Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики факультету інформатики

Мова програмування Python та її математичні бібліотеки

Текстова частина до курсової роботи
за спеціальністю “Прикладна математика”

Керівник курсової роботи

Кандидат фіз.-мат. наук доц.,
доц. *Глибовець А.М.*
(прізвище та ініціали)

_____
(підпис)
“____” _____ 2017 р.

Виконав студент ФІН-3
Кузів П.М.

“____” _____ 2017 р.

Київ 2017

ЗМІСТ

Тема: <u>Мова програмування Python та її математичні бібліотеки</u>

Календарний план виконання роботи:

№ п/п	Назва етапу курсової роботи	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу	17.10.2016	
2.	Огляд технічної літератури за темою роботи.	07.11.2016	
3.	Виконати аналіз існуючих досліджень за темою.	28.11.2016	
4.	Дослідити синтаксис та базові структури мови Python.	12.12.2016	
5.	Написання першого розділу.	16.01.2017	
6.	Аналіз математичних бібліотек Python.	20.02.2017	
7.	Написання другого розділу.	20.03.2017	
8.	Дослідити моель фільтра спаму на основі теореми Байєса.	10.04.2017	
9.	Реалізація модель простого фільтра спаму на основі Байєсвіської класифікації.	17.04.2017	
10.	Аналіз виконаної роботи з керівником, написання доповіді.	24.04.2017	
11.	Корегування роботи.	08.05.2017	
12.	Остаточне оформлення пояснювальної робоити на слайдів.	12.05.2017	
13.	Захист курсової роботи.	25.05.2017	

Студент Кузів П.М.

Керівник Глибовець А.М.

“_____” _____

Анотація

У даній курсовій роботі розглянуто мову програмування Python 3, її синтаксис, структури даних, наведено приклади їх створення та використання. Окрім того, досліджено можливості математичних бібліотек NumPy, Pandas та MatPlotLib, описано їхні особливості та наведено приклади використання. Також реалізовано алгоритм фільтру спаму на основі Баєсівської класифікації, використовуючи мову програмування Python 3.

ВСТУП

Унаслідок динамічного розвитку сфери дослідження інформації науковці потребували інструменту для роботи над великими обсягами даних, який б зміг симулювати та обробляти дані, швидко візуалізувати результати для звітів та публікацій. Стрімкий розвиток електронно обчислювальної техніки та програмування мав задовольнити ці потреби. 1964 Карлом Енгельманом було створено систему обчислювальної алгебри - Mathlab. Вона стала першою спробою вирішити проблему, що виникла. Під плином часу ринок заповнився багатим різноманіттям продуктів, що стали конкурентами Mathlab. Помітне місце серед цих інструментів займає інтерпритовна мова програмування Python.

Лаконічний синтаксис, потужність виражальних можливостей, багатий набір додаткових модуль - все це зробило Python популярною мовою програмування широкого застосування. Багато бібліотек для дослідження даних були створені, надихаючись рішеннями Mathlab. Попри те, Python зміг перевершити свого опонента, оскільки пропонує багатший набір інструментів, зберігаючи простий синтаксис та читальність коду. Незважаючи на появу великих конкурентів, таких як мова програмування R чи Julia, Python впевнено зберігає свої позиції. Саме з цих міркувань темою даної роботи було обрано мова програмування Python та її математичні бібліотеки.

Курсова робота містить три розділи.

У першому розділі розглянуто сам Python, описано його базові можливості та синтаксис, структури даних та їх особливості, а також реалізацію об’єктно орієнованої парадигми.

У другому розділі детально розібрано основні математичні бібліотеки, а саме: NumPy, Pandas, MatPlotLib. Описано основні структури даних NumPy та Pandas, наведено приклади використання та особливості роботи. Продемонстровано можливості візуалізації даних бібліотеки MatPlotLib, наведено приклади створення відповідних графіків.

У третьому розділі досліджено математичне представлення алгоритму визначення спаму на основі теореми Байєса, а також продемонстровано реалізацію цього алгоритму, використовуючи мову програмування Python.

Постановка задачі.

1. Проаналізувати мову програмування Python, її синтаксис, можливості, особливості та призначення.
2. Дослідити математичні бібліотеки NumPy, Pandas, MatPlotLib, описати їх особливості та навести приклади використання.
3. Продемонструвати можливості Python на практиці, реалізувавши модель спам фільтр на основі теореми Байєса.

Розділ 1. ОГЛЯД МОВИ ПРОГРАМУВАННЯ PYTHON

1.1. Базовий синтаксис та філософія мови

Python є інтерпритовною об’єктно орієнтовною мовою програмування високого рівня з динамічною типізацією. Розроблена 1990 року нідерландським програмістом Гвідо ван Россумом. Інтерпритовність, структури даних високого рівня та динамічне зв’язування - все це зробило Python надзвичайно популярною для швидкої розробки програмного забезпечення та наукових досліджень. На сьогодні Python має три версії: Python1, Python2, Python3. Активно використовують другу та третю версії. На жаль. Python2 та Python3 не є сумісними. У цій курсовій буде розглянуто Python версії 3.6.1. Дана мова програмування має свою філософію, з якою можна ознайомитися набравши в інтерпритаторі “`import this`”. Загалом, найбільш популярним та важливим є твердження: “Існує один і лише один очевидний спосіб зробити щось”.

Багато мов програмування використовують фігурні дужки, щоб виділити блоки коду. Python для цього використовує відступи.

```
for  i in [1, 2, 3]
      print(i)
```

Лістинг 1.1.1

Це робить код програм читальним, але й змушує програмістів буде дуже обережним у форматуванні. Також інтерпритатор ігнорує порожні символи в круглих та квадратних дужках, що є зручним для довгих обчислень.

```
long_computation = (1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9
                    10  + 11 + 12 + 13)
```

Лістинг 1.1.2

Для того, щоб позначити, що речення продовжується в наступному рядку використовують символ ‘\’:

```
five_plus_five_plus_five = 5 + \
            5 + 5
```

Лістинг 1.1.3

Деякі опції Python не є доступними за замовчуванням. Вони збережені в окремих пакетах - модулях. Модулі можуть входити в стандартний дистрибутив мови, або ж поставлятися як додаткові зовнішні бібліотеки. Функціонал Python легко розширити додатковим модулем, що може бути створений використовуючи як низькорівневу мову програмування C, так і сам Python. Для того, щоб імпортувати певну бібліотеку використовують синтаксис:

```
import re as regex
```

Лістинг 1.1.4

Даний код імпортує стандартний модуль ‘re’ для роботи з регулярними виразами, та надає йому псевдонім ‘regex’.

Python є мультипарадигменою мовою та підтримує: об’єктно орієнтовану, процедурну та функціональну парадигми. Для того, щоб формувати логічні блоки коду дана мова програмування підтримує функції. Функція може приймати нуль або більше аргументів і повертає відповідний результат. Для визначення функцій використовують наступний синтаксис:

```
def my_function(a, b, c):
        return a + b + c
```

Лістинг 1.1.5

У Python функції фактично є об’єктами або ж функторами. Це дозволяє присвоювати функції змінним. Лямбда вирази або анонімні функції є елементом функціонального програмування, який підтримує Python. Приклад використання продимонстровано в Лістингу 1.1.6.

```
def apply_to_one(func):
      return func(1)
apply_to_one(lambda x: x * 5)
```

Лістинг 1.1.6

Параметри функції можуть мати стандартні значенні:

```
def print_message(message=”hello python”):
      print (message)
```

Лістинг 1.1.7

Також при передачі параметрів функції зберігати порядок аргументів не обов’язково, адже їх можна визначити за іменем:

```
def multiplication(a = 1, b = 1):
      return a * b
multiplication(b = 3, a = 2)
```

Лістинг 1.1.8

Коли у програмі виникає помилка, Python створює виняткову ситуацію. Якщо виняткову ситуацію жодним чином не ловлять, то це викличе аварійну зупинку виконання. Для обробки помилок використовують “try - except” блоки:

```
try:
      print 0 / 0
except ZeroDivisionError:
      print "cannot divide by zero"
```

Лістинг 1.1.9

Існують мови програмування, де створення виняткових ситуацій є поганою практикою (наприклад Ruby), у Python це навпаки допомагає зробити код зрозумілішим. Для побудови логічних ланцюжків виконання програми з відповідними умовами у Python використовують оператори `if, elif, else`. Як і в інших мовах програмування логічні умови можуть мати будь-яку глибину вкладень.

```
if x < 0:
	if (x > -5):
		print (“x less than 0 but more than -5”)
	print(“x less than 0”)
elif x > 0:
	print(“x more than 0”)
else:
	print(“x equals 0”)
```

Лістинг 1.1.10

Досить часто в програмах доводиться використовувати цикли. Python підтримує `while` та `for in` цикли. Цикл `while` схожий на той, який є в мові програмування C:

```
while x < 10:
	print (x)
	x += 1
```

