MSc Computer Science
London Campus

Est. 1841 | YORK ST JOHN UNIVERSITY

Module Title	Big Data
Module code	COM7036M
Student ID	230150452
Submission Date	27/05/2024

Student Declaration:

I confirm that I have read and understood the University Policy and Procedures on Academic Misconduct and that the work submitted is my own.	✓
I confirm that I have processed and produced this submission in accordance with the University guidelines and the assessment brief regarding the use of generative AI. Where appropriate, I have acknowledged where and how it has been used. OR Where generative AI is not permitted in this assignment. I confirm that it has not been used in any part of the process or production of this submission https://www.yorksj.ac.uk/policies-and-documents/generative-artificial-intelligence/	✓ ✓

Table of Contents

Section A

a) Dimensions of Big Data

The term "big data" describes the vast volumes of information produced by numerous sources. Its potential value, diversity, rapidity of creation, and variation in quality all contribute to its complexity (Klarity Analytics, 2015). A comprehensive understanding of the dimension of Big Data is necessary for efficient management and analysis. Below is a breakdown of V's of the Big Data:

- Volume: This dimension is used to describe the enormous amounts of data that are created and gathered from many sources. Processing and analysing such massive datasets effectively call for sophisticated ways to store and strong processing power. Large volumes of data must be handled and processed quickly, which requires the use of tools like cloud storage platforms (like Amazon S3) and distributed computing frameworks (like Hadoop, Spark).
- Velocity: The pace at which data is generated, gathered, and processed—including real-time streams from sources like social media and Internet of Things devices—is referred to as velocity. For immediate insights, fast data must be processed in actual time. Rapid collection and analysis are made possible by technologies like Apache Kafka and Apache Flink, which are essential for applications like fraud detection and tailored suggestions.
- Variety: Different data formats and types are included in the term "variety," including unstructured (text, pictures), semi-structured (XML, JSON), and structured (databases). The integration process, storage, and analysis are made more difficult by this variability. Managing and extracting insights from heterogeneous data sources requires solutions such as NoSQL databases, data lakes, and specific tools for unstructured data (e.g., text NLP).
- Veracity: It is the quality and dependability of data in terms of accuracy, consistency, and reliability. Reliability in decision-making depends on high-quality data. When big data insights are backed by methods like data validation, cleansing, and robust data governance protocols that maintain data integrity, they become dependable and valuable.
- Variability: Inconsistencies in data flows, such as modifications to formats, quality, and volume, are referred to as variability. For consistent performance, managing it necessitates adjusting to shifting data loads. This necessitates the use of strong data processing systems and management techniques to guarantee accurate analytics, particularly when data-driven applications are at their busiest.
- Value: Value refers to the benefits and insights extractable from data, emphasizing economic and strategic value. Big Data initiatives aim to derive valuable insights driving business decisions and competitive advantages. Utilizing data analytics, mining, and business intelligence tools transforms raw data into actionable insights, optimizing operations, enhancing customer experiences, and identifying new business opportunities.

b) Benefits of Different File Formats in Big Data:

The advantages of different file formats in big data are as below:

- Structured Data Formats (e.g., CSV, JSON, XML): Structured data formats are well-suited for storing and processing tabular data with predefined schemas. They offer simplicity, ease of use, and compatibility with a wide range of tools and systems. For example, CSV files are commonly used for exchanging data between different applications, while JSON and XML formats are popular for web-based data exchange and API integration (Ramos, 2020).
- Parquet: Parquet is a columnar storage format designed for big data processing frameworks such as Apache Hadoop and Apache Spark It reduces storage space and improves query performance by utilizing effective compression and encoding techniques. Parquet is the best choice for analytical tasks involving the scanning of big datasets.
- Avro: Avro is a framework for data serialization that provides support for schema evolution, complex data structures, and compact binary encoding. It is widely used in big data ecosystems for data serialization, messaging, and data exchange between different systems. Avro's support for schema evolution enables organizations to evolve their data schemas over time without breaking compatibility.
- ORC (Optimized Row Columnar): This is a columnar storage format designed for storing structured data in Apache Hive, a data warehouse infrastructure built on top of Hadoop. ORC offers advanced compression and encoding techniques, enabling faster query performance and efficient storage utilization. It is commonly used for data warehousing and analytical processing in big data environments.
- HBase: HBase is a scalable, distributed NoSQL database that runs on top of the Hadoop Distributed File System (HDFS). It enables real-time random read/write access to large amounts of structured data, making it ideal for scenarios requiring low-latency data access, such as real-time analytics, monitoring, and personalization.

Section B

Introduction

Big data represents the amalgamation of vast and intricate datasets, challenging to manage using conventional methods. It encompasses structured, semi-structured, and unstructured data sourced from diverse channels like social media, sensors, and business transactions (Botelho and Bigelow, 2022). The sheer volume, speed (velocity), and diversity (variety) of big data offer a spectrum of possibilities and hurdles across industries. Delving into big data analysis unveils invaluable insights into consumer behaviours, market dynamics, and operational optimizations, empowering organizations to navigate complexities and drive informed decisions.

Netflix Movie and TV Shows Dataset Overview

The Netflix Movie and TV Shows dataset, sourced from Kaggle, provides an extensive collection of data reflecting the diverse content available on the esteemed streaming platform. It encompasses a variety of details including titles, genres, release dates, ratings, IMDB rating and descriptions for both movies and TV shows, offering a comprehensive glimpse into Netflix's vast library. With approximately 8,800 records distributed across 15 columns,

this dataset offers valuable insights into entertainment preferences, audience demographics, and content consumption behaviors. Researchers, analysts, and enthusiasts can delve into this dataset to uncover trends that shape the landscape of streaming media and influence viewership patterns and cultural dynamics.

Additionally, the creation of this dataset involved the meticulous merging of two distinct datasets: "title.ratings" (measuring 23.7 MB) and "netflix_titles" (weighing 3.24 MB). Through this integration, ratings data seamlessly blends with detailed attributes of Netflix titles, providing a comprehensive perspective on the platform's content ecosystem. This cohesive merging process enables nuanced analysis, empowering researchers to identify correlations between ratings and various facets of Netflix movies and TV shows. Ultimately, this amalgamation fosters insightful discoveries and facilitates informed decision-making in the realm of streaming media analytics.

1. Data Understanding and Pre-processing:

In Netflix Movies and TV shows Dataset, especially in the realm of Big Data, data preprocessing and exploration play pivotal roles. These tasks involve handling vast volumes of data, cleaning, transforming, and preparing it for analysis and recommendation system.

1.1. Importing Essential Libraries:

The code imports Pandas, NumPy, Matplotlib, Seaborn, Plotly Express, and NLTK for data manipulation, visualization, and text processing. It also includes modules for date parsing, machine learning algorithms, and model persistence.

Figure 1: Importing necessary Libraries.

1.2 Loading data

Loading Data

```
]: # Importing two datasets netflix titles and title ratings.
movies = pd.read_csv('netflix_titles.csv')
imdb = pd.read_csv('title.ratings.tsv', sep='\t')
```

Figure 2:Loading Data

The code imports the Netflix titles dataset from a CSV file into a pandas DataFrame named "movies", and the IMDb ratings dataset from a TSV file into another DataFrame named "imdb". This allows for easy data manipulation and analysis using the panda's library.

1.3 Understanding and knowledge gaining about dataset

Initial dataset inspection involves examining the dataset's structure and features at the start of an analysis. This includes tasks like checking the first and last rows, summarizing statistics for both numerical and categorical variables, and identifying data types.

```
# Inspect the data
df.head(3)   #To print the first three rows of the dataset.
```

Figure 3: First three rows

The above figure shows that quick preview the first three rows of a dataset using "df.head(3)".

```
]: df.tail(3)   #To print the Last three rows of the dataset.
```

Figure 4:Last three rows

The above figure shows that quick preview the last three rows of a dataset using "df.tail (3)".

```
[20]: df.info()   #Summary of the DataFrame.
<class 'pandas.core.frame.DataFrame'>
Index: 8807 entries, 0 to 8806
Data columns (total 15 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   show_id        8807 non-null   object
 1   type           8807 non-null   object
 2   title          8807 non-null   object
 3   director       6173 non-null   object
 4   cast           7982 non-null   object
 5   country        7976 non-null   object
 6   date_added     8797 non-null   object
 7   release_year   8807 non-null   int64
 8   rating         8803 non-null   object
 9   duration       8804 non-null   object
 10  listed_in      8807 non-null   object
 11  description    8807 non-null   object
 12  tconst         8807 non-null   object
 13  averageRating  8807 non-null   float64
 14  numVotes       8807 non-null   int64
dtypes: float64(1), int64(2), object(12)
memory usage: 1.1+ MB
```

Figure 5:Summary of the DataFrame

In figure 5, “df.info()” command offers a compact overview of the DataFrame “df”, detailing the data types for each column, the count of non-null values, and memory usage. It serves as a helpful tool for grasping the Data Frame’s structure and pinpointing any missing data.

