IBM

University College of Enginnering Villupuram

Department of Computer Science and Engineering

Experience Based Project Learning - IBM (E2324)

completed the project

Personalized Recommendation System

Submitted By ,
PC.Vaishnavi (422522104051)

Team Members :

Roll No	Name :
422522104027	S.Akalya
422522104007	K.Harinitha
422522104034	G.Muthazhagi
422522104001	E.S.Mangala Yazhini

Personalized Recommendation System

Abstract :

This project aims to develop a personalised content based recommendation system using data science techniques and AI. A recommendation system is a subclass of Information filtering Systems that seeks to predict the rating or the preference a user might give to an item to recommend the user with the revelavent content and predict their choice in advance. These systems have become fundamental in enhancing customer experiences and improving operational efficiency for both business-to-business (B2B) and business-to-consumer (B2C) enterprises.Knowledge discovery techniques can be applied to the problem of making personalized recommendations about items or information during a user's visit to a website.

Introduction :

In the rapidly evolving digital landscape, the integration of advanced technologies is paramount for maintaining competitive advantage. One such technology that has gained significant traction is the recommendation system, an AI-driven tool designed to predict and suggest products or services to users based on various data points. These systems analyze vast amounts of data, including user behavior, past purchases, search history, and demographic information, to generate personalized recommendations.The importance of recommendation systems extends beyond simple product suggestions; they are crucial in enhancing the overall customer experience by making interactions more personalized and relevant. In general, recommendation systems can recognize patterns regarding similar products and users that even the best sellers of the companies cannot do. This ability of the recommendation systems is indisputable.Broadly, the recommender systems are classified into three categories :

1. Collaborative filtering :

Methods for recommender systems that are primarily based on previous interactions between users and the target items are known as collaborative filtering methods. As a result, all past data about user interactions with target objects will be fed into a collaborative filtering system.

2. Content-based filtering :

Content-based filtering is one popular technique of recommendation or recommender systems. It is widely used approach to provide personalized recommendations to users. These systems are based on the idea that a user's preferences can be predicted based on their previous interactions with items, such as their viewing and purchasing history.

3. Hybrid approaches :

This system combine both collaborative and content-based filtering to improve and build a recommendation system with higher quality and overcome the limitations of either approach when used independently.

Project Objectives :

Building a recommendation system aims to achieve several key objectives that enhance user experience, improve business metrics, and provide valuable insights. A Recommender System predicts the possibility that a user would favor an item. The main objectives of building a recommendation system.

Relevance :

This is the core objective – to recommend items that are truly relevant to the user's interests and needs. Recommendation systems achieve this by analyzing user data like past purchases, ratings, or browsing history.

Accuracy :

A good recommendation system goes beyond just being relevant. It should also be accurate in its predictions. This means suggesting items that the user is highly likely to enjoy or find useful, minimizing irrelevant recommendations.

Diversity :

While relevance is important, a system shouldn't just recommend similar items all the time. Diversity ensures users discover new items that fall outside their usual preferences, preventing them from getting stuck in a filter bubble.

Novelty :

Similar to diversity, novelty encourages exploration by suggesting new and trending items the user might not have encountered before. This can spark fresh interests and keep the user engaged.

User Satisfaction :

Ultimately, the goal is to create a satisfying user experience. This means providing recommendations that are helpful, save users time and effort, and make the platform more enjoyable to use.

Theese are the primary goals of the recommendation system are to improve the user experience by providing pertinent content recommendations.

System Requirements :

Building a content-based recommendation system requires a combination of hardware and software components to effectively process, analyze, and generate content based recommendations based on the content attributes of items. The key system requirements for Building Recommendation system :

Hardware Requirements :

CPU :

A powerful multi-core processor to handle data processing and algorithm computations efficiently and effectively.

Memory (RAM) :

Sufficient RAM (16GB or more) to load and process large datasets in memory, which is crucial for performance.

Storage :

SSD: For faster data access and processing.

HDD: For storing large datasets and backups.

GPU (specialized processor) :

It is used when working with large datasets, the parallel processing power of a GPU can significantly speed up data processing and model training.

Network:

High-speed internet connection for accessing data sources,and cloud services when needed for developing the project.

Software :

1. Operating System :

Windows 10 operating system

2. Programming language :

Python 3.10

3. Libraries :

- Pandas
- Numpy
- Matplotlib pyplot
- Seaborn
- Sklearn

4. Development Environment :

Jupyter Notebook or JupyterLab: It is used to for interactive development and testing of the program .

Methodology :

1. Data Collection :

The data collection step is foundational for building a recommendation system. It involves gathering and curating the data necessary to train and evaluate the recommendation algorithms. For a content-based movie recommendation system, this typically includes acquiring detailed information about movies.

Objectives :

1. **Gather Comprehensive Data :** Collect all relevant information about the items movies that will be recommended.Obtain a dataset from a source like IMDb, TMDb, or a public dataset on Kaggle

2. **Ensure Data Quality :** Ensure that the data is accurate, complete, and up-to-date to improve the recommendation system's effectiveness.

2. Data Preprocessing :

Data preprocessing is a critical step in building a recommendation system. It involves cleaning, transforming, and preparing the raw data to make it suitable for analysis and modeling. For a content-based movie recommendation system, preprocessing ensures that the textual data used for feature extraction and similarity calculations is in an optimal format.

Objectives :

1. **Clean Data :** Remove noise and irrelevant information from the dataset. This is needed to build a perfect recommendation system.

2. **Handle Missing Values :** Address any missing data to prevent issues during modeling.

3. **Enhance Data Quality :** Ensure consistency, accuracy, and completeness of the data to ensure the recommendation system function properly.

3. Data Tansformation :

Data transformation is the process of converting, cleansing, and structuring data into a usable format that can be analyzed to support decision making processes, and to propel the growth of an organization.Data transformation is an essential process in the development of

recommendation systems. It encompasses the conversion of raw data into a format suitable for analysis and modeling. This step holds particular significance in preparing the data for processing by machine learning algorithms and other analytical methods utilized in recommendation systems.

Objectives :

1. **Convert Data into Usable Formats :** Transform raw data into numerical or categorical formats that are suitable for analysis.

2. **Feature Engineering :** Create new features or modify existing ones to better capture the underlying patterns in the data.Create new features or transform existing ones to enhance the dataset. For instance, combine genres into a single string.

3. **Normalization and Scaling :** Ensure that numerical features are on a comparable scale to improve the performance of algorithms.

4. **Encoding Categorical Data :** Convert categorical data into numerical formats that can be processed by machine learning models.

Feature Extraction :

Feature extraction is a process in machine learning and natural language processing (NLP) where relevant information is extracted from raw data to create feature vectors, which are then used as input for machine learning algorithms. In the context of text data, feature extraction involves converting text documents into numerical vectors that can be understood and processed by machine learning models.

Term Frequency (TF) :

Measures how frequently a term occurs in a document. It is calculated as the number of times a term appears in a document divided by the total number of terms in the document.

Document Frequency (IDF) :

Measures the importance of a term across the entire corpus. It is calculated as the logarithm of the total number of documents divided by the number of documents containing the term.

TF-IDF :

It is the product of TF and IDF.It reflects how important a term is to a document in the corpus.