Лістинг 1.1.11

Цикл `for` дещо відрізняється від того, який ми звикли бачити в мові C. Замість того, щоб завжди ітерувати через арифметичну прогресію із заданим кроком, Python ітерує через всі елементи заданого ряду (списку чи стрічки). Наприклад:

```
languages = [“C++”, “Python”, “JavaScript”]
for language in languages:
	print(language)
```

Лістинг 1.1.12

Python дозволяє повноцінно використовувати парадигму об’єктно орієнтованого програмування, створюючи ієрархії класів, використовуючи поліморфізм та інші властивості цієї парадигми. Щоб визначити клас використовують наступний синтаксис:

```
class ClassName(ParentClass1, [ParentClass2, … ParentClassn]):
```

Лістинг 1.1.13

ParentClass - клас від якого роблять наслідування, вказувати не обов’язково. Для визначення конструктора та деструктура в Python використовують

спеціальні методи `__init__` та `__del__`. Ключове слово `self` вказує на даний об’єкт. Кожен метод повинен містити `self` у своїй семантиці як перший аргумент, якщо необхідно посилання на даний об’єкт у середині метода. При виклику методу явно параметр `self` передавати не потрібно. Окрім цього, дана мова програмування надає можливість перевизначати оператори використовуючи методи: `__add__(self, other) (+), __iadd(self, other)__ (+=), __mul__(self, other) (*), __lt__(self, other) (<), __gt(self, other)__ (>), __le__(self, other) (<=), __ge__(self, other) (>=)` та інші. Також Python дозволяє перевизначати `getters & setters`, які він викликатиме неявно при доступі до атрибутів об’єктів, `__hash__(self)` метод для перевизначення хеш-значення об’єкта, `__string__(self)` метод для перевизначення подання об’єкта в стрічковому типі, `__bool__(self)` метод - для тестування об’єкта на істинність чи хибу (повинен повертати `True` чи `False`).

1.2. Основні типи, структури даних та операції над ними.

Основними стандартними типами в Pyhon є: числові типи, ряди, мапи та виняткові ситуації. Існує три різні числові типи: цілі, дійсні та комплексні. Цілі типи мають необмежену точність. Дійсні числа `(float numbers)` реалізовані як тип `double` в мові С. Комплексні числа мають дійсну та уявну частини, які представляють у вигляді дійсних чисел. Для оголошення комплексного числа використовують конструктор `complex(re, im)`. Числові типи підтримують усі арифметичні операції над числами: додавання, множення, віднімання, ділення, знаходження остачі, піднесення в степінь, заперечення, модуль. До третьої версії Python за замовчування інтерпритував оператор ділення `‘/’` як цілочисельне ділення. Починаючи з третьої версії існує два оператори ділення: `‘/’` - ділення, результатом якого може бути

дійсне число, ‘//’ - цілочисельне ділення. Окрім того, над цілим числовим типом у Python визначені бітові операції: диз’юнкція (“|”), кон’юнкція (“&”), виключне або (“^”), лівий та правий логічні зсуви (“<<”, “>>”), заперечення (“~”).

Ряди складаються з трьох основних типів - списку (list), кортежу (tuples), набору (set), діапазону (range). Найфундаментальніша структурою даних в Python є список. Список - це впорядкована колекція даних. За своїми властивостями він нагадує масив в інших мовах програмування (C, Java). Для створення використовують синтаксис:

```
mixed_list = [1, False, “Python”, complex(1, 2)]
```

Лістинг 1.2.1

Список може містити елементи різних типів. Доступ здійснюється за допомогою оператора квадратні дужки ([]). Доступатися до елементів списку можна також через від’ємні числа, у такому разів зі списку повертають елемент починаючи з кінця:

```
mixed_list[-1]
(1+2j) ##output
```

Лістинг 1.2.2

Оператор ‘[]’ можна використовувати для того, щоб “розрізати” список:

```
x = range(10)
first_three = x[:3] # [-1, 1, 2]
three_to_end = x[3:] # [3, 4, ..., 9]
one_to_four = x[1:5] # [1, 2, 3, 4]
last_three = x[-3:] # [7, 8, 9]
without_first_and_last = x[1:-1] # [1, 2, ..., 8]
copy_of_x = x[:] # [-1, 1, 2, ..., 9]
```

Лістинг 1.2.3

Для перевірки чи містить список певний об’єкт Python має оператор “in”:

```
1 in [1, 2, 3] #True
1 in [2, 3, 0] #False
```

Лістинг 1.2.4

Метод append використовують для додавання одного елемента в кінець. Також над списком визначена операція **“+”**, що створює новий список, конкатинуючи два інші. Щоб розширити список іншим використовують метод extend.

```
x = [1, 2, 3]
x.append(1) # x now [1, 2, 3, 1]
x.extend(range(2)) # x now [1, 2, 3, 1, 1, 2]
x + [5, 7] # return [1, 2, 3, 1, 1, 2, 5, 7], x still [1, 2, 3, 1, 1, 2]
```

Лістинг 1.2.4

Python дозволяє “розпаковувати” список в змінні при відомій довжині:

```
x, y = [1, 2] # now x equals 1, y equals 2
```

Лістинг 1.2.5

Наступною важливою структурою є кортеж. Кортеж є дуже схожим на список, окрім однієї властивості: кортеж є незмінним (immutable). Для визначення кортежа використовують круглі дужки або ж просто перераховують необхідні елементи через оператор кому:

```
tuple = (1, 2)
another_tuple = 5, “string”
```

Лістинг 1.2.6

Як і список, кортеж можна використовувати для множинного присвоєння:

```
x, y = 1, 2 # now x is 1, y is 2
x, y = y, x # Pythonic way to swap variables; now x is 2, y is 1
```

Лістинг 1.2.7

Через специфіку for циклу в Python існує тип Range, який репрезинтує незмінний (immutable) ряд чисел і найчастіше використовується для ітерації циклу задану кількість раз з певним кроком. Синтаксис виглядає наступним чином:

```
range (start = 0, stop, step = 1)
```

Лістинг 1.2.8

Range можна перетворювати в список чи кортеж, проте Range має велику

перевагу над ними - завжди займає однакову кількість пам’яті, незважаючи на величину діапазону, який він зберігає. Отож, для ітерації від 0 до 9 слід писати:

```
for i in range(10):
      print(i) #output 0 1 2 3 4 5 6 7 8 9
```

Лістинг 1.1.9

Наступною структурою даних, що відносять до рядів, є набір (set). Від списку та кортежу набір відрізняється тим, що він є колекцією унікальних даних.

```
s = set() # s is empty now
s.add(1) # s = {1}
s.add(2) # s equals {1, 2}
s.add(2) # s still equals {1, 2}
x = len(s) # x equals 2
y = 2 in s # return True
z = 3 in s # return False
```

Лістинг 1.1.10

Набір використовують в двох основних випадках. Оператор in є дуже швидкою операцією в цій структурі даних. Якщо ми маємо велику колекцію даних у якій часто доведеться перевіряти на наявність елемента, то варто обрати набір, а не список. Наступний випадок коли необхідно використовувати набір - це знаходження унікальних елементів в колекції:

```
item_list = [1, 2, 1, 2, 3, 3]
unique_item_list = set(item_list) # equals [1, 2, 3]
```

Лістинг 1.1.11

До типу відображення (mapping) відносять словник (dictionary). Словник - це структура даних, яка співставляє значення з ключем і дозволяє швидко отримувати елемент за відповідним значенням ключа. Для створення словника використовують синтаксис, зображений у Лістингу 1.1.11.

```
empty_dict = {} # Pythonic
```

```
empty_dict2 = dict() # less Pythonic
grades = { "Joel" : 80, "Tim" : 95 } # dictionary literal
joels_grade = grades["Joel"] # equals 80
```

Лістинг 1.1.11

Якщо значення за заданим ключем не існує, то Python створює виняткову ситуацію “KeyError”. Щоб уникнути цієї помилки варто спочатку перевірити на наявність ключа в словника за допомогою оператора `in`. Також можна використати метод `get`, який приймає значення, що буде повернено уразі відсутності ключа, другим параметром:

```
jim_grades = grades.get(“Jim”, 0) # value 0
```

Лістинг 1.1.12

Для того, щоб додати новий елемент до словника, використовують оператор квадратні дужки:

```
grades[“Joseph”] = 83
```

Лістинг 1.1.13

Якщо ключ вже існує, значення буде оновлено. Ключ словника повинен бути незмінним `(immutable)`, тому список неможливо використовувати як ключ у цій структурі даних. Якщо потрібно використовувати один ключ для багатьох значень, то варто обрати кортеж.