1.4 Handling missing values

Missing value handling involves dealing with data points that are missing or unknown in a dataset. This process includes identifying missing values, determining how to manage them, and applying methods like imputation or deletion to address the issue. We use missing value handling for Maintaining Data Integrity, Ensuring Analytical Validity, Improving Model Performance. The following steps that have been taken:

a) Duplicate value Count

```
# Duplicate Value Count
print(f"Duplicate rows: {len(df[df.duplicated()])}")
```

```
Duplicate rows: 0
```

Figure 6: Duplicate Value Count

The code helps with data cleaning and analysis by identifying the existence and number of duplicate records in the dataset. It does this by counting and printing duplicate rows in a Pandas DataFrame "df."

b) Visualize null Value

The above code is used to create a heatmap that visualizes the presence of the null(missing) values in the dataset using Seaborn and Matplotlib libraries. Where “df.isnull()” returns a “DataFrame” with “True” for nulls and “False” for non-nulls.

```
# Null Value Heatmap
plt.figure(figsize=(6, 4))
sns.heatmap(df.isnull(), cbar=False)
plt.show()
```

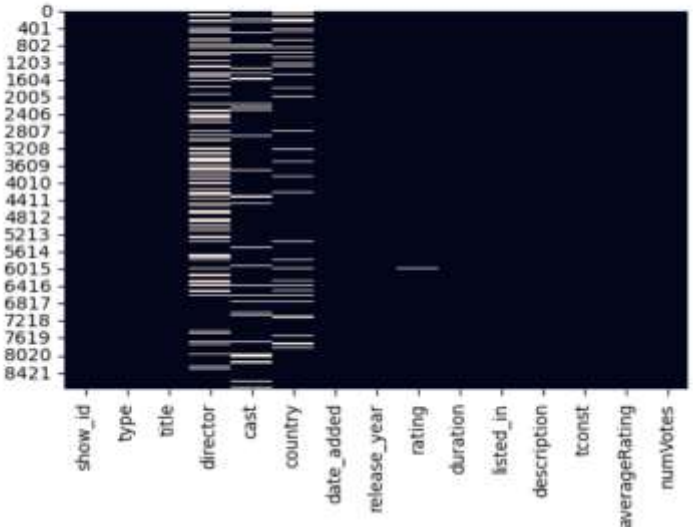

Figure 7: Null Value Visualize

```
[91]: # Checking number of null values
      print(df.isnull().sum())
```

```
show_id            0
type               0
title              0
director        2634
cast             825
country          831
date_added        10
release_year       0
rating             4
duration           3
listed_in          0
description        0
tconst             0
averageRating      0
numVotes           0
dtype: int64
```

Figure 8: Null values in number

The code displayed in Figure 10, prints the total count of missing values in each column of the DataFrame "df". According to the figure, columns such as "director", "cast", "country", "date_added", "release_year", "rating", and "duration" exhibit missing values.

c) Handling missing values

In figure 9, the code outlines crucial data cleaning steps on the DataFrame "df". Initially, missing values in the "country" column were filled with the most common value, ensuring completeness. Missing entries in "director" and "cast" were replaced with "Unknown" for clarity. Rows with missing values in "date_added", "rating", and "duration" were dropped. Dates in "date_added" were parsed for consistency and accuracy in handling date data.

```
[92]: #DEALING WITH MISSING VALUES
      df['cast'] = df['cast'].fillna('Unavailable')
      df['director'] = df['director'].fillna('Unavailable')
      df['country'] = df['country'].fillna(df['country'].mode()[0])
      df['date_added'] = df['date_added'].fillna(df['date_added'].mode()[0])
      df['rating'] = df['rating'].fillna(df['rating'].mode()[0])
      df['duration'] = df['duration'].fillna(df['duration'].mode()[0])
      df.isnull().sum()
```

```
[92]: show_id          0
      type             0
      title            0
      director         0
      cast             0
      country          0
      date_added       0
      release_year     0
      rating           0
      duration         0
      listed_in        0
      description      0
      tconst           0
      averageRating    0
      numVotes         0
      dtype: int64
```

Figure 9: Handeling Missing values

1.5 Label Encoding

Label encoding gives each category and label a unique numerical code to transform categorical data into numerical forms. In figure 10, the code transforms the “type” column into numeric values and creates dummy variables for the 'country' column. Additionally, it encodes the “rating” column into numeric values using “LabelEncoder”.

```
# Encode categorical variables
label_encoder = LabelEncoder()
df['type_encoded'] = label_encoder.fit_transform(df['type'])
# df['director_encoded'] = label_encoder.fit_transform(df['director'].astype(str))
df = pd.get_dummies(df, columns=['country'], prefix='country')
df['rating_encoded'] = label_encoder.fit_transform(df['rating'].astype(str))
```

Figure 10: Label Encoding

1.6 TF-IDF Vectorization

```
from sklearn.feature_extraction.text import CountVectorizer
# #using tfidf

tfidf = TfidfVectorizer(stop_words='english',max_features=20000)
tfidf_m = tfidf.fit_transform(rec_data['bag'])
tfidf_m
```

```
<8807x20000 sparse matrix of type '<class 'numpy.float64'>'
        with 252804 stored elements in Compressed Sparse Row format>
```

Figure 11: TF-IDF vectorization

This above code, initializes a TF-IDF vectorizer tfidf with English stop words removed and a maximum of 20,000 features, and transforms the 'bag' column of the DataFrame “rec_data” into a TF-IDF matrix “tfidf_m”. This is done in preparation for implementing K-means clustering.

1.7 Dimensionality Reduction

The goal of dimensionality reduction is to enhance model performance and streamline computation by minimizing the number of features in a dataset while maintaining the information that is essential. There many ways to reduce dimensionality among them Principal Component Analysis (PCA) is the most common one. In the figure 12, It initializes a PCA object “pca_tuned” with the specified number of components, 5000. Then, it converts the TF-IDF matrix “tfidf_m” to a dense array “x_dense” and fits the PCA model to this data. Finally, it transforms the data using PCA and prints the resulting shape.

```
[51]: from sklearn.decomposition import PCA

pca_tuned = PCA(n_components=5000)
x_dense = tfidf_m.toarray()
pca_tuned.fit(x_dense)
x = pca_tuned.transform(x_dense)
print(x.shape)
```

```
(8807, 5000)
```

Figure 12: Dimensionality Reduction usingn PCA

2. Data Wrangling

The process of cleaning, manipulating, and organizing raw data to make it ready for analysis is called data wrangling, which is also referred to as data munging. It starts with gathering data from many sources, such as databases and APIs, and then cleaning it to fix errors, handle missing numbers, and get rid of duplicates. After that, the data is cleaned up by aggregating and normalizing it, added new fields or merged existing ones, and then quality- and accuracy-checked. After being cleansed and converted, the data is saved for analysis, enhancing its consistency, quality, and usefulness. Data wrangling saves time and promotes well-informed decision-making by improving comprehension and supporting sophisticated analysis methods like machine learning.

2.1 Merging Dataset

```
# Merging both the datasets using merge.
df = pd.merge(left=movies, right=imdb, left_index=True, right_index=True)
df.columns
```

```
Index(['show_id', 'type', 'title', 'director', 'cast', 'country', 'date_added',
       'release_year', 'rating', 'duration', 'listed_in', 'description',
       'tconst', 'averageRating', 'numVotes'],
      dtype='object')
```

Figure 13: Merging Datasets

The code loads Netflix titles from a CSV file into a DataFrame named "movies" and IMDb ratings from a TSV file into another DataFrame called “imdb” Using Pandas, it merges these DataFrames based on their indexes, facilitating seamless data manipulation and analysis.

