**КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**

**ІМЕНІ ТАРАСА ШЕВЧЕНКА**

Факультет комп’ютерних наук та кібернетики

Кафедра теорії та технології програмування

**Звіт**

до лабораторної роботи № 3

Виконав студент 4-го курсу

Групи ТТП-42

Ходаков Максим Олегович

Київ – 2025

**Короткий опис області**

Даний дата сет візуалізує список книжок наявних на книжковій платформі, разом з рейтингом та кількістю рецензій. Також для кожної книжки наявні автор, мова написання, дата випуску та видавництва.

**DataSet**

Посилання: <https://www.kaggle.com/datasets/bahramjannesarr/goodreads-book-datasets-10m>

Приблизно 10 000 000 книг доступні в архівах сайту, і ці набори даних збираються з них. для запитів через API ми використовували бібліотеку python Goodreads, набори даних оновлюватимуться кожні 2 дні.

Ці набори даних дуже добре підходять для виконання двох завдань:

1 . Створення системи рекомендацій книг на основі 10 млн книг

2 . Використання колонок опису для НЛП

Репозиторій на Github

Посилання на проект на github: <https://github.com/BahramJannesar/GoodreadsBookDataset>

Зміст

Приблизно 10 000 000 книг доступні в архівах сайту, і ці набори даних збираються з них. для запитів через API ми використовували бібліотеку python Goodreads, набори даних оновлюватимуться кожні 2 дні.

Приклади даних books в JSON:

{

"Id": "5107",

"Name": "The Catcher in the Rye",

"RatingDist1": "1:133165",

"RatingDist2": "2:224884",

"RatingDist3": "3:553476",

"RatingDist4": "4:808278",

"RatingDist5": "5:891037",

"pagesNumber": 277,

"RatingDistTotal": "total:2610840",

"PublishMonth": 30,

"PublishDay": 1,

"Publisher": "Back Bay Books",

"CountsOfReview": 44046,

"PublishYear": 2001,

"Language": "eng",

"Authors": "J.D. Salinger",

"Rating": 3.8,

"ISBN": "0316769177",

"Count of text reviews": 55539,

"Description": "The hero-narrator of The Catcher in the Rye is an ancient child of sixteen, a native New Yorker named Holden Caulfield. Through

circumstances that tend to preclude adult, secondhand description, he leaves his prep school in Pennsylvania and goes underground in New York City for

three days. "

}

Кількість файлів при завантажені архіву:

A screenshot of a computer

Description automatically generated

Приклади завантажених .CSV

Books:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Name** | **RatingDist1** | **pagesNumber** | **RatingDist4** | **RatingDistTotal** | **PublishMonth** | **PublishDay** | **Publisher** | **CountsOfReview** | **PublishYear** | **Language** | **Authors** | **Rating** | **RatingDist2** | **RatingDist5** | **ISBN** | **RatingDist3** |
| Harry Potter and the Half-Blood Prince (Harry Potter, #6) | 1:9896 | 652 | 4:556485 | total:2298124 | 16 | 9 | Scholastic Inc. | 28062 | 2006 | eng | J.K. Rowling | 4.57 | 2:25317 | 5:1546466 |  | 3:159960 |

UserRating:

A screen shot of a computer screen

Description automatically generated

Основні поля для роботи з книгами:

Name – назва книги

RatingDist1 – кількість оцінок 1 для книги

RatingDist2 – кількість оцінок 2 для книги

RatingDist3 – кількість оцінок 3 для книги

RatingDist4 – кількість оцінок 4 для книги

RatingDist5 – кількість оцінок 5 для книги

PublishYear – рік випуску

PublishMonth – місяць випуску

PublishDay – день випуску

CountsOfReview – кількість рецензій

Publisher – видавець

Language – мова книги

Authors – автори

**Task 1**

Прогнозування рейтингу за допомогою лінійної регресії

### Лінійна регресія

Лінійна регресія — це метод прогнозування, який використовується для передбачення значень залежної змінної на основі незалежних змінних. Вона передбачає наявність лінійного зв'язку між незалежними змінними (в даному випадку, характеристиками книги) та залежною змінною (рейтинг).

Задача лінійної регресії полягає в тому, щоб знайти ці коефіцієнти таким чином, щоб мінімізувати помилку прогнозу, яку зазвичай вимірюють за допомогою метрики середньоквадратичної помилки (MSE).

### Метрики оцінки моделі

1. **R² (Коефіцієнт детермінації):** R² — це метрика, яка вимірює, яку частину змінності залежної змінної можна пояснити змінними незалежними змінними. Значення R² коливається від 0 до 1, де значення близьке до 1 означає, що модель добре пояснює варіативність у даних.
2. **Mean Squared Error (MSE):** Це метрика, яка вимірює середнє квадратичне відхилення між передбаченими значеннями та фактичними. Менше значення MSE свідчить про кращу модель.
3. **Mean Absolute Error (MAE):** Метріка, яка вимірює середнє абсолютне відхилення прогнозованих значень від фактичних.

## Підготовка даних

### 1. Завантаження та очищення даних

Ми завантажили набір даних за допомогою бібліотеки Pandas з CSV-файлу за допомогою функції pd.read\_csv. Використано роздільник ;, оскільки дані зберігаються у форматі CSV з таким роздільником.

### 2. Обробка пропущених значень

У процесі роботи з даними деякі значення можуть бути відсутні. У нашому випадку ми використовуємо SimpleImputer для заповнення відсутніх значень у числових колонках середнім значенням. Для категоріальних змінних ми заміняємо пропущені значення на найбільш часті.

### 3. Перетворення категоріальних змінних

Категоріальні змінні, такі як мова книги, були перетворені в числові за допомогою техніки **one-hot encoding**. Це дозволяє перетворити кожне категоріальне значення на окрему бінарну змінну.

## Алгоритм побудови моделі

1. **Розподіл даних:** Для тренування моделі ми розділили дані на тренувальну та тестову вибірки за допомогою функції train\_test\_split з параметром test\_size=0.2, що означає, що 20% даних було відкладено для тестування моделі.
2. **Побудова моделі лінійної регресії:** За допомогою класу LinearRegression з бібліотеки sklearn.linear\_model була побудована модель, яка тренувалась на тренувальних даних.
3. **Оцінка моделі:** Для оцінки моделі використовували метрики **R²** та **Mean Squared Error** (MSE).
4. **Прогнозування:** Після тренування моделі, ми використовуємо її для прогнозування рейтингу книги на основі введених користувачем даних.

**Код програми**

import pandas as *pd*import matplotlib.pyplot as *plt*from sklearn.linear\_model import LinearRegression  
from sklearn.model\_selection import train\_test\_split  
from sklearn.metrics import mean\_squared\_error, r2\_score  
from sklearn.impute import SimpleImputer  
from sklearn.preprocessing import OneHotEncoder  
  
  
*# Функція для отримання введених даних від користувача*def get\_user\_input():  
 print("Введіть розподіл оцінок для кожної категорії (від 1 до 5) та кількість відгуків, а також мову:")  
 *rating\_dist1* = float(input("Введіть розподіл оцінки 1 (кількість відгуків з оцінкою 1): "))  
 *rating\_dist2* = float(input("Введіть розподіл оцінки 2 (кількість відгуків з оцінкою 2): "))  
 *rating\_dist3* = float(input("Введіть розподіл оцінки 3 (кількість відгуків з оцінкою 3): "))  
 *rating\_dist4* = float(input("Введіть розподіл оцінки 4 (кількість відгуків з оцінкою 4): "))  
 *rating\_dist5* = float(input("Введіть розподіл оцінки 5 (кількість відгуків з оцінкою 5): "))  
 *counts\_of\_review* = int(input("Введіть загальну кількість відгуків: "))  
 *language* = input("Введіть мову (наприклад, 'eng'): ")  
 return [*rating\_dist1*, *rating\_dist2*, *rating\_dist3*, *rating\_dist4*, *rating\_dist5*, *counts\_of\_review*, *language*]  
  
  
*# Завантаження даних (для тренування моделі)  
df* = pd.read\_csv('CSV\_BI\_Lab1\_data\_source.csv', sep=';')  
  
*# Видалення ком у колонці 'Rating' та перетворення її на тип float  
df*['Rating'] = *df*['Rating'].replace({',': '.'}, regex=True).astype(float)  
  