1.3. Бібліотеки для Data Science

Python став популярним не лише в науці про дані, але й в розробці веб та десктопних застосувань, плагінів для найрізноманітніших інстурментів. Усе це завдяки простоті синтаксису, потужності виражальних можливостей, легкій розширюваності та багатому вибору бібліотек. Для дослідників даних Python пропонує інтерактивну консоль IPython. Вона працює в браузері, має багатшу палітру опцій ніж стандартний інтерпритатор Python. Зокрема,

IPython дозволяє використовувати HTML, LaTeX, інтерактивно будувати графіки, зберігати документи у відповідних форматах (Html, Pdf і т.д.). NumPy є найфундаментальнішим пакетом для виконання обчислень лінійної алгебри в Python. Більшу частину бібліотеки написано на C/C++ та FORTRAN, що забезпечує її ефективність. Якщо необхідно працювати з табличною структурою даних, варто використати бібліотеку Pandas. Вона містить потужний DataFrame об’єкт, що фактично є багатовимірним масивом, який дозволяє виконувати складні алгебраїчні обчислення. SciPy є ще одним базовим пакетом для наукових досліджень. Він дещо розширює базовий функціонал NumPy і містить широкий спектр функцій для лінійної алгебри, інтерполяції та кластеризації. Для математичної статистики існує бібліотека SymPy. ЇЇ функціонал дозволяє проводити алгебраїчні та геометричні обчислення, працювати з дискретною математикою та навіть квантовою фізикою. Разом з NumPy та SciPy часто використовують matplotlib, яка дозволяє інтерактивно будувати різного типу графіки та діаграми. Існують ще десятки інших пакетів, але надалі увагу буде зосереджено на NumPy, Pandas та matplotlib.

2. МАТЕМАТИЧНІ БІБЛІОТЕКИ

2.1 Бібліотека NumPy

NumPy є скороченням від Numerical Python. Головним об’єктом в NumPy є однорідний багатовимірний масив. Даний масив є таблицею елементів (зазвичай чисел) однакового типу, доступ до яких здійснюється за допомогою кортежів додатніх чисел. У NumPy виміри називають осями, а кількість осей - порядком.

```
>>> import numpy as np
>>> a = np.array([[1, 0, 0], [0, 1, 2]])
```

Лістинг 2.1.1

Масив у Лістингу 2.1.1 має порядок 2, перший вимір з розмірністю 2 та другий вимір - 3. Клас масиву NumPy є `ndarray` (або його псевдонім - `array`). Найважливішими атрибутами цього класу є: `ndarray.ndim` -- кількість вимірів (порядок); `ndarray.shape` -- розмірність кожного виміру, який відображають за допомогою кортежа; `ndarray.size` -- загальна кількість номерів в масивів (добуток значень `ndarray.shape`); `ndarray.dtype` -- об'єкт що описує типи елементів (NumPy визначає додаткові типи надих, наприклад `numpy.int32`, `numpy.int16`, `numpy.uint32`, `numpy. float64`); `ndarray.itemsize` -- розмір кожного елемента в масиві у байтах (напркиклад, елементи типу `dtype.float64` мають `itemsize` 8 (64/8)); `ndarray.data` -- атрибут, що містить елементи.

Існує декілька способів створити масив NumPy. Першим і найбільше вживаним способом є створення масиву за допомогою звичайного списку чи кортежа.

```
import numpy as np
a = np.array([1, 2, 3])
a # return array([1, 2, 3])
a.dtype # return dtype('int62')
```

Лістинг 2.1.2

Конструктор `array` перетворює ряд рядів в двовимірний масив, ряд рядів рядів - тривимірний. Також при створені масиву можна явно вказувати тип елементів:

```
c  = np.array([[1,2]], dtype=complex) # return array(1 + 0.j, 2 + 0.j)
```

Лістинг 2.1.2

Досить часто при створенні вміст масиву є невідомим, проте відомою є його довжина. Для таких випадків NumPy надає декілька функцій, що створюють

масив з тимчасовими елементами. Функція `zeros` створює масив, заповнений нулями, `ones` -- заповнений одиницями, а `empty` -- випадковими числами. Дані функції першим параметром приймають кортеж у якому вказують порядок масиву та розмірність виміру. Також додатково можна вказати тип елементів масиву.

```
>>> np.zeros( (3,4) )
array([[ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.]])
>>> np.ones( (2,3,4), dtype=np.int16 )                # dtype can also be
specified
array([[[ 1, 1, 1, 1],
        [ 1, 1, 1, 1],
        [ 1, 1, 1, 1]],
       [[ 1, 1, 1, 1],
        [ 1, 1, 1, 1],
        [ 1, 1, 1, 1]]], dtype=int16)
>>> np.empty( (2,3) )                                 # uninitialized, output may
vary
array([[  3.73603959e-262,   6.02658058e-154,   6.55490914e-260],
       [  5.30498948e-313,   3.14673309e-307,   1.00000000e+000]])
```

Лістинг 2.1.3

Для створення масиву з ряду чисел NumPy надає функцію `arrange(start_number, stop_number, step)`.

```
>>> np.arange( 0, 2, 0.3 )                 # it accepts float arguments
array([ 0. ,  0.3,  0.6,  0.9,  1.2,  1.5,  1.8])
```

Лістинг 2.1.4

Якщо `arrange` використовує дійсні аргументи, передбачити числа, якими буде заповнити масив майже неможливо. Коли необхідно створити масив з певною кількістю елементів у якомусь діапазоні слід використовувати функцію `linspace(start_number, stop_number, quantity_of_elements)`.

Усі арифметичні операції над масивами виконується так само, як і над скалярними елементами. Результатом є новий масив. На від міну від

багатьох матричних мов, операція множення в масивах NumPy також виконується по-елементно.

```
>>> A = np.array( [[1,1],
...            [0,1]] )
>>> B = np.array( [[2,0],
...            [3,4]] )
>>> A*B                         # elementwise product
array([[2, 0],
       [0, 4]])
```

Лістинг 2.1.5

На додаток, існують операції суміщені з присвоєння. Результатом таких операцій є модифікація лівостороннього значення. Коли проводять операції над масивами різних типів, результатом буде більш загальним або більш точний тип.

```
>>> a = np.ones(3, dtype=np.int32)
>>> b = np.linspace(0,pi,3)
>>> b.dtype.name
'float64'
```

Лістинг 2.1.6

Окрім бінарних операцій присутній багатий набір унарних, які реалізовані за допомогою методів класу ndarray. Деякі з них: `ndarray.sum` - сума всіх елементів масиву; `ndarray.min` -- мінімальний елемент масиву; `ndarray.max` - максимальний елемент масиву. Окрім них, NumPy також містить загальні математичні функції такі, як: `sin, cos, exp`.

Масиви в NumPy можна розрізати (slice), як списки в Python. Проте існує декілька особливостей у порівнянні зі списком: розрізання в NumPy можна застосовувати до багатовимірних масивів, розрізання не може бути використаним для розширення розміру масиву, оскільки розмір масиву NumPy є незмінним. Як і при роботі зі списком, внаслідок операції розрізання результатом буде масив NumPy відсилок на обрані елементи, а не

масив копії елементів. Тобто, будь-які зміни над масивом, утвореного внаслідок розрізання, приведуть до змін базового. Синтаксис операції розрізання наступний:

```
nparray[starting_index = 0, stopping_index = n, step = 1]
```

Лістинг 2.1.7

Негативні початковий та кінцевий індекси інтерпритують як `starting_index + n, stopping_index + n`, де `n` - розмір масиву, що розрізають. Негативний крок індексує масив у зворотньому порядку. Для розрізання багатовимірного масиву використовують структуру даних кортеж. Кожен елемент кортежу вказує які елементи необхідно обрати на кожному вимірі. У Лістингу 2.1.8 обрано перші елементи у кожному рядку матриці.

```
a = np.arange(10).reshape(3, 2)
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
>>> a[0:3, 0:1] # example 1
array([[1],
       [4],
       [7]])
>>> a[:, :1] # example 2
array([[1],
       [4],
       [7]])
>>> a[..., :1] # example 3
array([[1],
       [4],
       [7]])
```

Лістинг 2.1.8

У першому прикладі використано найбільш повний синтаксис. Перший елемент кортежу обирає елементи у 1-му вимірі від 0-я включно до 3-ох не включно, другий - у 2-му вимірі елементи від 0-я до 1, тобто лише перший елементи. У другому прикладі запис є дещо компактнішим: у першому вимірі обрано всі елементи, тому праву і ліву границі вибору можна не вказувати; у другому вимірі обрано один елемент спочатку, тому початковий елемент

вибору не вказано. Python дозволяє робити запис ще компактнішим, використовуючи об’єкт `Elipsis` (...). `Elipsis` - це спеціальний об’єкт, який у розрізанні підставляє необхідну кількість “:” для створення кортежу довжини розмірності NumPy масиву(ndarray.ndim). Приклад 3 у лістингу є не зовсім еквівалентним попереднім двом, оскільки він працюватиме для масивів будь-якого виміру, і обиратиме перші елементи в останньому вимірі. Також, якщо результатом розрізання необхідно отримати копії елеменів, треба використати функцію copy(). Кожен numpy.newaxis (що є псевдонімом None) в кортежі вибору розширює розмірність масиву на 1-ну вісь. Додана розмірність є позицією у кортежі вибору:

```
>>> a[:, np.newaxis, :].shape
(3, 1, 3)
```

