Box Office: Predicting Movie Revenue with Data Science

Yuwenqian Chen, Yuxuan Weng, Senhui Zhao

Group Member

Yuxuan Weng

demonstrated exceptional coding skills throughout the project, taking the lead in implementing various machine learning models. His proficient coding abilities allowed them to efficiently translate complex algorithms into functional and optimized code.

Yuwenqian Chen

demonstrated strong analytical skills and attention to detail by crafting a well-structured and insightful conclusion for the project. Yuwenqian also took the lead in creating the presentation slides, effectively summarizing the key findings.

Senhui Zhao

made a valuable contribution to the team by meticulously cleaning and organizing the data, ensuring its integrity and quality. Senhui also played a pivotal role in preparing the presentation slides, distilling complex information into concise and visually appealing slides that effectively communicated key project aspects to audience.

Background

This project aims to predict movie revenue by exploring various factors, including cast, crew, plot keywords, budget, posters, release dates, languages, production companies, and countries. The project involves conducting exploratory data analysis to identify useful features, counting the number of movies released by day of the week, month, and year, identifying movie genre trend shifting patterns, and integrating external datasets. Three models will be built, trained on the training set, and evaluated on the test set using accuracy metrics such as Residual Standard Error. The dataset will be split further to include a cross-validation set to improve the model's performance.

Data Munging

- Load Data
- Data Exploration
- Clean Data
  - Merge datasets
  - Handle missing value
  - Handle duplicate value
  - Handle outlier
- Parse Json
  - Extract information from merged data frame

Load Data

```
movie_fdr_path = '/content/drive/MyDrive/movie'
credit_df = pd.read_csv(os.path.join(movie_fdr_path, 'tmdb_5000_credits.csv'))
movie_df = pd.read_csv(os.path.join(movie_fdr_path, 'tmdb_5000_movies.csv'))
```

Load two csv datasets:

- tmdb_5000_credits.csv
- tmdb_5000_movies.csv

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4803 entries, 0 to 4802
Data columns (total 20 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   budget                4803 non-null   int64  
 1   genres                4803 non-null   object 
 2   homepage              1712 non-null   object 
 3   id                    4803 non-null   int64  
 4   keywords              4803 non-null   object 
 5   original_language     4803 non-null   object 
 6   original_title        4803 non-null   object 
 7   overview              4800 non-null   object 
 8   popularity            4803 non-null   float64
 9   production_companies  4803 non-null   object 
 10  production_countries  4803 non-null   object 
 11  release_date          4802 non-null   object 
 12  revenue               4803 non-null   int64  
 13  runtime               4801 non-null   float64
 14  spoken_languages      4803 non-null   object 
 15  status                4803 non-null   object 
 16  tagline               3959 non-null   object 
 17  title                 4803 non-null   object 
 18  vote_average          4803 non-null   float64
 19  vote_count            4803 non-null   int64  
dtypes: float64(3), int64(4), object(13)
memory usage: 750.6+ KB
```

Explore Movies dataset

```
movie_df.info()
```

According to the output, we found out that there are

20 variables and 4803 observations

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4803 entries, 0 to 4802
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   movie_id   4803 non-null   int64 
 1   title      4803 non-null   object
 2   cast       4803 non-null   object
 3   crew       4803 non-null   object
dtypes: int64(1), object(3)
memory usage: 150.2+ KB
```

Explore Credits dataset

credit_df.info()

According to the output, we found out that there are

4 variables and 4803 observations

Clean Data

Merge Datasets

We merge two dataframes, movie_df and credit_df, using a common column 'id' in movie_df and 'movie_id' in credit_df.

And we remove the column containing the redundant information from the original credit_df dataframe.

```
merged_df = pd.merge(movie_df, credit_df, left_on='id', right_on="movie_id")
```

```
merged_df = merged_df.drop("movie_id", axis=1)
```

```
merged_df.columns
```

```
Index(['budget', 'genres', 'homepage', 'id', 'keywords', 'original_language',
       'original_title', 'overview', 'popularity', 'production_companies',
       'production_countries', 'release_date', 'revenue', 'runtime',
       'spoken_languages', 'status', 'tagline', 'title_x', 'vote_average',
       'vote_count', 'title_y', 'cast', 'crew'],
      dtype='object')
```

```
print(merged_df.isnull().sum())
```

```
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_x                    0
vote_average               0
vote_count                 0
title_y                    0
cast                       0
crew                       0
dtype: int64
```

Handle Missing Value

This table gives the total number of missing values in each column.