Cosine similarity :

Cosine similarity is a metric used to measure the similarity between two vectors in a

multidimensional space, often used in recommendation systems. It calculates the cosine of the angle between the two vectors, which indicates how closely related the vectors are in terms of their orientation. In the context of recommendation systems, cosine similarity is used to find similarities between item vectors or user vectors.

Model Evauation :

Model evaluation in recommendation systems involves assessing the performance and effectiveness of the recommendation algorithms in generating relevant and accurate recommendations for users. Several evaluation metrics and techniques are used to measure the quality of recommendations.

Evaluation Metrics :

Accuracy Metrics: These metrics measure the accuracy of the recommendations. Common accuracy metrics include precision, recall, and F1-score.

Precision: Precision measures the proportion of recommended items that are relevant to the user's interests out of all the recommended items

Recall:Recall measures the proportion of relevant items that are successfully recommended out of all the relevant items in the dataset.

F1-score: F1-score is the harmonic mean of precision and recall, providing a single metric that balances both precision and recall.

These accuracy metrics are fundamental in evaluating the performance of recommendation systems. Depending on the specific requirements and objectives of the recommendation system, different combinations of these metrics may be used to assess its effectiveness in providing relevant and accurate recommendations to users.

Model Deployment :

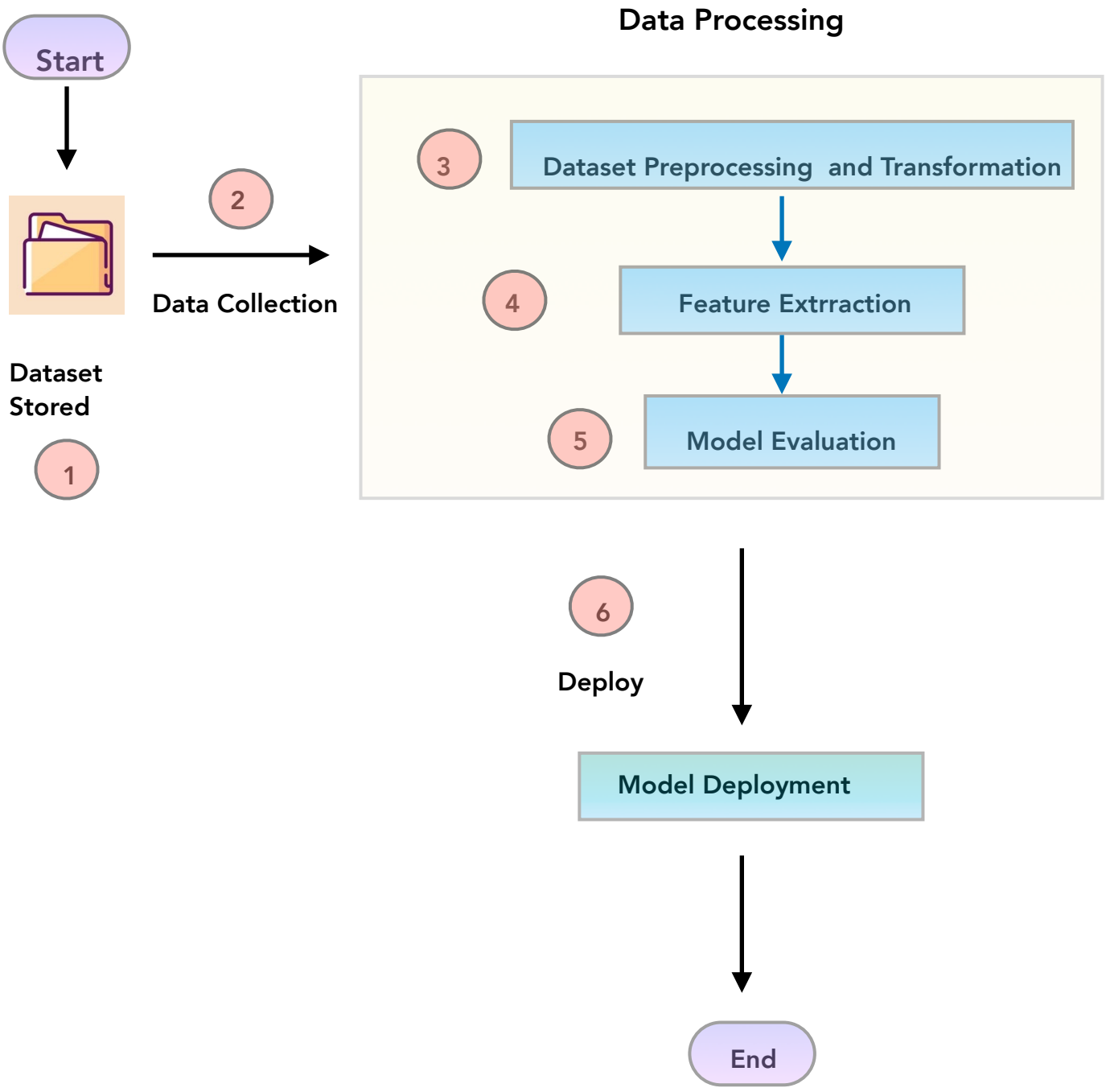
In data science, recommendation system deployment refers to the process of making the developed recommendation models or algorithms accessible and operational in a production environment where they can be used to generate recommendations for end-users or customers.

Proposed work :

Our project aims to develop a recommendation system for personalized e-commerce recommendations. The proposed work will begin with the collection and preprocessing of large-scale user-item interaction data from our e-commerce platform. We will conduct

thorough exploratory data analysis to gain insights into user preferences and item characteristics. Based on these insights, explore various recommendation algorithms, and deploy the optimal models into our production environment for continuous monitoring and maintenance. These models will be evaluated using metrics such as precision, recall, and F1-score to assess their performance. Once the optimal recommendation models are selected, we will deploy them into our production environment, seamlessly integrating them with our existing platform.

Flow chart :



Implementation of content based recommendation system :

 # 1.Data Collection :

 # Import the necessary libraies

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

 # Importing the dataset as DataFrame using pandas

```
movies = pd.read_csv("//content//drive//MyDrive//CSV_Files//Movies.csv")
credit = pd.read_csv("//content//drive//MyDrive//CSV_Files//Credit.csv")
```

 # Overview of the dataset

 # movies dataset

```
movies.head(2)
movies.tail(2)
```

 # Info of the movies dataset

```
movies.info()
```

 # Describe the movies dataset

```
movies.describe()
```

 # Shape of movies dataset

```
movies.shape
```

 # credit dataset

```
credit.head(2)
credit.tail(2)
```

 # Info of the credit dataset

```
credit.info()
```

 # Describe the credit dataset

```
credit.describe()
```

 # Shape of credit dataset

```
credit.shape
```

 # 2.Data Preprocessing :

 # Check Null values in movies dataset

```
movies.isnull().sum()
```

 # Droping the unnecessary column in movies dataset

```
movies.drop(columns = ["tagline","homepage","status"],inplace = True)
movies.isnull().sum()
```

 # Filling the null values in overview column in movies dataset

```
movies["overview"].isnull().sum()
movies["overview"] = movies["overview"].fillna("")
movies["overview"].isnull().sum()
```

 # Droping the null value in the release date column

```
movies["release_date"].isnull().sum()
release_date_values = movies[movies["release_date"].isnull()].index
movies.drop(release_date_values , inplace = True
movies["release_date"].isnull().sum()
```