2.2 Feature Extraction

```
from dateutil import parser
df['date_added'] = df['date_added'].apply(lambda x: parser.parse(x, fuzzy=True) if pd.notnull(x) else None)
df['data_added_day'] = df['date_added'].dt.day
df['data_added_month'] = df['date_added'].dt.month
df['data_added_year'] = df['date_added'].dt.year
df.columns
```

```
Index(['show_id', 'type', 'title', 'director', 'cast', 'country', 'date_added',
       'release_year', 'rating', 'duration', 'listed_in', 'description',
       'tconst', 'averageRating', 'numVotes', 'data_added_day',
       'data_added_month', 'data_added_year'],
      dtype='object')
```

Figure 14: Feature Extraction

This piece of code isolates date elements from the “date_added” column of a DataFrame and generates fresh columns to hold them. Moreover, it computes the duration between the “data_added_month” and the month obtained from the “release_year” column. These adjustments facilitate the visualization of time series data.

2.3 Data Cleaning

```
# Text cleaning function
def clean_text(text):
    text = str(text).lower()
    text = re.sub('\[.*?\]', '', text)
    text = re.sub('https?://\S+|www\.\S+', '', text)
    text = re.sub('<.*?>+', '', text)
    text = re.sub('[%s]' % re.escape(string.punctuation), '', text)
    text = re.sub('\w*\d\w*', '', text)
    tokens = word_tokenize(text)
    stop_words = set(stopwords.words('english'))
    filtered_tokens = [word for word in tokens if word not in stop_words]
    lemmatizer = WordNetLemmatizer()
    lemmatized_tokens = [lemmatizer.lemmatize(word) for word in filtered_tokens]
    return ' '.join(lemmatized_tokens)
```

Figure 15: Data Cleaning

The “clean_text” function tokenizes text after converting it to lowercase, eliminating punctuation, HTML tags, URLs, and alphanumeric characters in preparation for analysis. For consistency in subsequent tasks like recommendation systems and natural language processing, it additionally eliminates stop words and lemmatizes terms. This improves the quality of the data, increasing efficiency and accuracy. By applying “clean_text,” combining textual features into “combined_features,” and displaying the preprocessed text, the code helps analysis by providing standardized, cleaner data that is ready for additional processing.

3. Descriptive Analytics

Descriptive analytics delves into historical data to discern patterns, trends, and correlations. Its aim is to summarize past events or behaviours, not to forecast future results. This analysis sheds light on past occurrences, aiding in the identification of crucial performance metrics, comprehension of customer behaviour, and evaluation of business performance. It serves as a cornerstone for advanced analytics methods like predictive and prescriptive analytics. By comprehending historical data, organizations can make informed decisions, refine processes, and enhance overall performance. Here is the analysis based on the DataFrame “df”:

3.1 Distribution of Movies and TV shows

This code generates a customized pie chart illustrating the distribution of movies versus TV shows in a DataFrame, with “Plotly Express” automatically calculating the percentage of each category based on their frequencies in the 'type' column, showing 69.7% for movies and 30.3% for TV shows (Figure 16).

3.1 Distribution of Movies and TV Shows

[19]:
```
# Define custom colors
custom_colors = ['#0096df', '#1cc4e7']  # Blue and orange

# Create pie chart with custom colors
fig = px.pie(df, names='type', width=600, height=400, color_discrete_sequence=custom_colors)
fig.update_layout(title='Distribution of Movies vs TV Shows', title_x=0.5)

# Show the plot
fig.show()
```

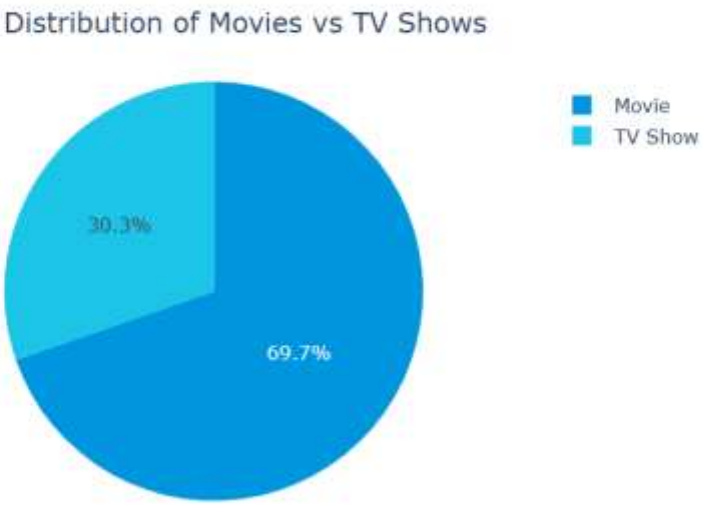

Figure 16: Distribution of Movies vs Tv shows

3.2 Release year Trend

This code snippet uses Seaborn to visualize release year trends for movies and TV shows. It uses "sns.histplot()" to create histograms displaying the distribution of release years for each type, with "plt.legend()" adding a color-coded legend. Notably, the period from 2019 to 2021 saw the highest number of releases, as shown by the peaks in the histograms.

```
# Plotting the trends in release years
plt.figure(figsize=(12, 6))
sns.histplot(df[df['type'] == 'Movie']['release_year'], kde=True, color='blue', label='Movies', bins=30)
sns.histplot(df[df['type'] == 'TV Show']['release_year'], kde=True, color='red', label='TV Shows', bins=30)
plt.title('Release Year Trends for Movies and TV Shows')
plt.xlabel('Release Year')
plt.ylabel('Frequency')
plt.legend()
plt.show()
```

