ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ АВТОМАТИЗОВАНИХ СИСТЕМ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «АНАЛІЗ МЕТОДІВ І ТЕХНОЛОГІЙ ОБРОБКИ ДАНИХ В ІНФОРМАЦІЙНИХ СИСТЕМАХ»

на здобуття освітнього ступеня магістра
зі спеціальності 126 Інформаційні системи та технології
освітньо-професійної програми Інформаційні системи та технології

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ *Максим МАРЦЕНЮК*
(підпис) *Ім'я, ПРІЗВИЩЕ здобувача*

Виконав:
здобувач вищої освіти
група ІСДМ-61 Максим МАРЦЕНЮК

Керівник:
науковий ступінь, вчене звання Оксана ТКАЛЕНКО
к.т.н., доцент

Рецензент:
науковий ступінь, вчене звання _____
Ім'я, ПРІЗВИЩЕ

Київ 2023

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут Інформаційних технологій

Кафедра Інженерії програмного забезпечення автоматизованих систем

Ступінь вищої освіти Магістр

Спеціальність Інформаційні системи та технології

Освітньо-професійна програма Інформаційні системи та технології

ЗАТВЕРДЖУЮ
Завідувач кафедрою ІПЗАС

_____Каміла СТОРЧАК
«____» _____20___ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ СТУДЕНТУ
Марценюку Максиму Олеговичу
(прізвище, ім'я, по батькові здобувача)

1.Тема кваліфікаційної роботи: «Аналіз методів і технологій обробки даних в інформаційних системах»

керівник кваліфікаційної роботи Оксана ТКАЛЕНКО, к.т.н., доцент
(ім'я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від «19» жовтня 2023 року № 145.

2. Строк подання кваліфікаційної роботи: 29 грудня 2023 року.

3. Вихідні дані до кваліфікаційної роботи: Методи http GET, POST, PUT, PATCH, DELETE;
Протоколи HTTP, HTTPS, SSH;
Бібліотеки Matplotlib, NumPy, Imageio.
Науково-технічна література з питань, пов'язаних з наукою про дані.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Дослідження особливостей Data Science (науки про дані).

2. Інструменти для аналізу та обробки даних в Data Science.
3. Практична реалізація обробки даних з використанням алгоритму сортування.

5. Перелік ілюстративного матеріалу: *презентація*
   1. Застосування Data Science.
   2. Математичні інструменти для аналізу даних.
   3. Методи HTTP.
   4. Рекомендації по навчанню та дослідженням в області Data Science.

6. Дата видачі завдання: 19 жовтня 2023 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз наявної науково-технічної літератури	19.10 – 27.10.23	
2	Дослідження методів та технологій обробки даних	28.10 – 02.11.23	
3	Використання бібліотеки NumPy	03.11 – 09.11.23	
4	Використання бібліотеки Matplotlib	10.11 – 15.11.23	
5	Реалізація алгоритму сортування даних	16.11 – 21.11.23	
6	Розробка рекомендацій для організації досліджень в області Data Science	22.11 – 10.12.23	
7	Оформлення роботи: вступ, висновки, реферат	11.12 – 20.12.23	
8	Розробка демонстраційних матеріалів	21.12 – 28.12.23	

Здобувач вищої освіти _____ Максим МАРЦЕНЮК
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Керівник роботи
кваліфікаційної роботи _____ Оксана ТКАЛЕНКО
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: ___ стор., ___ рис., ___ табл., ___ джерел.

Мета роботи – проведення комплексного аналізу методів і технологій обробки даних в інформаційних системах, а також розробка рекомендацій для організації навчання та досліджень в області Data Science.

Об'єкт дослідження – процес обробки даних в інформаційних системах.

Предмет дослідження – методи і технології обробки даних в інформаційних системах.

Короткий зміст роботи: Досліджені методи та технології обробки даних в інформаційних системах, визначені рекомендації для вивчення Data Science, досліджені математичні інструменти для аналізу даних. Здійснено обробку даних в інтерактивному середовищі Jupyter Notebook, яке є популярним інструментом серед дослідників, програмістів та аналітиків даних. Здійснено обробку числових даних з використанням бібліотеки NumPy – однієї з основних бібліотек для наукових обчислень в Python. Виконано візуалізацію даних з використанням бібліотеки Matplotlib, яка призначена для створення різних типів графіків, діаграм, графічних представлень даних та інших візуальних зображень для подачі інформації у зрозумілій і апеляційній формі. Визначено роль штучного інтелекту в аналізі даних. Виконано практичну реалізацію обробки даних з використанням алгоритму сортування. Розроблені рекомендації для організації навчання та досліджень в області Data Science.

КЛЮЧОВІ СЛОВА: ДАНІ, ІНФОРМАЦІЙНА СИСТЕМА, МОВА ПРОГРАМУВАННЯ PYTHON, ВІЗУАЛІЗАЦІЯ, ТЕХНОЛОГІЯ, АЛГОРИТМ, СОКЕТ, ПРОТОКОЛ, ІНТЕРАКТИВНЕ СЕРЕДОВИЩЕ.

ABSTRACT

Text part of the master`s qualification work: ____ pages, ____ pictures, ____ table, ____ sources.

The purpose of the work is to carry out a comprehensive analysis of data processing methods and technologies in information systems, as well as to develop recommendations for the organization of training and research in the field of Data Science.

Object of research is the process of data processing in information systems.

Subject of research is methods and technologies of data processing in information systems.

Summary of the work: Methods and technologies of data processing in information systems were studied, recommendations for studying Data Science were determined, mathematical tools for data analysis were studied. Data processing was carried out in the interactive Jupyter Notebook environment, which is a popular tool among researchers, programmers and data analysts. Processing of numerical data was carried out using the NumPy library - one of the main libraries for scientific calculations in Python. Data visualization was performed using the Matplotlib library, which is designed to create various types of graphs, charts, graphs, and other visuals to present information in an understandable and appealing way. The role of artificial intelligence in data analysis was determined. A practical implementation of data processing using a sorting algorithm was performed. Recommendations for the organization of training and research in the field of Data Science were developed.

KEYWORDS: DATA, INFORMATION SYSTEM, PYTHON PROGRAMMING LANGUAGE, VISUALIZATION, TECHNOLOGY, ALGORITHM, SOCKET, PROTOCOL, INTERACTIVE ENVIRONMENT.

ЗМІСТ

ВСТУП

Актуальність теми: За останні десятиліття спостерігається значний приріст обсягу та різноманітності даних, що генеруються та зберігаються інформаційними системами. Це вимагає розробки та впровадження ефективних методів та технологій обробки даних. Багато галузей, включаючи бізнес, медицину, науку, громадський сектор все більше покладаються на аналіз даних для прийняття стратегічних та оперативних рішень. Ефективні методи обробки даних стають ключовими для досягнення успіху у цих галузях. Завдяки розвитку технологій обчислення, стає доступним використовувати потужні обчислювальні ресурси для обробки та аналізу великих обсягів даних, що відкриває нові можливості для розвитку методів обробки даних. Дослідження та аналіз методів і технологій обробки даних в інформаційних системах має велику актуальність для розвитку різних сфер діяльності.

Метою роботи є проведення комплексного аналізу методів і технологій обробки даних в інформаційних системах, а також розробка рекомендацій для організації навчання та досліджень в області Data Science.

Для досягнення поставленої мети необхідно *виконати наступні завдання*:

- дослідити особливості Data Science;
- дослідити інструменти для аналізу та обробки даних;
- виконати обробку даних, використовуючи бібліотеку NumPy мови програмування Python;
- програмно реалізувати алгоритм сортування в інтерактивному середовищі Jupyter Notebook;
- розробити рекомендації для організації навчання та досліджень в області Data Science.

Об'єкт дослідження – процес обробки даних в інформаційних системах.

Предметом дослідження є методи і технології обробки даних в інформаційних системах.

Наукова новизна отриманих результатів полягає у практичній реалізації обробки даних з використанням алгоритму сортування; розробці рекомендацій для організації навчання та досліджень в області Data Science.

Постійно розробляються нові методи та інструменти для обробки даних, такі як машинне навчання, штучний інтелект, обробка природної мови та інші. Дослідження їхнього застосування в інформаційних системах є актуальним завданням.

Апробація результатів магістерської роботи:

Марценюк М.О. «Застосування методу лінійної регресії для аналізу даних». Тези доповіді на І Всеукраїнській науково-технічній конференції «Технологічні горизонти: дослідження та застосування інформаційних технологій для технологічного прогресу України і Світу». – Київ, 28 листопада 2023 року.

1 ДОСЛІДЖЕННЯ ОСОБЛИВОСТЕЙ DATA SCIENCE (НАУКИ ПРО ДАНІ)

1.1 Життєвий цикл Data Science

Наука про дані (Data Science) – це міждисциплінарна галузь про наукові методи, процеси і системи, які стосуються добування знань із даних у різних формах, як структурованих так і неструктурованих.

Наука про дані є продовженням деяких галузей аналізу даних, таких як статистика, класифікація, кластеризація, машинне навчання, добування даних і передбачувальна аналітика.

Станом на теперішній час у мережі Інтернет зберігається велика кількість даних. Якщо у 2010 році даних було 2 зетабайти (1 зетабайт – 1 млрд гігабайт), прогнозується, що у 2025 році їх вже буде 150 зетабайти. Тобто даних дуже багато і потрібно, щоб хтось їх обробляв.

Коли у нас є дуже великий об’єм даних і нас перестає цікавити якийсь повсякденний аналіз даних, ми хочемо передбачати, наприклад, який буде рівень продажів у 2024-2025 році. Можливо є якісь зв’язки між даними, наприклад, якщо це є торгівельний бізнес, то можливо там є якісь асоціації, наприклад, цей товар частіше всього купується з таким то товаром і вибудовування такої логіки продажів, щоб їх підняти.

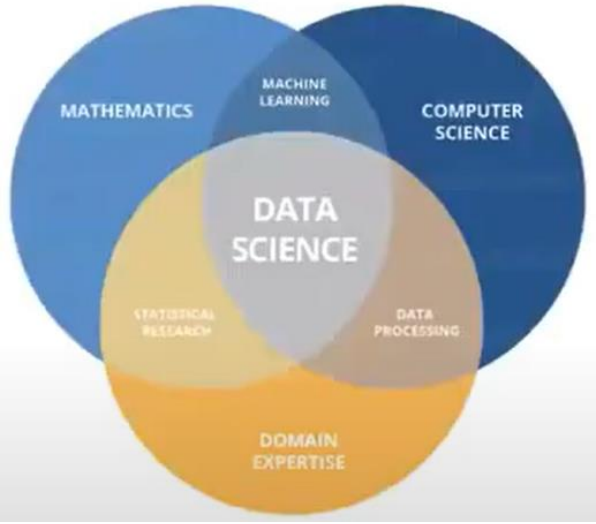

Рисунок 1.1 – Data Science

Data Science складається з таких основних галузей, як математика, програмування, саме розуміння у тій доменній частині, в якій ми працюємо. Про математику потрібно знати: основні поняття математичного аналізу та алгебри; статистику та теорію ймовірності; алгоритми машинного навчання. При вивченні даного напрямку потрібно знати: *алгоритми та структури даних*, тому що даних дуже багато і якщо писати не правильні запити, то ми можемо якусь інформацію отримувати не 20 секунд, а декілька хвилин, якщо алгоритм буде дуже перенавантажений; *методи візуалізації даних* (для візуалізації даних в 90% вакансій використовують або Power BI, або Tablue); *мови програмування* (Python, R та ін.), *бази даних, методи роботи з даними* (Python, SQL – для взаємодії з різними базами даних; *статистичні пакети* (математика, основні математичні методи, теорема Баяса, як приклад, і т.д.). Важливе розуміння в доменній частині, в якій ми працюємо. Також для роботи з даними потрібно розуміти проблематики бізнесу та бізнес-процесів, виконувати правильну постановку завдання.

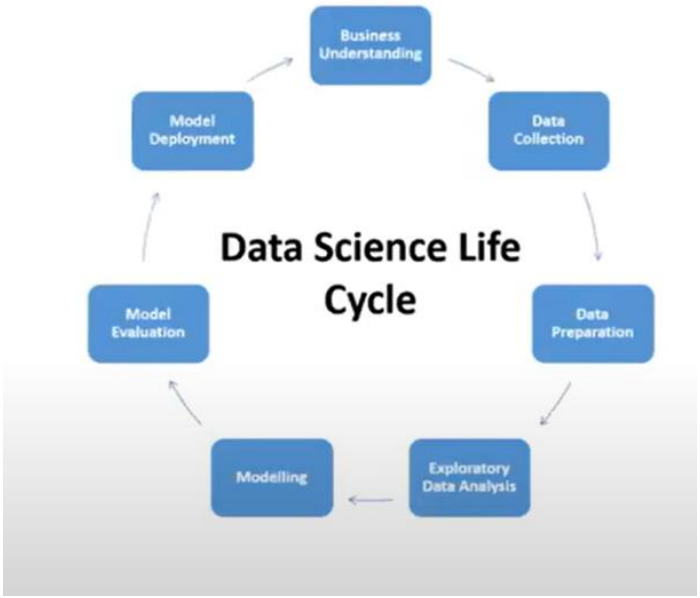

Рисунок 1.2 – Життєвий цикл Data Science

Всі ми починаємо з того, що стикаємося з якимось бізнес-питанням, наприклад, якщо брати медицину, придумали якісь нові ліки від короновірусу. І їх проекспериментувати взагалі чи вони якусь користь приносять чи ні. Для цього ми створюємо якийсь експеримент, збираємо 1000 чоловік й тестуємо на них цей

препарат. Тобто це у нас відбувається збір даних. Далі після того, як ми зібрали дані, ми дані повинні підготувати: почистити їх від всяких викидів. Після того, як ми дані почистили, ми починаємо їх аналізувати: дивимось медіану по віку, медіану кому допомогло, дивимося, які метрики для нас є важливими і після цього аналізу даних ми створюємо модель просту, щоб вона хоча б видавала приблизно те, що нам потрібно. Після того, як ми створили модель, ми вже повертаємося до нашого аналізу і підключаємо метрики, які ми побачили, що вони важливі й дуже добре модель буде передбачувати. Наприклад, якщо у нас є колір волосся, вік, ще якісь метрики, то зрозуміло, що колір волосся користі моделі не принесе, тобто в моделі це буде зайвий фактор, на який вона буде звертати увагу і видавати хибні твердження. Після того, як ми модель покращили до максимуму, модель вже йде у використання, її інтерпретують у якісь сайти або ще щось.

Тобто до нас звертається керівник з певною проблемою. Ми з ним обговорюємо деякі питання і після цього довгий та непростий шлях аналітика даних. Дата-аналітик думає, як краще зібрати дані, звідки їх зібрати, як їх зберігати, як краще їх візуалізувати, чи краще їх зберігати в базі даних чи краще у файлі Excel, це все непросто і це те, що керівник зазвичай не бачить.

Аналітик даних (Data Analyst або дата-аналітик) – це спеціаліст з аналізу великих даних, який збирає дані, виконує обробку даних і робить висновки. На базі його звітів у компаніях приймають важливі рішення. Професія аналітика даних знаходиться на стику IT, менеджменту та математики.

Data collection – це збір даних з різних джерел. Перед тим як збирати дані, ми повинні вияснити: що ми хочемо виміряти або проаналізувати; визначити, які типи даних використовуються та яка кількість інформації нам потрібна.

Наприклад, ми хочемо виміряти або проаналізувати подіють ліки чи ні. Який тип даних нам потрібний: числовий, текстовий, нас цікавитиме true/false, чи будемо ми підходити до кожного пацієнта й питати в нього про його почуття, тобто записувати саме текстове враження пацієнта від препарату. Якщо, наприклад, велика кількість пацієнтів і неможливо кожного перепитати і все вірно

занести в базу даних, можуть дати якусь анкету (основні питання й так/ні). Яка кількість інформації нам потрібна. Наприклад, якщо це якийсь продуктовий магазин і ми хочемо побачити товар продається чи ні, там треба невелика кількість людей. Якщо це медичний препарат, то треба велику кількість людей, щоб дізнатися чи препарат дійсно допомагає. Якщо, наприклад, є A/B тест, то ми можемо за допомогою калькулятора розрахувати, яка кількість користувачів нам потрібна.

Data processing – систематична цілеспрямована послідовність дій над даними. Основні етапи data processing: формалізація даних; фільтрація даних; очищення даних; перетворення даних; захист даних та ін.

Перед аналізом даних треба дані очистити: відфільтрувати, перетворити, захистити. Якщо ми дивимось на дані, то у нас така часта помилка – пропуски, тобто даних часто банально немає. Можемо або їх просто видалити, якщо їх невелика кількість (якщо їх 1% всього) або спробувати якось їх заповнити, наприклад, якщо це вік людини, то можна просто його заповнити середнім. Наприклад, у нас є 10000 чоловік, а у 57 – немає віку, то можна вписати просто середнє арифметичне всіх інших. Очистити дані також, перевести у вірний для нас формат, щоб модель його сприймала, дивлячись яку модель ми використовуємо: деякі моделі не сприймають текстовий формат і дані потрібно перетворювати в числовий. Це все виконується під час дата data processing.

Далі йде дата аналіз – ми будуємо якісь гіпотези і використовуємо різні статистичні методи, тобто вивчаємо дані: data visualization (part of Business intelligence); exploratory data analysis; summary statistics; hypothesis testing.

Після чого у нас йде дата візуалізація. Для data visualization використовуються різні графічні засоби – звіти, графіки, діаграми. Загальноприйнятим засобом візуалізації даних є інформаційні панелі (dashboards), на яких результати відображаються у вигляді індикаторів і шкал, що дозволяють контролювати поточні значення вибраних показників, порівнювати їх з мінімально/максимально допустимими і таким чином виявляти потенційні загрози для бізнесу. Візуалізацію можна зробити в Python, Jupyter Notebook – багато коду, різні

функції. Щоб керівник, який не розуміється на програмуванні, щось розумів, то для цього використовується або PowerBI або Tablue, де ми будуємо різні діаграми, можемо різні метрики вивести йому і він вже це сприйме набагато краще.

Рисунок 1.3 - Приклад дата візуалізації

Машинне навчання (МН, Machine Learning, ML) – великий підрозділ штучного інтелекту, що вивчає методи побудови алгоритмів, здатних навчатися. Машинне навчання застосовується, коли вже нас не цікавить сам аналіз даних. Дата-аналітик аналізує інформацію і видає готовий аналіз. Data Scientist використовує машинне навчання для того, щоб передбачити на основі цього аналізу якісь події. Наприклад, сфера продажів (магазини): зібрали за день багато інформації і далі хочемо створити якісь зв’язки типу: чіпси частіше всього купують з пивом і зробити так, що на вході продаються чіпси, а десь у кінці перед касою продається пиво і поки людина банально дійде то тієї каси, можливо вона ще щось собі захоче взяти. Або якщо ми хочемо передбачити ріст продаж, наприклад, створити якусь модель – лінійну регресію і хочемо побачити, що у 2023 році у нас каса була 1000$, а у вже у 2024 році – буде 5000$. Наприклад, машинне навчання ефективно використовується у сфері таксі. Збиралися дані і потрібно було проаналізувати, скільки приблизно часу буде тривати поїздка. Ми коли замовляємо якесь таксі, то нам кажуть: ось воно буде через 2-3 хв. Це і є явний приклад машинного навчання. Воно розраховує шляхи, дивиться на погоду і видає результат.