 # Droping the null value in the run time column

```
movies["runtime"].isnull().sum()
runtime_values = movies[movies["runtime"].isnull()].index
movies.drop(runtime_values , inplace = True)
movies["runtime"].isnull().sum()
```

 # Check Null values in movies dataset after processing

```
movies.isnull().sum()
```

 # Check Null values in credit dataset

```
credit.isnull().sum()
```

 # 3.Data Validation :

 # Validate the movies dataset

```
movies.count()
```

 # Validate the credit dataset

```
credit.count()
```

 # 4.Data Formatting

 # Data formatting in the movies dataset
 # Convert all columns with revelant content into list of strings

 # Datatype of overview column

```
type(movies.loc[0]["overview"])
```

 # Convert the overview column into list of strings

```
movies["overview"] = movies["overview"].apply(lambda x : x.split())
```

 # Datatype of necessary columns in movies dataset

```
type(movies.loc[0]["genres"])
type(movies.loc[0]["keywords"])
type(movies.loc[0]["production_countries"])
type(movies.loc[0]["production_companies"])
type(movies.loc[0]["spoken_languages"])
```

 # Function to convert the columns into list datatype in movies dataset

```
import ast
def convert(text) :
    l = []
    for i in ast.literal_eval(text):
        l.append(i["name"])
    return l
```

 # Converting the columns into list datatype in movies dataset

```
movies["genres"] = movies["genres"].apply(convert)
movies["keywords"] = movies["keywords"].apply(convert)
movies["production_countries"] = movies["production_countries"].apply(convert)
movies["production_companies"] = movies["production_companies"].apply(convert)
movies["spoken_languages"] = movies["spoken_languages"].apply(convert)
```

 # Data formatting in the credit dataset
 # Datatype of cast column and crew column

```
type(credit.loc[0]["cast"])
type(credit.loc[0]["crew”])
```

 # Function to convert the columns into list datatype in credit dataset

```
def convert_cast(text) : l = []
    count = 0
    for i in ast.literal_eval(text):
        count += 1
        if count <= 10 :
            l.append(i["name"])
    return l

def convert_crew(text) :
    l = []
    for i in ast.literal_eval(text):
        if i['job'] == "Director" or i["job"] == "Original Music Composer" or i["job"]
          == "Editor" or i["job"] == "Writer":
            l.append(i["name"])
            break
    return l
```

 # Converting the columns into list datatype in credit dataset

```
credit["cast"] = credit["cast"].apply(convert_cast)
credit["crew"] = credit["crew"].apply(convert_crew)
```

```
# Conversion of white spaces to avoid error
# Function to convert the white spaces

def convert_whitespaces(text) :
    l = []
    for i in text :
        l.append(i.replace(" ",""))
    return l

# Converting the whitespaces

movies["genres"] = movies["genres"].apply(convert_whitespaces)
movies["keywords"] = movies["genres"].apply(convert_whitespaces)
movies["overview"] = movies["overview"].apply(convert_whitespaces)
credit["cast"] = credit["cast"].apply(convert_whitespaces)
credit["crew"] = credit["crew"].apply(convert_whitespaces)

# Overview of the dataset after conversion of columns into suitable datatypes

movies.head(2)
credit.head(2)

# 4.Merging the datasets

dataset = movies.merge(credit , on = "title")
dataset.head(2)

# Combining the required column into a single column

 dataset["tags"] = dataset["genres"] + dataset["keywords"]+ dataset["overview"]
+dataset["cast"]+ dataset["crew"]


# Convert the text in the tags column to lower case
# Function to convert the text to lower case

def lower_case(text) :
    l = []
    for i in text :
        i = i.lower()
        l.append(i)
    return l
```

```
# Converting to lowercase
 dataset["tags"] = dataset["tags"].apply(lower_case)
 dataset.head(2)

# Convert the tags column into string datatype

dataset["tags"] = dataset["tags"].apply(lambda x : " ".join(x))
dataset.head(2)

df = pd.DataFrame(dataset)

# Droping the unnecessary columns

df.drop(columns = ["genres","budget","keywords","original_language","overview",
                   "popularity","production_companies","production_countries",
                   "spoken_languages","vote_average","vote_count","movie_id",
                   "cast","crew","revenue","runtime","release_date","original_title"],
                   inplace =True)
df.head(2)


# 5.Exploratory data analysis
# Univariate Analysis

# Histogram for budget

plt.figure(figsize = (3.5,3))
sns.set(rc={'axes.facecolor':'#d4d4f7', 'figure.facecolor':'white'})
sns.histplot(data = dataset , x = "budget" , bins = 10 , color = "darkcyan",edgecolor =
"black" ,linewidth = 0.65)
plt.xlabel("Budget",weight='bold')
plt.ylabel("Frequency",weight='bold')
plt.title("Histogram for Budget",weight='bold')
plt.show()

# Histogram for Revenue

plt.figure(figsize = (3.5,3))
sns.set(rc={'axes.facecolor':'#d4d4f7', 'figure.facecolor':'white'})
sns.histplot(data = dataset , x = "revenue" , bins = 10 ,color = "#c23d5e",edgecolor =
"black" ,linewidth = 0.65)
```

```
plt.xlabel("Revenue",weight='bold')
plt.ylabel("Frequency",weight='bold')
plt.title("Histogram for Revenue",weight='bold')
plt.show()
```

 # Histogram for Popularity

```
plt.figure(figsize = (3.5,3))
sns.set(rc={'axes.facecolor':'#d4d4f7', 'figure.facecolor':'white'})
sns.histplot(data = dataset , x = "popularity" , bins = 10 , color = "#2eb8b8",edgecolor
= "black" ,linewidth = 0.45) plt.xlabel("Popularity",weight='bold')
plt.ylabel("Frequency",weight='bold')
plt.title("Histogram for Popularity",weight='bold')
plt.show()
```

 # Histogram for Run time

```
plt.figure(figsize = (3.5,3))
sns.set_style("darkgrid")
sns.set(rc={'axes.facecolor':'#d4d4f7', 'figure.facecolor':'white'})
sns.histplot(data = dataset , x = "runtime" , bins = 10  , color = "hotpink",edgecolor =
"black",linewidth = 0.45) plt.xlabel("Run Time",weight='bold')
plt.ylabel("Frequency",weight='bold')
plt.title("Histogram for Run Time",weight='bold')
plt.show()
```

 # Histogram for Vote Average

```
plt.figure(figsize = (4.5,4))
sns.set(rc={'axes.facecolor':'#d4d4f7', 'figure.facecolor':'white'})
sns.histplot(data = df , x = "vote_average" , bins = 10 , color = "#ff751a",
edgecolor = "black" ,linewidth = 0.65)
plt.xlabel("Vote Average",weight='bold')
plt.ylabel("Frequency",weight='bold')
plt.title("Histogram for Vote Average",weight='bold')
plt.show()
```

 # Histogram for Vote count

```
plt.figure(figsize = (4.5,4))
sns.set(rc={'axes.facecolor':'#d4d4f7', 'figure.facecolor':'white'})
sns.histplot(data = df , x = "vote_count" , bins = 10,
```

```
color ="#53c68c",edgecolor = "black" ,linewidth = 0.65 )
plt.xlabel("Vote Count",weight='bold')
plt.ylabel("Frequency",weight='bold')
plt.title("Histogram for Vote Count",weight='bold')
plt.show()
```