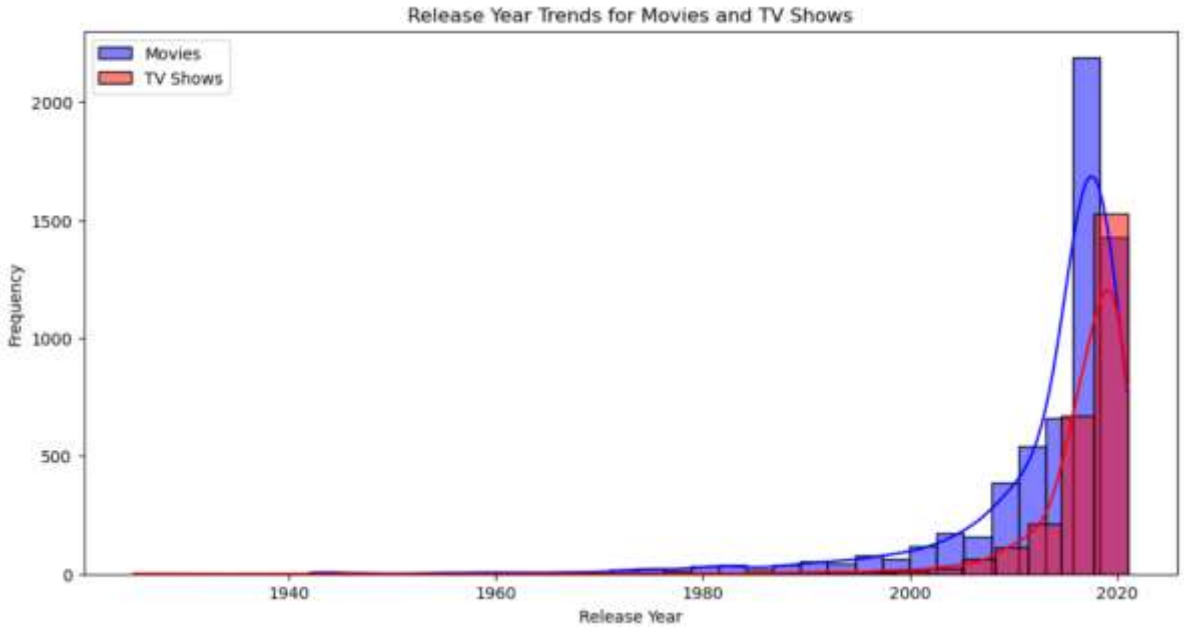

Figure 17: Release year trends for Movies and TV shows

3.3 Country

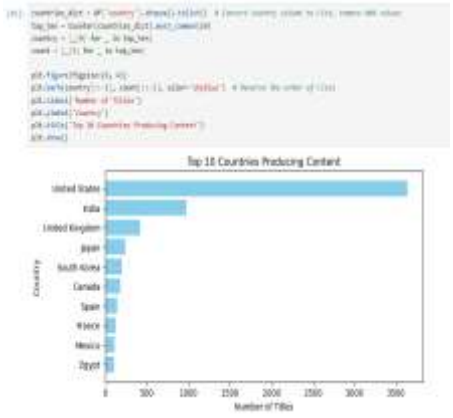

Figure 18: Top 10 Countries

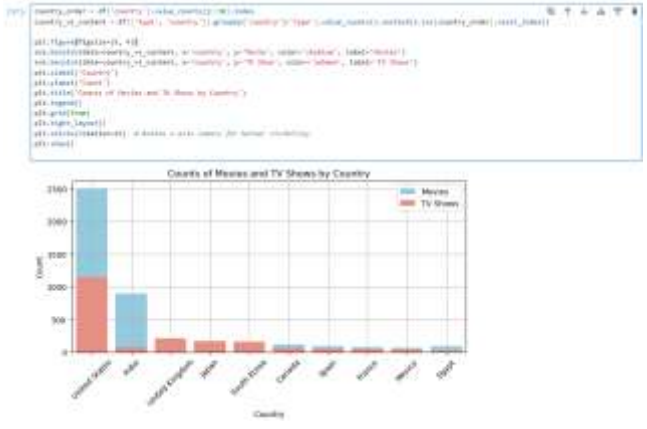

Figure 19: Content vs Country

Figures 18 and 19 showcase content production trends by country. Utilizing functions like “Counter()” and “plt.barh()”, Figure 18, highlights the top 10 content-producing nations. Meanwhile, Figure 18, employing “groupby()” and “sns.barplot()”, reveals the dominance of the United States in content creation, with India and the UK also significant contributors. Overall, these figures emphasize the global landscape of content creation, with distinct patterns observed in different regions.

3.4 Genera

This code first splits the “listed_in” column into individual genres, then counts the occurrences of each genre using “pd.Series().value_counts()”. It selects the top 10 genres and creates a horizontal bar plot using “sns.barplot()” to visualize the results. Notably, it presents international movies at the top, followed by dramas and comedies.

```
[62]: # Split and count genres
genres = df['listed_in'].str.split(',').apply(lambda x: [i.strip() for i in x])
all_genres = [genre for sublist in genres for genre in sublist]
genre_counts = pd.Series(all_genres).value_counts().head(10)
# Plotting the most common genres
plt.figure(figsize=(8, 6))
sns.barplot(x=genre_counts.values, y=genre_counts.index, palette="coolwarm", orient='horizontal')
plt.title('Top 10 Most Common Genres on Netflix')
plt.xlabel('Number of Titles')
plt.ylabel('Genre')
plt.show()
```

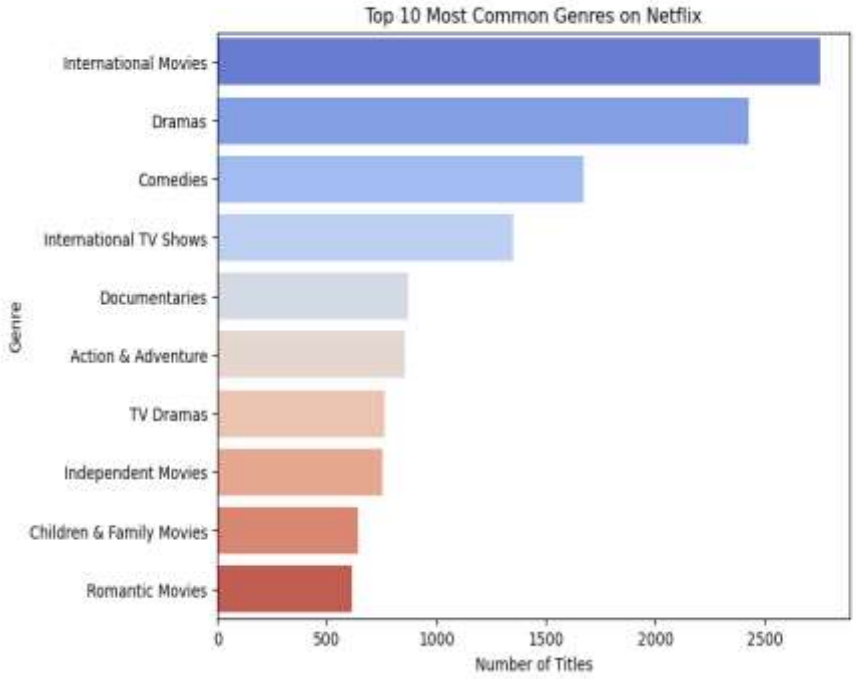

Figure 20: Top 10 Genera

3.5 Age Rating

The below code snippet creates a histogram to visualize the frequency of content ratings in the dataset. It uses the “value_counts()” function to generate a frequency table and sorts it with “sort_index()”. The “plt.bar()” function is then used to plot the data. The “plt.xticks(rotation=45)” function rotates the x-axis labels for better readability. The most common content rating is 'TV-MA', followed by 'TV-14'.

```
[60]: # Assuming 'df' is your DataFrame with the dataset, and 'rating' is the column containing content ratings
# Create a frequency table for content ratings
rating_freq = df['rating'].value_counts().sort_index()
# Plot a histogram for content ratings
plt.figure(figsize=(8, 4))
plt.bar(rating_freq.index, rating_freq.values, color='skyblue')
plt.title('Frequency of Content Ratings')
plt.xlabel('Content Rating')
plt.ylabel('Frequency')
plt.xticks(rotation=45)  # Rotate x-axis labels for better readability if needed
plt.grid(axis='y', linestyle='--', alpha=0.5)  # Add grid lines for y-axis
plt.show()
```

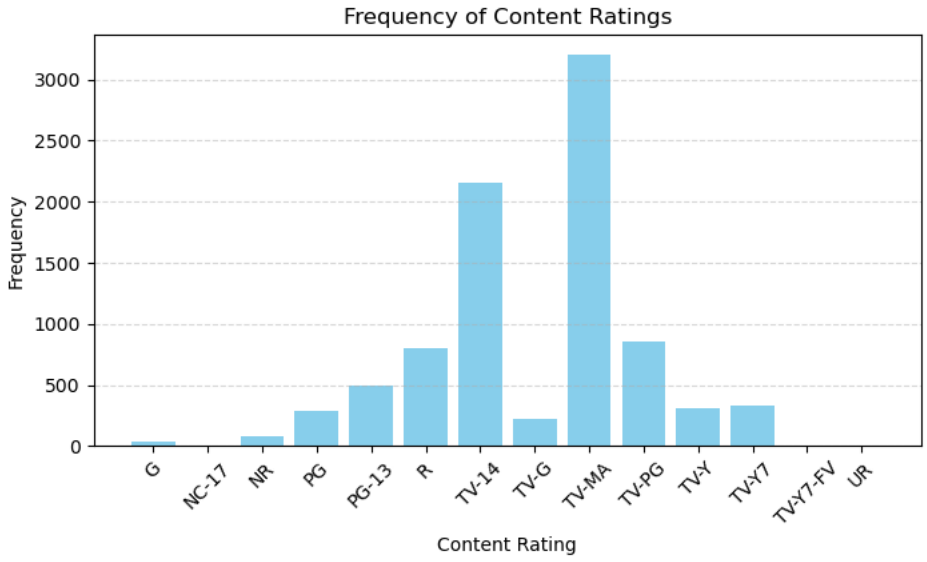

Figure 21: Frequency of Content Rating

3.6 Director

```
[79]: # Top directors in the dataset
top_directors = data['director'].dropna().value_counts().head(10)
plt.figure(figsize=(12, 8))
sns.barplot(x=top_directors.values, y=top_directors.index, palette='coolwarm')
plt.title('Top 10 Directors')
plt.xlabel('Count')
plt.ylabel('Director')
plt.show()
```