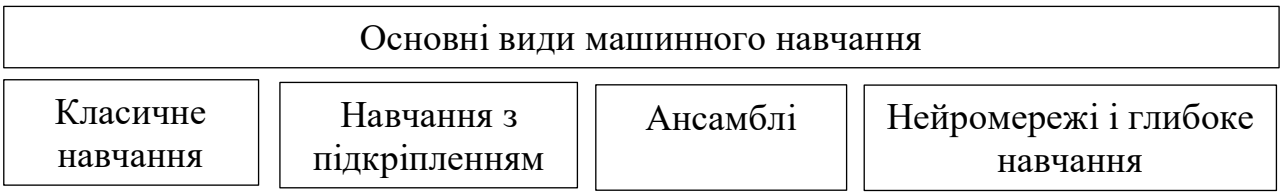

Рисунок 1.4 – Основні види машинного навчання

При використанні класичного навчання застосовуються прості дані та зрозумілі ознаки. Для навчання з підкріпленням характерна особливість при якій даних немає, але є середовище з яким можна взаємодіяти. Ансамблі застосовуються, якщо є якісь проблеми. При використанні нейромереж і глибокого навчання дані є складними з незрозумілими ознаками.

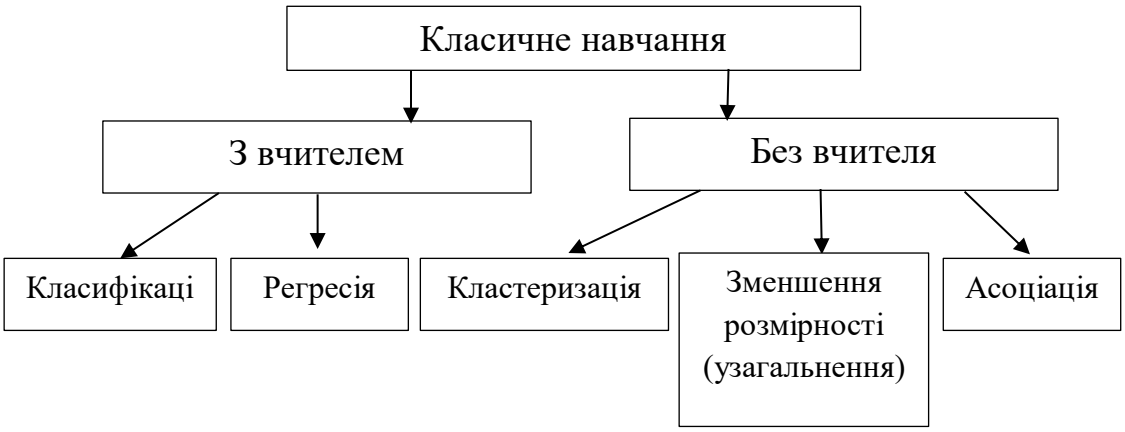

Рисунок 1.5 – Класичне навчання

У класичному навчанні використовуються дані, які наперед категоризовані або числові. У навчанні без вчителя використовуються дані без жодних міток. Задача класифікації дозволяє передбачити категорію, наприклад, розкласти речі за кольором. Задача регресії дозволяє передбачити значення, наприклад, розкласти речі за довжиною. Задача кластеризації дозволяє поділити дані за подібністю. Зменшення розмірності (узагальнення) дозволяє знайти залежності. Асоціація дозволяє виявити послідовності, наприклад, знайти речі, які ми часто носимо разом.

Навчання із вчителем – це коли нас цікавлять історичні дані. Це те, що було раніше. Навчання без вчителя – це коли ми можемо надати прямо зараз дані, і машинне навчання вже щось з ними зробить, залежно від того, що ми будемо

використовувати. Моделі з вчителем поділяються на два типи: класифікація і регресія. Без вчителя: кластеризація, асоціація, зменшення розмірності.

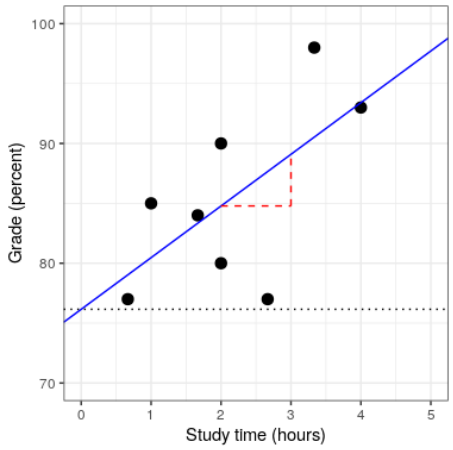

Рисунок 1.6 – Приклад регресії

Якщо не дивитися на X та Y, наприклад, уявити, що по осі Y розміщений прибуток людини, а по X - витрати протягом місяця, то можемо створити графік зелені крапочки, що чим більше людина заробляє, тим більше вона витрачає. Наприклад, людина заробляє 20000 грн., а витрачає 16000 грн. А нам цікаво, скільки б людина витрачала б, якби заробляла 40000 грн. І ось за допомогою регресії ми можемо це дізнатися. Спочатку регресія створюється рандомним чином. А потім ми бачимо, що під час покращення цієї моделі, вона стає все більш і більш ближчою до правди.

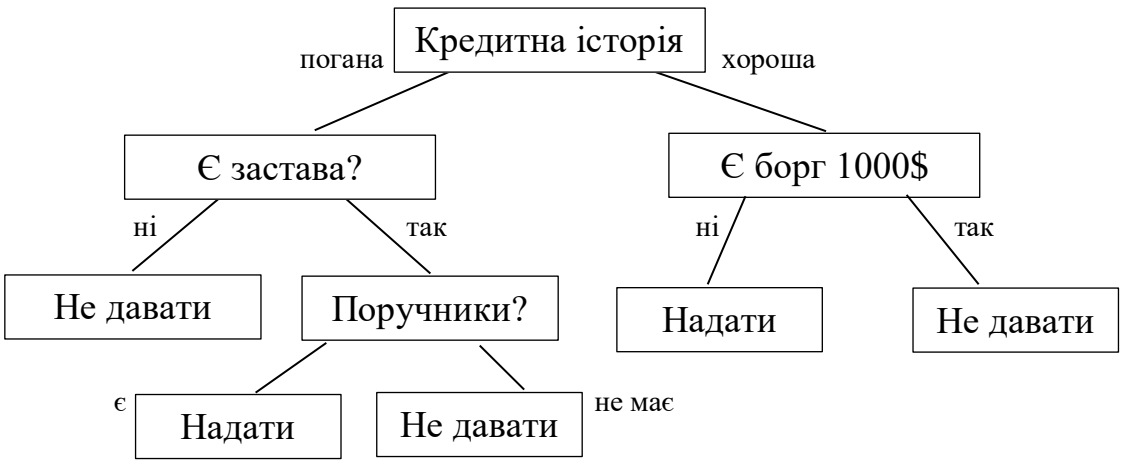

Рисунок 1.7 - Метод класифікації (дерево рішень)

Наприклад, при оформленні кредитів також використовується Data Science (коли банкір визначає можна клієнту давати кредит чи ні). Тобто у нас модель на історичних даних виявила якісь закономірності: тобто люди з таким минулим виплачують кредити, а з таким – не виплачують. І воно на прикладі рис.1.7 вирішує: якщо кредитна історія хороша, то там тільки одне питання – є борг чи немає боргу. Якщо немає – видаємо кредит, якщо є – не видаємо. Якщо кредитна історія погана – то там вже йде більше питань у моделі: чи є застава, чи є поручники, який заробіток, яка освіта і т.д. Це ще також називається деревом рішень. Тобто це приклад, як використовується класифікація.

Тепер поговоримо про навчання без вчителя (рис.1.8) – це кластеризація. Наприклад, використовується тоді, коли потрібно всіх клієнтів розділити на типи. Ми даємо моделі визначену кількість даних і у нас умовно клієнти розділилися на три типи: ті, які часто звертаються і приносять дохід проєкту; ті, які рідко звертаються і ті, які приносять дуже великий дохід.

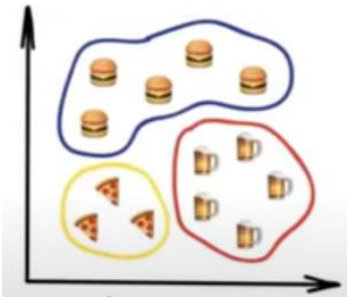

Рисунок 1.8 – Навчання без вчителя – кластеризація (Clustering)

Рисунок 1.9 – Навчання без вчителя – асоціація

Приклад зоомагазину. Data Scientist дали дані, які збирали цілий місяць і він за допомогою машинного навчання дивиться: якщо купується мисочка для кота, то одночасно з цим купується три пакети корму або якщо купується шампунь для котів, то одночасно з цим купується якась розчіска для котів або для собак. І ось можна вибудувати такі асоціації, щоб створити якісь дуже цікаві пропозиції: наприклад, знижка. Наприклад, береш три пакетики корму і на мисочку буде скидка 10%. І це може призвести до збільшення продаж.

Сфер застосування машинного навчання дуже багато. Це спам-фільтри, розпізнавання зображень, сегментація ринку, прогнозування показників, розпізнавання поведінки клієнтів, створення ботів, синтез і розпізнавання мови та ін. Це сфера медицини, транспорту (де й коли краще проїхати), фінанси (продажі), банкінг (чи давати кредит чи ні) та ін. ChatGPT – це теж Data Science.

Для аналізу методів та технологій обробки даних в інформаційних системах рекомендую послідовно вивчати наступні теми: 1) Середовище розробки – Anaconda та Jupyter Notebook; 2) Основи програмування на Python, функціональне програмування, базові бібліотеки; 3) Математичний апарат для Data Science – основи математичного аналізу, лінійна алгебра, теорія ймовірності та статистики, методи оптимізації. При вивченні Python треба знати основні бібліотеки та статистичні пакети, які знадобляться у подальшій роботі. Треба вміти складати запити, так як при погано складеному запиті дані будуть діставатися не 20 сек, а 3-4 хв; 4) Робота з даними – дані у форматах csv, json, xml; робота з API; бази даних та SQL; Numpy та Pandas; 5) Візуалізація даних – візуалізація у Python; візуалізація у Tableau\Power BI. Numpy, Pandas – бібліотеки Python, які нам допомагають взаємодіяти з даними: дивитися, чистити, створювати математичні залежності; 6) Машинне навчання – лінійні моделі, ансамблеві моделі, кластеризація, аналіз соціальних мереж, асоціативні зв’язки, нейронні мережі та їх архітектура.

1.2 Методи та технології обробки даних в інформаційних системах

Обробка даних включає в себе різноманітні ***методи*** і техніки для аналізу, перетворення та витягування інформації з даних. Мною були досліджені та проаналізовані наступні методи обробки даних:

1) Збір даних. Цей етап включає збір даних з різних джерел, таких як бази даних, веб-сторінки, датчики, журнали тощо.

2) Очищення даних. Під час очищення даних ми повинні видалити аномалії, помилки та відсутні дані. Це може включати в себе обробку дублікатів, виправлення помилок у форматі даних тощо.

3) Трансформація даних. Ми можемо змінювати формат даних, об’єднувати дані з різних джерел, перетворювати дані з одного формату в інший, створювати нові ознаки та багато іншого.

4) Витягування даних. Цей процес включає в себе виділення корисної інформації з великих обсягів даних за допомогою різних алгоритмів та методів.

5) Агрегація даних. Ми можемо об’єднувати дані для отримання загальних висновків та статистичних показників.

6) Візуалізація даних. Ми можемо використовувати графіки та інші візуальні елементи для відображення даних у зрозумілій формі.

7) Аналіз даних. Ми можемо використовувати статистичні методи, машинне навчання та інші інструменти для отримання інсайтів з даних.

8) Зберігання даних. Для подальшого використання і резервного копіювання даних їх зазвичай зберігають у спеціальних базах або хмарних сховищах.

Ці методи можуть використовуватися як окремо, так і в поєднанні, в залежності від конкретних завдань і об’єму даних.

Технології обробки даних широко розповсюджені і розвиваються швидко. Мною були проаналізовані деякі з найважливіших технологій, які використовуються в обробці даних:

1) Бази даних. Реляційні бази даних (наприклад, MySQL, PostgreSQL, Oracle, SQL Server) використовуються для зберігання і організації структурованих даних. Крім того, нереляційні бази даних (наприклад, MongoDB, Cassandra, Redis)

використовуються для зберігання неструктурованих або напівструктурованих даних.

2) Big Data технології. Технології, такі як Apache Hadoop і Apache Spark дозволяють обробляти та аналізувати великі обсяги даних (Big Data) на розподілених обчислювальних кластерах.

3) Хмарні обчислення. Платформи хмарних обчислень, такі як Amazon Web Services (AWS), Microsoft Azure та Google Cloud Platform (GCP) надають інфраструктуру та інструменти для обробки, аналізу та зберігання даних у хмарі.

4) Машинне навчання (ML) та штучний інтелект (AI). Машинне навчання і штучний інтелект використовуються для автоматизації обробки та аналізу даних, створення прогнозів та роботи з навчанням даних.

5) Інтернет речей (IoT). IoT технології дозволяють збирати дані в режимі реального часу з різних пристроїв та датчиків, що використовуються у різних галузях, таких як медицина, транспорт, промисловість та ін.

6) Обробка природної мови (NLP). Технології NLP використовуються для аналізу та розуміння текстових даних, таких як соціальні медіа, статті, коментарі тощо.

7) Велика аналітика даних. Це включає в себе інструменти і техніки для виявлення шаблонів, трендів та інсайтів у даних, такі як OLAP (аналіз у режимі онлайн), дашборди та BI (бізнес-інтелект) системи.

8) Автоматизація процесів роботи з даними. Ця технологія використовується для автоматизації рутинних завдань обробки даних і включає в себе інструменти для роботи з ETL (екстракція, трансформація, завантаження) процесами.

9) Блокчейн. Технологія блокчейн використовується для забезпечення безпеки та надійності даних, особливо у фінансових та інших галузях.

Ці технології спільно використовуються для обробки та аналізу даних у різних сферах бізнесу і досліджень. Вибір конкретної технології залежить від потреб та вимог проєкту.

Python – популярна мова програмування для обробки даних, яка має багато бібліотек та інструментів для роботи з даними. Мною були досліджені наступні методи та технології обробки даних з використанням Python:

1) Pandas. Бібліотека Pandas надає зручні інструменти для роботи з табличними даними. Вона дозволяє читати, записувати, очищати, фільтрувати, групувати та агрегувати дані, а також виконувати операції з об'єднанням та злиттям даних. Pandas надає структури даних, такі як DataFrame та Series для аналізу та маніпуляції даними.

2) NumPy. NumPy – це бібліотека для обробки числових даних у Python. Вона надає підтримку для багатовимірних масивів та числових операцій, що роблять її ідеальною для обробки числових даних, таких як масиви, матриці і вектори. NumPy надає ефективні функції для обчислень, обробки масивів і операцій лінійної алгебри.

3) Matplotlib та Seaborn. Ці бібліотеки використовуються для візуалізації даних. Вони дозволяють побудувати графіки, діаграми та інші візуальні представлення даних.

4) Scikit-Learn. Ця бібліотека надає інструменти для машинного навчання та статистичного аналізу даних. Вона включає в себе алгоритми для класифікації, регресії, кластеризації, відбору ознак та інше.

5) TensorFlow і PyTorch. Ці бібліотеки використовуються для роботи з нейронними мережами та глибоким навчанням.

6) SQL та бази даних. Python також може використовуватися для роботи з базами даних за допомогою бібліотек, таких як SQLAlchemy або більш спеціалізованих бібліотек для конкретних систем управління базами даних (наприклад, psycopg2 для PostgreSQL).

7) Регулярні вирази. Python має модуль *re*, який дозволяє виконувати обробку тексту за допомогою регулярних виразів.

8) Зберігання даних. Python може використовуватися для зберігання даних у різних форматах, таких як CSV, JSON, Excel, SQLite та інші.

9) SciPy. SciPy – це бібліотека, яка містить багато наукових та інженерних функцій для обробки даних, включаючи оптимізацію, статистику, інтерполяцію та багато іншого.

10) Dask. Dask – це бібліотека для розподіленого обчислення, що дозволяє обробляти великі обсяги даних та обчислень в паралель.

11) Apache Spark. Це розподілена платфора для обробки даних, яка може взаємодіяти з Python завдяки PySpark.

12) Бібліотеки для обробки текстових даних. SpaCy і NLTK використовуються для обробки текстових даних і виконання завдань обробки природної мови (NLP).

13) Бібліотеки для роботи з геоданими. Геопросторова обробка даних може бути виконана за допомогою бібліотек, таких як GeoPandas і Folium.

Вибір визначених інструментів залежить від завдання і типу даних, з якими нам потрібно працювати.

1.3 Організація підходу до вирішення задач в тому числі наукових з аналізу та обробки даних

Дані – це нафта XXI століття. Банківські рахунки, медичні картки пацієнтів, контент на YouTube, TikTok, Instagram, пости в соцмережах, облік товарів, карти у Google Map - все це дані, відомості про все і всіх. І той, хто вміє аналізувати цю інформацію, буде фахівцем, навички та знання якого будуть дуже цінними у будь-якому куточку світу.

Рисунок 1.10 – Аналіз даних в IT

Аналіз даних – з **наукової точки зору** це вивчення сукупності інформації, де є мета отримання висновків, які дозволяють далі компанії, організації прийняти подальше рішення, наприклад, інвестувати чи брати якісь дані для їхнього огляду чи базуватися на якихось висновках інших експертів. Все це в IT називається вивченням чи інтерпретацією бази даних. Це розглядається для того, щоб досягти вирішення якоїсь проблеми, при цьому ще не забуваємо, що даними можна торгувати, тобто з цього можна отримати дуже гарні гроші, продаючи, наприклад, статистичні показники.

Data Science – наука про дані. Але це все таки розділ інформатики, який і вивчає проблему аналізу, обробки, подання даних, сприйняття у цифровому форматі. І об'єднує він методи обробки даних у великих можливостях, у високому паралелізмі, статистичних методах, інтелектуальних аналізах, застосуванні штучного інтелекту та ін. Однак деякі експерти вважають, що це визначення помилкове, що Data Science – це не наука про дані, а це те, з чого ми можемо добувати знання, використовуючи інші методи, об'єкти.

Data Science – міждисциплінарна область, в якій застосовуються різні процеси, різні наукові методи, алгоритми, системи отримування знань, розуміння даних та інше. Data Science – концепція поєднання статистики, аналізу даних, машинного навчання та пов'язаних з цим методів. Якщо говорити про сучасних вчених, то можна згадати передових вчених у цій галузі, які розказують, що Data Science та аналіз даних на сучасному етапі розвитку технологій – це IV революція для науки. У XX ст. ми всі ведемо обчислення, базуючись на комп'ютерах. У XXI ст. використання методів, основаних на аналізі даних – це вже великий прогрес, коли ми можемо застосувати в роботі величезний об'єм даних, зробити з них статистику і видобути якісь інші корисні показники, якусь суттєву інформацію і зробити висновки вже не просто спостерігаючи чи дивлячись на дані, а вже проаналізувавши, побачивши якісь залежності.

Все це можна робити тільки, базуючись на сучасних комп'ютерах, процесорах, на тому, що дає нам IT.

