Форма № 24

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій
(повне найменування факультету)

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)
магістр
(ступінь вищої освіти)

на тему ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ МЕТОДІВ АНАЛІЗУ ПРИБУТКІВ ТА РЕЙТИНГУ ФІЛЬМІВ НА ОСНОВІ ДАНИХ З ІНДУСТРІЇ КІНО
RESEARCH AND SOFTWARE IMPLEMENTATION OF METHODS FOR ANALYZING FILM REVENUE AND RATINGS BASED ON THE FILM INDUSTRY DATA

Виконав(ла): студент(ка) 6 курсу, групи КНТ-214м
Спеціальності 122 Комп'ютерні науки
(код і найменування спеціальності)

Освітня програма (спеціалізація)
Системи штучного інтелекту

СТИЧУК М.О.
(ПРІЗВИЩЕ та ініціали)
Керівник ЛЕОЩЕНКО С.Д.
(ПРІЗВИЩЕ та ініціали)
Рецензент ПОЛЯКОВ М.О.
(ПРІЗВИЩЕ та ініціали)

2025

Форма № 25

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет КНТ
Кафедра програмних засобів
Ступінь вищої освіти магістр
Спеціальність 122 Комп'ютерні науки
(код і найменування)
Освітня програма (спеціалізація) Системи штучного інтелекту
(назва освітньої програми (спеціалізації)

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
_____ Сергій СУББОТІН
"____" _____2025 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

СТИЧУК Маргарити Олегівни
(ПРІЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Дослідження та програмна реалізація методів аналізу прибутків та рейтингу фільмів на основі даних з індустрії кіно. Research and Software Implementation of Methods for Analyzing Film Revenue and Ratings Based on the Film Industry Data

керівник проєкту (роботи) д-р філос., доцент, ЛЕОЩЕНКО Сергій Дмитрович ,
(науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім'я, по батькові)
затверджені наказом закладу вищої освіти від " 30 " вересня 2025 року № 447
2. Строк подання студентом проєкту (роботи) 16 грудня 2025 року
3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз проблеми та постановка завдань дослідження. 2. Матеріали і методи. 3. Розробка архітектури програми. 4. Розробка програми. 5. Експлуатація, тестування та експериментальне дослідження програми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-5 Основна частина	ЛЕОЩЕНКО С.Д., доцент		
Нормоконтролер	КАЛІНІНА М.В., асистент		

7. Дата видачі завдання “ 30 ” вересня 2025 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	2-3 тижні	Розділ 1
3	Розробка та удосконалення методів, моделей й алгоритмів	4-5 тижні	Розділ 2
	вирішення задачі.		
4	Розробка архітектури програми.	6 тиждень	Розділ 3
5	Розробка програми.	7-8 тижні	Розділ 4
6	Тестування та експериментальне дослідження	9 тиждень	Розділ 5
	програмного забезпечення.		
7	Оформлення пояснювальної записки та документів до неї.	10-11 тижні	Додатки
8	Нормоконтроль та рецензування.	12 тиждень	
9	Захист роботи.	12 тиждень	

Студент(ка) _____ Маргарита СТИЧУК
(підпис) (Імя ПРІЗВИЩЕ)

Керівник проєкту (роботи) _____ Сергій ЛЕОЩЕНКО
(підпис) (Імя ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра: 176 с., 3 табл., 162 рис., 3 дод., 39 джерел.

КІНОІНДУСТРІЯ, ПРИБУТОК, РЕЙТИНГ, ШТУЧНИЙ ІНТЕЛЕКТ, МАШИННЕ НАВЧАННЯ, НЕЙРОННІ МЕРЕЖІ, API, AWS, ВІЗУАЛІЗАЦІЯ.

Об’єкт дослідження – процес аналізу прибутків та рейтингу фільмів.

Предмет дослідження – програмні засоби для аналізу прибутків та рейтингу фільмів.

Мета роботи – підвищення точності прогнозування майбутньої прибутковості кінострічок.

Матеріали, методи та технічні засоби: мова програмування – Python, середовище розробки – PyCharm. Основні використані бібліотеки: kagglehub, requests, pandas, scikit-learn, tensorflow, matplotlib, streamlit, altair, numpy, boto3, dotenv. Допоміжні засоби і програми: AWS S3.

Результати. У ході виконання роботи було розроблено програмне забезпечення для аналізу прибутків та рейтингу фільмів на основі даних з індустрії кіно.

Висновки. Розроблено програмне забезпечення, яке отримує дані з різних джерел за допомогою API запитів, формує набір даних, аналізує патерни вподобань глядачів, залежність прибутків і рейтингу від цього, шукає зв’язок між характеристиками фільму, проводить прогнозування, візуалізує результати і вивантажує дані у хмарне сховище.

Галузь використання – аналіз патернів у фільмах та уподобань користувачів, а також як це впливає на рейтинги і прибутки у кіноіндустрії.

Економічна ефективність. При використанні розробленої програми у користувачів є можливість безкоштовно отримати аналіз фільмів і уподобань

глядачів на основі даних, які можна також безкоштовно отримати з хмарного сховища.

Термін окупності роботи – 1 рік 8 міс.

ABSTRACT

Explanatory note to the diploma qualifying work of the master: 176 pages, 3 tables, 162 figures, 3 appendixes, 39 sources.

FILM INDUSTRY, REVENUE, RATING, ARTIFICIAL INTELLIGENCE, MACHINE LEARNING, NEURAL NETWORKS, API, AWS, VISUALIZATION.

Object of study is the process of analyzing film revenue and ratings.

Subject of study are software tools for analyzing film revenue and ratings.

The purpose of the work is increasing the accuracy of forecasting the future profitability of films.

Materials, methods, and tools: Python programming language, PyCharm development environment. Main libraries used: kagglehub, requests, pandas, scikit-learn, tensorflow, matplotlib, streamlit, altair, numpy, boto3, dotenv. Tools and applications: AWS S3.

Results. In the course of the research, software for analyzing film revenue and ratings was developed.

Conclusions. A software that retrieves data from various sources using API queries, forms a dataset, analyzes viewer preference patterns, revenue and rating dependencies from this, looks for correlations between film characteristics, makes predictions, visualizes the results, and uploads the data to cloud storage has been developed.

The field of use is the analysis of patterns in movies, user preferences, and how this affects ratings and revenues in the film industry.

Economic efficiency. When using the developed software, users have the opportunity to get free movie analysis and viewer preferences based on data that can also be obtained free of charge from cloud storage.

The payback period is 1 year 8 months.

ЗМІСТ

С.

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API – Application Programming Interface;

AWS – Amazon Web Services;

IMDB – Internet Movie Database;

ROI – Return On Investment;

S3 – Simple Storage Service;

ПЗ – програмне забезпечення;

ШІ – штучний інтелект.

ВСТУП

Індустрії кіно на сьогоднішній день вже більше 100 років. Від високобюджетних блокбастерів до короткометражних і анімаційних фільмів, які отримали Оскар, кіноіндустрія розширилася від скромного початку до гіганта. Фільми сьогодні – це не лише розвага для мільйонів людей або витвори мистецтва, а ще й багато мільярдна індустрія, кожна помилка в якій може коштувати компаніям і інвесторам грошей та суспільної підтримки [1].

Тому актуальною є задача розробки ПЗ, що дозволить провести аналіз прибутків та рейтингів фільмів з подальшою можливістю прогнозування.

Метою є розробка ПЗ для аналізу і прогнозування прибутку та рейтингу фільмів на основі даних з індустрії кіно.

Для досягнення поставленої мети в роботі потрібно вирішити такі задачі:

– виконати дослідження предметної області;

– проаналізувати терміни і області штучного інтелекту, машинного навчання, нейронних мереж, API та AWS:

– дослідити використання штучного інтелекту у кіноіндустрії;

– дослідити хмарну інтеграцію за допомогою AWS;

– дослідити можливість завантаження даних зі сторонніх сервісів за допомогою API;

– розробити програму для аналізу прибутку та рейтингу фільмів з подальшою можливістю прогнозування.

Об’єкт дослідження – процес аналізу прибутку та рейтингу фільмів з подальшою можливістю прогнозування.

Предмет дослідження – програмні засоби для аналізу даних з індустрії кіно з подальшою можливістю прогнозування.

Методи дослідження. Для розробки ПЗ застосовані методи машинного навчання, інтелектуального аналізу даних, нейронні мережі, сторонні сервіси

для отримання даних за допомогою API та хмарна інтеграція за допомогою AWS.

Наукова новизна одержаних результатів. Розроблено ПЗ для аналізу прибутку і рейтингу фільмів з подальшою можливістю прогнозування, яке вбирає в себе можливості детального аналізу, візуалізації, прогнозування та можливість легко завантажити набір даних.

Практичне значення одержаних результатів полягає у тому, що розроблене ПЗ для аналізу прибутків і рейтингів кіно з подальшою можливістю прогнозування може бути застосовано у сфері кіноіндустрії для аналізу реального прикладу фільму, розуміння патернів всієї індустрії і уподобань глядачів.

1 АНАЛІЗ ПРОБЛЕМИ ТА ПОСТАНОВКА ЗАВДАНЬ ДОСЛІДЖЕННЯ

Перший розділ буде присвячено детальному аналізу предметної області – індустрії кіно.

1.1 Аналіз індустрії кіно

Виробництво привабливого та прибуткового фільмового контенту є головним пріоритетом для довгострокового процвітання кіноіндустрії. Досягнення в цифрових технологіях, збільшення доступності великих даних, швидке поширення аналітичних методів та посилена конкуренція з боку контенту, що створюється користувачами, створили безпрецедентні вимоги для продюсерів фільмів, що є приводом для використання аналітики в виробництві контенту [2].

Прогнозування фільму є важливим способом прогнозування доходів та популярності фільму. Деякі з критеріїв при розрахунку успіху фільму, наприклад, можуть включати бюджет, акторів, режисера, продюсера, місця постановки, сценариста, день виходу фільму, конкуруючі випуски фільму в той же час, музику, місце випуску та цільову аудиторію. Аналіз даних про фільми може бути неймовірно потужним і може робити обґрунтовані припущення, але не може визначити долю окремого проекту з абсолютною впевненістю. Для деяких фільмів успіх буде продажем квитків, для інших - маржа прибутку, огляди, соціальні розмови, франчайзингові варіанти або нагороди критиків [3].

Глобальне виробництво кіно досягає історичних висот і перевищує рівень до пандемії(тоді виробництво художніх фільмів впало на 40%), що означає значний підйом для індустрії. У 2023 році кінорежисери випустили 9511 фільмів у порівнянні з лише 5656 у 2020 році та 9328 у 2019 році - 68% зростання від мінімуму пандемії та 2% зростання від попереднього піку. Це представлено на рисунку 1.1. Ці дані ґрунтуються на нових даних WIPO та

Omdia, які будуть представлені в майбутньому Глобальному індексі інновацій 2025 року.

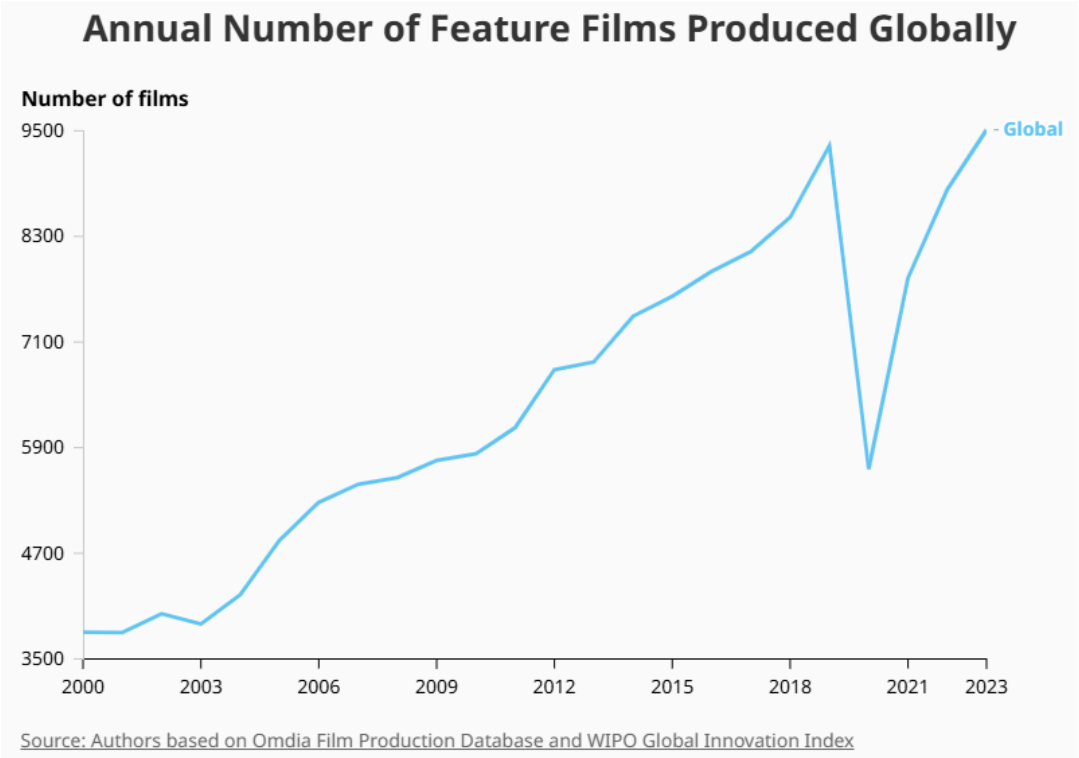


Рисунок 1.1 – Річна кількість випущених фільмів по всьому світу [4]

На рисунку 1.2 представлено кількість випущених фільмів по країнах. Наприклад, Індія залишається світовим лідером у виробництві фільмів, виробляючи понад 2500 фільмів, що більш ніж втричі перевищує виробництво другого найбільшого виробника.

Індія міцно зберігає свої позиції провідного світового кінопродюсера з більш ніж 2500 фільмами. Китай займає друге місце з майже 800 фільмами. Це перший випадок в історії кінематографа, коли Китай обігнав як Японію, так і Сполучені Штати з точки зору виробництва фільмів. Японія займає третє місце другий рік поспіль, з 676 фільмами, що відображає стабільне зростання виробництва. США, які зайняли друге місце в 2022 році, опустилися на четверте місце з 510 фільмами. Іспанія тримається на п'ятій позиції і першої в Європейському Союзі вже третій рік поспіль [4].

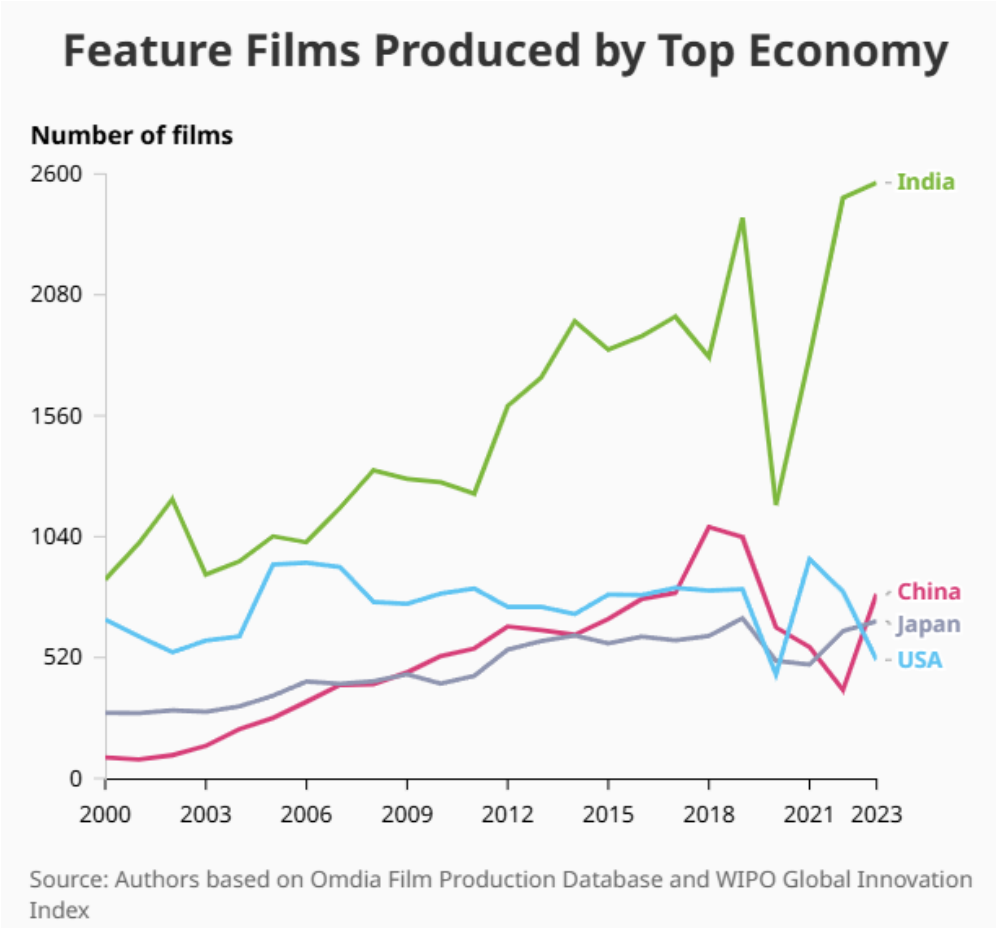


Рисунок 1.2 – Кількість випущених фільмів по країнах [4]

Також важливим є, наприклад, сам жанр фільму, тому що статистично люди ходять на бойовики більше. Станом на 2022 рік, згідно зі статистикою кіноіндустрії, найпопулярнішим жанром серед випущених фільмів був бойовик із доходом у розмірі 3 983 514 713 доларів США та часткою ринку 53,43%. На фільми цієї категорії було продано близько 434 407 251 квитків. Нижче на рисунку 1.3 представлено графік доходів по жанрах за 2022 рік.

Також, сам тип проєкту має значення. Станом на 2022 рік, статистика кіноіндустрії показує, що категорія живих фільмів була найкасовішим типом створення фільмів, що призвело до загального доходу в розмірі 5 750 204 733 доларів США та продажу 6 27 066 832 квитків на внутрішньому ринку в цьому жанрі. Це представлено на рисунку 1.4 [5].

Market Search For Each Genre in 2022

	Genre	Movies	2022 Gross	Tickets	Share
1	Action	54	$3,983,514,713	434,407,251	53.44%
2	Adventure	28	$980,219,074	106,894,112	13.15%
3	Comedy	60	$691,114,062	75,366,828	9.27%
4	Horror	39	$634,295,308	69,170,685	8.51%
5	Drama	146	$620,493,407	67,665,521	8.32%
6	Thriller/Suspense	46	249,909,212	27,252,892	3.35%
7	Romantic Comedy	11	$105,384,522	11,492,308	1.41%
8	Musical	6	$61,505,843	6,707,287	0.83%
9	Reality	1	$57,743,451	6,296,996	0.77%
10	Black Comedy	3	$44,614,220	4,865,235	0.60%

Source: Enterprise Apps Today

Рисунок 1.3 – Графік доходів по жанру за 2022 рік [5]

Market Share for Each Production Method in 2022

	Production Method	2022 Gross	Tickets	Share
1	Live Action	5,749,887,671	$627,032,263	77.13%
2	Digital Animation	1,005,999,359	$109,705,483	13.49%
3	Animation/Live Action	690,143,456	$75,261,008	9.26%
4	Stop-Motion Animation	6,333,703	$690,698	0.08%
5	Multiple Production Methods	1,802,918	$196,609	0.02%
6	Hand Animation	106,946	$11,661	0.00%

Source: Enterprise Apps Today

Рисунок 1.4 – Графік доходів за типом створення фільму [5]

З цього можна зробити висновок, що кіноіндустрія поступово оговтується від пандемії і виробництво фільмів продовжує і буде продовжувати рости в наступні роки. Саме тому аналіз уподобань глядачів та глобальних трендів є як ніколи важливим для цієї індустрії.

1.2 Штучний інтелект

Спочатку потрібно розібрати саме поняття штучного інтелекту, щоб зрозуміти як воно може допомогти в нашому проєкті.

Говорячи простими словами, штучний інтелект (ШІ) – це технологія, яка дозволяє комп’ютерам імітувати людське мислення, навчання, приймання рішень, автономність та креативність.

Для простішого розуміння нижче представлено рисунок 1.5, на якому відображено розвиток концепцій ШІ, машинного навчання, глибокого навчання та генеративного інтелекту. В цій роботі буде використано машинне навчання і нейронні мережі, які ми й розберемо нижче [6].

Концепцію ШІ було створено у 1950-х роках. У 1950 році Алан Тьюрінг опублікував свою роботу «Комп’ютерні Машини та Інтелект», яка потім стала відомою, як Тест Тьюринга, які експерти використовували для вимірювання інтелекту комп’ютерів. У 1952 році комп’ютерний вчений Артур Семюель розробив програму для гри в шашки, яка була першою такою, що грала самостійно. У 1955 році Джон Маккарті провів семінар у Дартмуті на тему “Штучний інтелект”, що є першим таким використанням слова. Термін “штучний інтелект” почав популяризуватись з тих часів [7].

1.3 Машинне навчання

Розібравши більш ширше поняття ШІ, можна перейти по визначення машинного навчання.

Машинне навчання (machine learning) – це окрема галузь ШІ, що зосереджена на методах, які можуть вивчати закономірності навчальних даних і, згодом, робити точні висновки щодо нових даних. Ця здатність розпізнавання закономірностей дозволяє моделям машинного навчання приймати рішення або прогнозувати без явних, жорстко кодованих інструкцій.

Центральна концепція роботи машинного навчання полягає в тому, що якщо оптимізувати продуктивність моделі на наборі даних завдань, які адекватно нагадують реальні проблеми, для яких вона буде використовуватися, за допомогою процесу, який називається навчанням моделі, модель може робити точні прогнози щодо нових даних, які вона бачить у своєму кінцевому випадку використання [8].

Навчання моделі – це крок машинного навчання, на якому відбувається навчання, і яке є найважливішим кроком у життєвому циклі моделі. У машинному навчанні навчання включає коригування параметрів моделі. Ці параметри включають ваги та зміщення в математичних функціях, що складають їхні алгоритми. Метою цього коригування є отримання точних результатів. Конкретні значення цих ваг та зміщень, які є кінцевим результатом навчання моделі, є відчутним проявом «знань» моделі.
Математично метою цього навчання є мінімізація функції втрат, яка кількісно визначає похибку виходів моделі на навчальних запитах. Коли вихід функції втрат падає нижче певного заздалегідь визначеного порогу, тобто похибка моделі на навчальних завданнях достатньо мала, модель вважається навченою.

Цей робочий процес триває ітераційно, доки не будуть досягнуті задовільні результати. Адекватне навчання може також вимагати коригування гіперпараметрів структурних виборів, які впливають на процес навчання, але самі по собі не є навчальними, у процесі, який називається налаштуванням гіперпараметрів [9].

1.3.1 Життєвий цикл машинного навчання

На рисунку 1.5 представлено типовий життєвий цикл машинного навчання.

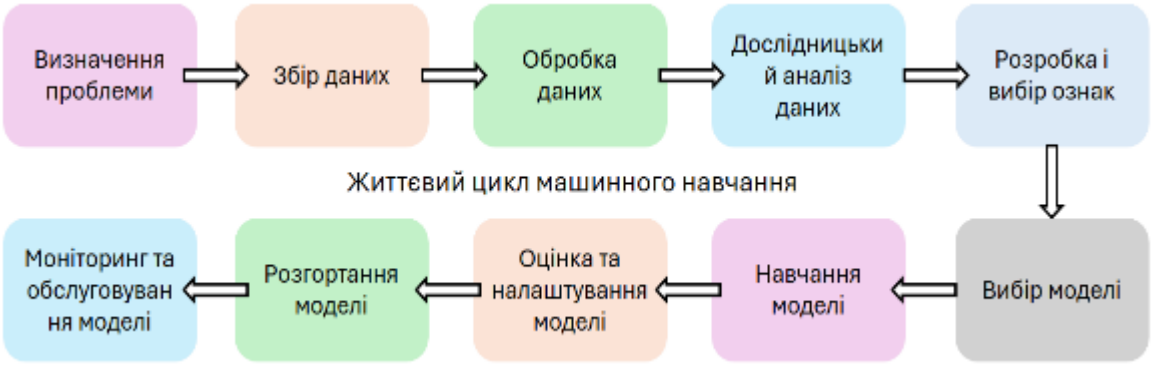


Рисунок 1.5 – Життєвий цикл машинного навчання [10]

Першим кроком є визначення та чітке визначення бізнес-проблеми (Problem Statement). Добре сформульована проблема забезпечує основу для всього життєвого циклу. На цьому етапі ретельно розробляються такі важливі речі, як цілі проекту, бажані результати та обсяг завдання.

Друга фаза починається зі збору даних (Data Collection), будь то числа, зображення чи текст. Це так звані навчальні дані, і чим більше даних зібрано, тим кращою буде програма. Вона включає в себе систематичний збір наборів даних, які можна використовувати як необроблені дані для навчання моделі. Якість та різноманітність даних безпосередньо впливають на продуктивність моделі.

Ось деякі основні характеристики для збору даних:

- релевантність: зібрані дані повинні бути релевантними до визначеної проблеми;
- якість: забезпечення якості даних шляхом врахування таких факторів, як точність та етичне використання;
- кількість: збір достатнього обсягу даних для навчання надійної моделі;
- різноманітність: включення різноманітних наборів даних для охоплення широкого спектру сценаріїв та закономірностей;

Третім кроком відбувається очищення та попередня обробка даних (Data Cleaning and Preprocessing). Необроблені дані часто бувають неохайними та неструктурованими, і якщо ми використовуємо ці дані безпосередньо для

навчання, це може призвести до низької точності. Нам потрібно виконати очищення та попередню обробку даних, що часто включає:

- очищення даних: вирішення таких проблем, як відсутні значення, викиди та невідповідності в даних;
- попередня обробка даних: стандартизація форматів, масштабування значень та кодування категоріальних змінних для узгодженості;
- якість даних: забезпечення належної організації та підготовки даних до змістовного аналізу.

Четвертий крок – дослідницький аналіз даних (Exploratory Data Analysis), який використовується для пошуку закономірностей та характеристик, прихованих у даних. Під час цього аналізу надаються закономірності, тенденції та розуміння, які можуть бути невидимими неозброєним оком. Ось основні характеристики дослідницького аналізу даних:

- дослідження: використовують статистичні та візуальні інструменти для дослідження закономірностей у даних;
- закономірності та тенденції: визначають основні закономірності, тенденції та потенційні проблеми в наборі даних;
- аналіз: отримують цінну інформацію для прийняття обґрунтованих рішень на пізніших етапах;
- прийняття рішень: цей аналіз використовуйте для розробки ознак та вибору моделі.

П’ятим кроком є розробка та вибір ознак (Feature Engineering & Selection).

Розробка та вибір ознак – це трансформаційний процес, який включає вибір лише релевантних ознак для підвищення ефективності та прогнозування моделі, одночасно зменшуючи складність. Ось основні особливості розробки та вибору ознак:

- розробка ознак: створення нових ознак або трансформація існуючих для кращого відображення закономірностей та зв'язків;

- вибір ознак: визначення підмножини ознак, які найбільше впливають на продуктивність моделі;
- експертиза предметної області: використання знань предметної області для розробки ознак, які роблять значний внесок у прогнозування;
- оптимізація: збалансування набору ознак для точності та мінімізації обчислювальної складності.

Шостий крок – вибір моделі (Model Selection). Для створення гарної моделі машинного навчання вибір моделі є дуже важливою частиною, оскільки нам потрібно знайти модель, яка відповідає визначеній нами проблемі, характеру даних, складності проблеми та бажаним результатам. Ось основні характеристики вибору моделі:

- складність: треба враховувати складність проблеми та характер даних під час вибору моделі;
- фактори прийняття рішень: треба оцінити такі фактори, як продуктивність, інтерпретованість та масштабованість під час вибору моделі;
- експериментування: є сенс в експериментуванні з різними моделями, щоб знайти найкращу для вирішення проблеми.

Крок сім – навчання моделі (Model Training). З вибраною моделлю життєвий цикл машинного навчання переходить до процесу навчання моделі. Цей процес включає в себе доступ до історичних даних, що дозволяє їй вивчати закономірності, зв'язки та залежності в наборі даних. Ось основні характеристики навчання моделі:

- ітеративний процес: навчання моделі ітеративно, коригування параметрів для мінімізації помилок та підвищення точності;
- оптимізація: точне налаштування моделі для оптимізації її прогностичних можливостей;
- валідація: ретельне навчання моделі для забезпечення точності до нових невидимих даних.

Крок вісім – оцінка та налаштування моделі (Model Evaluation and Tuning). Оцінка моделі включає ретельне тестування на основі валідації або

тестових наборів даних для перевірки точності моделі на нових невидимих даних. Це дає уявлення про сильні та слабкі сторони моделі. Якщо модель не досягає бажаного рівня продуктивності, нам може знадобитися повторне налаштування моделі та коригування її гіперпараметрів для підвищення точності прогнозування. Ось основні характеристики оцінки та налаштування моделі:

- метрики оцінювання: такі метрики, як точність, прецизійність, повнота та F1-оцінка, допоможуть у оцінці продуктивності моделі;
- сильні та слабкі сторони: визначте сильні та слабкі сторони моделі за допомогою ретельного тестування;
- надійність моделі: ітеративне налаштування для досягнення бажаних рівнів стійкості та надійності моделі.

Крок дев’ять – розгортання моделі (Model Deployment). Тепер модель готова до розгортання для реального застосування. Це передбачає інтеграцію прогнозної моделі з існуючими системами, що дозволяє бізнесу використовувати це для прийняття обґрунтованих рішень. Ось основні функції розгортання моделі:

- інтеграція з існуючими системами;
- забезпечення прийняття рішень за допомогою прогнозів;
- забезпечення масштабованості та безпеки розгортання;
- надання API або конвеєрів для використання у виробництві.

Крок десять – моніторинг та обслуговування моделі (Model Monitoring & Maintenance). Після розгортання моделі необхідно контролювати, щоб забезпечити їхню належну роботу з часом. Регулярне відстеження допомагає виявити дрейф даних, падіння точності або зміну шаблонів, а для забезпечення надійності моделі в реальному світі може знадобитися повторне навчання.

Кожен з цих кроків є важливим для побудови успішної моделі машинного навчання, яка може надати цінну інформацію та прогнози [10].

1.3.2 Типи машинного навчання

Машинне навчання в основному поділяється на три основні типи:

- навчання з учителем;
- навчання без учителя;
- навчання з підкріпленням.

Також виділяють два додаткові типи: напівнавчене навчання та навчання з самонавчанням.

Навчання з учителем навчає моделі на маркованих даних для прогнозування або класифікації нових, невидимих даних. Алгоритми навчання з учителем зазвичай поділяються на два основні типи:

- класифікація: метою є прогнозування дискретних міток або категорій;
- регресія: метою є прогнозування неперервних числових значень.

На рисунку 1.6 представлено схему навчання з вчителем. Вхідними даними є набір маркованих даних, а також окремо набір цих міток. Алгоритм вчиться та виділяє особливості кожного класу і потім робить прогноз на наборі немаркованих даних.

Рисунок 1.6 – Навчання з вчителем [11]

Навчання без учителя знаходить закономірності або групи в немаркованих даних та поділяється на три основні категорії залежно від його призначення:

- кластеризація: групування точок даних у кластери на основі їхньої подібності або відмінностей;
- видобуток правил асоціації: пошук закономірностей між елементами у великих наборах даних;
- зменшення розмірності: спрощення набору даних, зменшивши кількість ознак, зберігаючи при цьому найважливішу інформацію.

На рисунку 1.7 представлено схему навчання без вчителя. Вхідними даними є немаркований набір даних, на основі якого буде вчитись наш алгоритм і після знаходження закономірностей або конкретних (на його думку) груп даних він зробить прогноз на новому наборі даних.

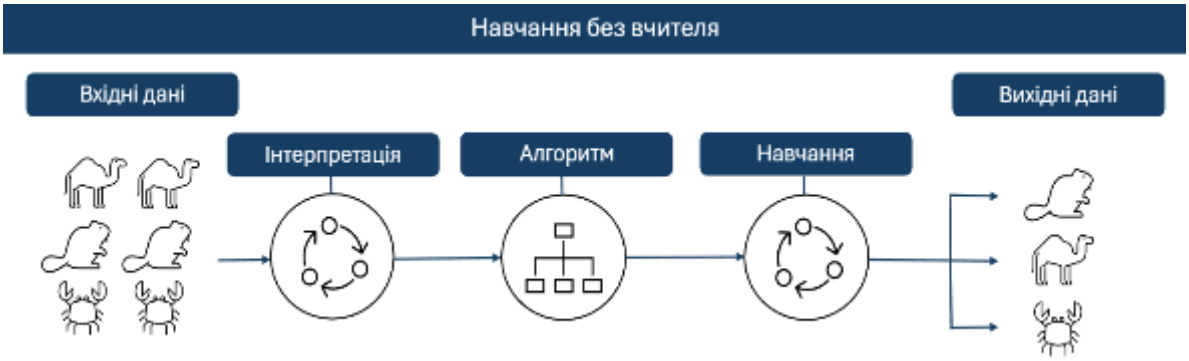


Рисунок 1.7 – Навчання без вчителя [11]

Навчання з підкріпленням – це алгоритм, що навчається методом спроб і помилок для максимізації винагород, ідеально підходить для завдань прийняття рішень.

На рисунку 1.8 представлено схему навчання з підкріпленням. Воно відрізняється тим, що працює над досягненням поставленої мети, а не досліджує дані для виявлення будь-яких закономірностей, які можуть існувати. Маючи на увазі певну мету, алгоритм діє методом спроб і помилок. Кожен хід отримує позитивний, негативний або нейтральний зворотний

зв'язок, який алгоритм використовує для вдосконалення загального процесу прийняття рішень. Алгоритми навчання з підкріпленням можуть працювати на макрорівні над досягненням мети проекту, навіть якщо це означає боротьбу з короткостроковими негативними наслідками. Навчання комп'ютера грі в шахи є гарним прикладом. Загальна мета – виграти гру, але це може вимагати жертвування фігурами в міру того, як гра триває [11].

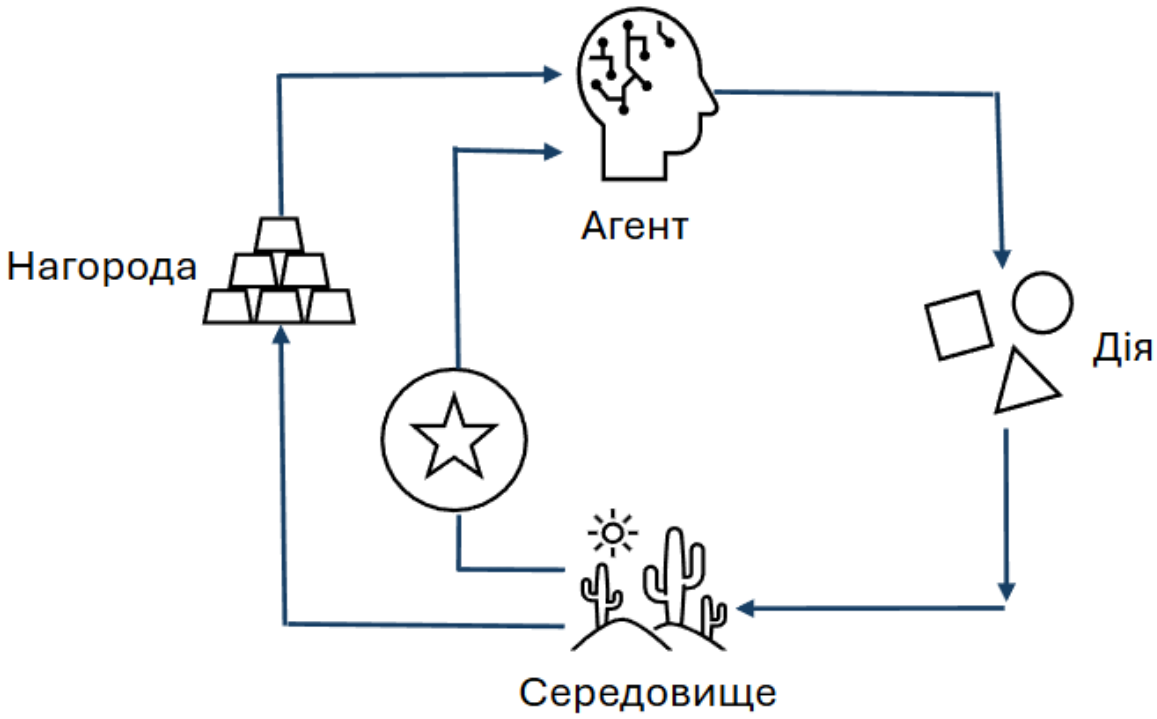


Рисунок 1.8 – Схема навчання з підкріпленням [11]

Загалом серед переваг машинного навчання можна виділити таки ключові моменти:

- оптимізація прийняття рішень та прогнозного аналізу: рішення на основі даних починаються з аналізу даних. Це очевидне твердження, але коли процес аналізу виконується вручну, він є трудомістким та ресурсомістким і може не дати достатньо глибоких висновків, щоб виправдати витрати. Машинне навчання може проаналізувати великі обсяги даних, щоб виявити тенденції та закономірності, щоб користувачі могли зосередитися на запитах та практичних результатах, а не на оптимізації ручної обробки даних;

- підвищення ефективності та автоматизація завдань: машинне навчання є основою багатьох технологій, які підвищують ефективність роботи працівників. Багато низькокогнітивних, повторюваних завдань, включаючи перевірку орфографії, а також оцифрування та класифікацію документів, тепер виконуються комп'ютерами завдяки машинному навчанню. Поєднуючи технології машинного навчання, можна робити прогнози на основі даних із поясненнями факторів, що вплинули на прогноз, допомагаючи керівникам намітити найкращі шляхи для своїх організацій;
- персоналізація та інновації в послугах: машинне навчання відкрило нові двері для взаємодії з клієнтами завдяки персоналізації. Історія покупок, історія переглядів, демографічні дані та додаткова інформація можуть бути використані для створення індивідуального профілю клієнта, який потім можна порівняти з аналогічними профілями, щоб робити прогнози щодо інтересів клієнтів. Це дозволяє використовувати системи пропозицій, автоматично генеровані знижки та інші види персоналізованої взаємодії, щоб клієнти поверталися.

До недоліків машинного навчання можна віднести такі:

- залежність від якості даних: якість даних є критично важливою як на етапі навчання, так і у виробництві. Високоякісні дані можуть призвести до точніших результатів, отриманих своєчасно та ефективно; низькоякісні дані можуть створювати неточності та спотворення в результуючих моделях;
- упередженість: дані можуть бути чистими, але чи вільні вони від упередженості? Як очевидний випадок, припустимо, ви хочете навчити систему машинного навчання розпізнавати собак на зображеннях, і у вас є надійний набір даних, що складається лише з фотографій лабрадорів та пуделів. Після навчання модель чудово розпізнає цих собак – можна сказати, що вона упереджена. Але коли їй показують зображення бульдога, вона каже, що не може знайти собаку. Створення правильного набору навчальних даних – один із найскладніших і найдорожчих аспектів створення інструментів машинного навчання, які працюють так, як ви хочете;

– безпека та конфіденційність даних: машинне навчання може створювати низку проблем безпеки. Дані, що використовуються в аналізі машинного навчання, можуть містити чутливу інформацію, не призначену для публічного використання. Щоб допомогти зменшити проблеми безпеки, компанії повинні використовувати низку політик, процедур та засобів контролю безпеки, включаючи практичне навчання персоналу [12].

1.4 Глибоке навчання

Глибоке навчання (deep learning) – це підмножина машинного навчання, що керується багатошаровими нейронними мережами, дизайн яких натхненний структурою людського мозку.

На відміну від явно визначеної математичної логіки традиційних алгоритмів машинного навчання, штучні нейронні мережі моделей глибокого навчання складаються з багатьох взаємопов'язаних шарів «нейронів», кожен з яких виконує математичну операцію. Використовуючи машинне навчання для регулювання сили зв'язків між окремими нейронами в сусідніх шарах, іншими словами, змінюючи ваги та зміщення моделі, мережу можна оптимізувати для отримання точніших результатів. Хоча нейронні мережі та глибоке навчання стали нерозривно пов'язаними одне з одним, вони не є точними синонімами: «глибоке навчання» стосується навчання моделей щонайменше з 4 шарами (хоча сучасні архітектури нейронних мереж часто набагато «глибші»).

На рисунку 1.9 представлено структуру стандартної глибокої нейронної мережі з 3 прихованими шарами. Штучні нейронні мережі складаються з взаємопов'язаних шарів штучних «нейронів» (або вузлів), кожен з яких виконує власну математичну операцію (так звану «функцію активації»). Існує багато різних функцій активації; нейронна мережа часто включає кілька функцій активації у свою структуру, але зазвичай усі нейрони в даному шарі мережі налаштовані на виконання однієї й тієї ж функції активації. У більшості нейронних мереж кожен нейрон у вхідному шарі з'єднаний з кожним з

нейронів у наступному шарі, які самі по собі з'єднані з нейронами в наступному шарі тощо. Вихід функції активації кожного вузла вносить частину вхідних даних, що надаються кожному з вузлів наступного шару. Найважливіше те, що функції активації, що виконуються в кожному вузлі, є нелінійними, що дозволяє нейронним мережам моделювати складні закономірності та залежності. Саме використання нелінійних функцій активації відрізняє глибоку нейронну мережу від (дуже складної) моделі лінійної регресії.

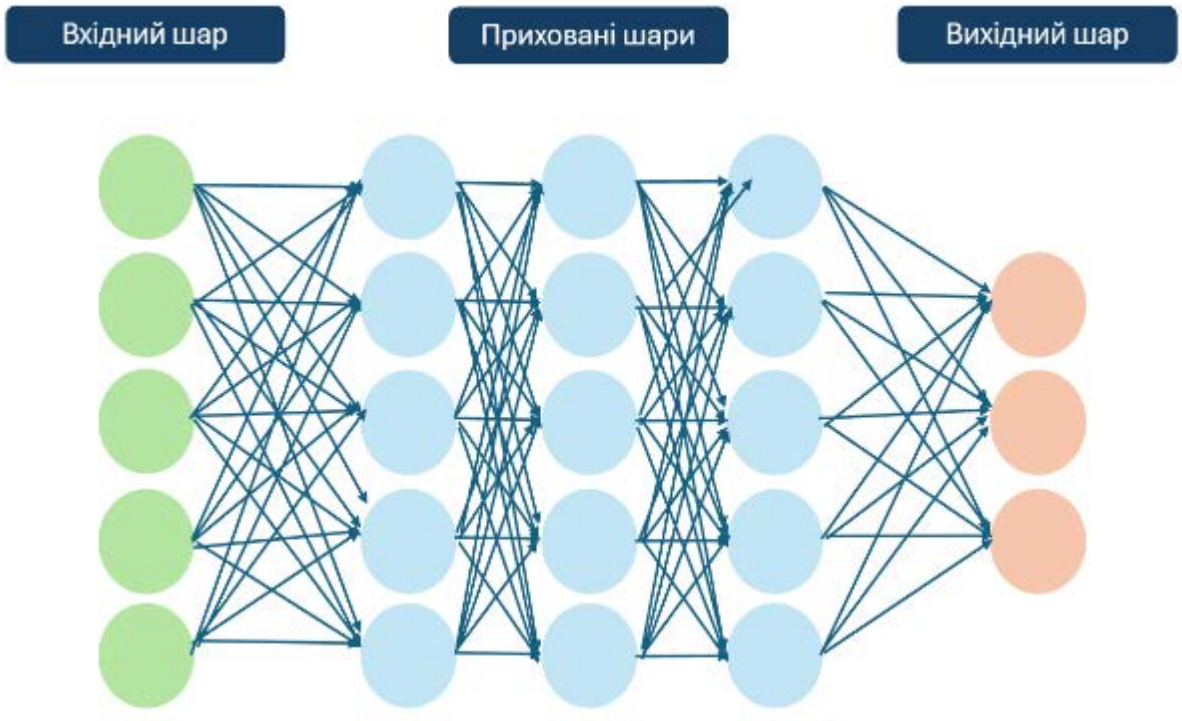


Рисунок 1.9 – Стандартна структура глибокої нейронної мережі з 3 прихованими шарами [13]

Кожен із цих безлічі міжнейронних зв'язків множиться на унікальну вагу, яка посилює (або зменшує) вплив кожного зв'язку. Вхідні дані, що надаються функції активації кожного нейрона, можна розуміти як зважену суму виходів кожного нейрона попереднього шару. Зазвичай до кожної функції активації також додається унікальний член зміщення, який функціонує подібно до члена зміщення загальної регресійної функції. Під час

навчання нейронна мережа «навчається» шляхом коригування кожної з цих ваг та членів зміщення, що дає точніші виходи. Це параметри моделі: коли ви читаєте, наприклад, про модель великої мови (LLM), яка має 8 мільярдів «параметрів», це число відображає кожен зважений міжнейронний зв'язок та специфічне для нейрона зміщення в нейронній мережі моделі.

Часто існує мало, якщо взагалі є, інтуїтивно зрозуміле пояснення – окрім грубого математичного – того, як значення окремих параметрів моделі, отримані нейронною мережею, відображають реальні характеристики даних. З цієї причини моделі глибокого навчання часто називають «чорними скриньками», особливо порівняно з традиційними типами моделей машинного навчання, що базуються на ручній інженерії ознак [13].

Загалом серед переваг глибокого навчання можна виділити таки ключові моменти:

- автоматичне навчання ознак: традиційне машинне навчання вимагає ручної розробки ознак, коли експерти витрачають багато часу на визначення того, які аспекти необроблених даних є найбільш релевантними для певного завдання. Глибоке навчання усуває це вузьке місце, автоматично виявляючи закономірності в даних завдяки своїй багаторівневій архітектурі;
- краща продуктивність у складних завданнях: перевага в продуктивності стає особливо помітною для завдань, що включають нелінійні зв'язки та розпізнавання складних образів, де традиційні лінійні моделі мають труднощі зі складними взаємодіями даних;
- обробка великомасштабних та різноманітних даних: сучасні нейронні мережі чудово справляються з масивними, складними наборами даних, на відміну від традиційних алгоритмів. Ця масштабованість виявляється цінною для неструктурованих даних, де розмірність та складність роблять традиційні підходи непрактичними;
- стійкість: добре навчені моделі глибокого навчання демонструють стійкість до шуму, відсутності даних та варіацій у вхідних даних. Розподілена

природа представлень нейронних мереж означає, що окремі збої нейронів або відсутні функції не призводять до катастрофічного зниження продуктивності.

