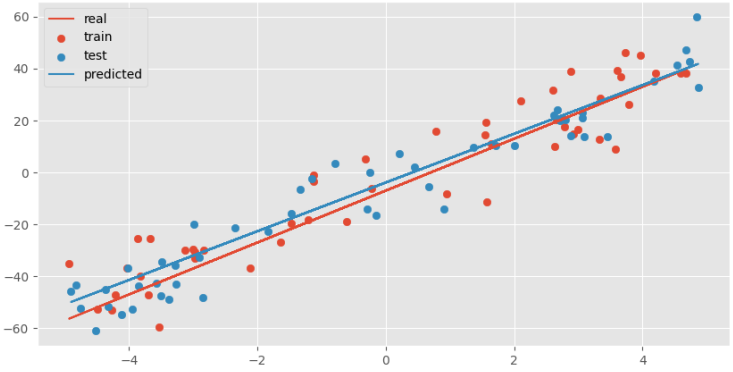
```
from sklearn.linear_model import LinearRegression
```

```
model = LinearRegression()


model.fit(X_train, y_train) #навчання моделі
w=model.coef_
b=model.intercept_


print(w, b)
```

[[9.66455856]] [-6.75231145]

Візуалізуємо передбачення.

```
y_pred=model.predict(X_test)


plt.figure(figsize=(10, 5))
plt.plot(X_train, y_real, label='real')
plt.scatter(X_train, y_train, label='train')
plt.scatter(X_test, y_test, label='test')
plt.plot(X_test,y_pred, label='predicted')
plt.legend()
plt.show()
```



Оцінка результатів

Обчислимо середньоквадратичну помилку на трейні та на тесті. Також обчислимо середню абсолютну помилку та коефіцієнт детермінації.

```
from sklearn.metrics import mean_squared_error



y_train_predicted = model.predict(X_train)

y_test_predicted = model.predict(X_test)



print('Train MSE: ', mean_squared_error(y_train,
y_train_predicted))

print('Test MSE: ', mean_squared_error(y_test,
y_test_predicted))
```

Train MSE: 57.99314109827129
Test MSE: 115.68319173283655

```
from sklearn.metrics import mean_absolute_error



```

```
print('Train MAE: ', mean_absolute_error(y_train,
y_train_predicted))

print('Test MAE: ', mean_absolute_error(y_test,
y_test_predicted))
```

Train MAE: 6.6854874075692
Test MAE: 8.534245983546114

```
print('Train R^2: ',model.score(X_train, y_train) )

print('Test R^2: ', model.score(X_test, y_test))
```

Train R^2: 0.895895154599857
Test R^2: 0.8728535134296519

Частина 2 - багатовимірна регресія

Датасет Boston

Працюватимемо з датасетом Boston. Цей датасет визначає середні ціни на нерухомість у районах Бостона у $1000. Приклади ознак міста: кількість злочинів душу населення, відсоток старих будинків у районі, кількість учнів однією вчителя тощо. Дані вже оцифровані там, де спочатку ознаки були якісними.

Завантажимо датасет, виведемо інформацію.

```
from sklearn.datasets import load_boston

from sklearn.metrics import accuracy_score, roc_curve,
roc_auc_score, f1_score

house_data = load_boston()



X = pd.DataFrame(house_data['data'],
columns=house_data['feature_names'])

y = house_data['target']
```

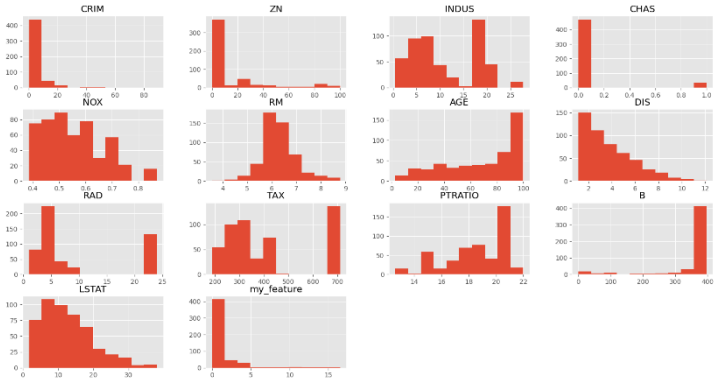
```
X['my_feature'] = X['CRIM'] * X['ZN']

print(X)
```

	CRIM	ZN	INDUS	CHAS	...	PTRATIO	B	LSTAT	my_feature
0	0.00632	18.0	2.31	0.0	...	15.3	396.90	4.98	0.11376
1	0.02731	0.0	7.07	0.0	...	17.8	396.90	9.14	0.00000
2	0.02729	0.0	7.07	0.0	...	17.8	392.83	4.03	0.00000
3	0.03237	0.0	2.18	0.0	...	18.7	394.63	2.94	0.00000
4	0.06905	0.0	2.18	0.0	...	18.7	396.90	5.33	0.00000
..	...	...	...	...	...	...	...	...	...
501	0.06263	0.0	11.93	0.0	...	21.0	391.99	9.67	0.00000
502	0.04527	0.0	11.93	0.0	...	21.0	396.90	9.08	0.00000
503	0.06076	0.0	11.93	0.0	...	21.0	396.90	5.64	0.00000
504	0.10959	0.0	11.93	0.0	...	21.0	393.45	6.48	0.00000
505	0.04741	0.0	11.93	0.0	...	21.0	396.90	7.88	0.00000