Лістинг 2.1.9

Масиви NumPy також можна індексувати за допомогою інших масивів, рядів (sequence-like objects). На відміну від розрізання, індексування повертає копію даних, а не масив відсилок. Індексувати можна за допомогою цілих чисел або бульових значень. При цілочисельному індексуванні в оператор квадратні дужки передають NumPy масив чи список з переліком номерів елементів, які необхідно вставити в новий масив.

```
>>> a = np.arange(10) + 2 # return array([2, 3, 4, 5, 6, 7, 8, 9, 10, 11])
>>> a[[1, 1, 0, 7]] # return array([3, 3, 2, 9])
>>> a[np.array([1, 1, 0, 7])] # array([3, 3, 2, 9])
```

Лістинг 2.1.10

Якщо будь-який з індексів виходить за межі масиву, буде створено виняткову ситуацію `IndexError`. Елементи багатовимірного масиву індексують рекурсивно або ж за допомогою списку вказуючи індекси для кожного елементу. Тому два способи вибору є еквівалентними у Лістингу 2.1.11.

```
>>> a = np.arange(10).reshape(2, 5)
array([[0, 1, 2, 3, 4],
       [5, 6, 7, 8, 9]])
>>> a[0][2] # return 2
```

```
>>> a[0, 2] # return 2
```

Лістинг 2.1.12

Для індексування багатовимірного масиву за допомогою цілочисельного списку вказують елементи, що треба обрати для кожної осі. Якщо необхідно обрати всі елементи певного вкладеного виміру, індексувати цю вісь не потрібно.

```
>>> a = np.arange(10).reshape(5, 2)
array([[0, 1],
       [2, 3],
       [4, 5],
       [6, 7],
       [8, 9]])
>>> a[[1, 2]] # selecting all elements with 1 and 2 indeces in 1 axe
array([[2, 3],
       [4, 5]])
>>> a[[1, 2], [1, 1]]
array([3, 5])
```

Лістинг 2.1.13

Застосовуючи оператори порівняння або логічні оператори (`<, >, <=, >=, ==, !=, |, &`) до масивів NumPy можна генерувати вектори (масив NumPy) бульових значень.

```
>>> a = np.arange(10) #return array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> a < -5 # return array([False, False, False, False, False, False, False, False,
False, False], dtype=bool)
```

Лістинг 2.1.14

Бульовий масив чи список можна використовувати для індексування. Бульовий масив повинен бути того самого розміру, що й масив, який індексують. При індексації в оператор квадратні дужки можна явно передавати масив бульових значень, або ж вказати вираз, що згенерує відповідний список. У Лістингу 2.1.15 обрано всі імена окрім ‘Bob’ і ‘Mike’.

```
>>> a = np.array(['Bob', 'Mike', 'Jakob', 'Duke'])
>>> a[(a != 'Bob') & (a != 'Mike')]
array(['Jakob', 'Duke'],
      dtype='<U5')
```

Лістинг 2.1.15

Іноді базових операцій розрізання та індексування не достатньо для створення власних алгоритмів, що відсутні у стандартних бібліотеках. Для таких випадків NumPy містить дюжину дещо складніших операцій. Найбільш вживанішою з таких операцій є маніпулювання формою масиву. Дана операція дозволяє змінювати форму масиву без копіювання його елементів. Щоб виконати цю маніпуляцію над конкретним масивом викликають функцію `reshape` аргументом передаючи кортеж, що містить нову форму масиву. Оскільки розмір масиву є сталою величиною, тому зміна форми не повинна змінювати загальну кількість елементів в масиві. Одним з вимірів, який передають у функцію `reshape` може дорівнювати -1. У цьому випадку значення виміру буде успадковано у масиву, над яким проводять зміну форми. У лістингу наведено приклад перетворення одновимірного масиву у матрицю 3x3.

```
>>> a = np.arange(9) # return array([0, 1, 2, 3, 4, 5, 6, 7, 8])
>>> a.reshape(3, 3)
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
```

Лістинг 2.1.17

Ще однією важливою опцією NumPy масиву є можливість конкатинування. Існують декілька функцій, що дозволяють проводити цю операцію. Найзагальнішою є `numpy.concatenate`. Вона приймає список, кортеж чи набір масивів і об’єднує їх відповідно до вказаної осі. Результат виконання фукнкції concatenate продемонстровано в Лістингу 2.1.18.

```
>>> arr1 = np.array([[1, 2, 3], [4, 5, 6]])
>>> arr2 = np.array([[7, 8, 9], [10, 11, 12]])
>>> np.concatenate([arr1, arr2], axis=0)
array([[ 1, 2, 3],
```

```
[ 4, 5, 6],
[ 7, 8, 9],
[10, 11, 12]])
```

Лістинг 2.1.19

Існують більш часткові функцію для конкатинування: `vstack` - об'єднує масиви за першим виміром, `hstack` - за другим виміром. Оберненою операцією до об'єднання є розрізання. Її здійснють за допомогою функції `split`. `split` розрізає масив вздовж осі на задану кількість масивів. У Лістингу 2.1.20 показано приклад використання.

```
>>>arr = randn(5, 2)
array([[ 0.1689, 0.3287],
       [ 0.4703, 0.8989],
       [ 0.1535, 0.0243],
       [-0.2832, 1.1536],
       [ 0.2707, 0.8075]])
>>>first, second, third = np.split(arr, [1, 2])
>>>first # return array([[ 0.1689, 0.3287]])
```

Лістинг 2.1.20

Для розрізання також існують більше зручні функції, як `hsplit`, `vsplit`, `dsplit`. Повний перелік фукнцій, що дозовляють виконувати розрізання та об'єднання масивів наведено у Таблиці 2.1.1.

Функція	Опис
`concatenate`	Найбільш загальна функція об'єднання. Об'єднує колекцію масивів вздовж однієї осі.
`vstack, row_stack`	Об'єднує масиви за рядами (вздовж нульової осі).
`hstack`	Об'єднує масиви за стовпчиками (вздовж першої осі).
`column_stack`	Схожа на hstack, але перетворює одновимірний масив в двовимірний.
`dstack`	Об'єднує масиви вздовж другої осі.
`split`	Розрізає масив вздовж певної осі і повертає кортеж нових масивів.
`hsplit / vsplit / dsplit`	Зручні функція для розрізу масиву за нульовою, першою, другою осями, відповідно.

Таблиця 2.1.1

2.2 Бібліотека pandas

Робота з бібліотекою pandas - це робота з двома основними структурами даних: Series, DataFrame. Хоча вони не є універсальним інструментом для будь-якої проблеми, вони забезпечують потужну базу для більшості застосувань. Series - це одновимірний об’єкт типу масив (array-like object), що містить масив даних (будь-якого NumPy типу) і асоційовний масив ярликів, що називають індексом. Щоб створити найпростіший об’єкт Series у конструктор треба передати лише масив даних, а масив індексів створюється автоматично зі значеннями від 0 до N - 1 (N - довжина масиву даних). Якщо стандартні індекси не підходять для певної задачі, можна задати свої, передаючи аргументом список індексів. Стрічкове подання об’єкту Series виглядає наступним чином: ліворуч - індекси, праворуч - значення. Щоб отримати масив, що містить дані об’єкта чи індекси, необхідно використати атрибути values чи index відповідно. Як і у звичайному масиві NumPy, доступ до значень Series здійснюється за допомогою оператора квадратні, у які передають відповідний індекс. Також над об’єктом Series можна виконувати скалярне множення, застосовувати математичні функції, фільтрувати за допомогою бульових масивів. Результат цих операцій збереже форму індекс-значення. Описані вище властивості продемонстровано у Лістингу 2.2.1

```
>>> s = pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])
a    1
b    2
c    3
d    4
```

```
dtype: int64
>>> s.values # return array([1, 2, 3, 4])
>>> s.index  # return Index(['a', 'b', 'c', 'd'], dtype='object')
>>> s['b']   # return 2
>>> s[[False, True, False, True]]
b    2
d    4
dtype: int64
```

Лістинг 2.2.1

Series можна уявляти, як впорядкований словник фіксованої довжини, оскільки словник також ставить у відповідність кожному ключу певне значення. Над Series можна виконувати багато тих самих операцій, що і над словником. Напрклад, для перевірки наявності певного ключа в Series можна використати оператор in. Ще одним способом створення Series - є передача в конструктор словника. Індексом у створеному об’єктів Series буде ключі словника, а значення, відповідно, значення словника. Створюючи Series за допомогою словника, також можна явно вказати індекс, при цьому, якщо значення за певним індексом не буде знайдемо в словнику, у Series за цим індексом лежатиме значення `NaN`. Для перевірки існування ненульових/нульових значень існують функції `pandas.notnull(series_obj)` та `pandas.isnull(series_obj)`. Важливою характеристикою для багатьох застосувань є те, що Series автоматично “вирівнює” по-різному індексовані дані в арифметичних операціях. Якщо у якомусь елементі арифметичної операції не існує певного індекса, що існує в іншому елементі, то в результуючому об’єкті значення за цим індексом буде `NaN`. Приклад такого випадку наведено в Лістингу 2.2.2

```
>>> a = pd.Series([1, 2, 3], ['a', 'b', 'c'])
>>> b = pd.Series([1, 2, 3], ['a', 'c', 'd'])
>>> a + b
a    2.0
b    NaN
```

```
c    5.0
d    NaN
dtype: float64
```

Лістинг 2.2.2

Також об’єкт Series та індекс цього об’єкту мають атрибут name. За замовчуванням значення цього атрибуту є порожнім.