До недоліків глибокого навчання можна віднести:

- високі обчислювальні витрати: навчання моделей глибокого навчання вимагає значних обчислювальних ресурсів, що може бути непрактичним для багатьох організацій. Великі мережі потребують потужних графічних процесорів зі значною пам'яттю, а час навчання може тривати від годин до тижнів залежно від складності;
- проблема «чорної скриньки»: нейронним мережам бракує інтерпретаційних можливостей, що створює серйозні обмеження для програм, які потребують підзвітності. Зі зростанням складності моделі надзвичайно важко зрозуміти, чому мережа зробила певні прогнози. Це створює значні проблеми в таких галузях, як охорона здоров'я, фінанси та право, де рішення мають бути поясненими;
- залежність від якості даних: успіх глибокого навчання значною мірою залежить від якості та кількості навчальних даних. Низька якість призводить до моделей, які створюють проблеми у своїх прогнозах;
- проблеми перенавчання та узагальнення: здатність глибоких мереж запам'ятовувати навчальні дані може призвести до перенавчання, коли моделі чудово працюють на навчальних прикладах, але погано на нових даних. Ця проблема посилюється з обмеженими навчальними даними або надмірно складними архітектурами;
- складність впровадження: успішне застосування глибокого навчання вимагає розуміння як теорії нейронних мереж, так і специфічних для предметної області аспектів. Темпи розвитку означають, що найкращі практики швидко розвиваються, що вимагає постійного навчання від практиків. Створення надійних систем виходить за рамки проектування моделей і включає попередню обробку даних, навчальну інфраструктуру та аспекти розгортання, вимагаючи експертизи в багатьох технічних областях [14].

1.5 Штучний інтелект у кіноіндустрії

На сьогоднішній день ШІ активно використовується у багатьох галузях і кіноіндустрія тут не виключення. Він трансформує виробництво фільмів, автоматизуючи завдання, забезпечуючи аналіз даних та розширюючи творчі можливості. Одним з найбільш трансформаційних впливів ШІ у кінопродукції є його здатність аналізувати величезні обсяги даних, щоб передбачити переваги аудиторії та успіх касового офісу. Досліджуючи тенденції та моделі поведінки глядачів, ШІ може допомогти кінематографістам адаптувати свій контент, щоб краще задовольнити потреби аудиторії, потенційно збільшуючи прибутковість фільму. Крім того, інструменти, що працюють за допомогою ШІ, можуть допомагати у написанні сценаріїв, пропонуючи розвиток сюжету, арки персонажів або діалог.

Наприклад, технологія активно використовується у препродакшені для:

– аналізу та оптимізації сценаріїв: це може допомогти режисерам покращити свій сценарій, щоб краще відповідати тенденціям ринку;

– аналітики ринкових тенденцій: розуміння тенденцій ринку та уподобань аудиторії має важливе значення для прийняття обґрунтованих рішень під час попереднього виробництва. Інструменти прогнозної аналітики, що керуються ШІ, аналізують величезну кількість даних з соціальних мереж, результатів продажу і демографію глядачів, щоб прогнозувати тенденції та уподобання;

– ШІ-генерації сторіборду: це процес, який займає багато часу і включає створення візуальних зображень кожної сцени у фільмі. Інструменти штучного інтелекту можуть автоматизувати цей процес, створюючи сторіборди зі сценаріїв, інтерпретуючи сценарій, ідентифікуючи ключові сцени та створюючи візуальні ескізи, які служать планом для кінематографістів. Це не тільки прискорює стадію препродакшену, але й гарантує, що візуальна розповідь відповідає намірам сценарію.

ШІ також використовують і у самому процесі створення фільму для:

– покращення роботи з камерою: вони можуть автоматично ставити кадри, відстежувати їх і регулювати фокус. Це підходить для створення точних і динамічних кадрів, які раніше було непросто досягти - системи стабілізації, керовані штучним інтелектом, забезпечують плавну і стабільну зйомку, зменшуючи необхідність в постпродукційних корекціях;

– автоматизації відстеження кадру: алгоритми ШІ можуть аналізувати сцену та автоматично коригувати композицію, щоб забезпечити оптимальне оформлення та фокусування. Ця технологія особливо корисна знову ж таки для динамічних кадрів, де ручні регулювання складні;

– автоматизації освітлення та спецефектів: системи освітлення, керовані ШІ, можуть регулювати освітлення в режимі реального часу на основі вимог сцени, забезпечуючи постійне освітлення та потрібний настрій.

У постпродакшені ШІ використовується найбільш активно:

– візуальні ефекти: ШІ розвивається у цьому напрямку семимильними кроками, спеціальні програми можуть генерувати зображення, відео, створювати 3д моделі, інтегрувати в кадр CGI елементи і т.і.;

– звуковий дизайн: ШІ може аналізувати аудіо, виявляти патерни та генерувати нові звукові ефекти, які підходять до конкретної сцени;

– редагування матеріалу: ШІ може аналізувати відзнятий матеріал та пропонувати можливе покращення кадру. Це може допомогти автоматизувати такі рутинні процеси, як корекція кольору, аудіо синхронізація і т.і. [15].

Згідно з дослідженням від market.us, на 2023 рік машинне навчання займало перше місце по відсотку застосувань на ринку штучного інтелекту у кіноіндустрії, займаючи 35%.

Це пояснюється тим, що машинне навчання дуже універсальне і його можна застосовувати на різних етапах створення фільму, приклади яких було приведено вище. Обробляючи великі датасети, щоб ідентифікувати патерни, і роблячи прогнози машинне навчання дозволяє зробити висновки, які

грунтуються на чітких даних і фактах, та таким чином допомогти кінорежисерам посилити привабливість їх проєкту для аудиторії, збільшити його потенційний успіх та рентабельність інвестицій на рекламні витрати. Хоча комп'ютерне бачення і глибоке навчання також швидко розвиваються і їх відсоток буде збільшуватись [16].

Рентабельність інвестицій(ROI або return on investment) – це показник, який визначає ефективність інвестицій, вимірюючи співвідношення прибутку до витрат. Він допомагає бізнесам оцінити рентабельність своїх вкладень і приймати обґрунтовані фінансові рішення. У формулі (1.1) представлено формулу рентабельності інвестицій:

$$ROI = \frac{(Дохід - Інвестиції) * 100\%}{Інвестиції}, \quad (1.1)$$

де *Дохід* – загальна сума коштів, отримана від інвестиції;

Інвестиції – сума коштів, вкладених у проєкт чи компанію.

Таким чином, ROI може бути позитивним, нульовим або негативним.

Позитивний ROI (понад 0) – інвестиції принесли прибуток. Чим вище значення, тим краща ефективність вкладення.

Нульовий ROI – інвестиції повернули лише вкладені кошти без додаткового прибутку. Ви не заробили, але й не втратили.

Негативний ROI (менш як 0) – інвестиції були збитковими. Ви втратили частину або всі вкладені кошти [17].

1.6 API

API – набір правил та протоколів, які дозволяють різним програмам взаємодіяти між собою. API використовується для побудови розподілених програмних систем, компоненти яких слабо пов'язані. Зазвичай додатками, які використовують API, є мобільні та хмарні додатки, вебзастосунки або розумні девайси. Ці набори пропонують інтерфейс багаторазового використання, до

якого легко можуть підключатись різні додатки. Але API не пропонує інтерфейсу користувача – вони невидимі та зазвичай ніякий кінцевий користувач не буде взаємодіяти з цим напряму. Замість цього, вся робота з API проходить «під капотом» застосунків і виконується тоді, коли потрібне з’єднання з іншими програмами. Єдині люди, котрі зазвичай мають справу з API, це розробники, які створюють застосунки або рішення з API [18].

Зрозуміло розглядати зв'язок API, як запит і відповідь між клієнтом і сервером. Програма, яка надсилає запит, є клієнтом, а сервер надає відповідь. API – це міст, що встановлює зв'язок між ними. На рисунку 1.10 представлено схему того, як працює API.

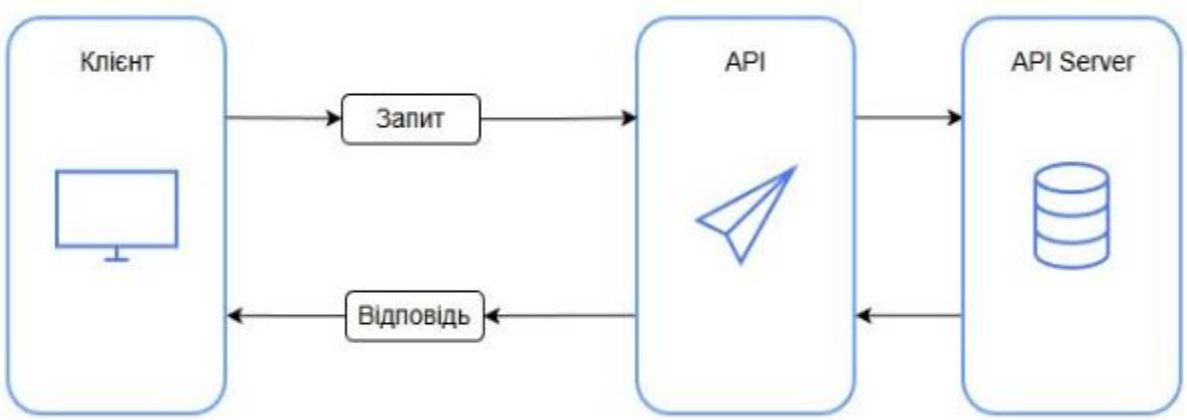


Рисунок 1.10 – Схема роботи API [20]

Простий спосіб зрозуміти, як працюють API, – це розглянути поширений приклад – обробку платежів третьою стороною. Коли користувач купує товар на сайті електронної комерції, сайт може запропонувати користувачеві «Сплатити за допомогою PayPal» або іншого типу сторонньої системи. Ця функція покладається на API для встановлення з'єднання:

- коли покупець натискає кнопку оплати, надсилається виклик API для отримання інформації. Це запит. Цей запит обробляється з програми на вебсервер через універсальний ідентифікатор ресурсу (URI) API та включає дієслово запиту (наприклад, get), заголовки, а іноді й тіло запиту;

- після отримання дійсного запиту з вебсторінки продукту API звертається до зовнішньої програми або вебсервера, у цьому випадку до сторонньої платіжної системи;
- сервер надсилає відповідь API із запитуваною інформацією;
- API передає дані до початкової програми-запитувача, у цьому випадку, вебсайту продукту [19].

Тепер обговоримо більш детально терміни клієнт, запит, сервер, відповідь.

Клієнт API відповідає за початок розмови, надсилаючи запит на сервер API. Запит може бути ініційований багатьма способами. Наприклад, користувач може ініціювати запит API, ввівши пошуковий термін або натиснувши кнопку. Запити API також можуть бути ініційовані зовнішніми подіями, такими як сповіщення від іншої програми.

Запит API виглядатиме та працюватиме по-різному залежно від типу API, але зазвичай він містить такі компоненти:

- кінцева точка (endpoint) – це виділена URL-адреса, яка надає доступ до певного ресурсу. Наприклад, кінцева точка /articles у додатку для блогів міститиме логіку для обробки всіх запитів, пов’язаних зі статтями;
- метод (method) – вказує на тип операції, яку клієнт хоче виконати над певним ресурсом. REST API доступні через стандартні методи HTTP, які виконують поширені дії, такі як отримання(get), створення(post), оновлення(put) та видалення даних(delete);
- параметри (parameters) – це змінні, які передаються кінцевій точці API, щоб надати API конкретні інструкції для обробки. Ці параметри можуть бути включені до запиту API як частина URL-адреси, у рядок запиту або в тіло запиту. Наприклад, кінцева точка /articles API для блогів може приймати параметр «тема», який вона використовуватиме для доступу та повернення статей на певну тему;

- заголовки запиту(request headers) – це пари ключ-значення, які надають додаткові відомості про запит, такі як тип вмісту або дані автентифікації;
- тіло запиту(request body) – це основна частина запиту, яка містить фактичні дані, необхідні для створення, оновлення або видалення ресурсу. Наприклад, якщо ви створюєте нову статтю в додатку для ведення блогу, тіло запиту, ймовірно, міститиме вміст статті, назву та автора [20].

У висновку, API спрощують проектування та розробку нових програм і сервісів, а також інтеграцію та управління існуючими за рахунок своєї безпеки, автоматизації та економічної ефективності.

1.7 AWS

AWS (Amazon Web Services) – це комплексна платформа хмарних обчислень, що надається Amazon. Вона включає поєднання пропозицій «інфраструктура як послуга» (IaaS), «платформа як послуга» (PaaS) та «програмне забезпечення як послуга» (SaaS). AWS пропонує такі інструменти, як обчислювальна потужність, сховище баз даних та послуги доставки контенту.

Компанія Amazon запустила свої перші вебсервіси у 2002 році на основі внутрішньої інфраструктури, яку компанія створила для управління своєю онлайн-торгівлею. У 2006 році вона почала пропонувати свої визначальні послуги IaaS. AWS була однією з перших компаній, яка запровадила модель хмарних обчислень з оплатою за використання, яка масштабується, щоб забезпечити користувачів обчислювальними ресурсами, сховищем та пропускною здатністю за потреби.

Маючи понад 200 сервісів, AWS пропонує низку пропозицій для приватних осіб, а також державних та приватних організацій для створення додатків та інформаційних сервісів усіх видів. Ці сервіси базуються на хмарних технологіях та, як правило, є економічно ефективними. Вони

взаємодіють з багатьма мовами програмування, спілкуються через різні мережі та взаємодіють з конкуруючими постачальниками хмарних послуг (CSP). AWS була першим розробником хмарних сервісів, і, як наслідок, має великий перелік пропозицій та клієнтську базу. Її використовують організації по всьому світу через її глобальну мережу центрів обробки даних.

AWS розділений на різні сервіси; кожен з них можна налаштувати по-різному залежно від потреб користувача. Користувачі можуть переглядати параметри конфігурації та окремі карти серверів для сервісу AWS. Портфоліо AWS включає такі категорії послуг, як, наприклад, обчислення, зберігання даних, бази даних, управління інфраструктурою, розробка додатків та інші.

AWS надає послуги з десятків центрів обробки даних, розподілених по 105 зонах доступності (AZ, availability zone) у регіонах по всьому світу. AZ – це місце, яке містить кілька фізичних центрів обробки даних. Регіон – це сукупність AZ, розташованих у географічній близькості, з’єднаних мережевими каналами з низькою затримкою. Компанія вибирає одну або кілька таких зон з різних причин, зокрема для відповідності вимогам, близькості до клієнтів та оптимізації доступності. Наприклад, клієнт AWS може запускати віртуальні машини та реплікувати дані в різних AZ, щоб досягти високонадійної, економічно ефективної хмарної інфраструктури з масштабованістю, стійкою до збоїв окремих серверів та всього центру обробки даних.

У роботі буде використано Amazon Simple Storage Service (Amazon S3), для зберігання набору даних у хмарному сховищі, з якого його можна буде в будь-який момент дістати.

Amazon S3 забезпечує масштабоване об'єктне сховище для резервного копіювання, збору та аналітики даних. IT-фахівець зберігає дані та файли як об'єкти S3, розмір яких може досягати п'яти терабайт, всередині контейнерів S3 для їхньої організації. Бізнес може заощадити гроші за допомогою S3 завдяки класу сховищ Infette Access або використовуючи Amazon Glacier для довгострокового холодного зберігання [21].

1.8 Patreon

Patreon – це платформа на основі підписки, яка дозволяє професійним творцям контенту, великому бізнесу та всім іншим отримувати пряму оплату за свій контент, продукти та послуги.

Передумова Patreon напрочуд проста: творці пропонують контент, послуги чи продукти, а користувачі за них платять. Ціни можуть бути встановлені як одноразові платежі або як частина моделі підписки – найчастіше можна побачити контент, що пропонується за багаторівневою структурою ціноутворення. Чим більше платить користувач, тим більше доступу та контенту він отримає. Платформа може використовуватися для одноразового проекту чи продукту, але з часом, коли він стає більш усталеним, творці можуть комфортніше використовувати її на довгостроковій основі [22].

У цій роботі Patreon буде використовано для отримання платного набору даних, який також буде доповнено на основі даних з таких сервісів, як, наприклад, TMDB (The Movie Database), OMDB (Open Movie Database).

1.9 Аналіз існуючих аналогів

1.9.1 IMDB

IMDB – вебсайт, який надає інформацію про мільйони фільмів і телевізійних програм, а також про їхніх акторів та знімальну групу. Назва є абревіатурою від Internet Movie Database. Його інтерфейс можна побачити на рисунку 1.11.

Серед переваг треба виділити зручний інтерфейс, наявність різного роду топів(наприклад, за оцінками), наявність повної інформації про фільм(або серіал) та простота використання.

Серед недоліків можна виділити неможливість безкоштовно використовувати IMDB API для доступу до цих даних(є лише застарілі, хоч і якісні, датасети з Kaggle), відсутність візуалізації та інструментів аналізу.

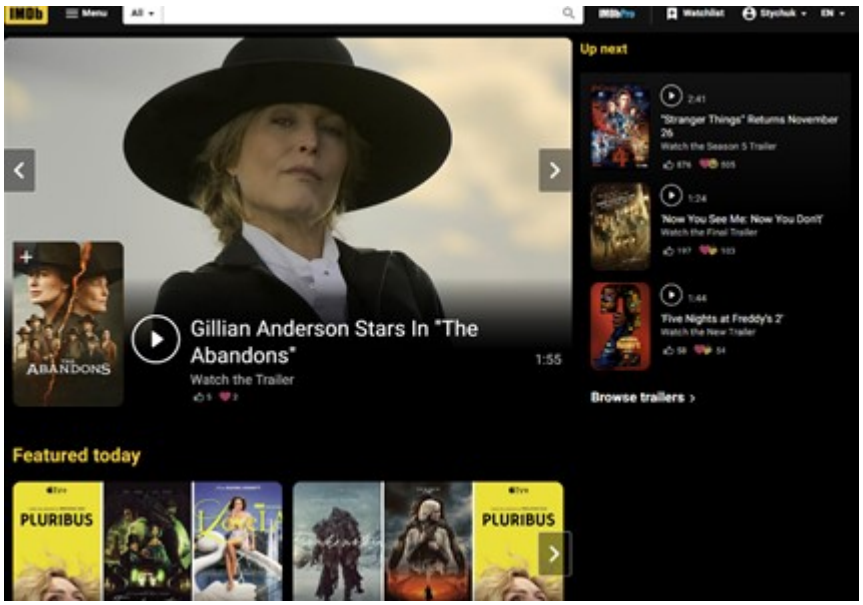


Рисунок 1.11 – IMDB [23]

1.9.2 Comscore

Comscore – американська глобальна компанія з вимірювання та аналітики медіа, що надає маркетингові дані та аналітику підприємствам, рекламним агентствам, бренд-маркетологам та видавцям [24]. На рисунку 1.12 представлено її інтерфейс.

Серед переваг треба відзначити, що це дорогий і дуже потужний проєкт з якісною аналітикою, який точно виходить за рамку дипломного проєкту, проте не відзначити його неможливо. Це один з лідерів на ринку і провідні компанії активно використовують його, наприклад, для передбачення популярності або зборів їх майбутнього фільму.

Але є і недоліки – беручи до увагу його глобальність та складність, він аж ніяк не може бути безкоштовним і звичайний користувач навряд чи буде користуватись цим сервісом. Також, Comscore не містить елементів прогнозування, так як ця компанія фокусується на зборі даних, які в свою чергу вже використовують інші компанії для прогнозування.

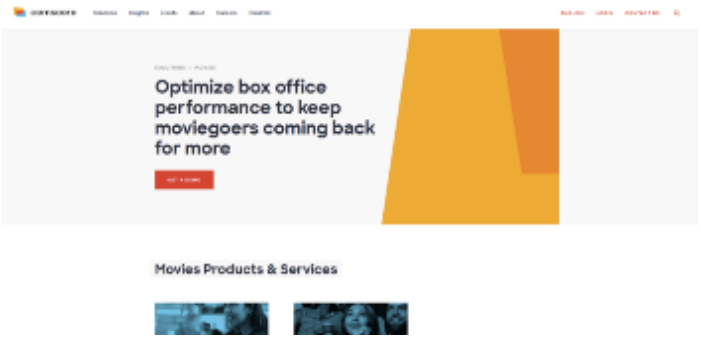


Рисунок 1.12 – Comscore [25]

1.9.3 Largo.ai

Largo.ai – це платформа, яка надає дуже детальний аналіз, який виходить далеко за межі тільки прибутків. Наприклад, платформа може робити пропозиції щодо кастингу акторів, аналізувати сценарій, його ключові слова, настрої, ну і звичайно робити прогнози щодо прибутковості фільму. Її інтерфейс представлено на рисунку 1.13.

Рисунок 1.13 – Largo.ai [26]

До переваг треба віднести потужну аналітику і прогнозування, яке не обмежується касовими зборами, та багато якісних даних про фільми, на яких вони базують свої прогнози.

До недоліків знову ж таки відноситься ціна – це дороге рішення для великих компаній.

1.9.4 Порівняння аналогів

Проведемо порівняння аналогів за певними критеріями, хід якого представлено у таблиці 1.1. У ній будуть порівнюватись вищеописані аналоги та застосунок, що розробляється «Програмне забезпечення для аналізу і прогнозування прибутків і рейтингу фільмів». Виділимо наступні критерії:

- безкоштовність;
- наявність детального аналізу;
- наявність прогнозування;
- наявність візуалізації даних;
- наявність якісних даних з багатьох джерел для прогнозів;
- можливість легко завантажити дані та/або поділитись ними.

Таблиця 1.1 – Порівняння аналогів за критеріями

Критерій порівняння	Порівняння аналогів і застосунку			
	IMDB	Comscore	Largo.ai	«Програмне забезпечення для аналізу і прогнозування прибутків і рейтингу фільмів»
Безкоштовність	+–	–	–	+
Детальний аналіз	+–	+	+	+
Прогнозування	–	–	–	+
Візуалізація даних	–	+	+	+
Якісні дані з багатьох джерел	+	+	+	+
Можливість легко завантажити дані	+–	–	–	–

З результатів порівняння видно, що аналоги різняться між собою. Треба відзначити, що загалом аналоги пропонують якісні рішення для бізнесів з надійними джерелами даних та навіть нестандартні підходи до аналізу і прогнозування(проте не кожен з аналогів пропонує аналіз і прогнозування).

Однак не кожен з них пропонує візуалізацію, можливість легко завантажити дані, а також безкоштовність.

Саме тому актуальною залишається задача розробки нового ПЗ, яке буде відповідати даним вимогам.

1.10 Технічне завдання

1.10.1 Призначення та область застосування програми

Програма, яку буде розроблено у результаті даної дипломної роботи, має на меті аналіз, візуалізацію даних, можливість легко завантажити дані, прогнозування та дані з декількох джерел.

Основна область застосування програми – це кіноіндустрія. Програма допоможе користувачам більш детально заглибитись у тренди та аналіз стрічок кіно – візуалізує основні патерни, дослідить залежність між собою певних характеристик, проведе аналіз рейтингів та прибутків, а також спробує передбачити їх на основі даних з відкритих(або майже відкритих) джерел. Таким чином, програма дасть користувачам можливість більш глибоко зрозуміти кіноіндустрію і вподобання глядачів.

1.10.2 Підстави для розробки

У сучасному світі використання машинного навчання стало скоріше правилом у кіноіндустрії і бізнесі, ніж винятком. Його використовують для глибоко аналізу всього навколо – від вподобань глядачів, можливості фільму стати касовим тріумфом до безпосередньо самого процесу створення фільмів. Існуючі аналоги пропонують цікаві рішення, але кожне з яких має свої обмеження, такі як, наприклад, відсутність візуалізації, більш детального аналізу, можливості легко завантажити дані або прогнозування.

Все вищесказане вимагає розробки доступного ПЗ для аналізу рейтингів і прибутків фільмів, з подальшою можливістю прогнозування цього.

Користувач отримає безкоштовний аналог, який ставить собі за мету робити все це.

1.10.3 Призначення розробки

Призначенням розробки є створення ПЗ, яке надає можливість аналізу прибутків і рейтингу фільмів з подальшою можливістю прогнозування цих даних.

1.10.4 Вимоги до функціональних характеристик

ПЗ має реалізовувати наступні характеристики:

- завантаження даних за допомогою API з декількох джерел;
- попередня обробка даних, групування даних з декількох наборів даних;
- аналіз трендів і патернів для прибутку та рейтингів;
- побудова моделей машинного навчання та нейронних мереж;
- візуалізація даних, представлення їх у зручній для користувача формі(як у вигляді окремих графіків, так і у вигляді вебсторінки у браузері);
- завантаження обробленого набору даних на хмарний сервіс, звідки його можна вивантажити в будь-який момент.

1.10.5 Вимоги до надійності

Вимоги до надійності ПЗ включають в себе:

– перевірку алгоритмів, що використовуються у програмі, на точність та стабільність результатів;

– виведення повідомлень для користувача у разі виникнення помилок;

– виведення повідомлень для користувача, які показують етапи роботи програми;

– виведення повідомлень для користувача, які показують результати роботи програми.

1.10.6 Умови експлуатації

Для експлуатації ПЗ необхідно мати стабільне підключення до мережі Інтернет для здійснення API запитів. Також додатково необхідно встановити необхідну версію Python з потрібними бібліотеками (які будуть прописані у файлі requirements.txt) та системою керуваннями пакетів pip, бути знайомим з AWS, щоб завантажити набір даних з їх хмарного сховища.

1.10.7 Вимоги до складу та параметрів технічних засобів

Комп’ютер користувача для коректної роботи з ПЗ має відповідати таким вимогам:

– стабільне підключення до мережі Інтернет для здійснення API запитів;

– операційна система: Windows 10 або вище;

– версія Python (не нижче за 3.12.1) та pip (не нижче за 24.3.1);

– вільний дисковий простір для встановлення необхідних бібліотек і роботи програми без збоїв: мінімум 2-3 ГБ загального вільного простору;

– оперативна пам’ять: мінімум 8 ГБ;

– процесор: не менше 4 ядер з тактовою частотою не менше 3,4 ГГц.

1.10.8 Необхідні стадії і терміни розробки

Розробка ПЗ для аналізу прибутку і рейтингу фільмів з подальшою можливістю прогнозування включає наступні стадії та терміни:

– аналіз вимог: визначення вимог до програми. Термін: 1 тиждень;

– проєктування системи: розробка архітектури та вибір технологічних засобів. Термін: 1 тиждень;

– реалізація та програмування: написання коду програми та реалізація визначених функцій. Термін: 2 тижні;

– тестування та налагодження: проведення випробувань для виявлення та усунення помилок. Термін: 1 тиждень.

1.10.9 Види випробувань

Для забезпечення якості та надійності буде проведено наступні види випробувань:

– функціональні тести: перевірка відповідності функціональних вимог програми;

– інтеграційні тести: перевірка взаємодії між різними частинами програми, а також з коректністю завантаження у хмарне сховище;

– системні тести: тестування програми в цілому.

1.10.10 Висновки за розділом

Загалом, у першому розділі було виконано аналіз предметної області ПЗ, що розробляється, – кіноіндустрії. Було також розібрано основні поняття штучного інтелекту, машинного навчання, нейронних мереж, API, AWS та як це активно застосовується у вищезгаданій галузі. Також було проведено порівняння аналогів і визначено основні моменти, на яких буде фокусуватись майбутня програма, поставлено технічне завдання.

2 МАТЕРІАЛИ І МЕТОДИ

Другий розділ буде присвячено аналізу матеріалів і методів, які було застосовано у ПЗ.

2.1 Вибір мови програмування

Для розробки ПЗ було використано мову програмування Python.

Python – це інтерпретована, об’єктно-орієнтована та високорівнева мова програмування загального призначення з суворою динамічною типізацією. Python також підтримує кілька парадигм програмування: структурне, об'єктно-орієнтоване, функціональне, імперативне і аспектно-орієнтоване.

Інтерпретованість означає, що початковий код при запуску не перетворюється одразу в машинний, як у компільованих мовах, а виконується рядок за рядком за допомогою спеціального інтерпретатора. Високорівневість – швидкість і зручність використання для людини за допомогою введення додаткових смислових конструкцій, які описують структури даних, операції над ними. Для виконання високорівневі мови перетворюються в машинний код. Низькорівневі мови наближені до машинного коду і є більш складними для їх використання у розробці, проте не містять зайві конструкції, що збільшує їх швидкодію. Динамічна типізація означає, що більша частина перевірок типів даних виконується під час виконання програми, а не під час компіляції. Це дозволяє змінити тип даних змінної у процесі виконання програми, проте може призвести до неочікуваних помилок з типами даних при роботі [27].

Далі буде порівняно мову програмування Python з іншими двома мовами, яка можна було використати для поставленої задачі: R та Java. Їх буде порівняно за наступними критеріями: швидкість, наявність бібліотек

машинного навчання, наявність необхідних IDE, масштабованість, розв’язання математичних задач, візуалізація даних:

– швидкість: Python та R є інтерпретованими мовами, тому вони можуть бути повільнішими за Java. Але для завдань машинного навчання, де у більшості своїй використовуються бібліотеки машинного навчання і їх вбудовані функції, ця різниця може стати майже непомітною. Також, незважаючи на швидкість, Java може потребувати більш об’ємного коду, ніж Python;

– бібліотеки машинного навчання: Python має широкий набір бібліотек різного призначення, серед яких є потужні бібліотеки для машинного навчання (TensorFlow, Scikit-learn і т.і.). R, що базується на статистиці, також пропонує велику колекцію бібліотек, спеціально розроблених для статистичного аналізу(caret, nnet). Java трохи відстає, але має хороші фреймворки машинного навчання, такі як DeepLearning4j, Weka та Tribuo, корисні, коли машинне навчання поєднується з технологіями великих даних;

– середовища розробки: Python має широкий вибір середовищ розробки – PyCharm, Visual Studio, Jupyter Notebook. Найпотужнішим і найпопулярнішим для R є RStudio. Для Java – Eclipse, NetBeans та Intellij IDEA;

– математичні задачі: тут лідерами є Python та R – мають інструменти і для класичної математики(проте, у Python таких все ж більше), так і конкретно для машинного навчання. Java тут відстає від них – довгі формули стають нечитабельними та заплутаними;

– візуалізація даних: Python пропонує потужні інструменти візуалізації, такі як Matplotlib, seaborn та Plotly, хоча вони вимагають більше кодування та налаштування. Створений з урахуванням візуалізації даних, R забезпечує безперешкодну інтеграцію з бібліотеками, такими як ggplot2, що дозволяє користувачам створювати візуально насичені, інтерактивні та складні графіки з меншими зусиллями. По цьому пункту Java також трохи

відстає: мова має бібліотеки, але їх різноманітність не дуже вражає(tablesaw, jfreechart);

– масштабованість: тут лідером є Java – це гарний вибір для великий і довгих проєктів. R же, навпаки, краще використовувати для невеликих наукових проєктів, які не будуть багато розростатись [28]-[30].

Нижче представлено таблицю 2.1, яка представляє порівняння мов програмування.

Таблиця 2.1 – Порівняння мов програмування

Критерій	Python	R	Java
Швидкість	++	++	++
Бібліотеки машинного навчання	++	++	+
Середовища розробки	++	+	++
Математичні задачі	++	++	+
Візуалізація даних	++	++	+
Масштабованість	+	+	++

Таким чином, проаналізувавши результати, Python було обрано мовою програмування, яка найбільше підходить саме під цей проєкт.

2.2 Вибір середовища розробки

У цьому розділі буде представлено порівняння трьох середовищ розробки – PyCharm, Visual Studio Code та Jupyter Lab – за такими категоріями:

– ціна: всі три середовища є безкоштовними(у PyCharm є Pro версія, але вона є необов’язковою і середовище можна використовувати і без неї);

– спеціалізація під Python: тут лідером є PyCharm – середовище розробки спеціалізовано конкретно під Python, хоча дозволяє використовувати і інші мови програмування. VS Code є середовищем, яке дозволяє

використовувати велику кількість мов, але не має спеціалізації конкретно під Python. Jupyter Lab також підтримує багато мов програмування, проте першочергово розроблявся як раз для Python та R;

– вимоги до ресурсів: Тут найвибагливішим до ресурсів є PyCharm, оскільки це середовище спроєктовано для великих проєктів. Золотою серединою є VS Code, але його вибагливість також залежить від кількості встановлених плагінів. Найменш вибагливим у цьому контексті є Jupyter Lab;

– інтеграція з системами контролю версій: PyCharm має прекрасну інтеграцію «з коробки», до VS Code можна довантажити необхідні розширення і воно стане таким же потужним. А от Jupyter Lab має погану інтеграцію з системами контролю версій через і там це робиться менш зручно;

– підтримка розширень: VS Code є безумовним лідером у цій сфері, проте PyCharm теж має багато розширень, а більшість необхідних функцій, на відміну від VS Code, вже встановлені «з коробки». Jupyter Lab має найменшу кількість розширень порівняно з іншими;

– інтеграція з бібліотеками машинного навчання: всі ці середовища мають гарну інтеграцію з бібліотеками машинного навчання, проте PyCharm все ж виділяється [31].

Нижче представлено таблицю 2.2, яка представляє порівняння середовищ розробки для Python. Проаналізувавши результати, PyCharm було обрано середовищем розробки, яке найбільше підходить саме під цей проєкт.

Таблиця 2.2 – Порівняння середовищ розробки для Python

Критерій	PyCharm	VS Code	Jupyter Lab
Ціна	+	++	++
Спеціалізація під Python	++	+	+
Вимоги до ресурсів	+-	+	++
Інтеграція з системами контролю версій	++	++	+-

Продовження таблиці 2.2

Критерій	PyCharm	VS Code	Jupyter Lab
Підтримка розширень	++	++	+
Інтеграція з бібліотеками машинного навчання	++	+	+

2.3 Вибір бібліотек для машинного навчання

У цій роботі було використано дві бібліотеки для машинного навчання – scikit-learn та tensorflow.

Scikit-learn було використано для побудови моделей машинного навчання. Вона побудована на бібліотеках Matplotlib, NumPy та SciPy та надає алгоритми для завдань машинного навчання, включаючи класифікацію, регресію, кластеризацію, зменшення розмірності, попередньої обробки даних та метрик оцінювання.

Scikit-learn пропонує зрозумілу документацію і простоту у використанні, широкий вибір алгоритмів, зрозумілі метрики оцінювання та гарну інтеграцію з іншими бібліотеками Python, що у комплексі є чудовим вибором для цієї роботи. Мінусом може бути те, що вона не пропонує комплексних рішень для нейронних мереж, саме тому для них було використано бібліотеку tensorflow [32].

Tensorflow також є популярною бібліотекою для машинного і глибокого навчань. Вона пропонує широкий спектр застосувань – від обробки природньої мови(Natural Language Proccesion) та комп’ютерного зову(Computer Vision) до прогнозування часових рядів, а також дозволяє розгортати моделі на різних платформах [33].

2.4 Вибір бібліотек для візуалізації даних

У цій роботі було використано три бібліотеки для візуалізації даних – matplotlib та streamlit+altair.

Matplotlib – це потужна та універсальна бібліотека для побудови графіків з відкритим кодом для Python, розроблена для того, щоб допомогти користувачам візуалізувати дані в різних форматах. Вона має великий вибір графіків, більше можливостей кастомізації цих графіків, ніж, наприклад, Seaborn, гарну інтеграцію з NumPy та міжплатформну сумісність. Саме через це графіки у самій програмі побудовані за допомогою Matplotlib [34].

Інші дві бібліотеки – streamlit та altair – було використано у парі для того, щоб зробити візуалізацію даних на вебсторінці.

Streamlit – це бібліотека Python з відкритим кодом для створення інтерактивних вебдодатків, використовуючи лише Python. Вона ідеально підходить для створення інформаційних панелей, вебдодатків на основі даних, інструментів звітності та інтерактивних інтерфейсів користувача без використання HTML, CSS або JavaScript. Звичайно, її можливості є обмеженими, на відміну від використання комбінації HTML, CSS та JavaScript, проте вона проста у використанні і дозволяє швидко створити потрібну вам вебсторінку, де можна гарно представити якісь результати проєкту, графіки і т.і. [35]. Тому якщо вам не потрібно створювати якийсь комплексний вебсайт, а просто потрібна сторінка для візуалізації даних, то streamlit підійде ідеально.

Streamlit використовувалась для розмітки сторінки і графіків, а для створювання самих графіків було використано бібліотеку altair. Головною причиною чому було вибрано саме її для використання зі streamlit – графіки з matplotlib відображались би на вебсторінці, як статичні картинки, а altair дозволяє динамічно змінювати їх масштаб, додавати підказки при наведенні.

Altair – це декларативна бібліотека статистичної візуалізації на Python, розроблена для спрощення створення чіткої та інформативної графіки з мінімальним кодом. Побудована на базі Vega-Lite, Altair зосереджена на

простоті, читабельності та ефективності. На відміну від імперативних бібліотек, таких як matplotlib або seaborn, де потрібно вказати, як має бути побудований кожен елемент графіка, Altair дозволяє зосередитися на тому, що ви хочете візуалізувати для даних та їх взаємозв'язків [36].

2.5 Вибір бібліотек для API

У цій роботі було використано дві бібліотеки для API – kagglehub та requests.

Kagglehub було використано тому, що більшість датасетів з проєкту є саме наборами з Kaggle. І за допомогою kagglehub можна швидко завантажити необхідні датасети одразу у pandas DataFrame або набір даних Hugging Face. Бібліотеку requests було використано для одного з датасетів, тому що він був розміщений не на Kaggle. Для доступу до нього було оформлено підписку на Patreon та потім завантажено дані за допомогою отриманого звідти ключа з OMDBApi. Бібліотека requsts – це простий та потужний інструмент для надсилання HTTP-запитів та взаємодії з вебресурсами. Вона дозволяє легко надсилати запити GET, POST, PUT, DELETE, PATCH, HEAD до вебсерверів, обробляти відповіді та працювати з REST API та завданнями вебскрейпінгу [37].

2.6 Вибір бібліотек для AWS

У цій роботі було використано дві бібліотеки для AWS – boto3 та dotenv.

Dotenv використано для отримано назви бакету aws, id, ключа та назви регіону, а boto3 вже використано для завантаження обробленого датасету на AWS S3 за допомогою даних з dotenv.

Boto3 – це комплект розробки програмного забезпечення AWS для Python, який дозволяє розробникам писати програмне забезпечення, що

використовує такі сервіси, як Amazon S3 та Amazon EC2. Тому для завантаження на AWS S3 було використано саме цю бібліотеку.

Dotenv – бібліотека Python, яка дозволяє зчитувати пари ключ-значення з файлу .env та встановлювати їх як змінних середовища. Це саме те, що потрібно у нашому проєкті, у файлі .env збережено назву бакету aws, id, ключ та назву регіону, що дає інформацію у який бакет буде завантажено наш оброблений датасет.

2.7 Завантаження та об’єднання наборів даних

Усі набори даних, які було використано у ПЗ, було завантажено за допомогою API. Було вирішено не використовувати якийсь окремий набір даних, а об’єднати набори з декількох джерел, що може допомогти заповнити пропущенні значення, а також вирішено додати у основний набір(IMDB Dataset) додаткові колонки з інших датасетів. Таким чином на виході було отримано більш повний з точки зору інформації датасет.

На рисунку 2.1 представлено необроблений основний набір даних з IMDB. Він містить у собі топ 10 000 фільмів за період 1915–2023 років. Набір даних складається з 13 колонок: ID, назва фільму, рік виходу фільму, рейтинг користувачів IMDb, Рейтинг Metascore, валовий дохід фільму, кількість голосів на IMDb, тривалість фільму в хвилинах, жанр(и) фільму, сертифікація або рейтинг фільму, короткий сюжет фільму, режисер(и) фільму, актори фільму.

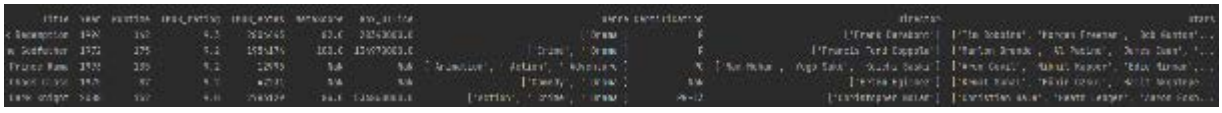


Рисунок 2.1 – Необроблений основний набір даних

Було видалено колонки ID, короткий сюжет фільму та актори фільму, створено колонку категоріального значення рейтингу фільму – вона

складається з трьох категорій(Bad, Average, Good). Також колонку жанрів було перетворено на бінарні колонки для кожного жанру. З інших наборів даних для заповнення пропущених значень було взято колонки сертифікації фільму, рейтингу MetaScore та доходу фільму. Також з інших наборів було додано додаткові колонки дня та місяця випуску, мови, країни випуску. Пропущені значення колонок заповнено даними з інших допоміжних наборів даних.

Другий необроблений допоміжний набір представлено на рисунку 2.2. Це топ 10 000 фільм TMDB. Звідти взято тільки колонки мови та дати релізу(яку потім було розбито на день і місяць, так як рік у основному наборі вже є).

Dataset1 example:

	Title	Language	rel_date
0	John Wick	en	2014-10-22
1	Ad Astra	en	2019-09-17
2	Bad Boys for Life	en	2020-01-15
3	The Lion King	en	2019-07-12
4	Jurassic World: Fallen Kingdom	en	2018-06-06

Рисунок 2.2 – Другий набір даних

Третій необроблений допоміжний набір представлено на рисунку 2.3. Це 45 000 фільмів з Movie Lens, які випущено до 2017 року. Звідти взято колонки доходу фільму та дати релізу(яку потім було розбито на день і місяць, так як рік у основному наборі вже є).

Dataset2 example:

	release_date	Box_Office	Title
0	1995-10-30	373554033.0	Toy Story
1	1995-12-15	262797249.0	Jumanji
2	1995-12-22	0.0	Grumpier Old Men
3	1995-12-22	81452156.0	Waiting to Exhale
4	1995-02-10	76578911.0	Father of the Bride Part II

Рисунок 2.3 – Третій набір даних

Четвертий необроблений допоміжний набір представлено на рисунку 2.4. Це дохідність фільмів з 2000 по 2024 роки. Звідти взято колонки доходу фільму та країни виробництва.

```
Dataset3 example:
                    Title   Box_Office                                  Country
0  Mission: Impossible II  546388108.0                 United States of America
1              Gladiator  460583960.0  United Kingdom, United States of America
2              Cast Away  429632142.0                 United States of America
```

Рисунок 2.4 – Четвертий набір даних

П’ятий необроблений допоміжний набір представлено на рисунку 2.5. Це набір даних з OMDB, який складається з фільмів основного набору. Звідси взято колонки сертифікату фільму, дати релізу(яку потім було розбито на день і місяць, так як рік у основному наборі вже є), мови, країни, рейтингу Metascore та доходу фільму.

```
Dataset4 example:
                          Title Certification     Released               Language        Country MetaScore  BoxOffice
0            The Shawshank Redemption            R  14 Oct 1994                English  United States        82   28767189
1                     The Godfather            R  24 Mar 1972  English, Italian, Latin  United States       100  136381073
2  Ramayana: The Legend of Prince Rama           PG  03 Nov 1997                English   India, Japan       N/A
```

Рисунок 2.5 – П’ятий набір даних

2.8 Завантаження обробленого набору даних на AWS S3

Об’єднаний та оброблений набір даних завантажується на AWS S3, як .csv файл. На рисунку 2.6 представлено цей завантажений на S3 файл. Перед завантаженням потрібно вставити у файл .env значення, які відповідають значенням вашого бакету aws, id, ключу та назві регіону.

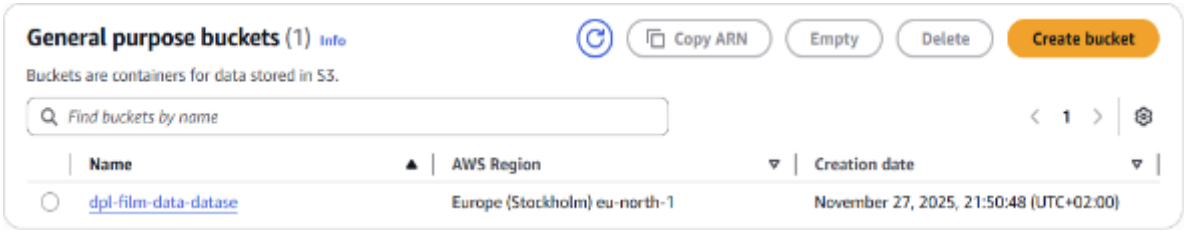


Рисунок 2.6 – Набір даних у AWS

2.9 Візуалізація даних за допомогою Streamlit та Altair

У додаток до візуалізації статичних графіків за допомогою Matplotlib, було використано streamlit та altair, за допомогою яких було створено вебсторінку, на якій можна динамічно змінювати масштаб графіків, відображати підказки при наведенні. На рисунках 2.7–2.9 представлено сторінку з цими графіками.