[506 rows x 14 columns]

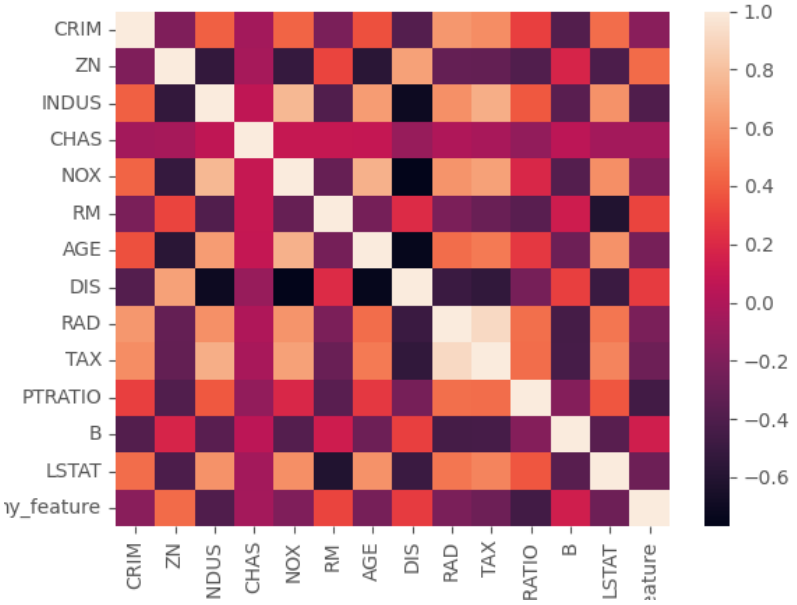
Візуалізуємо розподіл ознак у датасеті.

```
xhist = X.hist(X.columns, figsize=(10, 10))

plt.show()
```



Подивимося на те, як скорельовані дані.

```
import seaborn as sns


sns.heatmap(X.corr())

plt.show()
```



Зробимо передбачення: які чинники виявляться найбільш значущими та в який бік?

Навчання на реальних даних

Розіб'ємо вибірку на train і test у співвідношенні 70:30.

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,
y, train_size=0.7)

print(y_train.shape, y_test.shape)
```

(354,) (152,)

Виконаємо нормування даних.

```
from sklearn.preprocessing import StandardScaler



scaler = StandardScaler()



X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)
```

Навчимо лінійну регресію та підрахуємо її якість на тесті.

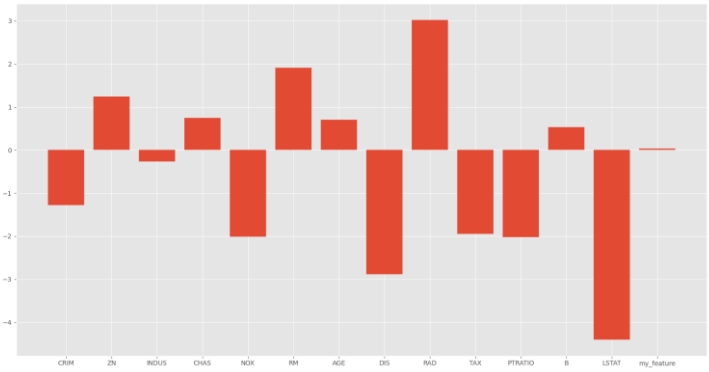
```
model = LinearRegression()



model.fit(X_train, y_train)



y_train_prediction = model.predict(X_train)

y_test_prediction = model.predict(X_test)

########################################

print('Train MSE: ', mean_squared_error(y_train,
y_train_prediction))

print('Test MSE: ', mean_squared_error(y_test,
y_test_prediction))
```

```
print('Train MAE: ', mean_absolute_error(y_train,
y_train_prediction))

print('Test MAE: ', mean_absolute_error(y_test,
y_test_prediction))
```

Train MSE: 20.78485127531905
Test MSE: 24.74617687682064
Train MAE: 3.128932104525232
Test MAE: 3.684746083278115

Візуалізуємо ваги, що вийшли