DataFrame - це таблична структура даних, що містить впорядковані колекції стовпців, кожен з яких може бути різних типів (числовий, стрічковий і т.д.). DataFrame має індекс для рядка і для стовпчика водночас. Даний об’єкт можна уявляти, як словник, що містить об’єкти Series. Існують декілька способів створення DataFrame. Найбільше загальним є створення за допомогою словника списків чи NumPy масивів однакової довжини. У результаті буде створено DataFrame з неявно створеним індексом і відсортованим стовпчиками. Якщо в конструктор явно передати стовпці, то вони будуть впорядковані так, як були передані. Як і з об’єктом Series, якщо в DataFrame передати стовпчик, який відсутнім у переданому словнику, то значення за цим стовпчиком буде `NaN`. До стовпця в DataFrame можна доступитися в стилі словника - через квадратні дужки, або ж через атрибут. Результатом буде об’єкт Series. Стовпчики можна модифікувати присвоєнням скалярного значення чи масиву значень. При присвоєнні масиву стовпчику, довжина масиву повинна дорівнювати розміру об’єкта DataFrame. Під час присвоєння до стовпчика, який не існує, новий стовпець буде створено. У лістингу 2.2.3 наведемо приклад створення DataFrame об’єкт, модифікації та видалення стовпчика. Іншою досить часто вживаною формою даних є словник вкладених словників. При переданні в конструктор DataFrame інтерпритуватиме зовнішні ключі, як стовпчики, а внутрішні, як індекси для рядків.

```
>>> data = {'a': [1, 2, 3], 'b': [2, 4, 6], 'c': [3, 9, 27]}
>>> d = pd.DataFrame(data, columns = ['a', 'b', 'c', 'd'])
   a  b   c    d
```

```
0  1  2   3  NaN
1  2  4   9  NaN
2  3  6  27  NaN
>>> d['e'] = 1
>>> d['d'] = np.array([1, 8, 8])
   a  b   c  d  e
0  1  2   3  1  1
1  2  4   9  8  1
2  3  6  27  8  1
>>> del d['e']
>>> del d['d']
>>> d
   a  b   c
0  1  2   3
1  2  4   9
2  3  6  27
```

Лістинг 2.2.3

Створений DataFrame можна транспонувати викликавши `dataFrame_obj.T`. DataFrame можна створювати також із словника об’єктів Series. Поведінка конструктора у цьому випадку така ж сама, як при створенні із словника словників. Якщо стовпчики об’єкту DataFrame різного типу, найбільш загальний тип буде обрано, щоб вмістити всі дані. У таблиці 2.2.1 наведено повний перелік даних про те, за допомогою чого можна створити DataFrame об’єкт.

Тип	Коментар
2D NumPy масив	Матриця даних.
Словник масивів, списків чи кортежів	Кожен масив стає стовпчиком в DataFrame. Кожен масив повинен бути однакової довжини.
NumPy масив	Інтерпритують, як словник масивів.
Словник Series об’єктів	Кожне значення стає стовпчиком. Індекси з кожного Series об’єднують разом в рядок індекса.
Словник словників	Кожен вкладений словник стає стовпчиком. Ключі об’єднують в рядок індекса.
Список словників чи Series	Кожен елемент списку стає рядком DataFrame. Об’єднання ключів словника чи Series стає назвою стовпців DataFrame.

Список списків чи кортежів	Інтерпритують, як двовимірний NumPy масив.
Інший DataFrame	Якщо не вказано інших параметрів, DataFrame буде скопійовано.

Таблиця 2.2.1

Об’єкт Index відповідає за збереження назв осей та іншої метаінформації. Будь-який масив чи список назв осей, що використовують при створенні неявно перетворюється в Index. Даний об’єкт є незмінним (immutable). Незмінність є важливою рисою оскільки це дозволяє безпечно надавати доступ до Index, запобігаючи непердбачуваній модифікації. Index має багатий набір методів, найбільш вживанішими з них є: `append` - конкатинує додатковий Index, створюючи новий об’єкт, intersection - знаходить перетин об’єктів Index, `delete` - створює новий Index видаляючи елемент на і-тій позиції.

Важливим методом в об’єктах pandas є reindex, що дозволяє створювати новий об’єкт з даними, який узгоджується з новим індексом. Якщо при переіндексуванні деякий індекс був відсутнім в базовому об’єкті, у новому об’єкті за цим індексом лежатиме значення `NaN`. Іноді необхідно заповнити ці порожні значення. Для цього параметром `method` передають потрібну функцію (напрклад `ffill` -вставляє значення, яке було перед цим ключем), що виконає інтерполяцію. Разом з DataFrame reindex може змінювати як і індекс (рядок), так і стовпчик або обидва одразу. Для переіндексування стовпчиків використовують ключове слово `columns`, передаючи список нових індексів. У Лістингу 2.2.3 створено DataFrame об’єкт та зроблено переіндексування рядків використовуючи для інтерполяції функцію `bfill`.

```
>>> frame = pd.DataFrame(np.arange(9).reshape(3, 3), index=['a', 'c', 'd'],
...     columns=['Kyiv', 'New York', 'Berlin'])
>>> frame
   Kyiv  New York  Berlin
a     0         1       2
```

```
c     3         4       5
d     6         7       8
>>> frame.reindex(['a', 'b', 'c', 'd'], method='bfill')
   Kyiv  New York  Berlin
a     0         1       2
b     3         4       5
c     3         4       5
d     6         7       8
```

Лістинг 2.2.3

Для видалення одного чи більше елементів з певної осі необхідно використати функцію `drop`, передаючи парметром список чи масив значень, які необхідно видалити. Також, якщо видалення проводять над об’єктом DataFrame, необхідно вказати вісь з якої видаляють значенням (за замовчуванням видаляють значення з індексу (рядка)). У Лістингу 2.2.4 видалено стовпчик ‘Kyiv’ та рядки ‘a’, ‘d’.

```
>>> frame = frame.drop(['a', 'd'])
   Kyiv  New York  Berlin
c     3         4       5
>>> frame.drop(['Kyiv'], axis=1)
   New York  Berlin
c         4       5
```

Лістинг 2.2.4

Індексування об’єкту Series відбувається за тими правилами, що й індексування NumPy масиву, за винятком того, що для індексування Series та DataFrame можна використовувати не лише цілі числа, а й значення індексу. При індексуванні за значеннями індексу індексують до кінцевої точки включно. Результатом іднексування DataFrame буде один або більше стовпчиків. Для індексування DataFrame за рядками використовуючи назви рядків існує спеціальне поле `ix`. Це поле дозволяє обирати певну частину рядків та стовпчиків подібно до того, як це в NumPy масивах. Приклад такого індексування продемонстровано в Лістингу 2.2.5

```
>>> frame.ix[['d'], ['Berlin']]
   Berlin
d       8
```

Лістинг 2.2.5

Більш повний набір функцій для індексування DataFrame об’єктів зображено в Таблиці 2.2.3.

Функція	Коментар
`obj[val]`	Обирає стовпчик за назвою.
`obj.ix[val]`	Обирає рядок чи декілька рядків за назвою чи порядковим індексом.
`obj.ix[:, val]`	Обирає стовпчики за назвою чи порядковим індексом до кінцевої точки включно
`obj.ix[val1, val2]`	Обирає рядки і стовпчики водночас.
`get_value`	Обирає одне значення за рядком та стовпчиком.

Таблиця 2.2.3

Для лексикографічного сортування Series чи DataFrame визначений метод `sort_index(axis=0, ascending=True)`, передаючи вісь за якою необхідно виконати сортування та порядок сортування (за замовчуванням - зростаючий). Щоб відсортувати DataFrame за декількома стовпчиками водночас необхідно передати аргументом “by” список потрібних стовпців. Функція order призначена для сортування Series за значеннями. Значення NaN внаслідок виконання order помістяться в кінець.