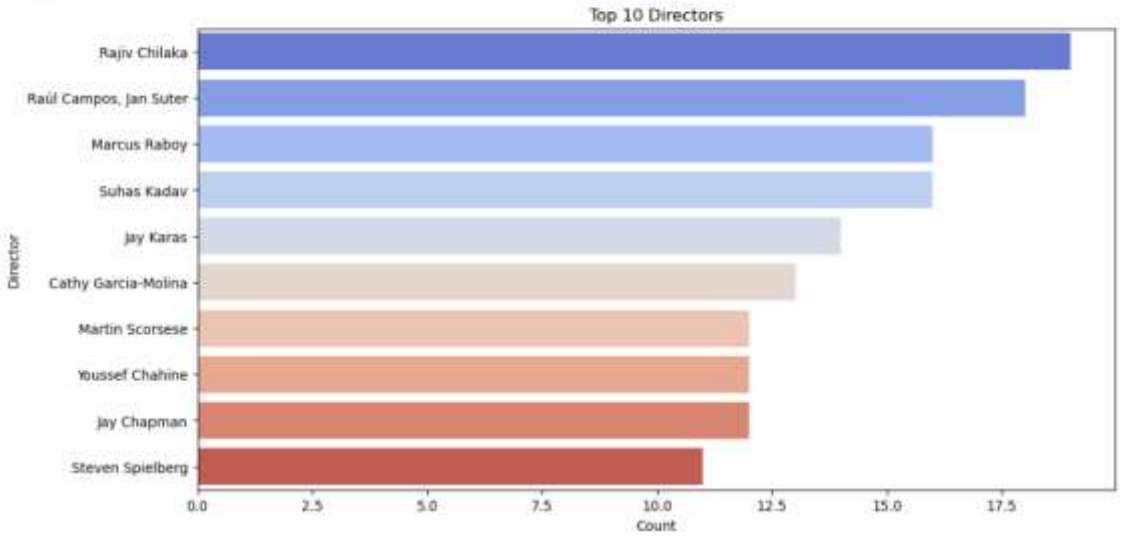

Figure 22: Top 10 Directors

This code snippet generates a horizontal bar plot showing the top 10 directors based on the count of movies or TV shows they have directed. In the plot, Rajiv Chilaka, is in the top which have directed highest number of the movies.

4. Diagnostic Analytics

Diagnostic analysis is the process of examining data to identify the main causes of problems, anomalies, or patterns. To understand the reasons behind outcomes, it involves examining the linkages, patterns, and factors impacting observable events. Diagnostic analysis provides a deeper understanding of the causes of problems or successes, enabling companies to take well-informed decisions and exact action to address issues or take advantage of fortunate circumstances.

4.1 Trend of adding movies over day

```
[23]: # Grouping data by type and day of addition, then counting the occurrences
type_counts = df.groupby(['type', 'data_added_day']).size().reset_index(name='count')
# Filter data for movies and TV shows separately
movies_data = type_counts[type_counts['type'] == 'Movie']
tv_shows_data = type_counts[type_counts['type'] == 'TV Show']
# Plotting
plt.figure(figsize=(8, 4))
sns.barplot(data=movies_data, x='data_added_day', y='count', color='skyblue', label='Movies')
sns.barplot(data=tv_shows_data, x='data_added_day', y='count', color='salmon', label='TV Shows')
plt.title('Trends of Adding Movies and TV Shows Over Days In Different Years')
plt.xlabel('Day of Year')
plt.ylabel('Number of Titles Added')
plt.legend(title='Type', loc='upper right')
plt.grid(True)
plt.tight_layout()
plt.show()
```

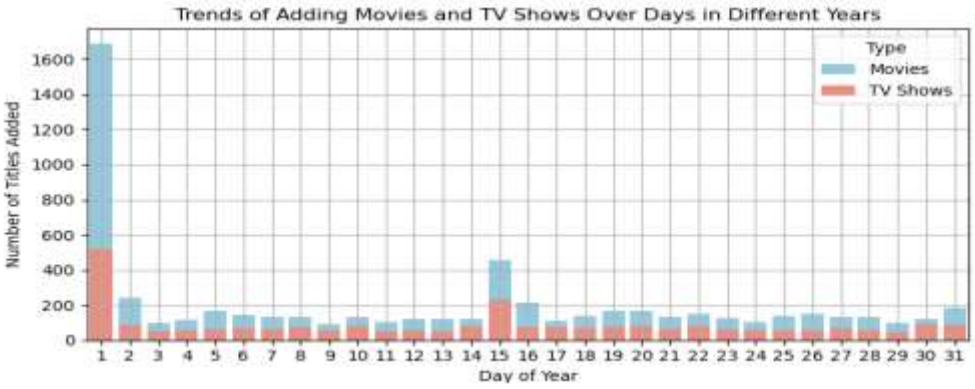

Figure 23: Trend of adding movies over days

In above Figure 23, presents diagnostic analysis functions focusing on movie and TV show additions. “group_data_by_day_and_type()” categorizes data by type and addition day. “filter_movies_data()” and “filter_tv_shows_data()” respectively filter movie and TV show data. The “plot_additions_trends()” function utilizes seaborn for trend visualization, highlighting important days for new content additions. Noteworthy patterns indicate new content often being uploaded at the beginning or middle of a month. Many viewers receive their pay checks around this time, leading to increased disposable income for entertainment subscriptions or rentals.

4.2 Annual Trends in Netflix Content Additions

Figure 24 shows that , how the number of Netflix titles has changed over the years, using the “data_added_year” column to track trends. By using seaborn's lineplot function, it creates a

visual representation of these trends (Figure 24). The analysis shows a steady increase in titles up until 2018, followed by a significant surge from 2018 to 2020, suggesting a major expansion of Netflix's content library during this time. However, after 2020, there's a noticeable decrease in additions, which could indicate shifts in how content is acquired or changes in the market. Additionally, the code separates out the yearly counts for movies and TV shows, revealing consistent growth in both categories since 2015, with notable peaks in 2019 and 2020. However, there's a clear drop in additions for both types in 2021, likely due to data limitations or other external factors. This underscores the importance of thorough diagnostic analysis in understanding how content trends evolve over time.

```
[49]: # Count the number of entries and group by year and type (movie/TV show).
type_counts = df.groupby(['type', 'data_added_year']).size().reset_index(name='count')
# Calculating the total count of entries per year
total_data = type_counts.groupby('data_added_year')['count'].sum().reset_index()
# Separately filter data for TV series and movies.
movies_data = type_counts[type_counts['type'] == 'Movie']
tv_shows_data = type_counts[type_counts['type'] == 'TV Show']
# Plotting
plt.figure(figsize=(6, 3))
# Plot line graph for movies data
sns.lineplot(data=movies_data, x='data_added_year', y='count', marker='o', label='Movies')
# Plot line graph for TV shows data
sns.lineplot(data=tv_shows_data, x='data_added_year', y='count', marker='o', label='TV Shows')
# Plot line graph for total data (movies and TV shows combined)
sns.lineplot(data=total_data, x='data_added_year', y='count', marker='o', label='Movies and TV Shows')
# Set plot title and axis labels
plt.title('Trends of Adding Movies and TV Shows Over Time')
plt.xlabel('Year')
plt.ylabel('Number of Titles Added')
# Add legend and grid
plt.legend(title='Year')
plt.grid(True)
# Display the plot
plt.show()
```