*# Обробка відсутніх значень  
df* = *df*.dropna(subset=['Rating'])  
  
*# Заповнення відсутніх значень у колонках ознак середнім значенням  
imputer* = SimpleImputer(strategy='mean')  
*df*[['RatingDist1', 'RatingDist2', 'RatingDist3', 'RatingDist4', 'RatingDist5',  
 'CountsOfReview']] = *imputer*.fit\_transform(  
 *df*[['RatingDist1', 'RatingDist2', 'RatingDist3', 'RatingDist4', 'RatingDist5', 'CountsOfReview']])  
  
*# Переконатися, що всі ознаки мають числові значення (на випадок наявності строкових значень)  
df*[['RatingDist1', 'RatingDist2', 'RatingDist3', 'RatingDist4', 'RatingDist5', 'CountsOfReview']] = *df*[  
 ['RatingDist1', 'RatingDist2', 'RatingDist3', 'RatingDist4', 'RatingDist5', 'CountsOfReview']].apply(pd.to\_numeric,  
 errors='coerce')  
  
*# Кодування колонки 'Language' за допомогою OneHotEncoder  
encoder* = OneHotEncoder(sparse\_output=False) *# Виправлення: використовуємо sparse\_output=False замість sparse=False  
language\_encoded* = *encoder*.fit\_transform(*df*[['Language']])  
*language\_encoded\_df* = pd.DataFrame(*language\_encoded*, columns=*encoder*.get\_feature\_names\_out(['Language']))  
  
*# Додавання закодованої мови до DataFrame  
df* = pd.concat([*df*, *language\_encoded\_df*], axis=1)  
  
*# Вибір ознак для тренування моделі (включаючи закодовану мову)  
X* = *df*[['RatingDist1', 'RatingDist2', 'RatingDist3', 'RatingDist4', 'RatingDist5', 'CountsOfReview'] + list(  
 *language\_encoded\_df*.columns)]  
*y* = *df*['Rating']  
  
*# Переконатися, що немає NaN значень в X та y  
X* = *X*.dropna()  
*y* = *y*[*X*.index] *# Узгоджуємо y з ознаками (після видалення NaN з X)  
  
# Розподіл на тренувальні та тестові дані  
X\_train*, *X\_test*, *y\_train*, *y\_test* = train\_test\_split(*X*, *y*, test\_size=0.2, random\_state=42)  
  
*# Тренування моделі лінійної регресії  
model* = LinearRegression()  
*model*.fit(*X\_train*, *y\_train*)  
  
*# Оцінка моделі  
y\_pred* = *model*.predict(*X\_test*)  
*r2* = r2\_score(*y\_test*, *y\_pred*)  
*mse* = mean\_squared\_error(*y\_test*, *y\_pred*)  
  
*# Отримання введених даних від користувача та передбачення рейтингу  
user\_input* = get\_user\_input()  
  
*# Перетворення введених даних користувача в DataFrame з відповідними назвами колонок  
user\_input\_data* = *user\_input*[:-1] *# Видалення мови з введених даних для one-hot кодування  
user\_input\_language* = *user\_input*[-1] *# Витягуємо мову окремо  
  
# Перетворення введених даних користувача у DataFrame для ознак  
user\_input\_df* = pd.DataFrame([*user\_input\_data*], columns=*X*.columns[:-len(*language\_encoded\_df*.columns)])  
  
*# One-hot кодування введеної мови  
user\_input\_language\_encoded* = *encoder*.transform([[*user\_input\_language*]])  
*user\_input\_language\_df* = pd.DataFrame(*user\_input\_language\_encoded*, columns=*encoder*.get\_feature\_names\_out(['Language']))  
  
*# Об'єднуємо введені дані з закодованою мовою  
user\_input\_final* = pd.concat([*user\_input\_df*, *user\_input\_language\_df*], axis=1)  
  
*# Прогнозуємо за допомогою моделі  
prediction* = *model*.predict(*user\_input\_final*)  
  
*# Виведення результатів*print(f"Прогнозований рейтинг: {*prediction*[0]:.2f}")  
print(f"R2 Score: {*r2*:.2f}")  
print(f"Mean Squared Error: {*mse*:.2f}")  
  
*# Візуалізація  
  
# 1. Фактичні vs Прогнозовані рейтинги*plt.figure(figsize=(8, 6))  
plt.scatter(*y\_test*, *y\_pred*, color='blue', alpha=0.6)  
plt.plot([*y*.min(), *y*.max()], [*y*.min(), *y*.max()], color='red', linestyle='--')  
plt.title('Фактичні vs Прогнозовані Рейтинги')  
plt.xlabel('Фактичні Рейтинги')  
plt.ylabel('Прогнозовані Рейтинги')  
plt.show()  
  
*# 2. Важливість ознак (за допомогою коефіцієнтів)*plt.figure(figsize=(8, 6))  
*features* = *X*.columns  
*importance* = *model*.coef\_  
plt.barh(*features*, *importance*, color='green')  
plt.title('Важливість Ознак (Коефіцієнти Лінійної Регресії)')  
plt.xlabel('Значення коефіцієнтів')  
plt.ylabel('Ознаки')  
plt.show()  
  
*# 3. Розподіл рейтингів*plt.figure(figsize=(8, 6))  
plt.hist(*y*, bins=20, color='orange', edgecolor='black')  
plt.title('Розподіл Рейтингів')  
plt.xlabel('Рейтинг')  
plt.ylabel('Частота')  
plt.show()

### Опис програми

#### 1. Завантаження та попередня обробка даних

Дані завантажуються з CSV-файлу за допомогою бібліотеки **Pandas**. У наборі даних є кілька колонок, зокрема:

* **Rating** (рейтинг),
* **RatingDist1, RatingDist2, RatingDist3, RatingDist4, RatingDist5** (розподіл оцінок за категоріями),
* **CountsOfReview** (кількість відгуків),
* **Language** (мова книги).

У колонці **Rating** можуть бути присутні коми замість точок для позначення десяткових дробів. Вони заміняються на точки за допомогою методу replace, після чого значення перетворюються на тип float.

#### 2. Заповнення відсутніх значень

У даних можуть бути відсутні деякі значення. Наприклад, в колонках **RatingDist1, RatingDist2, RatingDist3, RatingDist4, RatingDist5** і **CountsOfReview** можуть бути пропуски. Ми використовуємо **SimpleImputer** для заповнення цих пропусків середнім значенням по кожній колонці. Крім того, пропуски у колонці **Language** заповнюються найбільш частим значенням за допомогою методу mode().

#### 3. One-Hot Encoding для категоріальних змінних

Мова є категоріальною змінною, тому її необхідно перетворити на числові значення. Для цього використовуємо техніку **One-Hot Encoding** за допомогою **OneHotEncoder** з бібліотеки **sklearn**. Цей метод перетворює категоріальні змінні в кілька бінарних колонок, де кожна колонка відповідає окремій категорії.

#### 4. Створення моделі лінійної регресії

Після попередньої обробки даних вибираються ознаки (колонки) для тренування моделі:

* **RatingDist1, RatingDist2, RatingDist3, RatingDist4, RatingDist5** (розподіл оцінок),
* **CountsOfReview** (кількість відгуків),
* **Language** (перетворена за допомогою OneHot Encoding).

Ми використовуємо метод **train\_test\_split** для поділу даних на тренувальну та тестову вибірки (80% для тренування і 20% для тестування).

#### 5. Тренування моделі

Модель **LinearRegression** з бібліотеки **sklearn** тренується на вибраних ознаках. Модель навчається визначати залежність між оцінками та іншими ознаками (розподіли оцінок, кількість відгуків, мова).

#### 6. Оцінка моделі

Після тренування ми оцінюємо модель за допомогою двох основних метрик:

* **R² Score** — показник якості моделі.
* **Mean Squared Error (MSE)** — вимірює середнє квадратичне відхилення прогнозів від фактичних значень.

#### 7. Прогнозування для введених даних

Після тренування моделі, прогнози можуть бути зроблені для введених користувачем значень. Користувач вводить розподіл оцінок для кожної категорії, кількість відгуків і мову, після чого модель прогнозує рейтинг на основі цих значень.