Рисунок 2.7 – Вебсторінка з графіками

Рисунок 2.8 – Вебсторінка з графіками

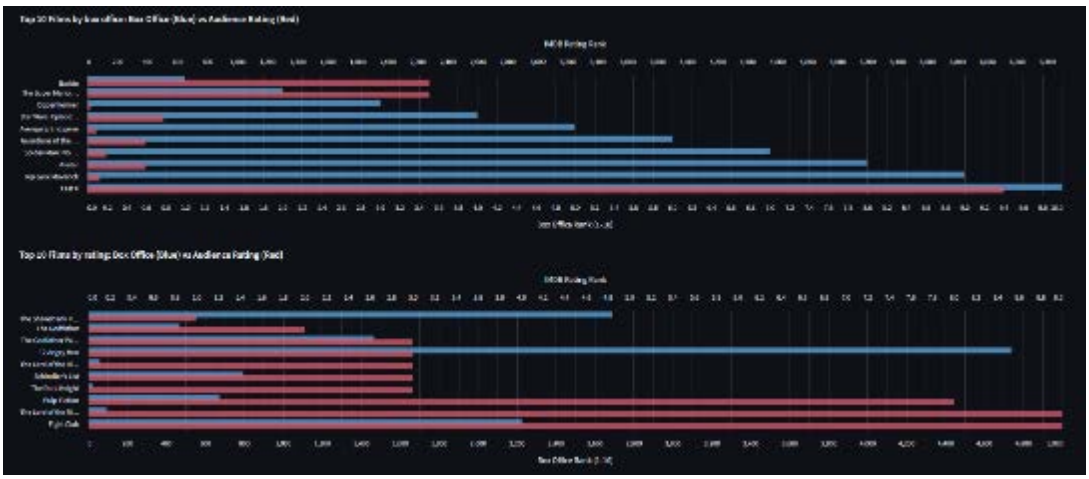


Рисунок 2.9 – Вебсторінка з графіками

Треба звернути увагу на те, що файл зі streamlit треба запускати не як звичайний Python файл. Треба у консолі перейти до папки scripts нашого проєкту і прописати команду «streamlit run to_streamlit.py», після чого відкриється потрібна вебсторінка. Запуск streamlit представлено на рисунку 2.10.

```
(venv) PS D:\serious_stuff\uni\6.1\diploma\file-revenue-rating-analyzer> cd scripts
(venv) PS D:\serious_stuff\uni\6.1\diploma\file-revenue-rating-analyzer\scripts> streamlit run to_streamlit.py
```

Рисунок 2.10 – Запуск streamlit

2.10 Висновки за розділом

У розділі було проведено детальний аналіз та обгрунтовано вибір програми засобів – мови програмування, середовища розробки, необхідних для виконання завдання бібліотек. Для реалізації програми було проаналізовано мови Python, Java та R і обрано Python, оскільки вона має підходящі бібліотеки для машинного навчання і візуалізації, має широкий вибір середовищ розробки та прекрасно підходить під поставлене завдання. Серед середовищ розробкки було порівняно PyCharm, VS Code та Jupyter Lab. Було обрано PyCharm, тому що воно має гарну інтеграцію з бібліотеками машинного навчання, є безкоштовним, проте дуже функціональним зі

спеціалізацією під Python, а також має інтеграцію з системами контролю версій. Для машинного навчання і нейронних мереж було обрано scikit-learn та tensorflow. За візуалізацію даних відповідають matplotlib, altair та streamlit.

3 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ

Третій розділ буде присвячено розробці архітектури програми.

3.1 Опис основних вимог до програми

ПЗ має надати можливість аналізу прибутків і рейтингу фільмів з подальшою можливістю прогнозування цих даних. До функціональних вимог програми віднесено:

- завантаження даних за допомогою API з декількох джерел;
- попередня обробка даних, групування даних з декількох наборів даних у один;
- аналіз трендів і патернів для прибутку та рейтингів;
- побудова моделей машинного навчання та нейронних мереж;
- візуалізація даних, представлення їх у зручній для користувача формі(як у вигляді окремих вікон з графіками, так і у вигляді вебсторінки у браузері);
- завантаження обробленого набору даних на хмарний сервіс, звідки його можна вивантажити в будь-який момент.

3.2 Логічна структура програми

Програму було побудовано за модульним принципом. Логічна структура програми складається з наступних компонентів:

– Fetch_data.py: перший для запуску модуль програми, який відповідає за отримання необхідних наборів даних та завантаження кожного з них окремо в data/raw;

– preprocessing.py: модуль, який відповідає за попередню обробку даних, об’єднання наборів, додавання нових колонок. Викликає у собі to_precossed.py для завантаження отриманого набору даних у data/processed;

– to_processed.py: модуль, який відповідає за завантаження отриманого обробленого набору даних у data/processed;

– upload_to_s3.py: модуль, який відповідає за завантаження отриманого набору даних на AWS S3 і який використовує дані з файлу .env;

– trends.py: модуль, який відповідає за створення графіків тенденцій індустрії кіно та виводу їх на екран;

– to_streamlit.py: модуль, який відповідає за створення вебсторінки, яка буде відображати інтерактивні графіки тенденцій індустрії кіно;

– models.py: модуль, який відповідає за будування моделей машинного навчання і нейронних мереж, а також вивід результатів на екран.

На рисунку 3.1 представлено структуру проєкту.

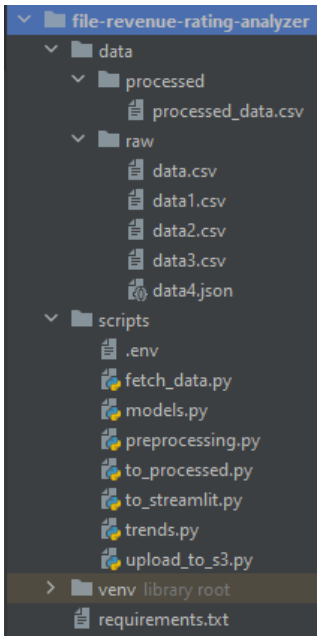


Рисунок 3.1 – Структура проєкту

У папці data є дві папки raw і processed, які потрібні для завантаження туди наборів даних. Папку raw створено для неочищених наборів даних, папку processed – для обробленого набору даних, який містить у собі дані зі всіх

наборів. Папку scripts було створено для модулів програми. Більш детально кожен з модулів буде розібрано у наступному розділі – розробка програми.

На рисунку 3.2 представлено модель взаємодії користувача з ПЗ у вигляді діаграми прецедентів. Спочатку відбувається завантаження даних за допомогою API на пристрій. Наступний крок – попередня обробка цих даних, об’єднання наборів даних і можливе завантаження їх на пристрій. Після чого йде створення моделей машинного навчання і нейронних мереж для прогнозування – для цього необхідно брати дані з пристрою. Така ж ситуація і з завантаженням на AWS S3 та відображенням трендів – для цього необхідно завантажити оброблений набір на пристрій.

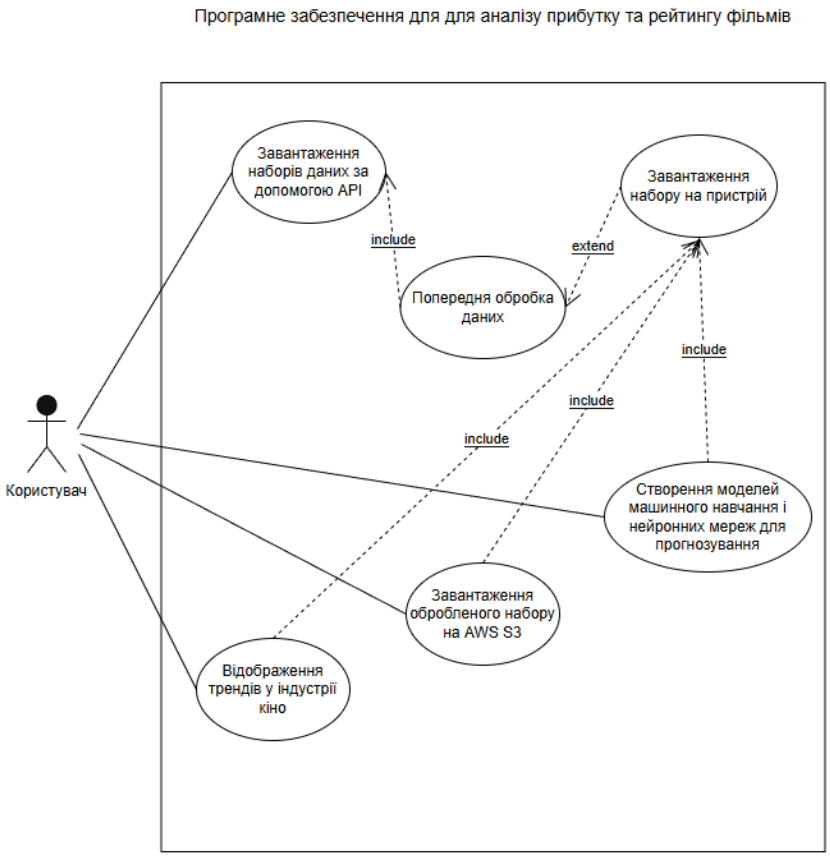


Рисунок 3.2 – Діаграма прецедентів

На рисунку 3.3 представлено діаграму діяльності для скрипту fetch_data.py, який відповідає за завантаження даних. Всередині є перевірки на наявність даних, які користувач бажає завантажити, а також на вдалість завантаження наборів.

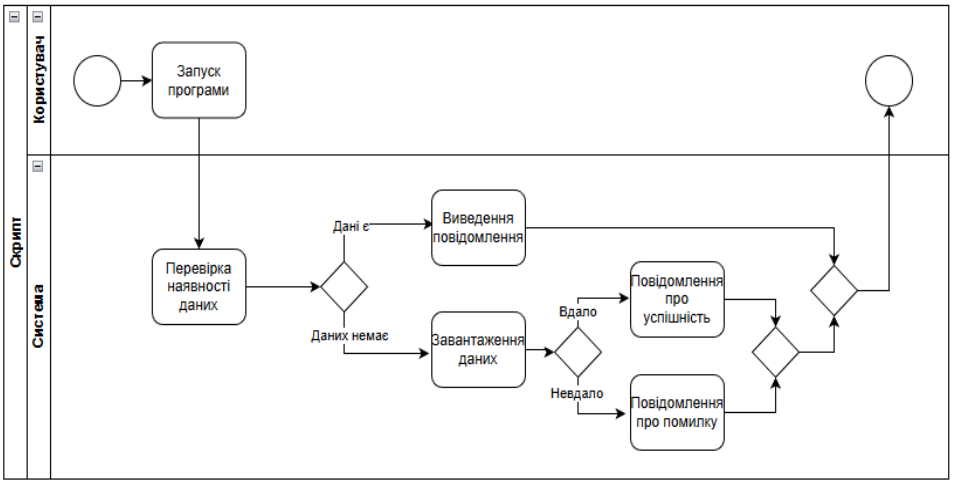


Рисунок 3.3 – Діаграма діяльності fetch_data.py

На рисунку 3.4 представлено діаграму діяльності для скрипту processing.py, який відповідає за попередню обробку даних, об’єднання наборів, і який викликає у собі скрипт to_processed.py, який призначений для збереження цього фінального набору у папку data/processed.

На рисунку 3.5 представлено діаграму діяльності для скрипту to_processed.py, який відповідає за збереження даних у папку data/processed. У ньому прописана перевірка на наявність даних на пристрої – якщо їх немає, то завантажити без додаткових дій, якщо є, то треба обрати потрібність перезапису.

На рисунку 3.6 представлено діаграму діяльності скрипту upload_to_s3.py, який відповідає за завантаження обробленого набору даних на AWS S3. У програмі прописана перевірка на наявність на пристрої необхідних оброблених даних, а також перевірка на правильність облікових даних AWS.

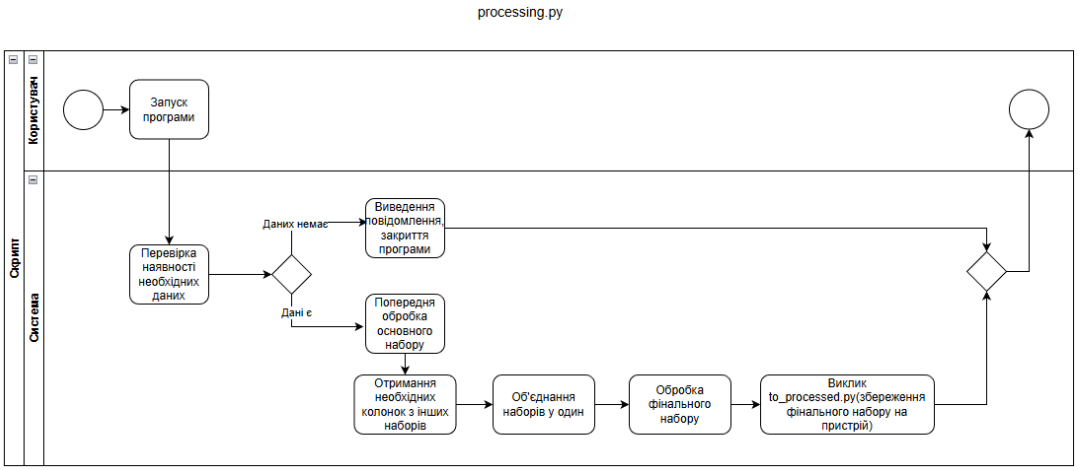


Рисунок 3.4 – Діаграма діяльності processing.py

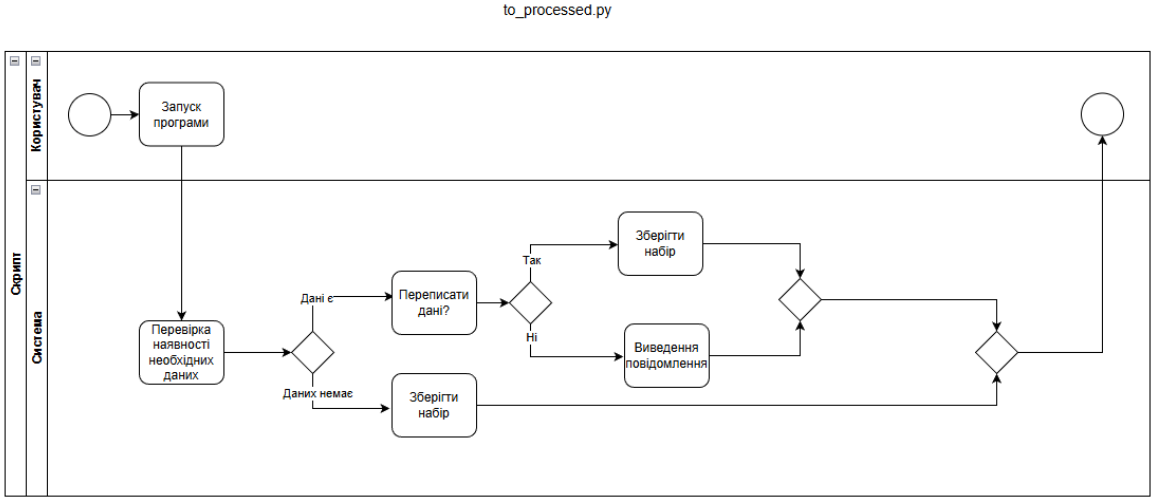


Рисунок 3.5 – Діаграма діяльності to_processed.py

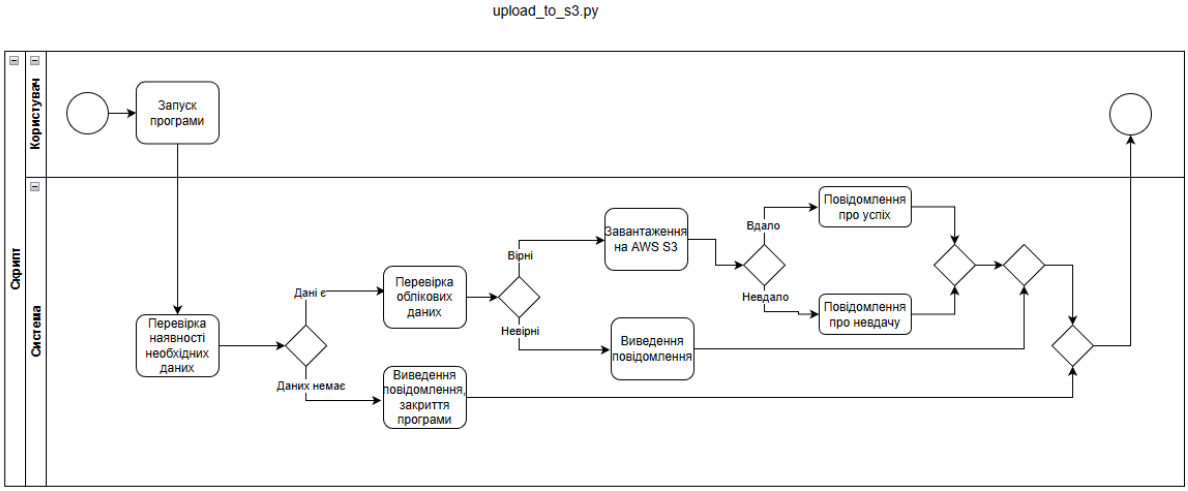


Рисунок 3.6 – Діаграма діяльності upload_to_s3.py

На рисунку 3.7 представлено діаграму діяльності скрипту trends.py, який відповідає за виведення графіків з тенденціями індустрії кіно. У програмі прописана перевірка на наявність даних на пристрої – якщо їх немає, то виводиться повідомлення про помилку і програма закривається, якщо вони є – на екран виводяться графіки.

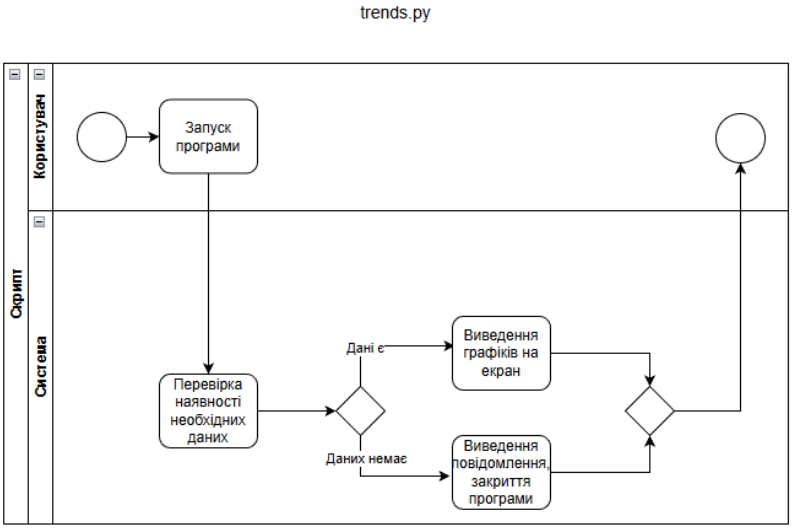


Рисунок 3.7 – Діаграма діяльності trends.py

На рисунку 3.8 представлено діаграму діяльності скрипту to_streamlit.py, який відповідає за виведення графіків з тенденціями індустрії кіно у вигляді вебсторінки. У програмі прописана перевірка на наявність даних на пристрої – якщо їх немає, то виводиться повідомлення про помилку і програма закривається, якщо вони є – відкривається вебсторінка з графіками.

На рисунку 3.9 представлено діаграму діяльності скрипту models.py, який відповідає за створення моделей машинного навчання і нейронних мереж для прогнозування даних. У програмі прописана перевірка на наявність даних на пристрої – якщо їх немає, то виводиться повідомлення про помилку і програма закривається, якщо вони є – відбувається виведення важливостей ознак, побудова моделей для прогнозування і виведення результатів на екран.

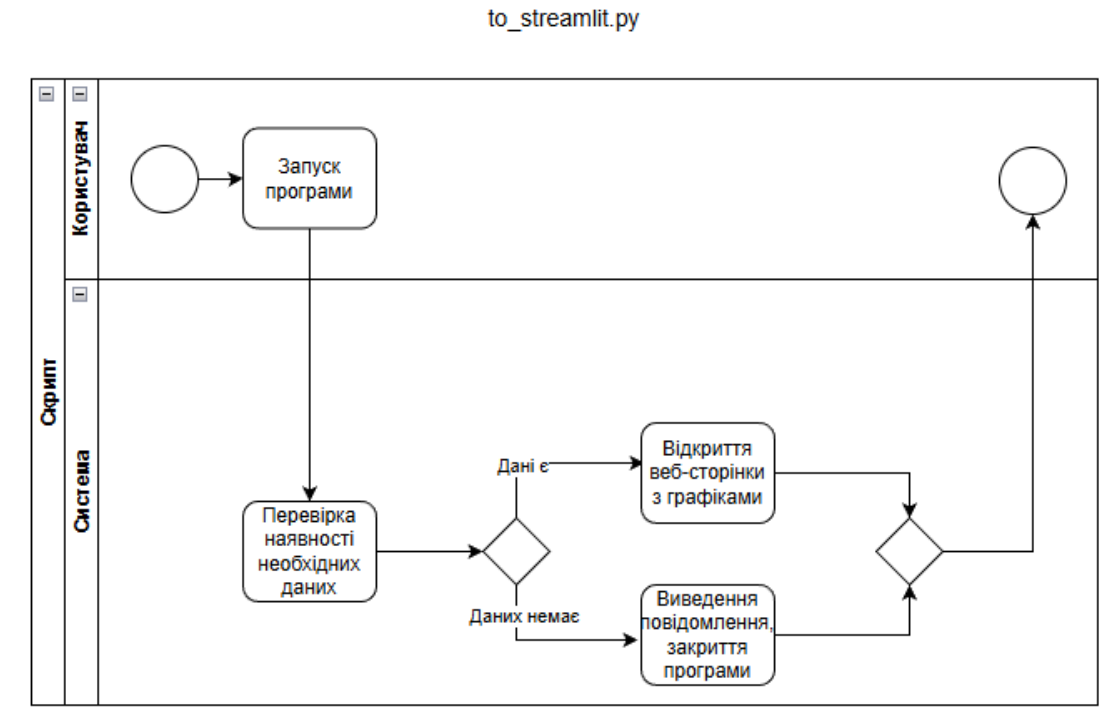


Рисунок 3.8 – Діаграма діяльності to_streamlit.py

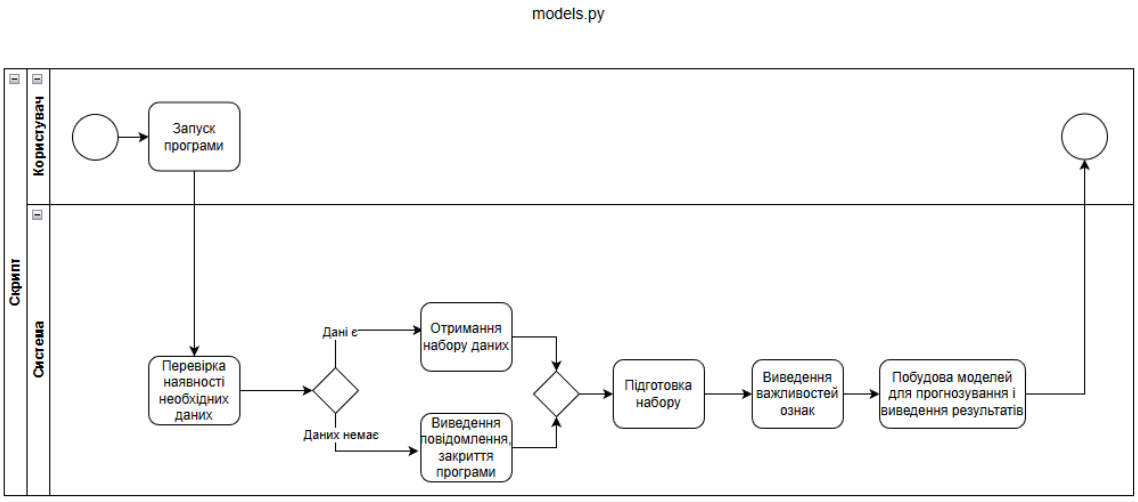


Рисунок 3.9 – Діаграма діяльності models.py

3.3 Висновки за розділом

У розділі було описано основні функціональні вимоги до програми – завантаження даних, попередня обробка, збереження на пристрій та AWS S3, створення графіків тенденцій та побудова моделей машинного навчання та нейронних мереж. Також представлено логічну структуру програми, діаграму

прецедентів та діаграми діяльності для кожного скрипту. Програма складається з 7 скриптів, кожен з яких відповідає за свою окрему функціональну вимогу.

4 РОЗРОБКА ПРОГРАМИ

Четвертий розділ буде присвячено розробці програми.

4.1 Опис основних модулів і функцій

Розберемо перший модуль для завантаження даних – fetch_data.py. На рисунку 4.1 представлено функцію main() цього модулю. Кожен окремий модуль проводить підрахунок часу, який буде витрачено на його виконання, і виводить його на екран. Для цього використано бібліотеку time. Також для всіх модулів використовується модуль Path бібліотеки pathlib, що дозволяє зручно працювали з файлами і директоріями, так як має корисні функції для цього. У цій функції також наперед створюються папки для наборів даних data/raw та data/processed, тому користувачу не потрібно створювати їх вручну.

```
if __name__ == '__main__':
    start = time.time()
    print(f"-------\nStarting fetching data script...\n-------")
    pd.set_option('display.max_rows', 30)
    pd.set_option('display.max_columns', 50)
    pd.set_option('display.width', 5000)

    path = Path("../data/raw")
    path.mkdir(parents=True, exist_ok=True)
    Path("../data/processed").mkdir(parents=True, exist_ok=True)
    cache_path = Path.home() / ".cache/kagglehub/datasets"
    get_movies(path, cache_path)

    end = time.time()
    duration = end - start
    print(f"-------\nFetching script duration: {duration:0.3f} seconds\n-------")
```

Рисунок 4.1 – Функція main() для fetch_data.py

Функцію get_movies_main() для завантаження основного набору даних представлено на рисунку 4.2. У ній прописано перевірку – чи дійсно потрібного набору даних немає у директорії, куди за замовчуванням

завантажуються набори даних з Kaggle. Якщо файл є, то на екран виводиться повідомлення про його існування і програма переходить до завантаження наступних наборів. Якщо ж файлу немає, то починається процес його завантаження. Якщо виникне помилка при завантаженні, то на екран виведеться відповідне повідомлення і програма закінчить свою роботу. Після завантаження цей файл також копіюється у папку data/raw. Якщо при копіюванні виникла якась помилка, то програма сповістить користувача про це і виведе необхідне повідомлення на екран. Програма не завершує свою роботу, як у випадку з помилкою завантаження набору, так як ще залишається опція ручного копіювання набору у цю папку.

```
def get_movies_main(path, cache_path): #imdb dataset  1 usage
    print(f"-------\nDownloading is about to start...\n-------")
    new_path = path
    cache_dir = Path(cache_path)
    try:
        if (cache_dir / "dk123891").is_dir():
            print(f"Main dataset is already on your device..\n")
        else:
            print(f"Downloading the main dataset..\n")
            actual_path = kagglehub.dataset_download("dk123891/10000-movies-data")
            actual_path = os.path.join(actual_path, "data.csv")
            dest_path = os.path.join(new_path)
            try:
                shutil.copy(actual_path, dest_path)
            except FileNotFoundError:
                print(f"There is no such file: {actual_path}")
            except Exception as e:
                print(f"Copy error: {e}")
    except Exception as e:
        print(f"Failed to load main dataset: {e}\nClosing the program...")
        sys.exit(1)
    handle_movies_main(path, cache_path)
```

Рисунок 4.2 – Функція завантаження основного набору даних

Після цього викликається функція handle_movies_main() для первинної обробки – відбору необхідних колонок і видалення дублікатів екземплярів з однаковою назвою (див. рис. 4.3). Також тут створюється маска для отримання усіх назв фільмів, а також назв фільмів, які мають пропущені значення у

колонках рейтингу MetaScore, сертифікації і доходів фільму. Дані цих колонок будуть шукатись у інших датасетах, як обов'язкові колонки, у наступних функціях. А також з інших наборів буде потім завантажено додаткові колонки місяця, дня, мови та країни. Функція первинної обробки викликає у собі функцію для завантаження додаткових наборів даних (див. рис. 4.4).

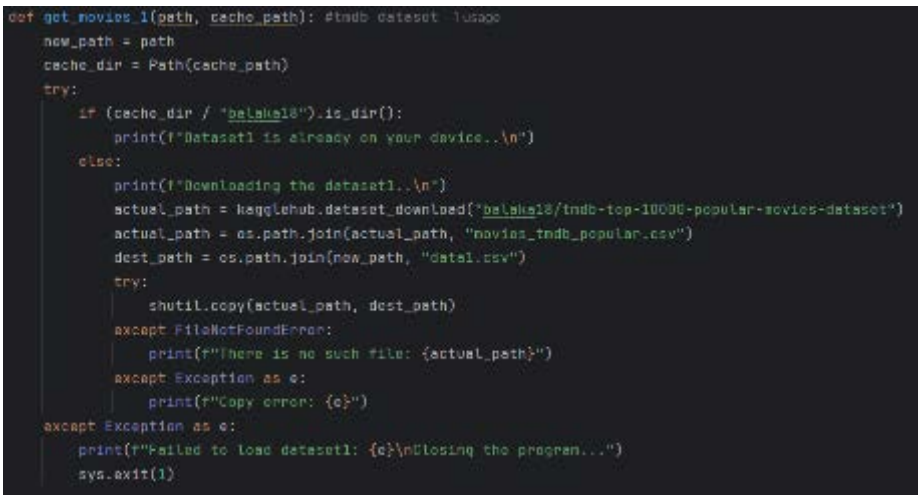
```
def handle_movies_main(path, cache_path):  1 usage
    df = pd.read_csv(os.path.join(path, "data.csv"))
    df.rename( mapper: {'Movie Name': 'Title', 'Year of Release': 'Year', 'Run Time in minutes': 'Runtime',
               'Movie Rating': 'IMDB_rating', 'Votes': 'IMDB_Votes', 'Gross': 'Box_Office'}, axis=1, inplace=True)
    df.drop_duplicates( subset: ['Title'], keep='first', inplace=True)

    mask_metascore = df['MetaScore'].isna()
    isna_metascore = df[mask_metascore]['Title']
    mask_box_office = df['Box_Office'].isna()
    isna_box_office = df[mask_box_office]['Title']
    mask_certification = df['Certification'].isna()
    isna_certification = df[mask_certification]['Title']
    mask_titles = df['Title']
    masks = [mask_titles, isna_metascore, isna_box_office, isna_certification]
    get_movies_additionally(path, cache_path, masks)
```

Рисунок 4.3 – Функція для первинної обробки

```
def get_movies_additionally(path, cache_path, masks):  1 usage
    print(f"-------\nAttempting to download additional necessary datasets...\n-------")
    get_movies_1(path, cache_path)
    get_movies_2(path, cache_path)
    get_movies_3(path, cache_path)
    get_movies_4(path, masks)
```

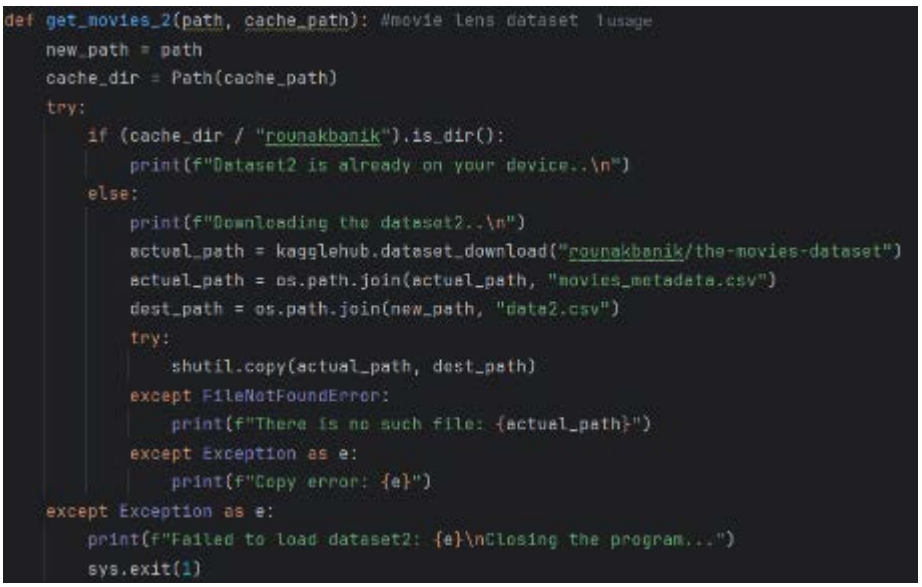
Рисунок 4.4 – Функція для завантаження додаткових наборів

На рисунках 4.5-4.7 представлено функції для завантаження додаткових наборів з kaggle. Для цього знову прописані перевірки на наявність цих наборів у директорії за замовчуванням, успішність завантаження та копіювання набору у data/raw.



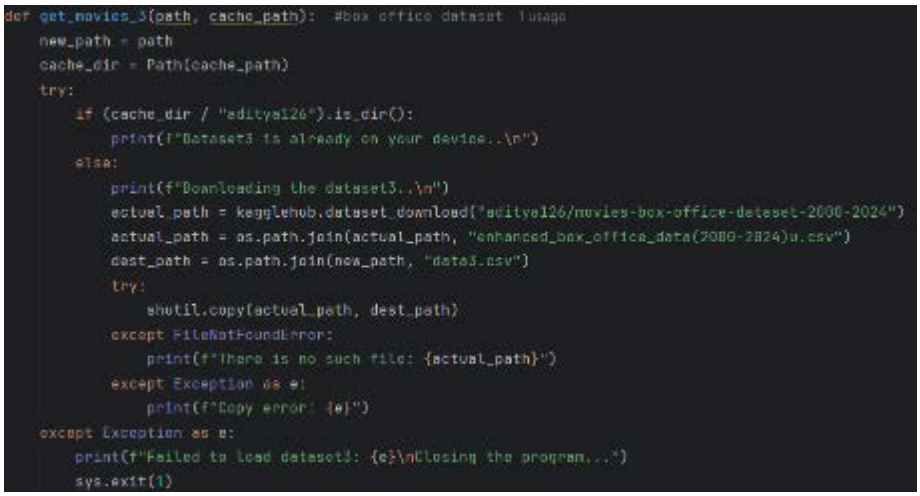
```
def get_movies_1(path, cache_path): #tmdb dataset 1usage
    new_path = path
    cache_dir = Path(cache_path)
    try:
        if (cache_dir / "batuhan18").is_dir():
            print(f"Dataset1 is already on your device..\n")
        else:
            print(f"Downloading the dataset1..\n")
            actual_path = kagglehub.dataset_download("batuhan18/tmdb-top-10000-popular-movies-dataset")
            actual_path = os.path.join(actual_path, "movies_tmdb_popular.csv")
            dest_path = os.path.join(new_path, "data1.csv")
            try:
                shutil.copy(actual_path, dest_path)
            except FileNotFoundError:
                print(f"There is no such file: {actual_path}")
            except Exception as e:
                print(f"Copy error: {e}")
    except Exception as e:
        print(f"Failed to load dataset1: {e}\nClosing the program...")
        sys.exit(1)
```

Рисунок 4.5 – Завантаження додаткового набору даних



```
def get_movies_2(path, cache_path): #movie lens dataset 1usage
    new_path = path
    cache_dir = Path(cache_path)
    try:
        if (cache_dir / "rounakbanik").is_dir():
            print(f"Dataset2 is already on your device..\n")
        else:
            print(f"Downloading the dataset2..\n")
            actual_path = kagglehub.dataset_download("rounakbanik/the-movies-dataset")
            actual_path = os.path.join(actual_path, "movies_metadata.csv")
            dest_path = os.path.join(new_path, "data2.csv")
            try:
                shutil.copy(actual_path, dest_path)
            except FileNotFoundError:
                print(f"There is no such file: {actual_path}")
            except Exception as e:
                print(f"Copy error: {e}")
    except Exception as e:
        print(f"Failed to load dataset2: {e}\nClosing the program...")
        sys.exit(1)
```

Рисунок 4.6 – Завантаження додаткового набору даних



```
def get_movies_3(path, cache_path):  #box office dataset 1usage
    new_path = path
    cache_dir = Path(cache_path)
    try:
        if (cache_dir / "aditya126").is_dir():
            print(f"Dataset3 is already on your device..\n")
        else:
            print(f"Downloading the dataset3..\n")
            actual_path = kagglehub.dataset_download("aditya126/movies-box-office-dataset-2000-2024")
            actual_path = os.path.join(actual_path, "enhanced_box_office_data(2000-2024)u.csv")
            dest_path = os.path.join(new_path, "data3.csv")
            try:
                shutil.copy(actual_path, dest_path)
            except FileNotFoundError:
                print(f"There is no such file: {actual_path}")
            except Exception as e:
                print(f"Copy error: {e}")
    except Exception as e:
        print(f"Failed to load dataset3: {e}\nClosing the program...")
        sys.exit(1)
```

Рисунок 4.7 – Завантаження додаткового набору даних

На рисунку 4.8 представлено API ключ, який отримано за допомогою підписки на Patreon, для завантаження четвертого додаткового набору. На рисунку 4.9 представлено завантаження четвертого додаткового набору даних. У цій функції завантажуються дані тільки тих фільмів, які є у основному наборі даних. Для цього використовується перша змінна у списку masks, яка відповідає за назви фільмів з основного набору. У функції також знову прописані перевірки на наявність цього набору у папці data/raw, успішність завантаження та завантаження набору у data/raw. Також при завантаженні на екран виводиться прогрес завантаження, щоб користувач міг розуміти скільки приблизно часу на це піде.

```
API_KEY_OMDB = '13b2754f'
url_omdb = 'https://www.omdbapi.com/'
```

Рисунок 4.8 – API ключ для четвертого набору даних

```
def get_movies_4(path, masks): #omdb dataset 1usage
    new_path = os.path.join(path, 'data4.json')
    file_dir = Path(path)

    result = []
    title_suc, title_skip = 0, 0
    title_count = len(masks[0])
    try:
        if (file_dir / "data4.json").is_file():
            print(f"Dataset4 is already on your device..\n")
        else:
            print(f"Downloading the dataset4..\n")
            for i, title in enumerate(masks[0]):
                params_omdb = {'apikey': API_KEY_OMDB, 't': title}
                try:
                    response = requests.get(url_omdb, params=params_omdb)
                    response.raise_for_status()
                    data_omdb = response.json()
                    if data_omdb.get('Response') == 'False':
                        title_skip += 1
                    else:
                        title_suc += 1
                        output = f"{i+1}) Title {title} was successfully loaded. {i+1}/{title_count}"
                        print(f"\r{output:<100}", end="")
                    result.append(data_omdb)
                    time.sleep(0.2)
                except requests.exceptions.HTTPError as e:
                    print(f"\nFailed to load dataset4, HTTP error occurred: {e}\nClosing the program...")
                    sys.exit(1)
                except Exception as e:
                    print(f"\nFailed to load dataset4: {e}\nClosing the program...")
                    sys.exit(1)
            print(f"\n{result[:10]}\n")
            print(f"Successfully downloaded titles: {title_suc}")
            print(f"Skipped titles: {title_skip}")
            try:
                with open(new_path, 'w', encoding='utf-8') as f:
                    json.dump(result, f, indent=4, ensure_ascii=False)
                print(f"Dataset4 was successfully created.")
            except Exception as e:
                print(f"Failed to save dataset4: {e}\nClosing the program...")
                sys.exit(1)
    except Exception as e:
        print(f"Failed to load dataset4: {e}\nClosing the program...")
        sys.exit(1)
```

Рисунок 4.9 – Завантаження четвертого додаткового набору даних

На рисунку 4.10 представлено основну функцію main() модулю preprocessing.py, який відповідає за попередню обробку даних.

```
def main():
    start = time.time()
    print(f"-------\nStarting preprocessing script...\n-------")
    pd.set_option('display.max_rows', 50)
    pd.set_option('display.max_columns', 50)
    pd.set_option('display.width', 5000)

    path = "../data/raw"
    if Path(path).is_dir():
        print(f"Necessary directory exists, preparing for preprocessing...")
        handle_movies_main(path)
    else:
        print(f"Error: Necessary directory cannot be found, please run fetch_data.py script")
        sys.exit(1)
    end = time.time()
    duration = end - start
    print(f"-------\nPreprocessing script duration: {duration:0.3f} seconds\n-------")
```

Рисунок 4.10 – Функція main() функції попередньої обробки даних

Тут проводиться підрахунок часу на виконання, а також перевірка на наявність папки data/raw. Після чого викликається функція handle_movies_main(), яку представлено на рисунку 4.11.

```
def handle_movies_main(path):  1 usage
    print(f"-------\nMain Dataset\n-------")
    if (Path(path) / 'data.csv').is_file():
        print(f"Necessary dataset exists, preparing for preprocessing...")
    else:
        print(f"Error: Necessary dataset cannot be found, please run fetch_data.py script")
        sys.exit(1)

    df = pd.read_csv(os.path.join(path, "data.csv"))
    df.drop(columns=['Unnamed: 0', 'Description'], inplace=True)
    df.rename( mapper={'Movie Name': 'Title', 'Year of Release': 'Year', 'Run time in minutes': 'Runtime',
               'Movie Rating': 'IMDB_rating', 'Votes': 'IMDB_Votes', 'Gross': 'Box_Office'}, axis=1, inplace=True)
    print(f"\nDataset example:\n{df.head(5)}\n")

    df.drop_duplicates( subset ['Title'], keep='first', inplace=True)
    print(f"Duplicates have been dropped...\n")
    print(f"All not a number values from dataset:\n{df.isna().sum()}\n")

    mask_metascore = df['MetaScore'].isna()
    isna_metascore = df[mask_metascore]['Title']
    mask_box_office = df['Box_Office'].isna()
    isna_box_office = df[mask_box_office]['Title']
    mask_certification = df['Certification'].isna()
    isna_certification = df[mask_certification]['Title']
    mask_titles = df['Title']
    masks = [mask_titles, isna_metascore, isna_box_office, isna_certification]
    get_additional_columns(path, masks, df)
```

Рисунок 4.11 – Функція handle_movies_main()

Ця функція відповідає за перейменування необхідних колонок основного набору, вивід на екран усіх колонок з основного набору, де є пропущені значення(рейтинг MetaScore, сертифікація і доходи фільму). Потім викликається функція для отримання додаткових колонок місяця, дня, мови та країни фільму та об’єднання наборів (див. рис. 4.12).

```
def get_additional_columns(path, masks, df):  1 usage
    df1 = get_additional_columns1(path, masks)
    df2 = get_additional_columns2(path, masks)
    df3 = get_additional_columns3(path, masks)
    df4 = get_additional_columns4(path, masks)
    datasets_join(df, df1, df2, df3, df4, masks)
```

Рисунок 4.12 – Функція заповнення пропущених значень

На рисунках 4.13–4.16 представлено функції для отримання додаткових колонок з інших наборів.

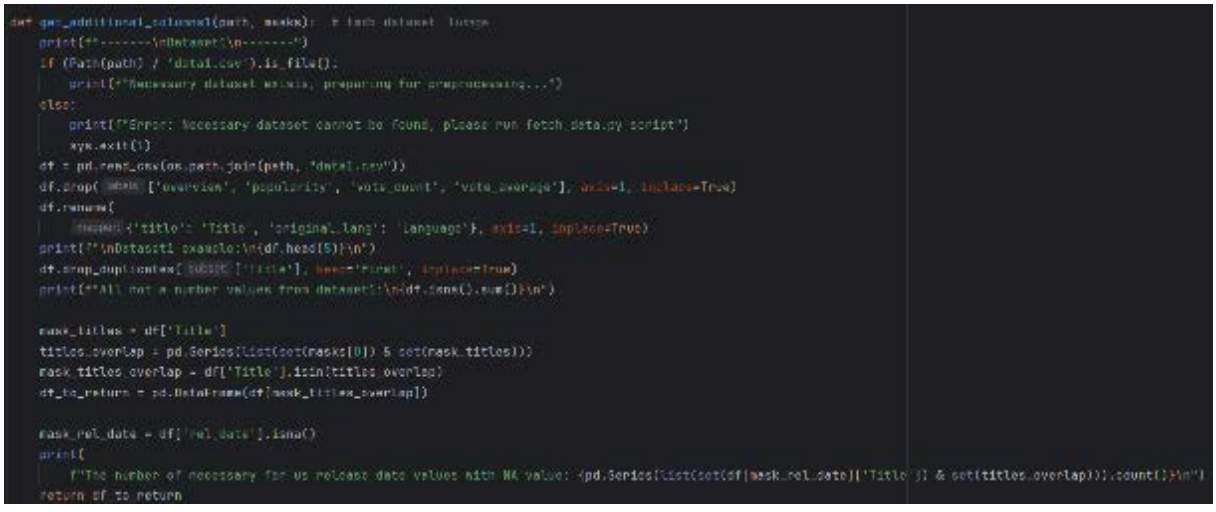


Рисунок 4.13 – Функція отримання додаткових колонок

Рисунок 4.14 – Функція отримання додаткових колонок

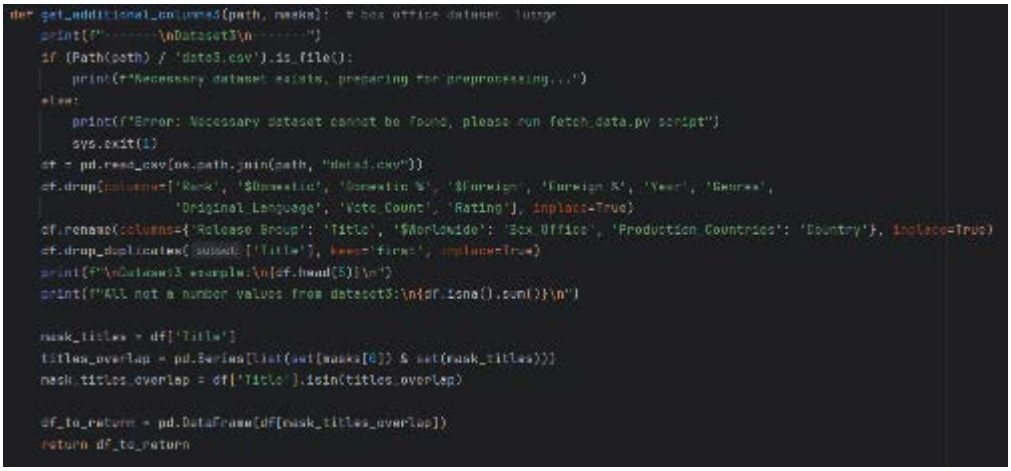


Рисунок 4.15 – Функція отримання додаткових колонок

Рисунок 4.16 – Функція отримання додаткових колонок

У них проводиться перевірка чи існують відповідні набори даних у data/raw. Якщо ні – то виводиться відповідне повідомлення і програма завершує роботу. Потім з додаткового набору вибираються потрібні колонки для нашого основного. У кінці на екран виводиться кількість пропущених значень для необхідних нам колонок.