The variables 'homepage', 'overview', 'release_date', 'runtime', and 'tagline' have varying numbers of missing values in the dataset.

Handle Missing Value

```
merged_df = merged_df.drop(['homepage', 'tagline'], axis=1)
merged_df = merged_df.dropna(subset=['runtime', 'release_date', 'overview'])
print(merged_df.isnull().sum())
```

We removed the 'homepage' and 'tagline' columns from the dataset as they are not useful for our data analysis.

As the variables 'runtime', 'release_date', and 'overview' have only a few missing values, we can remove the corresponding rows from the dataset.

Handle Duplicate Value

```
#check duplication
print("movie duplicated: ",merged_df.duplicated().sum())
```

```
movie duplicated:  0
```

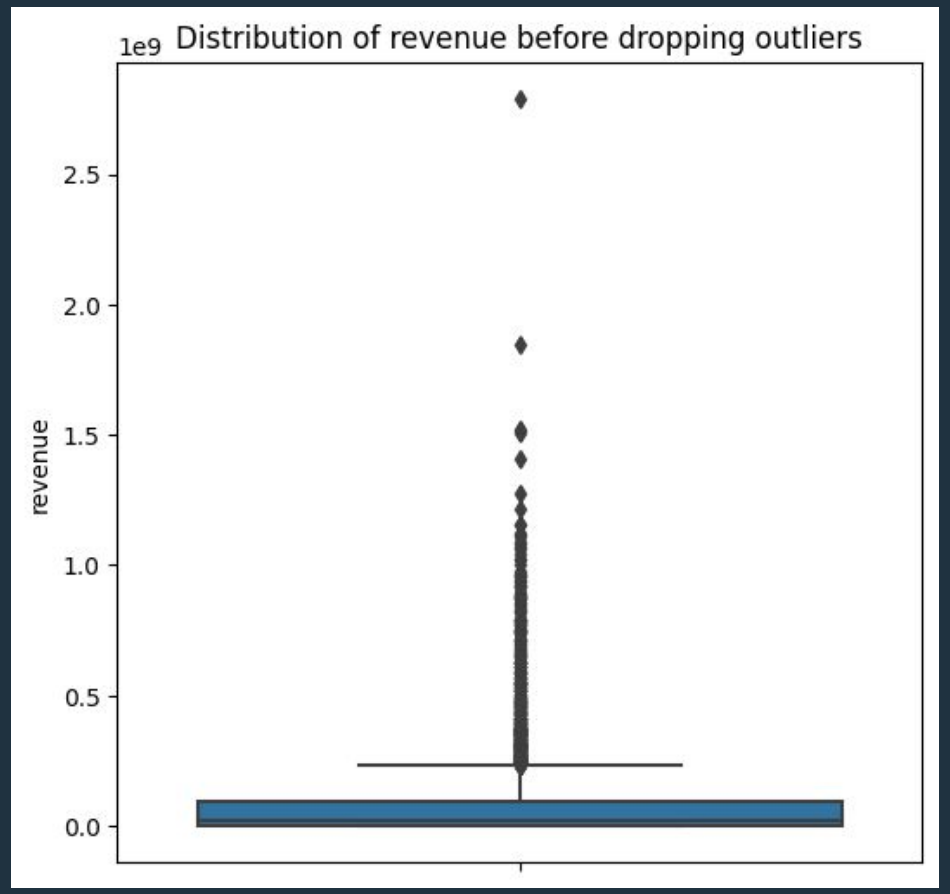
From this output, we can found out that there is no duplicate value in merged dataset.