#### 8. Візуалізація результатів

Для візуалізації результатів використовуються такі графіки:

* **Фактичні vs Прогнозовані рейтинги**: Графік для порівняння фактичних значень рейтингу з прогнозованими.
* **Важливість ознак**: Графік, що показує коефіцієнти для кожної з ознак моделі, тим самим визначаючи їхню важливість.
* **Розподіл рейтингів**: Побудова гістограми для аналізу розподілу рейтингів у наборі даних.

**Лістинг та графіки**

**A white background with black text

Description automatically generated**

**A white background with black and white clouds

Description automatically generated**

Рис.1. Лістинг програми

A graph of a number of bars

Description automatically generated with medium confidence

Рис.2. Розподіл рейтингів

A graph with green lines

Description automatically generated

Рис.3. Графік важливості ознак

A graph with blue dots and red lines

Description automatically generated

Рис.4. Рейтинги

**Task 2**

Кластеризація рейтингу за допомогою алгоритму Kmeans

Кластеризація — це метод машинного навчання, що використовується для групування об'єктів у класи або групи (кластер) на основі подібності між ними. Це один з основних методів в аналізі даних, коли потрібно знайти структуру в даних без попереднього розмітки.

**Алгоритм KMeans** — один з найбільш відомих алгоритмів кластеризації, який належить до методів навчання без учителя. Метою KMeans є поділ набору даних на KKK кластерів, де кожен кластер містить елементи, що мають схожі характеристики.

Основні етапи роботи алгоритму KMeans:

1. **Вибір кількості кластерів (K):** Алгоритм вимагає вказати кількість кластерів KKK, яку потрібно отримати в результаті кластеризації. Це може бути зроблено вручну або за допомогою методів, таких як метод ліктя або силуетний аналіз.
2. **Ініціалізація центрів кластерів:** Початково алгоритм вибирає випадкові точки як центри KKK кластерів. Ці точки називаються **центроїдами**.
3. **Призначення кожної точки кластеру:** Для кожної точки даних знаходиться найближчий центроид, і точка призначається до цього кластера.
4. **Оновлення центрів кластерів:** Після того, як всі точки будуть розподілені по кластерах, для кожного кластера обчислюється новий центр — середнє значення всіх точок, що належать до цього кластера.
5. **Повторення етапів 3 та 4:** Кроки 3 та 4 повторюються до тих пір, поки центри кластерів не стабілізуються (тобто не змінюються суттєво між ітераціями).

**Ключові терміни в кластеризації:**

* **Центроїд** — це центр кластера, який обчислюється як середнє значення всіх точок, що належать цьому кластеру.
* **Inertia** (WSS, Within-cluster Sum of Squares) — це міра розсіяння точок в кожному кластері, виміряна як сума квадратів відстаней від кожної точки до центроїда цього кластера. Чим менше значення Inertia, тим краще розділені кластери.
* **Silhouette Score** — вимірює наскільки правильно класифіковані точки в своїх кластерах порівняно з іншими кластерами. Значення варіюється від -1 до 1, де 1 — це ідеальна кластеризація.
* **Davies-Bouldin Index** — це міра схожості між кластерами, де менші значення вказують на кращу кластеризацію.

**2. Метрики оцінки якості кластеризації**

* **Silhouette Score:** Оцінює якість кластеризації шляхом вимірювання середньої відстані між точками в кластері та відстані між точками з іншого кластеру. Чим вище значення, тим краще.
* **Davies-Bouldin Index:** Вимірює середню схожість між кожною парою кластерів. Чим менше значення, тим краще.
* **Inertia (WSS):** Це сума квадратів відстаней від кожної точки до її центроїда. Менше значення означає, що кластери більш згуртовані.

**Опис роботи програми**

**1. Завантаження даних**

df = pd.read\_csv('CSV\_BI\_Lab1\_data\_source.csv', sep=';')

Ми завантажуємо дані з CSV-файлу, використовуючи бібліотеку **Pandas**. У даному випадку файл має роздільник ;, тому використовуємо параметр sep=';'.

**2. Попередня обробка даних**

**Обробка пропущених значень**

df = df.dropna(subset=['Rating'])

Ми видаляємо всі рядки, де відсутній рейтинг (**Rating**). Це потрібно, щоб не потрапити в ситуацію, коли модель не може працювати з відсутніми значеннями.

**Заповнення відсутніх значень**

imputer = SimpleImputer(strategy='mean')

df[['RatingDist1', 'RatingDist2', 'RatingDist3', 'RatingDist4', 'RatingDist5', 'CountsOfReview']] = imputer.fit\_transform(df[['RatingDist1', 'RatingDist2', 'RatingDist3', 'RatingDist4', 'RatingDist5', 'CountsOfReview']])

Тут ми використовуємо **SimpleImputer** для заповнення пропущених значень у числових колонках. Значення заміщуються середнім значенням по кожній колонці.

**3. Кластеризація**

**Нормалізація ознак**

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

Для нормалізації даних ми використовуємо **StandardScaler**, який здійснює масштабування кожної ознаки до середнього значення 0 і стандартного відхилення 1. Це необхідно для того, щоб уникнути домінування одних ознак над іншими при кластеризації, оскільки алгоритм KMeans заснований на відстані між точками.

**Виконання кластеризації**

kmeans = KMeans(n\_clusters=3, random\_state=42)

kmeans.fit(X\_scaled)

df['KMeans\_Cluster'] = kmeans.labels\_

Ми застосовуємо **KMeans** з K=3K = 3K=3 кластерами до нормалізованих даних. Алгоритм класифікує кожну точку в один з трьох кластерів. Результати кластеризації зберігаються в новій колонці KMeans\_Cluster.

**4. Оцінка якості кластеризації**

**Silhouette Score**

silhouette = silhouette\_score(X\_sampled, kmeans.labels\_[sample\_indices])

Використовуємо **Silhouette Score** для оцінки якості кластеризації. Це допомагає зрозуміти, наскільки правильно об'єкти віднесені до своїх кластерів.

**Davies-Bouldin Index**

davies\_bouldin = davies\_bouldin\_score(X\_sampled, kmeans.labels\_[sample\_indices])

Використовуємо **Davies-Bouldin Index**, щоб виміряти схожість між кластерами. Чим менше значення, тим кращі кластери.

**Inertia**

inertia = kmeans.inertia\_

**Inertia** вимірює відстань між точками і їх центроїдами в кожному кластері. Чим менше значення Inertia, тим краще кластеризація.

**5. Візуалізація результатів кластеризації**

**Графіки для оцінки кластеризації:**

* **Розподіл кластерів за ознаками RatingDist1 і RatingDist2:**

plt.scatter(df['RatingDist1'], df['RatingDist2'], c=df['KMeans\_Cluster'], cmap='viridis', alpha=0.6)

* **Розподіл кластерів за ознаками RatingDist3 і RatingDist4:**

plt.scatter(df['RatingDist3'], df['RatingDist4'], c=df['KMeans\_Cluster'], cmap='viridis', alpha=0.6)

* **Візуалізація результатів з центрами кластерів:**

plt.scatter(df['RatingDist1'], df['RatingDist2'], c=df['KMeans\_Cluster'], cmap='viridis', alpha=0.6)

centers = kmeans.cluster\_centers\_

plt.scatter(centers[:, 0], centers[:, 1], c='red', marker='X', s=200, label='Cluster Centers')

Ці графіки допомагають наочно побачити, як групуються об'єкти за різними характеристиками. Колір точок на графіку вказує на приналежність точки до конкретного кластера.

**6. Виведення результатів**

Наприкінці програми ми виводимо:

* **Кількість елементів у кожному кластері.**
* **Центри кластерів**.
* **Статистику для кожного кластера** (середнє, мінімум, максимум для кожної ознаки).