На рисунку 4.17 представлено функцію datasets_join(), яка відповідає за виведення інформації про кількість фільмів у п’ятьох отриманих наборах даних, а також викликає функції для заповнення пропущених значень даними з додаткових наборів та злиття наборів.

```
def datasets_join(df, df1, df2, df3, df4, masks):  1 usage
    print(f"-------\nJoining the datasets..\n-------")
    print(f"\nMain dataset example:\n{df.head(3)}\n")
    print(f"\nDataset1 example:\n{df1.head(3)}\n")
    print(f"\nDataset2 example:\n{df2.head(3)}\n")
    print(f"\nDataset3 example:\n{df3.head(3)}\n")
    print(f"\nDataset4 example:\n{df4.head(3)}\n")
    print(f"Number of movies in main dataset: {len(df.index)}")
    print(f"Number of movies in dataset1: {len(df1.index)}")
    print(f"Number of movies in dataset2: {len(df2.index)}")
    print(f"Number of movies in dataset3: {len(df3.index)}")
    print(f"Number of movies in dataset4: {len(df4.index)}")
    additional_columns = add_additional_columns(df, df1, df2, df3, df4, masks)
    necessary_columns = add_necessary_columns(df, df2, df3, df4, masks)
    merge_dfs(df, additional_columns, necessary_columns, masks)
```

Рисунок 4.17 – Функція datasets_join()

Функція add_additional_columns відповідає за додавання нових колонок і викликає у собі відповідні функції для кожної (див. рис. 4.18).

```
def add_additional_columns(df, df1, df2, df3, df4, masks):  1 usage
    print(f"-------\nAdditional columns..\n-------")
    language = add_original_lg(df, df1, df4, masks)
    country = add_country(df, df3, df4, masks)
    day, month = add_release_date(df, df1, df2, df4, masks)
    additional_columns = pd.DataFrame({'Day': day, 'Month': month, 'Language': language, 'Country': country})
    return additional_columns
```

Рисунок 4.18 – Функція add_additionl_columns()

На рисунках 4.19-4.21 представлено функції для отримання додаткових колонок дня, місяця, мови і країни. У відповідних колонках викидуються пропущені значення, проводиться зміна типів даних, заміна значень, а потім створюється об’єкт типу series для завантаження у набір. У функції для отримання днів і місяців, спочатку обирається колонка дати релізу, яка потім розділяється на день і місяць.

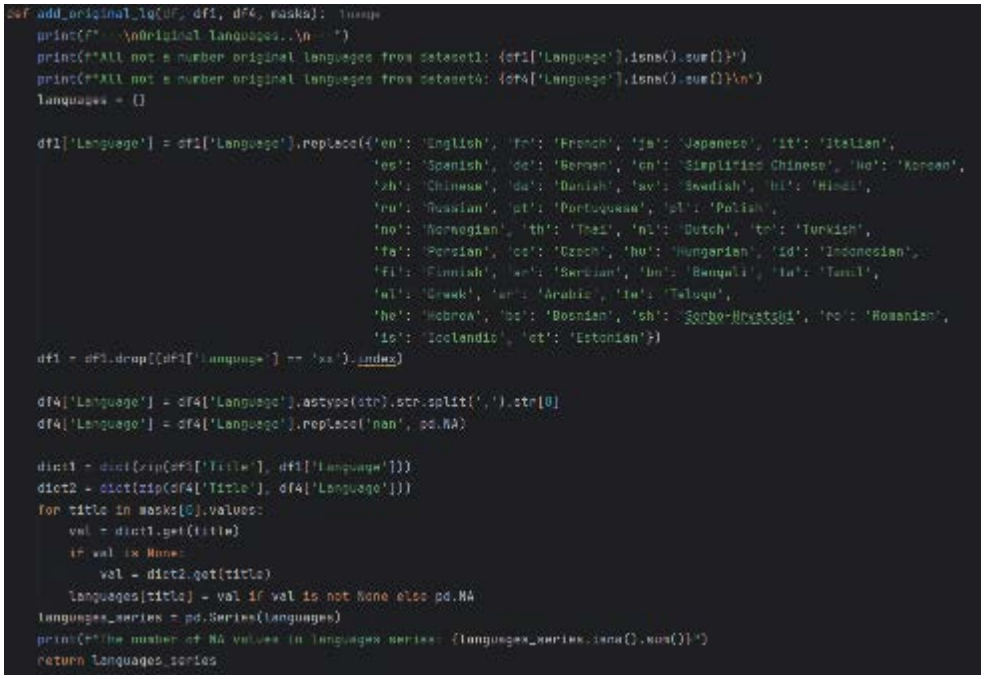


Рисунок 4.19 – Функція для додаткової колонки мови

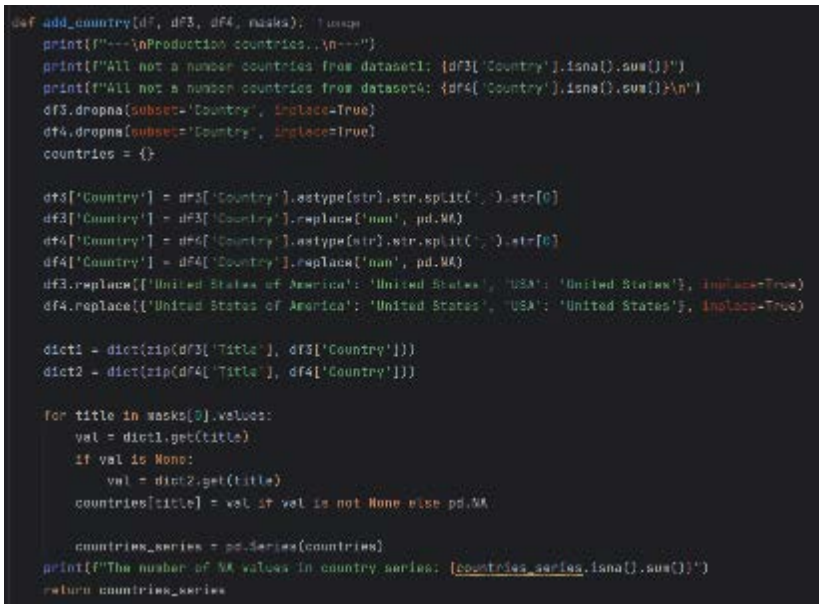


Рисунок 4.20 – Функція для додаткової колонки країни

```python
def add_release_date(df, df1, df2, df4, masks):  1 usage
    print(f"---\nRelease dates..\n---")
    print(f"All not a number release dates from dataset1: {df1['rel_date'].isna().sum()}")
    print(f"All not a number release dates from dataset2: {df2['release_date'].isna().sum()}")
    print(f"All not a number release dates from dataset4: {df4['Released'].isna().sum()}\n")
    df4.dropna(subset=['Released'], inplace=True)

    days, months = {}, {}
    df1_day, df1_month, df2_day, df2_month, df4_day, df4_month = {}, {}, {}, {}, {}, {}

    df1['rel_date'] = df1['rel_date'].astype(str)
    df2['release_date'] = df2['release_date'].astype(str)
    df4['Released'] = df4['Released'].astype(str)

    dict1 = dict(zip(df1['Title'], df1['rel_date']))
    dict2 = dict(zip(df2['Title'], df2['release_date']))
    dict3 = dict(zip(df4['Title'], df4['Released']))

    for elem in df4.values.tolist():
        df4_day.update({elem[0]: elem[2][:2]})
        df4_month.update({elem[0]: elem[2][3:6]})
    for elem in df1.values.tolist():
        df1_day.update({elem[0]: elem[2][8:]})
        df1_month.update({elem[0]: elem[2][5:7]})
    for elem in df2.values.tolist():
        df2_day.update({elem[2]: elem[0][8:]})
        df2_month.update({elem[2]: elem[0][5:7]})

    for title in masks[0].values:
        if dict1.get(title) is not None:
            days[title] = df1_day.get(title)
            months[title] = df1_month.get(title)
        elif dict2.get(title) is not None:
            days[title] = df2_day.get(title)
            months[title] = df2_month.get(title)
        elif dict3.get(title) is not None:
            days[title] = df4_day.get(title)
            months[title] = df4_month.get(title)
        else:
            days[title] = pd.NA
            months[title] = pd.NA

        days_series = pd.Series(days)
        months_series = pd.Series(months)
    print(f"The number of NA values in days: {days_series.isna().sum()}")
    print(f"The number of NA values in months: {months_series.isna().sum()}")
    return days_series, months_series
```

Рисунок 4.21 – Функція для додаткових колонок дня і місяця

На рисунку 4.22 представлено функцію для заповнення пропущених значень, яка в собі викликає функцію для кожної колонки пропущеного значення.

```python
def add_necessary_columns(df, df2, df3, df4, masks):  1 usage
    print(f"-------\nNecessary columns..\n-------")
    rated = add_certification(df, df4, masks)
    box_office = add_box_office(df, df2, df3, df4, masks)
    metascore = add_metascore(df, df4, masks)
    necessary_columns = pd.DataFrame({'Certification': rated, 'Box_Office': box_office, 'MetaScore': metascore})
    return necessary_columns
```

Рисунок 4.22 – Загальна функція для заповнення пропущених значень

На рисунках 4.23–4.25 представлено функції для заповнення пропущених значень з трьох колонок основного набору.

```
def add_certification(df, df4, masks):  1 usage
    print(f"---\nCertification..\n---")
    print(f"All not a number certifications from dataset4: {df4['Certification'].isna().sum()}\n")

    dict1 = dict(zip(df4['Title'], df4['Certification']))
    rated = {}
    for title in masks[0].values:
        val = dict1.get(title)
        rated[title] = val if val is not None else pd.NA
        rated_series = pd.Series(rated)
    print(f"The number of NA values in rated series: {rated_series.isna().sum()}")
    return rated_series
```

Рисунок 4.23 – Функція для заповнення пропущених значень сертифікації

```
def add_box_office(df, df2, df3, df4, masks):  1 usage
    print(f"---\nBox Office..\n---")
    print(f"All not a number box offices from dataset2: {df2['Box_Office'].isna().sum()}")
    print(f"All not a number box offices from dataset3: {df3['Box_Office'].isna().sum()}")
    print(f"All not a number box offices from dataset4: {df4['BoxOffice'].isna().sum()}\n")
    df3.dropna(subset=['Box_Office'], inplace=True)

    dict1 = dict(zip(df2['Title'], df2['Box_Office']))
    dict2 = dict(zip(df3['Title'], df3['Box_Office']))
    dict3 = dict(zip(df4['Title'], df4['BoxOffice']))
    box_office = {}

    for title in masks[0].values:
        val = dict1.get(title)
        if val is None:
            val = dict2.get(title)
        if val is None:
            val = dict3.get(title)
        box_office[title] = val if val is not None else pd.NA
        box_office_series = pd.Series(box_office)
    print(f"The number of NA values in box office series: {box_office_series.isna().sum()}")
    return box_office_series
```

Рисунок 4.24 – Функція для заповнення пропущених значень доходів

```
def add_metascore(df, df4,masks):  1 usage
    print(f"---\nMetascore..\n---")
    print(f"All not a number metascores from dataset4: {df4['MetaScore'].isna().sum()}\n")

    dict1 = dict(zip(df4['Title'], df4['MetaScore']))
    metascore = {}

    for title in masks[0].values:
        val = dict1.get(title)
        metascore[title] = val if val is not None else pd.NA
        metascore_series = pd.Series(metascore)
    print(f"The number of NA values in metascore series: {metascore_series.isna().sum()}")
    return metascore_series
```

Рисунок 4.25 – Функція для заповнення пропущених значень рейтингу metascore

На рисунку 4.26 представлено частину коду функції merge_dfs(). Це його фінальна частина, де проводиться об’єднання всіх значень у один набір,

а також виклик функції to_processed() для збереження цього набору у data/proessed.

Рисунок 4.26 – Частина коду функції для об’єднання наборів у один

На рисунку 4.27 представлено головну функцію main() у модулі to_processed.py для завантаження отриманого набору даних у data/processed. У ній передбачено перевірку наявності файлу у папці. Якщо він вже існує, то можна обрати перезаписати. Якщо ж файла не існує, то його буде автоматично завантажено. Також передбачено перевірку на успішність збереження. Під кінець на екран виводяться відповідні повідомлення і підраховується кількість витраченого часу.

Рисунок 4.27 – Функція для завантаження даних на пристрій

На рисунку 4.28 представлено функцію main() модулю upload_to_s3.py, який відповідає за завантаження набору даних з data/processed на AWS S3.

```
def main():
    start = time.time()
    print(f"-------\nStarting AWS S3 script...\n-------")
    path = Path('../data/processed') / 'processed_data.csv'
    if path.is_file():
        print(f"File processed_data.csv exists, preparing to upload...")
        upload_file_to_s3(path, s3_file_name: None)
    else:
        print(f"Error: file processed_data.csv doesn't exist. Stopping the program..")
        sys.exit(1)
    end = time.time()
    duration = end - start
    print(f"\nAWS S3 script duration: {duration:0.3f} seconds")
```

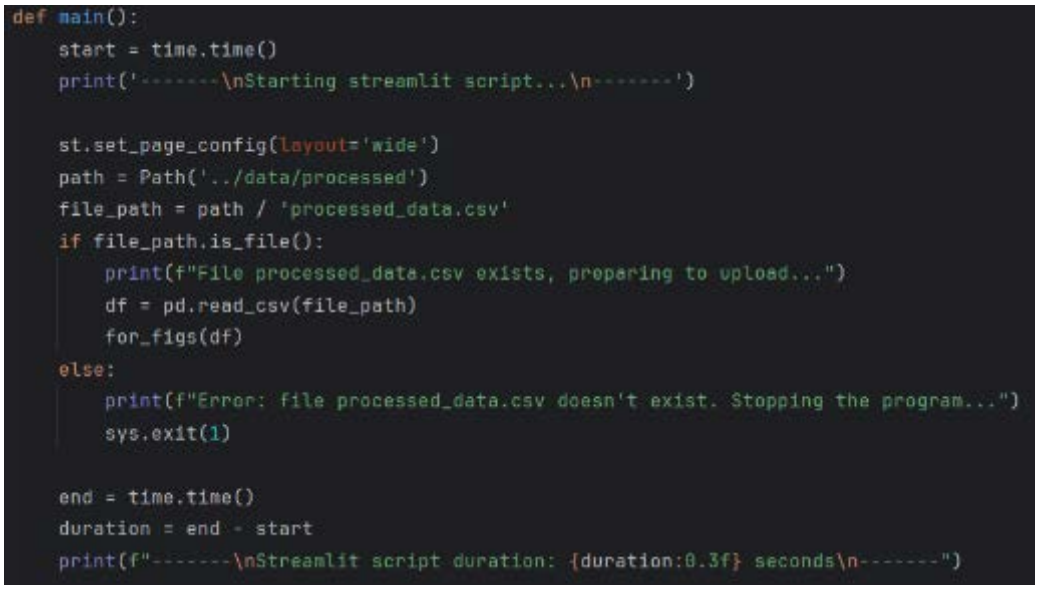
Рисунок 4.28 – Функція main() для завантаження на AWS S3

У ній проводиться перевірка на наявність файлу набору даних у data/processed, якщо файлу не існує, то на екран виводиться відповідне повідомлення і програма завершує свою роботу. Якщо файл існує, то викликається функція для завантаження даних. Тут проводиться декілька перевірок для .env файлу, у якому треба вписати свої дані ключів, бакету та регіону. На екран буде виведено відповідні повідомлення у разі успішності або неуспішності виконання операції. Також на рисунку 4.29 представлено функцію для беспросередньо завантаження даних на AWS S3.

На рисунку 4.30 представлено основну функцію main() для виведення графіків на вебсторінку за допомогою бібліотеки streamlit у модулі to_streamlit.py. У функції проводиться перевірка чи існує необхідний файл обробленого набору даних у data/processed. Якщо ні, то на екран виводиться відповідне повідомлення та програма завершує роботу. Якщо файл існує, то викликається функція для створення графіків і відповідної вебсторінки for_figs().

```python
def upload_file_to_s3(file_path, s3_file_name=None):  1 usage
    bucket_name = os.getenv('S3_BUCKET_NAME')
    aws_access_key = os.getenv('AWS_ACCESS_KEY_ID')
    aws_secret_key = os.getenv('AWS_SECRET_ACCESS_KEY')
    region = os.getenv('AWS_REGION')

    if region == 'Your_aws_region' or not region:
        print("Error: You haven't configured the aws_region in .env file yet.")
        return False
    elif bucket_name == 'You_bucket_name' or not bucket_name:
        print("Error: You haven't configured the bucket_name in .env file yet.")
        return False
    elif aws_access_key == 'Your_access_key' or not bucket_name:
        print("Error: You haven't configured the aws_access_key in .env file yet.")
        return False
    elif aws_secret_key == 'Your_secret_access_key' or not bucket_name:
        print("Error: You haven't configured the aws_secret_key in .env file yet.")
        return False
    if s3_file_name is None:
        s3_file_name = os.path.basename(file_path)
    print(f"Uploading '{file_path}' in the bucket '{bucket_name}'...")

    try:
        s3_client = boto3.client(
            *args: 's3',
            aws_access_key_id=aws_access_key,
            aws_secret_access_key=aws_secret_key,
            region_name=region
        )
        s3_client.upload_file(file_path, bucket_name, s3_file_name)
        print(f"File was successfully uploaded to AWS. File name: {s3_file_name}")
        return True

    except FileNotFoundError:
        print(f"Error: File {file_path} not found.")
        return False
    except NoCredentialsError:
        print("Error: Wrong AWS access keys.")
        return False
    except ClientError as e:
        print(f"Error: {e}")
        return False
    except InvalidRegionError:
        print(f"Error: invalid aws region format: {region}. Please check your .env file.")
        return False
    except Exception as e:
        print(f"Unexpected error: {e}")
```

Рисунок 4.29 – Функція для завантаження на AWS S3



```python
def main():
    start = time.time()
    print('-------\nStarting streamlit script...\n-------')

    st.set_page_config(layout='wide')
    path = Path('../data/processed')
    file_path = path / 'processed_data.csv'
    if file_path.is_file():
        print(f"File processed_data.csv exists, preparing to upload...")
        df = pd.read_csv(file_path)
        for_figs(df)
    else:
        print(f"Error: file processed_data.csv doesn't exist. Stopping the program...")
        sys.exit(1)

    end = time.time()
    duration = end - start
    print(f"-------\nStreamlit script duration: {duration:0.3f} seconds\n-------")
```

Рисунок 4.30 – Функція main() для отримання графіків через streamlit

Функція for_figs() відповідає за створення графіків і розміщення їх на вебсторінці. Графіки виводяться у дві колонки та можуть відображати значення при наведенні миші та змінювати масштаб. Частину цієї функції представлено на рисунку 4.31.

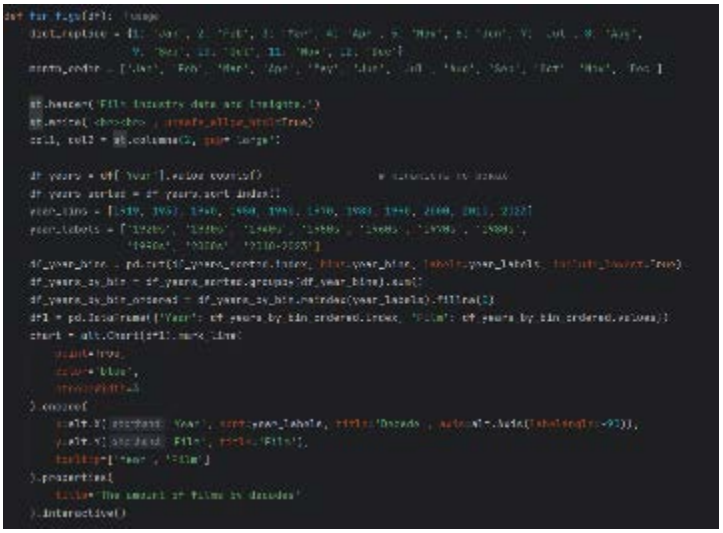


Рисунок 4.31 – Функція для отримання графіків через streamlit

На рисунку 4.32 представлено функцію main() для створення графіків у модулі trends.py. У ній проводиться перевірка чи існує файл набору даних у папці data/processed. Якщо ні, то на екран виводиться відповідне повідомлення та програма завершує роботу. Якщо файл існує то викликається функція for_figs(), частина якої представлена на рисунку 4.33.

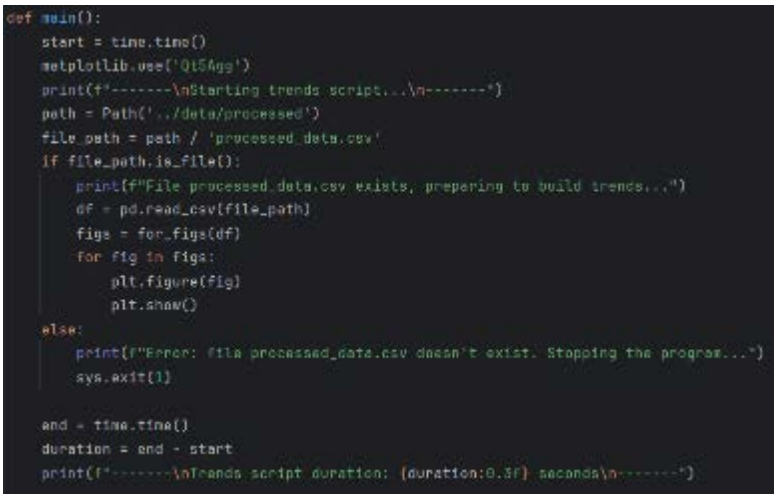


Рисунок 4.32 – Функція main() для створення графіків у trends.py

```python
def for_figs(df):  1 usage
    figs = []
    dict_replace = {1: 'Jan', 2: 'Feb', 3: 'Mar', 4: 'Apr', 5: 'May', 6: 'Jun', 7: 'Jul', 8: 'Aug',
                    9: 'Sep', 10: 'Oct', 11: 'Nov', 12: 'Dec'}
    month_order = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']
    fig4_colors = ["#e4a5ff", "#deabff", "#d8b1ff", "#d1b7ff", "#cbbdff", "#c5c4ff", "#bfcaff", "#b8d0ff", "#b2d6ff",
                   "#acdcff"]

    # ----------First slide
    fig, ax_array = plt.subplots( nrows: 2,  ncols: 1, figsize=(16, 10))  # кількість по роках
    plt.subplots_adjust(hspace=0.4)
    df_years = df['Year'].value_counts()
    df_years_sorted = df_years.sort_index()
    year_bins = [1919, 1930, 1940, 1950, 1960, 1970, 1980, 1990, 2000, 2010, 2023]
    year_labels = ['1920s', '1930s', '1940s', '1950s', '1960s', '1970s', '1980s',
                   '1990s', '2000s', '2010-2023']
    df_year_bins = pd.cut(df_years_sorted.index, bins=year_bins, labels=year_labels, include_lowest=True)
    df_years_by_bin = df_years_sorted.groupby(df_year_bins).sum()
    bin_labels = [str(interval) for interval in df_years_by_bin.index]
    bar_heights = df_years_by_bin.values
    bars1 = ax_array[0].bar(bin_labels, bar_heights, color='#476591')
    ax_array[0].bar_label(bars1, fmt='%d', padding=.5, fontsize=10)
    ax_array[0].set_ylabel('The amount of films', fontsize=14)
    ax_array[0].set_title('Films by years', fontsize=16, fontweight='bold')
    ax_array[0].set_xlabel('Year', fontsize=14)

    df_countries = df['Country'].value_counts()  # кількість по топ 10 країнах
    bars2 = ax_array[1].barh(df_countries.index.tolist()[:10], df_countries.values.tolist()[:10], color='#476591')
    ax_array[1].bar_label(bars2, fmt='%d', padding=.5, fontsize=10)
    ax_array[1].set_ylabel('The amount of films', fontsize=14)
    ax_array[1].set_title('Films by top-10 countries', fontsize=16, fontweight='bold')
    ax_array[1].set_xlabel('Country', fontsize=14)
    fig1 = fig
    figs.append(fig1)
    plt.close(fig1)
```

Рисунок 4.33 – Функція для отримання графіків

На рисунку 4.34 представлено функцію main() у модулі models.py, який відповідає за створення моделей машинного навчання та нейронних мереж для прогнозування рейтингів та дохідності фільмів. У функції передбачено перевірка на існування набору даних у data/processed. Якщо ні, то на екран виводиться відповідне повідомлення та програма завершує роботу. Якщо набір існує, то він зчитується у програму. Після цього викликається функція для підготовки деяких колонок у датасеті. Її представлено на рисунку 4.35. У наборі залишаються перші 70 режисерів по кількості фільмів, інші замінюються на категорію «Інші». Те ж саме відбувається і з колонками країни і мови. Колонка рейтингу тепер є категоріальної з трьома класами.

```python
def main():
    start = time.time()
    matplotlib.use('TkAgg')
    print(f"-------\nStarting models script...\n-------")
    pd.set_option('display.max_rows', 3000)
    pd.set_option('display.max_columns', 50)
    pd.set_option('display.width', 5000)
    path = Path('../data/processed')
    file_path = path / 'processed_data.csv'
    try:
        if file_path.is_file():
            print(f"Dataset was successfully found.")
            df = pd.read_csv(file_path)
        else:
            print(f"Failed to found dataset.")
            sys.exit(1)
    except Exception as e:
        print(f"Error: {e}")

    df = prepare_dataset(df)
    feature_importance(df)

    predict_rating_class_ml(df)
    predict_rating_class_nn(df)

    predict_box_office_ml(df)
    predict_box_office_nn(df)

    end = time.time()
    duration = end - start
    print(f"\nModels script duration: {duration:0.3f} seconds")
```

Рисунок 4.34 – Функція main() у модулі models.py

```python
def prepare_dataset(df):  1 usage
    df.drop(columns=['Title'], inplace=True)
    compress_column(df, column_name: 'Director', top_n: 70, other_label: 'Another')
    compress_column(df, column_name: 'Country', top_n: 15, other_label: 'Another')
    compress_column(df, column_name: 'Language', top_n: 10, other_label: 'Another')

    cols_to_encode = ['Director', 'Country', 'Language', 'Certification', 'Rating_Class']
    for col in cols_to_encode:
        df[col] = pd.factorize(df[col])[0]
    return df
```

Рисунок 4.35 – Підготовка колонок для важливостей ознак

На рисунку 4.36 представлено функцію для створення графіків важливостей ознак для колонки рейтингу і для колонки доходів.

```
def feature_importance(df):  1 usage
    print(f"---\nFeature importances..\n---")
    print(f"Example:\n{df.head(1)}\n")

    x_box_office = df.drop(columns=['Box_Office'])
    y_box_office = df['Box_Office']
    x_rating = df.drop(columns=['IMDB_rating', 'Rating_Class'])
    y_rating = df['Rating_Class']
    x_box_train, x_box_test, y_box_train, y_box_test = train_test_split( *arrays: x_box_office, y_box_office, test_size=0.2,
                                                                         shuffle=True,random_state=45)
    x_rating_train, x_rating_test, y_rating_train, y_rating_test = train_test_split( *arrays: x_rating, y_rating, test_size=0.2,
                                                                           shuffle=True, random_state=45)


    cols_to_scale = ['Year', 'Runtime', 'MetaScore', 'IMDB_Votes', 'IMDB_rating']
    scaler = StandardScaler()
    scaler.fit(x_box_train[cols_to_scale])
    x_box_train.loc[:, cols_to_scale] = scaler.transform(x_box_train[cols_to_scale])
    x_box_test.loc[:, cols_to_scale] = scaler.transform(x_box_test[cols_to_scale])

    forest_box = RandomForestRegressor(n_estimators=50)
    y_box_train_log = np.log1p(y_box_train)
    forest_box.fit(x_box_train, y_box_train_log)
    importances = forest_box.feature_importances_
    feat_importances = pd.Series(importances, index=x_box_train.columns)
    plt.figure(figsize=(16, 10))
    feat_importances.plot(kind='barh', color='#488D83')
    plt.title("Feature Importances for Box Office")
    plt.xlabel("Importance")
    plt.ylabel("Column")
    plt.show()

    forest_box2 = RandomForestClassifier(n_estimators=50)
    forest_box2.fit(x_rating_train, y_rating_train)
    importances2 = forest_box2.feature_importances_
    feat_importances2 = pd.Series(importances2, index=x_rating_train.columns)
    plt.figure(figsize=(16, 10))
    feat_importances2.plot(kind='barh', color='#448688')
    plt.title("Feature Importances for Rating")
    plt.xlabel("Importance")
    plt.ylabel("Column")
    plt.show()
```

Рисунок 4.36 – Функція про графіків важливостей ознак

На рисунку 4.37 представлено функцію для прогнозуваввння рейтингу за допомогою класифікатора випадкового лісу. Спочатку виконується функція prepare_data(), яку представлено на рисунку 4.38. В ній набір даних підготавлюється перед створенням кожної з моделей, проводиться нормалізація, логарифмування колонок голосів та тривалості. Після підготовки набору, на екран виводяться результати класифікації загалом для всіх класів і для кожного класу окремо.

```
def predict_rating_class_ml(df):        #rating_class, classification, ml  1 usage
    print(f"-------\nClassification: Rating Class(Random Forest)...\n-------")

    X_train, X_test, y_train, y_test = prepare_data(df, target_col='Rating_Class', purpose='classification', leakage_cols=['IMDB_rating'])
    clf = RandomForestClassifier(n_estimators=35, random_state=42, class_weight='balanced')
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)

    print("\n---ML classification results:---")
    df_metrics = pd.DataFrame()
    df_metrics['Metrics'] = ['Accuracy', 'Misclassification', 'Precision', 'Recall', 'F1-score']
    df_metrics['Rating ML'] = ['{0:.4f}'.format(accuracy_score(y_test, y_pred)),
                               '{0:.4f}'.format(zero_one_loss(y_test, y_pred)),
                               '{0:.4f}'.format(precision_score(y_test, y_pred, average='weighted', zero_division=1)),
                               '{0:.4f}'.format(recall_score(y_test, y_pred, average='weighted', zero_division=1)),
                               '{0:.4f}'.format(f1_score(y_test, y_pred, average='weighted'))]
    print(f'{df_metrics}\n')
    print(classification_report(y_test, y_pred))
```

Рисунок 4.37 – Функція прогнозування рейтингу випадковим лісом

```
def prepare_data(df, target_col, purpose, leakage_cols=None):  4 usages
    drop_list = [target_col]
    if leakage_cols:
        drop_list.extend(leakage_cols)

    if target_col == 'Box_Office':
        df['IMDB_Votes'] = np.log1p(df['IMDB_Votes'])
        df['Runtime'] = np.log1p(df['Runtime'])

    X = df.drop(columns=drop_list)
    y = df[target_col]

    if purpose == 'classification':
        X_train, X_test, y_train, y_test = train_test_split( *arrays: X, y, stratify=y, test_size=0.2, random_state=42, shuffle=True)
        X_train.drop(columns=['Western', 'War', 'Musical', 'Film-Noir'], inplace=True)
        X_test.drop(columns=['Western', 'War', 'Musical', 'Film-Noir'], inplace=True)
        y_train.drop(columns=['Western', 'War', 'Musical', 'Film-Noir'], inplace=True)
        y_test.drop(columns=['Western', 'War', 'Musical', 'Film-Noir'], inplace=True)
    else:
        X_train, X_test, y_train, y_test = train_test_split( *arrays: X, y, test_size=0.2, random_state=42, shuffle=True)

    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    print(f"Train dataset shape: {X_train_scaled.shape}")
    print(f"Test dataset shape: {X_test_scaled.shape}")
    return X_train_scaled, X_test_scaled, y_train, y_test
```

Рисунок 4.38 – Функція prepare_data()

На рисунку 4.39 представлено функцію для класифікації рейтингу за допомогою нейронної мережі. Спочатку викликається функція prepare_data(), яку було представлено вище. Потім відбувається ініціалізація моделі, підбір кількості шарів, нейронів, функції помилок, оптимізатора. Як і у випадку з випадковим лісом, на екран виводяться результати у вигляді таблиці загальні по всім класам і по кожному класу окремо. Також виводиться графік матриці помилок – користувач може наглядно побачити скільки значень модель вгадала правильно, а де помилилась.

```python
def predict_rating_class_nn(df):           #rating_class, classification, nn  1 usage
    print(f"-------\nClassification: Rating Class(Neural Network)...\n-------")
    X_train, X_test, y_train, y_test = prepare_data(df, target_col='Rating_Class', purpose='classification', leakage_cols=['IMDB_rating'])

    num_classes = df['Rating_Class'].nunique()
    model = Sequential([
        Input(shape=(X_train.shape[1],)),
        Dense(units=128, activation='relu'),
        BatchNormalization(),
        Dropout(0.4),
        Dense(units=64, activation='relu'),
        BatchNormalization(),
        Dropout(0.3),
        Dense(units=32, activation='relu'),
        Dense(num_classes, activation='softmax')
    ])
    opt = tf.keras.optimizers.Adam(learning_rate=0.001)
    model.compile(optimizer=opt, loss='sparse_categorical_crossentropy', metrics=['accuracy'])

    history = model.fit(X_train, y_train, epochs=25, batch_size=32, validation_split=0.2, verbose=1)

    y_pred_probs = model.predict(X_test)
    y_pred = np.argmax(y_pred_probs, axis=1)

    print("\nMetrics report:")
    print(classification_report(y_test, y_pred, zero_division=0))

    cm = confusion_matrix(y_test, y_pred)

    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
    plt.xlabel('Predicted class')
    plt.ylabel('Actual class')
    plt.title('Classification: Confusion Matrix for Rating Classes')
    plt.show()

    loss, acc = model.evaluate(X_test, y_test, verbose=0)
    print("\n---NN classification results:---")
    print(f"Accuracy: {acc:.4f}")
```

Рисунок 4.39 – Функція прогнозування рейтингу нейронною мережею

На рисунку 4.40 представлено функцію для прогнозування доходів фільму за допомогою регресора випадкового лісу. Як і у випадку зі всіма функціями, на екран виводяться результати.

```python
def predict_box_office_ml(df):             #box_office, regression, ml  1 usage
    print(f"-------\nRegression: Box Office(Random Forest)...\n-------")

    X_train, X_test, y_train, y_test = prepare_data(df, target_col='Box_Office', purpose='regression')

    regr = RandomForestRegressor(n_estimators=50, random_state=42)
    regr.fit(X_train, y_train)
    y_pred_log = regr.predict(X_test)

    print("\n---ML Regression Results:---")
    print(f"MAE (Mean Average Error in $): {mean_absolute_error(y_test, y_pred_log):,.0f}")
    print(f"R2 Score: {r2_score(y_test, y_pred_log):.4f}")
```

Рисунок 4.40 – Функція прогнозування доходів фільму випадковим лісом

На рисунку 4.41 представлено функцію прогнозування доходу фільму за допомогою нейронної мережі. Всі функції використовують функцію

prepare_data() для своєї роботи. Відбувається ініціаліація моделі, підбір кількості нейронів і шарів, оптимізатора і функції помилок.

```
def predict_box_office_nn(df):            #box_office, regression, nn
    print(f"-------\nRegression: Box Office(Neural Network)...\n-------")

    X_train, X_test, y_train, y_test = prepare_data(df, 'Box_Office', 'regression')

    y_train = np.log1p(y_train)
    y_test = np.log1p(y_test)

    model = Sequential([
        Input(shape=(X_train.shape[1],)),
        Dense(128, activation='relu'),
        BatchNormalization(),
        Dropout(0.1),
        Dense(64, activation='relu'),
        BatchNormalization(),
        Dropout(0.1),
        Dense(32, activation='relu'),
        BatchNormalization(),
        Dense(16, activation='relu'),
        Dense(1, activation='linear')
    ])

    opt = tf.keras.optimizers.Adam(learning_rate=0.001)
    model.compile(optimizer=opt, loss='mean_absolute_error')
    history = model.fit(X_train, y_train, epochs=100, batch_size=32, validation_split=0.2, verbose=1)

    y_pred_log = model.predict(X_test).flatten()

    y_test_real = np.expm1(y_test)
    y_pred_real = np.expm1(y_pred_log)

    print("\n---NN Regression Results---")
    print(f"MAE (Mean Average Error in $): {mean_absolute_error(y_test_real, y_pred_real):,.0f}")
    r2 = r2_score(y_test_real, y_pred_real)
    print(f"R2 Score: {r2:.4f}")
```

Рисунок 4.41 – Функція прогнозування доходу фільмів нейронною мережею

4.2 Висновки за розділом

У розділі було описано роботу модулів і функцій програми. Програма складається з семи модулів. Детально розібрано кожен з модулів, які функції викликаються і як вони одна з одною взаємодіють.

5 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

П’ятий розділ буде присвячено експлуатації, тестуванню та експериментальному дослідженню програми.

5.1 Вимоги для експлуатації

Функції системи описано у ТЗ п. 1.10.4. Вимоги до технічних засобів наведено у ТЗ п. 1.10.7.

Програму не вийде запустити одразу, якщо на комп'ютері не встановлено необхідне забезпечення.

Спочатку треба переконатись, що операційною системою є Windows 10 або вище та присутнє стабільне підключення до мережі Інтернет. Так як дані будуть завантажуватись за допомогою API запитів, то нестабільне підключення може призвести до помилок у отриманні даних, а таким чином і у всій подальшій роботі програми.

Також потрібно мати вільний дисковий простір мінімум 2-3 ГБ для встановлення необхідних бібліотек та просто роботи програми без збоїв. Процесор і оперативна пам'ять теж потрібні відповідати мінімальним вимогам – не менше 4 ядер та мінімум 8ГБ. Якщо ж комп'ютер не має таких характеристик, то запуск програми може бути довгим та ресурсозатратним або взагалі неможливим.

Якщо все вищезгадане наявне, то тепер потрібно скачати Python. Для цього потрібно перейти на офіційний сайт https://www.python.org/. Там у категорії «Download» вибрати останню наявну версію Python. Це показано на рисунку 5.1. Після цього буде відкрито сторінку, де розказано про зміни у цій версії, а також представлено можливість завантажити Python. Щоб завантажити Python, натисніть на «Windows installer (64-bit)» у розділі «Files». Це представлено на рисунку 5.2.

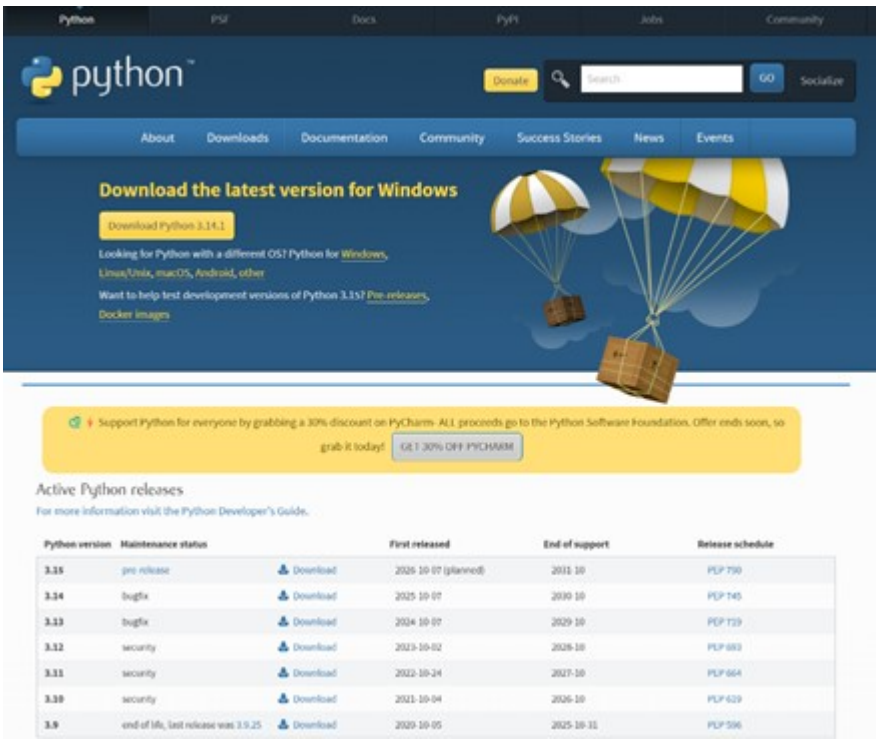


Рисунок 5.1 – Встановлення Python [38]

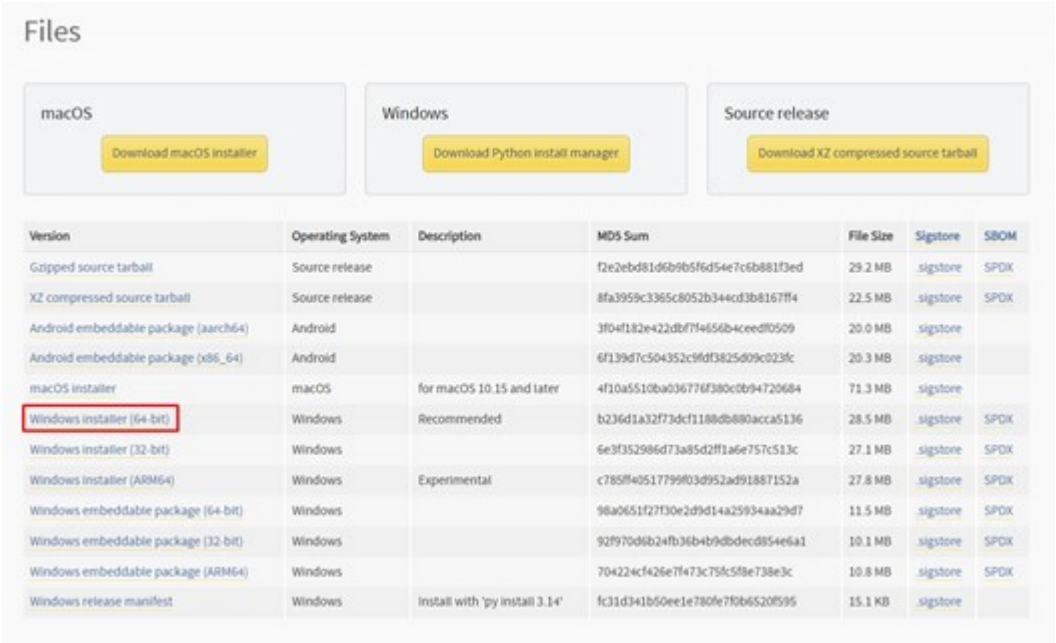


Рисунок 5.2 – Вибір інсталятора [38]

Потім потрібно вибрати місце на комп’ютері, куди потрібно встановити файл, після чого треба його відкрити. На рисунку 5.3 представлено запущений інсталятор. Кнопка «Unstall Now» завантажує Python у директорію за замовчуванням і цей спосіб є рекомендованим, проте якщо все ж хочеться

встановити в іншу директорію, то можна використати кнопку «Customize Installation». Також необхідно поставити дві галочки у чекбоксах знизу. Після цього можна вибирати необхідний спосіб встановлення і чекати завантаження.

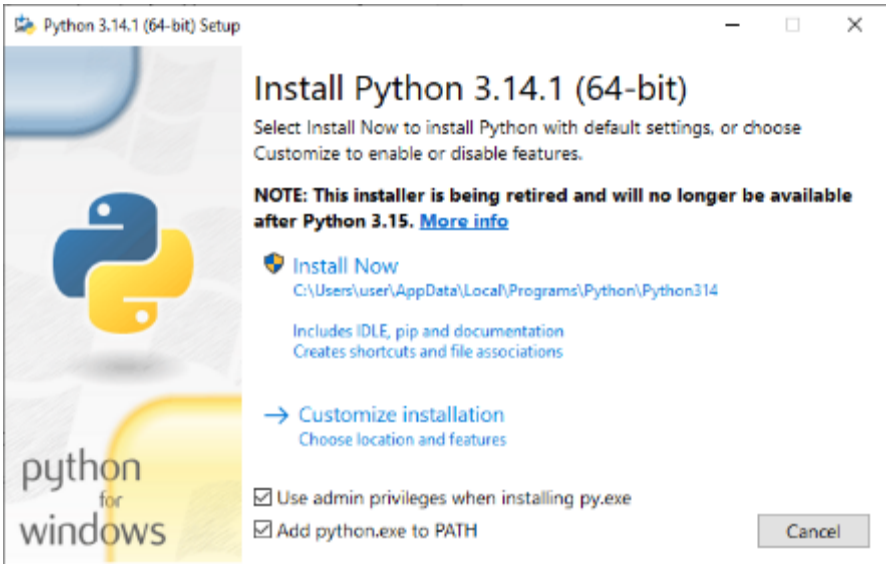


Рисунок 5.3 – Завантаження Python

Після успішного встановлення треба перевірити чи дійсно все працює правильно. Для цього спочатку треба відкрити командний рядок – для цього натискаємо сполучення клавіш Win+R, після чого вводимо у вікно команду cmd. Це представлено на рисунку 5.4.

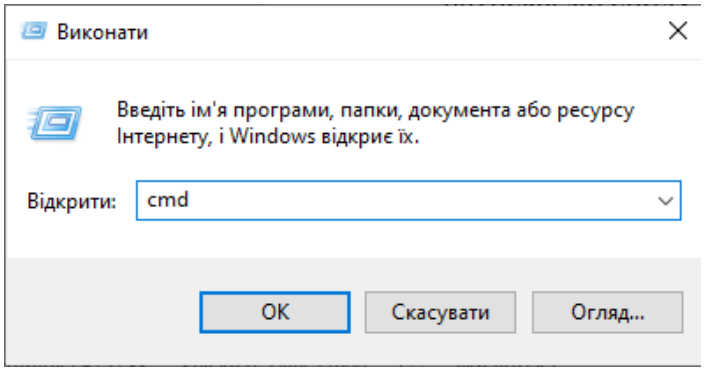


Рисунок 5.4 – Відкриття командного рядка

Після цього можна натискати клавішу «Ок» і буде відкрито командний рядок. Його представлено на рисунку 5.5.

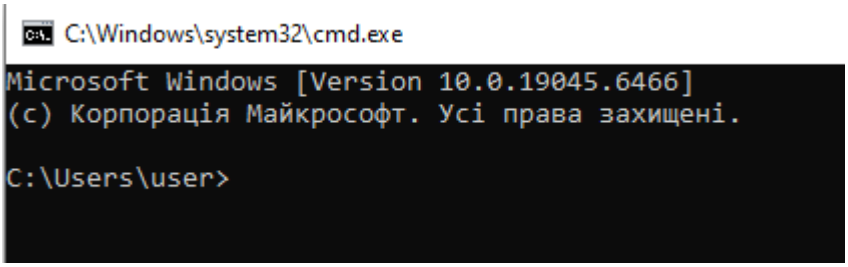


Рисунок 5.5 – Командний рядок

Тепер треба перевірити чи правильно все було встановлено і прописати команду «python». Ця команда спрацює, якщо при встановленні Python було поставлено галочку у чекбоксі «add python.exe to PATH». Якщо це не було зроблено, то додавати python.exe до змінних оточень доведеться самостійно. На рисунку 5.6 представлено результат роботи цієї команди. Як можна побачити, на консоль виведено встановлену версію Python. Якщо у консоль виведено рядок з помилкою, то Python було встановлено неправильно і треба повторити всі кроки.