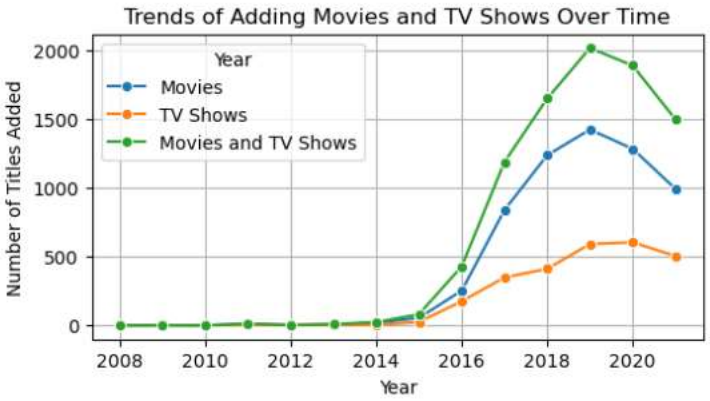

Figure 24:Annual Trends in Netflix Content Additions

4.3 Hypothesis testing

In the below figure the script compares the IMDb ratings of TV series and movies using a t-test, using the “ttest_ind()” function from the “scipy.stats” module. It determines whether there is a noteworthy variation in the ratings. The T-statistic of 0.769 and a p-value of 0.442 suggest that there's no significant difference in IMDb ratings between movies and TV shows. This implies we fail to reject the null hypothesis, indicating that both types of content receive similar ratings.

```
[113]: from scipy.stats import ttest_ind

# Extract IMDb ratings for movies and TV shows
movies_ratings = df[df['type'] == 'Movie']['averageRating']
tvshows_ratings = df[df['type'] == 'TV Show']['averageRating']

# Drop any missing values in the ratings
movies_ratings.dropna(inplace=True)
tvshows_ratings.dropna(inplace=True)

# Perform a t-test
t_stat, p_value = ttest_ind(movies_ratings, tvshows_ratings, equal_var=False)

print(f"T-statistic: {t_stat}")
print(f"P-value: {p_value}")

# Hypothesis testing result
alpha = 0.05  # significance level

if p_value < alpha:
    print("Reject the null hypothesis - There is a significant difference in IMDb ratings between movies and TV shows.")
else:
    print("Fail to reject the null hypothesis - There is no significant difference in IMDb ratings between movies and TV shows.")
```

```
T-statistic: 0.7692603043716981
P-value: 0.4417741488145337
Fail to reject the null hypothesis - There is no significant difference in IMDb ratings between movies and TV shows.
```

Figure 25: Hypothesis Testing

5. Predictive Analytics

Predictive analysis is a branch of data analytics that uses statistical techniques, machine learning algorithms, and historical data to estimate the likelihood of future events based on past performance. Its main goal is to predict events, patterns, and behaviour in the future and provide insights that help organizations make wise decisions (Kelleher, Mac Namee and D'Arcy, 2020).

Recommendation System

A recommendation system is a filtering mechanism that assesses data from diverse users and sources to anticipate their interests, providing relevant products to the right audience. Conversely, the recommendation system is a machine learning algorithm that proposes items users may enjoy, drawing from their previous selections. These systems employ two distinct methodologies, Collaborative filtering and Content based Filtering (Aggarwal, 2016).

Content-Based Filtering

The recommendation technique which makes suggestions for content that are like ones the consumer has previously liked is known as content-based Filtering. In the code, Content-based filtering for movie recommendations suggests similar movies based on the attributes of a given movie. In content-based recommendation systems, two methods used are K-means clustering and cosine similarity.

1. K-Means Clustering

K-means clustering is a method that groups items with similar attributes together (Altexsoft, 2021). In movie recommendations, it categorizes films into clusters based on features like genres, actors, directors, and themes. Each cluster comprises movies with comparable characteristics.

```
58]: # K-means clustering
     k = 200
     kmeans = MiniBatchKMeans(n_clusters=k)
     kmeans.fit(x)
     rec_data['cluster'] = kmeans.predict(x)
```

Figure 26: K-means CLustering

In the initial stage, the above code performs K-means clustering with 200 clusters using the “MiniBatchKMeans” algorithm. It fits the data “(tfidf_m)”, which is vectorized using TFIDF vectorization, to the clusters and assigns each data point to a cluster. The cluster assignments are then stored in the 'cluster' column of the “rec_data dataframe”.

Elbow Method: Determining the Ideal Number of Clusters (K)

```
[29]: from sklearn.cluster import KMeans

      # Create a list to store the sum of squared errors for each K value
      Sum_of_Squared_Errors = []

      # Iterate over range of K values and compute SSE for each value
      for k in range(1, 10):
          # Initialize the k-means model with the current value of K
          kmeans = KMeans(n_clusters=k, init='k-means++', random_state=42)
          # Fit the model to the data
          kmeans.fit(tfidf_m)
          # Compute the sum of squared errors for the model
          Sum_of_Squared_Errors.append(kmeans.inertia_)

      # Plot the SSE values against the range of K values
      plt.plot(range(1, 10), Sum_of_Squared_Errors)
      plt.title('Elbow Method for K-Means Clustering')
      plt.xlabel('Number of clusters (K)')
      plt.ylabel('Sum of Squared Errors (SSE)')
      plt.show()
```

Figure 27: Elbow Method

This code snippet uses the Elbow Method to find the optimal number of clusters for K-means clustering. It initializes "KMeans" models for K values from 1 to 9 with 'k-means' initialization and a fixed random state. Each model is fitted to the data, and the sum of squared errors (SSE)

is calculated and stored in a list called "Sum_of_Squared_Errors". The resulting graph helps determine the ideal number of clusters at the elbow point in the SSE curve.

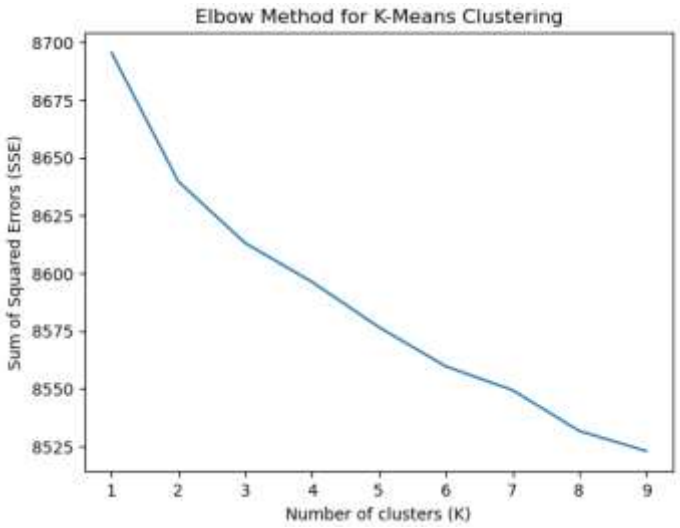

Figure 28: Elbow Method Graph

Clustering Evaluation:

In this section Three metrics are used in the code to assess the quality of a K-means clustering model: the Silhouette Score, Davies-Bouldin Index, and Calinski-Harabasz Index. “evaluate_clustering_model” function accepts the clustering model, the data, and the cluster labels as inputs, calculates the three metrics, and displays the results. As in the above figure the result shows that the model used 9 clusters for k means clustering, silhouette score is 0.0033, Calinski-Harabasz Score is 24.0414 and the davies-Bouldin score is 11.7240.

```
]: from sklearn.metrics import silhouette_score, davies_bouldin_score, calinski_harabasz_score

def evaluate_clustering_model(model, data, labels):
    silhouette = silhouette_score(data, labels)
    davies_bouldin = davies_bouldin_score(data, labels)  # Convert to dense array
    calinski_harabasz = calinski_harabasz_score(data, labels)  # Convert to dense array
     # Print the evaluation results
    print(f"Number of clusters: {k}")
    print(f"Silhouette score: {silhouette:.4f}")
    print(f"Calinski-Harabasz score: {calinski_harabasz:.4f}")
    print(f"Davies-Bouldin score: {davies_bouldin:.4f}")

    return {
        "Silhouette Score": silhouette,
        "Davies-Bouldin Index": davies_bouldin,
        "Calinski-Harabasz Index": calinski_harabasz
    }
```

```
]: y_kmeans = kmeans.predict(x)# Predict on the model
labels = kmeans.labels_     # Get the cluster labels for each point in the data
unique_labels = np.unique(labels) # Get the unique cluster labels
df['kmeans_cluster'] = labels
scores_dict_kmeans = evaluate_clustering_model(kmeans, x , y_kmeans)
```

```
Number of clusters: 9
Silhouette score: 0.0033
Calinski-Harabasz score: 24.0414
Davies-Bouldin score: 11.7240
```

Figure 29: Clustering Evaluation

Plotting the Clusters

```
plt.figure(figsize=(8, 6), dpi=120)
for i in unique_labels:
    plt.scatter(x[labels == i, 0], x[labels == i, 1], s=20, label='Cluster {}'.format(i))
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s=100, marker='x', c='black', label='Cluster centers')
plt.title('KMeans clustering with {} clusters'.format(len(unique_labels)))
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.show()
```