**Код програми**

import pandas as *pd*import matplotlib.pyplot as *plt*from sklearn.cluster import KMeans  
from sklearn.preprocessing import StandardScaler  
from sklearn.impute import SimpleImputer  
from sklearn.metrics import silhouette\_score, davies\_bouldin\_score  
import numpy as *np  
  
# Завантаження даних  
df* = pd.read\_csv('CSV\_BI\_Lab1\_data\_source.csv', sep=';')  
  
*# Видаляємо коми в колонці 'Rating' і перетворюємо її на тип float  
df*['Rating'] = *df*['Rating'].replace({',': '.'}, regex=True).astype(float)  
  
*# Обробка відсутніх значень  
df* = *df*.dropna(subset=['Rating'])  
  
*# Заповнення відсутніх значень в інших ознаках середнім значенням  
imputer* = SimpleImputer(strategy='mean')  
*df*[['RatingDist1', 'RatingDist2', 'RatingDist3', 'RatingDist4', 'RatingDist5', 'CountsOfReview']] = *imputer*.fit\_transform(*df*[['RatingDist1', 'RatingDist2', 'RatingDist3', 'RatingDist4', 'RatingDist5', 'CountsOfReview']])  
  
*# Вибір ознак для кластеризації  
X* = *df*[['RatingDist1', 'RatingDist2', 'RatingDist3', 'RatingDist4', 'RatingDist5']]  
  
*# Нормалізація ознак  
scaler* = StandardScaler()  
*X\_scaled* = *scaler*.fit\_transform(*X*)  
  
*# Кластеризація за допомогою KMeans  
kmeans* = KMeans(n\_clusters=3, random\_state=42)  
*kmeans*.fit(*X\_scaled*)  
*df*['KMeans\_Cluster'] = *kmeans*.labels\_  
  
*# Виведення інформації про кількість елементів в кожному кластері*print("Кількість елементів в кожному кластері:")  
print(*df*['KMeans\_Cluster'].value\_counts())  
  
*# Виведення координат центрів кластерів*print("\nЦентри кластерів (KMeans):")  
print(*kmeans*.cluster\_centers\_)  
  
*# Статистика по кожному кластеру для кожної ознаки*print("\nСтатистика по кожному кластеру:")  
for *cluster* in range(3): *# 3 кластери, як визначено в KMeans  
 cluster\_data* = *df*[*df*['KMeans\_Cluster'] == *cluster*]  
 print(f"\nКластер {*cluster*}:")  
 print(*cluster\_data*[['RatingDist1', 'RatingDist2', 'RatingDist3', 'RatingDist4', 'RatingDist5']].describe())  
  
*# Виведення розподілу кожного кластера по ознаках*print("\nРозподіл по кожному кластеру для ознак RatingDist1, RatingDist2:")  
print(*df*.groupby('KMeans\_Cluster')[['RatingDist1', 'RatingDist2']].mean())  
  
*# Додаткові метрики:  
# Використання підвибірки (наприклад, 10% даних)  
sample\_size* = int(0.1 \* *X\_scaled*.shape[0]) *# 10% від усіх даних  
sample\_indices* = np.random.choice(*X\_scaled*.shape[0], *sample\_size*, replace=False)  
*X\_sampled* = *X\_scaled*[*sample\_indices*]  
  
*# 1. Silhouette Score  
silhouette* = silhouette\_score(*X\_sampled*, *kmeans*.labels\_[*sample\_indices*])  
print(f"\nSilhouette Score: {*silhouette*:.4f}")  
  
*# 2. Davies-Bouldin Index  
davies\_bouldin* = davies\_bouldin\_score(*X\_sampled*, *kmeans*.labels\_[*sample\_indices*])  
print(f"\nDavies-Bouldin Index: {*davies\_bouldin*:.4f}")  
  
*# 3. Inertia (WSS)  
inertia* = *kmeans*.inertia\_  
print(f"\nInertia (WSS): {*inertia*:.4f}")  
  
*# Візуалізація результатів кластеризації KMeans (1)*plt.figure(figsize=(8, 6))  
plt.scatter(*df*['RatingDist1'], *df*['RatingDist2'], c=*df*['KMeans\_Cluster'], cmap='viridis', alpha=0.6)  
plt.title('KMeans Clustering (RatingDist1 vs RatingDist2)')  
plt.xlabel('RatingDist1')  
plt.ylabel('RatingDist2')  
plt.colorbar(label='Cluster')  
plt.show()  
  
*# Візуалізація результатів кластеризації KMeans (2)*plt.figure(figsize=(8, 6))  
plt.scatter(*df*['RatingDist3'], *df*['RatingDist4'], c=*df*['KMeans\_Cluster'], cmap='viridis', alpha=0.6)  
plt.title('KMeans Clustering (RatingDist3 vs RatingDist4)')  
plt.xlabel('RatingDist3')  
plt.ylabel('RatingDist4')  
plt.colorbar(label='Cluster')  
plt.show()  
  
*# Візуалізація результатів кластеризації KMeans з центрами кластерів*plt.figure(figsize=(8, 6))  
plt.scatter(*df*['RatingDist1'], *df*['RatingDist2'], c=*df*['KMeans\_Cluster'], cmap='viridis', alpha=0.6)  
*centers* = *kmeans*.cluster\_centers\_  
plt.scatter(*centers*[:, 0], *centers*[:, 1], c='red', marker='X', s=200, label='Cluster Centers')  
plt.title('KMeans Clustering with Centers (RatingDist1 vs RatingDist2)')  
plt.xlabel('RatingDist1')  
plt.ylabel('RatingDist2')  
plt.colorbar(label='Cluster')  
plt.legend()  
plt.show()  
  
*# Додатково: відображення перших кількох рядків даних з класифікацією*print("\nПерші кілька рядків з класифікацією:")  
print(*df*[['RatingDist1', 'RatingDist2', 'KMeans\_Cluster']].head())

Лістинг програми

A computer screen shot of a number

Description automatically generated

A number of numbers on a white background

Description automatically generated

A table of numbers with black text

Description automatically generated

A screenshot of a computer

Description automatically generated

A chart with different colored dots

Description automatically generated

Рис.5. Dist1 та Dist2

A chart with a number of dots

Description automatically generated with medium confidence

Рис.6. Dist3 та Dist4

A chart with different colored dots

Description automatically generated

Рис.7. Кластеризація з центрами

**Task 3**

Кластеризація мов та рев’ю за допомогою алгоритму Kmeans

**Кластеризація** — це метод навчання без учителя, який дозволяє поділити об'єкти на групи (класи, кластери), де об'єкти в одному кластері є схожими, а між кластерами — значні відмінності. У випадку з даними книг класифікація на основі подібних характеристик дозволяє отримати групи книг, які мають схожі відгуки, оцінки та інші фактори.

Алгоритм **KMeans** є одним з найбільш популярних алгоритмів кластеризації. Відмінність цього методу полягає в тому, що кількість кластерів KKK необхідно вказати заздалегідь. Алгоритм працює наступним чином:

1. Вибирає KKK початкових центрів кластерів (центроїдів).
2. Призначає кожну точку до найближчого центроїда.
3. Обчислює нові центроїди для кожного кластера.
4. Повторює кроки 2 та 3, поки центри кластерів не стабілізуються (тобто, класи не змінюються).

### 2. **Методи оцінки кластеризації**

* **Silhouette Score** — це міра, яка показує, як правильно кожна точка кластеризована. Значення варіюються від -1 до 1:
  + Значення близьке до 1 вказує на те, що точка добре класифікована в свій кластер.
  + Значення близьке до 0 вказує на те, що точка знаходиться на межі двох кластерів.
  + Значення близьке до -1 свідчить про неправильну класифікацію.
* **Davies-Bouldin Index** — це метрика, яка вимірює середню схожість між кожною парою кластерів. Чим менше значення індексу, тим краща кластеризація.
* **Inertia (WSS)** — це міра того, як сильно точки в кластері відрізняються від центроїда. Чим менше значення Inertia, тим більше згуртовані кластери.

### 3. **PCA (Principal Component Analysis)**

**PCA** — це метод зменшення розмірності, який дозволяє звести кілька взаємозалежних ознак до меншої кількості незалежних компонент. Цей метод корисний, коли дані мають велику кількість ознак і потрібно їх візуалізувати або зменшити обчислювальні витрати.

Після кластеризації ми застосовуємо PCA для проекції даних на дві основні компоненти (2D), що дозволяє зручніше візуалізувати результати кластеризації.

## Опис програми

### 1. Завантаження та обробка даних

df = pd.read\_csv('CSV\_BI\_Lab1\_data\_source.csv', sep=';')

df = df.dropna(subset=['Language', 'CountsOfReview']) # Видалення рядків з пропущеними значеннями

Ми завантажуємо набір даних за допомогою бібліотеки **Pandas** і використовуємо роздільник ;, оскільки це формат CSV-файлу. Далі видаляємо рядки, де відсутні значення в колонках Language та CountsOfReview, оскільки вони критично важливі для кластеризації.