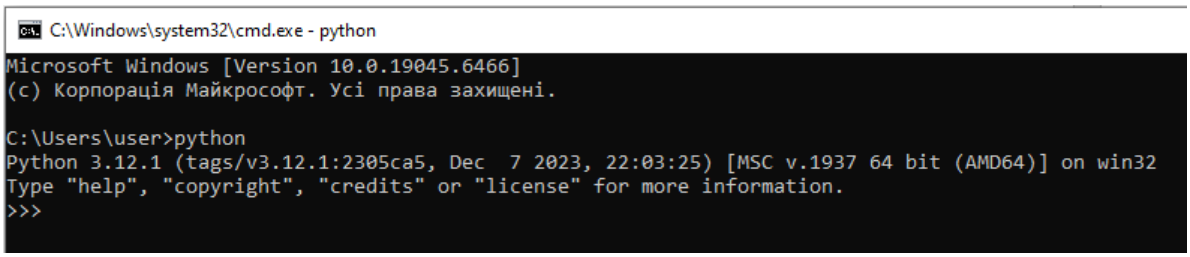


Рисунок 5.6 – Перевірка версії Python

Зверніть увагу – якщо буде помилково встановлено версію Python нижчу за 3.4, то у комплект не буде входити менеджер пакетів pip і його потрібно буде встановити самостійно. Проте рекомендується використовувати новіші версії Python. Для перевірки встановлення pip треба ввести у командний рядок команду pip. Результат роботи команди представлено на рисунку 5.7.

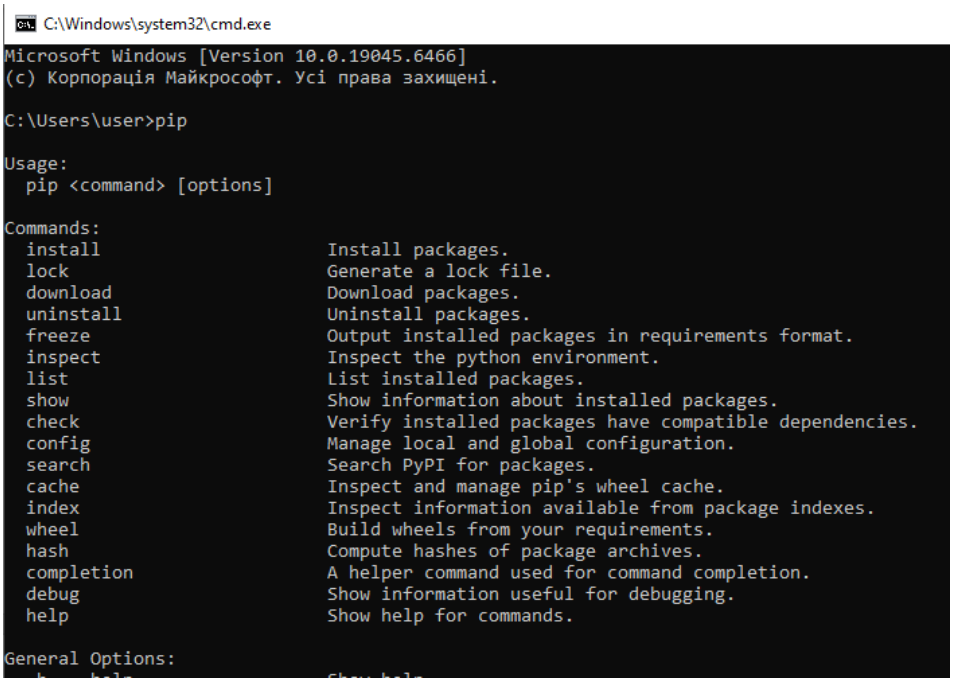


Рисунок 5.7 – Перевірка встановлення pip

Після успішного встановлення Python і pip потрібно встановити необхідні для роботи програми бібліотеки, інакше програма буде видавати помилки і не буде працювати. Для цього у папці проєкту (див. рис. 5.8) є файл requirements.txt. Його представлено на рисунку 5.9.

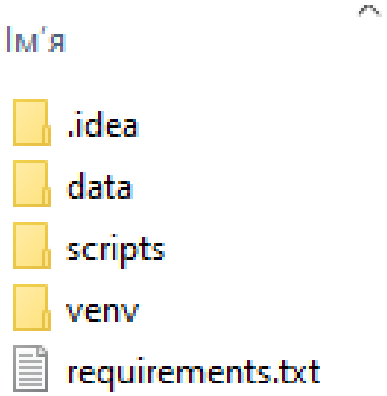


Рисунок 5.8 – Папка проєкту

Рисунок 5.9 – Файл requirements.txt

Для встановлення бібліотек за допомогою цього файлу потрібно перейти у командному рядку у папку проєкту (див. рис. 5.10), після чого прописати там команду pip install -r requirements.txt (див. рис. 5.11). Після цього потрібно почекати виконання програми – при успішному виконанні на екран буде виведено повідомлення, як на рисунку 5.12. Необхідні бібліотеки встановлені і можна переходити до наступного кроку.

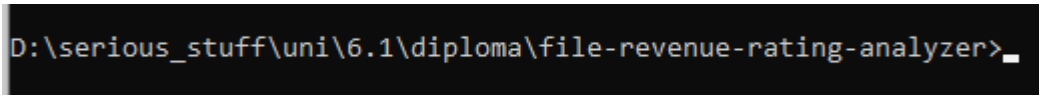


Рисунок 5.10 – Папка проєкту

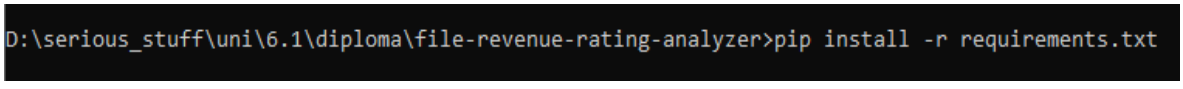


Рисунок 5.11 – Команда для встановлення бібліотек

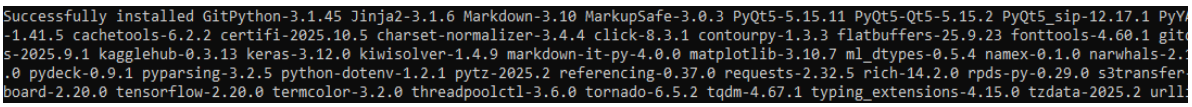


Рисунок 5.12 – Встановлені бібліотеки

Також, для виконання скрипту upload_to_s3.py потрібно зареєструватись у AWS і заповнити файл .env даними свого бакету, куди буде завантажено .csv файл обробленого набору даних. Файл .env зі значеннями за замовченням представлено на рисунку 5.13. Їх потрібно змінити на свої. Для цього заходимо на офіційну сторінку https://aws.amazon.com/s3/getting-started/ і натискаємо «Create account» (див. рис. 5.14). Після цього на екран виведеться сторінка, як на рисунку 5.15 – тут потрібно зареєструватись або увійти у свій кабінет, якщо ви вже зареєстровані.

```
AWS_ACCESS_KEY_ID=Your_access_key
AWS_SECRET_ACCESS_KEY=Your_secret_access_key
AWS_REGION=Your_aws_region
S3_BUCKET_NAME=You_bucket_name
```

Рисунок 5.13 – Файл .env

Рисунок 5.14 – Створення акаунту на AWS [39]

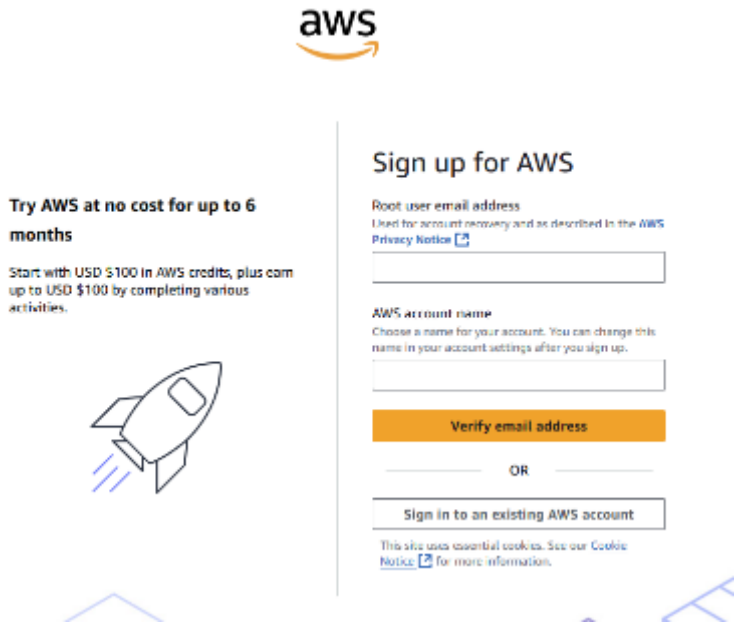


Рисунок 5.15 – Реєстрація на AWS [39]

В тому випадку, якщо ви вже зареєстровані, то потрібно натиснути «Sign in to an existing AWS account» і потім «Sign in using root user email» (див. рис. 5.16).

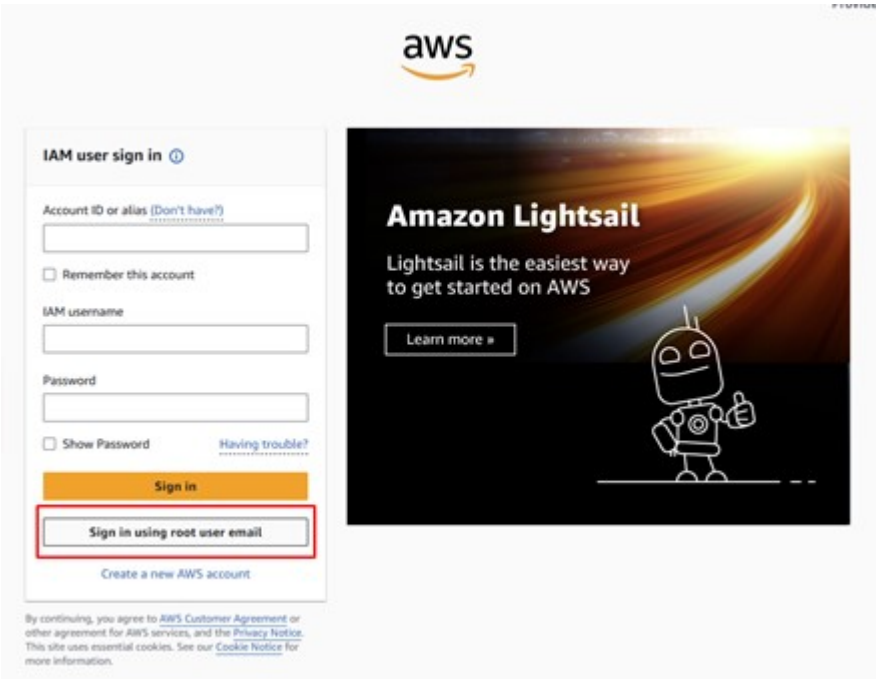


Рисунок 5.16 – Авторизація на AWS [39]

Після чого на екран виведеться сторінка, як на рисунку 5.17, де потрібно ввести ваші дані. Після завершення реєстрації або авторизації на екран виведеться сторінка, як на рисунку 5.18.

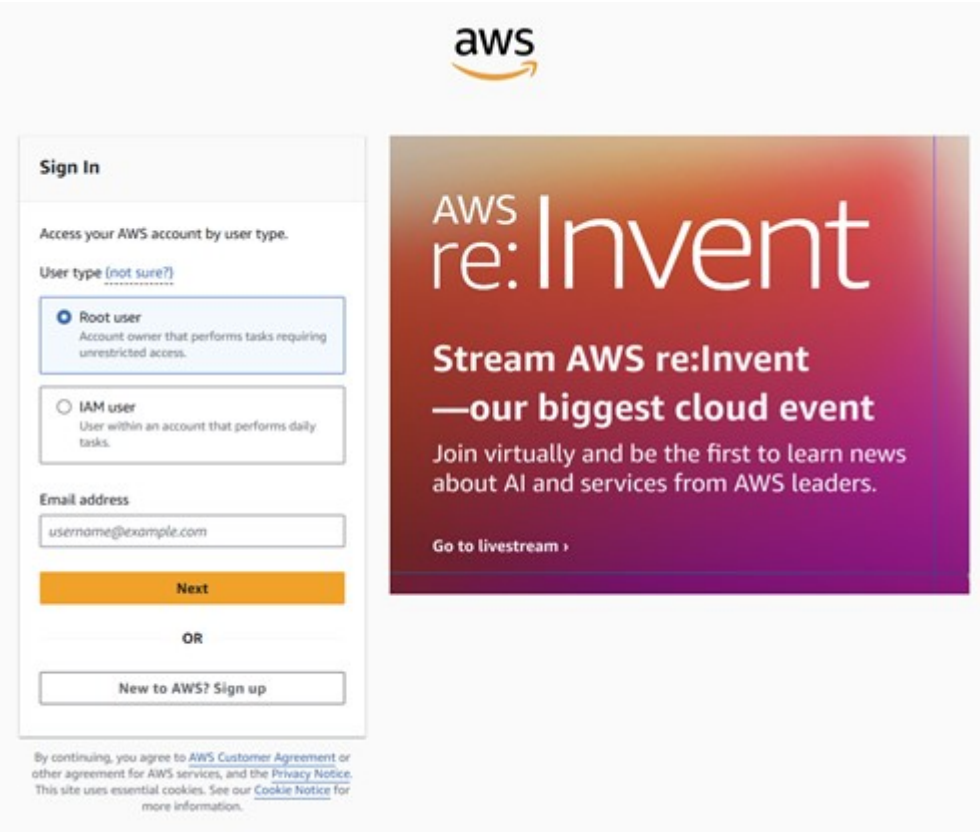


Рисунок 5.17 – Авторизація на AWS [39]

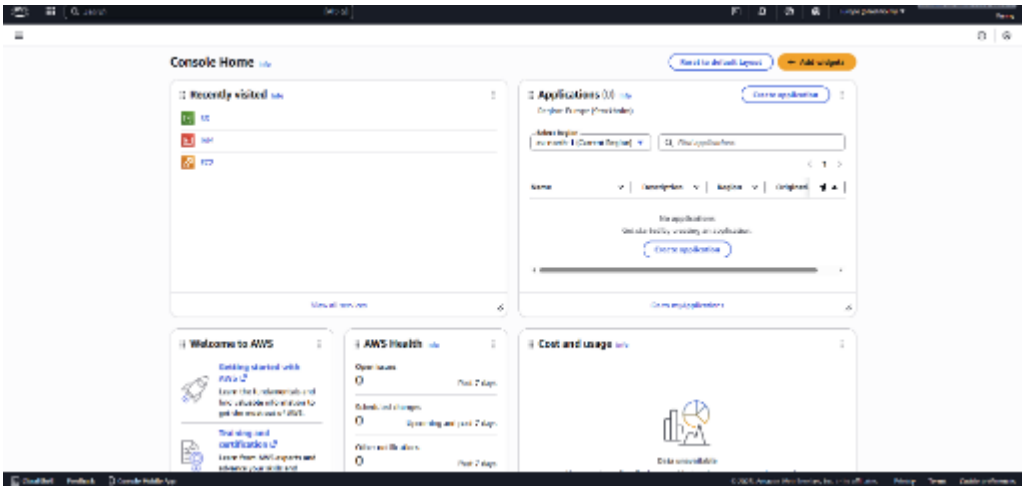


Рисунок 5.18 – Завершення реєстрації на AWS [39]

Тепер треба буде створити IAM роль та бакет для того, щоб в подальшому завантажити туди наш набір даних за допомогою скрипта upload_to_s3.py. Для цього у пошуку треба ввести IAM і вибрати необхідну опцію (див. рис. 5.19). далі в меню зліва треба вибрати «Users» (див. рис. 5.20), а потім «Create user» (див. рис. 5.21). На рисунку 5.21 видно, що у мене вже є користувач, якого використовували для цього проєкту.

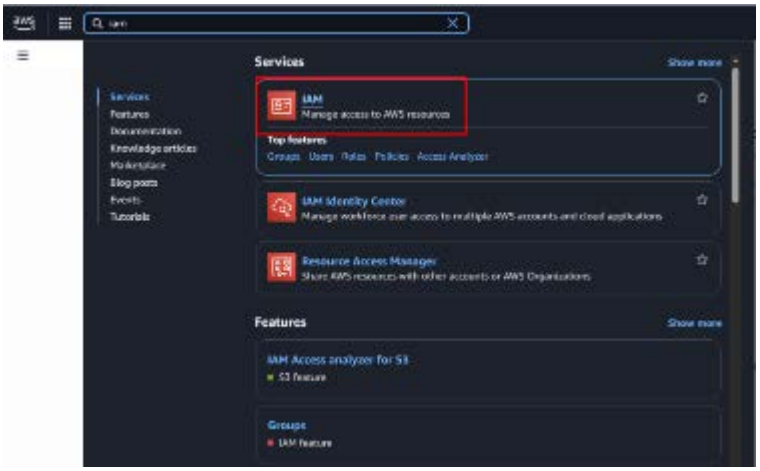


Рисунок 5.19 – IAM роль [39]

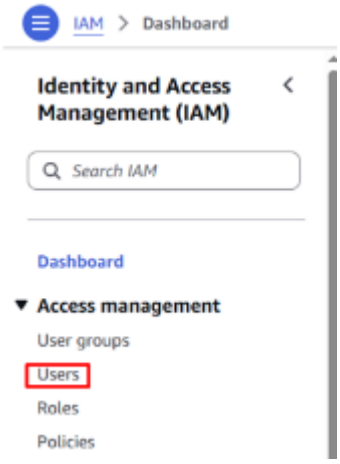


Рисунок 5.20 – Вибір користувача [39]

Рисунок 5.21 – Створення нового користувача [39]

Тепер треба дати новому користувачу ім’я і натиснути «Next» (див. рис. 5.22). Після чого треба, як на рисунку 5.23, вибрати «Attach policies directly» і обрати «AmazonS3FullAccess» та натиснути «Next». У кінці треба натиснути «Create User» (див. рис. 5.24) і нового користувача буде створено.

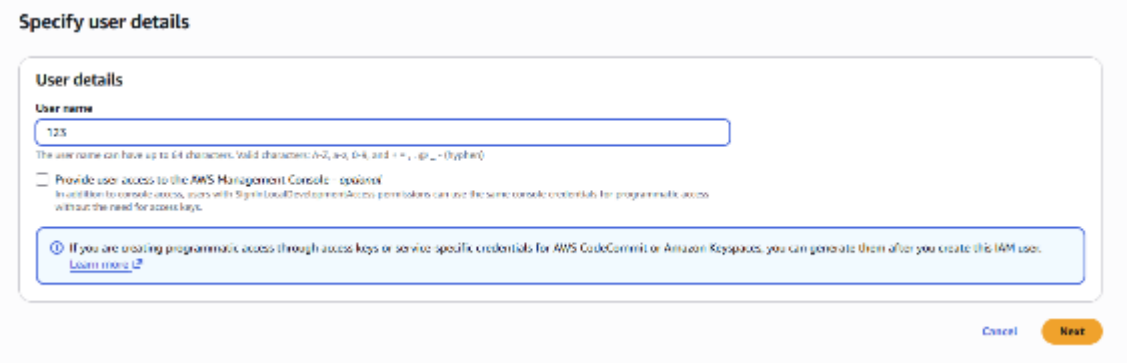


Рисунок 5.22 – Ім’я нового користувача [39]

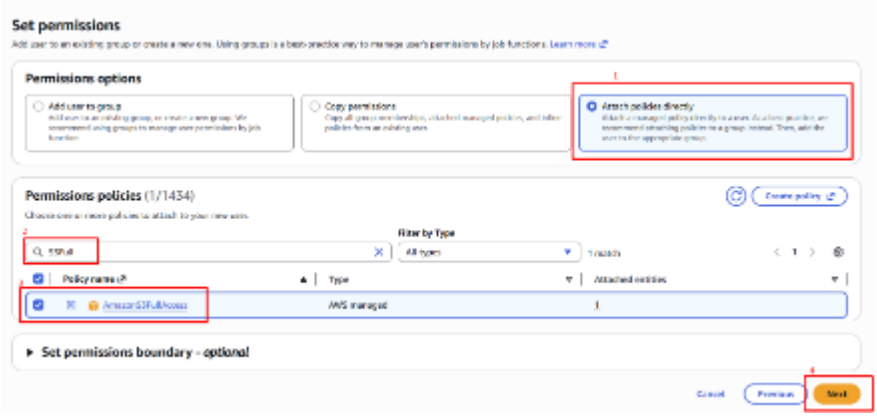


Рисунок 5.23 – Видання доступу [39]

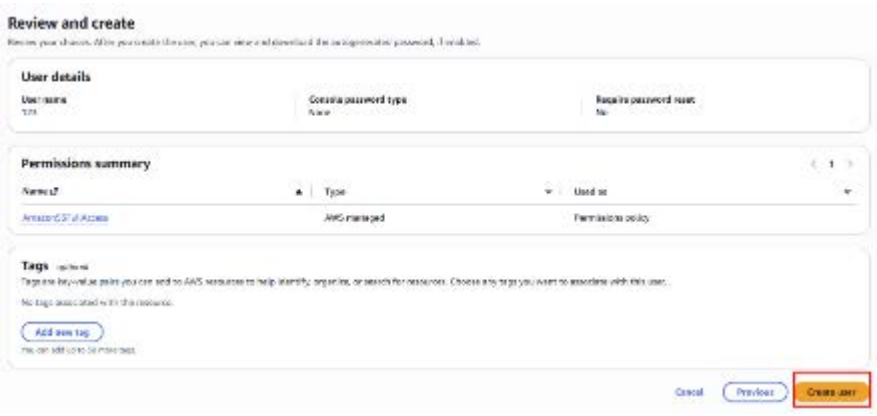


Рисунок 5.24 – Підтвердження створення нового користувача [39]

Після створення користувача потрібно отримати списки ключів. Для цього у списку ваших користувачів виберіть тільки що створеного, оберіть «Security credentials» і потім «Create access key» (див. рис. 5.25).

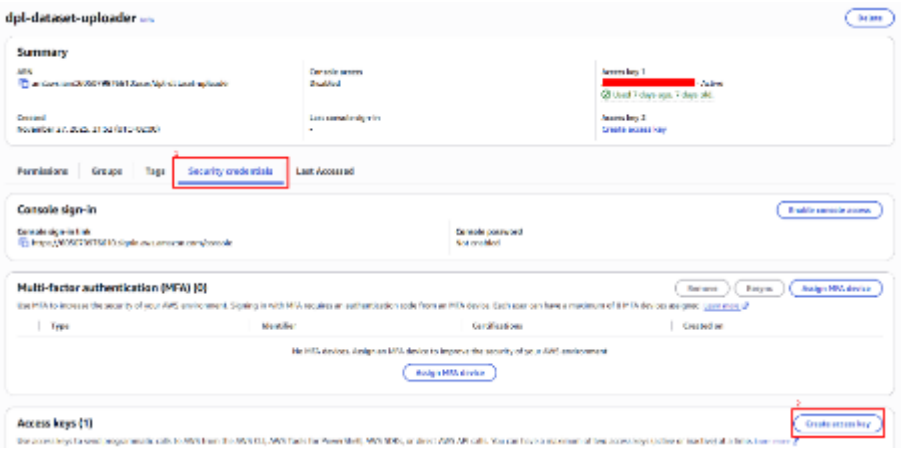


Рисунок 5.25 – Створення ключів доступу [39]

Тепер необхідно вибрати «Local code», поставити галочку у чекбоксі знизу «I understand...» і натиснути «Next» (див. рис. 5.26). Після цього буде відкрито сторінку створення тегу опису, це можна пропустити, натиснувши «Create access key». Після чого потрібно запам’ятати або завантажити собі це на пристрій, натиснувши «Download .csv file» (див. рис. 5.27).

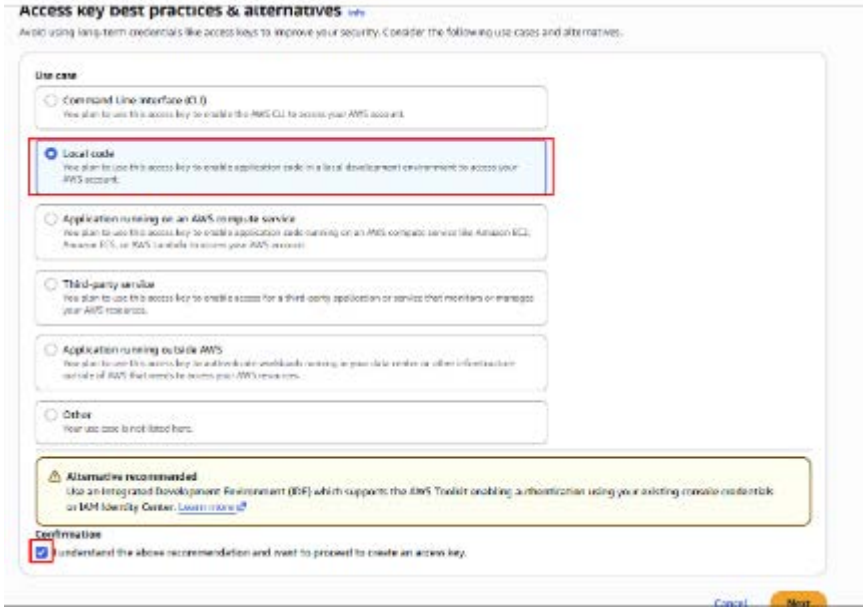


Рисунок 5.26 – Створення ключів доступу [39]

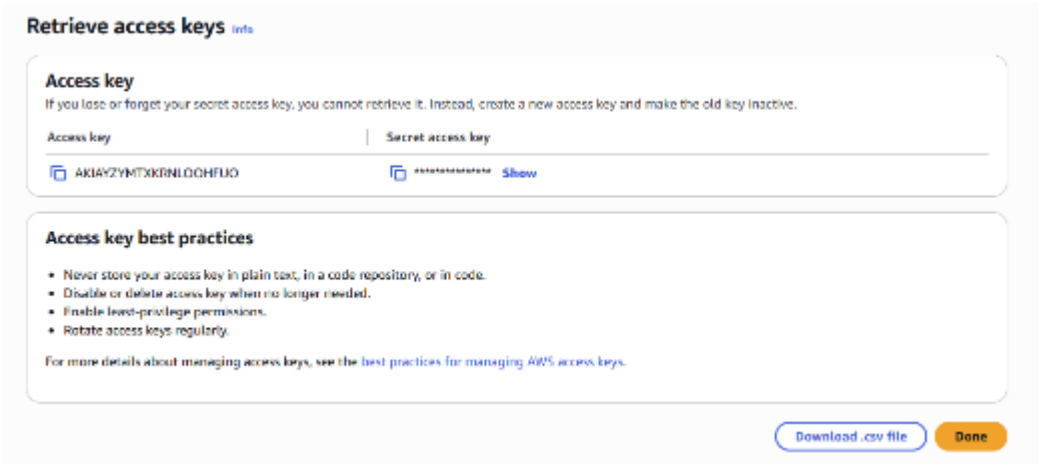


Рисунок 5.27 – Отримання ключів доступу [39]

Після цього нам залишилось створити бакет, який буде тримати у собі завантажений набір даних. Для цього у пошуку треба ввести «S3» і обрати необхідну опцію (див. рис. 5.28).

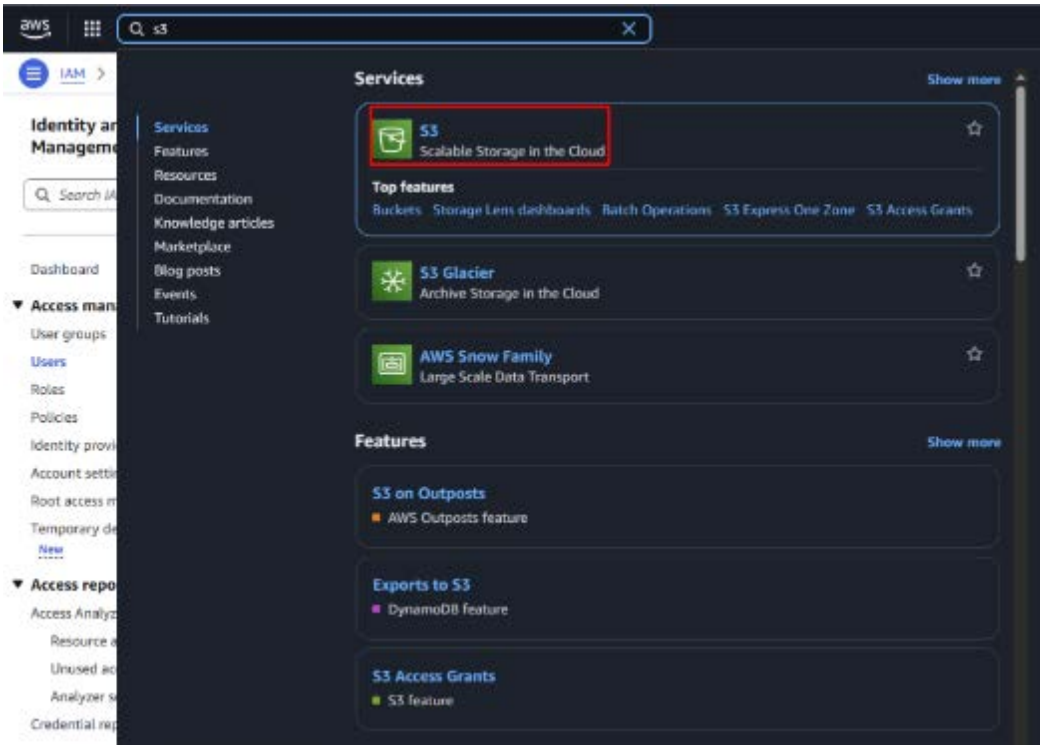


Рисунок 5.28 – Створення бакету [39]

Після цього потрібно натиснути «Create bucket» для створення бакету (див. рис. 5.29).

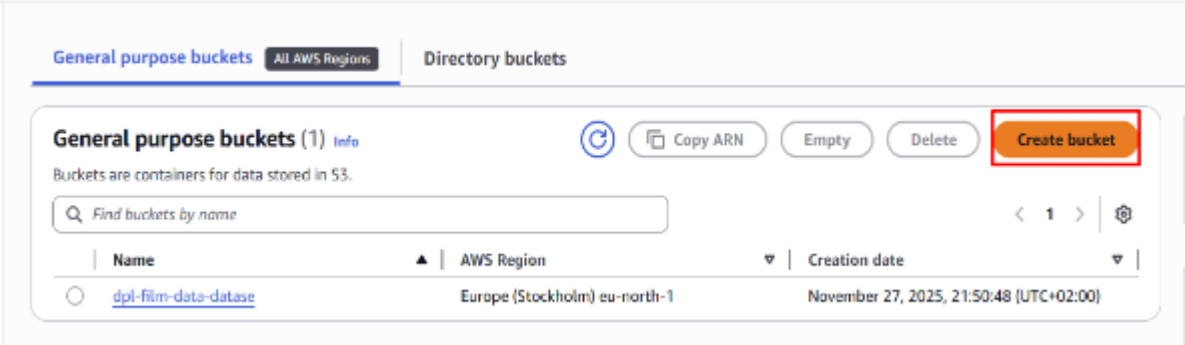


Рисунок 5.29 – Створення бакету [39]

Після цього треба записати у файл .env свої ключі, назву бакеті та регіон і зберегти зміни.

Тепер все готово для роботи і можна переходити до експлуатації програми.

5.2 Експлуатація програми

Після успішного встановлення всього з пункту 5.1 можна переходити до запуску програми. Першим необхідно запустити файл fetch_data.py – він відповідає за завантаження наборів даних. Але навіть якщо ви запустите інші файли до завантаження даних, то вони виведуть на екран відповідні повідомлення. Наприклад, файл preprocessing.py видає таке повідомлення при спробі запуску без попередньо встановлених наборів даних (див. рис. 5.30).

```
D:\serious_stuff\uni\6.1\diploma\file-revenue-rating-analyzer\scripts>python preprocessing.py
-------
Starting preprocessing script...
-------
Error: Necessary directory cannot be found, please run fetch_data.py script
```

Рисунок 5.30 – Повідомлення про помилку

Для запуску програми треба у командному рядку перейти у папку скриптів проєкту і виконати команду «python “назва необхідного файлу”», приклад запуску fetch_data.py представлено на рисунку 5.31.

```
D:\serious_stuff\uni\6.1\diploma\file-revenue-rating-analyzer\scripts>python fetch_data.py
-------
Starting fetching data script...
-------
```

Рисунок 5.31 – Запуск fetch_data.py

Встановлення необхідних наборів даних займе деякий час, тож треба буде почекати. На екран буде виведено необхідні повідомлення, які інформують користувача щодо статусу завантаження. Після встановлення наборів даних на екран буде виведено повідомлення про успішне завантаження даних і час, витрачений на роботу цього скрипту (див. рис. 5.32). Якщо ж ви запустите цей файл скрипт повторно, то програма виведе на екран повідомлення, що необхідні набори даних вже встановлені (див. рис. 5.33).

```
-------
Fetching script duration: 0.049 seconds
-------
```

Рисунок 5.32 – Завершення скрипту fetch_data.py

```
-------
Starting fetching data script...
-------
-------
Downloading is about to start...
-------
Main dataset is already on your device..

-------
Attempting to download additional necessary datasets...
-------
Dataset1 is already on your device..

Dataset2 is already on your device..

Dataset3 is already on your device..

Dataset4 is already on your device..

-------
Fetching script duration: 0.049 seconds
-------
```

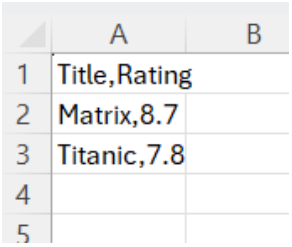
Рисунок 5.33 – Повторний запуск fetch_data.py

Другим файлом необхідно запускати скрипт preprocessing.py для попередньої обробки набору даних, який під час свого виконання запустить скрипт to_processed.py для збереження обробленого набору, який було отримано після об’єднання завантажених наборів, у папку data/processed. Ці папки створювати не потрібно, програма створить їх сама. Якщо файл to_processed.py буде запущено не з файлу preprocessing,py, а вручну, то він просто збереже набір за замовчуванням у цю папку, тому для правильної роботи подальших скриптів потрібно, щоб to_processed.py було викликано саме з preprocessing.py. Там також передбачено функцію повторного збереження цього файлу, тож навіть якщо ви помилково запустили скрипт to_processed.py і він зберіг у data/processed набір даних за замовчуванням з назвою processed_data.csv, то у вас буде можливість просто перезаписати його на правильний набір, якщо файл to_processed буде викликано з preprocessing.py. Результат роботи запуску to_processed.py вручну представлено на рисунках 5.34-5.35. Програма запропонує перезаписати файл, якщо він існує. Для цього потрібно ввести необхідну відповідь у консоль, якщо відповідь не буде відповідати потрібному формату, то програма буде пропонувати ввести відповідь ще раз. Після збереження набору на екран буде виведено повідомлення про успішне завантаження даних і час, витрачений на роботу цього скрипту.

Тепер розглянемо правильний запуск файлу processing.py, який потім у собі запускає to_processed.py. На рисунку 5.36 представлено запуск скрипту processing.py. Спочатку на екран виводиться стартове повідомлення, приклад основного набору даних, далі виводяться кількість пропущених значень по колонці. З цього можна побачити, що пропущені значення в основному наборі є тільки в колонках рейтингу MetaScore, дохідності фільму та сертифікації фільму.

```
D:\serious_stuff\uni\6.1\diploma\file-revenue-rating-analyzer\scripts>python to_processed.py
-------
Starting saving processed data script...
-------
The processed dataset is already on your device..
Do you want to save it? Y/n
Enter your answer:abc
You should only choose between Y and n. Please try again..

Do you want to save it? Y/n
Enter your answer:Y
Preparing to save processed dataset..
Saved successfully!
-------
Saving processed data script duration: 8.216 seconds
-------
```

Рисунок 5.34 – Результат запуску to_processed.py вручну



	A	B
1	Title,Rating	
2	Matrix,8.7	
3	Titanic,7.8	
4		
5		

Рисунок 5.35 – Результат запуску to_processed.py вручну

```
D:\serious_stuff\uni\6.1\diploma\file-revenue-rating-analyzer\scripts>python preprocessing.py
-------
Starting preprocessing script...
-------
Necessary directory exists, preparing for preprocessing...
-------
Main Dataset
-------
Necessary dataset exists, preparing for preprocessing...

Dataset example:
                              Title  Year  Runtime  IMDB_rating  IMDB_Votes  MetaScore   Box_Office
        Stars
0            The Shawshank Redemption  1994      142          9.3     2804443       82.0   28340000.0
Bob Gunton'...
1                       The Godfather  1972      175          9.2     1954174      100.0  134970000.0
es Caan', '...
2  Ramayana: The Legend of Prince Rama  1993      135          9.2       12995        NaN          NaN  ['
ie Mirman',...
3                     The Chaos Class  1975       87          9.2       42231        NaN          NaN
it Akçatepe...
4                     The Dark Knight  2008      152          9.0     2786129       84.0  534860000.0
'Aaron Eckh...

Duplicates have been dropped...

All not a number values from dataset:
Title              0
Year               0
Runtime            0
IMDB_rating        0
IMDB_Votes         0
MetaScore       1968
Box_Office      2812
Genre              0
Certification    361
Director           0
Stars              0
dtype: int64
```

Рисунок 5.36 – Запуск preprocessing.py

Після цього на екран виводиться така ж інформація про всі додаткові набори даних – приклад набору, всі пропущені значення. Результати цих виводів представлено на рисунках 5.37-5.40.

```
-------
Dataset1
-------
Necessary dataset exists, preparing for preprocessing...

Dataset1 example:
                          Title Language    rel_date
0                      John Wick       en  2014-10-22
1                       Ad Astra       en  2019-09-17
2              Bad Boys for Life       en  2020-01-15
3                  The Lion King       en  2019-07-12
4  Jurassic World: Fallen Kingdom       en  2018-06-06

All not a number values from dataset1:
Title       0
Language    0
rel_date    7
dtype: int64

The number of necessary for us release date values with NA value: 0
```

Рисунок 5.37 – Перший додатковий набір даних

```
-------
Dataset2
-------
Necessary dataset exists, preparing for preprocessing...
D:\serious_stuff\uni\6.1\diploma\file-revenue-rating-analyzer\scripts\preprocessing.py:99: DtypeWarning:
  df = pd.read_csv(os.path.join(path, "data2.csv"))

Dataset2 example:
  release_date   Box_Office                     Title
0   1995-10-30  373554033.0                 Toy Story
1   1995-12-15  262797249.0                   Jumanji
2   1995-12-22          0.0            Grumpier Old Men
3   1995-12-22   81452156.0           Waiting to Exhale
4   1995-02-10   76578911.0  Father of the Bride Part II

All not a number values from dataset2:
release_date    81
Box_Office       0
Title            0
dtype: int64

The number of necessary for us release date values with NA value: 0
```

Рисунок 5.38 – Другий додатковий набір даних

```
-------
Dataset3
-------
Necessary dataset exists, preparing for preprocessing...

Dataset3 example:
                   Title   Box_Office                                    Country
0  Mission: Impossible II  546388108.0                United States of America
1              Gladiator  460583960.0  United Kingdom, United States of America
2              Cast Away  429632142.0                United States of America
3         What Women Want  374111707.0  United Kingdom, United States of America
4                Dinosaur  349822765.0                United States of America

All not a number values from dataset3:
Title           0
Box_Office      0
Country       200
dtype: int64
```

Рисунок 5.39 – Третій додатковий набір даних

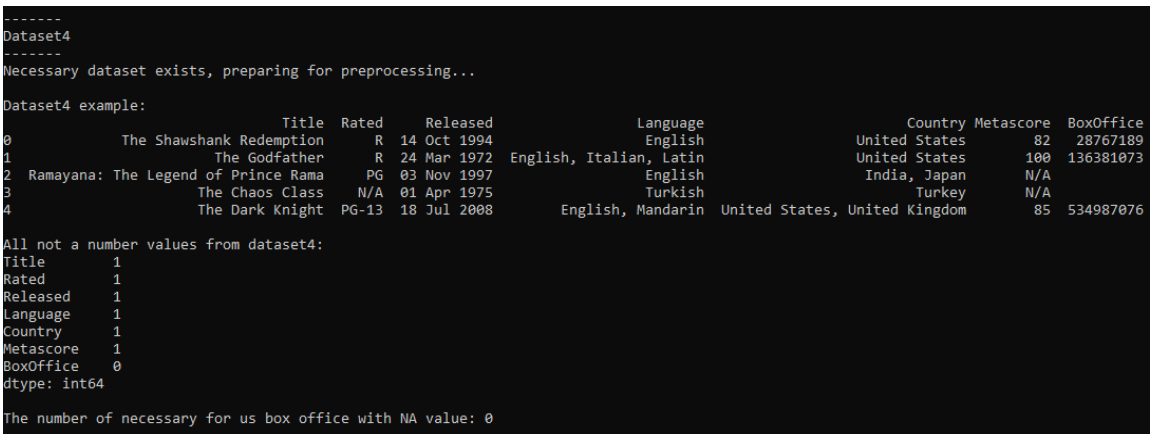


Рисунок 5.40 – Четвертий додатковий набір даних

Після цього на екран виводиться повідомлення порівняння наборів для об’єднання, вивід прикладів з кожного набору, а також кількість екземплярів у кожному з них (див. рис. 5.41).

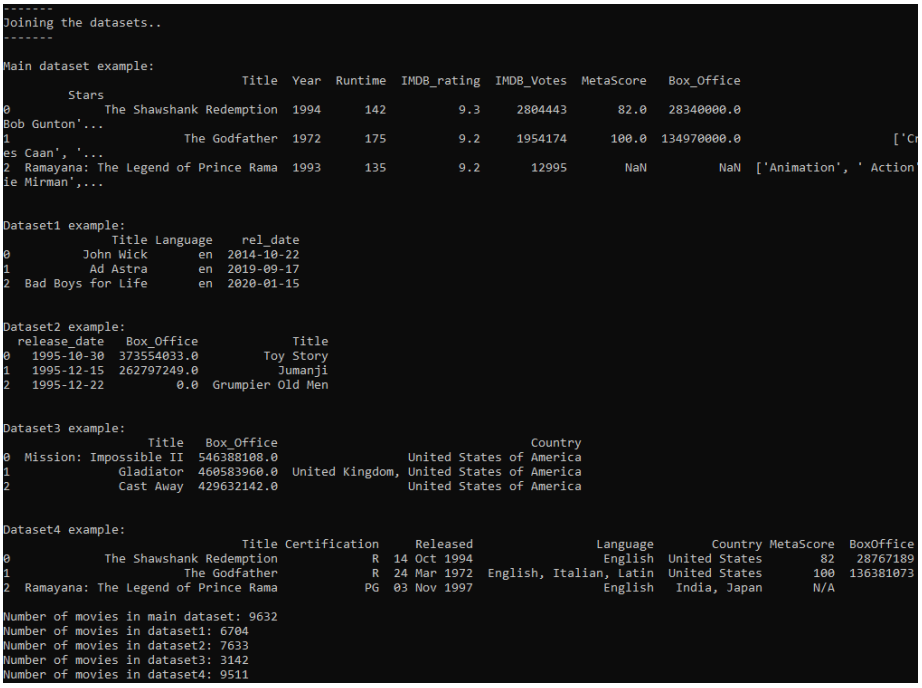


Рисунок 5.41 – Порівняння наборів для об’єднання

Потім на екран виводиться інформація щодо додавання додаткових колонок до основного набору даних – порівняння кількості пропущених значень додаткових колонок з додаткових наборів даних. Потім у додаткових наборах шукають значення по всіх назвах фільмів з основного набору і на

екран виводиться фінальна кількість пропущених значень в такому додатковому наборі. Це представлено на рисунку 5.42.

На рисунку 5.43 представлено таку ж інформацію щодо обов’язкових колонок – це колонки додаткових наборів, у яких шукають значення, щоб заповнити пропущенні значення основного набору. Цими колонками є рейтинг MetaScore, дохідність та сертифікація фільму.