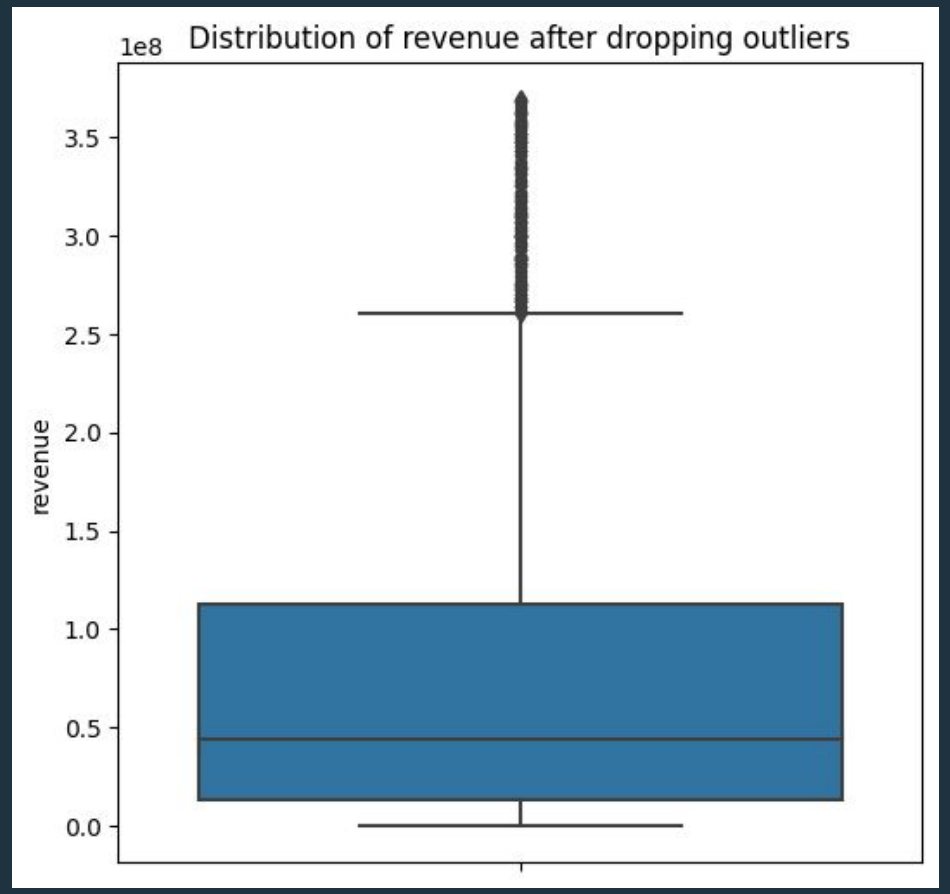
Handle Outlier

1. Use sns to generates a box-and-whisker plot of the distribution of values in the target column. Check if there are outliers in target values.
2. Setting the highest and lowest values to a predetermined percentile within the range of the data.

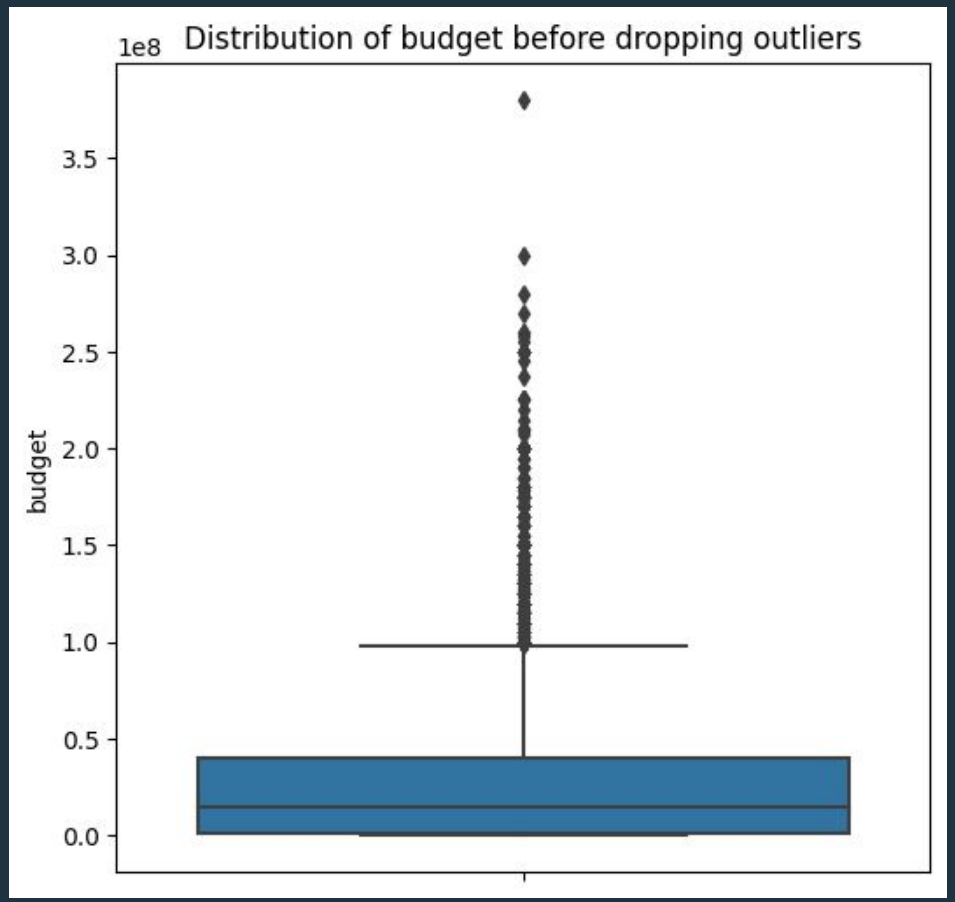
We will handle outlier in variable ‘revenue’, ‘budget’, ‘vote_count’.