Data Science – наука XXI ст., це те за чим стоїть академічний розвиток і почався він приблизно з 2010 року, тоді була популяризація концепції великих даних і вивчення цих даних. Це було розмитим, люди приблизно оцінювали ці дані, можна було їх якось погрупувати, поспостерігати за ними – які ж там можуть бути причини і наслідки, і саме потужна обробка даних прийшла трохи пізніше.

У наш час ми часто плутаємо такі концепції як бізнес-аналітика, інтелектуальний аналіз даних, прогнозоване моделювання, статистика із Data Science і з аналізом даних. Проте у багатьох випадках це вже інші історії і це вже не є розмитим поняттям, зараз це все чітко має розмежування. Подивимося, де ж воно розходиться.

Перше, що можна сказати – це бізнес-аналіз. Звичайно в цьому аналізі є структуровані цифрові дані, які дуже обмежують картину навколишнього світу. В Data Science ми можемо брати будь-які дані достатньо для відображення картинки навколишнього світу з повною повнотою. Тому що в Data Science ми можемо взяти не обмежену кількість даних, ми можемо їх урізати, але вони урізаються теж за певними алгоритмами, які в пізнішому не впливають на результат. Тобто, якщо ми помилково щось виріжемо, то ця ймовірність достатньо не висока і результат буде правильним.

Якщо подивитися на основні цілі бізнес-аналізу і Data Science, то бізнес-аналіз – це про аналіз попередніх даних, щоб виявити тенденцію в бізнесі і оцінити вплив попередніх подій на найближче майбутнє. Але в Data Science ми говоримо про прогнозування майбутніх результатів з метою прийняття обгрунтованих рішень, які впливають не тільки на найближче майбутнє, але й точно на декілька років вперед. Якщо вже звернутися до кінцевого результату, то в обох випадках вирішальну роль відіграють фахівці. Головна різниця між цими двома спеціальностями – це те, що експерт у бізнес-аналітиці здатний побачити картину, яка була в минулому до поточного моменту, тоді як Data Scientist повинен не тільки розуміти, що було в минулому і до поточного моменту, але й розуміти як в майбутньому він повинен застосувати ці дані. Це теоретичні підході і різниця між

професіями. Якщо ми вирішили піти в Data Science, маємо пам'ятати про поняття передбачення майбутнього на даних, так як від нашого рішення, нашого результату буде залежати майбутнє не тільки наближене, а це може бути прогноз на декілька років і він суттєво важливий.

На самому початку ми звертаємося до такого поняття як captured – захоплення. Збір даних, введення даних, прийом. Далі переходимо до mainteam – це підтримка, тобто ми зберігаємо десь дані, очищаємо їх і підготовлюємо. На цьому етапі ми також говоримо і про обробку даних. Дуже часто буває таке, що ми отримуємо величезну кількість даних, де є пропущені рядки, де є пропущена важлива інформація, наприклад, таблиця, в якій має бути 8 колонок, але одна сторінка вміщає в себе тільки 3 колонки. Тоді ми повинні розуміти, що скоріше за все цю сторінку нам потрібно видалити або замінити на середній результат чи якийсь інший, при цьому щоб це не впливало на фінальний наш результат, висновки. Далі процес йде в обробці, тобто це інтелектуальний аналіз даних (кластеризація, моделювання даних, узагальнення). Далі йде аналіз – пошуковий чи підтверджуючий: ми шукаємо якісь підтвердження в якості аналізу, в тому що наші прогнози будуть правдиві. Фінальна стадія – communicate – інформування про результати, про передавання даних, візуалізацію і прийняття рішень. Відповідно з цього всього фахівець Data Science повинен вміти не лише вибудовувати та аналізувати, а й обробляти величезні масиви даних, використовуючи безліч різних інструментів (потрібно буде більше, ніж 1 фреймворк на Python). Однозначного опису цієї професії поки що не має, ця професія доволі динамічна, вона дуже сильно змінюється, змінюються до неї вимоги спеціалістів, змінюються очікування, змінюються задачі, а також постійно збільшується кількість даних.

Аналізуючи ринок праці, визначаємо, що фахівець з вивчення даних має вміти:

- Отримувати необхідну інформацію з різних джерел;
- Використовувати інформаційні потоки в реальному часі, тобто тільки сучасну інформацію;
- Встановлювати приховані закономірності у масових даних;

- Статистично аналізувати всі дані для їх ухвалення у грамотне бізнес-рішення.

Data Scientist має бути цікавим, орієнтованим на наслідки, які будуть від отриманого результату. Добре знати особливості галузі, мати комунікаційні навички, які дозволяють пояснювати технічні результати іншим нетехнічним колегам чи бізнес-колегам і багато хто з роботодавців хоче бачити досвід у галузі статистики, лінійної алгебри, моделювання, алгоритмів та ін.

Проаналізувавши вищевказані аспекти в області науки про дані, мною були визначені наступні рекомендації для вивчення Data Science:

1. Вивчення основ програмування. Data Science вимагає розуміння програмування. Популярні мови програмування для Data Science – Python та R. вивчення цих мов допоможе нам розуміти основи синтаксису даних та алгоритмів.

2. Розуміння статистики. Статистика є ключовою складовою Data Science. Вивчення основних концепцій статистики, таких як ймовірність, розподіли, статистичні тести та регресія допоможе нам розуміти та застосовувати аналіз даних.

3. Ознайомлення з основами машинного навчання. Машинне навчання є однією з ключових галузей Data Science. Вивчення основних алгоритмів машинного навчання, таких як лінійна регресія, дерева рішень та методу опорних векторів допоможе нам розуміти, як застосовувати ці алгоритми до аналізу даних.

4. Бази даних та SQL. Розуміння базових принципів баз даних та мови SQL є важливим для ефективної роботи з великими обсягами даних, зберігання та витягування необхідної інформації.

5. Опанування інструментів Data Science. Існує багато інструментів та бібліотек, які сприяють роботі з даними в Data Science. Наприклад, вивчення бібліотек Python, таких як NumPy, Pandas та Scikit-Learn допоможе нам ефективно обробляти та аналізувати дані.

6. Візуалізація даних. Вміння графічно відтворювати та візуалізувати дані, використовуючи інструменти, такі як Matplotlib, Seaborn, Tableau, допомагає в сприйнятті та представленні результатів аналізу даних.

7. Застосування глибинного навчання. Розуміння основних принципів та практичний досвід роботи з нейронними мережами та глибинним навчанням дозволяють спеціалістові розв’язувати складні завдання, такі як розпізнавання образів та обробка природньої мови.

8. Проєктні завдання та практика. Практичний досвід є невід’ємною частиною вивчення Data Science. Потрібно намагатися вирішувати реальні задачі, працювати над проєктами та аналізувати дані. Це допоможе нам закріпити свої знання та навики, а також розвинути креативне мислення.

Все більше компаній приймають рішення на основі даних та використовують штучний інтелект для автоматизації процесів.

2 ІНСТРУМЕНТИ ДЛЯ АНАЛІЗУ ТА ОБРОБКИ ДАНИХ В DATA SCIENCE

2.1 Математичні інструменти для аналізу даних

Для того, щоб бути Data Scientist, вченим, фахівцем, не достатньо мати знання в галузі Python чи аналітичні вміння. Дуже важливо згадати *математику університетського рівня* або ж проходити додаткові курси. В цій галузі застосовуються *інтеграли, похідні* – треба розуміти, що таке похідні, як їх знаходити, обчислювати, знати що таке диференціальні рівняння, які потрібні для аналізу, для зображення даних, особливе потрібне тут знаходження мінімуму і максимуму, як знаходити *локальний мінімум та максимум*. Ця інформація дуже необхідна.

Знати треба для фахівця Data Science – *лінійну алгебру (вектори).* Найпопулярніші вектори – власні вектори в Data Science, їх концепція використовується в алгоритмі машинного аналізу, машинного навчання, в алгоритмі аналізу головних компонентів. Знання власного вектору – необхідне адже завдяки ньому ми можемо зменшити вимір даних з n-вимірного до k-вимірного.

Матриці. Це про лінійну алгебру. *Лінійна алгебра* – базове в аналізі даних, особливо, якщо їх дуже багато.

Основи *теорії ймовірності*. Теорія ймовірності – необхідна в Data Science, так як всі висновки будуть робитися на ймовірностях і нігде не буде 100%-результату. У першу чергу нам тут допоможе *теорема Баєса* і це те, що знаходиться на першому місці для обчислення і розуміння аналізу даних, і розуміння того, що *математика* – це дуже важливий інструмент, який нам може знадобитися в різних неочікуваних сферах життя.

Ще на зображеннях (рис.2.1) є *нормальний розподіл* або *статистика* та основні її поняття – це теж дуже важливий етап. Статистика потрібна для перевірки гіпотез і оцінки найбільшої вірогідності.

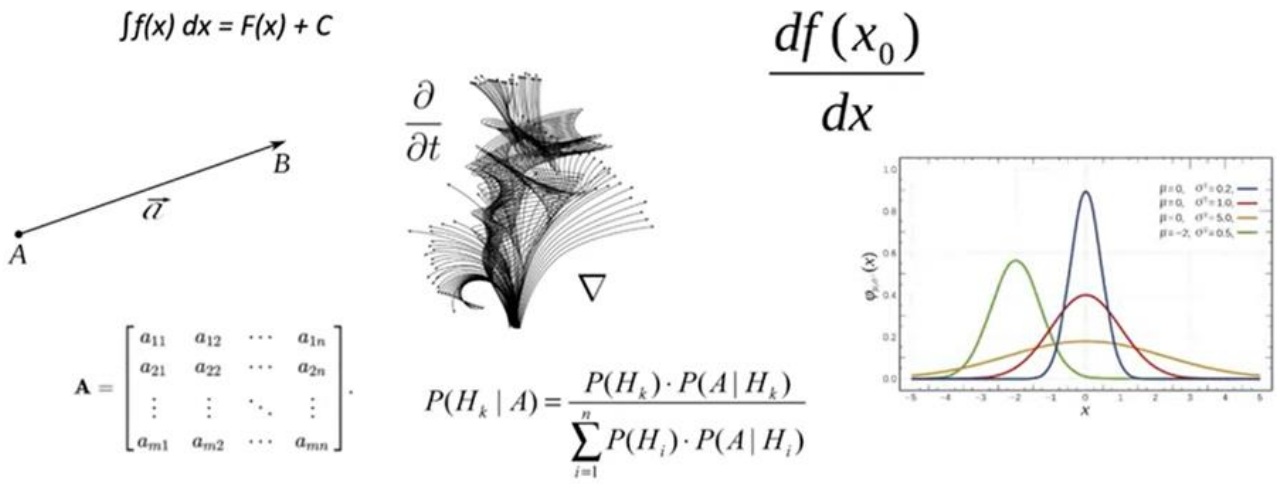

Рисунок 2.1 – Математичні інструменти для аналізу даних

Теорема Баєса і різні розподіли – це одне з таких фундаментальних понять, які будуть запитувати у фахівця Data Science, Data Analyst, в людини, яка хоче працювати з числами, але в інформаційному полі, тобто в сфері IT.

Важливим елементом вивчення є *теорія оптимізації*. Фахівці Data Science мають розуміти її досконало.

Алгоритми і структури даних. Алгоритм сортування (бульбашкою, швидкий). Фахівець в галузі Data Science повинен вміти опрацьовувати складність алгоритмів, тобто вираховувати анотацію О.

Потрібно вміти використовувати всі можливості Python (рис.2.2): фундаментальні (налаштування віртуального середовища, розуміння, які є типи даних, як писати функції, як працювати з файлами, що таке об’єкти і класи та інше).

Додатково може знадобитися робота з базами даних, особливо з базою SQL – часто звідти приходять дані, тобто вони використовуються як вхідні дані і туди відбувається запис фінальних даних після всіх очисток, після того, як ми прибрали непотрібну для нас інформацію.

2.2 Використання можливостей мов програмування Python, R для обробки даних

Python відкриває нам ще більше можливостей для обробки даних, тобто в Python є різні бібліотеки, які спрямовані для обробки великих об’ємів даних, для взаємодії з математикою (як просто вираховувати синуси, косинуси і не треба писати свої функції). Допомагає нам з цим бібліотека **NumPy**. Ще є **Pandas** – дуже часто застосовується, особливо вона є актуальною, якщо ми говоримо про очистку даних і про групування даних. Тобто ми можемо зробити групування даних за якоюсь певною ознакою або видалення їх теж за певною ознакою.

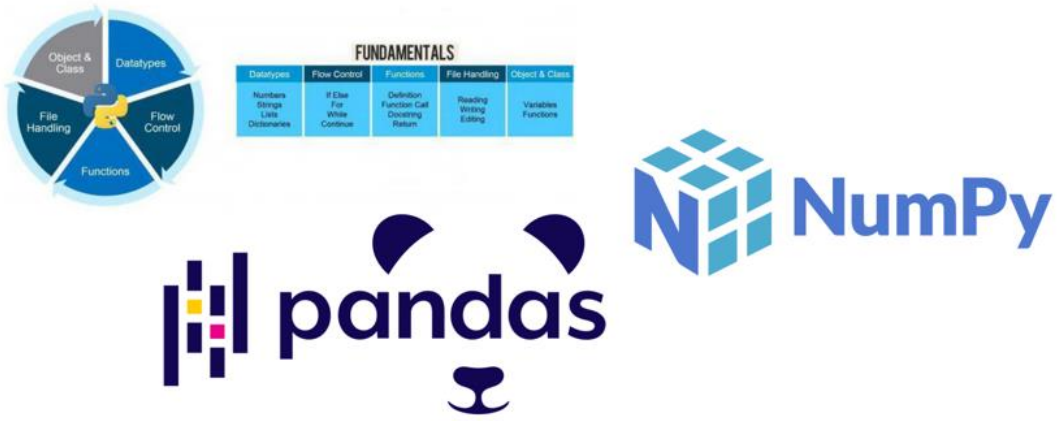

Рисунок 2.2 – Можливості Python

Data Scientist дуже часто використовують такий блокнот як **Jupyter Notebook**, який дозволяє писати і інтерпретувати на ньому код Python у локальному браузері, у web-браузері. Jupyter Notebook дозволяє легко мати справу з кодом, можна виконувати код по частинам, по рядкам, по блокам, як завгодно. Тому Jupyter Notebook часто використовується на практиці, на співбесідах або просто на тренуванні.

Актуально для обробки даних також використовувати середовище **Google Colab** – це різновид Jupyter Notebook, але він від Google і особливо підходить для машинного навчання, для аналізу даних і повністю працює у хмарі. Тобто нам не потрібно дуже навантажувати свій ноутбук, а іноді в ньому не вистачає

процесора, пам’яті або ще чогось, щоб швиденько опрацювати величезний об’єм даних.

При цьому, що Jupyter Notebook, що Colab – безкоштовні, не потребують якогось попереднього налаштування, встановлення пакетом і можна швидко налаштувати доступ по апаратному ключу і працювати з ним.

Приклади застосування Data Science – аналіз структурованих даних за певними критеріями; обробка зображень; статистика даних; перевірка даних; візуалізація.

Jupyter Notebook можемо відкривати локально у web-браузері, можемо писати Python-код та інтерпретувати його. Він дуже зручний і простий. У ньому можна як писати Python-код так і виконувати аналіз даних.

Почнемо з того, які у нас є функції. Елементарна функція – print (рис.2.3). Дописуємо тип даних. Якщо потрібно скористатися звичайним калькулятором, то можемо звернутися до Jupyter Notebook.

Логічні типи даних (True, False). І як бачимо простоту *обробки даних в Jupyter*.

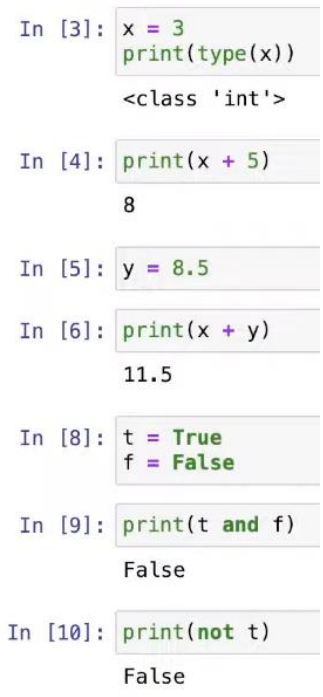

Рисунок 2.3 – Обробка даних в Jupyter Notebook

Можна працювати також зі словниками, наборами, кортежами. У блокноті ще також можна писати функції.

Мова програмування **R** є дуже популярною в області обробки та аналізу даних. Вона має свої особливості та переваги, які роблять її потужним інструментом для роботи з даними:

- векторна обробка: R розглядає дані як вектори, що дозволяє виконувати операції над цілими векторами даних без використання циклів. Це сприяє швидкому та ефективному обчисленню;

- розширені структури даних: R має багато розширених структур даних, таких як вектори, матриці, списки, фрейми даних (data frames) і фактори, які спеціально призначені для обробки різних видів даних;

- багато пакетів: R має велику кількість пакетів (бібліотек), що розширюють її функціональність. Ці пакети надають інструменти для статистичного аналізу, візуалізації, машинного навчання, роботи з графіками, обробки текстових даних;

- велика спільнота користувачів: R має активну та велику спільноту користувачів і розробників. Це означає, що ми можемо легко знайти допомогу, поради і пакети для нашого визначеного завдання;

- інтерактивність: R дозволяє взаємодіяти з даними у режимі реального часу та використовувати його для швидкого експериментування з даними та аналізу результатів;

- візуалізація даних: R має потужні інструменти для візуалізації даних, включаючи бібліотеку ggplot2, яка дозволяє легко створювати графіки високої якості;

- велика підтримка статистичних методів: R включає в себе багато вбудованих статистичних функцій та бібліотек, які роблять її ідеальним інструментом для статистичного аналізу даних;

- публікація результатів: R дозволяє легко створювати звіти та презентації, включаючи динамічні звіти, які можна оновлювати при зміні даних.

Загалом, мова програмування R є потужним інструментом для обробки, аналізу та візуалізації даних і широко використовується у багатьох галузях, включаючи науку про дані, статистику, біоінформатику, фінанси.

Мова R має також свої недоліки та обмеження, які важливо враховувати при виборі мови для роботи з даними:

- R не завжди відзначається високою продуктивністю, особливо при роботі з великими обсягами даних. Деякі операції можуть бути повільними і для великих наборів даних можуть виникати обмеження;
- R може мати обмежену пам’ять і обробка великих даних може вимагати багато оперативної пам’яті. Це може призвести до проблем при роботі з великими обсягами даних;
- R більше спеціалізується на статистичному аналізі даних та візуалізації, тому вона може бути менш ефективною для інших видів завдань, таких як розробка web-додатків або обробка великих потоків даних у реальному часі;
- хоча R має велику кількість пакетів для різних завдань, деякі з них можуть бути погано документовані або не підтримуються активно, що може призвести до проблем з їх використанням;
- для новачків мова R може виявитися складною для вивчення через свої специфічні функції та синтаксис, особливо для тих, хто не має досвіду у статистиці;
- R не завжди є найкращим вибором для масштабованих проєктів або обробки даних у реальному часі, де інші мови, такі як Python або Java можуть бути більш підходящими.

Незважаючи на ці недоліки, R залишається потужним інструментом для статистичного аналізу даних та візуалізації.

Вибір мови програмування для роботи з даними повинен залежати від конкретних потреб та завдань проєкту.