 # Top 5 genres

```
genres_count = dataset["genres"].explode().value_counts()
genres_df = pd.DataFrame(genres_count).reset_index()
plt.figure(figsize = (4.5,4))
sns.set(rc={'axes.facecolor':'#d4d4f7', 'figure.facecolor':'white' , "grid.color" : "grey"})
sns.barplot(x = genres_df["genres"].head(5),y = genres_df["count"].head(5),data = df ,
width = 0.5,edgecolor = "black" ,palette = "Spectral",linewidth = 0.45)
plt.xlabel("Genres",weight='bold')
plt.ylabel("Number of Movies",weight='bold')
plt.title("Top 5 Movies by Genres",weight='bold')
plt.xticks(rotation = 45 ,ha = "right")
plt.tight_layout() # Adjust layout to prevent overlap
plt.show()
```

 # Top 5 Production countries

```
countries_count = dataset["production_countries"].explode().value_counts()
countries_count_df = pd.DataFrame(countries_count).reset_index()
plt.figure(figsize = (5,5.5))
sns.set(rc={'axes.facecolor':'#d4d4f7', 'figure.facecolor':'white' , "grid.color" : "grey"})
sns.barplot(x = countries_count_df["production_countries"].head(5),y
=countries_count_df["count"].head(5),data = df ,linewidth = 0.65 ,
palette="Spectral",edgecolor = "black" )
plt.xlabel("Production Countries",weight='bold') plt.ylabel("Number of
Movies",weight='bold')
plt.title("Top 5 Movies by Production Countries",weight='bold') plt.xticks(rotation =
45 ,ha = "right")
plt.tight_layout() # Adjust layout to prevent overlap
plt.show()
```

 # Top 5 Production companies

```
company_count = df["production_companies"].explode().value_counts()
company_count_df = pd.DataFrame(company_count).reset_index()
plt.figure(figsize = (7.5,5))
```

```
sns.set(rc={'axes.facecolor':'#d4d4f7', 'figure.facecolor':'white', "grid.color" : "grey"})
sns.barplot(x = company_count_df["count"].head(5),data =
df ,y=company_count_df["production_companies"].head(5),
palette="Spectral",edgecolor='black',width = 0.5,linewidth=0.65)
plt.ylabel("Production Companies",weight='bold')
plt.xlabel("Number of Movies",weight='bold')
plt.title("Number of Movies by Production Companies",weight='bold')
plt.tight_layout()
plt.show()
```

 # Top 5 Spoken languages

```
language_count = dataset["spoken_languages"].explode().value_counts()
language_df = pd.DataFrame(language_count).reset_index()
language_df = language_df[language_df["spoken_languages"].str.match(r"[a-zA-Z]+$")]
plt.figure(figsize = (4.5,4))
sns.set(rc = {"axes.facecolor" : "#dacffc" , "figure.facecolor" : "white"})
sns.barplot(x = language_df["spoken_languages"].head(5) , y
=language_df["count"].head(5) , palette= "Spectral" ,edgecolor ="black",width = 0.4)
plt.xlabel("Spoken Languages",weight='bold')
plt.ylabel("Number of Movies",weight='bold')
plt.title("Top 5 Spoken languages in Movies",weight='bold')
plt.xticks(rotation = 65 ,ha = "right")
plt.show()
```

 # Revenue based on Decades

```
revenue_data = pd.DataFrame(dataset)
revenue_data["release_date"] = pd.to_datetime(revenue_data["release_date"])
revenue_data["release_year"] = revenue_data['release_date'].dt.year
revenue_data = revenue_data.groupby('release_year')['revenue'].mean().reset_index()
revenue_data["decade"] = (revenue_data["release_year"]//10) *10
revenue_df = revenue_data.groupby('decade')['revenue'].sum().reset_index()
plt.figure(figsize=(5.5,4.5))
sns.set(rc={'axes.facecolor':'#d4d4f7', 'figure.facecolor':'white' , "grid.color" : "grey"})
sns.barplot( x =revenue_df['decade'].head(10) , y =
revenue_df['revenue'].head(10),data=revenue_df , palette = "Spectral")
plt.xlabel('Decades', weight='bold')
plt.ylabel('Revenue', weight='bold')
plt.title('Decades vs Revenue', weight='bold')
plt.xticks(rotation = 45)
plt.show()
```

 # Top 10 Popular Movies :

```
vote_df = pd.DataFrame(df[["title","vote_average","vote_count"]])
vote_df["vote"] = vote_df["vote_average"]*vote_df["vote_count"]
vote_df = vote_df.sort_values("vote",ascending = False)
plt.figure(figsize = (5.5,4))
sns.set(rc={'axes.facecolor':'#d4d4f7', 'figure.facecolor':'white' , "grid.color" : "grey"})
sns.barplot(x = vote_df["title"].head(10) , y = vote_df["vote"].head(10) , data =
vote_df ,edgecolor = "black", palette = "Spectral",width = 0.65,linewidth = 0.4)
plt.xlabel("Movies",weight='bold')
plt.ylabel("Vote count",weight='bold')
plt.title("Top 10 High Voted and Popular Movies",weight='bold')
plt.xticks(rotation =40 , ha = "right")
plt.show()
```

 # Top 10 Movies by Crew Size
 # Extracting top 10 movies based on the number of crew members
 # Function to get the number of crew members

```
def get_crew_size(crew_list):
    return len(crew_list)
crew_data = pd.DataFrame(df)
crew_data['crew_size'] = crew_data['crew'].apply(get_crew_size)
top_10_crew_size = crew_data.nlargest(10, 'crew_size')[['title', 'crew_size']]
plt.figure(figsize=(5.5, 4))
sns.barplot(y ='title', x = 'crew_size', data =top_10_crew_size,palette="Spectral")
plt.xlabel('Number of Crew Members')
plt.ylabel('Movie Title')
plt.title('Top 10 Movies by Crew Size')
plt.gca().invert_yaxis() # To display the movie with the highest crew at the top
plt.show()
```

 # Top 10 Movies by Cast Size
 # Extracting top 10 movies based on the number of cast members
 # Function to get the number of cast members

```
get_cast_size = lambda cast_list: len(cast_list)
cast_data = pd.DataFrame(df)
cast_data['cast_size'] = cast_data['cast'].apply(get_cast_size)
top_10_cast_size = cast_data.nlargest(10, 'cast_size')[['title', 'cast_size']]
plt.figure(figsize=(5.5, 4))
```

```
sns.set(rc={'axes.facecolor':'#d4d4f7', 'figure.facecolor':'white' , "grid.color" : "white"})
sns.barplot(y ='title', x = 'cast_size',data = top_10_cast_size ,palette ="Spectral")
plt.xlabel('Number of Cast Members',weight = "bold")
plt.ylabel('Movie Title',weight = "bold")
plt.title('Top 10 Movies by Cast Size',weight = "bold")
plt.show()
```