Досить часто доводиться створювати DataFrame об’єкти, читаючи дані з файлу. pandas містить набір функцій для виконання цієї операції. read_csv і read_table - найбільш вживані функції. Основними властивостями функцій читання бібліотеки pandas є:

а) функції автоматично здійснюють індексування, створюючи відповідні індекси, стовпчики та метадані;

б) функції автоматично конвертують значення в потрібний тип;

в) функції автоматично парсять час та дату;

г) ітерувати можна не весь файл;

д) під час парсингу пропускаються коментарі, дійсні числа, розділені комою, символи, що не мають чіткої інтерпритації.

Одним з найпоширенішим типом файлів, у якому зберігають дані для досліджень є csv-файли (coma-separated view). Для парсингу такого файлу функції `read_csv` передають шлях до файлу. Також для цього можна використати функцію `read_table`, передаючи шлях і роздільник ‘,’. Якщо парсинг пройшов успішно, результатом буде об’єкт `DataFrame`. Щоб дозволити pandas створити назви стовпчиків за замовчуванням, необхідно у функцію `read_csv` передати аргумент `header=None`. Якщо потрібно вказати власні назви, треба аргументом names передати список назв. За необхідності можна примусово додати власні індекси (рядки) вказавши їх параметром `index_col`. Якщо у файлі присутні символи, що ідентифікують пропущені дані (NA , -1.#IND , and NULL), у створеному DataFrame об’єкті ці дані будуть позначені `NaN`. Список символі, що ідентифікують пропущені дані можна розширити передавши аргументом na_values список або ж словником, у якому для кожного стовпчика вказані ці символи.

2.3 Візуалізація даних за допомогою MatPlotLib

Розробку бібліотеки MatPlotLib було почато 2002 року Фернандо Перерізом. Метою було створення інструменту для інтерактивної побудови графіків з функціоналом схожим на той, що є в MATLAB. Найчастіше matplotlib використовують разом з IPython. Для роботи з `matplotlib` `IPython` запускають використовуючи `pylab` мод `(ipython --pylab)`. Графіки в matplotlib зберігються в об’єктів Figure. Для створення нового графіка викликають функцію `pyplot.figure()`. При створенні нового об’єкта `Figure`, відкриваються нове порожнє вікно. Щоб отримати посилання на

графік, який відображено в активному вікні необхідно виконати функцію `plot.gcf() (get current figure).` Створити графік у порожній фігурі неможливо, для цього необхідно додати один або більше підграфіків. Приклад створення трьох підграфіків продемонстровано в Лістингу 2.3.1

```
In [5]: figure = pyplot.figure()
In [6]: ax1 = figure.add_subplot(2, 2, 1)
In [7]: ax2 = figure.add_sublot(2, 2, 2)
In [8]: pyplot.plot([1.5, 3.5, -2, 1.6]) # plot on the last used subplot
```

Лістинг 2.3.1

При побудові графіка у Лістингу 2.3.1 за допомогою функції `plot`, буде використано останній створений підграфік. Для створення багатьох об'єктів Figure та підграфіків водночас існує зручна функція `pyplot.subplots`, що повертає кортеж створених об'єктів. Синтаксис створення продемонстровано в Лістингу 2.3.2

```
In [0]: fig, axes = pyplot.subplots(2, 3)
```

Лістинг 2.3.2

У Таблиці 2.3.1 наведено переліком аргументів для функції `subplots`.

Аргумент	Короткий опис
`nrow`	Кількість рядків для підграфіків.
`ncols`	Кількість стовпчиків для підграфіків.
`sharex`	Всі підграфіки повинні використовувати одну вісь X.
`sharey`	Всі підграфіки повинні використовувати одну вісь У.

Таблиця 2.3.1

За замовчуванням pyplot створює між підграфіками відступи, щоб позбутися цих відступів використовують функцію `subplots_adjust(left=None, bottom=None, right=None, top=None, wspace=None, hspace=None).` Основна функція matplolib - plot приймає масив кординат абсцис і масив координат ординат і додатково індикатор кольору та стилю ліній. Стиль ліній задають парметром `linestyle`, кольору -- color. Для найбільш часто

вживаних кольорів існують скорочення (наприклад g - green), для задання будь-якого кольору використовують хеш кольорів. Також графік може мати деякі маркери. Тип маркеру передають параметром `marker`. Окрім цього, можна змінювати тип інтерполяцій (за замовчуванням - лінійний) параметром `drawstyle`. На Рисунку 2.3.1 побудовано графіки з різними типами інтерполяцій: на першому - лінійна, на другому - по-крокова (steps-post), та маркером на першому графіці типу `marker='o'`.

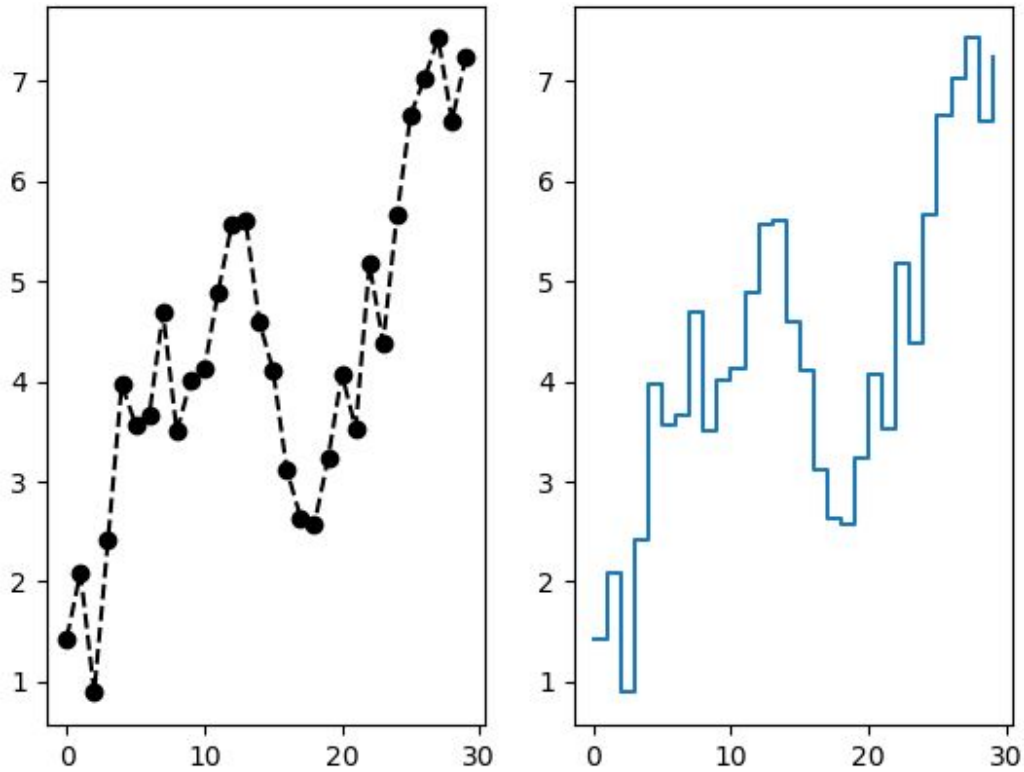

Рисунок 2.3.1

Для більшості операцій побудови графіків існує два основні шляхи: використовувати процедурний pyplot інтерфейс та більш об'єктно-орієнтований - стандартне matplotlib API. pyplot інтерфейс створений для інтерактивного використання, та складається з методів на кшталт `xlim, xticks, xticklabels`. Вони допомагають контролювати діапазон графіка, підказки та їх розташування. Якщо ці функції викликати

без аргументів повернеться їх значення відповідних парметрів, якщо виклкикати з аргументами, значення відповідних властивостей буде змінено. Функція `xlim` - селектор-модифікатор розміру абсициси, `ylim`() - розміру ординати, xticks - встановлює підказки для осі абсцис, `yticks` - осі ординат, xticklabels - текстові підказки для абсциси, `yticklabel` - ординати. Для зміни назви графіка використовуть функцію `setlabel`. У Лістингу 2.3.3 створено графік, змінено назву, додано текстові підказки для осі абсцис та змінено її назву. Результат зображений на Рисунку 2.3.2

```
In [54]: fig, axe = pyplot.subplots(1, 1)
In [57]: axe.plot(randn(1000).cumsum())
In [59]: ticks = axes.set_xticks([0, 250, 500, 750, 1000])
In [62]: labels = axes.set_xticklabels(['one', 'two', 'three', 'four', 'five'],
    ...: rotation=30, fontsize='small')
In [64]: axe.set_title('Sample plot')
In [65]: axe.set_xlabel('Stages')
```