2.3 Обробка числових даних з використанням бібліотеки NumPy

Переходимо до бібліотеки **NumPy** – вона є науковою основою для всіх обчислень на Python і саме ця бібліотека надає високопродуктивний обрахунок у

багатовимірному просторі. Якщо ми говоримо про матриці, про якісь диференціальні рівняння, про складні теореми ймовірності, то тоді це вже не є просто лінійними обчисленнями і тут вже потрібні більш потужні можливості.

Імпортуємо NumPy. Якщо виникають проблеми, повертаємося до терміналу, вводимо: pip install NumPy. Далі робимо матрицю першого рангу. «a» стало об’єктом класу NumPy (рис.2.4). Маємо три значення в одному вимірі.

```
In [1]: import numpy as np

In [2]: a = np.array([1, 2, 3])

In [3]: print(type(a))
        <class 'numpy.ndarray'>

In [4]: print(a.shape)
        (3,)
```

Рисунок 2.4 – Операції з використанням NumPy

Задамо двовимірну матрицю – два рядки по три елементи (рис.2.5).

```
In [5]: a[0] = 7

In [6]: print(a)
        [7 2 3]

In [8]: b = np.array([[1,2,3],[4,5,6]])

In [9]: print(b.shape)
        (2, 3)
```

Рисунок 2.5 – Двовимірна матриця

Так само можемо робити й інші функції з матрицями. Можемо зробити нульову матрицю (рис.2.6).

```
In [10]: a = np.zeros((2,2))

In [11]: print(a)
         [[0. 0.]
          [0. 0.]]
```

Рисунок 2.6 – Нульова матриця

Далі зробимо матрицю, яка буде складатися з одиниць (рис.2.7) й щоб це була двовимірна матриця з одним рядком.

```
In [12]: b = np.ones((1, 2))

In [13]: print(b)
         [[1. 1.]]
```

Рисунок 2.7 - Двовимірна матриця з одним рядком

Наприклад, зробимо матрицю заповнену повністю п’ятірками (рис.2.8).

```
In [14]: c = np.full((2, 3), 5)

In [15]: print(c)
         [[5 5 5]
          [5 5 5]]
```

Рисунок 2.8 – Матриця заповнена п’ятірками

Ідентична матриця (рис.2.9).

```
In [16]: d = np.eye(2)
         print(d)
         [[1. 0.]
          [0. 1.]]
```

Рисунок 2.9 – Ідентична матриця

Можемо задавати матриці з рандомними значеннями, в цьому і суть NumPy – легко взаємодіяти із числами, легко їх придумувати, генерувати, постійно придумувати щось нове. Наприклад, тип boolean можна застосовувати не тільки для змінних, але й для порівняння матриць і це дуже легко з NumPy.

Створимо нову матрицю (рис.2.10). Й скажемо, що ми хочемо знайти елементи, які є більшими, ніж 2.

```
In [17]: a = np.array([[1,2], [3, 4], [5, 6]])

In [18]: bool_idx = (a > 2)

In [19]: print(bool_idx)

[[False False]
 [ True  True]
 [ True  True]]
```

Рисунок 2.10 – Матриця знаходження елементів

Тобто робота з NumPy неосяжна.

Наступними прикладами можуть бути основні математичні функції, які працюють поелементно з масивами і також додавання векторів (рис.2.11).

```
In [1]: import numpy as np

In [2]: x = np.array([[1,2],[3,4]], dtype=np.float64)
        y = np.array([[5,6],[7,8]], dtype=np.float64)

In [3]: print(x + y)

[[ 6.  8.]
 [10. 12.]]
```

Рисунок 2.11 – Принцип додавання векторів (1)

Є гарна практика використовувати функції вбудовані в NumPy (рис.2.12). Виглядає професійніше, але результат той же.

```
In [4]: print(np.add(x, y))

[[ 6.  8.]
 [10. 12.]]
```

Рисунок 2.12 - Принцип додавання векторів (2)

Можемо знайти різницю з матрицею. Вона також може бути написана через мінус, або через окрему функцію в NumPy, як Subtract (рис.2.13).

```
In [5]: print(x - y)
        print(np.subtract(x, y))

[[-4. -4.]
 [-4. -4.]]
[[-4. -4.]
 [-4. -4.]]
```

Рисунок 2.13 – Різниця з матрицею

Так само й перемножити матриці можна як через знак множення так і через multiply функцію, результат буде однаковим (рис.2.14).

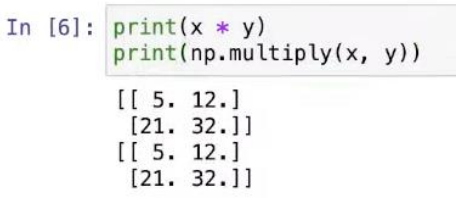

Рисунок 2.14 – Множення матриць

Можна виконувати зведення до степеню, взяття кореню. На відміну від MatLab у NumPy множення поелементне, а не матричне. І для цього ми використовуємо скалярний добуток векторів, а множення матриці на матрицю краще робити через модуль dot (рис.2.15).

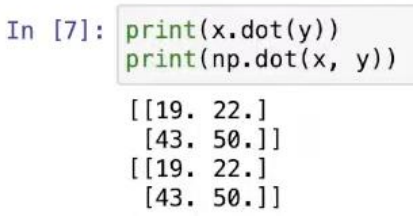

Рисунок 2.15 – Множення матриці на матрицю

Якщо не зрозуміло, де і який метод береться, то в NumPy є дуже чітка документація, тобто відкриваємо, дивимося приклади як вони застосовувалися, як ми можемо використати той чи інший елемент.

Для того, щоб розглядати більше прикладів з NumPy, треба читати документацію і проходити додаткові курси, оскільки інструменти NumPy і Pandas в Python доволі об’ємні і запам’ятати та показати все звичайно неможливо.

Треба далі вчити математику та бібліотеки, так як це все необхідне в Data Science.

Приклади застосування Data Science:

1) Аналіз структурованих даних за певними критеріями. Тобто, коли в нас є якась таблиця, це може бути база даних (наприклад, SQL-база даних і там є структура), там ми можемо легко сказати, що я хочу взяти місто і посортувати

його за певною кількістю людей в різних районах, які проживають. Чи кількість торгових центрів в різних районах. Для цього використовується бібліотека Pandas, вона дуже гарно і дуже швидко працює, і ми можемо самі подивитися приклади.

2) Обробка зображень. Аналізуючи якесь зображення, змінивши його колір, наприклад, або уявивши якого кольору воно може бути, змінивши певні критерії освітлення, ми можемо побачити якісь важливі деталі, коли мова йде про історичне дослідження, наприклад. Це все робиться за допомогою інструментів Python і Data Science.

3) Для статистики даних в першу чергу необхідна математика і презентація даних.

4) Перевірка даних. Буває, що ми маємо величезний об'єм даних і не ясно, чи є там якісь помилки, пропуски, дублікації. Це все можна зробити нашими інструментами і дублікації видаляються елементарно, помилки знаходяться так само і структуризація відбувається не набагато важче. Це все можна реалізувати за допомогою Pandas.

5) Візуалізація важлива, в першу чергу, для клієнта – для якого ми робимо свою роботу, адже людина, яка не з цієї сфери, вона не фахівець, вона може дуже складно уявляти дані, про що ми їй говоримо, чому ми їх описуємо математичними формулами. Коли це виглядає у формі діаграми чи кола – це наглядніше. За допомогою візуалізації можна показати свій фінальний результат. Тоді нас будуть цінувати як фахівців.

2.4 Візуалізація даних з використанням бібліотеки Matplotlib

Бібліотека Matplotlib – це одна з найпотужніших бібліотек для візуалізації даних у мові програмування Python. Вона призначена для створення різних типів графіків, діаграм, графічних представлень даних та інших візуальних зображень для подачі інформації у зрозумілій і апеляційній формі. Основні цілі бібліотеки Matplotlib включають:

1) Візуалізацію даних: Matplotlib дозволяє створювати графіки для відображення різноманітних видів даних, включаючи числові дані, рядки, текст та інше. Ми можемо будувати лінійні графіки, стовпчасті діаграми, колові діаграми, гістограми та багато інших видів графіків.

2) Аналіз та дослідження даних: Matplotlib може бути використаний для відображення різноманітних статистичних даних та розподілів, що сприяє аналізу та візуалізації взаємозв'язків у даних.

3) Подання результатів: бібліотека дозволяє створювати графічні представлення для включення у наукові доповіді, документацію, презентації, статті, веб-сайти та багато іншого.

4) Побудову інтерактивних графіків: Matplotlib має інструменти такі як бібліотеку mpl_toolkits та деякі додаткові розширення, які дозволяють створювати інтерактивні графічні додатки для взаємодії з графіками у реальному часі.

5) Роботу з багатьма платформами: Matplotlibпідтримує багато способів візуалізації даних, включаючи роботу із стандартними рисунками, збереження графіків у файлі (png, pdf, svg та інші) і видавання графіків на різні виведення (екран, принтер, HTML-сторінки).

Matplotlib – потужний і розширюваний інструмент для візуалізації даних, який використовується як у наукових дослідженнях, так і в промислових додатках для візуалізації інформації та роботи з даними.

Наприклад, ми в результаті отримуємо якісь координати точок, які нам нічого не дають. Але, коли ми намалюємо ці точки на осі координат, тоді ми побачимо, що можливо це були там якісь спади та зльоти або точки на глобусі, де були якісь землетруси, наприклад, чи ще щось.

Різні графіки, діаграми, все це у нас залежить від того, як ми подамо матеріал і наші дослідження, наші результати – це все також треба вміти і це все також можна показати за допомогою інструментів Python. Це може бути SciPy, це може бути бібліотека Pillow, може бути Matplotlib-бібліотека та інші.

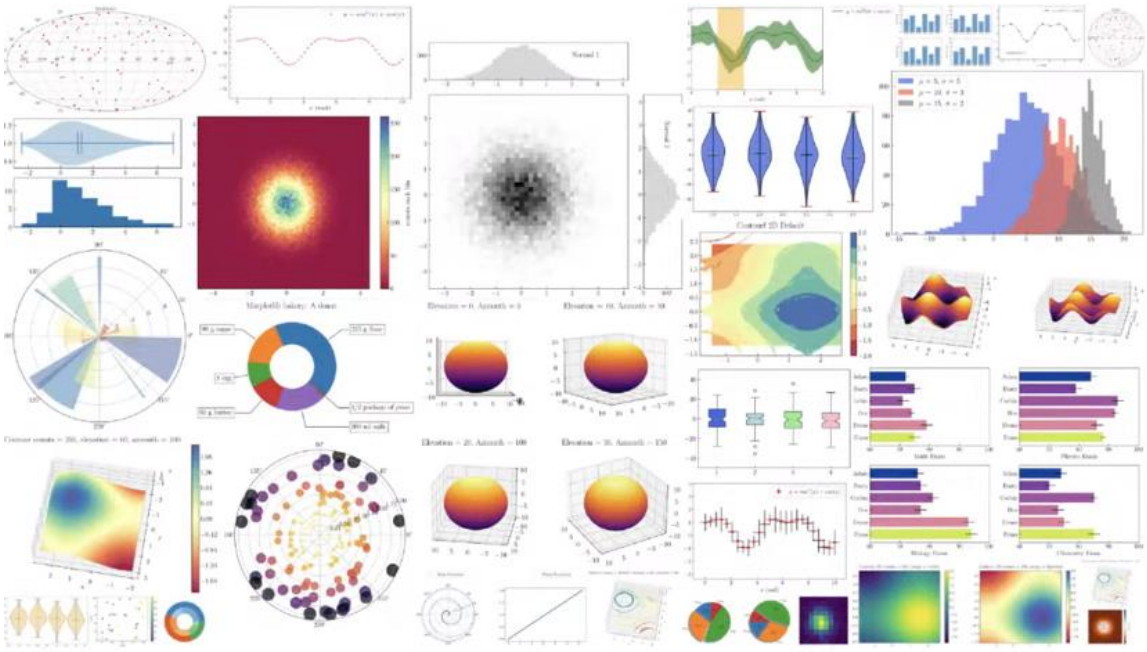

Рисунок 2.16 - Графічне відображення даних

Повертаємося до практики, щоб вчитися показувати результати. Першою у списку стоїть бібліотека PIL. NumPy надає високопродуктивний, багатовимірний масив інструментів для обчислення та керування даними, але PIL базується саме на NumPy і разом вони використовують велику кількість функцій, які корисні для інженерів, якщо їх поєднувати разом. Найкращий спосіб ознайомитися з бібліотекою Pillow – переглянути документацію. Але виділимо частини, які є корисними. І почнемо операції із зображеннями.

На основі бібліотеки PIL ми можемо працювати із зображеннями різних форматів: png, jpeg, bmp та ін. Коли ми завантажуємо файл із зображенням, насправді ми створюємо нове зображення, яке поєднується на різні екземпляри і ми створюємо екземпляр класу image.PIL. Перейдемо до демонстрації цієї функції. У цьому прикладі будемо працювати тільки з картинкою, тому я пропоную імпортувати тільки image із бібліотеки PIL. Картинка заготовлена (у форматі jpeg). Ми його будемо фотошопити за допомогою Python.

Спочатку ми відкриваємо наше зображення. І ми можемо почати форматування із зміни розміру. Ми можемо змінити кількість пікселів на цьому зображенні, змінити саму його форму.

```
In [1]: from PIL import Image

In [2]: image = Image.open('cat.jpeg')

In [5]: new_image = image.resize((500, 500))

In [6]: new_image.save('cat_new_size.jpeg')
```

Рисунок 2.17 – Робота з бібліотекою Pillow

Далі для наших прикладів розглянемо бібліотеку Matplotlib – це важлива бібліотека для візуалізації даних, з цією бібліотекою можна побудувати різні графіки. На рис.2.16 все в основному побудовано з Matplotlib. Коротко розглянемо, яка ж у нас буде система побудови графіків. Візьмемо NumPy, бо ми будемо працювати із складними формулами. Модуль pyplot нам допоможе зібрати до купи графік.

Наприклад, побудуємо систему координат. Скажемо, що у нас *x* буде ось така функція (рис.18), при цьому *y* у нас відповідає просто *sin x*. Задали функції, а далі звертаємося до нашої бібліотеки plt і дивимося, як вона нам побудує залежність *y* від *x*.

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt

In [2]: x = np.arange(0, 3 * np.pi, 0.1)
        y = np.sin(x)

In [3]: plt.plot(x, y)
Out[3]: [<matplotlib.lines.Line2D at 0x7fb0989168b0>]
```

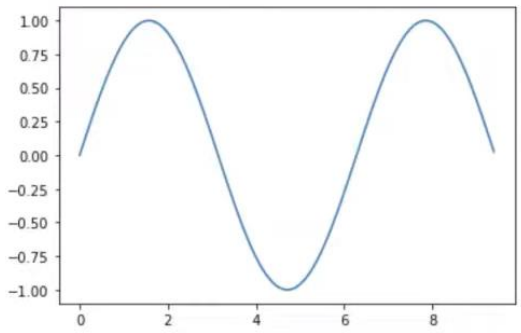

Рисунок 2.18 – Використання бібліотек NumPy і Matplotlib (1)

Зробимо так, щоб не було у нас ніяких зайвих надписів (рис.2.19).

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt

In [2]: x = np.arange(0, 3 * np.pi, 0.1)
        y = np.sin(x)

In [5]: plt.plot(x, y)
        plt.show()
```

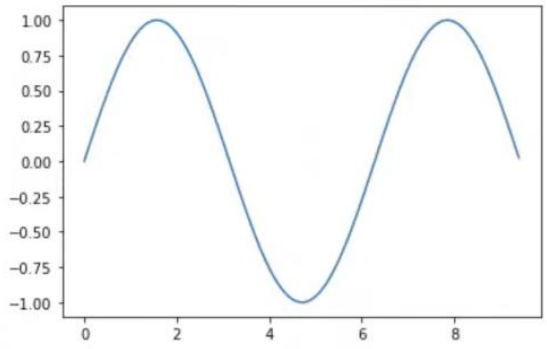

Рис.19 - Використання бібліотек NumPy і Matplotlib (2)

Для красивого графіку нам потрібно додати назву, підписати осі.

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt

In [2]: x = np.arange(0, 3 * np.pi, 0.1)
        y_sin = np.sin(x)
        y_cos = np.cos(x)

In [3]: plt.plot(x, y_sin)
        plt.plot(x, y_cos)
        plt.show()
```

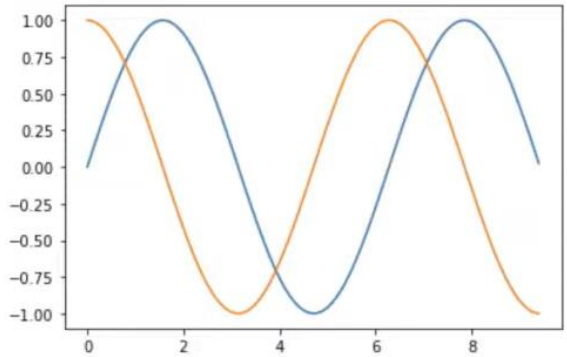

Рисунок 2.20 – Оптимізація графіку

Додаємо назву графіку (рис.2.21).

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt

In [2]: x = np.arange(0, 3 * np.pi, 0.1)
        y_sin = np.sin(x)
        y_cos = np.cos(x)

In [4]: plt.plot(x, y_sin)
        plt.plot(x, y_cos)
        plt.title('Sine and Cosine')
        plt.show()
```

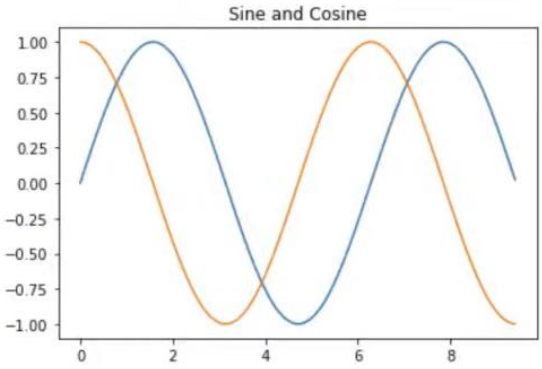

Рисунок 2.21 – Графік з назвою

Додаємо пояснення (рис.2.22).

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt

In [2]: x = np.arange(0, 3 * np.pi, 0.1)
        y_sin = np.sin(x)
        y_cos = np.cos(x)

In [5]: plt.plot(x, y_sin)
        plt.plot(x, y_cos)
        plt.title('Sine and Cosine')
        plt.legend(['Sine', 'Cosine'])
        plt.show()
```

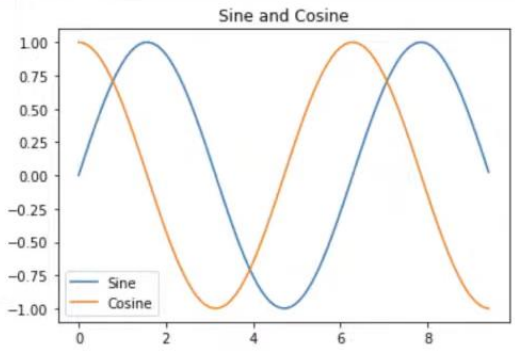

Рисунок 2.22 – Графік з поясненням

Добавляємо назви осей координат (рис.2.23).

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt

In [2]: x = np.arange(0, 3 * np.pi, 0.1)
        y_sin = np.sin(x)
        y_cos = np.cos(x)

In [6]: plt.plot(x, y_sin)
        plt.plot(x, y_cos)
        plt.xlabel('x axis label')
        plt.ylabel('y axis label')
        plt.title('Sine and Cosine')
        plt.legend(['Sine', 'Cosine'])
        plt.show()
```

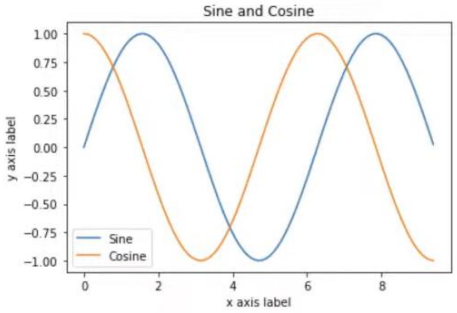

Рисунок 2.23 – Графік з назвами осей координат

Такий графік можна надавати клієнту, фінальному замовнику і пояснювати.