### 2. Перетворення категоріальних змінних

encoder = OneHotEncoder(sparse\_output=False)

language\_encoded = encoder.fit\_transform(df[['Language']])

language\_encoded\_df = pd.DataFrame(language\_encoded, columns=encoder.get\_feature\_names\_out(['Language']))

Для перетворення категоріальної змінної **Language** в числову форму використовується **OneHotEncoder**, що дозволяє створити нові бінарні змінні для кожної унікальної мови. Це необхідно, оскільки **KMeans** працює тільки з числовими ознаками.

### 3. Заповнення пропущених значень

imputer = SimpleImputer(strategy='mean')

df[['CountsOfReview'] + language\_encoded\_df.columns.tolist()] = imputer.fit\_transform(df[['CountsOfReview'] + language\_encoded\_df.columns.tolist()])

Пропущені значення для числових колонок (у тому числі **CountsOfReview**) заповнюються середнім значенням кожної колонки, що дозволяє зберегти цілісність даних для кластеризації.

### 4. Нормалізація даних

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

Для коректної кластеризації необхідно нормалізувати дані. Це робиться через **StandardScaler**, який змінює середнє значення кожної ознаки до 0, а стандартне відхилення — до 1. Це необхідно, щоб жодна з ознак не мала значно більшого масштабу, ніж інші.

### 5. Кластеризація за допомогою KMeans

kmeans = KMeans(n\_clusters=3, random\_state=42)

kmeans.fit(X\_scaled)

df['KMeans\_Cluster'] = kmeans.labels\_

Алгоритм **KMeans** виконує кластеризацію на трьох кластерах. Після виконання кластеризації кожен елемент отримує мітку кластера в колонці KMeans\_Cluster.

### 6. Оцінка якості кластеризації

#### Silhouette Score

silhouette = silhouette\_score(X\_sampled, kmeans.labels\_[sample\_indices])

Для оцінки якості кластеризації використовуємо **Silhouette Score**, що показує, наскільки добре розділені кластери.

#### Davies-Bouldin Index

davies\_bouldin = davies\_bouldin\_score(X\_sampled, kmeans.labels\_[sample\_indices])

**Davies-Bouldin Index** оцінює схожість між кластерами. Чим менше значення, тим краще.

#### Inertia (WSS)

inertia = kmeans.inertia\_

**Inertia** вимірює згуртованість кластерів — чим менше значення Inertia, тим кращі кластери.

### 7. Візуалізація результатів кластеризації

#### Візуалізація PCA

pca = PCA(n\_components=2)

X\_pca = pca.fit\_transform(X\_scaled)

Для зручності візуалізації ми використовуємо **PCA** для зменшення вимірності даних до 2D. Це дозволяє нам побудувати графік, на якому кожен кластер видно в двовимірному просторі.

#### Графіки результатів кластеризації

plt.scatter(X\_pca[:, 0], X\_pca[:, 1], c=df['KMeans\_Cluster'], cmap='viridis', alpha=0.6)

Ми будуємо графік, на якому кожна точка представлена двома компонентами після **PCA**, а колір точки вказує на приналежність до конкретного кластера.

Код програми

import pandas as *pd*import matplotlib.pyplot as *plt*from sklearn.cluster import KMeans  
from sklearn.preprocessing import StandardScaler, OneHotEncoder  
from sklearn.impute import SimpleImputer  
from sklearn.metrics import silhouette\_score, davies\_bouldin\_score  
import numpy as *np*from sklearn.decomposition import PCA  
  
*# Завантаження даних  
df* = pd.read\_csv('CSV\_BI\_Lab1\_data\_source.csv', sep=';')  
  
*# Обробка відсутніх значень  
df* = *df*.dropna(subset=['Language', 'CountsOfReview']) *# Видаляємо рядки з пропущеними значеннями в Language та CountsOfReview  
  
# Перетворення категоріальної змінної 'Language' у числову за допомогою OneHotEncoder  
encoder* = OneHotEncoder(sparse\_output=False)  
  
*# Перетворення Language в числові ознаки  
language\_encoded* = *encoder*.fit\_transform(*df*[['Language']])  
  
*# Створення DataFrame для закодованих колонок  
language\_encoded\_df* = pd.DataFrame(*language\_encoded*, columns=*encoder*.get\_feature\_names\_out(['Language']))  
  
*# Додавання закодованих колонок до основного DataFrame  
df* = pd.concat([*df*, *language\_encoded\_df*], axis=1)  
  
*# Заповнення пропущених значень для числових колонок, включаючи CountsOfReview  
imputer* = SimpleImputer(strategy='mean')  
*df*[['CountsOfReview'] + *language\_encoded\_df*.columns.tolist()] = *imputer*.fit\_transform(*df*[['CountsOfReview'] + *language\_encoded\_df*.columns.tolist()])  
  
*# Вибір ознак для кластеризації  
X* = *df*[*language\_encoded\_df*.columns.tolist() + ['CountsOfReview']]  
  
*# Нормалізація ознак  
scaler* = StandardScaler()  
*X\_scaled* = *scaler*.fit\_transform(*X*)  
  
*# Кластеризація за допомогою KMeans  
kmeans* = KMeans(n\_clusters=3, random\_state=42)  
*kmeans*.fit(*X\_scaled*)  
*df*['KMeans\_Cluster'] = *kmeans*.labels\_  
  
*# Виведення інформації про кількість елементів в кожному кластері*print("Кількість елементів в кожному кластері:")  
print(*df*['KMeans\_Cluster'].value\_counts())  
  
*# Виведення координат центрів кластерів*print("\nЦентри кластерів (KMeans):")  
print(*kmeans*.cluster\_centers\_)  
  
*# Статистика по кожному кластеру для кожної ознаки*print("\nСтатистика по кожному кластеру:")  
for *cluster* in range(3): *# 3 кластери, як визначено в KMeans  
 cluster\_data* = *df*[*df*['KMeans\_Cluster'] == *cluster*]  
 print(f"\nКластер {*cluster*}:")  
 print(*cluster\_data*[*language\_encoded\_df*.columns.tolist() + ['CountsOfReview']].describe())  
  
*# Виведення розподілу кожного кластера по ознаках*print("\nРозподіл по кожному кластеру для ознак Language та CountsOfReview:")  
print(*df*.groupby('KMeans\_Cluster')[*language\_encoded\_df*.columns.tolist() + ['CountsOfReview']].mean())  
  
*# Додаткові метрики:  
# Використання підвибірки (наприклад, 10% даних)  
sample\_size* = int(0.1 \* *X\_scaled*.shape[0]) *# 10% від усіх даних  
sample\_indices* = np.random.choice(*X\_scaled*.shape[0], *sample\_size*, replace=False)  
*X\_sampled* = *X\_scaled*[*sample\_indices*]  
  
*# 1. Silhouette Score  
silhouette* = silhouette\_score(*X\_sampled*, *kmeans*.labels\_[*sample\_indices*])  
print(f"\nSilhouette Score: {*silhouette*:.4f}")  
  
*# 2. Davies-Bouldin Index  
davies\_bouldin* = davies\_bouldin\_score(*X\_sampled*, *kmeans*.labels\_[*sample\_indices*])  
print(f"\nDavies-Bouldin Index: {*davies\_bouldin*:.4f}")  
  
*# 3. Inertia (WSS)  
inertia* = *kmeans*.inertia\_  
print(f"\nInertia (WSS): {*inertia*:.4f}")  
  
*# Візуалізація результатів кластеризації KMeans  
# Використовуємо PCA для зменшення вимірності до 2D для зручнішої візуалізації  
pca* = PCA(n\_components=2)  
*X\_pca* = *pca*.fit\_transform(*X\_scaled*)  
  
*# Візуалізація кластерів з центрами*plt.figure(figsize=(8, 6))  
plt.scatter(*X\_pca*[:, 0], *X\_pca*[:, 1], c=*df*['KMeans\_Cluster'], cmap='viridis', alpha=0.6)  
*centers* = *pca*.transform(*kmeans*.cluster\_centers\_)  
plt.scatter(*centers*[:, 0], *centers*[:, 1], c='red', marker='X', s=200, label='Cluster Centers')  
plt.title('KMeans Clustering with Centers (PCA Projection)')  
plt.xlabel('PCA Component 1')  
plt.ylabel('PCA Component 2')  
plt.colorbar(label='Cluster')  
plt.legend()  
plt.show()  
  