 # Bivariate analysis

 # Regression plot

 # First plot

```
fig, axes = plt.subplots(2, 2, figsize=(12, 9))
fig.subplots_adjust(wspace=0.35, hspace=0.25)
```
 # Plotting on the first subplot
```
sns.regplot(x="popularity", y="revenue", data=dataset ,color =
"#ff9999" ,scatter_kws={'edgecolor': '#ff5050','linewidths': 0.3,'s' : 25},
line_kws={'color': '#009999','linewidth':1.8}, ax=axes[0][0])
axes[0][0].set_xlabel("Popularity", fontweight='bold',fontsize = 12)
axes[0][0].set_ylabel("Revenue", fontweight='bold',fontsize = 12)
axes[0][0].set_title("Popularity vs Revenue", fontweight='bold',fontsize =12)
```

 # Plotting on the second subplot
```
sns.regplot(x = "budget", y = "revenue" , data = dataset ,color =
"#ffa64d" ,scatter_kws={'edgecolor': '#ff8000','linewidths': 0.3,'s' : 25},line_kws={'color':
'#4da6ff','linewidth':1.8}, ax=axes[0][1])
axes[0][1].set_xlabel("Budget", fontweight='bold',fontsize = 12)
axes[0][1].set_ylabel("Revenue", fontweight='bold',fontsize = 12)
axes[0][1].set_title("Budget vs Revenue", fontweight='bold',fontsize =12)
```

 # Plotting on the third subplot
```
sns.regplot(x = "popularity", y = "vote_count" , data = dataset ,color =
"#4d79ff" ,scatter_kws={'edgecolor': '#002db3','linewidths': 0.3,'s' : 25},
line_kws={'color': '#d24dff','linewidth':1.8}, ax=axes[1][0])
axes[1][0].set_xlabel("Popularity", fontweight='bold',fontsize = 12)
axes[1][0].set_ylabel("Vote Count", fontweight='bold',fontsize = 12)
axes[1][0].set_title("Popularity vs Vote Count", fontweight='bold',fontsize =12)
```

```
# Plotting on the fourth subplot
sns.regplot(x = "budget", y = "vote_count" ,data = dataset ,color =
"#39ac73" ,scatter_kws={'edgecolor': '#26734d','linewidths': 0.3,'s' :
25},line_kws={'color': '#ff66b3','linewidth':1.8}, ax=axes[1][1])
axes[1][1].set_xlabel("Budget", fontweight='bold',fontsize = 12)
axes[1][1].set_ylabel("Vote Count ", fontweight='bold',fontsize = 12)
axes[1][1].set_title("Budget vs Vote count of Movie",fontweight='bold',fontsize = 12)
# Adjust layout for better display
plt.tight_layout()
plt.show()

# Second Plot

fig, axes = plt.subplots(2, 2, figsize=(12, 9))
fig.subplots_adjust(wspace=0.35, hspace=0.25)

# Plotting on the first subplot
sns.regplot(x = "crew_size", y = "vote_count" ,data = crew_data ,color ="#ff66b3" ,
scatter_kws={'edgecolor': '#cc0066','linewidths': 0.3,'s' : 25},line_kws={'color':
'#6666ff','linewidth':1.8}, ax=axes[0][0])
axes[0][0].set_xlabel("Crew Size", fontweight='bold',fontsize = 12)
axes[0][0].set_ylabel("Vote Count ", fontweight='bold',fontsize = 12)
axes[0][0].set_title("Crew Size vs Vote count of Movie",fontweight='bold',fontsize = 12)

# Plotting on the second subplot
sns.regplot(x = "cast_size", y = "vote_count" ,data = cast_data ,color
="#39ac73" ,scatter_kws={'edgecolor': '#26734d','linewidths': 0.3,'s' :
25},line_kws={'color': '#ff66b3','linewidth':1.8}, ax=axes[0][1])
axes[0][1].set_xlabel("Cast Size", fontweight='bold',fontsize = 12)
axes[0][1].set_ylabel("Vote Count ", fontweight='bold',fontsize = 12)
axes[0][1].set_title("Cast Size vs Vote count of Movie",fontweight='bold',fontsize = 12)

# Plotting on the third subplot

sns.regplot(x = "budget", y = "popularity" ,data = df ,color = "#b366ff" ,
scatter_kws={'edgecolor': '#6600cc','linewidths': 0.3,'s' : 25},line_kws={'color':
'#ff7733','linewidth':1.8}, ax=axes[1][0])
axes[1][0].set_xlabel("Budget", fontweight='bold',fontsize = 12)
axes[1][0].set_ylabel("Popularity", fontweight='bold',fontsize = 12)
axes[1][0].set_title("Budget and Popularity of Movie",fontweight='bold',fontsize = 12)
```

```
# Plotting on the fourth subplot
sns.regplot(x = "runtime", y = "popularity" ,data = cast_data ,color =
"#c6538c" ,scatter_kws={'edgecolor': '#993366','linewidths': 0.3,'s' :
25},line_kws={'color': '#00e673','linewidth':1.8}, ax=axes[1][1])
axes[1][1].set_xlabel("Run Time", fontweight='bold',fontsize = 12)
axes[1][1].set_ylabel("Popularity", fontweight='bold',fontsize = 12)
axes[1][1].set_title("Run Time and Popularity of Movie",fontweight='bold',fontsize = 12)
plt.show()

# Multivariate Analysis
# Heat map

numerical_data = pd.DataFrame(dataset)
numerical_data.drop(columns
=["genres","keywords","original_language","original_title","overview","production_co
mpanies","production_countries", "release_date","spoken_languages","title",
"cast","crew","id","tags"],inplace = True)
numerical_data.dropna(inplace = True) corr_matrix = numerical_data.corr()
sns.heatmap(corr_matrix ,annot = True,linewidths=.5 ,cmap="YlGnBu")
plt.show()

# Pairplot

plt.figure(figsize = (10,10))
sns.set(rc={'axes.facecolor':'#d4d4f7', 'figure.facecolor':'white' , "grid.color" : "white"})
sns.pairplot(data = numerical_data ,diag_kind="kde",diag_kws =
{"color":"hotpink"},kind="reg",plot_kws={'color':'#00b3b3','scatter_kws': {'s':
6.5 ,'edgecolor':'darkcyan' },'line_kws':       {'color':'#ff4d4d','linewidth':0.95}})
plt.show()

# 6. Build Recommendation system

# Using Tf and Idf to create tf_idf matrix

# cosine similarity to find similarity among the movies

# Overview of the processed dataset to build recommendation system

df.head()
```

```
# Import the libraries needed for find similar movies

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity


# Create an instance of TdidfVectorizer

tf_idf = TfidfVectorizer(stop_words='english')

# Transforming Text Data into a TF-IDF Matrix

tfidf_matrix = tf_idf.fit_transform(df['tags'])

# Calculating Cosine Similarity

cos_similarity = cosine_similarity(tfidf_matrix, tfidf_matrix)

# Creating a Mapping of Movie Titles to Indices

data_set = pd.Series(df.index ,index = df["title"]).drop_duplicates()

# Function to calculate the similarity between the movies

def get_recommendation(title ,cosine = cos_similarity) :
    if title not in list(df["title"]) :
        print(f"{title} not present in the dataset")
        return
    movie_index = data_set[title]
    sim_scores = cosine[movie_index]
    sim_scores = list(enumerate(sim_scores))
    sim_scores = sorted(sim_scores , key = lambda x : x[1] , reverse = True)
    top_recommendation = sim_scores[1:15]
    l = []
    for i in top_recommendation :
        l.append(i[0])
    movies_list = df["title"].iloc[l]
    recommended_list = []
    for movie in movies_list:
        recommended_list.append(movie)
    return recommended_list
```

```
def print_movie(get_recommendation):
        recommended_list = get_recommendation
        print("\nRecommended movies : \n")
        for recommended_movie in recommended_list :
                print(recommended_movie)
```