За допомогою методу subplot ми можемо зробити два однакових зображення, які будуть виглядати поряд, для чого задаємо параметрами функцій, які передаються саме в subplot (рис.2.24). Встановлюємо subplot і задаємо йому висоту 2 і ширину 1, остання 1 – початок. У другому прикладі початок далі (останній параметр 2), так як треба, щоб вони не накладалися один на одного.

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt

In [2]: x = np.arange(0, 3 * np.pi, 0.1)
        y_sin = np.sin(x)
        y_cos = np.cos(x)

In [7]: plt.subplot(2, 1, 1)
        plt.plot(x, y_sin)
        plt.title('Sine')

        plt.subplot(2, 1, 2)
        plt.plot(x, y_cos)
        plt.title('Cosine')
        plt.show()
```

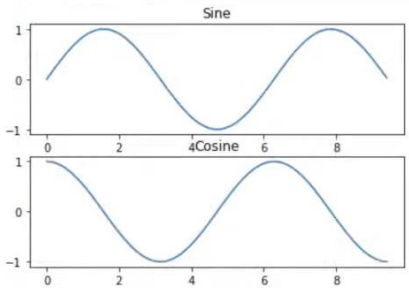

Рисунок 2.24 – Використання методу subplot

Наступний інструмент – бібліотека Imageio – ця бібліотека відносно сучасна. Іноді в літературі ми можемо знайти використання бібліотеки SciPy. Нам потрібно ще використовувати попередню бібліотеку Matplotlib. Засвоюємо попередні наші знання. Тепер відкриваємо зображення нашого кота. Якщо ми в попередніх бібліотеках використовували image.open, то тут вже трошки по іншому, хоча іноді буває, що ці інструменти дуже плутаються між собою, але все залежить від практики, в ході якої ми швидко розуміємо, що і де використовувати. В Imageio для читання зображення використовується функція imread. У цьому прикладі я пропоную профотошопити зображення і сказати, що ми змінимо його тон. Зображення не сприймається бібліотекою, а переводиться у векторну інтерпретацію, тому ми можемо його помножити ще на вектор, де задаємо відсотковість кольорів. Тепер ми скажемо, що ми хочемо побачити початкове зображення і як же нам вдалося профотошопити зображення (рис.2.25). Показуємо початкове зображення з бібліотекою Matplotlib, робимо для цього обов'язково subplot, у нас буде декілька зображень і кажемо їй показати зображення. Звертаємо увагу, що функція show змінилася, бо зараз ми показуємо зображення, а не графік.

```
In [1]: import imageio
        import numpy as np
        import matplotlib.pyplot as plt

In [2]: img = imageio.imread('cat.jpeg')
        img_tin = img * [1, 0.7, 0.9]

In [3]: plt.subplot(1, 2, 1)
        plt.imshow(img)

Out[3]: <matplotlib.image.AxesImage at 0x7f983807d6d0>
```

Рисунок 2.25 – Використання бібліотеки Imageio

Тепер покажемо, як же ми його профотошопили (рис.2.26). Для цього робимо інший subplot. Розташування робимо трошки іншим. Іноді стається ось так (рис.2.26):

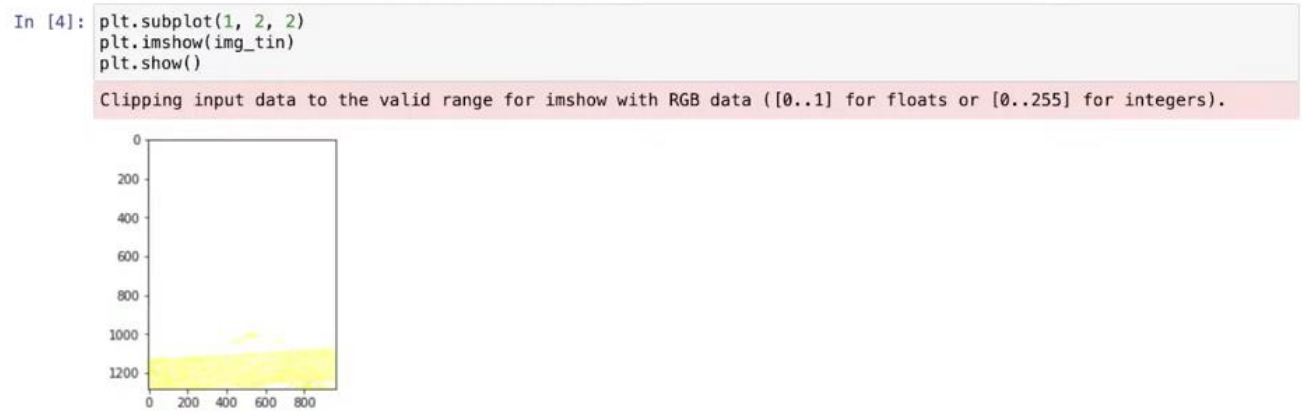

Рисунок 2.26 – Збереження зображення як набору даних

Тобто зображення зберігається не як зображення, а як набір даних, з якими ми можемо проводити різні маніпуляції. Щоб бачити відтюнінговане зображення, потрібно добавити ось таку функцію NumPy (рис.2.27):

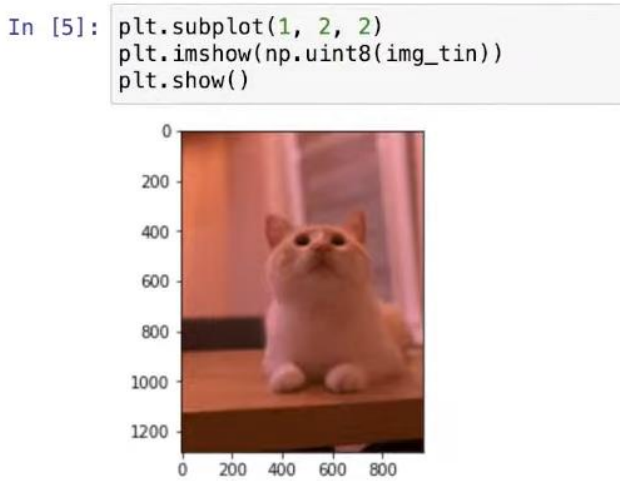

Рисунок 2.27 – Використання функції uint8 у NumPy

Тепер все нормально. Можемо зобразити їх разом (рис.2.28).

```
In [1]: import imageio
        import numpy as np
        import matplotlib.pyplot as plt

In [2]: img = imageio.imread('cat.jpeg')
        img_tin = img * [1, 0.7, 0.9]

In [3]: plt.subplot(1, 2, 1)
        plt.imshow(img)

        plt.subplot(1, 2, 2)
        plt.imshow(np.uint8(img_tin))
        plt.show()
```

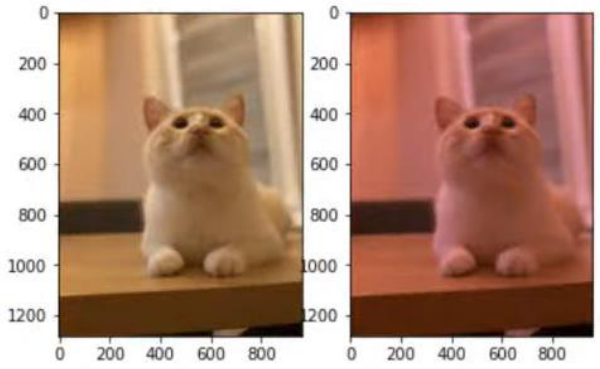

Рисунок 2.28 - Зображення

Ми відфотошопили наше зображення з використанням бібліотеки. Можна зробити дане зображення ще більш червоним (рис.2.29).

```
In [1]: import imageio
        import numpy as np
        import matplotlib.pyplot as plt

In [2]: img = imageio.imread('cat.jpeg')
        img_tin = img * [1, 0.3, 0.7]

In [3]: plt.subplot(1, 2, 1)
        plt.imshow(img)