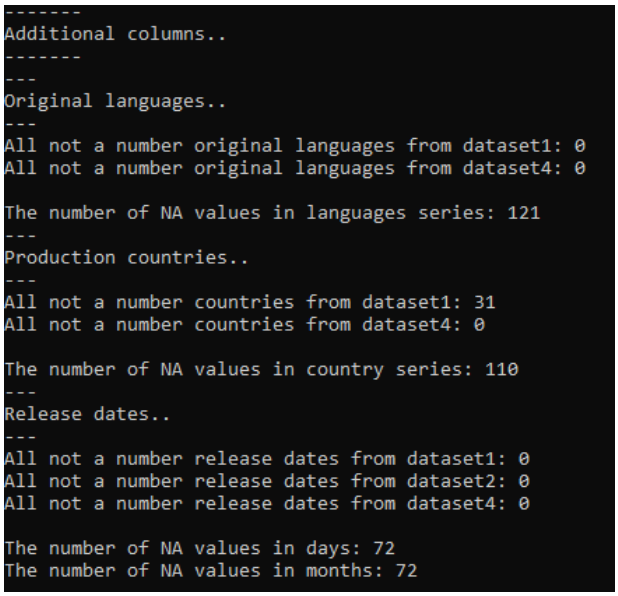


Рисунок 5.42 – Додаткові колонки

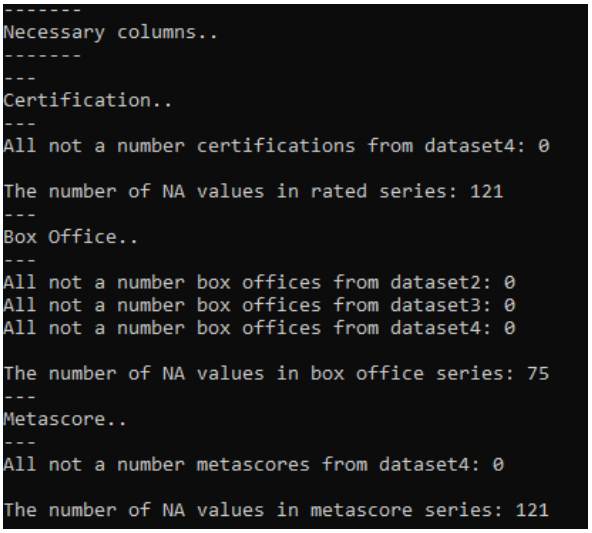


Рисунок 5.43 – Обов’язкові колонки

Потім на екран виводиться повідомлення про конкатенацію наборів даних, приклади їх екземплярів, а також кількість дублікатів у всіх трьох наборах (основний набір, набір додаткових колонок і набір обов'язкових колонок). Це представлено на рисунку 5.44.

```
-------
Concatenating dataframes..
-------
Main df:
                     Title  Year  Runtime  IMDB_rating  IMDB_Votes  MetaScore  Box_Office
0  The Shawshank Redemption  1994      142          9.3     2804443       82.0  28340000.0  ['D

Additional df:
                      Day Month Language        Country
The Shawshank Redemption  23    09  English  United States

Necessary df:
                      Certification  Box_Office MetaScore
The Shawshank Redemption             R  28341469.0        82

Main dataframe title duplicates: 0
Additional dataframe title duplicates: 0
Necessary  dataframe title duplicates: 0
```

Рисунок 5.44 – Приклади екземплярів наборів для конкатенації

Після цього в цій же функції на екран виводиться результат злиття цих трьох наборів, а також порівняння кількості пропущених значень у початковому основному наборі і у цьому, а також різниця у відсотках (див. рис. 5.45).

```
Merged df:
                     Title  Year  Runtime  IMDB_rating  IMDB_Votes  Me
0  The Shawshank Redemption  1994      142          9.3     2804443

NA difference table:
              Main_df  Result_df Diffenence, %
Box_Office     2812.0       43.0         98.47
Certification   361.0       22.0         93.91
Country             -      110.0             -
Day                 -       72.0             -
Director          0.0        0.0             -
Genre             0.0        0.0             -
IMDB_Votes        0.0        0.0             -
IMDB_rating       0.0        0.0             -
Language            -      121.0             -
MetaScore      1968.0       56.0         97.15
Month               -       72.0             -
Runtime           0.0        0.0             -
Stars             0.0        0.0             -
Title             0.0        0.0             -
Year              0.0        0.0             -
```

Рисунок 5.45 – Порівняння кількості пропущених значень при конкатенації

Після цього на екран виводиться бінарні колонки для жанрів, кількість колонок набору з ними і без них, а також відсоток видалених через пропущені значення екземплярів (див. рис. 5.46).

```
Binarized genres dataframe:
   Action  Adventure  Animation  Biography  Comedy  Crime  Drama  F
sical  Mystery  Romance  Sci-Fi  Sport  Thriller  War  Western
0       0          0          0          0       0      0      1
    0        0        0       0      0         0    0        0

The amount of previous columns: 13
The amount of new columns: 34

Dataset with binarized columns:
                     Title  Box_Office Certification        Countr
nguage MetaScore Month  Runtime  Year  Action  Adventure  Animation
ilm-Noir  History  Horror  Music  Musical  Mystery  Romance  Sci-Fi
0  The Shawshank Redemption  28340000.0             R  United State
nglish      82.0    09      142  1994       0          0          0
       0        0       0      0        0        0        0       0

The number of final dataset elements: 9632
The number of final dataset elements after all preprocessing: 7241
The percent of the whole dataset that have been dropped: 24.82%
```

Рисунок 5.46 – Бінарні колонки конкатенованого набору

Потім на екран виводиться кількість класів і кількість екземплярів кожного класу по колонці рейтингу та приклад екземпляру з фінального набору (див. рис. 5.47).

```
Rating class distribution:
Rating_Class
Average    3113
Good       2618
Bad        1510
Name: count, dtype: int64
Final dataset:                     Title  Box_Office Certification        Country  Day
0  The Shawshank Redemption    28340000            R  United States   23  Frank Darabont
```

Рисунок 5.47 – Дистриб’юція класів

У кінці програма запропонує зберегти отриманий набір у папку data/processed. Для цього треба вписати у консоль один з двох варіантів відповіді. Якщо буде введено неправильне значення, то користувач буде повинен вводити значення до того моменту, поки не введе правильне (див. рис. 5.48). Якщо вибрати «Y», то файл .csv отриманого набору даних буде збережено у папку data/processed. Якщо ж вибрати «n», то файл не буде збережено. У кінці програма підрахує кількість секунд, які було витрачено на виконання усього скрипту.

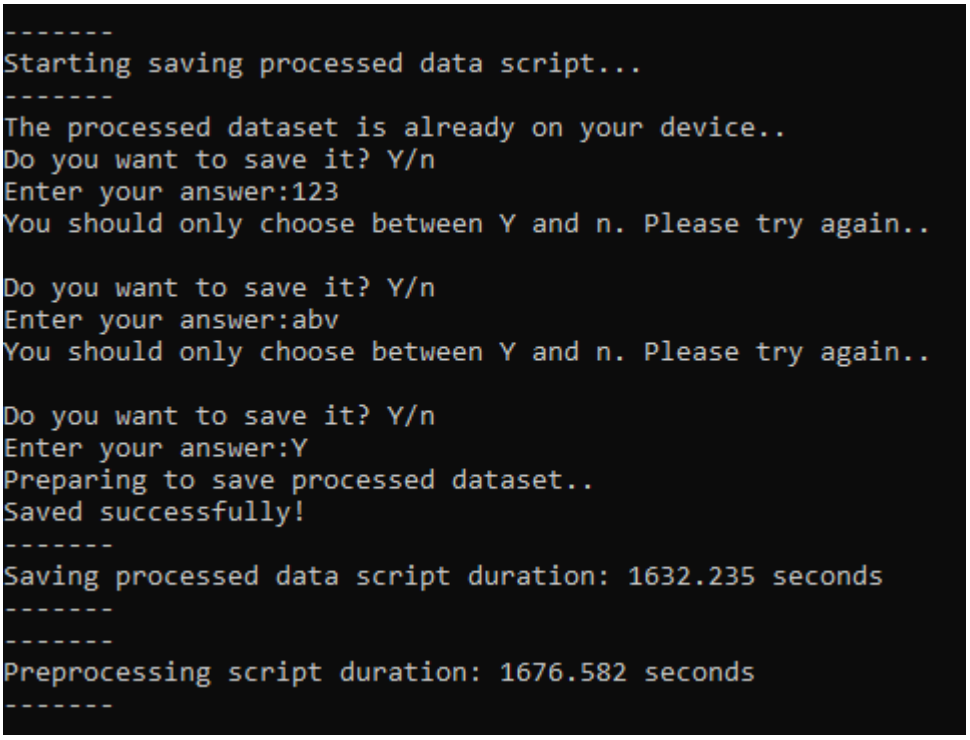


Рисунок 5.48 – Збереження набору

Далі порядок запуску скриптів не є важливим і користувач може запускати їх у якому завгодно порядку. Припустимо, користувач хоче завантажити на AWS S3 збережений у data/processed набір даних. Для цього йому необхідно виконати всі кроки пов’язані з AWS з пункту 5.1.

Перед запуском скрипту потрібно ввести дані свого бакету у файл .env у папці проєкту. Якщо скрипт upload_to_s3.py буде запущено зі значеннями за замовченням, то він виведе на екран відповідне повідомлення (див. рис. 5.49).

```
D:\serious_stuff\uni\6.1\diploma\file-revenue-rating-analyzer\scripts>python upload_to_s3.py
-------
Starting AWS S3 script...
-------
File processed_data.csv exists, preparing to upload...
Error: You haven't configured the aws_region in .env file yet.

AWS S3 script duration: 0.013 seconds
```

Рисунок 5.49 – Повідомлення про помилку

На рисунку 5.50 буде представлено запуск скрипту з попередньо внесеними даними до .env файлу. На рисунку 5.51 представлено результат роботи.

```
D:\serious_stuff\uni\6.1\diploma\file-revenue-rating-analyzer\scripts>python upload_to_s3.py
-------
Starting AWS S3 script...
-------
File processed_data.csv exists, preparing to upload...
Uploading '..\data\processed\processed_data.csv' in the bucket 'dpl-film-data-datase'...
File was successfully uploaded to AWS. File name: processed_data.csv

AWS S3 script duration: 1.304 seconds
```

Рисунок 5.50 – Правильне використання upload_to_s3.py скрипту

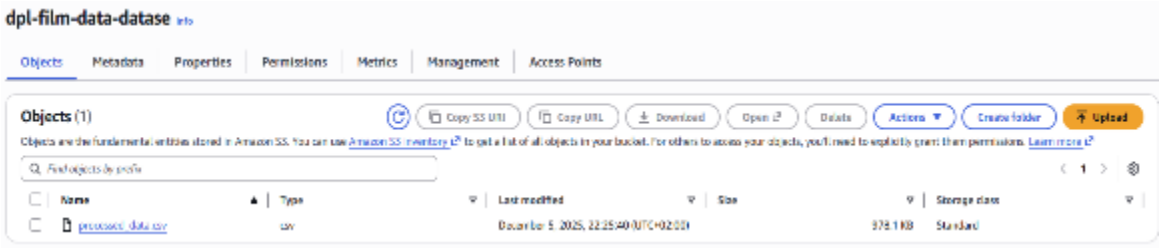


Рисунок 5.51 – Результат роботи скрипту upload_to_s3.py

Далі розберемо приклад запуску скрипту trends.py – він виводить на екран графіки тенденцій індустрії кіно. На рисунку 5.52 представлено все, що буде виведено у консолі у результаті роботи. На рисунках 5.53-5.57 представлено всі графіки, які буде виведено на екран.

```
-------
Starting trends script...
-------
File processed_data.csv exists, preparing to build trends...
D:\serious_stuff\uni\6.1\diploma\file-revenue-rating-analyzer\s
  df_years_by_bin = df_years_sorted.groupby(df_year_bins).sum()
-------
Trends script duration: 5.857 seconds
-------
```

Рисунок 5.52 – Результат роботи trends.py

На рисунку 5.53 представлено два графіки: кількість фільмів по десятиліттях та кількість фільмів по топ 10 країнах. Можна побачити, як кількість фільмів постійно зростає. Важливо наголосити, що цей графік представляє інформацію тільки з цього набору даних, а не з даних про всі фільми, які колись були створені людством.

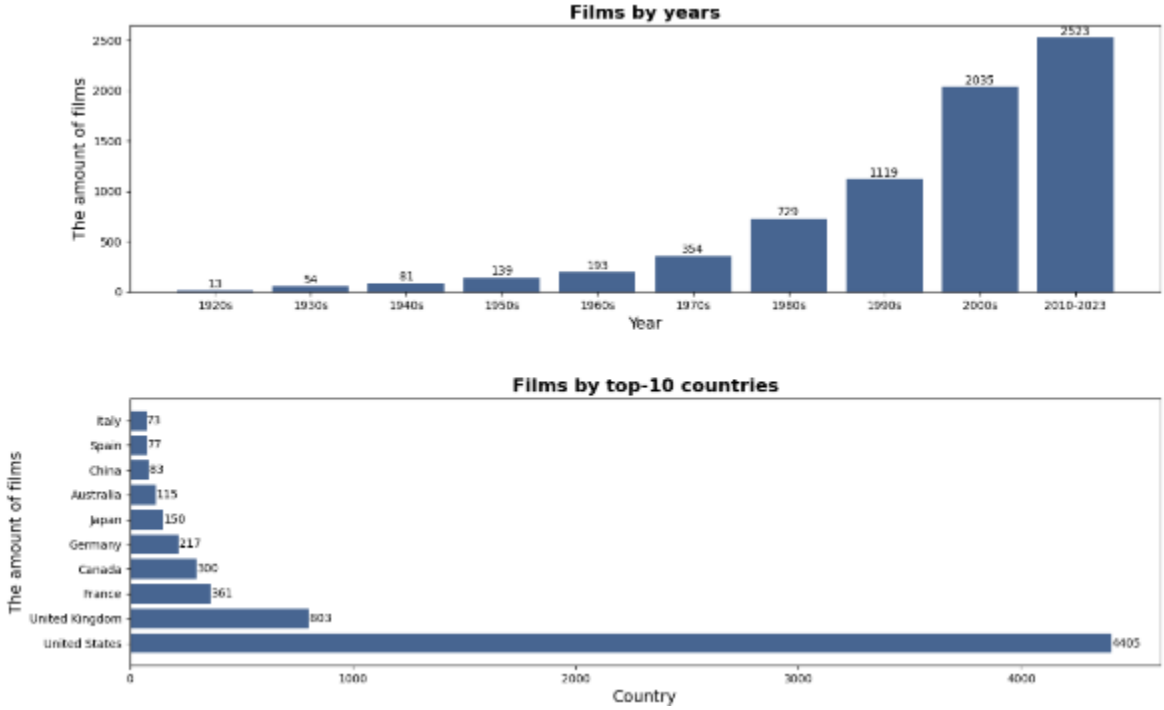


Рисунок 5.53 – Результат роботи trends.py

На рисунку 5.54 представлено два графіки: кількість фільмів по місяцях та кількість фільмів по днях. Вересень – найактивніший місяць з точки зору випусків фільмів, а січень, навпаки, найменш активний. Також можна назвати дати, в які кількість випущених фільмів доволі суттєво відрізняється від інших – 1, 7, 12, 13, 15 та 25.

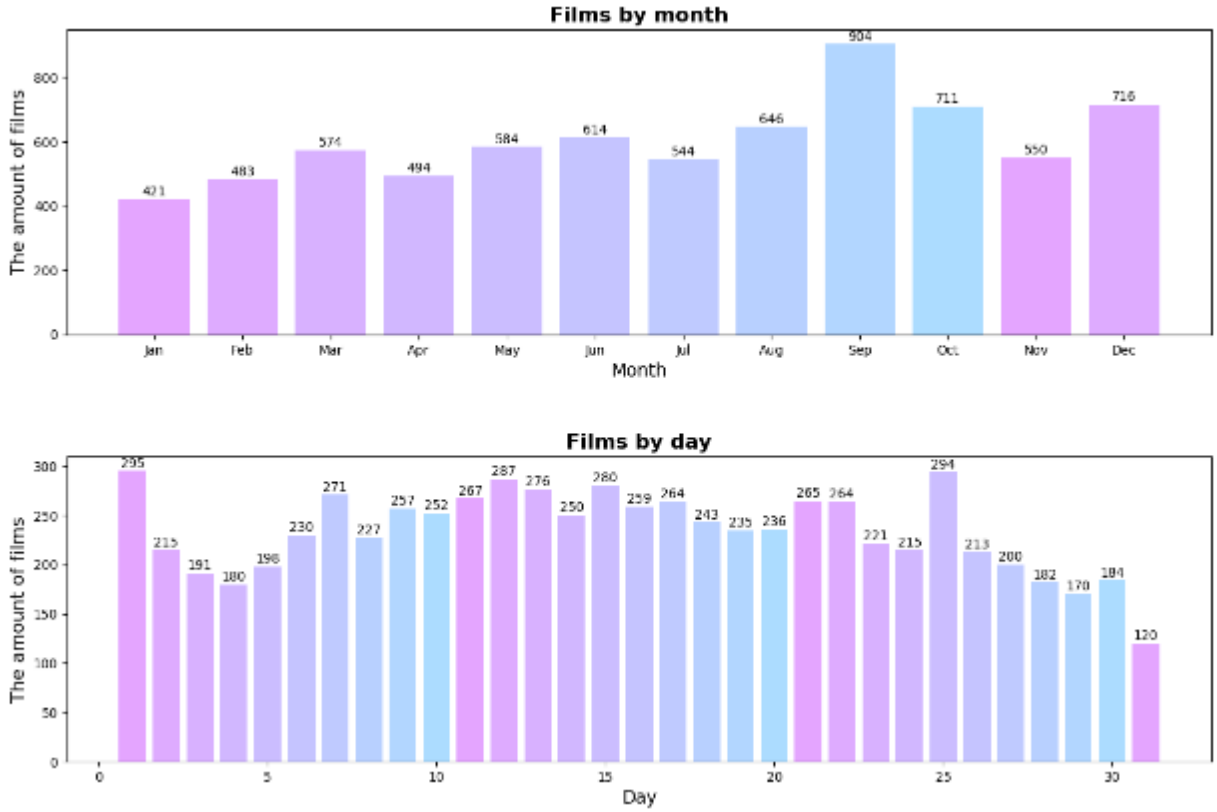


Рисунок 5.54 – Результат роботи trends.py

На рисунку 5.55 представлено два графіки: кількість фільмів по топ 10 режисерах та кількість фільмів по жанрах. Драма і комедія займають лідерські позиції серед інших жанрів.

На рисунку 5.56 представлено два графіки: кількість фільмів по сертифікації та кількість фільмів по тривалості. Можна побачити, що більшість фільмів триває 90-110 хвилин.

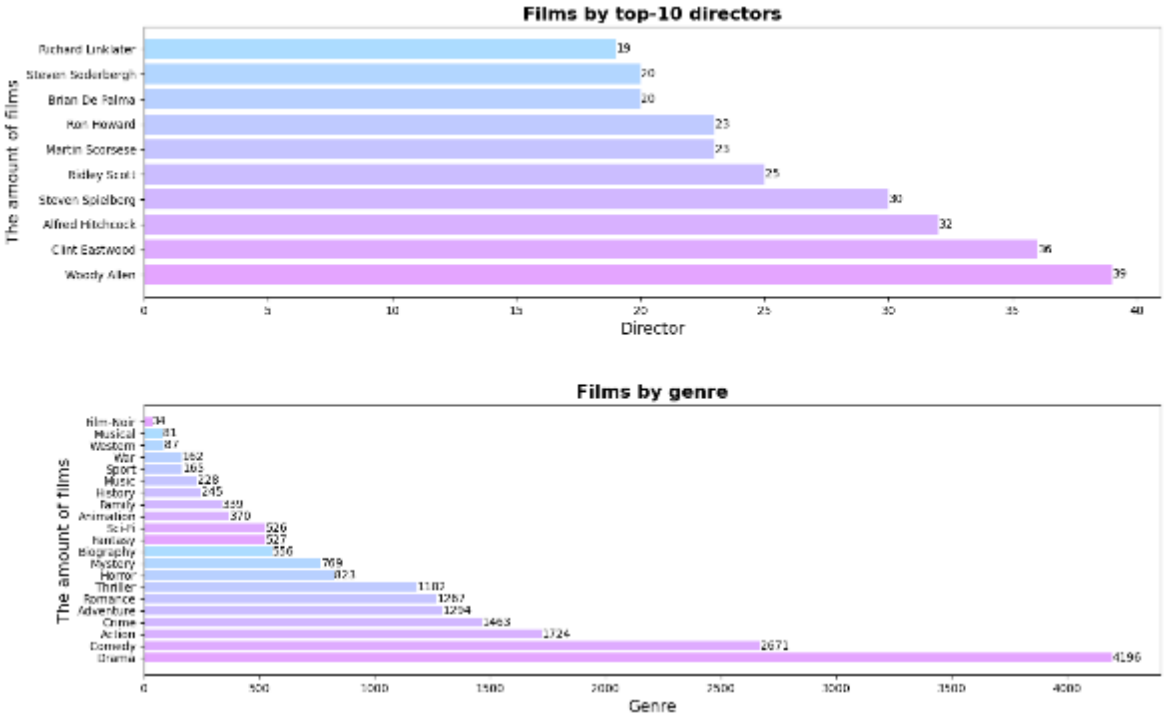


Рисунок 5.55 – Результат роботи trends.py

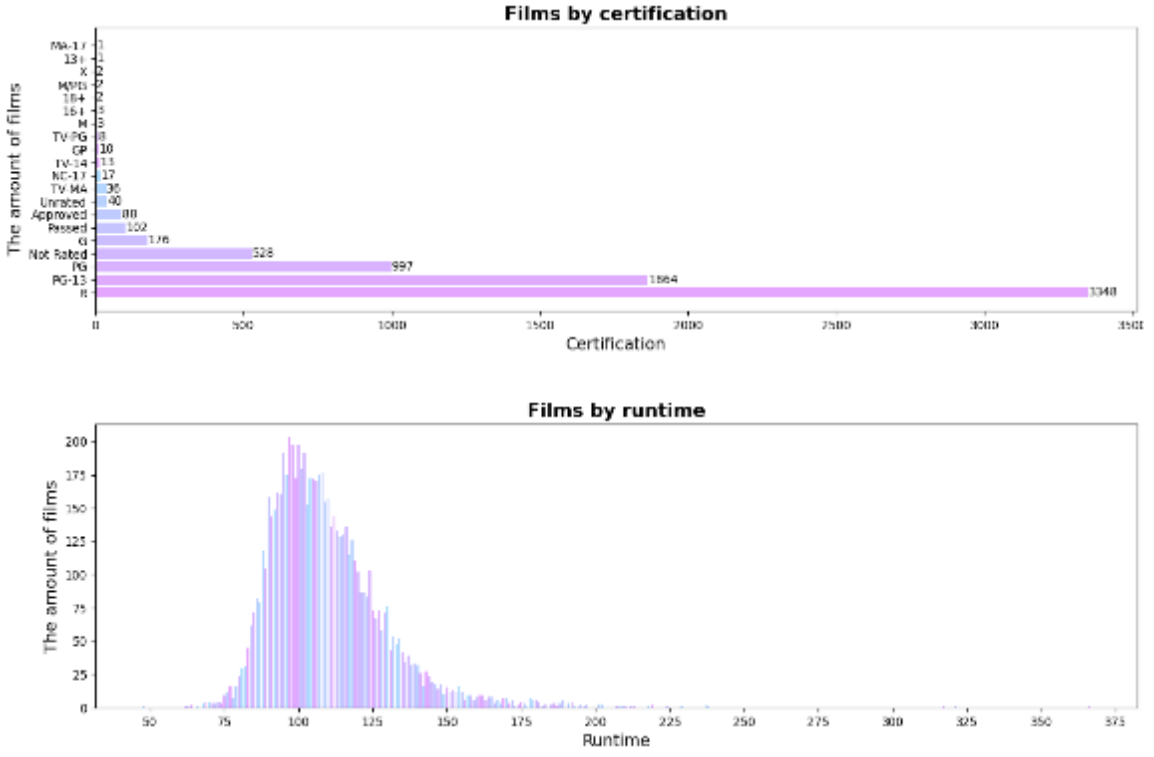


Рисунок 5.56 – Результат роботи trends.py

На рисунку 5.57 представлено два графіки-порівняння: порівняння місця по рейтингу і дохідності для топ 10 фільмів по рейтингу та порівняння

місця по рейтингу і дохідності для топ 10 фільмів по дохідності. Можна побачити, що часто дохідність та рейтинг фільму можуть йти різними шляхами і бути першим фільмом по рейтингу не буде означати, що фільм також буде першим по касових зборах.

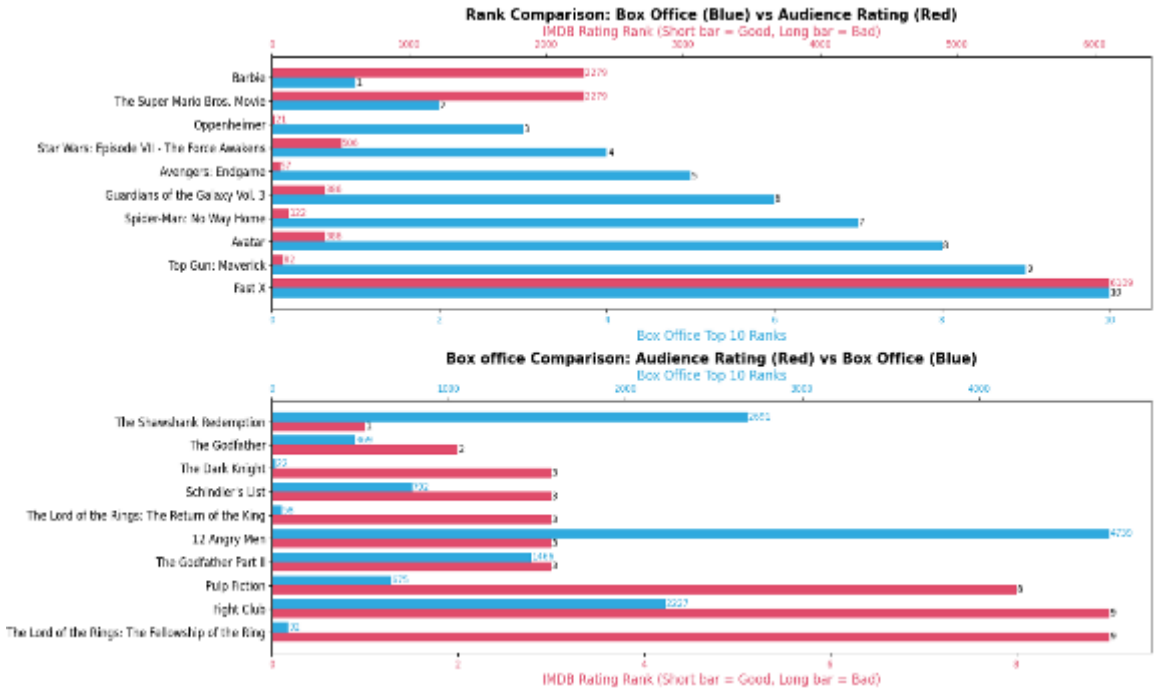


Рисунок 5.57 – Результат роботи trends.py

Після чого можна виконати запуск скрипту to_streamlit.py. Він виводить ці ж графіки, але створює для цього окрему вебсторінку, де й розміщує їх. Також відмінність цих графіків полягає в тому, що вони інтерактивні та можуть змінювати свій масштаб. Запускати його потрібно командою «streamlit run to_streamlit.py» (див. рис. 5.58).

```
-------
Starting streamlit script...
-------
File processed_data.csv exists, preparing to upload...
```

Рисунок 5.58 – Запуск скрипту to_streamlit.py

Після чого відкриється вебсторінка з графіками. Результат роботи представлено на рисунках 5.59-5.63.

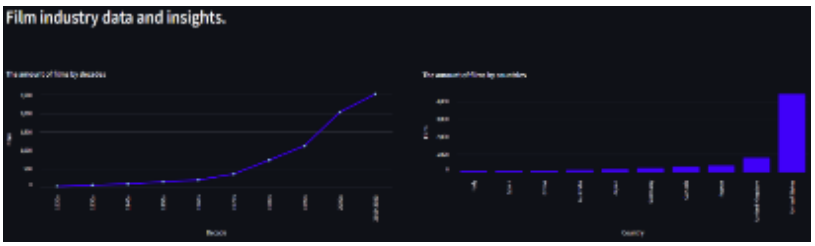


Рисунок 5.59 – Результат роботи скрипту to_streamlit.py

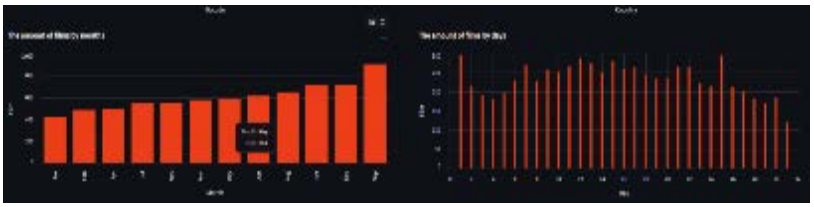


Рисунок 5.60 – Результат роботи скрипту to_streamlit.py

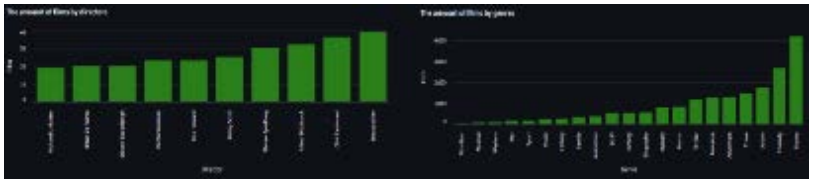


Рисунок 5.61 – Результат роботи скрипту to_streamlit.py

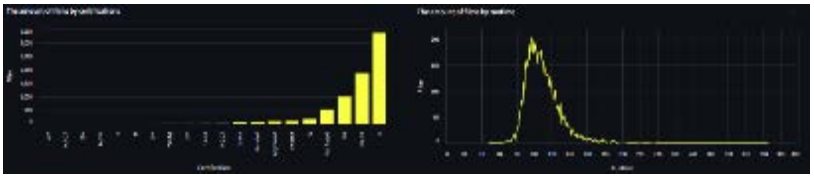


Рисунок 5.62 – Результат роботи скрипту to_streamlit.py

Рисунок 5.63 – Результат роботи скрипту to_streamlit.py

Останнім запустимо скріпт models.py – він відповідає за створення моделей машинного навчання та нейронних мереж. Спочатку на екран виводиться приклад екземпляру набору даних та відбувається підрахунок важливостей ознак для дохідності та рейтингу фільмів (див. рис. 5.64-5.66).

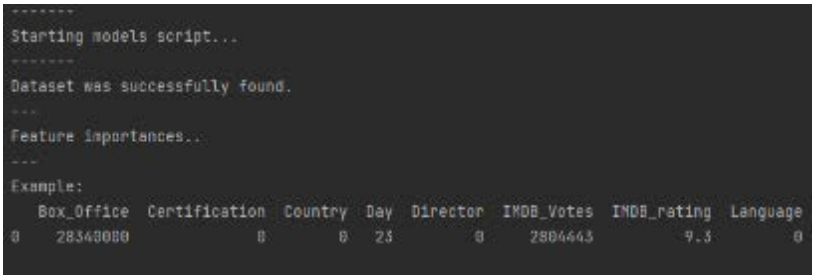


Рисунок 5.64 – Приклад екземпляру набору даних

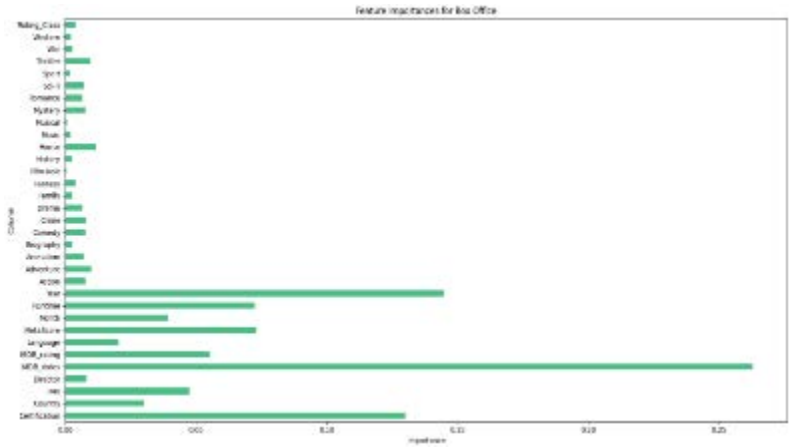


Рисунок 5.65 – Важливість ознак для дохідності

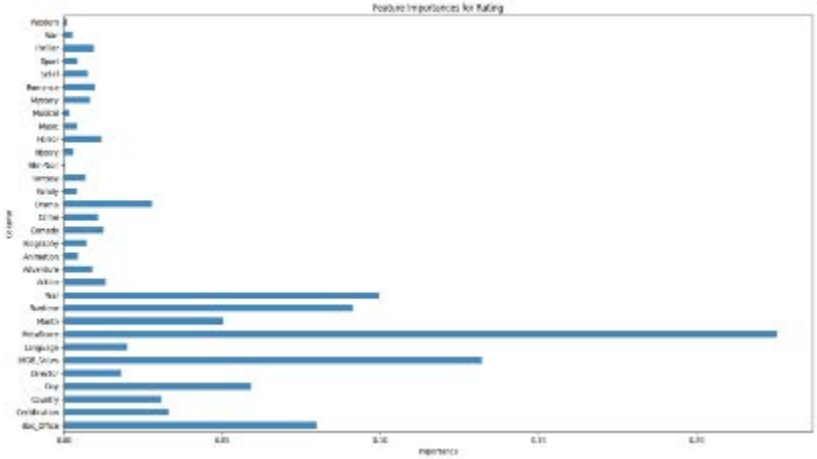


Рисунок 5.66 – Важливість ознак для рейтингу

Після цього на екран буде виведено результати класифікації колонки рейтингу фільму методом випадкового лісу (див. рис. 5.67).

```
-------
Classification: Rating Class(Random Forest)...
-------
Train dataset shape: (5792, 28)
Test dataset shape: (1449, 28)

---ML classification results:---
              Metrics Rating ML
0            Accuracy    0.7219
1  Missclassification    0.2781
2           Precision    0.7272
3              Recall    0.7219
4            F1-score    0.7221

              precision    recall  f1-score   support

           0       0.81      0.76      0.78       524
           1       0.66      0.74      0.70       623
           2       0.73      0.61      0.66       302

    accuracy                           0.72      1449
   macro avg       0.73      0.70      0.72      1449
weighted avg       0.73      0.72      0.72      1449
```

Рисунок 5.67 – Класифікація рейтингу фільму випадковим лісом

Спочатку виводяться загальна точність і хибність передбачень. Вони тримаються на доволі високому рівні – 0.72 та 0.27 відповідно. Проте ці параметри гарно працюють для класифікацій з двома класами. В нашому випадку класів більше і нам потрібно отримати додаткові відомості. Нижче виведено таблицю результатів по класах 0,1,2(«Good», «Average», «Bad»). Можна побачити по значенням precision(точність), що класи розпізнаються непогано, проте другий клас має значення 0.66, що менше за інші два. Це означає, що з всіх значень, що були віднесені до цього класу 66% занесено правильно, інші 34% належать іншим класам. Це може статись тому, що другий клас відповідає за «Average» клас, який містить найбільшу кількість фільмів у собі. Класифікатор буде помилково заносити інші менш кількісно

представлені класи до цього. Для інших двох класів значення точності є вищим – 0.73 та 0.81. Значення recall(повнота) також є доволі високими, проте третій клас має менше значення за інших – 0.61. Це означає, що серед усіх значень класу 2(«Bad» клас), модель змогла знайти 61%. Це може бути тому, що цей клас є менш представленим кількісно, ніж інші два. Значення f1-score є середнім гармонійним між точністю і повнотою, а support відповідає за кількість екземплярів класу у вибірці для тестування.

На рисунку 5.68 представлено результат роботи нейронної мережі для класифікації рейтингу. На екран виведено таблицю по класах. Можемо побачити, що значення точності та повноти відрізняються лише на 1-3%. Це може бути тому, що загальна кількість екземплярів у наборі доволі мала для нейронної мережі(менше за 10000). Проте можна також відзначити, що нейронна мережа розпізнала краще клас 2(«Bad» клас), ніж випадковий ліс – значення повноти у нейронної мережі дорівнює 0.69 проти 0.61 у випадкового лісу. За рахунок цього загальна точність передбачення за допомогою нейронної мережі підвищилась на 0,2 порівняно з випадковим лісом.

```
Epoch 40/40
580/580 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.7140 - loss: 0.6207 - val_accuracy: 0.7239 - val_loss: 0.5948
46/46 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step

Metrics report:
              precision    recall  f1-score   support

           0       0.83      0.78      0.81       524
           1       0.69      0.73      0.71       623
           2       0.71      0.69      0.70       302

    accuracy                           0.74      1449
   macro avg       0.74      0.74      0.74      1449
weighted avg       0.75      0.74      0.74      1449


---NN classification results:---
Accuracy: 0.7426
```

Рисунок 5.68 – Класифікація рейтингу фільму нейронної мережею

На рисунку 5.69 представлено матрицю помилок для класифікації за допомогою нейронної мережі. Зліва у стовбцях представлено реальних клас екземпляру, знизу у рядках представлено передбачений клас для цього

екземпляру. Головна діагональ відповідає за правильно передбачені значення для кожного з класів. Матриця показує нам те, що показувала і таблиця на рисунку 5.68. Можна побачити, що клас 0 і 2 доволі часто закидує до класу 1.

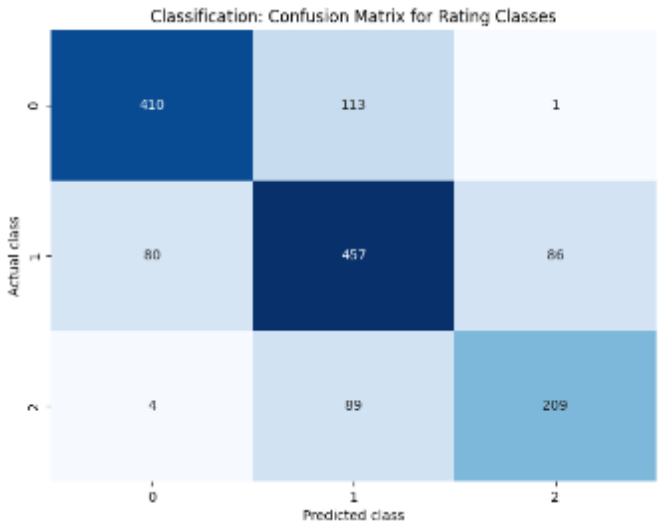


Рисунок 5.69 – Матриця помилок

На рисунку 5.70 представлено результат регресії методом випадкового лісу для доходу фільмів. Бачимо, що R2 score дорівнює 0.65 – ця метрика показує на скільки наша модель добре передбачає значення доходів. Максимальним значенням є 1, але такі результати можливі майже в «стерильних» умовах. У більшості випадків у житті, результати будуть менші. Це можна пояснити тим, що у значеннях доходів є викиди, велика різниця між найменшими та найбільшими значеннями. Значення середньої абсолютної помилки у доларах дорівнює 20 283 126 доларів. Це означає, що модель в середньому помиляється на приблизно 20 мільйонів доларів. Тобто, наприклад, якщо модель передбачила дохід фільму в 40 мільйонів доларів, то реальна цифра буде десь між 20 і 60 мільйонами доларів.

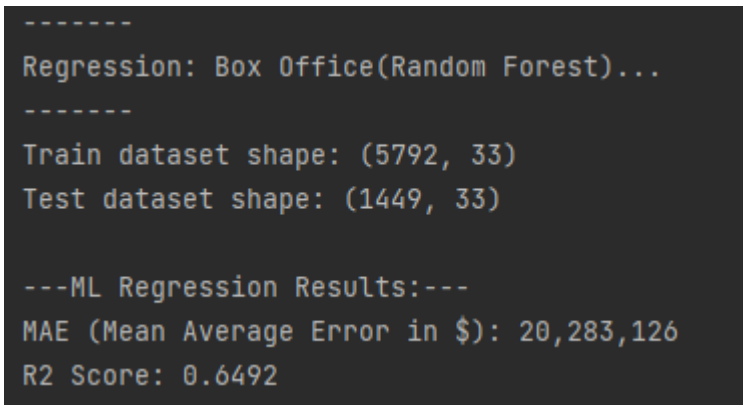


Рисунок 5.70 – Регресія для доходу методом випадкового лісу

На рисунку 5.71 представлено результат регресії для доходу за допомогою нейронної мережі. Як можна побачити – результати є дещо гіршими за випадковий ліс. Хоча середньо абсолютна помилка відрізняється не більше, ніж на 200000 тисяч доларів, значення r2 впало до 0.55. Це може бути тому, що загальна кількість екземплярів у наборі доволі мала для нейронної мережі(менше за 10000) та розкид значень дохідності занадто великий. Також важливим є і той факт, що не всі дані існують у відкритому доступі, і такі значення, як, наприклад, кількість показів у кінотеатрі, середня кількість глядачів за сеанс і т.і. дуже складно, а іноді і неможливо отримати. Також важливими є і маркетинговий бюджет, наявність і якість промо компанії, хайп у соцмережах. Це доводить, що ознаки, використані у цій роботі є безперечно важливими, але далеко не єдиними, які впливають на дохідність фільмів.

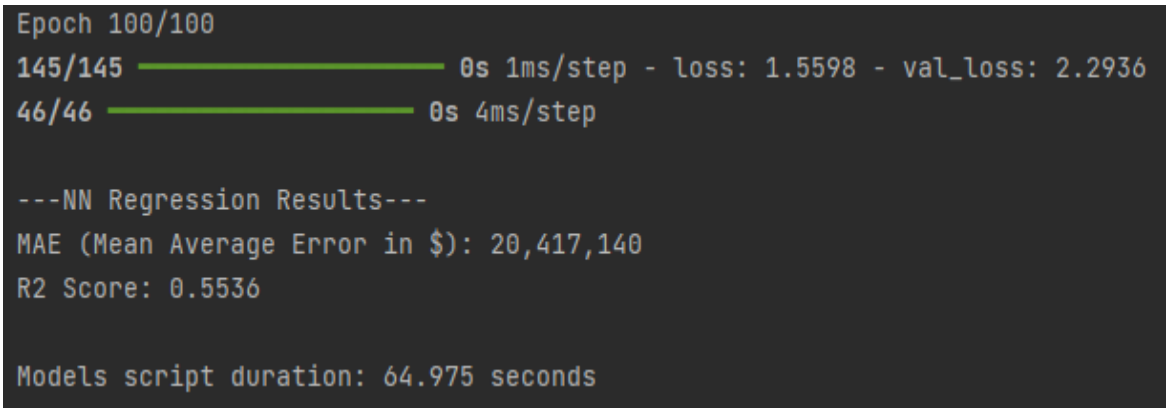


Рисунок 5.71 – Регресія для доходу за допомогою нейронної мережі

5.3 Тестування програми

Програму було протестовано на відповідність функціональним вимогам. Вона повністю відповідає їм, тому що забезпечує:

- завантаження даних з декількох джерел за допомогою API;
- попередню обробку даних, групування декількох наборів у один;
- збереження обробленого набору даних на пристрій та на AWS S3;
- тестування моделей алгоритмів класифікації та регресії;
- аналіз, візуалізацію і порівняння результатів;
- візуалізації патернів.

Програма виконує свої функції та працює без збоїв, функції не конфліктують одна з одною.

5.4 Висновки за розділом

У розділі було описано повний список вимог для експлуатації програми: встановлення Python, перевірка наявності pip, встановлення необхідних бібліотек для проєкту за допомогою requirements.txt, реєстрація на AWS, створення користувача, бакету, отримання ключів доступу та занесення цієї інформації до .env файлу. Також було покроково розписано експлуатацію програми, пояснено її результати та проведено тестування функціональних вимог.

ВИСНОВКИ

В процесі виконання дипломної роботи, як результат, було розроблено ПЗ для аналізу прибутку та рейтингу фільмів з подальшою можливістю прогнозування на основі даних з індустрії кіно. Проведено детальний аналіз предметної області, технічної літератури і матеріалів, пов’язаних з індустрією кіно, її розвитком, впливом на суспільство і тенденціями, виконано ознайомлення з аналогами, розібрано терміни штучного інтелекту, машинного навчання, нейронних мереж, API, розписано алгоритм роботи з AWS S3. Також було визначено вимоги до функціональних характеристик, надійності, складу та параметрів технічних засобів, прописано умови експлуатації. Було обрано мову програмування, середовище розробки та бібліотеки для роботи, було обґрунтовано вибір кожного пункту та представлено порівняльну характеристику. Детально і в доступній формі з необхідними скріншотами було описано процес для встановлення Python, pip, необхідних бібліотек, а також реєстрація на AWS, створення IAM ролі, ключів доступу та бакету для набору даних. У кінці було розібрано роботу програми та підбито підсумки її виконання.

Наукова цінність роботи полягає в тому, що було представлено ПЗ, яке вдосконалило аспекти всіх проаналізованих аналогів. Програма повністю відповідає функціональним вимогам.

Програма має практичну цінність, оскільки надає користувачам можливість мати змогу більш детально аналізувати і візуалізувати дані, легко завантажувати дані на пристрій та на хмарний сервіс AWS S3, прогнозувати результати.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Evolution Of Filmmaking | How To Make A Film In The 21st Century. Boiling Point Media [Electronic resource]. – Access mode: https://boilingpointmedia.com/from-silent-movies-to-modern-day-blockbusters-the-timeline-of-film-evolution/.

2. Leveraging Analytics to Produce Compelling and Profitable Film Content [Text] / [R. Behrens, N. Foutz, M. Franklin, J. Funk, F. Gutierrez-Navratil, J. Hofmann, U. Leibfried] // Journal of Cultural Economics. – 2019. – Special Issue. – P. 44. – Access mode: https://research.gold.ac.uk/id/eprint/28045/1/Producers%20Analytics%20Manuscript%20Accepted%20Draft%20Uncorrected.pdf.

3. L. Kharb. Forecasting Movie Rating Through Data Analytics / L. Kharb, D. Chahal, Vagisha // Data Science and Analytics. – Singapore : Springer Singapore, 2020. – P. 249–257. – Access mode: https://www.researchgate.net/publication/341673885_Forecasting_Movie_Rating_Through_Data_Analytics#pf8.

4. Global Film Production Hits Historic High, Surpassing Pre-Pandemic Levels. global-innovation-index [Electronic resource]. – Access mode: https://www.wipo.int/en/web/global-innovation-index/w/blogs/2025/global-film-production.

5. Film Industry Statistics 2024 By The Distributor, Running Time, Demographic and Box Office Revenue. Enterprise Apps Today [Electronic resource]. – Access mode: https://www.enterpriseappstoday.com/stats/film-industry-statistics.html.

6. Stryker C., Kavlakoglu E. What Is Artificial Intelligence (AI)? | IBM [Electronic resource]. – Access mode: https://www.ibm.com/think/topics/artificial-intelligence.

7. What is the history of artificial intelligence (AI)? Tableau [Electronic resource]. – Access mode: https://www.tableau.com/data-insights/ai/history.

8. Bergmann D. What is Machine Learning? | IBM [Electronic resource]. – Access mode: https://www.ibm.com/think/topics/machine-learning.

9. Bergmann D., Stryker C. What Is Model Training? | IBM [Electronic resource]. – Access mode: https://www.ibm.com/think/topics/model-training.

10. GeeksforGeeks. Machine Learning Lifecycle – GeeksforGeeks [Electronic resource]. – Access mode: https://www.geeksforgeeks.org/machine-learning/machine-learning-lifecycle/.

11. Machine Learning Tutorial – GeeksforGeeks [Electronic resource]. – Access mode: https://www.geeksforgeeks.org/machine-learning/machine-learning/.

12. Demystifying Machine Learning: A Comprehensive Guide [Electronice resource]. – Access mode: https://www.oracle.com/europe/artificial-intelligence/machine-learning/what-is-machine-learning/.

13. Bergmann D. What Is Deep Learning? | IBM [Electronic resource]. – Access mode: https://www.ibm.com/think/topics/deep-learning#763338456.

14. GeeksforGeeks. Advantages and Disadvantages of Deep Learning – GeeksforGeeks [Electronic resource]. – Access mode: https://www.geeksforgeeks.org/deep-learning/advantages-and-disadvantages-of-deep-learning/.

15. How AI is Transforming the Film Industry. UNIT.LT [Electronic resource]. – Access mode: https://unit.lt/blog/how-ai-is-transforming-the-film-industry/.