 # 7.Testing of the Recommendation system

```
print_movie(get_recommendation("The Dark Knight Rises"))
print_movie(get_recommendation("Spider-Man"))
print_movie(get_recommendation("Iron Man 3"))
print_movie(get_recommendation("Jurassic World"))
print_movie(get_recommendation("Toy Story 3"))
```

 # 8.Model Evaluation

 # Define functions for evaluation metrics

```
def precision_at_k(actual, predicted, k):
        actual_set = set(actual)
        predicted_set = set(predicted[:k])
        common_items = actual_set.intersection(predicted_set)
        precision = len(common_items) / k if k > 0 else 0
        return precision

def recall_at_k(actual, predicted, k):
        actual_set = set(actual)
        predicted_set = set(predicted[:k])
        common_items = actual_set.intersection(predicted_set)
        recall = len(common_items) / len(actual) if len(actual) > 0 else 0
        return recall

def f1_score(precision, recall):
        f1 = 2 * (precision * recall) / (precision + recall)
        if (precision +recall) > 0 else 0
        return f1
```

```
# Function to select random movies
import random
def select_elements(lst):
    # Ensure the list is not empty
    if not lst:
        return None # or raise an exception
     # Randomly select at least one element
    num_elements_to_select = random.randint(1, len(lst))
    selected_elements = random.sample(lst, num_elements_to_select)
    return selected_elements

# Evaluate Precision@K, Recall@K, and F1 Score

def evaluate(title):
    actual = get_recommendation(title)
    actual = actual[:5]
    actual_preferences = select_elements(actual)
    # Generate recommendations using the recommendation function
    recommended_movies = get_recommendation(title)
    k=3
    precision = precision_at_k(actual_preferences, recommended_movies, k)
    recall = recall_at_k(actual_preferences, recommended_movies, k)
    f1 = f1_score(precision, recall)
    print("Precision@{}: {:.4f}".format(k, precision))
    print("Recall@{}: {:.4f}".format(k, recall))
    print("F1 Score: {:.4f}".format(f1))


evaluate("Spider-Man")
evaluate("Ice Age")
evaluate("Toy Story")
evaluate("Iron Man")
```

Output :

1. Data collection :

Overview of movies dataset :

	budget	genres	homepage	id	keywords	original_language	original_title	overview	popularity	production_companies	production_countries	release_date	revenue	runtime	spoken_languages	status	tagline	title	vote_average	vote_count
0	23700000	[{"id": 28, "name"	http://www.avatarmovie.com/	1999	[{"id": 1463, "name":	en	Avatar	In the 22nd century, a paraplegic	150.437577	[{"name": "Ingenious Film	[{"iso_3166_1": "US", "name":	2009-12-10	2787965(	162.0	[{"iso_639_1": "en"	Released	Enter the World	Avatar	7.2	11800
1	30000000	[{"id": 12, "name"	http://disney.go.com/disneypictu	285	[{"id": 270, "name": "ocean"}, {"id": 726	en	Pirates of the Caribbean: At	Captain Barbossa, long believed to	139.082615	[{"name": "Walt Disney Pictures", "id": 2}, {"	[{"iso_3166_1": "US", "name": "United	2007-05-19	9610000(	169.0	[{"iso_639_1": "en", "name":	Released	At the end of the world	Pirates of the Cari	6.9	4500

Overview of credit dataset :

	movie_id	title	cast	crew
0	19995	Avatar	[{"cast_id": 242, "character": "Jake Sully", "...	[{"credit_id": "52fe48009251416c750aca23", "de...
1	285	Pirates of the Caribbean: At World's End	[{"cast_id": 4, "character": "Captain Jack Spa...	[{"credit_id": "52fe4232c3a36847f800b579", "de...

Describe movies dataset :

	budget	id	popularity	revenue	runtime	vote_average	vote_count
count	4.803000e+03	4803.000000	4803.000000	4.803000e+03	4801.000000	4803.000000	4803.000000
mean	2.904504e+07	57165.484281	21.492301	8.226064e+07	106.875859	6.092172	690.217989
std	4.072239e+07	88694.614033	31.816650	1.628571e+08	22.611935	1.194612	1234.585891
min	0.000000e+00	5.000000	0.000000	0.000000e+00	0.000000	0.000000	0.000000
25%	7.900000e+05	9014.500000	4.668070	0.000000e+00	94.000000	5.600000	54.000000
50%	1.500000e+07	14629.000000	12.921594	1.917000e+07	103.000000	6.200000	235.000000
75%	4.000000e+07	58610.500000	28.313505	9.291719e+07	118.000000	6.800000	737.000000
max	3.800000e+08	459488.000000	875.581305	2.787965e+09	338.000000	10.000000	13752.000000

Describe credit dataset :

	movie_id
count	4803.000000
mean	57165.484281
std	88694.614033
min	5.000000
25%	9014.500000
50%	14629.000000
75%	58610.500000
max	459488.000000

2. Data Preprocessing :

Null values in movies dataset :

```
budget                     0
genres                     0
homepage                3091
id                         0
keywords                   0
original_language          0
original_title             0
overview                   3
popularity                 0
production_companies       0
production_countries       0
release_date               1
revenue                    0
runtime                    2
spoken_languages           0
status                     0
tagline                  844
title                      0
vote_average               0
vote_count                 0
dtype: int64
```

Null values in credit dataset :

```
movie_id    0
title       0
cast        0
crew        0
dtype: int64
```

After preproseccing movies dataset :

```
budget                  0
genres                  0
id                      0
keywords                0
original_language       0
original_title          0
overview                0
popularity              0
production_companies    0
production_countries    0
revenue                 0
spoken_languages        0
status                  0
title                   0
vote_average            0
vote_count              0
dtype: int64
```

3. Data Validation :

Validate the movies dataset :

```
budget                4800
genres                4800
id                    4800
keywords              4800
original_language     4800
original_title        4800
overview              4800
popularity            4800
production_companies  4800
production_countries  4800
release_date          4800
revenue               4800
runtime               4800
spoken_languages      4800
title                 4800
vote_average          4800
vote_count            4800
```

Validate the credit dataset :

```
movie_id    4803
title       4803
cast        4803
crew        4803
dtype: int64
```

4. Data Merging :

Overview of the merged dataset :

	genres	id	keywords	title	overview	movie_id	cast	crew
0	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	19995	[{"id": 1463, "name": "culture clash"}, {"id":...	Avatar	In the 22nd century, a paraplegic Marine is di...	19995	[{"cast_id": 242, "character": "Jake Sully", "...	[{"credit_id": "52fe48009251416c750aca23", "de...
1	[{"id": 12, "name": "Adventure"}, {"id": 14, "...	285	[{"id": 270, "name": "ocean"}, {"id": 726, "na...	Pirates of the Caribbean: At World's End	Captain Barbossa, long believed to be dead, ha...	285	[{"cast_id": 4, "character": "Captain Jack Spa...	[{"credit_id": "52fe4232c3a36847f800b579", "de...