        plt.subplot(1, 2, 2)
        plt.imshow(np.uint8(img_tin))
        plt.show()
```

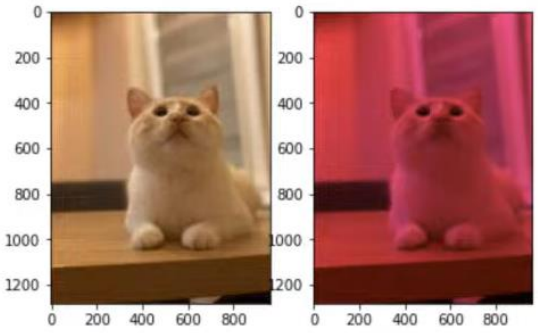

Рисунок 2.29 – Підсилення червоного кольору в зображенні під час його обробки

Ми розглянули доволі багато прикладів про графічне відображення даних, про те, як з ними можна працювати, про те як можна їх показувати поряд, окремо, комбінувати, тюнінгувати.

Рекомендації: використовувати всі ці інструменти для відображення нашого аналізу, наших результатів обробки даних на Python.

2.5 Роль штучного інтелекту в аналізі даних

На сьогоднішній день можна зустріти таку думку, що без штучного інтелекту не можливий аналіз даних (рис.2.30).

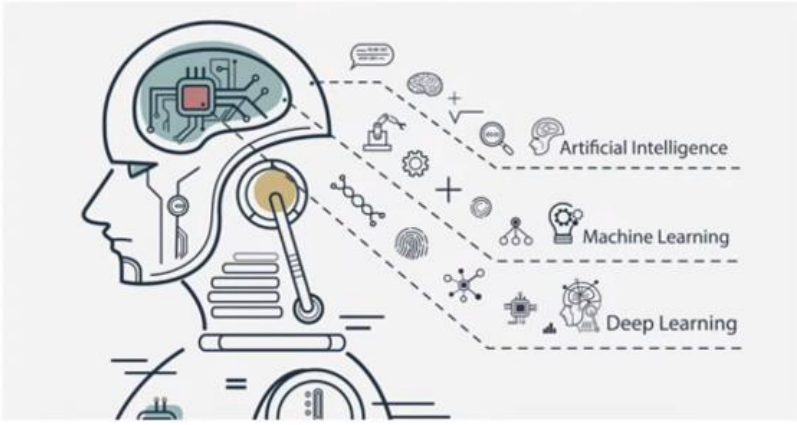

Рисунок 2.30 – Роль штучного інтелекту

Штучний інтелект – більш загальний термін, який охоплює машинне навчання, тоді як глибинне навчання – це тип машинного навчання. Штучний інтелект охоплює низку інших технологій і простіше кажучи – це просто машина, яка може імітувати або втілювати характеристики людського інтелекту. Штучний інтелект використовується в різних галузях для автоматизації, прогнозування, оптимізації завдань, які раніше виконувалися людьми. Він використовується в першу чергу для економії грошей, для економії часу, для економії ресурсів працівників, їм більше не доводиться робити однотипну роботу і повторювати завдання. *Штучний інтелект відіграє одну з найважливіших ролей в аналізі даних*, оскільки саме через штучний інтелект ми можемо запустити якусь модель

і він сам вже буде аналізувати дані або ми можемо запустити частину якоїсь моделі, наприклад, можемо сказати йому, будь-ласка, почисть нам дані від дублікатів, від повторів, від непотрібної нам інформації, від пустих колонок. Якщо вже говорити про сучасний розвиток штучного інтелекту, то треба пам'ятати, що вже є такі поняття, як штучний вузький інтелект, загальний штучний інтелект і штучний суперінтелект. Для аналізу даних ми використовуємо третій, тому що всі інші більше поєднані з побутовими справами для яких не потрібно людині багато навантаження інтелектуального або вони занадто вузькі для того, щоб зрозуміти про весь спектр даних, який ми маємо.

Штучний інтелект бере свій початок саме з **машинного навчання** і саме там зосереджується навчання комп'ютерів, де вони навчаються без програмування, тобто вони отримують вже якісь завдання, але весь досвід йде на основі попередніх заданих моделей і машина вже сама навчається. Машинне навчання має три ключові особливості: *набір даних*, тобто це набір точок даних або зразків. Кожна така точка може бути числом, зображенням, словом, аудіофайлом, відео, чим завгодно. Набори використовуються для навчання моделей машинного навчання. *Функції* – це такі точкові дані, які є рішенням завдання, вони навчають моделі машинного навчання. Вони шукають де ж є такі основні опори і слідкують відповідності машини до заданого алгоритму, заданої моделі. *Алгоритм* – це процес або набір правил, які моделі машинного навчання використовують для аналізу даних, пошуку висновків, відповідей та ін.

Глибинне навчання – підмножина машинного навчання і воно відрізняється від інших типів тим, що навчається на величезних наборах даних, при цьому воно потребує мінімального втручання людини. Це може бути невідомий для людини об'єм даних, дуже великий, наш мозок це все може не сприймати, але це може сприймати глибинне навчання і саме на таких об'ємах даних йде далі його навчання. Одним із найцікавіших аспектів глибинного навчання для штучного інтелекту є те, що він може використовувати для навчання неструктуровані або немарковані дані. Тобто глибинне навчання має здатність навчити без нагляду. Це вже більше, як майбутнє штучного інтелекту.

Штучний інтелект, машинне навчання, глибинне навчання – частина одного предмету, але варто чітко розуміти відмінності, де штучний інтелект – це більш загальний термін для алгоритмів, які перевіряють дані, щоб знайти певні якісь шаблони або рішення. Він нагадує людську здатність виконувати проблеми і більшість проєктів штучного інтелекту використовують машинне навчання та глибинне навчання.

Машинне навчання – тип штучного інтелекту, використовує дані, алгоритми для вирішення однієї або кількох проблем.

Глибинне навчання використовує нейронні мережі не тільки для навчання, але й прогнозування неструктурованих даних.

Майбутнє Data Science – ми можемо управляти даними. Ми можемо натренувати комп'ютерну модель, яка буде сама розбиратися, який результат видати. Чи корисні нам дані. Неочікувані результати. Глибинне навчання – велика допомога, щоб розібратися з неструктурованими даними, оскільки неструктуровані дані – це те, де ми найбільше можемо припуститися помилки, тобто видалити щось дуже важливе або знівелювати чимось важливим.

Без математики неможливо пройти співбесіду, а без штучного інтелекту неможливо розвиватися як спеціалісту в Data Science. Ми пройшлися: робота з числами, робота з математичними формулами, відображення даних, робота із зображеннями, про побудову функцій. Треба розвивати здібності із вищевказаного переліку.

Аналіз має велику користь. Він має різне застосування як для компаній, так і для державних організацій або для тих, хто в першу чергу мають неприбуткові цілі. Це, наприклад, об'єкти господарювання, які хочуть покращити об'єм врожаю. Можливо хтось прагне зменшити недоїдання дітей в тій чи іншій країні. Завдяки інструментам Python ми можемо оцінювати рівень епідемії, наприклад, в дітей чи дорослих, пенсіонерів, різного віку людей в різних діапазонах, країнах, це все доступне з інструментами Python. Подібним чином компанія може аналізувати дані задоволення від клієнтів і проводити після опитування не тільки збір даних, але й аналізувати його, таким чином розробляти свої бізнес-стратегії.

Аналіз даних є актуальним в часи великих даних. Прикладом аналізу даних є, наприклад, поширення в мережах фейкової інформації. Якщо, наприклад, випадково у соціальній мережі написали якусь неправдиву інформацію про людину, країну, то це все можна виправити за допомогою Data Science.

У кар'єрному рості Data Science можна дивитися на наступні вектори: розвиток, як дата лідер, тобто від початку до продвинутого рівня. Data Science буде в тренді, тому що якісь нестандартні, креативні рішення можуть піти тільки від людського мозку, коли ми беремо до уваги критерії персональні чи людські або спиратися на наш попередній досвід.

3 ПРАКТИЧНА РЕЛІЗАЦІЯ ОБРОБКИ ДАНИХ З ВИКОРИСТАННЯМ АЛГОРИТМУ СОРТУВАННЯ

3.1 Програмування сокетів

Розробка алгоритму сортування – це процес створення програмного коду, який впорядковує елементи в масиві або списку у певному порядку. Розробка алгоритму сортування – це завдання, що вимагає уважності та ретельного тестування, оскільки правильність та ефективність сортування можуть вплинути на продуктивність та якість нашого програмного рішення.

Будемо говорити про *сокети* і що таке *клієнт-серверна архітектура*. Будь-що, коли у нас є user, тобто це ми. У нас є ноутбук або якийсь комп'ютер і ми цей ноутбук через Інтернет і ми з вами спілкуємося завжди із сервером. Коли ми заходимо у браузер, коли ми підключаємося до бази даних, коли ми навіть підключаємося до роутера, ми використовуємо сокети.

Сокет – це в принципі така абстракція, яка означає, що через сокет ми можемо працювати із файлами як на нашому комп'ютері, але насправді вони знаходяться не тут. Тобто це можна пояснити як спосіб спілкування між будь-якими двома пристроями. Це просто можуть бути два якихось елементи між якими ми спілкуємося. Тобто ці два елементи обмінюються байтами. Ми підключаємо і робимо базу даних на комп'ютері, ми побачимо, що те, що ми встановлюємо, буде називатися, наприклад, MySQL-сервер. Тобто він стоїть у нас на комп'ютері і ми зробили свою базу даних, наприклад, якісь там оцінки чи будь-що. І ми на своєму компі хочемо до цієї ж бази звернутися, так ось це і буде спілкування між об'єктами. Сокети нам дозволяють це зробити.

Протокол – правила, за якими спілкуються два об'єкти, тобто протоколи є різні і для кожного виду треба різний протокол. Самий популярний – це HTTP-протокол – це правила, за якими клієнт спілкується з будь-яким веб-сервером, наприклад, із сайтом. Або через HTTP-протокол ми можемо працювати з іншою програмою, яка знаходиться на сервері. І цей протокол, коли ми, наприклад, заходимо у браузер. Зайшли у браузер. Бачимо http-адресу (він виглядає як

посилання через слеж, це дуже схоже на файлову систему). Це один з прикладів, що http-протокол працює по правилам, які включають в себе наявність ось такого шляху до чогось. Тобто, коли програми спілкуються, вони знають по якому шляху до них звертатися.

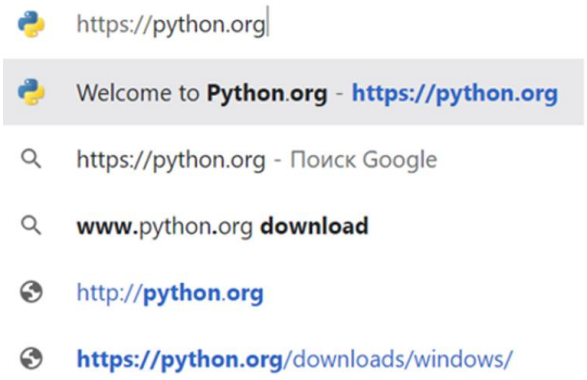

Рисунок 3.1 – Протокол HTTP

Мережевий рівень – це коли наш ноутбук спілкується з роутером, а роутер спілкується з провайдером. Тобто вони працюють по мережевим протоколам.

Рівень OSI	Протоколи
прикладний	HTTP, gopher, Telnet, DNS, DHCP, SMTP, SNMP, CMIP, FTP, TFTP, SSH, IRC, AIM, NFS, NNTP, NTP, SNTP, XMPP, FTAM, APPC, X.400, X.500, AFP, LDAP, SIP, IETF, RTP, RTCP, ITMS, Modbus TCP, BACnet IP, IMAP, POP3, SMB, MFTP, BitTorrent, e2k, PROFIBUS Це всього лише кілька найрозповсюдженіших протоколів прикладного рівня, яких існує неймовірно велика кількість. Всі їх неможливо описати в рамках даної статті.
представлення	ASN.1, XML, TDI, XDR, NCP, AFP, ASCII, Unicode
сеансовий	ASP, ADSP, DLC, Named Pipes, NBT, NetBIOS, NWLink, Printer Access Protocol, Zone Information Protocol, SSL, TLS, SOCKS, PPTP
транспортний	TCP, UDP, NetBEUI, AEP, ATP, IL, NBP, RTMP, SMB, SPX, SCTP, DCCP, STP, TFTP, RTP
мережевий	IPv4, IPv6, ICMP, IGMP, IPX, NWLink, NetBEUI, DDP, IPSec, SKIP
канальний (Ланки даних)	ARCnet, ATM, DTM, SLIP, SMDS, Ethernet, ARP, FDDI, Frame Relay, LocalTalk, Token Ring, PPP, PPPoE, StarLan, WiFi, PPTP , L2F, L2TP, PROFIBUS
фізичний	RS-232, RS-422, RS-423, RS-449, RS-485, ITU-T, RJ-11, T-carrier (T1, E1), модифікації стандарту Ethernet: 10BASE-T, 10BASE2, 10BASE5, 100BASE-TX, 100BASE-FX, 100BASE-T, 1000BASE-T, 1000BASE-TX, 1000BASE-SX

Рисунок 3.2 – Протоколи рівнів моделі OSI

UDP – протокол – це самий простий варіант, коли між клієнтом та сервером, вони більш нічого не роблять, крім того, що просто передають дані. Якщо ми глянемо для прикладу TCP-протокол, то тут вже йдуть стрілочки. Наприклад, коли ми заходимо на якийсь сайт, тут називається це рукостискання, а взагалі це називається «постукати». Тобто, коли ми хочемо до чогось підключитися, ми

стукаємося. Після цього сервер помічає, що до нього стукаються. Він в принципі тільки те й робить, що очікує, поки до нього хтось звернеться. Якщо це якийсь сайт, то сервер завжди запущений, завжди працює і він чекає, коли до нього хтось постукається і зайде. З'єднання – це коли сервер відповідає клієнту, типу сервер побачив, що до нього хтось стукається і він відправляє відповідь. Тобто ОК, я тут і ти можеш зі мною спілкуватися. Після цього, коли ми зайшли на сайт як клієнт нам уже показується стартова сторінка. Коли ми відкриваємо браузер і у нас дуже поганий Інтернет, сайт постійно крутиться – білий фон, але так і не відкривається. Це відбувається через те, що ми, коли із поганим Інтернетом зайшли на сайт, ми на сервер постукались, сервер нас побачив і він з'єднується до нас. І ми очікуємо, коли ми з'єднаємось і наш браузер з'єднується, тому що браузер розуміє, що ми постукались. Якщо у нас Інтернету немає, то ми не можемо зробити рукостискання і пишеться, що Інтернету не має, тому що у нас рукостискання не відбувається.

Сокет – це механізм, який дозволяє програмі взаємодіяти з іншими пристроями. Або пристрою взаємодіяти з пристроями через мережевий роз'єм, тобто в кожному комп'ютері є ці роз'єми за допомогою яких ми взаємодіємо. Наприклад, в Bluetooth-connection-і це буде спеціальний механізм, який відповідає за Bluetooth-з'єднання. Якщо у нас просто роутер та Інтернет, то це буде Ethernet.

Ще є таке поняття як порт. Коли ми відкриваємо Jupyter Notebook, то бачимо, що там написано localhost:8888/tree

Localhost – так називається локальна IP-адреса, це те саме, що 168.92.0.01, тобто наша власна адреса і на порту 8888 у нас вже зайнятий Jupyter Notebook. Тобто, якщо ми запускаємо якийсь інший процес, то тут вже буде стояти інший порт і ми з Python-ом взаємодіємо через браузер, використовуючи HTTPS-протоколи. Тобто це все зроблено так і ми заходимо у папки і це все через HTTPS-протокол і наш комп'ютер спілкується через HTTPS-протокол.

Сокет TCP забезпечує надійне передавання даних, а UDP створений для того, щоб це передавання було дуже швидким, тому що не має ніяких рукостискань.

Який сокет ми будемо робити, це вибираємо самі. Тобто в Python-і є модуль сокет, ми пишемо import Socket і налаштовуємо це на свої потреби.

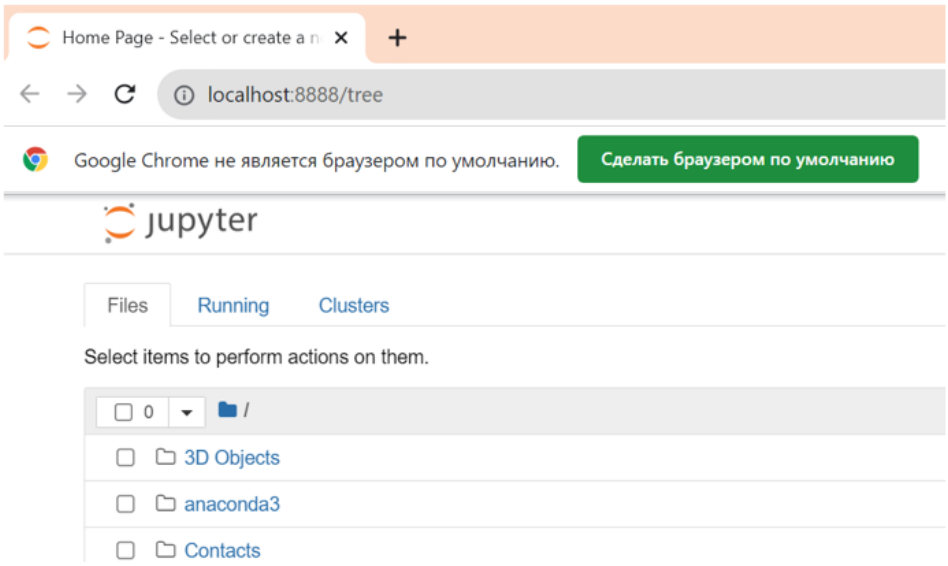

Рисунок 3.3 – localhost:8888/tree

Прикладний рівень. Всі веб-застосунки працюють по HTTP-протоколу.

На Github-і у нас був SSH-ключ. SSH-протокол працює, використовуючи цей ключ і канал відкривається тільки тоді, якщо в обох сторін є дві частини цього SSH-ключа, який згенерувався. Тобто те, що ми через консоль з'єднуємося, то з Git і з нашим комп'ютером працює через те, що в нас є SSH-ключ і SSH-ключ дозволяє створити сокет і спілкуватисся нам з Git.

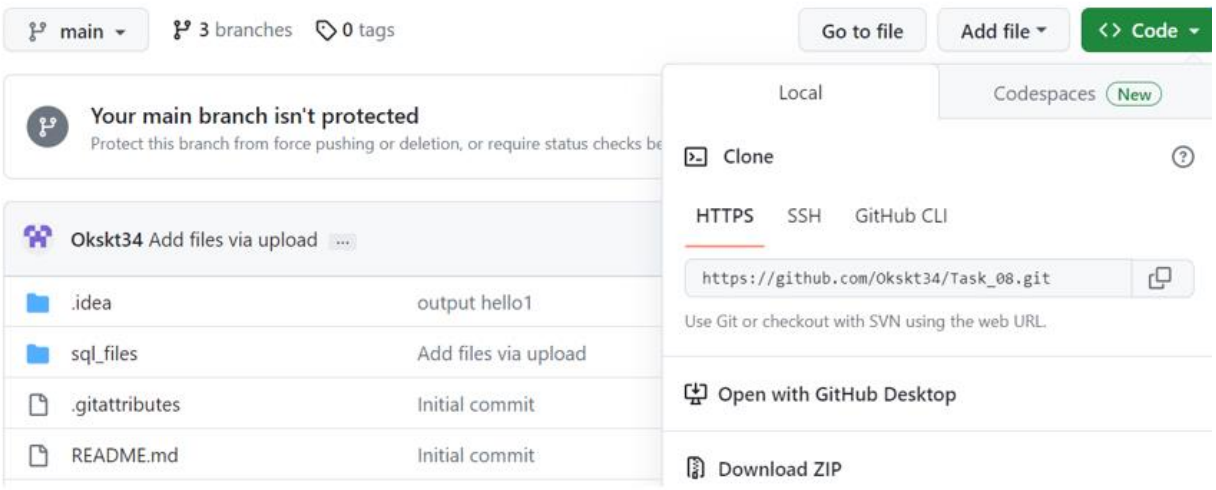

Рисунок 3.4 – SSH-протокол

IMAP – протокол, по якому працює e-mail. Коли ми листуємося - там свій протокол. FTP – для пересилання файлів. Secure Shell «безпечна оболонка» - мережевий протокол рівня застосунків, що дозволяє проводити віддалене управління комп'ютером і тунелювання TCP-з'єднань (наприклад, для передавання файлів).

Складові сокету – IP-адреса та порт. В Jupyter Notebook бачимо, що localhost – це IP-адреса, 8888 – це порт. За рахунок цього ми можемо відкрити Jupyter Notebook не на localhost-і, а на статичній IP-адресі, яку ми запитуємо у свого провайдера і ми зможемо стукатися у наш Jupyter Notebook через Інтернет, сидячи абсолютно в іншому місці. Головне, що він працює по HTTPS-протоколу і ми можемо це все налаштувати як вам потрібно.

Портів обмежена кількість на будь-якому пристрої. І там є приховані порти – ті, які ми не можемо змінювати. Системні порти 0-1023 ми не можемо змінювати. Зареєстровані – це ті, що ми можемо використовувати, 1024-49151 – ці порти можемо налаштувати і користуватися ними. Їх 48000. Динамічні та приватні порти – це внутрішні порти, які треба для внутрішньої роботи комп'ютера.

Так само сервер може і нам відправляти повідомлення. В принципі як це працює. У нас у комп'ютерах і будь в чому є елементи, які приймають дані, тобто вони приймають просто байти. Нулики і одинички вони приймають. Сокети і протоколи – це вже правила, які придумали люди, щоб ці байти, які приходять, обробляти.

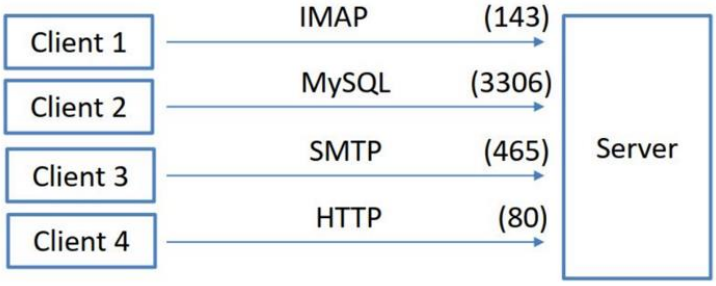

Рисунок 3.5 – Використання портів

Є сервер і ми можемо налаштувати порти для різних речей. Тобто один сервер з однією IP-адресою може тримати якийсь наш сайт (наприклад, сайт замовлень) і

так само цей сервер має базу даних, щоб швидко те, що люди клікають на сайті (людям здається, що вони клікають у себе), кожен їх клік відправляється на сервер і він у себе все це реєструє. Тобто як тільки сервер отримує інформацію в себе, що користувач клікнув, він у себе у базу даних запише і вони знаходяться на різних портах. А якщо ми хочемо щось змінювати або викликати на нашому сайті і ми пишемо порт і відповідно до цього порту підключений наш сайт і якщо ми хочемо щось з бази даних, то ми пишемо порт з бази даних і стукаємося у базу даних, щоб щось взяти, скачати, заселектити, змінити.

Бібліотека Socket надає можливості для створення і роботи з мережевими з'єднаннями за допомогою сокетів. Python має модуль socket, який дозволяє створювати сервери та клієнтські застосунки для мережевої комунікації. Сокети дозволяють програмістам створювати мережеві додатки, такі як web-сервери, чат-клієнти, мережеві ігри та інші, які можуть обмінюватися даними з іншими комп'ютерами через мережу.

Таблиця 3.1 – Методи бібліотеки Socket

№ п/п	Метод	Призначення
1	socket.bind (address)	Прив'язує сокет до адреси *address* (ініціалізує IP-адресу та порт). Сокет не повинен бути прив'язаний до цього
2	socket.listen ([backlog])	Переводить сервер у режим прийому з'єднань. Пареметр *backlog (int)* – кількість з'єднань, які прийматиме сервер
3	socket.accept ()	Приймає з'єднання та блокує програму в очікуванні повідомлення від клієнта. В результаті повертає кортеж: o *conn*: об'єкт з'єднання (сокет), який можна використовувати для надсилання / отримання даних; o *address:* адреса клієнта
4	socket.recv (bufsize [, flags])	Читає та повертає дані у двійковому форматі (набір байтів) із сокету. *Bufsize (int)* – максимальна кількість байтів в одному повідомленні
5	socket.send (bytes [, flags])	Надсилає дані клієнту і повертає кількість відправлених байт. Параметр *bytes (bytes)* – двійкові дані
6	socket.close ()	Закриває сокет

3.2 Використання методів протоколу HTTP

У нас, коли ми спілкуємося з сайтами або будь з чим, є деякі правила, по яким все це відбувається. Якщо ми візьмемо HTTP-протокол, у нього є методи, по яким він працює. І перший метод HTTP-протоколу – це GET. GET-метод використовується для отримання даних із сервера. Тобто, коли ми пишемо GET, у нас буде метод, ми його викликаємо і він у нас відправляється по нашому каналу до тієї адреси, в яку ми написали (в HTTP протоколу є адреса, яку ми написали й можемо туди піти) і коли сервер приймає наш запит, він бачить, що це GET і він відповідно реагує. Через метод GET ми не можемо нічого відправити. Метод GET просто для того, щоб постукатися.

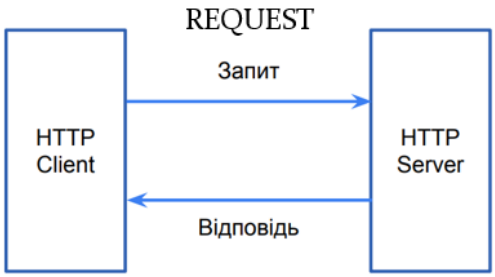

Рисунок 3.6 – Протокол HTTP (протокол прикладного рівня / клієнт-сервер / ідентифікація ресурсів)

POST використовується для надсилання даних на сервер. Тобто, коли ми робимо POST – це значить, що окрім того, що ми стукаємося, ми ще серверу хочемо щось прислати і сервер буде бачити, що це метод POST і він буде відповідно обробляти так як у нього прописано обробляти POST-методи.

Так само є PUT – метод для оновлення існуючих даних на сервері. Також є DELETE – це для видалення даних з сервера.

PATCH – це коли відправляємо якийсь шматочок для того, щоб оновити.

REQUEST – це запит, який відправляється від клієнта до сервера за допомогою протоколу HTTP. Запит може містити різні параметри – вони передаються або в тілі, або в заголовку. REQUEST повинен містити один із методів (GET, PUT,

DELETE) і він має містити URL-адресу, по якій він хоче звернутися. Якщо ми говоримо про HTTP-протокол, то REQUEST має метод REST.

Рисунок 3.7 – Формат протоколу HTTP

Модель OSI, протокол HTTP – не прив'язані ні до Python, ні до мови програмування – це просто як між собою працюють пристрої, використовуючи мережі. На бекенді налаштовують тільки HTTP або SSH, максимум.

Таблиця 3.2 – Методи HTTP

№ п/п	Назва методу	Призначення
1	GET	Отримати наявний ресурс
2	POST	Створити новий ресурс
3	PUT	Оновити існуючий ресурс
4	PATCH	Часткове оновлення існуючого ресурсу
5	DELETE	Видалити ресурс

REST – коли ми заходимо на будь-який сайт, у нас все скоріше за все відображається у правильному варіанті, хоча нам відправляється набір байтів, а чи він прийде, а як він прийде – це абсолютно не гарантовано. А REST – це архітектура, яка дозволяє спілкуватися по HTTP-протоколу таким чином, щоб все правильно і гарно виходило.

Таблиця 3.3 – Архітектурні обмеження REST

№ п/п	Архітектурні обмеження REST
1	Підтримка клієнт-сервер
2	Відсутність
3	Кешування даних
4	Однорідний інтерфейс
5	Шари абстракції
6	Запитування коду

Коли ми з клієнта заходимо на якийсь веб-сервер, то ми стукаємося до нього. Тобто наш браузер не просто переходить по посиланню. Наш браузер під капотом робить багато речей, які ми не бачимо. Тобто перше, що він робить – це стукається на сервер. Тільки тоді, коли він отримує фідбек, він для вас має два варіанти подій. Перше – він покаже, що сторінка не доступна, якщо він не достукався; він покаже, що відсутній Інтернет, якщо він не зможе стукатися і він буде очікувати (крутити колесо), показувати нам білий екран, тому що він розуміє, що нам від сервера прийшов REQUEST, тобто ми постукалися, сервер (сервер – це просто програма, яка очікує поки до неї будуть стукатися і вона відправила нам, тому що коли ми заходимо на сайт – сайт бачить, хто до нього зайшов). Вона відправила вам «привіт», мінімальну кількість байтів для того, щоб показати, що ми спілкуємося зараз з клієнтом. Якщо браузер це побачив, то він буде очікувати поки сервер після того, як «помахав рукою» не відправить всю ту кількість байтів, щоб завантажити сайт.

У нас є SQL – мова запиту для баз даних. І коли ми заходимо на якийсь сайт, у нас там просять ввести логін та пароль. Так от: логіни і паролі зберігаються у базі даних і коли ми заповнюємо ці дані, потім ці дані вставляться в SQL-запит, воно там буде щось дивитися і нам поверне результат. Так от, якщо веб-сервер не з’єднувати ще з якимось додатковим одним сервером, то у рядок логіну можна буде написати SQL-запит, тобто ці хакери просто сидять і вивчають (ну вони

здогадуються, тому що програмісти не дуже з великою фантазією по придумуванню імен) і починають підбирати всякі імена, наприклад, user_name, user_password і вони можуть угадати назви полів і сформувати такий запит, який виконає сервер і поверне йому результат, тобто так і виходить часто злив баз даних. Це просто хтось написав правильний SQL-підзапит, там де має бути просто якесь слово і веб-сервіс просто повернув (відповів) результат, виконав програму, програма виконалася із SQL-ін'єкцією (це так називається) і повертається те на що розраховували. Для цього до серверів підключають інші сервери, які виконують свої задачі: перевірку чи ще щось.

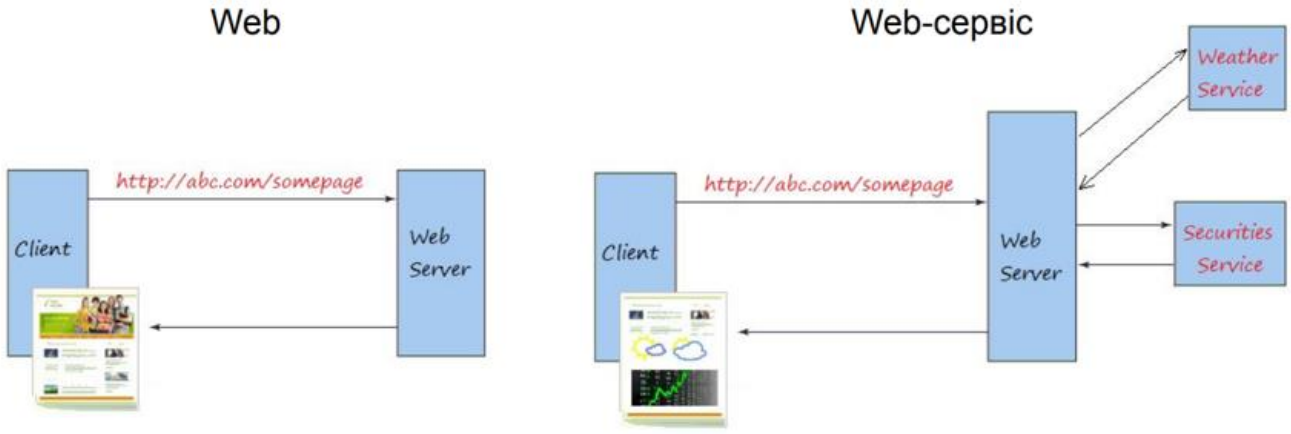

Рисунок 3.8 – Web і Web-сервіс

Іноді, якщо ми попадемо у компанію, яка не просто там займається аналітикою, а яка пише код і продає його клієнтам, то ми можемо стикнутися з мікросервісною архітектурою. Можлива наступна ситуація. Замість того, щоб код писати одним проєктом, якщо код буде дуже великим – він стане дуже заплутаним, то люди розробляють мікросервісну архітектуру. Наприклад, вони домоволяються, сервер 1 у нас відповідає за нашу сторінку, як ми в Інтернеті дивимося і т.д. Сервер 2 відповідає за базу даних. Сервер 3 відповідає за ML. Мікросервісна архітектура зроблена таким чином, що у нас об'єкти нашого коду розписані як окремі сервери, тобто це все три окремі програми. Плюс для кожного із цих серверів описані правила, з якими вони взаємодіють і наш сайт напряму спілкується з базою даних (наприклад, каталог товарів ми беремо з бази даних). А

коли наш користувач хоче щось купити, наш сайт відправляє у базу даних, що такий то товар треба відняти і такого то користувача треба добавити. І наша ML-модель може спілкуватися з базою даних. Але нам не треба, щоб сайт спілкувався з ML-моделлю, тому що це різні речі і вони не пов'язані. Ми робимо якісь висновки, заносимо це у базу даних. Сайт бере дані з бази даних. Але самі між собою вони не спілкуються. Для таких випадків придумали мікросерверну архітектуру, де люди сконцентровані на тому як ось такі правила узгодити і зробити, щоб це все працювало як така схема. І зазвичай люди тут працюють різні, різні команди іноді. І вони просто говорять: ми добавили нову функцію і тепер до нашого сервера ви можете звертатися так то; ок – ми тепер будемо так робити. І ще вони часто не знають про роботу один одного. Якщо компанія велика, то бажано не знати, що роблять інші.

3.2 Практична реалізація алгоритму сортування даних

Створимо два пустих файли, які назвемо: *клієнт* і *сервер*. Напишемо клієнта та сервера. Значить, що нам треба знати, щоб це писати. Спочатку треба буде заінсталити та заімпортити Flask – мініфреймворк, який дозволяє працювати з сокетами через HTTP-архітектуру. Крім Flask нам треба jsonify та requests. Заімпортили перший рядок. Далі, коли ми бачимо @ і щось написано – це називається декоратор. Декоратор – це коли ми кажемо Python-у, що це функція не проста, а особлива і її треба трохи по іншому обробляти. Такий декоратор знає, що нижче функція буде у нас щось робити на нашому сервері. Конкретно ця анотація означає, що ми цій функції маємо вказати шлях (наприклад, це буде '/sort' (сортування)) і нам треба написати, який метод HTTPS-протоколу буде підходящим для цієї функції. Напишемо функцію, наприклад, яка буде сортувати числа. Якщо це функція, яка сортує числа, значить вона має їх прийняти. Тобто клієнт має щось відправити серверу, значить це метод POST. Тепер ми пишемо сам sort. Тепер нам треба взяти дані. Кожна така функція (route()) оскільки це

POST, у неї є request. Тобто такі функції з таким декоратором щось приймають. І це можна отримати просто як request.json. *json* – це формат, в якому можна передавати будь-що структуровано, наприклад, масиви або словники можна передавати у форматі json – це просто правило, як вони там записані через коми і фігурні дужки. Тепер ми пишемо data.sort() - використаємо внутрішнє і return jsonify(data). Після цього ми можемо написати app.run().