16. AI in Film Market. Market.us [Electronic resource]. – Access mode: https://market.us/report/ai-in-film-market/.

17. Кошель Н. Що таке ROI, та Де порахувати рентабельність інвестицій. Netpeak Journal – Медіа про інтернет-маркетинг та онлайн-бізнес у деталях [Електрон. ресурс]. – Режим доступу: https://netpeak.net/uk/blog/shcho-take-roi-ta-de-porakhuvati-rentabel-nist-investitsiy/.

18. Biehl, M. API Architecture: The Big Picture for Building APIs [Text] / M. Biehl. – CreateSpace Independent Publishing Platform, 2015. – 190 p. – Access mode: https://www.google.com.ua/books/edition/API_Architecture/6D64DwAAQBAJ?hl=uk&gbpv=1&dq=api+features&pg=PA80&printsec=frontcove.

19. Goodwin M. What Is an API (Application Programming Interface)? | IBM [Electronic resource]. – Access mode: https://www.ibm.com/think/topics/api.

20. What is an API? A Beginner's Guide to APIs | Postman. Postman: The World's Leading API Platform [Electronic resource]. – Access mode: https://www.postman.com/what-is-an-api/.

21. Kirvan P., Barney N., Gillis A. S. What is AWS? Ultimate guide to Amazon Web Services [Electronic resource]. – Access mode: https://www.techtarget.com/searchaws/definition/Amazon-Web-Services.

22. Chillingworth A. How does Patreon work? Your complete guide | Epidemic Sound [Electronic resource]. – Access mode: https://www.epidemicsound.com/blog/how-does-patreon-work/.

23. IMDb: Ratings, Reviews, and Where to Watch the Best Movies & TV Shows. IMDb [Electronic resource]. – Access mode: https://www.imdb.com/.

24. Comscore – Wikipedia [Electronic resource]. – Access mode: https://en.wikipedia.org/wiki/Comscore.

25. Comscore is a trusted currency for planning, transacting, and evaluating media across platforms. Comscore, Inc. [Elecntonic resource]. – Access mode: https://www.comscore.com/.

26. Largo – We accelerate filmmaking with the instant insights [Electronic resource]. – Access mode: https://home.largo.ai/.

27. Python (programming language) – Wikipedia [Electronic resource]. – Access mode: https://en.wikipedia.org/wiki/Python_(programming_language).

28. R vs. Python for machine learning. Educative: AI-Powered Interactive Courses for Developers [Electronic resource]. – Access mode: https://www.educative.io/blog/r-vs-python-machine-learning.

29. Python vs R for Data Science: Which Should You Learn – Datacamp [Electronic resource]. – Access mode: https://www.datacamp.com/blog/python-vs-r-for-data-science-whats-the-difference.

30. Spec India. Java vs Python in Machine Learning: 2025 Guide. SPEC INDIA [Electronic resource]. – Access mode: https://www.spec-india.com/blog/java-vs-python-for-machine-learning.

31. 11 Best IDEs for Python developers in 2025. Pieces – Long term memory for your whole workstream [Electronic resource]. – Access mode: https://pieces.app/blog/best-ide-for-python.

32. GeeksforGeeks. What is python scikit library? – GeeksforGeeks [Electronic resource]. – Access mode: https://www.geeksforgeeks.org/machine-learning/what-is-python-scikit-library/.

33. GeeksforGeeks. Introduction to TensorFlow – GeeksforGeeks [Electronic resource]. – Access mode: https://www.geeksforgeeks.org/python/introduction-to-tensorflow/.

34. Introduction to Matplotlib – GeeksforGeeks [Electronic resource]. – Access mode: https://www.geeksforgeeks.org/python/python-introduction-matplotlib/.

35. Streamlit – A faster way to build and share data apps [Electronic resource]. – Access mode: https://streamlit.io/.

36. Vega-Altair: Declarative Visualization in Python – Vega-Altair 6.0.0 documentation [Electronic resource]. – Access mode: https://altair-viz.github.io/.

37. Requests: HTTP for Humans™ – Requests 2.32.5 documentation [Electronic resource]. – Access mode: https://requests.readthedocs.io/en/latest/.

38. Download Python. Python.org [Electronic resource]. – Access mode: https://www.python.org/downloads/.

39. S3 Getting Started. Amazon Web Services, Inc. [Electronic resource]. – Access mode: https://aws.amazon.com/s3/getting-started/.

ДОДАТОК А

Технічне завдання

А.1 Призначення та область застосування програми

Програма, що була розроблена, має на меті аналіз прибутку та рейтингу фільмів на основі доступних даних з індустрії кіно з подальшою можливістю прогнозування.

Основною областю застосування програми є кіноіндустрія.

А.2 Підстави для розробки

Підставою для розробки є завдання на дипломну кваліфікаційну роботу на тему «Дослідження та програмна реалізація методів аналізу прибутків та рейтингу на основі даних з індустрії кіно», затверджене наказом Національного університету «Запорізька політехніка» № 447 від 30 вересня 2025 р.

А.3 Основні вимоги до програми, що розробляється

А.3.1 Вимоги до функціональних характеристик

Розроблене ПЗ для аналізу прибутку та рейтингу фільмів з подальшою можливістю прогнозування забезпечує виконання таких функцій:

- завантаження даних за допомогою API з декількох джерел;
- попередня обробка даних, групування даних з декількох наборів даних;
- аналіз трендів і патернів для прибутку та рейтингів;
- побудова моделей машинного навчання та нейронних мереж;
- візуалізація даних, представлення їх у зручній для користувача формі(як у вигляді окремих графіків, так і у вигляді вебсторінки у браузері);
- завантаження обробленого набору даних на хмарний сервіс, звідки його можна вивантажити в будь-який момент.

А.3.2 Вимоги до інтерфейсу програми

Інтерфейс ПЗ для аналізу прибутку та рейтингу фільмів з подальшою можливістю прогнозування повинен бути зрозумілим для користувача, а також повідомляти його про всі головні кроки алгоритму дії програми та виводити на екран повідомлення про помилки, якщо такі будуть.

А.3.3 Вимоги до надійності

Вимоги до надійності ПЗ включають в себе:

– перевірку алгоритмів, що використовуються у програмі, на точність та стабільність результатів;

– виведення повідомлень для користувача у разі виникнення помилок;

– виведення повідомлень для користувача, які показують етапи роботи програми;

– виведення повідомлень для користувача, які показують результати роботи програми.

А.3.4 Умови експлуатації

Для експлуатації ПЗ необхідно мати стабільне підключення до мережі Інтернет для здійснення API запитів. Також додатково необхідно встановити необхідну версію Python з потрібними бібліотеками (які будуть прописані у файлі requirements.txt) та системою керуваннями пакетів pip, бути знайомим з AWS, щоб завантажити набір даних з їх хмарного сховища.

А.3.5 Вимоги до складу та параметрів технічний засобів

Комп’ютер користувача для коректної роботи з ПЗ має відповідати таким вимогам:

– стабільне підключення до мережі Інтернет для здійснення API запитів;

– операційна система: Windows 10 або вище;

– версія Python(не нижче за 3.12.1) та pip(не нижче за 24.3.1);

– вільний дисковий простір для встановлення необхідних бібліотек і роботи програми без збоїв: мінімум 2-3 ГБ загального вільного простору;

– оперативна пам’ять: мінімум 8 ГБ;

– процесор: не менше 4 ядер з тактовою частотою не менше 3,4 ГГц.

А.3.6 Необхідні стадії і терміни розробки

Розробка ПЗ для аналізу прибутку і рейтингу фільмів з подальшою можливістю прогнозування включає наступні стадії та терміни:

– аналіз вимог: визначення вимог до програми. Термін: 1 тиждень;

– проєктування системи: розробка архітектури та вибір технологічних засобів. Термін: 1 тиждень;

– реалізація та програмування: написання коду програми та реалізація визначених функцій. Термін: 2 тижні;

– тестування та налагодження: проведення випробувань для виявлення та усунення помилок. Термін: 1 тиждень.

А.3.7 Види випробувань

Для забезпечення якості та надійності буде проведено наступні види випробувань:

– функціональні тести: перевірка відповідності функціональних вимог програми;

– інтеграційні тести: перевірка взаємодії між різними частинами програми, а також з коректністю завантаження у хмарне сховище;

– системні тести: тестування програми в цілому.

ДОДАТОК Б

Текст програми

fetch_data.py:

```
import os.path
import os
from pathlib import Path

import kagglehub
import shutil
import pandas as pd
import requests
import time
import json
import sys

API_KEY_OMDB = '13b2754f'
url_omdb = 'https://www.omdbapi.com/'
def get_movies(path, cache_path):
    get_movies_main(path, cache_path)
def get_movies_main(path, cache_path): #imdb dataset
    print(f"-------\nDownloading is about to start...\n-------")
    new_path = path
    cache_dir = Path(cache_path)
    try:
        if (cache_dir / "dk123891").is_dir():
            print(f"Main dataset is already on your device..\n")
        else:
            print(f"Downloading the main dataset..\n")
            actual_path = kagglehub.dataset_download("dk123891/10000-movies-
data")
            actual_path = os.path.join(actual_path, "data.csv")
            dest_path = os.path.join(new_path)
            try:
                shutil.copy(actual_path, dest_path)
            except FileNotFoundError:
                print(f"There is no such file: {actual_path}")
            except Exception as e:
                print(f"Copy error: {e}")
    except Exception as e:
        print(f"Failed to load main dataset: {e}\nClosing the program...")
        sys.exit(1)
    handle_movies_main(path, cache_path)

def handle_movies_main(path, cache_path):
    df = pd.read_csv(os.path.join(path, "data.csv"))
```

```
    df.rename({'Movie Name': 'Title', 'Year of Release': 'Year', 'Run Time in minutes': 'Runtime',
               'Movie Rating': 'IMDB_rating', 'Votes': 'IMDB_Votes', 'Gross': 'Box_Office'}, axis=1, inplace=True)
    df.drop_duplicates(['Title'], keep='first', inplace=True)

    mask_metascore = df['MetaScore'].isna()
    isna_metascore = df[mask_metascore]['Title']
    mask_box_office = df['Box_Office'].isna()
    isna_box_office = df[mask_box_office]['Title']
    mask_certification = df['Certification'].isna()
    isna_certification = df[mask_certification]['Title']
    mask_titles = df['Title']
    masks = [mask_titles, isna_metascore, isna_box_office, isna_certification]
    get_movies_additionally(path, cache_path, masks)
def get_movies_additionally(path, cache_path, masks):
    print(f"-------\nAttempting to download additional necessary datasets...\n-------")
    get_movies_1(path, cache_path)
    get_movies_2(path, cache_path)
    get_movies_3(path, cache_path)
    get_movies_4(path, masks)
    return 0
def get_movies_1(path, cache_path): #tmdb dataset
    new_path = path
    cache_dir = Path(cache_path)
    try:
        if (cache_dir / "balaka18").is_dir():
            print(f"Dataset1 is already on your device..\n")
        else:
            print(f"Downloading the dataset1..\n")
            actual_path = kagglehub.dataset_download("balaka18/tmdb-top-10000-popular-movies-dataset")
            actual_path = os.path.join(actual_path, "movies_tmdb_popular.csv")
            dest_path = os.path.join(new_path, "data1.csv")
            try:
                shutil.copy(actual_path, dest_path)
            except FileNotFoundError:
                print(f"There is no such file: {actual_path}")
            except Exception as e:
                print(f"Copy error: {e}")
    except Exception as e:
        print(f"Failed to load dataset1: {e}\nClosing the program...")
        sys.exit(1)

def get_movies_2(path, cache_path): #movie lens dataset
```

```
    new_path = path
    cache_dir = Path(cache_path)
    try:
        if (cache_dir / "rounakbanik").is_dir():
            print(f"Dataset2 is already on your device..\n")
        else:
            print(f"Downloading the dataset2..\n")
            actual_path = kagglehub.dataset_download("rounakbanik/the-movies-dataset")
            actual_path = os.path.join(actual_path, "movies_metadata.csv")
            dest_path = os.path.join(new_path, "data2.csv")
            try:
                shutil.copy(actual_path, dest_path)
            except FileNotFoundError:
                print(f"There is no such file: {actual_path}")
            except Exception as e:
                print(f"Copy error: {e}")
    except Exception as e:
        print(f"Failed to load dataset2: {e}\nClosing the program...")
        sys.exit(1)

def get_movies_3(path, cache_path):  #box office dataset
    new_path = path
    cache_dir = Path(cache_path)
    try:
        if (cache_dir / "aditya126").is_dir():
            print(f"Dataset3 is already on your device..\n")
        else:
            print(f"Downloading the dataset3..\n")
            actual_path = kagglehub.dataset_download("aditya126/movies-box-office-dataset-2000-2024")
            actual_path = os.path.join(actual_path, "enhanced_box_office_data(2000-2024)u.csv")
            dest_path = os.path.join(new_path, "data3.csv")
            try:
                shutil.copy(actual_path, dest_path)
            except FileNotFoundError:
                print(f"There is no such file: {actual_path}")
            except Exception as e:
                print(f"Copy error: {e}")
    except Exception as e:
        print(f"Failed to load dataset3: {e}\nClosing the program...")
        sys.exit(1)

def get_movies_4(path, masks): #omdb dataset
```

```
    new_path = os.path.join(path, 'data4.json')
    file_dir = Path(path)

    result = []
    title_suc, title_skip = 0, 0
    title_count = len(masks[0])
    try:
        if (file_dir / "data4.json").is_file():
            print(f"Dataset4 is already on your device..\n")
        else:
            print(f"Downloading the dataset4..\n")
            for i, title in enumerate(masks[0]):
                params_omdb = {'apikey': API_KEY_OMDB, 't': title}
                try:
                    response = requests.get(url_omdb, params=params_omdb)
                    response.raise_for_status()
                    data_omdb = response.json()
                    if data_omdb.get('Response') == 'False':
                        title_skip += 1
                    else:
                        title_suc += 1
                        output = f"{i+1}) Title {title} was successfully loaded. {i+1}/{title_count}"
                        print(f"\r{output:<100}", end="")
                    result.append(data_omdb)
                    time.sleep(0.2)
                except requests.exceptions.HTTPError as e:
                    print(f"\nFailed to load dataset4, HTTP error occurred: {e}\nClosing the program...")
                    sys.exit(1)
                except Exception as e:
                    print(f"\nFailed to load dataset4: {e}\nClosing the program...")
                    sys.exit(1)
            print(f"\n{result[:10]}\n")
            print(f"Successfully downloaded titles: {title_suc}")
            print(f"Skipped titles: {title_skip}")
            try:
                with open(new_path, 'w', encoding='utf-8') as f:
                    json.dump(result, f, indent=4, ensure_ascii=False)
                print(f"Dataset4 was successfully created.")
            except Exception as e:
                print(f"Failed to save dataset4: {e}\nClosing the program...")
                sys.exit(1)
    except Exception as e:
        print(f"Failed to load dataset4: {e}\nClosing the program...")
```

```
        sys.exit(1)

if __name__ == '__main__':
    start = time.time()
    print(f"-------\nStarting fetching data script...\n-------")
    pd.set_option('display.max_rows', 30)
    pd.set_option('display.max_columns', 50)
    pd.set_option('display.width', 5000)

    path = Path("../data/raw")
    path.mkdir(parents=True, exist_ok=True)
    Path("../data/processed").mkdir(parents=True, exist_ok=True)
    cache_path = Path.home() / ".cache/kagglehub/datasets"
    get_movies(path, cache_path)

    end = time.time()
    duration = end - start
    print(f"-------\nFetching script duration: {duration:0.3f} seconds\n-------")
```

preprocessing.py:

```
import os
import time
from pathlib import Path
import sys

import pandas as pd
import ast
from sklearn.preprocessing import MultiLabelBinarizer

import to_processed


def main():
    start = time.time()
    print(f"-------\nStarting preprocessing script...\n-------")
    pd.set_option('display.max_rows', 50)
    pd.set_option('display.max_columns', 50)
    pd.set_option('display.width', 5000)

    path = "../data/raw"
    if Path(path).is_dir():
        print(f"Necessary directory exists, preparing for preprocessing...")
        handle_movies_main(path)
    else:
```

```
        print(f"Error: Necessary directory cannot be found, please run fetch_data.py script")
        sys.exit(1)
    end = time.time()
    duration = end - start
    print(f"-------\nPreprocessing script duration: {duration:0.3f} seconds\n-------")


def handle_movies_main(path):
    print(f"-------\nMain Dataset\n-------")
    if (Path(path) / 'data.csv').is_file():
        print(f"Necessary dataset exists, preparing for preprocessing...")
    else:
        print(f"Error: Necessary dataset cannot be found, please run fetch_data.py script")
        sys.exit(1)

    df = pd.read_csv(os.path.join(path, "data.csv"))
    df.drop(columns=['Unnamed: 0', 'Description'], inplace=True)
    df.rename({'Movie Name': 'Title', 'Year of Release': 'Year', 'Run Time in minutes': 'Runtime',
               'Movie Rating': 'IMDB_rating', 'Votes': 'IMDB_Votes', 'Gross': 'Box_Office'}, axis=1, inplace=True)
    print(f"\nDataset example:\n{df.head(5)}\n")

    df.drop_duplicates(['Title'], keep='first', inplace=True)
    print(f"Duplicates have been dropped...\n")
    print(f"All not a number values from dataset:\n{df.isna().sum()}\n")

    mask_metascore = df['MetaScore'].isna()
    isna_metascore = df[mask_metascore]['Title']
    mask_box_office = df['Box_Office'].isna()
    isna_box_office = df[mask_box_office]['Title']
    mask_certification = df['Certification'].isna()
    isna_certification = df[mask_certification]['Title']
    mask_titles = df['Title']
    masks = [mask_titles, isna_metascore, isna_box_office, isna_certification]
    get_additional_columns(path, masks, df)

def get_additional_columns(path, masks, df):
    df1 = get_additional_columns1(path, masks)
    df2 = get_additional_columns2(path, masks)
    df3 = get_additional_columns3(path, masks)
    df4 = get_additional_columns4(path, masks)
    datasets_join(df, df1, df2, df3, df4, masks)
```

```
def get_additional_columns1(path, masks):  # tmdb dataset
    print(f"-------\nDataset1\n-------")
    if (Path(path) / 'data1.csv').is_file():
        print(f"Necessary dataset exists, preparing for preprocessing...")
    else:
        print(f"Error: Necessary dataset cannot be found, please run fetch_data.py
script")
        sys.exit(1)
    df = pd.read_csv(os.path.join(path, "data1.csv"))
    df.drop(['overview', 'popularity', 'vote_count', 'vote_average'], axis=1,
inplace=True)
    df.rename(
        {'title': 'Title', 'original_lang': 'Language'}, axis=1, inplace=True)
    print(f"\nDataset1 example:\n{df.head(5)}\n")
    df.drop_duplicates(['Title'], keep='first', inplace=True)
    print(f"All not a number values from dataset1:\n{df.isna().sum()}\n")

    mask_titles = df['Title']
    titles_overlap = pd.Series(list(set(masks[0]) & set(mask_titles)))
    mask_titles_overlap = df['Title'].isin(titles_overlap)
    df_to_return = pd.DataFrame(df[mask_titles_overlap])

    mask_rel_date = df['rel_date'].isna()
    print(
        f"The number of necessary for us release date values with NA value:
{pd.Series(list(set(df[mask_rel_date]['Title']) & set(titles_overlap))).count()}\n")
    return df_to_return

def get_additional_columns2(path, masks):  # movie lens dataset
    print(f"-------\nDataset2\n-------")
    if (Path(path) / 'data2.csv').is_file():
        print(f"Necessary dataset exists, preparing for preprocessing...")
    else:
        print(f"Error: Necessary dataset cannot be found, please run fetch_data.py
script")
        sys.exit(1)
    df = pd.read_csv(os.path.join(path, "data2.csv"))
    df.drop(
        columns=['budget', 'adult', 'original_title', 'belongs_to_collection', 'genres',
'homepage', 'id', 'imdb_id',
                 'original_language', 'overview', 'popularity', 'poster_path',
'production_companies',
                 'production_countries', 'runtime', 'spoken_languages', 'status', 'tagline',
                 'video', 'vote_average', 'vote_count'], inplace=True)
```

```
    df.rename(columns={'title': 'Title', 'revenue': 'Box_Office'}, inplace=True)
    print(f"\nDataset2 example:\n{df.head(5)}\n")
    df.drop_duplicates(['Title'], keep='first', inplace=True)

    df.dropna(subset=['Title'], inplace=True)
    print(f"All not a number values from dataset2:\n{df.isna().sum()}\n")

    mask_titles = df['Title']
    titles_overlap = pd.Series(list(set(masks[0]) & set(mask_titles)))
    mask_titles_overlap = df['Title'].isin(titles_overlap)

    df_to_return = pd.DataFrame(df[mask_titles_overlap])
    mask_release_date = df['release_date'].isna()
    print(
        f"The number of necessary for us release date values with NA value: {pd.Series(list(set(df[mask_release_date]['Title']) & set(titles_overlap))).count()}\n")
    return df_to_return

def get_additional_columns3(path, masks):  # box office dataset
    print(f"-------\nDataset3\n-------")
    if (Path(path) / 'data3.csv').is_file():
        print(f"Necessary dataset exists, preparing for preprocessing...")
    else:
        print(f"Error: Necessary dataset cannot be found, please run fetch_data.py script")
        sys.exit(1)
    df = pd.read_csv(os.path.join(path, "data3.csv"))
    df.drop(columns=['Rank', '$Domestic', 'Domestic %', '$Foreign', 'Foreign %', 'Year', 'Genres',
                     'Original_Language', 'Vote_Count', 'Rating'], inplace=True)
    df.rename(columns={'Release Group': 'Title', '$Worldwide': 'Box_Office', 'Production_Countries': 'Country'}, inplace=True)
    df.drop_duplicates(['Title'], keep='first', inplace=True)
    print(f"\nDataset3 example:\n{df.head(5)}\n")
    print(f"All not a number values from dataset3:\n{df.isna().sum()}\n")

    mask_titles = df['Title']
    titles_overlap = pd.Series(list(set(masks[0]) & set(mask_titles)))
    mask_titles_overlap = df['Title'].isin(titles_overlap)

    df_to_return = pd.DataFrame(df[mask_titles_overlap])
    return df_to_return

def get_additional_columns4(path, masks):
```

```
    print(f"-------\nDataset4\n-------")
    if (Path(path) / 'data4.json').is_file():
        print(f"Necessary dataset exists, preparing for preprocessing...")
    else:
        print(f"Error: Necessary dataset cannot be found, please run fetch_data.py script")
        sys.exit(1)
    df = pd.read_json(os.path.join(path, "data4.json"))
    df.drop(columns=['Year', 'Runtime', 'Genre', 'Director', 'Writer', 'Actors', 'Plot', 'Poster', 'Awards', 'Ratings',
                     'imdbRating', 'imdbVotes', 'imdbID', 'Type', 'DVD', 'Production', 'Website',
                     'Response', 'totalSeasons', 'Error'], inplace=True)
    df['BoxOffice'] = df['BoxOffice'].astype(str).str.replace(r'[^\d.]', '', regex=True)
    print(f"\nDataset4 example:\n{df.head(5)}\n")
    df.drop_duplicates(['Title'], keep='first', inplace=True)
    print(f"All not a number values from dataset4:\n{df.isna().sum()}\n")
    df.dropna(subset=['Title', 'Rated', 'Released', 'Language', 'Country', 'Metascore'], inplace=True)
    df.rename({'Rated': 'Certification', 'Metascore': 'MetaScore'}, axis=1, inplace=True)

    mask_titles = df['Title']
    titles_overlap = pd.Series(list(set(masks[0]) & set(mask_titles)))
    mask_titles_overlap = df['Title'].isin(titles_overlap)

    mask_box_office = df['BoxOffice'].isna()
    print(
        f"The number of necessary for us box office with NA value: {pd.Series(list(set(df[mask_box_office]['Title']) & set(titles_overlap))).count()}")
    df.dropna(subset=['BoxOffice'], inplace=True)
    df_to_return = pd.DataFrame(df[mask_titles_overlap])
    return df_to_return

def datasets_join(df, df1, df2, df3, df4, masks):
    print(f"-------\nJoining the datasets..\n-------")
    print(f"\nMain dataset example:\n{df.head(3)}\n")
    print(f"\nDataset1 example:\n{df1.head(3)}\n")
    print(f"\nDataset2 example:\n{df2.head(3)}\n")
    print(f"\nDataset3 example:\n{df3.head(3)}\n")
    print(f"\nDataset4 example:\n{df4.head(3)}\n")
    print(f"Number of movies in main dataset: {len(df.index)}")
    print(f"Number of movies in dataset1: {len(df1.index)}")
    print(f"Number of movies in dataset2: {len(df2.index)}")
    print(f"Number of movies in dataset3: {len(df3.index)}")
```

```
    print(f"Number of movies in dataset4: {len(df4.index)}")
    additional_columns = add_additional_columns(df, df1, df2, df3, df4, masks)
    necessary_columns = add_necessary_columns(df, df2, df3, df4, masks)
    merge_dfs(df, additional_columns, necessary_columns, masks)

def add_additional_columns(df, df1, df2, df3, df4, masks):
    print(f"-------\nAdditional columns..\n-------")
    language = add_original_lg(df, df1, df4, masks)
    country = add_country(df, df3, df4, masks)
    day, month = add_release_date(df, df1, df2, df4, masks)
    additional_columns = pd.DataFrame({'Day': day, 'Month': month, 'Language':
language, 'Country': country})
    return additional_columns

def add_original_lg(df, df1, df4, masks):
    print(f"---\nOriginal languages..\n---")
    print(f"All not a number original languages from dataset1:
{df1['Language'].isna().sum()}")
    print(f"All not a number original languages from dataset4:
{df4['Language'].isna().sum()}\n")
    languages = {}

    df1['Language'] = df1['Language'].replace({'en': 'English', 'fr': 'French', 'ja':
'Japanese', 'it': 'Italian',
                                               'es': 'Spanish', 'de': 'German', 'cn': 'Simplified Chinese',
'ko': 'Korean',
                                               'zh': 'Chinese', 'da': 'Danish', 'sv': 'Swedish', 'hi': 'Hindi',
                                               'ru': 'Russian', 'pt': 'Portuguese', 'pl': 'Polish',
                                               'no': 'Norwegian', 'th': 'Thai', 'nl': 'Dutch', 'tr': 'Turkish',
                                               'fa': 'Persian', 'cs': 'Czech', 'hu': 'Hungarian', 'id':
'Indonesian',
                                               'fi': 'Finnish', 'sr': 'Serbian', 'bn': 'Bengali', 'ta': 'Tamil',
                                               'el': 'Greek', 'ar': 'Arabic', 'te': 'Telugu',
                                               'he': 'Hebrew', 'bs': 'Bosnian', 'sh': 'Serbo-Hrvatski', 'ro':
'Romanian',
                                               'is': 'Icelandic', 'et': 'Estonian'})
    df1 = df1.drop((df1['Language'] == 'xx').index)

    df4['Language'] = df4['Language'].astype(str).str.split(',').str[0]
    df4['Language'] = df4['Language'].replace('nan', pd.NA)

    dict1 = dict(zip(df1['Title'], df1['Language']))
    dict2 = dict(zip(df4['Title'], df4['Language']))
    for title in masks[0].values:
        val = dict1.get(title)
```

```
        if val is None:
            val = dict2.get(title)
        languages[title] = val if val is not None else pd.NA
    languages_series = pd.Series(languages)
    print(f"The number of NA values in languages series: {languages_series.isna().sum()}")
    return languages_series
def add_country(df, df3, df4, masks):
    print(f"---\nProduction countries..\n---")
    print(f"All not a number countries from dataset1: {df3['Country'].isna().sum()}")
    print(f"All not a number countries from dataset4: {df4['Country'].isna().sum()}\n")
    df3.dropna(subset='Country', inplace=True)
    df4.dropna(subset='Country', inplace=True)
    countries = {}

    df3['Country'] = df3['Country'].astype(str).str.split(',').str[0]
    df3['Country'] = df3['Country'].replace('nan', pd.NA)
    df4['Country'] = df4['Country'].astype(str).str.split(',').str[0]
    df4['Country'] = df4['Country'].replace('nan', pd.NA)
    df3.replace({'United States of America': 'United States', 'USA': 'United States'}, inplace=True)
    df4.replace({'United States of America': 'United States', 'USA': 'United States'}, inplace=True)

    dict1 = dict(zip(df3['Title'], df3['Country']))
    dict2 = dict(zip(df4['Title'], df4['Country']))

    for title in masks[0].values:
        val = dict1.get(title)
        if val is None:
            val = dict2.get(title)
        countries[title] = val if val is not None else pd.NA

        countries_series = pd.Series(countries)
    print(f"The number of NA values in country series: {countries_series.isna().sum()}")
    return countries_series
def add_release_date(df, df1, df2, df4, masks):
    print(f"---\nRelease dates..\n---")
    print(f"All not a number release dates from dataset1: {df1['rel_date'].isna().sum()}")
    print(f"All not a number release dates from dataset2: {df2['release_date'].isna().sum()}")
```

```
    print(f"All not a number release dates from dataset4: {df4['Released'].isna().sum()}\n")
    df4.dropna(subset=['Released'], inplace=True)

    days, months = {}, {}
    df1_day, df1_month, df2_day, df2_month, df4_day, df4_month = {}, {}, {}, {}, {}, {}

    df1['rel_date'] = df1['rel_date'].astype(str)
    df2['release_date'] = df2['release_date'].astype(str)
    df4['Released'] = df4['Released'].astype(str)

    dict1 = dict(zip(df1['Title'], df1['rel_date']))
    dict2 = dict(zip(df2['Title'], df2['release_date']))
    dict3 = dict(zip(df4['Title'], df4['Released']))

    for elem in df4.values.tolist():
        df4_day.update({elem[0]: elem[2][:2]})
        df4_month.update({elem[0]: elem[2][3:6]})
    for elem in df1.values.tolist():
        df1_day.update({elem[0]: elem[2][8:]})
        df1_month.update({elem[0]: elem[2][5:7]})
    for elem in df2.values.tolist():
        df2_day.update({elem[2]: elem[0][8:]})
        df2_month.update({elem[2]: elem[0][5:7]})

    for title in masks[0].values:
        if dict1.get(title) is not None:
            days[title] = df1_day.get(title)
            months[title] = df1_month.get(title)
        elif dict2.get(title) is not None:
            days[title] = df2_day.get(title)
            months[title] = df2_month.get(title)
        elif dict3.get(title) is not None:
            days[title] = df4_day.get(title)
            months[title] = df4_month.get(title)
        else:
            days[title] = pd.NA
            months[title] = pd.NA

        days_series = pd.Series(days)
        months_series = pd.Series(months)
    print(f"The number of NA values in days: {days_series.isna().sum()}")
    print(f"The number of NA values in months: {months_series.isna().sum()}")
    return days_series, months_series
```

```
def add_necessary_columns(df, df2, df3, df4, masks):
    print(f"-------\nNecessary columns..\n-------")
    rated = add_certification(df, df4, masks)
    box_office = add_box_office(df, df2, df3, df4, masks)
    metascore = add_metascore(df, df4, masks)
    necessary_columns = pd.DataFrame({'Certification': rated, 'Box_Office':
box_office, 'MetaScore': metascore})
    return necessary_columns
def add_certification(df, df4, masks):
    print(f"---\nCertification..\n---")
    print(f"All not a number certifications from dataset4:
{df4['Certification'].isna().sum()}\n")

    dict1 = dict(zip(df4['Title'], df4['Certification']))
    rated = {}
    for title in masks[0].values:
        val = dict1.get(title)
        rated[title] = val if val is not None else pd.NA
        rated_series = pd.Series(rated)
    print(f"The number of NA values in rated series: {rated_series.isna().sum()}")
    return rated_series
def add_box_office(df, df2, df3, df4, masks):
    print(f"---\nBox Office..\n---")
    print(f"All not a number box offices from dataset2:
{df2['Box_Office'].isna().sum()}")
    print(f"All not a number box offices from dataset3:
{df3['Box_Office'].isna().sum()}")
    print(f"All not a number box offices from dataset4:
{df4['BoxOffice'].isna().sum()}\n")
    df3.dropna(subset=['Box_Office'], inplace=True)

    dict1 = dict(zip(df2['Title'], df2['Box_Office']))
    dict2 = dict(zip(df3['Title'], df3['Box_Office']))
    dict3 = dict(zip(df4['Title'], df4['BoxOffice']))
    box_office = {}

    for title in masks[0].values:
        val = dict1.get(title)
        if val is None:
            val = dict2.get(title)
        if val is None:
            val = dict3.get(title)
        box_office[title] = val if val is not None else pd.NA
        box_office_series = pd.Series(box_office)
```

```
    print(f"The number of NA values in box office series: {box_office_series.isna().sum()}")
    return box_office_series
def add_metascore(df, df4,masks):
    print(f"---\nMetascore..\n---")
    print(f"All not a number metascores from dataset4: {df4['MetaScore'].isna().sum()}\n")

    dict1 = dict(zip(df4['Title'], df4['MetaScore']))
    metascore = {}

    for title in masks[0].values:
        val = dict1.get(title)
        metascore[title] = val if val is not None else pd.NA
        metascore_series = pd.Series(metascore)
    print(f"The number of NA values in metascore series: {metascore_series.isna().sum()}")
    return metascore_series

def merge_dfs(df, additional_columns, necessary_columns, masks):
    print(f"-------\nConcatenating dataframes..\n-------")

    dict_replace = {'Jan': '01', 'Feb': '02', 'Mar': '03', 'Apr': '04', 'May': '05', 'Jun': '06', 'Jul': '07',
                    'Aug': '08', 'Sep': '09', 'Oct': '10', 'Nov': '11', 'Dec': '12'}
    print(f"Main df: \n{df.head(1)}\n")
    print(f"Additional df: \n{additional_columns.head(1)}\n")
    print(f"Necessary df: \n{necessary_columns.head(1)}\n")
    additional_columns.reset_index(names='Title', inplace=True)
    necessary_columns.reset_index(names='Title', inplace=True)

    print(f"Main dataframe title duplicates: {df.duplicated(subset='Title').sum()}")
    print(f"Additional dataframe title duplicates: {additional_columns.duplicated(subset='Title').sum()}")
    print(f"Necessary dataframe title duplicates: {necessary_columns.duplicated(subset='Title').sum()}\n")

    main_na = pd.Series(df.isna().sum())

    result_df = df.merge(additional_columns, on='Title', how='left')
    print(f"Merged df: \n{result_df.head(1)}\n")

    result_df.set_index('Title', inplace=True)
    necessary_columns.set_index('Title', inplace=True)
    result_df = result_df.combine_first(necessary_columns)
```

```
    result_df.reset_index(inplace=True)
    result_na = pd.Series(result_df.isna().sum())
    table = pd.DataFrame({'Main_df': main_na, 'Result_df': result_na})
    table = table.astype({'Result_df': float})
    table['Diffenence, %'] = 100 - (table['Result_df'] / table['Main_df'] * 100)
    table['Diffenence, %'] = table['Diffenence, %'].round(2)
    table['Diffenence, %'] = table['Diffenence, %'].fillna('-')
    table['Main_df'] = table['Main_df'].fillna('-')
    print(f"NA difference table: \n{table}\n")

    elem_was = result_df.shape[0]
    result_df.dropna(subset=['Box_Office', 'Certification', 'Country', 'Day',
'Language', 'MetaScore',
                             'Month'], inplace=True)
    for i, cn in enumerate(result_df['Country'].values):
        k = cn.find(',')
        if k != -1:
            country = cn[:k]
            if country == 'United States of America':
                country = 'United States'
            result_df.loc[i, 'Country'] = country

    result_df["Genre"] = result_df["Genre"].apply(safe_literal_eval)
    mlb = MultiLabelBinarizer()
    binarized_df = pd.DataFrame(
        mlb.fit_transform(result_df["Genre"]),
        columns=[c.strip() for c in mlb.classes_],
        index=result_df.index)
    print(f"Binarized genres dataframe: \n{binarized_df.head(1)}\n")

    result_df.drop(columns=['Genre', 'Stars'], inplace=True)
    final_df = pd.concat([result_df, binarized_df], axis=1)
    print(f"The amount of previous columns: {result_df.shape[1]}")
    print(f"The amount of new columns: {final_df.shape[1]}\n")
    print(f"Dataset with binarized columns: \n{final_df.head(1)}\n")

    final_df['Director'] = final_df['Director'].astype('string')
    columns = ['Title', 'Box_Office', 'Certification', 'Country', 'Day', 'Director',
'IMDB_Votes', 'IMDB_rating',
               'Language', 'MetaScore', 'Month', 'Runtime', 'Year']
    final_df.dropna(subset=columns, inplace=True)
    columns_to_int = ['Box_Office', 'Day', 'IMDB_Votes', 'MetaScore', 'Month',
'Runtime', 'Year']
    final_df = final_df[final_df['Box_Office'] != ""]
    final_df = final_df[final_df['Certification'] != "N/A"]
```

```
    final_df = final_df[final_df['MetaScore'] != "N/A"]
    final_df = final_df[final_df['Day'] != "N/"]
    final_df['Month'] = final_df['Month'].replace(dict_replace)
    final_df[columns_to_int] = final_df[columns_to_int].astype('int')
    regex_pattern = r"\[\'(.+)\'\]"
    final_df['Director'] = final_df['Director'].str.extract(regex_pattern)

    final_df['Language'] = final_df['Language'].astype(str).str.split(',').str[0]
    final_df['Language'] = final_df['Language'].replace('nan', pd.NA)
    final_df.replace({'United States of America': 'United States', 'USA': 'United
States'}, inplace=True)
    final_df['Country'] = final_df['Country'].astype(str).str.split(',').str[0]
    final_df['Country'] = final_df['Country'].replace('nan', pd.NA)
    final_df['Director'] = final_df['Director'].astype(str).str.split(',').str[0]
    final_df['Director'] = final_df['Director'].replace('nan', pd.NA)


    print(f"The number of final dataset elements: {elem_was}")
    elem_is = final_df.shape[0]
    percent = round((1 - elem_is / elem_was)*100, 2)
    print(f"The number of final dataset elements after all preprocessing: {elem_is}")
    print(f"The percent of the whole dataset that have been dropped: {percent}%\n")

    rating_bins = [0, 6, 7, 10]
    rating_labels = ['Bad', 'Average', 'Good']
    final_df['Rating_Class'] = pd.cut(final_df['IMDB_rating'], bins=rating_bins,
labels=rating_labels, include_lowest=True)
    print("\nRating class distribution:")
    print(final_df['Rating_Class'].value_counts())

    print(f"Final dataset: {final_df.head(1)}\n")

    to_processed.main(final_df)
    return 0

def safe_literal_eval(value):
    if pd.isna(value) or value in (None, 'nan'):
        return []
    try:
        evaluated = ast.literal_eval(value)
        if isinstance(evaluated, list):
            return [item.strip() for item in evaluated]
        return evaluated
    except (ValueError, SyntaxError):
        return []
```

```
if __name__ == '__main__':
    main()

to_processed.py:
from pathlib import Path
import sys
import time

import pandas as pd

def main(df):
    start = time.time()
    print('-------\nStarting saving processed data script...\n-------')
    save_path = Path('../data/processed')
    file_path = save_path / "processed_data.csv"
    try:
        if file_path.is_file():
            print(f"The processed dataset is already on your device..")
            while True:
                print(f"Do you want to save it? Y/n")
                input1 = input("Enter your answer:")
                if input1 == 'Y':
                    print(f"Preparing to save processed dataset..")
                    df.to_csv(file_path, index=False)
                    print("Saved successfully!")
                    break
                elif input1 == 'n':
                    print(f"No additional saving was needed..\n")
                    break
                else:
                    print(f"You should only choose between Y and n. Please try again..\n")
        else:
            print(f"Preparing to save processed dataset..")
            df.to_csv(file_path, index=False)
            print("Saved successfully!")
    except Exception as e:
        print(f"The problem has occurred: {e}\nClosing the program...")
        sys.exit(1)
    end = time.time()
    duration = end - start
    print(f"-------\nSaving processed data script duration: {duration:0.3f} seconds\n-
------")

if __name__ == "__main__":
```

```
    df = pd.DataFrame({
        'Title': ['Matrix', 'Titanic'],
        'Rating': [8.7, 7.8]
    })
    main(df)

upload_to_s3.py:
import boto3
import os
from botocore.exceptions import NoCredentialsError, ClientError,
InvalidRegionError
from dotenv import load_dotenv
from pathlib import Path
import time
import sys

load_dotenv()

def upload_file_to_s3(file_path, s3_file_name=None):
    bucket_name = os.getenv('S3_BUCKET_NAME')
    aws_access_key = os.getenv('AWS_ACCESS_KEY_ID')
    aws_secret_key = os.getenv('AWS_SECRET_ACCESS_KEY')
    region = os.getenv('AWS_REGION')

    if region == 'Your_aws_region' or not region:
        print("Error: You haven't configured the aws_region in .env file yet.")
        return False
    elif bucket_name == 'You_bucket_name' or not bucket_name:
        print("Error: You haven't configured the bucket_name in .env file yet.")
        return False
    elif aws_access_key == 'Your_access_key' or not bucket_name:
        print("Error: You haven't configured the aws_access_key in .env file yet.")
        return False
    elif aws_secret_key == 'Your_secret_access_key' or not bucket_name:
        print("Error: You haven't configured the aws_secret_key in .env file yet.")
        return False
    if s3_file_name is None:
        s3_file_name = os.path.basename(file_path)
    print(f"Uploading '{file_path}' in the bucket '{bucket_name}'...")

    try:
        s3_client = boto3.client(
            's3',
            aws_access_key_id=aws_access_key,
            aws_secret_access_key=aws_secret_key,
```

```
            region_name=region
        )
        s3_client.upload_file(file_path, bucket_name, s3_file_name)
        print(f"File was successfully uploaded to AWS. File name: {s3_file_name}")
        return True

    except FileNotFoundError:
        print(f"Error: File {file_path} not found.")
        return False
    except NoCredentialsError:
        print("Error: Wrong AWS access keys.")
        return False
    except ClientError as e:
        print(f"Error: {e}")
        return False
    except InvalidRegionError:
        print(f"Error: invalid aws region format: {region}. Please check your .env
file.")
        return False
    except Exception as e:
        print(f"Unexpected error: {e}")


def main():
    start = time.time()
    print(f"-------\nStarting AWS S3 script...\n-------")
    path = Path('../data/processed') / 'processed_data.csv'
    if path.is_file():
        print(f"File processed_data.csv exists, preparing to upload...")
        upload_file_to_s3(path, None)
    else:
        print(f"Error: file processed_data.csv doesn't exist. Stopping the program..")
        sys.exit(1)
    end = time.time()
    duration = end - start
    print(f"\nAWS S3 script duration: {duration:0.3f} seconds")

if __name__ == "__main__":
    main()
```