*# Додатково: відображення перших кількох рядків даних з класифікацією*print("\nПерші кілька рядків з класифікацією:")  
print(*df*[['Language', 'CountsOfReview', 'KMeans\_Cluster']].head())

Лістинг та результати роботи

Кількість елементів в кожному кластері:

KMeans\_Cluster

0 117951

2 1857

1 1

Name: count, dtype: int64

Центри кластерів (KMeans):

[[ 1.59334125e-04 7.96653163e-05 2.75990931e-04 1.37986392e-04

7.96653163e-05 7.96653163e-05 3.65125695e-04 -1.24729557e-01

7.48973401e-03 3.89781096e-02 2.39009918e-04 7.96653163e-05

1.12664594e-04 3.62988331e-03 7.96653163e-05 2.25339065e-04

2.54297431e-03 7.96653163e-05 1.78142269e-04 1.37986392e-04

5.22560778e-04 1.59334125e-04 7.96653163e-05 7.96653163e-05

1.12664594e-04 -2.85491680e-03 1.12664594e-04 1.12664594e-04

9.05671360e-04 8.84319626e-04 2.98108633e-04 7.96653163e-05

4.50757179e-04 7.96653163e-05 1.12664594e-04 8.58743483e-04

7.96653163e-05 1.12664594e-04 3.73720795e-04 1.59334125e-04

4.14031777e-04 1.59334125e-04 6.66864097e-04 1.59334125e-04

4.71423478e-04 1.78142269e-04 7.96653163e-05 3.29196157e-03

7.96653163e-05 3.73720795e-04 7.96653163e-05 7.96653163e-05

7.96653163e-05 7.96653163e-05 2.75990931e-04 1.78142269e-04

7.96653163e-05 1.37986392e-04 7.96653163e-05 6.94886971e-04

-7.35049312e-04]

[-1.01149727e-02 -5.05737552e-03 -1.75206708e-02 -8.75975935e-03

-5.05737552e-03 -5.05737552e-03 -2.31791931e-02 -2.20953622e-01

-4.75469115e-01 -2.47443865e+00 -1.51730134e-02 -5.05737552e-03

-7.15226131e-03 -2.30435073e-01 -5.05737552e-03 -1.43051497e-02

-1.61435072e-01 -5.05737552e-03 -1.13089659e-02 -8.75975935e-03

-3.31736094e-02 -1.01149727e-02 -5.05737552e-03 -5.05737552e-03

-7.15226131e-03 3.46131838e+02 -7.15226131e-03 -7.15226131e-03

-5.74945331e-02 -5.61390658e-02 -1.89247639e-02 -5.05737552e-03

-2.86153175e-02 -5.05737552e-03 -7.15226131e-03 -5.45154212e-02

-5.05737552e-03 -7.15226131e-03 -2.37248339e-02 -1.01149727e-02

-2.62838870e-02 -1.01149727e-02 -4.23343849e-02 -1.01149727e-02

-2.99272716e-02 -1.13089659e-02 -5.05737552e-03 -2.08982863e-01

-5.05737552e-03 -2.37248339e-02 -5.05737552e-03 -5.05737552e-03

-5.05737552e-03 -5.05737552e-03 -1.75206708e-02 -1.13089659e-02

-5.05737552e-03 -8.75975935e-03 -5.05737552e-03 -4.41133548e-02

-6.31422940e-17]

[-1.01149727e-02 -5.05737552e-03 -1.75206708e-02 -8.75975935e-03

-5.05737552e-03 -5.05737552e-03 -2.31791931e-02 7.92256162e+00

-4.75469115e-01 -2.47443865e+00 -1.51730134e-02 -5.05737552e-03

-7.15226131e-03 -2.30435073e-01 -5.05737552e-03 -1.43051497e-02

-1.61435072e-01 -5.05737552e-03 -1.13089659e-02 -8.75975935e-03

-3.31736094e-02 -1.01149727e-02 -5.05737552e-03 -5.05737552e-03

-7.15226131e-03 -5.05737552e-03 -7.15226131e-03 -7.15226131e-03

-5.74945331e-02 -5.61390658e-02 -1.89247639e-02 -5.05737552e-03

-2.86153175e-02 -5.05737552e-03 -7.15226131e-03 -5.45154212e-02

-5.05737552e-03 -7.15226131e-03 -2.37248339e-02 -1.01149727e-02

-2.62838870e-02 -1.01149727e-02 -4.23343849e-02 -1.01149727e-02

-2.99272716e-02 -1.13089659e-02 -5.05737552e-03 -2.08982863e-01

-5.05737552e-03 -2.37248339e-02 -5.05737552e-03 -5.05737552e-03

-5.05737552e-03 -5.05737552e-03 -1.75206708e-02 -1.13089659e-02

-5.05737552e-03 -8.75975935e-03 -5.05737552e-03 -4.41133548e-02

4.66880998e-02]]

Статистика по кожному кластеру:

Кластер 0:

Language\_afr Language\_ang ... Language\_zho CountsOfReview

count 117951.000000 117951.000000 ... 117951.000000 117951.000000

mean 0.000059 0.000015 ... 0.001128 156.972420

std 0.005823 0.002912 ... 0.025369 890.095830

min 0.000000 0.000000 ... 0.000000 0.000000

25% 0.000000 0.000000 ... 0.000000 8.000000

50% 0.000000 0.000000 ... 0.000000 152.000000

75% 0.000058 0.000015 ... 0.001110 157.634143

max 1.000000 1.000000 ... 1.000000 94850.000000

[8 rows x 61 columns]

Кластер 1:

Language\_afr Language\_ang ... Language\_zho CountsOfReview

count 1.0 1.0 ... 1.0 1.000000

mean 0.0 0.0 ... 0.0 157.634143

std NaN NaN ... NaN NaN

min 0.0 0.0 ... 0.0 157.634143

25% 0.0 0.0 ... 0.0 157.634143

50% 0.0 0.0 ... 0.0 157.634143

75% 0.0 0.0 ... 0.0 157.634143

max 0.0 0.0 ... 0.0 157.634143

[8 rows x 61 columns]

Кластер 2:

Language\_afr Language\_ang ... Language\_zho CountsOfReview

count 1857.0 1857.0 ... 1857.0 1857.000000

mean 0.0 0.0 ... 0.0 199.664786

std 0.0 0.0 ... 0.0 1401.602033

min 0.0 0.0 ... 0.0 0.000000

25% 0.0 0.0 ... 0.0 157.634143

50% 0.0 0.0 ... 0.0 157.634143

75% 0.0 0.0 ... 0.0 157.634143

max 0.0 0.0 ... 0.0 57034.000000

[8 rows x 61 columns]

Розподіл по кожному кластеру для ознак Language та CountsOfReview:

Language\_afr Language\_ang ... Language\_zho CountsOfReview

KMeans\_Cluster ...

0 0.000059 0.000015 ... 0.001128 156.972420

1 0.000000 0.000000 ... 0.000000 157.634143

2 0.000000 0.000000 ... 0.000000 199.664786

[3 rows x 61 columns]

Silhouette Score: 0.7675

Davies-Bouldin Index: 0.1907

Inertia (WSS): 7057888.4644

Перші кілька рядків з класифікацією:

Language CountsOfReview KMeans\_Cluster

1 eng 342.0 0

6 eng 133.0 0

8 eng 222.0 0

13 eng 276.0 0

17 eng 0.0 0

A graph of a diagram

Description automatically generated with medium confidence

Рис.8. Кластеризація по мові та Рев’ю

**Task 4**

Кореляції між мовою та кількості оглядів видавцем і кількості оглядів, автором та кількістю оглядів

**Кореляція та статистичні методи для аналізу взаємозв'язків**

Кореляція — це міра того, як дві змінні взаємопов'язані. Якщо між двома змінними існує сильна кореляція, це означає, що зміна однієї змінної призводить до зміни іншої. Важливі види кореляції включають:

* **Пірсонівська кореляція** — вимірює лінійну залежність між двома змінними. Вона використовується для числових змінних і варіюється від -1 до +1:
  + +1 означає ідеальну позитивну лінійну залежність.
  + -1 означає ідеальну негативну лінійну залежність.
  + 0 означає відсутність лінійної залежності.
* **Спірменівська кореляція** — вимірює монотонну залежність, що є більш гнучким методом для виявлення залежностей між змінними. Вона використовується для обчислення кореляцій між порядковими змінними або для не лінійних, але монотонних зв'язків.
* **Взаємна інформація (Mutual Information)** — міра залежності між двома змінними, яка вимірює спільну інформацію, що знаходиться в цих змінних. Це важливо для виявлення нелінійних залежностей.