```
from flask import Flask, jsonify, request
```

```
app = Flask(__name__)

@app.route('/sort', methods = ['POST'])
def sort():
    data = request.json
    data.sort()
    return jsonify(data)

app.run()
```

```
 * Serving Flask app '__main__'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
```

Запустили сервер. * - тут – значить це працює. Бачимо, що наш сервер запустився по вищевказаній IP-адресі і з портом 5000.

Заходимо на клієнта і дивимося, що має бути на клієнті. Перше, що потрібно нам зробити – це import requests. І ми можемо написати нашу функцію. Дивимося, що ми маємо зробити в цій функції. Нам треба постукатися до цієї функції і отримати від неї відповідь. Якщо requests – це відправити, то відповідь – це response (так просто називаємо змінну) і оскільки це POST-метод, то ми маємо не тільки постукати, а й звернутися, то ми пишемо, що це post, а в дужках – ми передаємо адресу (URL) і оскільки ми написали sort (це у нас іде по принципу шляху у файловій системі), то у клієнті ми через слеж пишемо sort, тому що на сервері ми написали шлях до цього методу def sort() – цей метод виконується і він знаходиться по цьому шляху. Наша IP-адреса + порт, тобто наш сокет і шлях далі до нього sort. Ми ще повинні відправити йому наші дані: json = numbers. Можна зразу це заретурнить, але у responce є параметр ok (це таке поле у екземпляра цього класу), яке повертає True або False, якщо хорошо чи ні. Ми повертаємо response.json(). Else ми не пишемо, тому що якщо ми зайшли сюди, то Returne

None ми вже не будемо бачити. Тепер викличемо sort_numbers. Ми отримали відсортований масив.

```
import requests
```

```
def sort_numbers(numbers):
    responce = requests.post('http://127.0.0.1:5000/sort', json = numbers)
    if responce.ok:
        return responce.json()
    return None
```

```
sort_numbers([1,4,2,5,6])
```

```
[1, 2, 4, 5, 6]
```

Ми викликали метод – sort_numbers. Цей метод через requests робить метод post до цього сокета – http://127... (оскільки це post і ми маємо щось передати), то ми в json-і передаємо оці numbers-и. Далі сервер їх постійно приймає і очікує, що до нього звернуться.

```
from flask import Flask, jsonify, request
```

```
app = Flask(__name__)

@app.route('/sort', methods = ['POST'])
def sort():
    data = request.json
    data.sort()
    return jsonify(data)

app.run()
```

```
 * Serving Flask app '__main__'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
127.0.0.1 - - [05/Nov/2023 16:23:42] "POST /sort HTTP/1.1" 200 -
```

І тут пишеться історія, хто до нас стукався. 127.0.0.1 – це є ми і ось пишеться, що до нас ми постукалися і просили POST-метод в директорії по шляху sort.

flask – фреймворк, тобто це така вже готова бібліотека через яку реалізовані ці всі процедури, як серверу спілкуватися з клієнтом. flask (з маленької) – це бібліотека. Flask – це клас. Flask(__name__) – ми так викликаємо конструктор. App – це наш application. Тобто, коли ми пишемо назву класу, а потім дужки, то в нас викликається inite, new і створюється новий екземпляр класу. App – це наш Application, тобто наш веб-застосунок, це екземпляр класу Flask. У класі Flask прописані всі ці речі як мають працювати методи, як спілкується між собою клієнт і сервер.

По HTTP-протоколам всі дані передаються через json. Тобто, якщо ми відправляємо масив, то ми насправді відправляємо не масив, а щось типу такого:

```
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 27,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    }
  ],
  "children": [],
  "spouse": null
}
```

Тобто це правила, по яким записуються будь-які дані. Просто ми перетворили наш масив і він записаний в іншому синтаксисі. Ми ж відправляємо набір байтів. У нас сервер написаний на якійсь мові (він може бути написаний навіть на java, в принципі він може бути написаний на чому завгодно), а ми на Python і хочемо до нього постукатися і ці всі правила придумані для того, щоб ми опустили синтаксис мов програмування, опустили все і просто домовилися на рівні байтів, як ми будемо оці всі дані передавати і розшифровувати. І просто домовилися, що для чого б там не було, в HTTPS-протоколах наші методи будуть приймати байти і розшифровувати їх або зашифровувати через *json.*

Ми викликаємо requests – це як запит. Сервер написаний на Flask, але знову ж таки нас не хвилює на чому написаний цей сервер. Ми – клієнт. Нам не треба встановлювати бібліотеки, які стоять на сервері. У нас є просто requests – це бібліотека, яка знає як звернутися до заліза на компах. Вона через оці наші спеціальні порти може стукатись. Комп’ютер, знаючи числа, може постукатися. Але, якщо ми візьмемо requests і вкажемо, що це post, тобто ми відправляємо на якийсь сервер щось: ми вказуємо сервер, на який відправляємо і вказуємо, що саме ми відправляємо. Відповідь ми зберігаємо в responce і ретурнимо response.json(), тобто те що ми отримали. Комп’ютер знає, куди йому стукатися.

Нам треба зупинити процес. Зупиняємо сервер (натискаємо на чорний квадрат) і він не працює.

В якості ще одного прикладу напишемо Фібоначчі.

```
from flask import Flask, jsonify, request
```

```
app = Flask(__name__)

@app.route('/sort', methods = ['POST'])
def sort():
    data = request.json
    data.sort()
    return jsonify(data)

@app.route('/fib', methods = ['POST'])
def fib():
    data = request.json
    res = [0,1]
    while len(res) < data:
        res.append(res[-1] + res[-2])
    return jsonify(res)

app.run()
```

```
 * Serving Flask app '__main__'
 * Debug mode: off
```

```
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
127.0.0.1 - - [05/Nov/2023 16:47:00] "POST /fib HTTP/1.1" 200 -
```

Сервер запустили. Тепер заходимо на Клієнта. Спробуємо постукатися. Можемо зразу тут стукатися: request.post(відправляємо нашу адресу), тільки тепер ми змінюємо шлях на fib, тому що ми встановили fib і в json-і у нас просто має прийняти одне число, ну нехай буде 8:

```
import requests
```

```
def sort_numbers(numbers):
    responce = requests.post('http://127.0.0.1:5000/sort', json = numbers)
    if responce.ok:
        return responce.json()
    return None
```

```
requests.post('http://127.0.0.1:5000/fib', json = 8).json()
```

```
[0, 1, 1, 2, 3, 5, 8, 13]
```

Ми постукалися і отримали 8 перших чисел Фібоначчі. Якщо 20, отримали перші 20 чисел:

```
requests.post('http://127.0.0.1:5000/fib', json = 20).json()
```

```
[0,
 1,
 1,
 2,
 3,
 5,
 8,
 13,
 21,
 34,
 55,
 89,
 144,
 233,
 377,
 610,
 987,
 1597,
 2584,
 4181]
```

При тому, що клієнта абсолютно не хвилює, яка реалізація, тому, коли це пишеться, часто може таке бути, що ML написаний на Python-і, а HTTP написаний на Java. Сайт запущений на Java, все працює на Java (I – HTTP:// Java; II – база даних; III – ML Python). Умовно вони тут не спілкуються, але таким чином, якщо ми пишемо мікросервісну архітектуру, то кожен сервіс може бути написаний на різній мові. І головне тільки – узгодити правила, по яким вони обмінюються байтами. Всі мікросервіси можуть знаходитися на одному пристрої, на одному глобальному сервісі. Найчастіше це все через Amazon роблять. Але кожен сервер написаний на окремій мові, тому що для чогось одна мова краща, а для чогось інша. Для ML – поки що найкращий Python. І вони просто узгоджують правила по яким передаються дані. Це будуть json-формати, скоріше за все. Іноді, якщо дані якісь великі специфічні, то можна придумати й інші способи, як приймаються дані. Це все налаштовується, наприклад, є Yaml – тут вже не має фігурних дужок, а просто йдуть відступи як у Python. І ми можемо налаштувати, щоб ваші дані спілкувалися через такий формат. В принципі це не має різниці, просто домовленість. І через Yaml можливо буде трохи швидше працювати, тому що не має фігурних дужок, а це також байти.

Потушимо наш сервер (в Jupyter Notebook натискаємо на чорний квадрат). Проекспериментуємо інший метод.

```
from flask import Flask, jsonify, request
```

```
app = Flask(__name__)

@app.route('/sort', methods = ['POST'])
def sort():
    data = request.json
    data.sort()
    return jsonify(data)

@app.route('/fib', methods = ['POST'])
def fib():
    data = request.json
    res = [0,1]
    while len(res) < data:
        res.append(res[-1] + res[-2])
    return jsonify(res)

app.run()
```

```
 * Serving Flask app '__main__'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
127.0.0.1 - - [05/Nov/2023 16:47:00] "POST /fib HTTP/1.1" 200 -
127.0.0.1 - - [05/Nov/2023 16:49:17] "POST /fib HTTP/1.1" 200 -
```

Напишемо метод GET. Робимо аналогічну процедуру. Зробимо кореневу папку, не будемо писати ніяких директорій. І напишемо GET метод.

```
from flask import Flask, jsonify, request
```

```
app = Flask(__name__)

@app.route('/sort', methods = ['POST'])
def sort():
    data = request.json
    data.sort()
    return jsonify(data)

@app.route('/fib', methods = ['POST'])
def fib():
    data = request.json
    res = [0,1]
    while len(res) < data:
        res.append(res[-1] + res[-2])
    return jsonify(res)

@app.route('/', methods = ['GET'])
def methods_info():
    methods = {
        'SORTING':'POST /sort',
        'FIBONACCI':'POST /fib'
    }
    return jsonify(methods)

app.run()
```

```
 * Serving Flask app '__main__'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
```

Працює. Спробуємо постукатися на клієнті. В GET-і ми маємо передати просто адресу, тому що GET більше нічого під собою не несе.

```
requests.post('http://127.0.0.1:5000/fib', json = 20).json()
```

```
[0,
 1,
 1,
 2,
 3,
 5,
 8,
 13,
 21,
 34,
 55,
 89,
 144,
 233,
 377,
 610,
 987,
 1597,
 2584,
 4181]
```