5. Data Formatting :

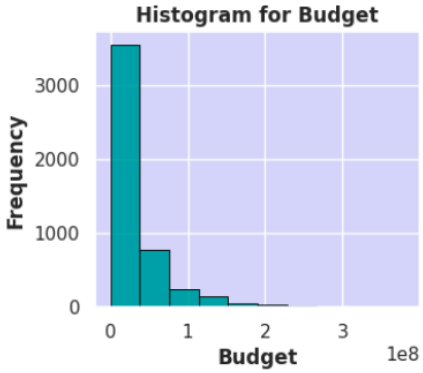
Overview of the formatted dataset :

	genres	id	keywords	title	overview	movie_id	cast	crew
0	[Action, Adventure, Fantasy, Science Fiction]	19995	[culture clash, future, space war, space colon...	Avatar	In the 22nd century, a paraplegic Marine is di...	19995	[Sam Worthington, Zoe Saldana, Sigourney Weaver]	[James Cameron]
1	[Adventure, Fantasy, Action]	285	[ocean, drug abuse, exotic island, east india ...	Pirates of the Caribbean: At World's End	Captain Barbossa, long believed to be dead, ha...	285	[Johnny Depp, Orlando Bloom, Keira Knightley]	[Gore Verbinski]

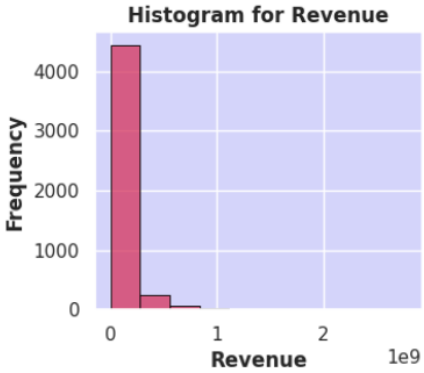
6. Exploratory Data Analysis :

Univariate Analysis :

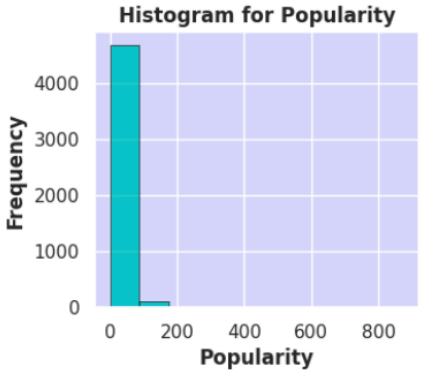
Histogram for Budget :



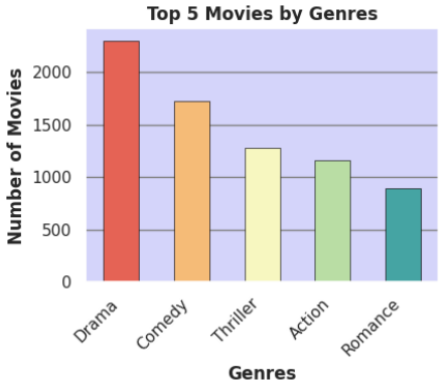
Histogram for Revenue :



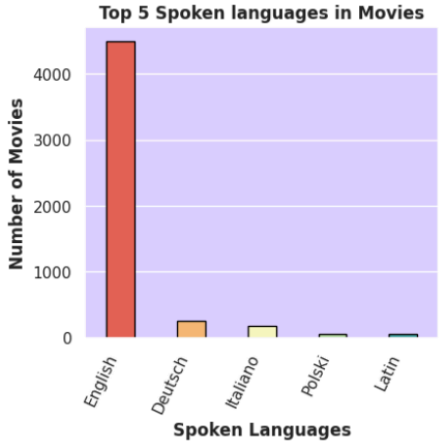
Histogram for Popuarity :



Bar plot for Genres :

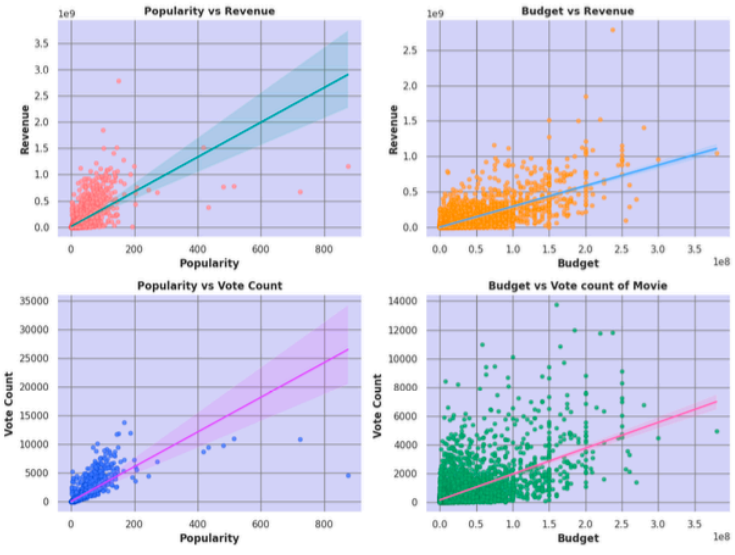


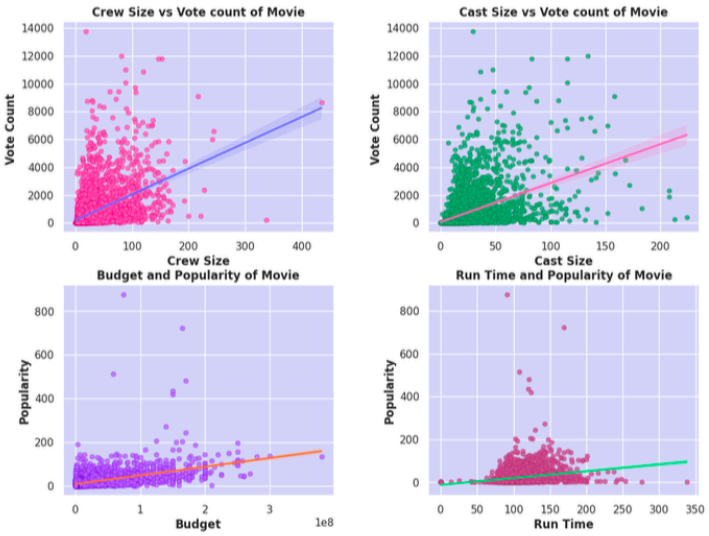
Bar plot for Spoken Languages :



Bivariate Analysis :

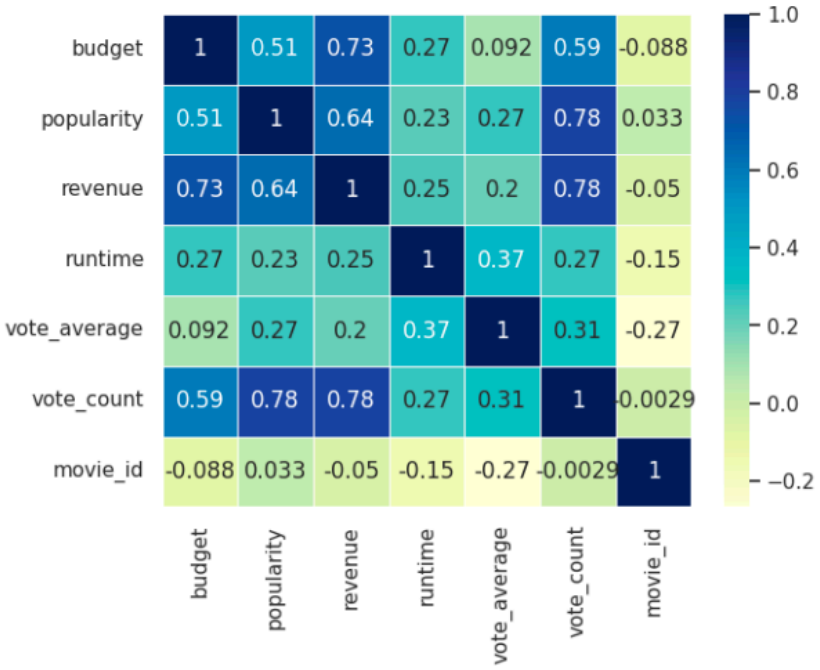
Regression plot :