```
plt.figure(figsize=(20, 8))

plt.bar(X.columns, model.coef_)

plt.show()
```



Ridge & Lasso

Спробуємо виправити проблему перенавчання для лінійної регресії. Помічено, що лінійна регресія перенавчається, коли вектор ваги має дуже великі значення. Постараємося уникнути цього, додавши до функції втрат додатковий доданок.

Ridge regression:

$$L(x,y) = \overline{(< w,x > -y_{true})^2} + \lambda||w||^2 \to \min_w.$$

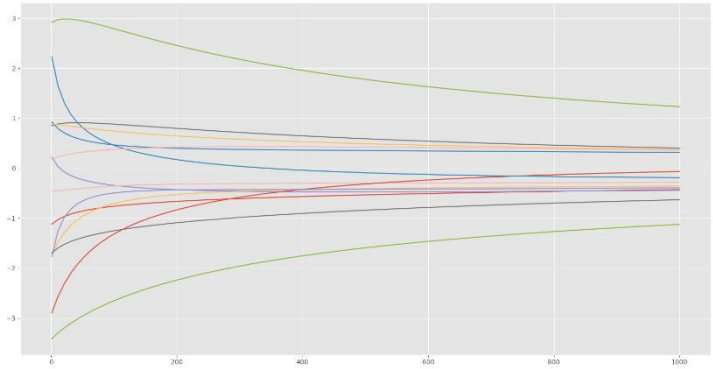
Lasso regression:

$$L(x,y) = \overline{(< w,x > -y_{true})^2} + \lambda||w||^2 \to \min_w.$$

```
from sklearn.linear_model import Ridge, Lasso
```

Ridge (L2-регуляризация) сильно штрафує за занадто великі ваги і не дуже малі. У разі збільшення коефіцієнта перед регуляризатором ваги змінюються плавно.

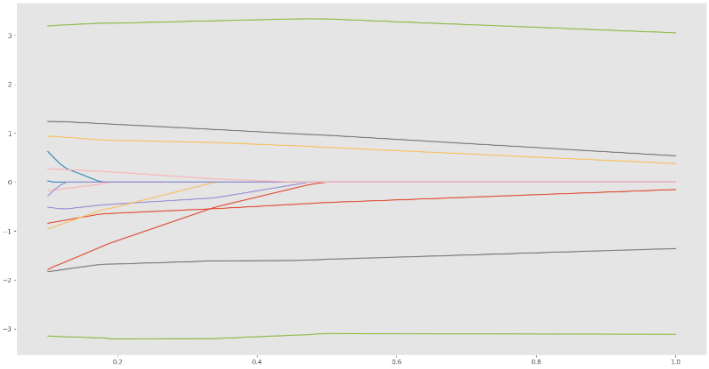
```
alphas = np.linspace(1, 1000, 100)



weights = np.empty((len(X.columns), 0))

for alpha in alphas:

    ridge_regressor = Ridge(alpha)

    ridge_regressor.fit(X_train, y_train)

    weights = np.hstack((weights,
ridge_regressor.coef_.reshape(-1, 1)))

plt.figure(figsize=(15,8))

plt.plot(alphas, weights.T)

plt.show()
```



Lasso однаково сильно штрафує малі та великі ваги, тому при досить великому коефіцієнті регуляризації багато ознак стають рівними нулю, при цьому залишаються лише найбільш інформативні. Цей факт можна використовувати для вирішення задачі відбору ознак.

```
alphas = np.linspace(0.1, 1, 100)



plt.figure(figsize=(10, 5))

weights = np.empty((len(X.columns), 0))

for alpha in alphas:

    lasso_regressor = Lasso(alpha)

    lasso_regressor.fit(X_train, y_train)

    weights = np.hstack((weights,
lasso_regressor.coef_.reshape(-1, 1)))

plt.figure(figsize=(15,8))

plt.plot(alphas, weights.T)

plt.grid()
```

```
plt.show()
```



6.4. Контрольні запитання

1. Про які типи регресії йдеться у лекції?
2. Опишіть процес створення передбачень моделі.
3. Напишіть, які типи графіків найбільше підходять для відображення певних типів даних аналізу.
4. Що використовують для оцінки результатів регресії/передбачень?
5. Вкажіть різницю між Ridge та Lasso регресіями

Список використаних джерел:

1. Нейронні мережі: теорія та практика : навч. посіб. / С. О. Субботін. Житомир : Вид. О. О. Євенок, 2020. 184 с.
2. Машинне навчання : навчальний посібник. Львів : Видавництво «Новий Світ - 2000», 2021. 315 с.
3. Artificial Intelligence: A Modern Approach / Stuart Russell and Peter Norvig. Upper Saddle River, New Jersey 07458, 2010. https://people.engr.tamu.edu/guni/csce421/files/AI_Russell_Norvig.pdf
4. MIT Deep Learning Book in PDF format (complete and parts) by Ian Goodfellow, Yoshua Bengio and Aaron Courville. https://github.com/janishar/mit-deep-learning-book-pdf
5. Neural Networks and Deep Learning by Michael Nielsen. https://github.com/antonvladyka/neuralnetworksanddeeplearning.com.pdf
6. Reinforcement Learning: An Introduction" by Richard S. Sutton and Andrew G. Barto. https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf
7. Computer Vision: Algorithms and Applications by Richard Szeliski. https://szeliski.org/Book/