trends.py:

```
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib
import time
```

```
import sys
import matplotlib.ticker as ticker
from pathlib import Path
import numpy as np

def for_figs(df):
    figs = []
    dict_replace = {1: 'Jan', 2: 'Feb', 3: 'Mar', 4: 'Apr', 5: 'May', 6: 'Jun', 7: 'Jul', 8: 'Aug',
                    9: 'Sep', 10: 'Oct', 11: 'Nov', 12: 'Dec'}
    month_order = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']
    fig4_colors = ["#e4a5ff", "#deabff", "#d8b1ff", "#d1b7ff", "#cbbdff", "#c5c4ff", "#bfcaff", "#b8d0ff", "#b2d6ff",
                   "#acdcff"]

    # -----------First slide
    fig, ax_array = plt.subplots(2, 1, figsize=(16, 10))  # кількість по роках
    plt.subplots_adjust(hspace=0.4)
    df_years = df['Year'].value_counts()
    df_years_sorted = df_years.sort_index()
    year_bins = [1919, 1930, 1940, 1950, 1960, 1970, 1980, 1990, 2000, 2010, 2023]
    year_labels = ['1920s', '1930s', '1940s', '1950s', '1960s', '1970s', '1980s',
                   '1990s', '2000s', '2010-2023']
    df_year_bins = pd.cut(df_years_sorted.index, bins=year_bins, labels=year_labels, include_lowest=True)
    df_years_by_bin = df_years_sorted.groupby(df_year_bins).sum()
    bin_labels = [str(interval) for interval in df_years_by_bin.index]
    bar_heights = df_years_by_bin.values
    bars1 = ax_array[0].bar(bin_labels, bar_heights, color='#476591')
    ax_array[0].bar_label(bars1, fmt='%d', padding=.5, fontsize=10)
    ax_array[0].set_ylabel('The amount of films', fontsize=14)
    ax_array[0].set_title('Films by years', fontsize=16, fontweight='bold')
    ax_array[0].set_xlabel('Year', fontsize=14)

    df_countries = df['Country'].value_counts()  # кількість по топ 10 країнах
    bars2 = ax_array[1].barh(df_countries.index.tolist()[:10], df_countries.values.tolist()[:10], color='#476591')
    ax_array[1].bar_label(bars2, fmt='%d', padding=.5, fontsize=10)
    ax_array[1].set_ylabel('The amount of films', fontsize=14)
    ax_array[1].set_title('Films by top-10 countries', fontsize=16, fontweight='bold')
    ax_array[1].set_xlabel('Country', fontsize=14)
    fig1 = fig
    figs.append(fig1)
    plt.close(fig1)
```

```
    # -----------Second slide
    fig, ax_array = plt.subplots(2, 1, figsize=(16, 10))
    plt.subplots_adjust(hspace=0.4)
    df['Month'] = df['Month'].replace(dict_replace)  # кількість по місяцях
    df_months = df['Month'].value_counts()
    df_months = df_months.reindex(month_order)
    bars3 = ax_array[0].bar(df_months.index, df_months.values, color=fig4_colors)
    ax_array[0].bar_label(bars3, fmt='%d', padding=.5, fontsize=10)
    ax_array[0].set_ylabel('The amount of films', fontsize=14)
    ax_array[0].set_title('Films by month', fontsize=16, fontweight='bold')
    ax_array[0].set_xlabel('Month', fontsize=14)

    df_days = df['Day'].value_counts()  # кількість по днях
    df_days = df_days.sort_index()
    bars4 = ax_array[1].bar(df_days.index, df_days.values, color=fig4_colors)
    ax_array[1].bar_label(bars4, fmt='%d', padding=.5, fontsize=10)
    ax_array[1].set_ylabel('The amount of films', fontsize=14)
    ax_array[1].set_title('Films by day', fontsize=16, fontweight='bold')
    ax_array[1].set_xlabel('Day', fontsize=14)
    fig2 = fig
    figs.append(fig2)
    plt.close(fig2)

    # -----------Third slide
    fig, ax_array = plt.subplots(2, 1, figsize=(16, 10))
    plt.subplots_adjust(hspace=0.4)
    df_directors = df['Director'].value_counts()  # кількість по топ 10 режисерах
    bars5 = ax_array[0].barh(df_directors.index.tolist()[:10],
df_directors.values.tolist()[:10], color=fig4_colors)
    ax_array[0].bar_label(bars5, fmt='%d', padding=.5, fontsize=10)
    ax_array[0].set_ylabel('The amount of films', fontsize=14)
    ax_array[0].set_title('Films by top-10 directors', fontsize=16, fontweight='bold')
    ax_array[0].set_xlabel('Director', fontsize=14)

    genres = ['Action', 'Adventure', 'Animation', 'Biography', 'Comedy', 'Crime',
'Drama', 'Family', 'Fantasy',
              'Film-Noir', 'History', 'Horror', 'Music', 'Musical', 'Mystery', 'Romance',
'Sci-Fi', 'Sport',
              'Thriller', 'War', 'Western']
    df_genres = df[genres].sum().sort_values(ascending=False)  # кількість по
жанрах
    bars6 = ax_array[1].barh(df_genres.index, df_genres.values, color=fig4_colors)
    ax_array[1].bar_label(bars6, fmt='%d', padding=.5, fontsize=10)
    ax_array[1].set_ylabel('The amount of films', fontsize=14)
```

```
    ax_array[1].set_title('Films by genre', fontsize=16, fontweight='bold')
    ax_array[1].set_xlabel('Genre', fontsize=14)
    fig3 = fig
    figs.append(fig3)
    plt.close(fig3)

    # -----------Forth slide
    fig, ax_array = plt.subplots(2, 1, figsize=(16, 10))
    plt.subplots_adjust(hspace=0.4)
    df_rated = df['Certification'].value_counts()  # кількість по сертифікату
    bars7 = ax_array[0].barh(df_rated.index, df_rated.values, color=fig4_colors)
    ax_array[0].bar_label(bars7, fmt='%d', padding=.5, fontsize=10)
    ax_array[0].set_ylabel('The amount of films', fontsize=14)
    ax_array[0].set_title('Films by certification', fontsize=16, fontweight='bold')
    ax_array[0].set_xlabel('Certification', fontsize=14)

    df_runtime = df['Runtime'].value_counts()  # кількість по тривалості
    bars8 = ax_array[1].bar(df_runtime.index, df_runtime.values, color=fig4_colors)
    ax_array[1].xaxis.set_major_locator(ticker.MultipleLocator(25))
    ax_array[1].tick_params(axis='x')
    ax_array[1].set_ylabel('The amount of films', fontsize=14)
    ax_array[1].set_title('Films by runtime', fontsize=16, fontweight='bold')
    ax_array[1].set_xlabel('Runtime', fontsize=14)
    fig4 = fig
    figs.append(fig4)
    plt.close(fig4)

    # -----------Fifth slide
    fig, ax_array = plt.subplots(2, 1, figsize=(16, 10))
    plt.subplots_adjust(hspace=0.4)

    df['Rank_Box'] = df['Box_Office'].rank(ascending=False, method='min')
    df['Rank_Rating'] = df['IMDB_rating'].rank(ascending=False, method='min')
    top_10_box = df.nsmallest(10, 'Rank_Box').sort_values('Rank_Box',
ascending=True)
    top_10_box = top_10_box.iloc[::-1]
    y_indexes_box = np.arange(len(top_10_box))

    bars9 = ax_array[0].barh(y_indexes_box - 0.2, top_10_box['Rank_Box'],
color='#2FA9DE', label='Box Office Rank',
                             align='center', height=0.4)
    ax_array[0].set_xlabel('Box Office Top 10 Ranks', color='#2FA9DE',
fontsize=14)
    ax_array[0].tick_params(axis='x', labelcolor='#2FA9DE')
    ax_array[0].set_yticks(y_indexes_box)
```

```
    ax_array[0].set_yticklabels(top_10_box['Title'], fontsize=12)
    ax_array[0].bar_label(bars9, fmt='%d', padding=1, color='black')
    ax2 = ax_array[0].twiny()
    bars2 = ax2.barh(y_indexes_box + 0.2, top_10_box['Rank_Rating'], color='#E04C6B', label='IMDB Rating Rank',
                     align='center', height=0.4)
    ax2.tick_params(axis='x', labelcolor='#E04C6B')
    ax2.set_xlabel('IMDB Rating Rank (Short bar = Good, Long bar = Bad)', color='#E04C6B', fontsize=14)
    ax2.bar_label(bars2, fmt='%d', padding=1, color='#E04C6B')
    plt.title('Rank Comparison: Box Office (Blue) vs Audience Rating (Red)', fontsize=16, fontweight='bold')
    plt.tight_layout()

    top_10_rating = df.nsmallest(10, 'Rank_Rating').sort_values('Rank_Rating', ascending=True)
    top_10_rating = top_10_rating.iloc[::-1]
    y_indexes_rating = np.arange(len(top_10_rating))

    bars10 = ax_array[1].barh(y_indexes_rating - 0.2, top_10_rating['Rank_Rating'], color='#E04C6B',
                              label='Rating Rank',
                              align='center', height=0.4)
    ax_array[1].set_xlabel('IMDB Rating Rank (Short bar = Good, Long bar = Bad)', color='#E04C6B', fontsize=14)
    ax_array[1].tick_params(axis='x', labelcolor='#E04C6B')
    ax_array[1].set_yticks(y_indexes_rating)
    ax_array[1].set_yticklabels(top_10_rating['Title'], fontsize=12)
    ax_array[1].bar_label(bars10, fmt='%d', padding=1, color='black')
    ax3 = ax_array[1].twiny()
    bars2 = ax3.barh(y_indexes_rating + 0.2, top_10_rating['Rank_Box'], color='#2FA9DE', label='Box Office Rank',
                     align='center', height=0.4)
    ax3.tick_params(axis='x', labelcolor='#2FA9DE')
    ax3.set_xlabel('Box Office Top 10 Ranks', color='#2FA9DE', fontsize=14)
    ax3.bar_label(bars2, fmt='%d', padding=1, color='#2FA9DE')
    plt.title('Box office Comparison: Audience Rating (Red) vs Box Office (Blue)', fontsize=16, fontweight='bold')
    plt.tight_layout()
    fig5 = fig
    figs.append(fig5)
    plt.close(fig5)
    return figs

def main():
```

```
    start = time.time()
    matplotlib.use('Qt5Agg')
    print(f"-------\nStarting trends script...\n-------")
    path = Path('../data/processed')
    file_path = path / 'processed_data.csv'
    if file_path.is_file():
        print(f"File processed_data.csv exists, preparing to build trends...")
        df = pd.read_csv(file_path)
        figs = for_figs(df)
        for fig in figs:
            plt.figure(fig)
            plt.show()
    else:
        print(f"Error: file processed_data.csv doesn't exist. Stopping the program...")
        sys.exit(1)

    end = time.time()
    duration = end - start
    print(f"-------\nTrends script duration: {duration:0.3f} seconds\n-------")


if __name__ == '__main__':
    main()
```

to_streamlit.py:

```
from pathlib import Path
import time
import sys

import streamlit as st
import pandas as pd
import altair as alt

def for_figs(df):
    dict_replace = {1: 'Jan', 2: 'Feb', 3: 'Mar', 4: 'Apr', 5: 'May', 6: 'Jun', 7: 'Jul', 8:
'Aug',
                    9: 'Sep', 10: 'Oct', 11: 'Nov', 12: 'Dec'}
    month_order = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov',
'Dec']

    st.header('Film industry data and insights.')
    st.write('<br><br>', unsafe_allow_html=True)
    col1, col2 = st.columns(2, gap='large')

    df_years = df['Year'].value_counts()                # кількість по роках
```

```
df_years_sorted = df_years.sort_index()
year_bins = [1919, 1930, 1940, 1950, 1960, 1970, 1980, 1990, 2000, 2010, 2023]
year_labels = ['1920s', '1930s', '1940s', '1950s', '1960s', '1970s', '1980s',
               '1990s', '2000s', '2010-2023']
df_year_bins = pd.cut(df_years_sorted.index, bins=year_bins, labels=year_labels, include_lowest=True)
df_years_by_bin = df_years_sorted.groupby(df_year_bins).sum()
df_years_by_bin_ordered = df_years_by_bin.reindex(year_labels).fillna(0)
df1 = pd.DataFrame({'Year': df_years_by_bin_ordered.index, 'Film': df_years_by_bin_ordered.values})
chart = alt.Chart(df1).mark_line(
    point=True,
    color='blue',
    strokeWidth=3
).encode(
    x=alt.X('Year', sort=year_labels, title='Decade', axis=alt.Axis(labelAngle=-90)),
    y=alt.Y('Film', title='Film'),
    tooltip=['Year', 'Film']
).properties(
    title='The amount of films by decades'
).interactive()

df_countries = df['Country'].value_counts()                    #ількість по країнах
df2 = pd.DataFrame({'Country': df_countries.index.tolist()[:10], 'Country_Films': df_countries.values.tolist()[:10]})
chart2 = alt.Chart(df2).mark_bar(
    color='blue'
).encode(
    x=alt.X('Country', title='Country', sort='y'),
    y=alt.Y('Country_Films', title='Film'),
    tooltip=['Country', 'Country_Films']
).properties(
    title='The amount of films by countries'
).interactive()


df['Month'] = df['Month'].replace(dict_replace)                # кількість по місяцях
df_months = df['Month'].value_counts()
df_months = df_months.reindex(month_order)
df3 = pd.DataFrame({'Month': df_months.index, 'Film': df_months.values})
chart3 = alt.Chart(df3).mark_bar(
    color='red'
).encode(
    x = alt.X('Month', title='Month', sort='y'),
```

```
        y = alt.Y('Film', title='Film'),
        tooltip=['Month', 'Film']
    ).properties(
        title='The amount of films by months'
    ).interactive()

    df_days = df['Day'].value_counts()                    # кількість по днях
    df_days = df_days.sort_index()
    df4 = pd.DataFrame({'Day': df_days.index, 'Film': df_days.values})
    chart4 = alt.Chart(df4).mark_bar(
        color='red'
    ).encode(
        # x=alt.X('Day', title='Day', sort='y'),
        x=alt.X('Day:Q', scale=alt.Scale(domain=[0, 32], nice=False)),
        y=alt.Y('Film', title='Film'),
        tooltip=['Day', 'Film']
    ).properties(
        title='The amount of films by days'
    ).interactive()


    df_directors = df['Director'].value_counts()          # кількість по топ 10
режисерах
    df2['Director'] = df_directors.index.tolist()[:10]
    df2['Director_Films'] = df_directors.values.tolist()[:10]
    chart5 = alt.Chart(df2).mark_bar(
        color='green'
    ).encode(
        x=alt.X('Director', title='Director', sort='y'),
        y=alt.Y('Director_Films', title='Film'),
        tooltip=['Director', 'Director_Films']
    ).properties(
        title='The amount of films by directors'
    ).interactive()

    genres = ['Action', 'Adventure', 'Animation', 'Biography', 'Comedy', 'Crime',
'Drama', 'Family', 'Fantasy',
              'Film-Noir', 'History', 'Horror', 'Music', 'Musical', 'Mystery', 'Romance',
'Sci-Fi', 'Sport',
              'Thriller', 'War', 'Western']
    df_genres = df[genres].sum().sort_values(ascending=False)          # кількість
по жанрах
    df5 = pd.DataFrame({'Genre': df_genres.index, 'Film': df_genres.values})
    chart6 = alt.Chart(df5).mark_bar(
        color='green'
```

```
    ).encode(
        x=alt.X('Genre', title='Genre', sort='y'),
        y=alt.Y('Film', title='Film'),
        tooltip=['Genre', 'Film']
    ).properties(
        title='The amount of films by genres'
    ).interactive()


    df_rated = df['Certification'].value_counts()              # кількість по сертифікату
    df6 = pd.DataFrame({'Certification': df_rated.index, 'Film': df_rated.values})
    chart7 = alt.Chart(df6).mark_bar(
        color='yellow'
    ).encode(
        x=alt.X('Certification', title='Certification', sort='y'),
        y=alt.Y('Film', title='Film'),
        tooltip=['Certification', 'Film']
    ).properties(
        title='The amount of films by certifications'
    ).interactive()

    df_runtime = df['Runtime'].value_counts()                  # кількість по тривалості
    df7 = pd.DataFrame({'Runtime': df_runtime.index, 'Film': df_runtime.values})
    chart8 = alt.Chart(df7).mark_line(
        color='yellow'
    ).encode(
        x=alt.X('Runtime', title='Runtime', sort='y'),
        y=alt.Y('Film', title='Film'),
        tooltip=['Runtime', 'Film']
    ).properties(
        title='The amount of films by runtime'
    ).interactive()


    df['Rank_Box'] = df['Box_Office'].rank(ascending=False, method='min')
    df['Rank_Rating'] = df['IMDB_rating'].rank(ascending=False, method='min')
    top_10_box = df.nsmallest(10, 'Rank_Box').sort_values('Rank_Box',
ascending=True)
    top_10_box = top_10_box.iloc[::-1]

    base = alt.Chart(top_10_box).encode(
        y=alt.Y('Title', sort=alt.EncodingSortField(field="Rank_Box",
order='ascending'), title=None)
    )
```

```
chart_box = base.mark_bar(height=10, opacity=0.8, color='#2FA9DE', yOffset=-3).encode(
    x=alt.X('Rank_Box', title='Box Office Rank (1-10)'),
    tooltip=['Title', 'Rank_Box']
)

chart_rating = base.mark_bar(height=10, opacity=0.8, color='#E04C6B', yOffset=3).encode(
    x=alt.X('Rank_Rating', title='IMDB Rating Rank'),
    tooltip=['Title', 'Rank_Rating']
)

chart9 = (chart_box + chart_rating).resolve_scale(
    x='independent'
).properties(
    title='Top 10 Films by box office: Box Office (Blue) vs Audience Rating (Red)',
    width=600
).configure_axis(
    labelFontSize=12,
    titleFontSize=14
).interactive()


top_10_rating = df.nsmallest(10, 'Rank_Rating').sort_values('Rank_Rating', ascending=True)
top_10_rating = top_10_rating.iloc[::-1]

base2 = alt.Chart(top_10_rating).encode(
    y=alt.Y('Title', sort=alt.EncodingSortField(field="Rank_Rating", order='ascending'), title=None)
)

chart_box2 = base2.mark_bar(height=10, opacity=0.8, color='#2FA9DE', yOffset=-3).encode(
    x=alt.X('Rank_Box', title='Box Office Rank (1-10)'),
    tooltip=['Title', 'Rank_Box']
)

chart_rating2 = base2.mark_bar(height=10, opacity=0.8, color='#E04C6B', yOffset=3).encode(
    x=alt.X('Rank_Rating', title='IMDB Rating Rank'),
    tooltip=['Title', 'Rank_Rating']
)
```

```
    chart10 = (chart_box2 + chart_rating2).resolve_scale(
        x='independent'
    ).properties(
        title='Top 10 Films by rating: Box Office (Blue) vs Audience Rating (Red)',
        width=600
    ).configure_axis(
        labelFontSize=12,
        titleFontSize=14
    ).interactive()

    with col1:
        st.altair_chart(chart, use_container_width=True)
        st.altair_chart(chart3, use_container_width=True)
        st.altair_chart(chart5, use_container_width=True)
        st.altair_chart(chart7, use_container_width=True)

    with col2:
        st.altair_chart(chart2, use_container_width=True)
        st.altair_chart(chart4, use_container_width=True)
        st.altair_chart(chart6, use_container_width=True)
        st.altair_chart(chart8, use_container_width=True)

    st.altair_chart(chart9, use_container_width=True)
    st.altair_chart(chart10, use_container_width=True)

def main():
    start = time.time()
    print('-------\nStarting streamlit script...\n-------')

    st.set_page_config(layout='wide')
    path = Path('../data/processed')
    file_path = path / 'processed_data.csv'
    if file_path.is_file():
        print(f"File processed_data.csv exists, preparing to upload...")
        df = pd.read_csv(file_path)
        for_figs(df)
    else:
        print(f"Error: file processed_data.csv doesn't exist. Stopping the program...")
        sys.exit(1)

    end = time.time()
    duration = end - start
    print(f"-------\nStreamlit script duration: {duration:0.3f} seconds\n-------")
```

```
if __name__ == '__main__':
    main()
```

models.py:

```
import time
import sys
from pathlib import Path
import os
import warnings

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib
import seaborn as sns

from sklearn.ensemble import RandomForestRegressor, RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, accuracy_score,
mean_absolute_error, r2_score, zero_one_loss, \
    precision_score, recall_score, f1_score, confusion_matrix, mean_squared_error,
mean_absolute_percentage_error, \
    median_absolute_error

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Input, BatchNormalization


os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
warnings.filterwarnings("ignore", category=FutureWarning)

def prepare_data(df, target_col, purpose, leakage_cols=None):
    drop_list = [target_col]
    if leakage_cols:
        drop_list.extend(leakage_cols)

    if target_col == 'Box_Office':
        df['IMDB_Votes'] = np.log1p(df['IMDB_Votes'])
        df['Runtime'] = np.log1p(df['Runtime'])

    X = df.drop(columns=drop_list)
    y = df[target_col]
```

```
    if purpose == 'classification':
        X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, test_size=0.2,
random_state=42, shuffle=True)
        X_train.drop(columns=['Western', 'War', 'Musical', 'Film-Noir'], inplace=True)
        X_test.drop(columns=['Western', 'War', 'Musical', 'Film-Noir'], inplace=True)
        y_train.drop(columns=['Western', 'War', 'Musical', 'Film-Noir'], inplace=True)
        y_test.drop(columns=['Western', 'War', 'Musical', 'Film-Noir'], inplace=True)
    else:
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42, shuffle=True)

    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    print(f"Train dataset shape: {X_train_scaled.shape}")
    print(f"Test dataset shape: {X_test_scaled.shape}")
    return X_train_scaled, X_test_scaled, y_train, y_test

def predict_rating_class_ml(df):        #rating_class, classification, ml
    print(f"-------\nClassification: Rating Class(Random Forest)...\n-------")

    X_train, X_test, y_train, y_test = prepare_data(df, 'Rating_Class', 'classification',
leakage_cols=['IMDB_rating'])
    clf = RandomForestClassifier(n_estimators=35, random_state=42,
class_weight='balanced')
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)

    print("\n---ML classification results:---")
    df_metrics = pd.DataFrame()
    df_metrics['Metrics'] = ['Accuracy', 'Missclassification', 'Precision', 'Recall', 'F1-
score']
    df_metrics['Rating ML'] = ['{0:.4f}'.format(accuracy_score(y_test, y_pred)),
                               '{0:.4f}'.format(zero_one_loss(y_test, y_pred)),
                               '{0:.4f}'.format(precision_score(y_test, y_pred,
average='weighted', zero_division=1)),
                               '{0:.4f}'.format(recall_score(y_test, y_pred,
average='weighted', zero_division=1)),
                               '{0:.4f}'.format(f1_score(y_test, y_pred, average='weighted'))]
    print(f"{df_metrics}\n")
    print(classification_report(y_test, y_pred))

def predict_rating_class_nn(df):            #rating_class, classification, nn
    print(f"-------\nClassification: Rating Class(Neural Network)...\n-------")
```

```
    X_train, X_test, y_train, y_test = prepare_data(df, 'Rating_Class', 'classification',
leakage_cols=['IMDB_rating'])

    num_classes = df['Rating_Class'].nunique()
    model = Sequential([
        Input(shape=(X_train.shape[1],)),
        Dense(128, activation='relu'),
        BatchNormalization(),
        Dropout(0.4),
        Dense(64, activation='relu'),
        BatchNormalization(),
        Dropout(0.3),
        Dense(32, activation='relu'),
        Dense(num_classes, activation='softmax')
    ])
    opt = tf.keras.optimizers.Adam(learning_rate=0.001)
    model.compile(optimizer=opt,              loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

    history = model.fit(X_train, y_train, epochs=40, batch_size=8,
validation_split=0.2, verbose=1)

    y_pred_probs = model.predict(X_test)
    y_pred = np.argmax(y_pred_probs, axis=1)

    print("\nMetrics report:")
    print(classification_report(y_test, y_pred, zero_division=0))

    cm = confusion_matrix(y_test, y_pred)

    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
    plt.xlabel('Predicted class')
    plt.ylabel('Actual class')
    plt.title('Classification: Confusion Matrix for Rating Classes')
    plt.show()

    loss, acc = model.evaluate(X_test, y_test, verbose=0)
    print("\n---NN classification results:---")
    print(f"Accuracy: {acc:.4f}")

def predict_box_office_ml(df):              #box_office, regression, ml
    print(f"-------\nRegression: Box Office(Random Forest)...\n-------")

    X_train, X_test, y_train, y_test = prepare_data(df, 'Box_Office', 'regression')
```

```
    regr = RandomForestRegressor(n_estimators=50, random_state=42)
    regr.fit(X_train, y_train)
    y_pred_log = regr.predict(X_test)

    print("\n---ML Regression Results:---")
    print(f"MAE (Mean Average Error in $): {mean_absolute_error(y_test, y_pred_log):,.0f}")
    print(f"R2 Score: {r2_score(y_test, y_pred_log):.4f}")

def predict_box_office_nn(df):              #box_office, regression, nn
    print(f"-------\nRegression: Box Office(Neural Network)...\n-------")

    X_train, X_test, y_train, y_test = prepare_data(df, 'Box_Office', 'regression')

    y_train = np.log1p(y_train)
    y_test = np.log1p(y_test)

    model = Sequential([
        Input(shape=(X_train.shape[1],)),
        Dense(128, activation='relu'),
        BatchNormalization(),
        Dropout(0.1),
        Dense(64, activation='relu'),
        BatchNormalization(),
        Dropout(0.1),
        Dense(32, activation='relu'),
        BatchNormalization(),
        Dense(16, activation='relu'),
        Dense(1, activation='linear')
    ])

    opt = tf.keras.optimizers.Adam(learning_rate=0.001)
    model.compile(optimizer=opt, loss='mean_absolute_error')
    history = model.fit(X_train, y_train, epochs=100, batch_size=32, validation_split=0.2, verbose=1)

    y_pred_log = model.predict(X_test).flatten()

    y_test_real = np.expm1(y_test)
    y_pred_real = np.expm1(y_pred_log)

    print("\n---NN Regression Results---")
    print(f"MAE (Mean Average Error in $): {mean_absolute_error(y_test_real, y_pred_real):,.0f}")
```

```
    r2 = r2_score(y_test_real, y_pred_real)
    print(f"R2 Score: {r2:.4f}")

def main():
    start = time.time()
    matplotlib.use('TkAgg')
    print(f"-------\nStarting models script...\n-------")
    pd.set_option('display.max_rows', 3000)
    pd.set_option('display.max_columns', 50)
    pd.set_option('display.width', 5000)
    path = Path('../data/processed')
    file_path = path / 'processed_data.csv'
    try:
        if file_path.is_file():
            print(f"Dataset was successfully found.")
            df = pd.read_csv(file_path)
        else:
            print(f"Failed to found dataset.")
            sys.exit(1)
    except Exception as e:
        print(f"Error: {e}")

    df = prepare_dataset(df)
    feature_importance(df)

    predict_rating_class_ml(df)
    predict_rating_class_nn(df)

    predict_box_office_ml(df)
    predict_box_office_nn(df)

    end = time.time()
    duration = end - start
    print(f"\nModels script duration: {duration:0.3f} seconds")

def prepare_dataset(df):
    df.drop(columns=['Title'], inplace=True)
    compress_column(df, 'Director', 70, 'Another')
    compress_column(df, 'Country', 15, 'Another')
    compress_column(df, 'Language', 10, 'Another')

    cols_to_encode = ['Director', 'Country', 'Language', 'Certification', 'Rating_Class']
    for col in cols_to_encode:
        df[col] = pd.factorize(df[col])[0]
    return df
```

```
def feature_importance(df):
    print(f"---\nFeature importances..\n---")
    print(f"Example:\n{df.head(1)}\n")

    x_box_office = df.drop(columns=['Box_Office'])
    y_box_office = df['Box_Office']
    x_rating = df.drop(columns=['IMDB_rating', 'Rating_Class'])
    y_rating = df['Rating_Class']
    x_box_train, x_box_test, y_box_train, y_box_test = train_test_split(x_box_office, y_box_office, test_size=0.2,
                                                        shuffle=True,random_state=45)
    x_rating_train, x_rating_test, y_rating_train, y_rating_test = train_test_split(x_rating, y_rating, test_size=0.2,
                                                            shuffle=True,
random_state=45)

    cols_to_scale = ['Year', 'Runtime', 'MetaScore', 'IMDB_Votes', 'IMDB_rating']
    scaler = StandardScaler()
    scaler.fit(x_box_train[cols_to_scale])
    x_box_train.loc[:, cols_to_scale] = scaler.transform(x_box_train[cols_to_scale])
    x_box_test.loc[:, cols_to_scale] = scaler.transform(x_box_test[cols_to_scale])

    forest_box = RandomForestRegressor(n_estimators=50)
    y_box_train_log = np.log1p(y_box_train)
    forest_box.fit(x_box_train, y_box_train_log)
    importances = forest_box.feature_importances_
    feat_importances = pd.Series(importances, index=x_box_train.columns)
    plt.figure(figsize=(16, 10))
    feat_importances.plot(kind='barh', color='#48BD83')
    plt.title("Feature Importances for Box Office")
    plt.xlabel("Importance")
    plt.ylabel("Column")
    plt.show()

    forest_box2 = RandomForestClassifier(n_estimators=50)
    forest_box2.fit(x_rating_train, y_rating_train)
    importances2 = forest_box2.feature_importances_
    feat_importances2 = pd.Series(importances2, index=x_rating_train.columns)
    plt.figure(figsize=(16, 10))
    feat_importances2.plot(kind='barh', color='#4486B8')
    plt.title("Feature Importances for Rating")
    plt.xlabel("Importance")
    plt.ylabel("Column")
    plt.show()
```

```
def compress_column(df, column_name, top_n, other_label='Another'):
    top_items = df[column_name].value_counts().nlargest(top_n).index
    df[column_name] = df[column_name].where(df[column_name].isin(top_items),
other=other_label)


if __name__ == '__main__':
    main()
```

ДОДАТОК В

Слайди презентації

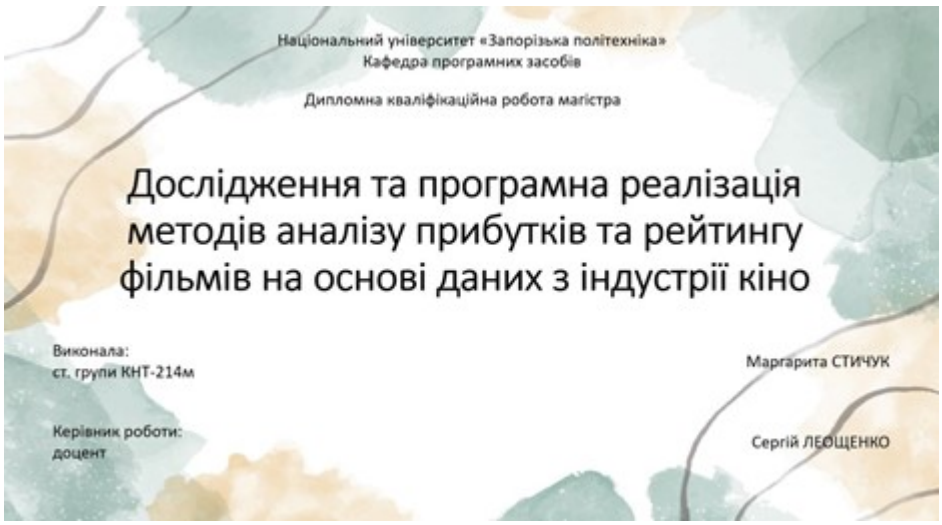


Рисунок В.1 – Слайд 1

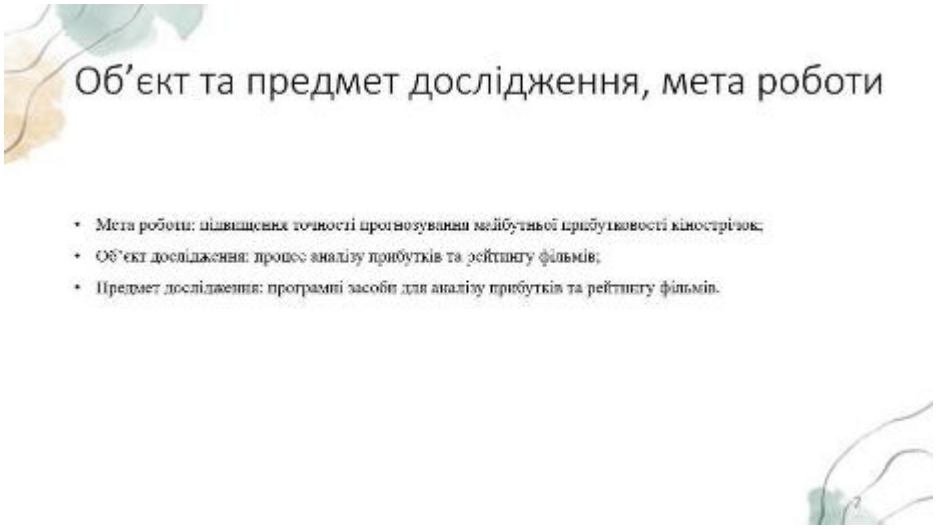


Рисунок В.2 – Слайд 2

Рисунок В.3 – Слайд 3

Рисунок В.4 – Слайд 4

Рисунок В.5 – Слайд 5

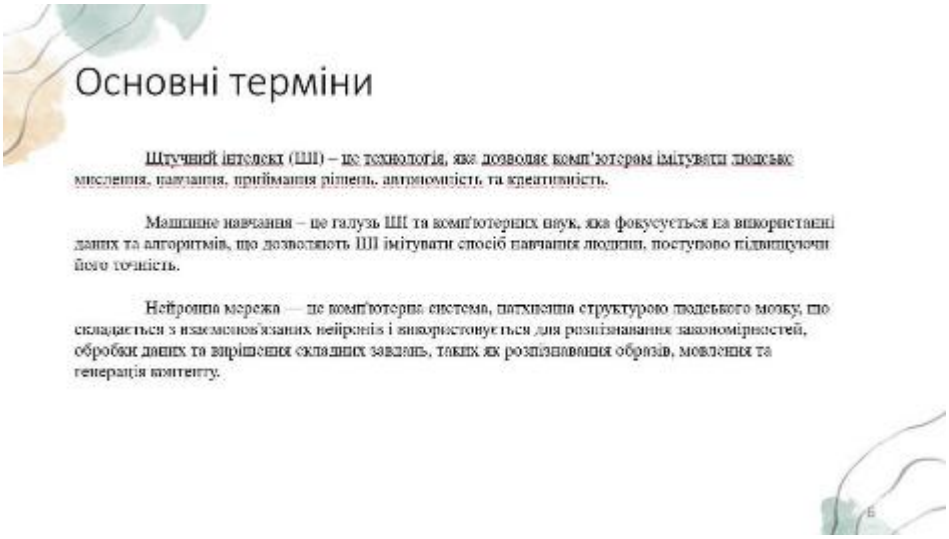


Рисунок В.6 – Слайд 6

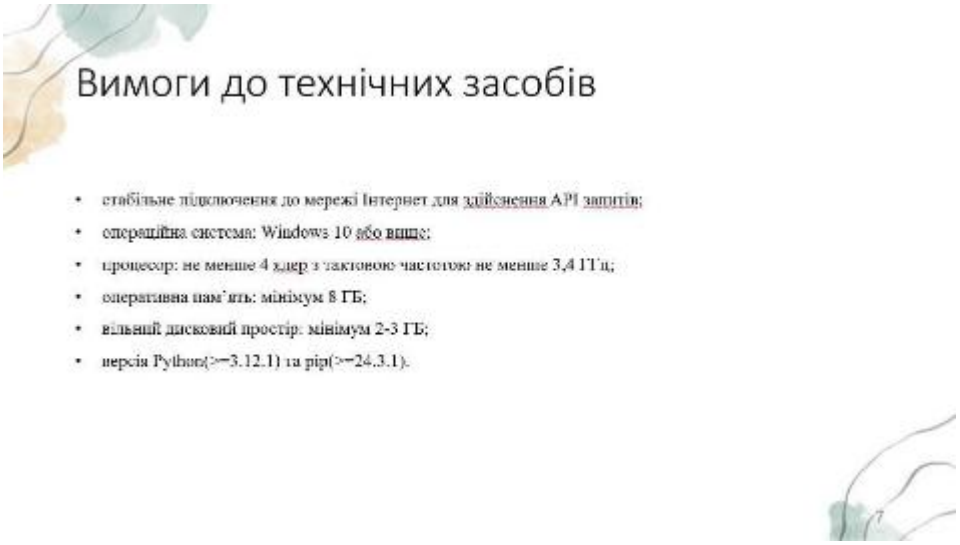


Рисунок В.7 – Слайд 7

Рисунок В.8 – Слайд 8

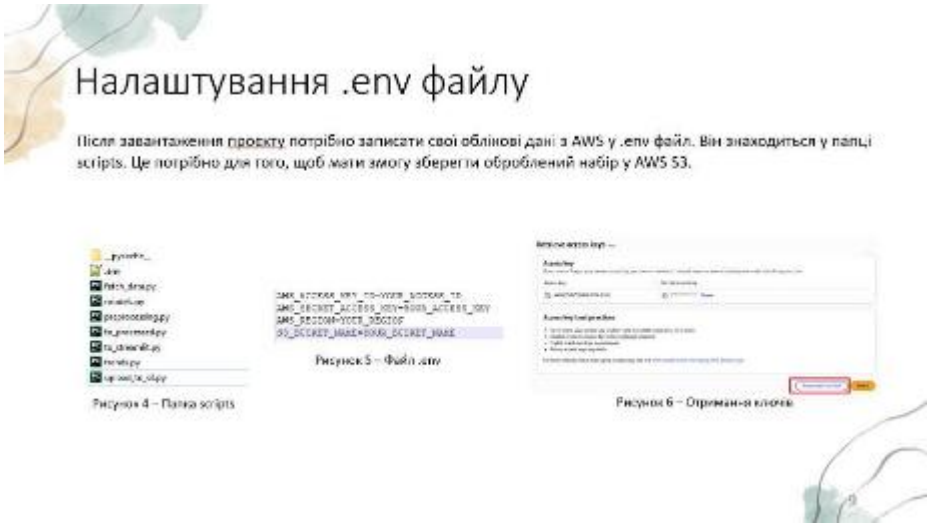


Рисунок В.9 – Слайд 9

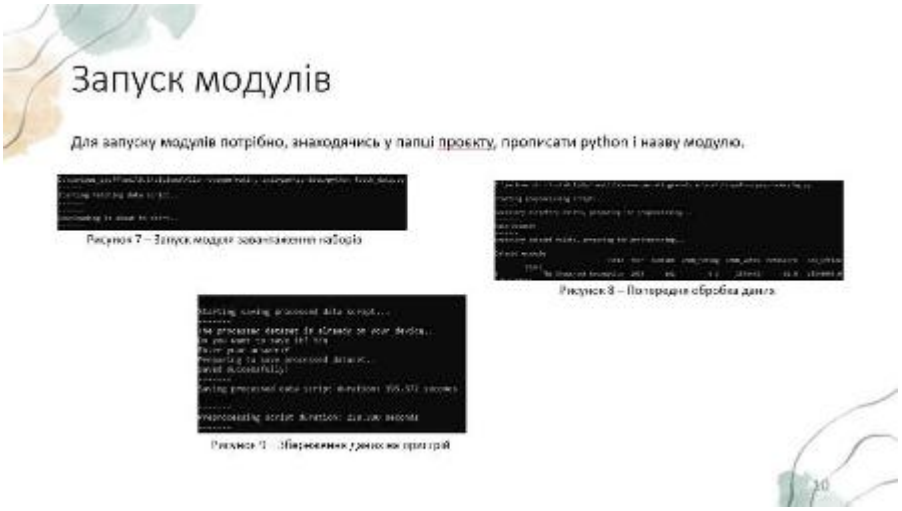


Рисунок В.10 – Слайд 10

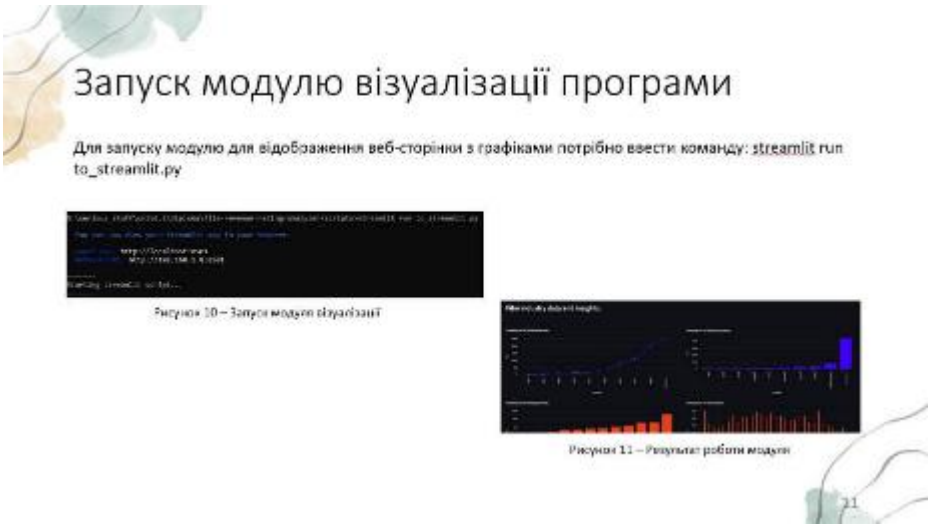


Рисунок В.11 – Слайд 11

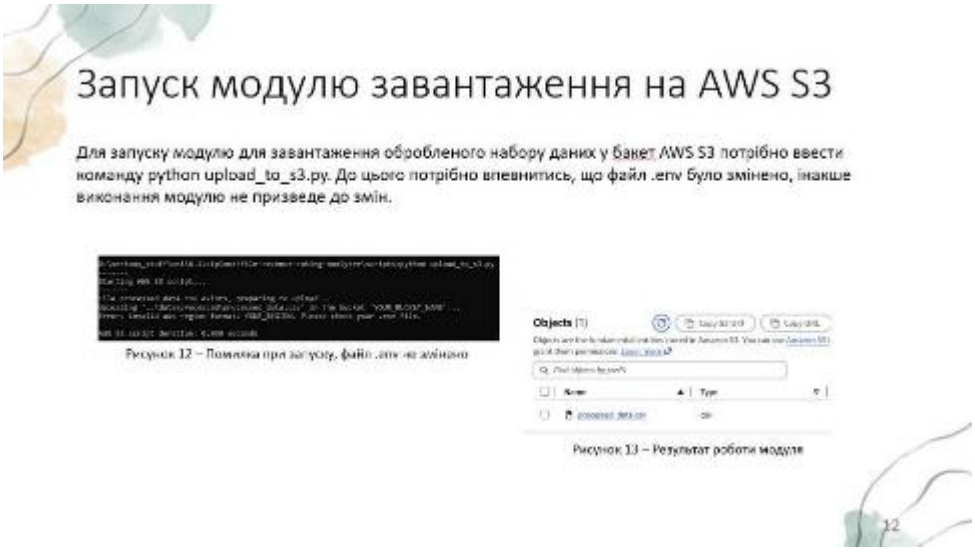


Рисунок В.12 – Слайд 12

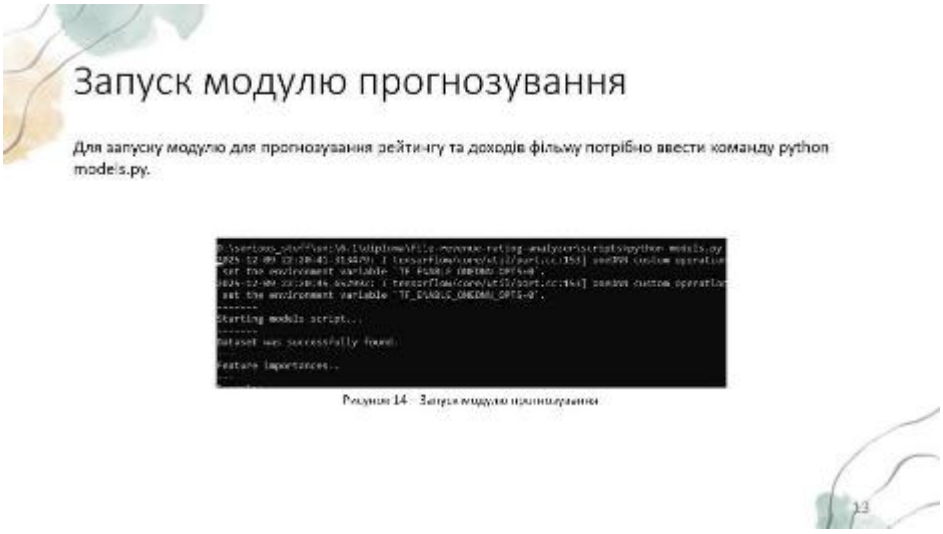


Рисунок В.13 – Слайд 13

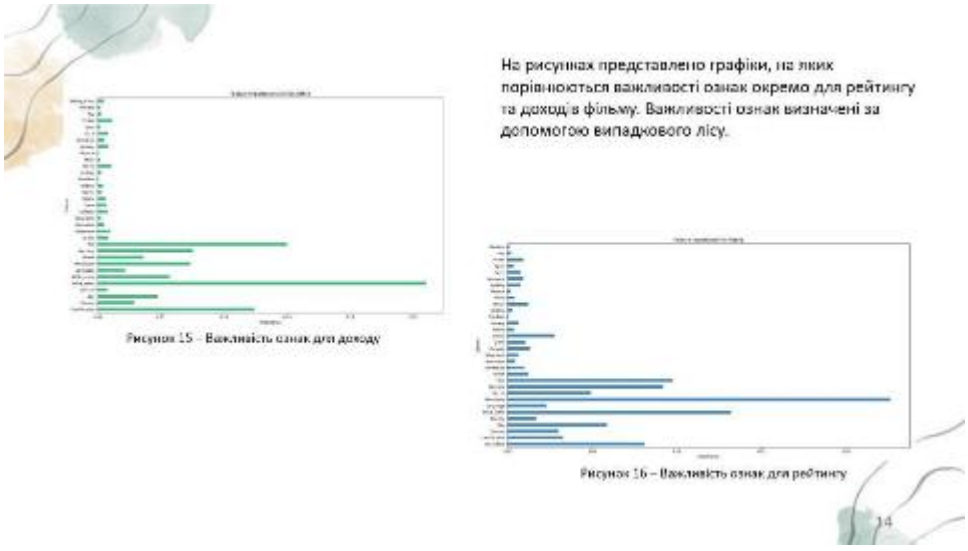


Рисунок В.14 – Слайд 14

Рисунок В.15 – Слайд 15

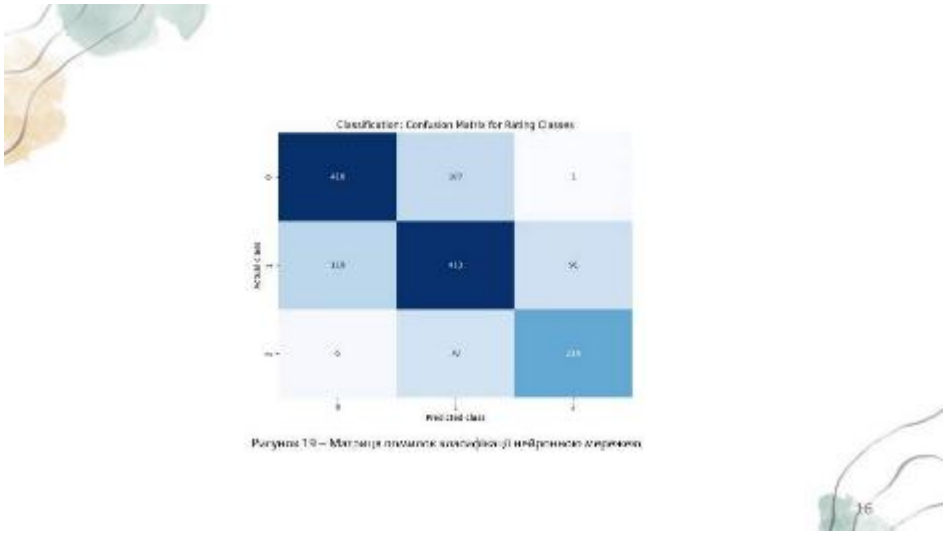


Рисунок 19 – Матриця помилок класифікації нейронною мережею

16

Рисунок В.16 – Слайд 16

Рисунок 20 – Регресія випадковим лісом

Рисунок 21 – Регресія нейронною мережею

Рисунок В.17 – Слайд 17

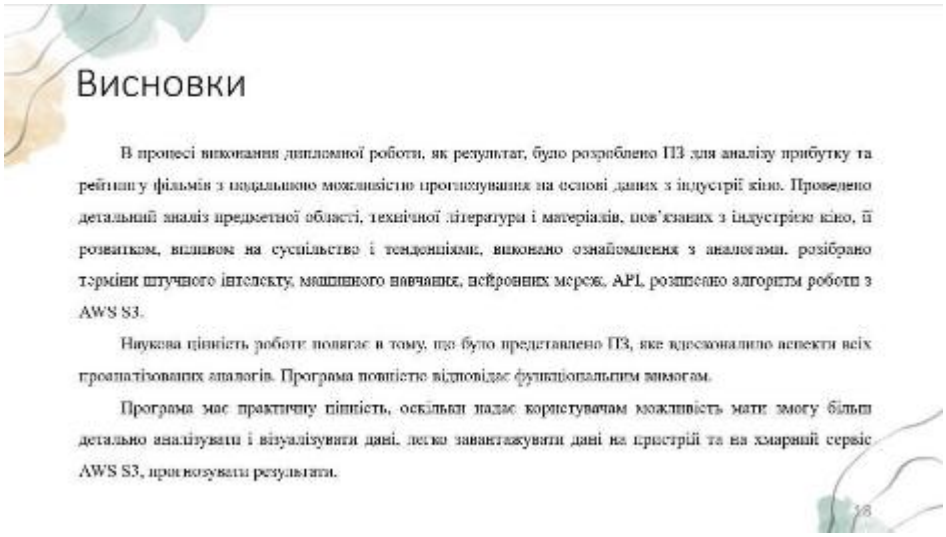


Рисунок В.18 – Слайд 18