**2. Хі-квадрат тест для категоріальних змінних**

Хі-квадрат тест (χ²) використовується для оцінки статистичної залежності між двома категоріальними змінними. Якщо p-значення тесту менше 0.05, це свідчить про значну залежність між змінними.

**3. Графічне представлення даних**

* **Гістограми** — використовуються для візуалізації розподілу числових даних.
* **Countplot** — використовується для візуалізації розподілу категоріальних змінних.
* **Розсіяні графіки** — використовуються для аналізу залежностей між двома змінними.

**Опис програми**

**1. Завантаження та попередня обробка даних**

df = pd.read\_csv('CSV\_BI\_Lab1\_data\_source.csv', sep=';')

df['Language'] = df['Language'].fillna(df['Language'].mode()[0])

df['Authors'] = df['Authors'].fillna(df['Authors'].mode()[0])

df['PublisherNaming'] = df['PublisherNaming'].fillna(df['PublisherNaming'].mode()[0])

Ми завантажуємо дані з CSV-файлу, використовуючи роздільник ;. Потім обробляємо пропущені значення в колонках категоріальних змінних **Language**, **Authors**, і **PublisherNaming**, заповнюючи їх найбільш поширеним значенням за допомогою методу **fillna()**.

**2. Обробка числових даних**

df['Rating'] = df['Rating'].replace({',': '.'}, regex=True).astype(float)

df['CountsOfReviewCategory'] = pd.cut(df['CountsOfReview'], bins=[0, 100, 500, 1000, 5000, 10000, 20000],

labels=["0-100", "101-500", "501-1000", "1001-5000", "5001-10000", "10001-20000"])

У колонці **Rating** ми заміняємо коми на крапки для коректного перетворення значень на тип **float**. Потім створюємо категоріальну змінну **CountsOfReviewCategory**, яка класифікує кількість оглядів за категоріями, щоб полегшити подальший аналіз залежностей.

**3. Заповнення пропущених значень**

imputer = SimpleImputer(strategy='mean')

df[['CountsOfReview', 'Rating']] = imputer.fit\_transform(df[['CountsOfReview', 'Rating']])

З використанням **SimpleImputer** ми заповнюємо пропущені значення в числових колонках **CountsOfReview** та **Rating** середнім значенням.

**4. Хі-квадрат тест**

def chi\_square\_test(column1, column2):

crosstab = pd.crosstab(column1, column2)

chi2, p\_val, \_, \_ = chi2\_contingency(crosstab)

print(f"\nChi-squared test між {column1.name} та {column2.name}:")

print(f"Chi-squared: {chi2}")

print(f"P-value: {p\_val}")

if p\_val < 0.05:

print("Є статистична залежність між цими змінними (р < 0.05).")

else:

print("Немає статистичної залежності між цими змінними (р >= 0.05).")

Функція **chi\_square\_test** здійснює перевірку залежності між двома категоріальними змінними за допомогою **хі-квадрат тесту**. Виводиться значення статистики хі-квадрат і p-значення. Якщо p-значення менше 0.05, ми стверджуємо, що між змінними є статистична залежність.

**5. Оцінка кореляцій між числовими змінними**

correlation\_matrix = df[['CountsOfReview', 'Rating']].corr(method='pearson')

Ми обчислюємо **Пірсонівську кореляцію** між змінними **CountsOfReview** і **Rating** для оцінки лінійної залежності.

spearman\_corr, \_ = spearmanr(df['CountsOfReview'], df['Rating'])

Також обчислюємо **Спірменівську кореляцію**, яка дає оцінку монотонної залежності між двома змінними.

mi = mutual\_info\_score(df['CountsOfReviewCategory'], df['Rating'])

Для оцінки **нелінійних залежностей** між категоріальною змінною **CountsOfReviewCategory** та числовою змінною **Rating** використовується **взаємна інформація**.

**6. Візуалізація результатів**

**Графіки для категоріальних змінних**

sns.countplot(data=df, x='Language', palette='Set2')

Ми будуємо **countplot** для категоріальної змінної **Language**, щоб побачити розподіл книг за мовами.

sns.countplot(data=df, x='Authors', palette='Set2', order=df['Authors'].value\_counts().index[:10])

Побудова графіка для **Authors** дозволяє візуалізувати топ 10 авторів, що зустрічаються в даних.

**Графік для категоріальної змінної CountsOfReviewCategory**

sns.countplot(data=df, x='CountsOfReviewCategory', palette='Set3')

Цей графік показує, як розподіляються книги за категоріями кількості оглядів.

**Розподіл кількості оглядів**

sns.histplot(df['CountsOfReview'], kde=True, color='blue', bins=30)

**Гістограма** для змінної **CountsOfReview** дозволяє побачити, як розподіляються відгуки по різних діапазонах.

**Залежність між кількістю оглядів та рейтингом**

sns.scatterplot(data=df, x='CountsOfReview', y='Rating', color='green')

**Розсіяний графік** показує залежність між кількістю оглядів і рейтингом. Це допомагає зрозуміти, чи є лінійна чи інша залежність між цими двома змінними.

Код програми

import pandas as *pd*import matplotlib.pyplot as *plt*import seaborn as *sns*from sklearn.preprocessing import OneHotEncoder  
from sklearn.impute import SimpleImputer  
from scipy.stats import chi2\_contingency, pearsonr, spearmanr  
from sklearn.metrics import mutual\_info\_score  
import numpy as *np  
  
# Завантаження даних  
df* = pd.read\_csv('CSV\_BI\_Lab1\_data\_source.csv', sep=';')  
  
*# Заповнення пропусків для категоріальних змінних 'Language', 'Authors', 'PublisherNaming'  
df*['Language'] = *df*['Language'].fillna(*df*['Language'].mode()[0]) *# Заповнюємо пропуски найбільш частим значенням  
df*['Authors'] = *df*['Authors'].fillna(*df*['Authors'].mode()[0]) *# Аналогічно для 'Authors'  
df*['PublisherNaming'] = *df*['PublisherNaming'].fillna(  
 *df*['PublisherNaming'].mode()[0]) *# Аналогічно для 'PublisherNaming'  
  
# Перетворення значень у колонці 'Rating' з комами на крапки і перетворення на float  
df*['Rating'] = *df*['Rating'].replace({',': '.'}, regex=True).astype(float)  
  
*# Створюємо категоріальну змінну для 'CountsOfReview' (для аналізу залежностей)  
df*['CountsOfReviewCategory'] = pd.cut(*df*['CountsOfReview'], bins=[0, 100, 500, 1000, 5000, 10000, 20000],  
 labels=["0-100", "101-500", "501-1000", "1001-5000", "5001-10000", "10001-20000"])  
  
  
*# Заповнення пропущених значень для числових колонок (якщо є пропущені значення)  
imputer* = SimpleImputer(strategy='mean')  
*df*[['CountsOfReview', 'Rating']] = *imputer*.fit\_transform(*df*[['CountsOfReview', 'Rating']])  
  
  
*# Функція для проведення хі-квадрат тесту*def chi\_square\_test(*column1*, *column2*):  
 *crosstab* = pd.crosstab(*column1*, *column2*)  
 *chi2*, *p\_val*, *\_*, *\_* = chi2\_contingency(*crosstab*)  
 print(f"\nChi-squared test між {*column1*.name} та {*column2*.name}:")  
 print(f"Chi-squared: {*chi2*}")  
 print(f"P-value: {*p\_val*}")  
 if *p\_val* < 0.05:  
 print("Є статистична залежність між цими змінними (р < 0.05).")  
 else:  
 print("Немає статистичної залежності між цими змінними (р >= 0.05).")  
  