Figure 30: Plotting clusters Code

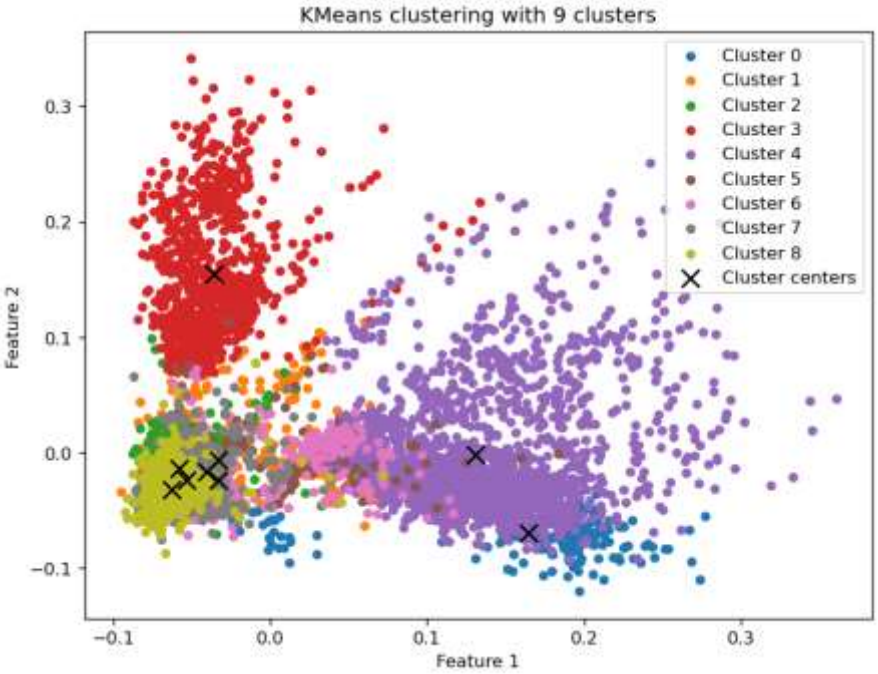

Figure 31: Plotted clusters graph

Above code creates a scatter plot to display “KMeans” clustering results. It starts by setting up a figure, then loops through each unique cluster label to plot data points in distinct colours for each cluster. Cluster centres are highlighted with black 'x' markers. The plot includes the number of clusters in the title, labelled axes for clarity, and a legend to distinguish between the clusters and their centres, followed by displaying the plot as shown in Figure 31.

2. Cosine Similarity using TF-IDF Vectorization:

In Natural Language Processing, cosine similarity evaluates the likeness between two texts by analysing their vector representations and computing the cosine of the angle between these vectors in a multi-dimensional space. A score of 1 indicates identical orientation, while scores closer to 0 signify lesser resemblance between the documents. Additionally, TF-IDF measures a word's significance within a document collection by factoring in its frequency within a document and its scarcity across the collection. Widely applied in NLP and automated text analysis, TF-IDF amplifies words frequently appearing in a document while diminishing the importance of terms occurring in many documents.

```
[63]: cosine_sim = cosine_similarity(tfidf_m, tfidf_m)
      cosine_sim.shape

[63]: (8790, 8790)
```

Figure 32: Applying Cosine Similarity

This code computes cosine similarity between all document pairs in a corpus using TF-IDF vectors. It yields a similarity matrix, with “cosine_sim.shape” indicating the corpus size. This is essential for various NLP tasks like information retrieval and clustering.

```python
def get_recommendations(title, cosine_sim):
    # Convert title to lowercase for case-insensitive matching
    title = title.lower()
    # Get the index of the movie that matches the title
    idx = rec_data.index[rec_data['title'].str.lower() == title].tolist()
    if not idx:
        raise ValueError(f"No movie with the title '{title}' found.")
    idx = idx[0]
    # Get the pairwise similarity scores of all movies with that movie
    sim_scores = list(enumerate(cosine_sim[idx]))
    # Sort the movies based on the similarity scores
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    # Get the scores of the 10 most similar movies
    sim_scores = sim_scores[1:6]
    # Get the movie indices
    movie_indices = [i[0] for i in sim_scores]
    # Return the top 10 most similar movies
    return rec_data['title'].iloc[movie_indices]
```

Figure 33: Recommendation using cosine

This function, “get_recommendations”, takes a movie title and a cosine similarity matrix as inputs to find the top 5 movies similar to the given title. It converts the title to lowercase for case-insensitive matching and searches for its index in the dataset. If not found, a “ValueError” is raised. Once the index is located, it retrieves similarity scores from the cosine similarity matrix, sorts them, and selects the top 5 most similar movies (excluding the input movie). The function then returns their titles from the dataset.

3.Random Forest Classifier:

```python
# Train Random Forest Classifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
rf_classifier.fit(X_train, y_train)
```

```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

Figure 34: Random Forest Classifier

In Netflix movies and TV shows, a Random Forest Classifier predicts the best month to release content based on various features. This ensemble learning algorithm builds multiple decision trees using random subsets of features and data, then aggregates their predictions for reliable and accurate results. It effectively handles high-dimensional data and reduces overfitting. The

algorithm ensures repeatability with a random state of 42 by initializing and training a Random Forest Classifier with 100 trees.

Evaluating Model

The Classification Report provides key performance metrics for each class in a classification model. It comprises support (number of examples), F1-score (harmonic mean of precision and recall), recall (ability to recognize positive instances), and precision (accuracy of positive predictions) for each class. The accuracy metric represents overall model performance across all classes, with macro and weighted averages offering insights into overall performance. In the figure 35, By contrasting the expected labels "(y_pred)" with the actual labels "(y_test)" of the test data "(X_test)", it produces a classification report to assess the effectiveness of the Random Forest Classifier (Atwan, 2022).

```
# Evaluate model
y_pred = rf_classifier.predict(X_test)
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

```
Classification Report:
              precision    recall  f1-score   support

         1.0       0.98      0.99      0.99       168
         2.0       0.94      0.97      0.95       155
         3.0       0.84      0.83      0.84       176
         4.0       0.70      0.74      0.72       162
         5.0       0.77      0.78      0.77       154
         6.0       0.82      0.84      0.83       161
         7.0       0.82      0.77      0.79       167
         8.0       0.86      0.82      0.84       174
         9.0       0.92      0.87      0.90       162
        10.0       0.94      0.97      0.96       156
        11.0       0.96      0.96      0.96       172
        12.0       0.96      0.98      0.97       176

    accuracy                           0.88      1983
   macro avg       0.88      0.88      0.88      1983
weighted avg       0.88      0.88      0.88      1983
```

Figure 35: Model Evaluation using Random Forest Classifier

6. Recommendations and Conclusions

6.1 Recommendation using k means clustering

The “get_recommendation” function provides movie recommendations based on a given title. It finds the movie's index in the “rec_data” DataFrame, which is generated using K-means clustering, while handling case insensitivity. After extracting the movie's cluster label, it filters movies in the same cluster and calculates cosine similarities using “linear_kernel”. It sorts these movies by similarity and computes distances for the top 10, returning the titles and

distances of the top 5 similar movies. If the input movie title is not found, it prints an error message and returns None. The code prompts the user for a movie name and uses “get_recommendation” to display top recommendations or an absence message.

```
from sklearn.metrics.pairwise import linear_kernel
from sklearn.metrics import pairwise_distances


def get_recommendation(root):
    try:
        # Find the index of the movie in the DataFrame
        idx = rec_data[rec_data['title'].str.lower() == root.lower()].index
        if idx.empty:
            raise ValueError(f"No movie with the title '{root}' found.")
        idx = idx[0]

        # Get the cluster of the movie
        cluster = rec_data.loc[idx, 'cluster']

        # Filter movies from the same cluster
        cluster_movies = rec_data[rec_data['cluster'] == cluster]

        # Calculate pairwise cosine similarities within the cluster
        cosine_similarities = linear_kernel(tfidf_m[idx:idx+1], tfidf_m[cluster_movies.index]).flatten()

        # Get the indices of movies sorted by similarity (excluding the input movie)
        related_docs_indices = [i for i in cosine_similarities.argsort()[::-1] if i != 0]

        # Get the distances for the top similar documents
        distances = pairwise_distances(tfidf_m[idx:idx+1], tfidf_m[cluster_movies.index[related_docs_indices[:10]]], metric='cosine').flatten()

        # Return both the titles and distances
        return cluster_movies.iloc[related_docs_indices[:5]]['title'], distances
    except ValueError as e:
        print(e)
        return None, None
```