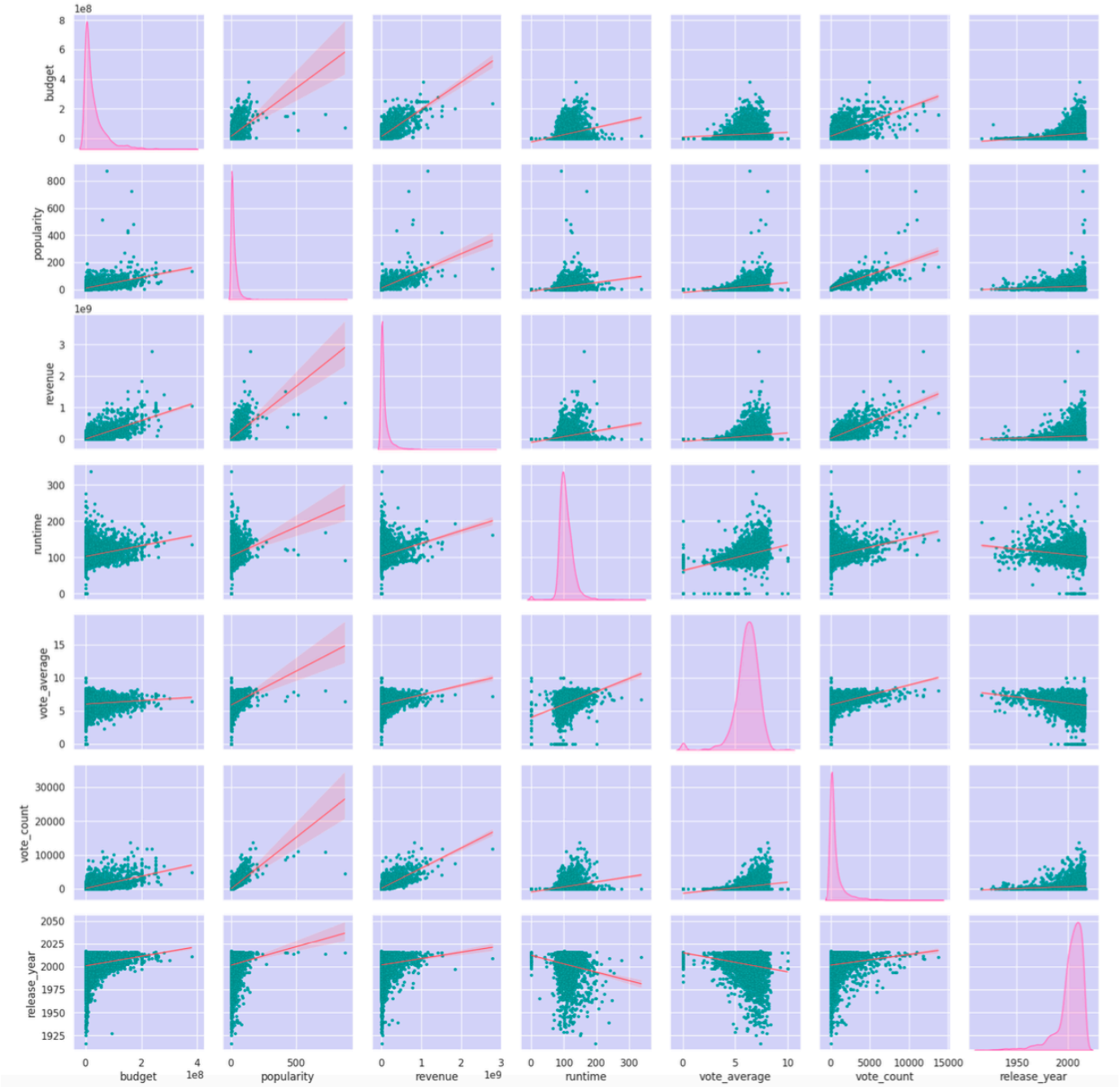


Multivariate Analysis :

Heat Map :



Pairplot :


budget
popularity
revenue
runtime
vote_average
vote_count
release_year
1e8
1e9

7. Testing of the Recommendation system :

Recommendation for "The Dark Knight Rises" :

```
Recommended movies :

The Dark Knight
Batman Begins
Batman Forever
Batman Returns
Batman
Batman
Slow Burn
Inception
Batman: The Dark Knight Returns, Part 2
Batman & Robin
Batman v Superman: Dawn of Justice
JFK
Sin City: A Dame to Kill For
Premium Rush
```

Recommendation for "Spider-Man":

```
Recommended movies :

Spider-Man 2
Spider-Man 3
The Amazing Spider-Man 2
The Amazing Spider-Man
Gremlins 2: The New Batch
Arachnophobia
Charlotte's Web
Hook
Small Soldiers
Spawn
Kick-Ass
Mystery Men
Man of Steel
Heavenly Creatures
```

Recommendation for "Iron-Man 3" :

```
Recommended movies for Iron Man 3 :

Iron Man
Iron Man 2
The Helix… Loaded
Avengers: Age of Ultron
Captain America: Civil War
Star Trek Into Darkness
Cradle 2 the Grave
X-Men
The Walk
The Avengers
The Abyss
```

Recommendation for "Jurassic World":

```
Recommended movies for Jurassic World :

The Lost World: Jurassic Park
Jurassic Park
The Helix… Loaded
National Lampoon's Vacation
Guardians of the Galaxy
The Cell
The Nut Job
Impostor
The 5th Wave
The Blood of Heroes
Jurassic Park III
Cloverfield
Terminator Salvation
Knowing
```

8. Model Evaluation :

Precision for "Spider-Man" :

```
Precision@3: 0.3333
Recall@3: 0.5000
F1 Score: 0.4000
```

Precision for "Ice Age" :

```
Precision@3: 0.6667
Recall@3: 0.5000
F1 Score: 0.5714
```

Precision for "Toy Story" :

```
Precision@3: 1.0000
Recall@3: 0.6000
F1 Score: 0.7500
```

Conclusion :

In conclusion, the personalized movie recommendation system, utilizing a content-based recommendation algorithm and developed through Python libraries, has showcased the efficiency of data-driven approaches in elevating user satisfaction and experience through customized suggestions. The project's meticulous approach to data acquisition, preprocessing, model training, testing, and evaluation has ensured the system's accuracy and efficacy. This contribution has led to a more engaging user experience, fostering customer loyalty and driving business growth in the digital landscape.