  
*# Перевірка залежностей між кількома парами колонок*chi\_square\_test(*df*['Language'], *df*['CountsOfReviewCategory']) *# Залежність між мовою та категорією кількості оглядів*chi\_square\_test(*df*['Authors'], *df*['CountsOfReviewCategory']) *# Залежність між авторами та категорією кількості оглядів*chi\_square\_test(*df*['PublisherNaming'],  
 *df*['CountsOfReviewCategory']) *# Залежність між видавцем та категорією кількості оглядів  
  
# 1. Pearson Correlation between numerical variables  
correlation\_matrix* = *df*[['CountsOfReview', 'Rating']].corr(method='pearson')  
print(f"\nPearson Correlation Matrix:")  
print(*correlation\_matrix*)  
  
*# 2. Spearman Correlation (for monotonic relationships)  
spearman\_corr*, *\_* = spearmanr(*df*['CountsOfReview'], *df*['Rating'])  
print(f"\nSpearman Correlation between 'CountsOfReview' and 'Rating': {*spearman\_corr*:.4f}")  
  
*# 3. Mutual Information (for non-linear relationships)  
# Заповнюємо пропущені значення перед розрахунком  
df*['CountsOfReviewCategory'] = *df*['CountsOfReviewCategory'].fillna(  
 *df*['CountsOfReviewCategory'].mode()[0]) *# заповнення категоріальної змінної  
mi* = mutual\_info\_score(*df*['CountsOfReviewCategory'], *df*['Rating'])  
print(f"\nMutual Information between 'CountsOfReviewCategory' and 'Rating': {*mi*:.4f}")  
  
*# Побудова графіків для категоріальних змінних*plt.figure(figsize=(10, 6))  
sns.countplot(data=*df*, x='Language', palette='Set2')  
plt.title('Розподіл по мовах')  
plt.ylabel('Кількість книг')  
plt.show()  
  
*# Побудова графіків для авторів*plt.figure(figsize=(10, 6))  
sns.countplot(data=*df*, x='Authors', palette='Set2', order=*df*['Authors'].value\_counts().index[:10]) *# Топ 10 авторів*plt.title('Топ 10 авторів')  
plt.ylabel('Кількість книг')  
plt.xticks(rotation=90)  
plt.show()  
  
*# Побудова діаграми для категоріальної змінної 'CountsOfReviewCategory'*plt.figure(figsize=(10, 6))  
sns.countplot(data=*df*, x='CountsOfReviewCategory', palette='Set3')  
plt.title('Розподіл по категоріях кількості оглядів')  
plt.ylabel('Кількість книг')  
plt.show()  
  
*# Побудова графіків для числових змінних*plt.figure(figsize=(10, 6))  
sns.histplot(*df*['CountsOfReview'], kde=True, color='blue', bins=30)  
plt.title('Розподіл кількості оглядів')  
plt.xlabel('Кількість оглядів')  
plt.ylabel('Частота')  
plt.show()  
  
*# Візуалізація залежності між кількістю оглядів та рейтингом*plt.figure(figsize=(10, 6))  
sns.scatterplot(data=*df*, x='CountsOfReview', y='Rating', color='green')  
plt.title('Залежність між кількістю оглядів та рейтингом')  
plt.xlabel('Кількість оглядів')  
plt.ylabel('Рейтинг')  
plt.show()

Лістинг та результати роботи

A screenshot of a computer

Description automatically generated



A white rectangular object with black lines

Description automatically generated

Рис.9. Графік значної переваги англомовних книг

A graph of different colored bars

Description automatically generated

Рис.10. Рейтинг авторів за кількістю книг

A graph with numbers and a bar

Description automatically generated with medium confidence

Рис.11. Розподіл по кількості оглядів до книг

A graph with numbers and lines

Description automatically generated

Рис.12. Розподіл по кількості оглядів за частотою

A green and black dotted graph

Description automatically generated

Рис.13. Розподіл по кількості оглядів за до рейтингу

**Висновок**

В рамках виконання лабораторної роботи було створено чотири моделі, кожна з яких реалізує різні підходи до аналізу даних та вирішення задач на основі наданих даних. Ось короткий огляд кожної з них:

1. **Модель прогнозування рейтингу на основі розподілу оцінок та кількості відгуків (rating\_predictor\_task.py)**:
   * Використовувалась модель **лінійної регресії** для прогнозування рейтингу книг на основі різних характеристик, таких як розподіл оцінок та кількість відгуків.
   * Важливою частиною моделі є **one-hot кодування** для категоріальної змінної "мова".
   * Модель продемонструвала задовільні результати, зокрема **R2 Score** і **Mean Squared Error**, що дозволяє оцінити ефективність прогнози на тестових даних.
   * Візуалізація фактичних та прогнозованих рейтингів дозволяє краще зрозуміти точність моделі.
2. **Кластеризація книг за допомогою KMeans (rating\_clusterization\_task.py)**:
   * Моделювання було здійснено за допомогою **кластеризації KMeans**, що дозволяє групувати книги в три кластери за розподілом оцінок.
   * Для нормалізації даних використано **StandardScaler**, що дозволяє привести всі змінні до одного масштабу.
   * Оцінка **Silhouette Score**, **Davies-Bouldin Index** та **Inertia** дають змогу оцінити якість кластеризації, що дозволяє вибрати найбільш оптимальну кількість кластерів.
   * Візуалізація кластерів із зазначенням центрів кластерів на графіку дозволяє чітко уявити, як книги групуються за їхніми характеристиками.
3. **Кластеризація за допомогою KMeans та аналіз залежностей між категорією оглядів, мовою та іншими параметрами (lang\_review\_clusterization\_task.py)**:
   * Вибір кластерів був також виконаний за допомогою **KMeans**, але цього разу додатково враховувались категоріальні змінні (мова книг).
   * Використання **PCA** для зменшення вимірності до 2D дозволяє створити наочніші графіки для кластерів.
   * Виявлено, що **залежність між кількістю оглядів і рейтингом** може бути виражена за допомогою різних методів кореляції, таких як **Pearson**, **Spearman** і **Mutual Information**.
   * Ця модель добре ілюструє, як кластеризація може бути використана для подальшого аналізу категорій книг за різними ознаками.
4. **Аналіз кореляцій та статистичних залежностей (correlation\_task.py)**:
   * В цій моделі проведено **хі-квадрат тест**, щоб визначити статистичні залежності між категоріями змінних (мова, автори, видавці) та категорією кількості відгуків.
   * Використовувались також **пірсонівська кореляція**, **спірменівська кореляція** та **взаємна інформація**, щоб оцінити зв'язок між кількістю відгуків і рейтингом.
   * Моделювання також включало побудову різноманітних графіків для візуалізації розподілу книг за мовами, авторами, категоріями відгуків та ін.
   * Це дозволило глибше аналізувати не тільки числові залежності, але й категоріальні, що дає змогу приймати обґрунтовані рішення про подальші аналізи.

**Загальний висновок:**

Всі чотири моделі виконують важливі завдання для аналізу та прогнозування даних про книги, а саме:

1. **Прогнозування рейтингів** за допомогою лінійної регресії.
2. **Кластеризація книг** на основі різних характеристик, що дозволяє групувати книги за схожими властивостями.
3. **Аналіз залежностей між різними параметрами**, такими як мова, автори та видавці.
4. **Кореляційний аналіз** для оцінки зв'язків між кількістю оглядів і рейтингами, а також для виявлення статистичних залежностей між різними змінними.

Усі моделі продемонстрували потенціал у розв'язанні конкретних бізнес-аналітичних задач, що може бути корисно для подальших досліджень, таких як сегментація ринку, прогнозування популярності книг і визначення важливих факторів, які впливають на рейтинг книг.

**Джерела**

1. **"Hands-On Regression Analysis with R: Build Effective Models to Solve Real-World Problems"** — Абдулл Ахмад, 2020.
2. **"Pattern Recognition and Machine Learning"** — Christopher M. Bishop, 2006.
3. **"Data Science for Business: What You Need to Know about Data Mining and Data-Analytic Thinking"** — Foster Provost, Tom Fawcett, 2013.
4. **"Applied Predictive Modeling"** — Max Kuhn, Kjell Johnson, 2013.
5. **"Elements of Statistical Learning: Data Mining, Inference, and Prediction"** — Trevor Hastie, Robert Tibshirani, Jerome Friedman, 2009.