Figure 36: Recommendation using K-Means

```
# Input movie name
name = input("Please enter a movie name: ")

# Example usage
result1, distances1 = get_recommendation(name)

if result1 is not None and distances1 is not None:
    print(f"Top recommendations for '{name}':")
    for i, (title, distance) in enumerate(zip(result1, distances1), 1):
        print(f"{i:>2}. {title:<30} Distance: {distance:.4f}")
else:
    print(f"No recommendations found for '{name}'.")
```

```
Please enter a movie name:  pk
Top recommendations for 'pk':
 1. PK                             Distance: 0.0000
 2. Dil Chahta Hai                 Distance: 0.8402
 3. Main Hoon Na                   Distance: 0.8714
 4. Chance Pe Dance                Distance: 0.8718
 5. Ek Main Aur Ekk Tu             Distance: 0.8720
```

Figure 37: Output

6.2 Movie recommendation using Cosine sim:

```
name_of_mov = input("Please enter a movie name: ")
# Example usage
try:
    recommendations = get_recommendations(name_of_mov, cosine_sim)
    print(f"Top recommendations for {name_of_mov}")
    for i, rec in enumerate(recommendations, 1):
        print(f"{i}. {rec}")
except ValueError as e:
    print(e)
```

```
Please enter a movie name:  pk
Top recommendations for pk
1. 3 Idiots
2. Sanju
3. Taare Zameen Par
4. Andaz Apna Apna
5. Dil Chahta Hai
```

Figure 38: Recommendation using Cosine sim

This code prompts the user to input a movie name and retrieves recommendations based on it using the “get_recommendations” function. If recommendations are found, it displays them with their indices. If no movie is found with the provided name, it raises a “ValueError” and prints an error message.

6.3 Predicting best month to release movie using RandomForest classifier:

```
# Define function to take input and make predictions
def get_user_input():
    # Get input from the user
    title = input("Enter the title: ")
    director = input("Enter the director: ")
    cast = input("Enter the cast: ")
    listed_in = input("Enter the listed in: ")
    description = input("Enter the description: ")
    content_type = input("Enter the content type (Movie/TV Show): ")
    country = input("Enter the country: ")
    release_year = int(input("Enter the uploaded Year: "))
    rating = input("Enter the rating: ")
    duration = input("Enter the duration: ")

    # Call the function to make predictions
    make_prediction(title, director, cast, listed_in, description, content_type, country, release_year, rating, duration)

# Call the function to take input and make predictions
get_user_input()
```

```
Enter the title:  Iron Man
Enter the director:  james
Enter the cast:  james bond, ranjnikant
Enter the listed in:  comedy, thriller
Enter the description:  this movie is the beautiful movie having a iron man
Enter the content type (Movie/TV Show):  movie
Enter the country:  United Kingdom
Enter the uploaded Year:  2024
Enter the rating:  TV-MA
Enter the duration:  90 min
According to the given data, the best month to release new content is: January
```

Figure 39: Predicting Using Random Forest Classifier

The below code predicts the best month using trained machine learning model (Random Forest Classifier). As shown in the above code, we first take input from the user for several features. Once the user provides this information, the “make_prediction()” function is invoked. This function utilizes the input data along with the trained model and TF-IDF vectorizer to predict the optimal month for releasing new content. The prediction is based on a combination of content features and historical data, enabling informed decision-making regarding content release timing.

6.4 Top 10 movies using IMDB Rating

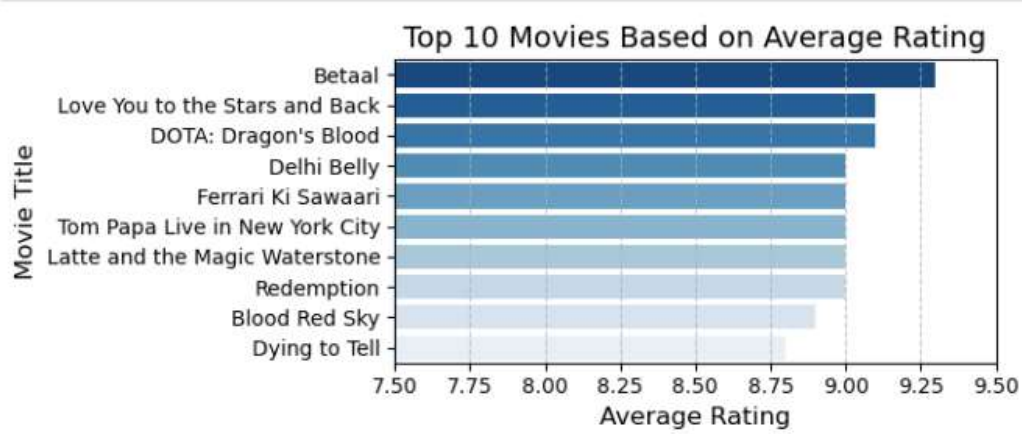

Figure 40: Top 10 Movies using IMDb Rating

The above figure shows that the top 10 movies based on the IMDB Rating in the Netflix Movies and TV shows dataset. To get this result first the dataset is sorted based on the average rating in descending order to ensure that the highest-rated movies appear at the top. Then, the top 10 movies with the highest average ratings are selected from the sorted dataset. In the process, we discovered that the film titled “Betaal” holds the highest rating, closely followed by “Love You to the Stars and Back”.

6.5 Best month to Release by number of movies release by month

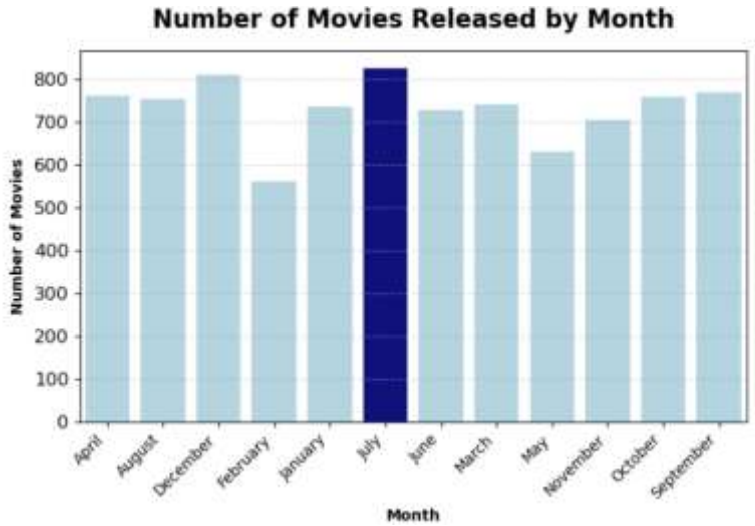

Figure 41: Best month to release new content

The provided visualization illustrates the pattern of movie releases over different months. It reveals that July had the highest number of movie releases compared to other months. This insight suggests a potential trend or preference in the streaming platform's content release strategy, highlighting July as a particularly busy month for new movie arrivals.

References

Klarity Analytics. (2015). *Dimensions of Big Data*. [online] Available at: https://www.klarity-analytics.com/2015/07/27/dimensions-of-big-data/.

Ramos, J. (2020). *Big Data File Formats Explained*. [online] Medium. Available at: https://towardsdatascience.com/big-data-file-formats-explained-dfaabe9e8b33.

Botelho, B. and Bigelow, S.J. (2022). *What is Big Data and Why is it Important?* [online] SearchDataManagement. Available at: https://www.techtarget.com/searchdatamanagement/definition/big-data.

Aggarwal, C.C. (2016). *Recommender Systems*. Cham Springer International Publishing Imprint: Springer.

Kelleher, J.D., Mac Namee, B. and D'Arcy, A. (2020). *Fundamentals of machine learning for predictive data analytics : algorithms, worked examples, and case studies*. Cambridge, Massachusetts: The MIT Press.

Altexsoft (2021). *Unsupervised Learning: Algorithms and Examples*. [online] AltexSoft. Available at: https://www.altexsoft.com/blog/unsupervised-machine-learning/.

Atwan, T. A. (2022*). Time Series Analysis with Python Cookbook: Practical Recipes for Exploratory Data Analysis, Data Preparation, Forecasting, and Model Evalua*tion. Packt Publishing.