```
requests.get('http://127.0.0.1:5000').json()
```

```
{'FIBONACCI': 'POST /fib', 'SORTING': 'POST /sort'}
```

І ми постукалися через GET. Через метод get ми йому нічого не передаємо, просто стукаємося по адресу і вказуємо що це метод get. І ми отримали результат, як ми можемо звернутися типу якщо Фібоначчі хочемо, то маємо викликати POST і вказати fib, якщо sorting, то post і sort. Через *requests.get* можна пробувати стукатися на всякі різні сайти і можливо на якомусь сайті, якщо ми дізнаємося теоретично IP і порт, на якому це знаходиться, то ми зможемо спробувати постукатися і щось з цього отримати. Тобто це не тільки працює з нашим комп'ютером, а з будь-чим в чого є адреса.

Ця тема дуже тісно пов'язана з архітектурою, хто як домовився, хто як придумав. Якщо нам пощастить працювати в компанії, де буде мікросервісна архітектура, то нам прийдеться щось із цього робити. На роботі вам дадуть список типу: хочеш отримати те, виклич такий то метод, постукайся сюди, передай те то і ти отримаєш те, що тобі треба, просто напишете те, що вам скажуть.

Коли ми напишемо код і захочемо, щоб він запускався кожен день в 10:00. Оскільки flask – це програма, яка робить вічно і чогось очікує, то у неї є ньюанси, тобто можна написати якісь рядки (декоратори) і можна буде вказати час, коли запустити. І функції, які будуть під цими декораторами будуть запускатися самі по собі на сервері. І ми можемо заімпортити будь-що, запустити його як Flask-файл і application буде запускати цю програму кожен день. Наприклад, типу зібрати якісь дані. Можна скачати програму Windows Scheduler, написати шлях до

програми і вказати години, коли ми хочемо, щоб запускалася програма. Або можна подивитися в Google, як це робити через Flask і запускати через flask те, що ми хочемо кожен день вранці.

3.4 Розробка рекомендацій для організації навчання та досліджень в області Data Science

Data Science у бізнесі можна розподілити на дві категорії: мале виробництво; середнє та велике виробництво. Мале виробництво – є якийсь завод, він щось виробляє і їм потрібна автоматизована статистика. Завод хоче щомісяця автоматизований звіт, щоб це був якийсь сервер, головний бухгалтер завантажував туди якісь дані про квартальні звіти, а потім йшла вже аналітика або навіть прогнозування. Це такий приклад використання Python.

Рисунок 3.9 - Вигляд Data Science у бізнесі

Середні та великі компанії в Україні – аутсорсингові або аутстафінгові. Дуже мало у нас є продуктових компаній, але вони є. Коли замовник хоче складне прогнозування для своєї компанії – це вже складний стек типу Tensorflow, Amazon, Python.

Аутсорсинг – це практика найму зовнішніх фахівців або компаній для виконання певних завдань або функцій, які раніше виконувалися в межах власної організації. Ця стратегія дозволяє компаніям зосередитися на своєму основному бізнесі, передаючи інші завдання або функції зовнішнім постачальникам, які спеціалізуються на цих областях. Основні переваги аутсорсингу включають

зниження витрат, збільшення ефективності, доступ до спеціалізованих знань та ресурсів, зменшення ризиків і більша гнучкість у реагуванні на зміни на ринку. Популярними областями для аутсорсингу є IT-послуги, бухгалтерія, клієнтська підтримка, логістика, виробництво. Аутсорсинг може бути як локальним, так і міжнародним, залежно від обсягу та складності завдань, а також від місця розташування зовнішніх постачальників.

Рисунок 3.10 – Аутсорсинг

Рисунок 3.11 – Аутстафінг

Аутстафінг (англ. outstaffing) – це форма організації робочого процесу, при якій компанія наймає зовнішнього постачальника (аутстафінгову компанія), яка надає робочу силу, яка працює під керівництвом клієнта. В цьому контексті аутстафінг відрізняється від аутсорсингу тим, що не передбачає передавання

завдань або функцій, але надає тимчасовий персонал, який фактично працює для компанії-клієнта, але має статус співробітника аутстафінгової компанії. Головна перевага аутстафінгу полягає в тому, що він дозволяє компаніям швидко і легко залучати та звільняти робочу силу в залежності від потреб, без зобов'язання до довготермінового найму і управління працівниками. Це особливо корисно для проєктів, які потребують додаткового персоналу на обмежений термін, а також для компаній, які не хочуть брати на себе додаткові адміністративні обов'язки і витрати, пов'язані з наймом нових співробітників. Аутстафінг може бути вигідним для обох сторін – клієнта і аутстафінгової компанії, оскільки він надає гнучкість і доступ до спеціалізованого персоналу.

Можна переводити свої дослідження у Excel, щоб клієнт міг сам робити аналіз.

Узагальнюючи вищевикладені дослідження, можна представити навики, якими має володіти спеціаліст в області Data Science (рис.3.12).

Рисунок 3.12 – Навики спеціаліста Data Science

Жовте коло – коли приходить клієнт і описує свою проблему, нам потрібно абстрагуватися і перевести це все у математичну площину, тобто зрозуміти. Наприклад, клієнт може сказати: я хочу знати, який стиль меблів я продаю. Тоді ми вже думаємо: це задача класифікації, нам потрібний набір даних зі стилями, нам потрібно побудувати штучну нейронну мережу, ми це все робимо і вирішуємо для клієнта завдання.

Червоне коло – у всьому IT дуже потрібні навики, а саме у Data Science потрібно нашими очима продивитися сотні або десятки статей або чого небудь іншого, пов'язаного з якимись дослідженнями і це все потрібно зробити навіть не за тиждень, а за день, тому що нам говорять, що вже завтра буде дзвінок з клієнтом і потрібно клієнту розповісти, яка в нас є експертиза і взагалі чи можна вирішити його питання чи ні. Це все потребує дуже багато уваги та вміння: швидко аналізувати та читати інформацію.

Синє коло – тобто коли ми на Python, у Keras чи Tensorflow зробили штучну нейронну мережу, натренували, це все добре, але після цього нам потрібно це інтегрувати у бізнес-середовище клієнта. У різних компаніях цим можуть займатися різні люди або це все може робити одна людина. Таких спеціалістів можна називати fullstack Data Scientist, тобто людина, яка не тільки розробляє якісь математичні моделі, а ще й має навики та розбирається у бекенді і робить це все.

Зелене коло – важливе, тому що замовники – люди, які не розбираються у математиці або програмуванні, але є замовники, які приходять і починають розказувати, як вам писати код. Це не дуже добре. Є замовники, які не розуміють, як це все працює і ми їм говоримо, що це все штучна нейронна мережа, генеративно-змагальна нейромережа, яка переводить стилі з однієї матриці в іншу, там є генератор та дискримінатор та ін. Айтішник не у всіх випадках це програміст. DOU – крутий веб-сайт, на якому є багато вакансій та статистичних досліджень.

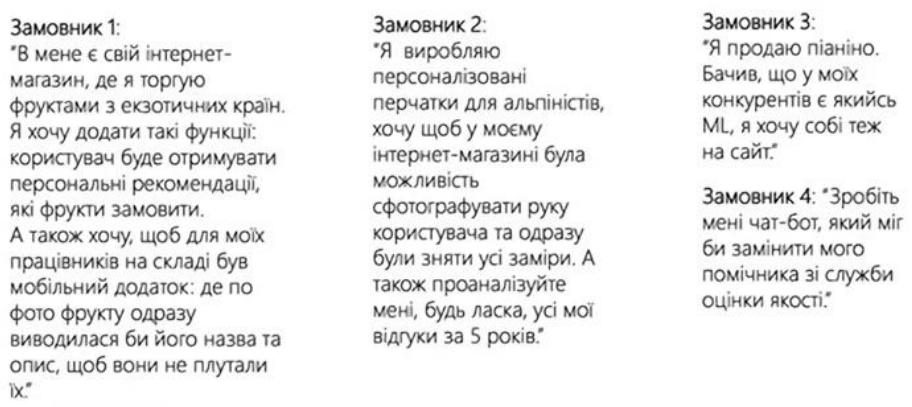

Рисунок 3.13 – Приклад типового замовлення у Data Scientist

Наприклад, першу задачу замовника ми розбиваємо на підзадачі. Що це класифікація фруктів, де дістати data set фруктів. Наприклад, мобільний додаток робить працівник аутсорсу, тобто компанія продає двох спеціалістів: Data Science та мобільного розробника.

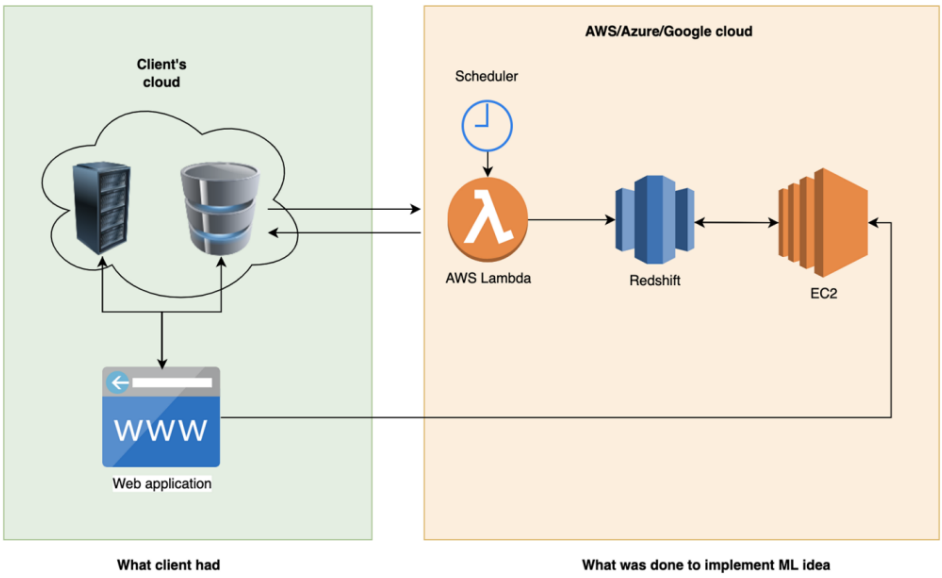

Рисунок 3.14 – Реалізація ML у бізнесі

На основі проведеного аналізу, мною визначені рекомендації для організації навчання та досліджень в області Data Science.

Таблиця 3.4 – Рекомендації: як побудувати своє навчання

№ п/п	Дії	Приклад
1	Визначитися, що вам подобається в IT	Це бекенд або Data Science. Аналітика або виконання чітких задач
2	Вивчити мову програмування	Опанувати Python. Вивчити основи мови від «що таке цикл» до ОПП та побудови власних модулів
3	Вивчити фреймворки	В залежності від першого рішення вивчити хоча б 1-2 фреймворки
4	Зробити резюме та почати пошук роботи	Проаналізувати вакансії, що потрібно від кандидатів, зробити декілька варіантів резюме, відіслати їх, очікувати співбесіди

Таблиця 3.5 – Рекомендації по навчанню: Python

№ п/п	Дії	Ресурси
1	Основи мови. Як працювати у IDE, як працює Python. Базові структури даних та синтаксичні конструкції	http://www.learnpython.org/en/Welcome - усі основи
2	Основи ООП, модулі та написання власних об'ємних програм	http://realpython.com/python3-object-oriented-programming/ - про ООП
3	Закріплення знань шляхом вирішення різних задач	http://leetcode.com/problemset/all/

http://perso.limsi.fr/pointal/

http://media/python:cours:mementopython3-english.pdf

Таблиця 3.6 - Рекомендації по навчанню: Python Backend

№ п/п	Дії	Ресурси
1	Вивчити основи як працює Інтернет (клієнт-сервер взаємодія, HTTP(S)). Вивчити HTML	An overview of HTTP - HTTP | MDN (mozilla.org) (що таке HTTP) How does the Internet work? - Learn web development | MDN (mozilla.org) (як працює Інтернет) HTML Basic (w3schools.com) (HTML основи)
2	Вивчити фреймворк. Зробити «по інструкції» будь-який веб-сервіс	Welcome to Flask — Flask Documentation (2.1.x) (palletsprojects.com) (Flask) The web framework for perfectionists with deadlines | Django (djangoproject.com) (Django)
3	Розробити власний веб-сервіс. Щоб точно можна було вважати себе розробником, то бонусом (для себе звісно) задеплоїти його (розмістити на хостингу)	Cloud Application Platform | Heroku (Heroku – безкоштовний хостинг)

Таблиця 3.7 - Рекомендації по навчанню та дослідженням в області Data Science

№ п/п	Дії	Ресурси
1	Вивчити основні фреймворки для роботи з даними. Вивчити математичну базу.	https://numpy.org/ https://pandas.pydata.org/ https://matplotlib.org/ https://scipy.org/
2	Вивчити класичні моделі машинного навчання та фреймворків. Математична база.	https://scikit-learn.org/stable/
3	Вивчення фреймворків для глибинного навчання. Математична база.	https://keras.io/ https://pytorch.org/ https://www.tensorflow.org/

Pandas: Pandas_Cheat_Sheet.pdf (pydata.org)

NumPy: Cheat sheet Numpy Python copy.indd

Машинне навчання: Best 15+ Machine Learning Cheat Sheets to Pin to Your Toilet Wall - Be on the Right Side of Change (finxter.com)

Спеціалістам в області Data Science необхідно мати глибокі знання в різних математичних галузях, оскільки математика є основою багатьох алгоритмів та методів, що використовуються в аналізі даних. Математичні аспекти, які повинні бути відомі спеціалісту в Data Science:

1. Статистика. Розуміння основних понять, таких як дисперсія, кореляція, регресія та розподіли ймовірностей, які важливі для аналізу даних та прийняття статистичних рішень.

2. Лінійна алгебра. Робота з матрицями та векторами важлива для багатьох методів машинного навчання, включаючи методи зменшення розмірності та рекомендаційні системи.

3. Оптимізація. Знання методів оптимізації, корисних для налаштування параметрів моделей машинного навчання та підбору гіперпараметрів.

4. Диференційна математика. Розуміння понять диференціювання та інтегрування корисно для роботи з градієнтами та оптимізацією у глибокому навчанні (нейронних мережах).

5. Теорія ймовірності. Знання теорії ймовірності важлива для роботи з ймовірнісними моделями, байєсівськими методами та багатьма іншими аспектами аналізу даних.

6. Теорія графів. Розуміння графової теорії корисне для аналізу мереж та структурованих даних.

7. Математична статистика. Глибше розуміння теоретичних аспектів статистики корисне для розробки нових методів та аналізу даних.

8. Теорія інформації. Знання понять, таких як ентропія і взаємна інформація, важливе для роботи з великими обсягами даних та алгоритмами обробки інформації.

Знання цих математичних концепцій допоможе спеціалістам в Data Science аналізувати дані, розробляти та вдосконалювати моделі машинного навчання, виконувати статистичні тести та приймати обгрунтовані рішення на основі даних.

Наука про дані є інтердисциплінарною галуззю, яка поєднує знання із статистики, програмування, машинного навчання, обробки сигналів, інформатики та інших галузей з метою аналізу, витягування інсайтів та прийняття рішень на основі даних. Основна особливість науки про дані полягає в її здатності працювати з великими обсягами даних, що може бути надзвичайно корисним для різних галузей. Підсумовуючи вище проведені дослідження узагальнимо особливості науки про дані:

- Обробка великих обсягів даних. Data Science здатна працювати з великими масивами даних, що не завжди може бути виконано іншими методами;
- Використання статистичних методів. Наука про дані використовує статистичні методи для аналізу даних, знаходження закономірностей і робить прогнози на їх основі;
- Машинне навчання. Data Science використовує методи машинного навчання для побудови моделей, які можуть робити передбачення на основі даних;

- Візуалізація даних. Важливою частиною Data Science є візуалізація даних, що допомагає зрозуміти складні залежності і структуру даних;
- Робота з різними джерелами даних. Data Science дозволяє обробляти дані з різних джерел, включаючи тексти, зображення, аудіо, сенсорні дані та інше;
- Застосування в різних галузях. Наука про дані знаходить застосування в багатьох галузях, включаючи медицину, маркетинг, науку про клімат, транспорт, технології тощо;
- Постійний розвиток. Data Science постійно розвивається, оскільки нові методи і технології виникають для роботи з даними.

Загалом, Data Science стала важливою галуззю, оскільки дозволяє отримувати цінні знання з даних, що допомагає вирішувати проблеми і приймати рішення у багатьох сферах життя. Вивчення науки про дані має чисельні переваги, які роблять її важливою галуззю для розв'язання різних завдань і вирішення проблем у різних галузях. Переваги вивчення науки про дані:

- Покращення прийняття рішень. Data Science допомагає приймати рішення на основі об'єктивних даних, зменшуючи вплив суб'єктивного біасу та інтуїції. Це особливо корисно у сферах бізнесу, медицини, фінансів тощо, де точність рішень може бути критично важливою.
- Виявлення та розуміння закономірностей. Data Science дозволяє виявляти складні зв'язки та закономірності в даних, що може призвести до нових відкриттів та інсайтів.
- Прогнозування та передбачення. Data Science дозволяє побудувати моделі для прогнозування майбутніх подій і результатів на основі історичних даних.
- Оптимізація бізнес-процесів. Data Science допомагає вдосконалювати бізнес-процеси шляхом аналізу даних, виявлення слабких місць і підвищення ефективності.
- Зростання конкурентноспроможності. Компанії та організації, які використовують Data Science для аналізу даних, мають перевагу над конкурентами в умовах ринкової конкуренції.

- Покращення медичної діагностики. Використання Data Science вмедицині допомагає виявляти ранні ознаки захворювань та розробляти більш точні методи діагностики.
- Підвищення якості послуг. Data Science допомагає компаніям аналізувати відгуки та поведінку клієнтів для покращення якості обслуговування та розвитку нових продуктів і послуг.
- Розробка інновацій. Data Science стимулює розробку нових технологій та підходів, що веде до інновацій та розвитку нових галузей.

Загалом, вивчення науки про дані сприяє покращенню якості прийняття рішень, розв’язанню проблем, оптимізації процесів та створенню нових можливостей у багатьох сферах діяльності.

ВИСНОВКИ

У результаті проведених досліджень визначено, що основними методами обробки даних є: збір даних, їх очищення, трансформація, витягування даних, агрегація, візуалізація, аналіз та зберігання даних. Ці методи можуть використовуватися як окремо, так і в поєднанні, в залежності від конкретних завдань і об'єму даних. Технології обробки даних широко розповсюджені і розвиваються швидко. Найважливішими технологіями з обробки даних є: бази даних; Big Data технології; хмарні обчислення; технології машинного навчання (ML) та штучного інтелекту (AI), які використовуються для автоматизації обробки та аналізу даних, створення прогнозів та роботи з навчанням даних; технології Інтернету речей (IoT); технології обробки природної мови (NLP); велика аналітика даних. Ці технології спільно використовуються для обробки та аналізу даних у різних сферах бізнесу і досліджень. Вибір конкретної технології залежить від потреб та вимог проєкту.

Бібліотека NumPy надає широкий вибір математичних функцій та операцій для числових обчислень, добре працює з великими обсягами даних та дозволяє виконувати обчислення на багатьох ядрах процесора, що підвищує ефективність обробки даних. NumPy використовує мову C під капотом для обчислень, що робить її дуже швидкою. Вона дозволяє виконувати операції над масивами чисел набагато швидше, ніж стандартні списки в Python. Враховуючи ці переваги, NumPy є важливим інструментом для роботи з числовими даними в Pythoni популярним вибором для наукових та інженерних досліджень, а також для розробки програмного забезпечення, яке вимагає обробки числових даних.

Бібліотека Matplotlib має численні переваги, які роблять її популярною серед аналітиків даних, науковців та розробників. Matplotlib надає зручний та інтуїтивний інтерфейс для створення графіків та діаграм, підтримує виведення графіків у різних форматах, таких як растрові та векторні зображення, що дозволяє використовувати їх у різних медіа та публікаціях. Matplotlib має велику спільноту користувачів і розробників, а також добре документовану бібліотеку,

що робить її доступною для навчання та вирішення проблем. Matplotlib допомагає зробити дані більш зрозумілими та інформативними.

Обсяги даних постійно зростають, що створює нові виклики та можливості для аналізу та використання даних. Це означає, що Data Science залишається актуальним і важливим. Інструменти та технології Data Science постійно вдосконалюються. Машинне навчання, глибоке навчання, обробка природної мови та інші нові техніки дозволяють розвивати більш точні та потужні моделі. Тому розроблені рекомендації для організації навчання та досліджень в області Data Science є важливими для набуття знань та навиків у цій сфері, адже наука про дані має значний вплив на суспільство, включаючи покращення медичних діагнозів, розв’язання складних глобальних проблем, покращення технологій та послуг. Це робить Data Science важливою галуззю для розв’язання сучасних проблем.

Усі ці фактори свідчать про те, що Data Science залишається дуже обіцяючою та швидко зростаючою галуззю з великими можливостями для кар’єри та внесення важливого внеску в розвиток різних галузей.

ПЕРЕЛІК ПОСИЛАНЬ

1. http://www.learnpython.org/en/Welcome
2. http://realpython.com/python3-object-oriented-programming/
3. http://leetcode.com/problemset/all/
4. http://perso.limsi.fr/pointal/
5. http://media/python:cours:mementopython3-english.pdf
6. https://numpy.org/
7. https://pandas.pydata.org/
8. https://matplotlib.org/
9. https://scipy.org/
10. https://scikit-learn.org/stable/
11. https://keras.io/
12. https://pytorch.org/
13. https://www.tensorflow.org/
14. W. Yu, W. Cheng, C. Aggarwal, K. Zhang, H. Chen, and Wei Wang. NetWalk: A flexible deep embedding approach for anomaly Detection in dynamic networks, ACM KDD Conference, 2018.
15. Jake VanderPlas Python Data Science Handbook: Essential Tools for Working with Data, 2022, 551p.
16. Heller, Nicholas et al. (2020). The KiTS19 Challenge Data: 300 Kidney Tumor Cases with Clinical Context, CT Semantic Segmentations, and Surgical Outcomes. arXiv: 1904. 00445 [q-bio.QM].
17. Hollo, Kaspar (2019). Exploring the Value of Weakly-Supervised Deep Learning Approaches for Artefact Segmentation in Brightfield Microscopy Images. URL: https://comserv. cs . ut . ee / home / files / hollo _ softwareengineering _ 2021.
18. Wang, Haofan et al. (2020). Score-CAM: Score-Weighted Visual Explanations for Convo-lutional Neural Networks. arXiv: 1910.01279 [cs.CV].
19. Yang, Guanyu et al. (2020). “Weakly-supervised convolutional neural networks ofrenal tumor segmentation in abdominal CTA images”. In: BMC Medical Imaging

20.1, p. 37. ISSN: 1471-2342. DOI: 10.1186/s12880- 020- 00435- w. URL: https://doi.org/10.1186/s12880-020-00435-w.

20. Selvaraju, Ramprasaath R. et al. (2019). “Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization”. In: International Journal of ComputerVision 128.2, pp. 336–359. DOI: 10 . 1007 / s11263 - 019 - 01228 - 7. URL: https://doi.org/10.1007\%2Fs11263-019-01228-7.

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

(Презентація)