Лістинг 2.3.3

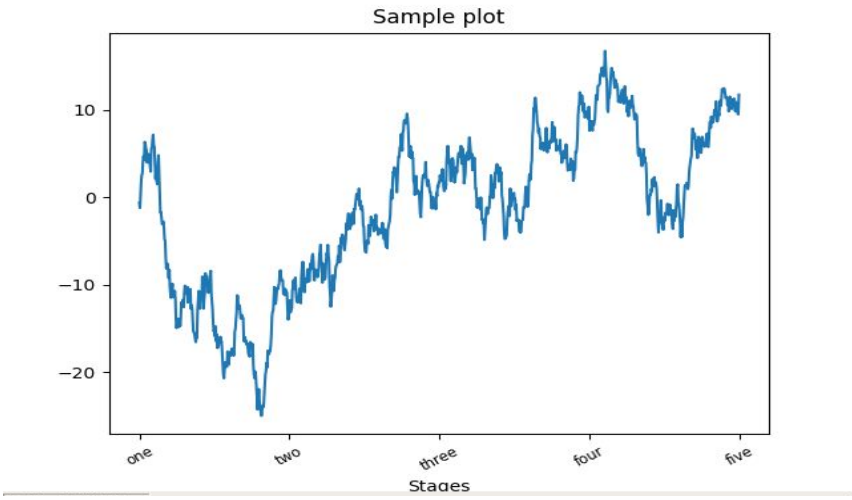

Рисунок 2.3.3

Для визначення елементів графіка необхідна легенда. Існує декілька способів це зробити. Найпростішим є додавання параметру `label` при створені графіка. Після створення необхідно викликати функцію `pyplot.legend`, яка створить легенду з відповідними позначеннями. Окрім `plot` методу, що будує графік за заданими точками, існує ряд функції які дозволяють

створювати більш складні фігури. Деякі з цих функцій наведені в Таблиці 2.3.2

Функція	Короткий опис
`pyplot.pie()`	Будує кругову діаграму.
`pyplot.hist()`	Будує гістограму.
`pyplot.contourf()`	Будує контур.
`pyplot.bar()`	Будує стовпчикову діаграму.
`pyplot.Polygon()`	Створює полігон за задними точками.
`pyplot.Circle()`	Малює коло.
`pyplot.Rectangle()`	Створює прямокутник.

Таблиця 2.3.2

matplotlib є досить низькорівневим інструментом для побудови графіків. Використовуючи API matplolib, pandas створила високорівненві функції, що створюють графіки базуючи на інформації, яку містить DataFrame чи Series об’єкт. DataFrame та Series мають plot метод для побудови різноманітних графіків. За замовчуванням даний метод будує лінійний гафік. Індекс Series об’єкту використовуються matplotlib для побудови осі абсцис. Підказки та ліміти осей можна регулювати використовуючи параметри `xticks` та `xlim`, `yticks` та `ylim`. Також `plot` метод приймає параметр `ax`, який вказує координатну сітку, на якій слід побудувати графік. За допомогою параметрів `label` та `style` можна вказати назву графіка та стиль. Метод `plot` об’єкту DataFrame будує окремий графік для кожного стовпчика. На Лістингу 2.3.4 побудовано графік DataFrame об’єкту, що складається з 4-ох стовпчиків. На Рисунку 2.3.4 зображено результат.

```
In [71]: df = pd.DataFrame(np.random.randn(10, 4).cumsum(0),
    ...: columns=['A', 'B', 'C', 'D'],
    ...: index=np.arange(0, 100, 10))
```

```
In [72]: df.plot()
```

Лістинг 2.3.4

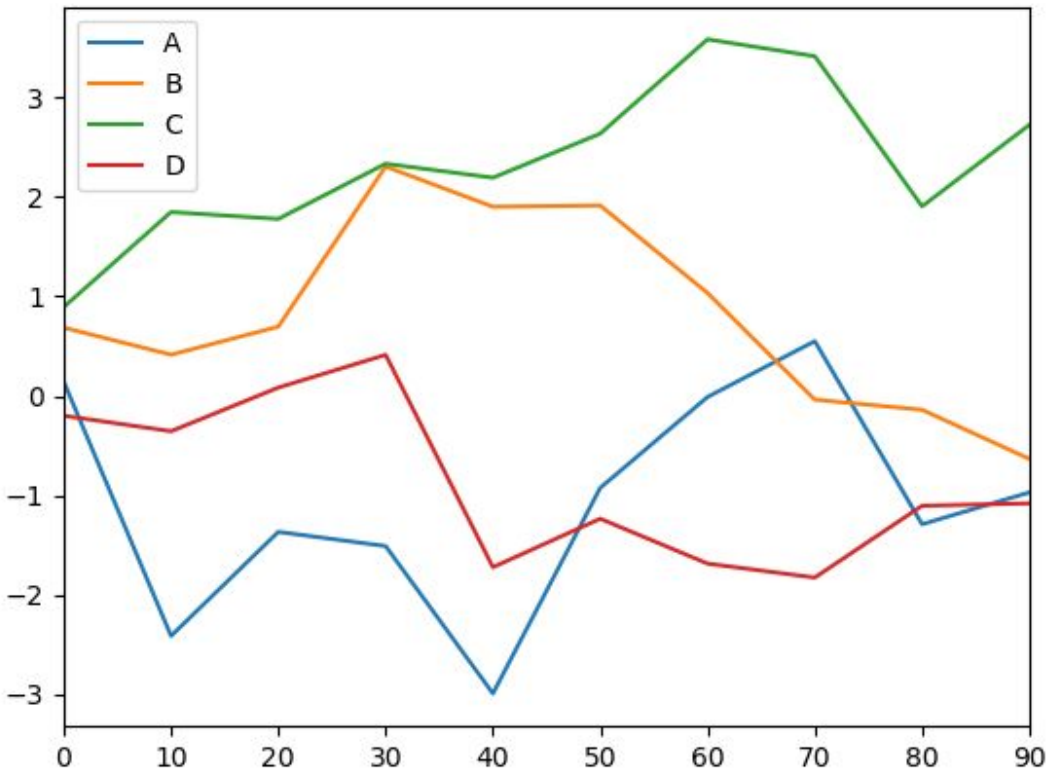

Рисунок 2.3.4

Щоб побудвати кожен графік на окремій сітці, необхідно вказати параметр `sublots=True`.

Окрім лінійних графіків можна будувати стовпчикові діаграми. Для цього у функцію `plot` передають параметр `kind` вказуючи тип `‘bar’` для вертикальних стовпчикових діаграм чи `‘barh’` - для горизонтальних. У Лістингу 2.3.5 побудовано горизонтальну та вертикальну стовпчикові діаграми для об’єкту DataFrame. На Рисунку 2.3.5 показано результат.

```
In [75]: df = pd.DataFrame(np.random.rand(6, 4),
    ...: index=['one', 'two', 'three', 'four', 'five', 'six'],
    ...: columns=pd.Index(['A', 'B', 'C', 'D'], name='Genus'))
In [78]: figure = pyplot.figure()
In [79]: ax1 = figure.add_subplots()
In [83]: ax2 = figure.add_subplot(2, 2, 2)
In [84]: ax1 = figure.add_subplot(2, 2, 1)
In [85]: df.plot(ax=ax1, kind="bar")
```

```
In [86]: df.plot(ax=ax2, kind="barh")
```

Лістинг 2.3.5

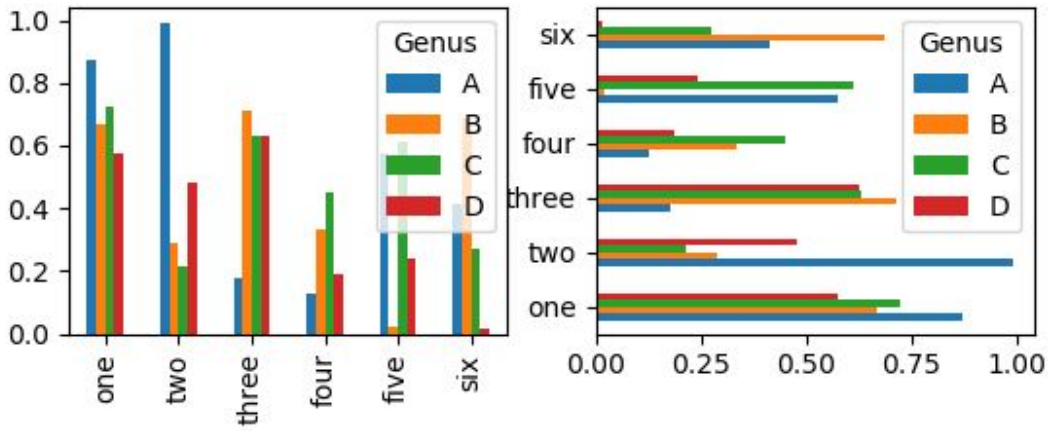

Рисунок 2.3.5

Окрім цих типів графіків можна, також можна створювати гістограми та діаграми розсіювання, використовуючи функції `hist` та `scatter` відповідно.

Розділ 3. МОДЕЛЬ СПАМ ФІЛЬТРУ НА ОСНОВІ НАЇВНОЇ КЛАСИФІКАЦІЇ БАЙЄСА

3.1 Умовна ймовірність. Теорема Байєса.

Умовна ймовірність події B у відношенні до події A називають ймовірність появи події B, за умови попередньої появи події A. Позначають умовну ймовірність P(B|A). Формула для знаходження умовної ймовірності наступна:

$$P(B|A) = \frac{P(A \cap B)}{P(A)}$$

Теорема Байєса є прямим застосуванням умовної ймовірності. З означення умовної ймовірності випливає, що:

$$P(A \cap B) = P(B|A) * P(A)$$

Оскільки $P(A \cap B)$ однаково залежить від A та B, то справидливе твердження:

$$P(A|B) * P(B) = P(A \cap B) = P(B|A) * P(A)$$

Це і є теоремою Байєса, яку формулють так:

$$P(B|A) = P(A|B) * P(B) / P(A)$$

3.2 Наївна класифікація Байєса.

Спам фільтр на основі класифікації Байєса є найпоширенішим статистичним фільтром спаму. Для реалізації цього фільтру, перш за все, необхідно розділити email-повідомлення на слова $w_1 ... w_n$. Позначимо email-повідомлення буковю E. Ймовірність отримання повідомлення E дорівнює ймовірності отримати послідовність слів $w_1 ... w_n$.

$$P(E) = P(w_1 ... w_n)$$

Щоб порахувати ймовірність $P(w_1 ... w_n)$ необхідно врахувати всі комбінації слів $w_1 ... w_n$. Навіть маючи величезний набір даних, зробити це не вдасться. Тому, роблять наївне припущення, що слова у повідомленнях є цілком незалежними один від одного. Іншими словами, ми припускаємо, що поява певного слова в повідомлення як спаму, жодним чином не впливає на ймовірність появи будь-якого іншого слова в повідомленні як спаму. Завдляки цьому припущенню, можна застосувати закон добутку ймовірностей:

$$P(w_1 ... w_n) = \prod_{i=1}^{n} P(w_i)$$

Тепер розділимо слова на два класи: спам (S) і неспам (H). Наступним кроком необхідно порахувати ймовірності появи повідомлення як спаму і як неспаму, тобто:

$$P(E|S)= P(w_1 \dots w_n |S)= \prod_{i=1}^{n} P(w_i|S) \text{ та } P(E|H)= P(w_1 \dots w_n |H) = \prod_{i=1}^{n} P(w_i|H)$$

Для визначення наскільки спамним є те чи інше слово, необхідно набір даних спаму та неспаму, завдяки яким порахуємо ймовірності для кожного слова, тобто порахуємо $P(w_i|S)$ (ймовірність, що email-повідомлення, яке містить слово w_i є спамом) і $P(w_i|H)$ (ймовірність, що email-повідомлення, яке містить слово w_i не є спамом). Для цього порахуємо частоту появи слова в обох групах слів. Після цього залишається використати теорему Байєса:

$$P(w_i|S) = \frac{P(w_i \cap S)}{P(S)}, \quad \text{а також} \quad P(w_i|H) = \frac{P(w_i \cap H)}{P(H)}$$

Це може викликати проблему

Тепер знайдемо відношення цих двох ймовірностей:

$$\frac{P(S|E)}{P(H|E)} = \frac{P(E|S) * P(S)}{P(E|H) * P(H)} = \frac{P(S)}{P(H)} * \prod_{i=1}^{n} \frac{P(w_i|S)}{P(w_i|H)}$$

У зв’язку з обмеженою точністю чисел з плаваючою точкою, об’ємні добутки можуть перетворити число на нуль, тому прологорифмуємо обидві частини рівняння:

$$\log \frac{P(S|E)}{P(H|E)} = \log \frac{P(S)}{P(H)} + \sum_{i=1}^{n} log \frac{P(w_i|S)}{P(w_i|H)}$$

Формула 3.2.1

Використовуючи Формулу 3.2.1 знайдемо логорифм відношення. Якщо результат буде менше за 0 - повідомлення не спам ($P(S|E) < P(H|E)$), більше за 0 - спам ($P(S|E) > P(H|E)$).

3.3. Реалізація моделі спам фільтра на Python

Перш за все, створимо функцію, яка буде розділяти повідомлення на окремі слова. У Лістингу 3.3.1 функція перетворює кожне повідомлення в нижній регістр і розділяє на слова, що складаються з букв і цифр.

```
def create_tokens(message):
    message = message.lower()
    all_words = re.findall("[a-z0-9']+", message)
    return set(all_words)
```

Лістинг 3.3.1

Наступна функція у Лістингу 3.3.2 рахуватиме слова в заданому наборі повідомлень, а повертатиме словник, у якому ключі - окремі слова, а значення - список, що містить кількість екземплярів слова у спам повідомленнях і неспам повідомленнях.

```
def count_words(training_set):
    counts = defaultdict(lambda: [0, 0])
    for message, is_spam in training_set:
        for word in create_tokens(message):
            counts[word][0 if is_spam else 1] += 1
    return counts
```

Лістинг 3.3.2

Наступним кроком рахуватимемо ймовірності появи слова в спамі та неспамі. Функція у Лістингу 3.3.3 повертає кортеж з трьома елементами: слово, ймовірність, що слово є спамом та ймовірність, що слово є неспамом.

```
def word_probabilities(counts, total_spams, total_non_spams, k=0.5):
    return [(w,
             (spam + k) / (total_spams + 2 * k),
             (non_spam + k) / (total_non_spams + 2 * k))
             for w, (spam, non_spam) in counts.items()]
```

Лістинг 3.3.3

Останнє, що треба зробити, це використати ймовірність для кожного слова, щоб знайти ймовірність того, що повідомлення є спамом.

```
def spam_probability(word_probs, message):
    message_words = create_tokens(message)
    log_prob_if_spam = log_prob_if_not_spam = 0.0
    for word, prob_if_spam, prob_if_not_spam in word_probs:
        if word in message_words:
            log_prob_if_spam += math.log(prob_if_spam)
            log_prob_if_not_spam += math.log(prob_if_not_spam)
        else:
            log_prob_if_spam += math.log(1.0 - prob_if_spam)
            log_prob_if_not_spam += math.log(1.0 - prob_if_not_spam)
    prob_if_spam = math.exp(log_prob_if_spam)
    prob_if_not_spam = math.exp(log_prob_if_not_spam)
    return prob_if_spam / (prob_if_spam + prob_if_not_spam)
```

Лістинг 3.3.4

Повна реалізація доступна на https://github.com/larkvincer/course_work.

ВИСНОВОК

Результатом даної роботи стало дослідження та опис мови програмування Python, її базового синтаксису, структур даних та особливостей їх використання. Окрім того, розглянуто основні математичні бібліотеки: NumPy, Pandas, MatPlotLib. Описано структури даних NumPy та Pandas, розглянуто специфіку їх будови та роботи, наведено приклади використання. Продемонстровано, як за допомогою MatPlotLib візуалізувати дані, створювати графіки, стовпчикові діаграми, гістограми, а також розглянуто можливості кастомізації.

З огляду на те, що мова програмування Python має досить багатий набір можливостей, як і кількість бібліотек, створених для неї, описати функціонал у повному обсязі неможливо. Однак зрозумівши базові елементи та принцип роботи, що розглянуті в даній курсові, можна впевнено використовувати Python.

Завдяки створеному скрипту, продемонстровано можливості Python при реалізації алгоритму класифікації повідомлень на основі теореи Байєса. У майбутньому даний скрипт може бути покращеним, створивши інтерфейс консольної взаємодії та змінивши алгоритм розбору повідомлень.

ВИКОРИСТАНА ЛІТЕРАТУРА

1. Allen B. D. Think like a Computer Scientist / B. D. Allen, J. Elkner, C. Meyers., 2007. – 356 с.
2. Grus J. Data Science from Scratch First Principles with Python / Joel Grus., 2015.
3. McKinney W. Think Python, How to Think Like a Computer Scientist-O'Reilly Media / Wes McKinney., 2015. – (2nd Edition).
4. Amith S. Doing Math with Python / Saha Amith. – 245 8th Street, San Francisco: No Starch Press, 2015.
5. Massaron L. Python for Data Science For Dummies / L. Massaron, J. Mueller. – River Street, Hoboken: Published by: John Wiley & Sons, Inc, 2015.
6. Sweigart A. Automate the Boring Stuff with Python / A. Sweigart. – San Francisco: No Starch Press.
7. Tianhao S. Spam Filtering based on Naive Bayes Classification / Sun Tianhao.
8. CSmining group [Електронний ресурс]. – 2010. – Режим доступу до ресурсу: http://csmining.org/index.php/spam-assassin-datasets.html.
9. NumPy Quick start [Електронний ресурс] – Режим доступу до ресурсу: https://docs.scipy.org/doc/numpy-dev/user/quickstart.html.
10. Bayes Theorem [Електронний ресурс] – Режим доступу до ресурсу: http://www.cut-the-knot.org/Probability/BayesTheorem.shtml.