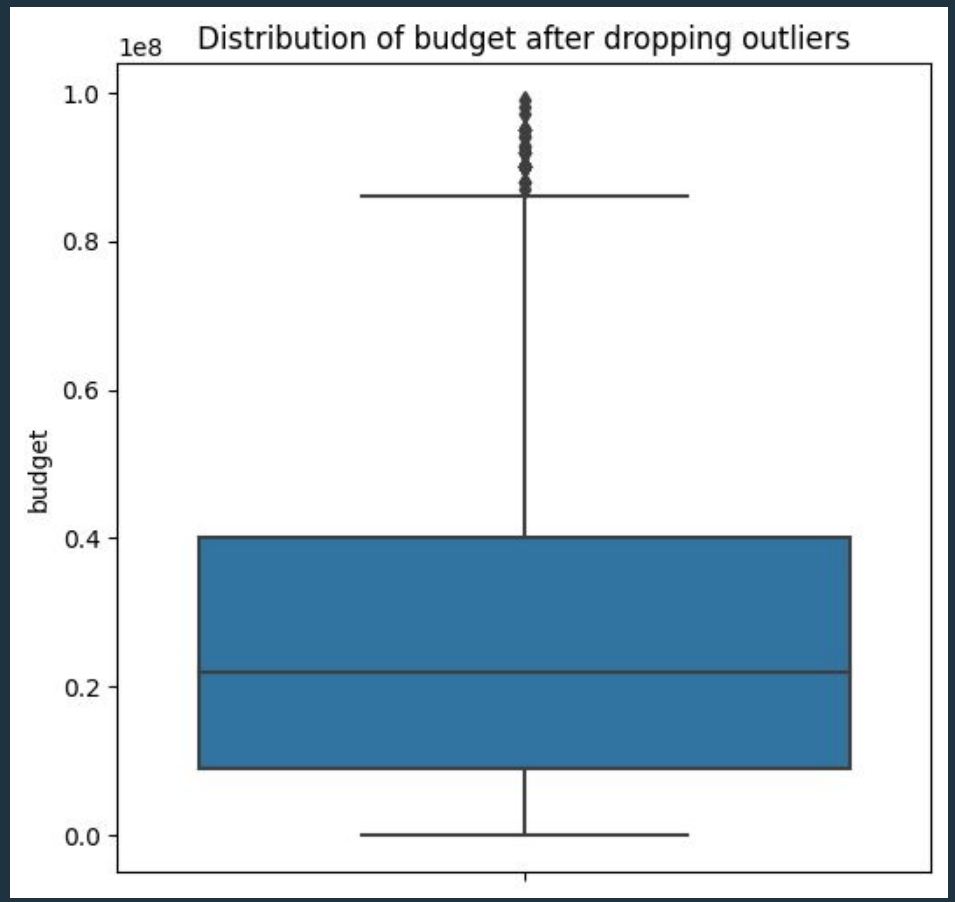
Drop Outliers in Revenue



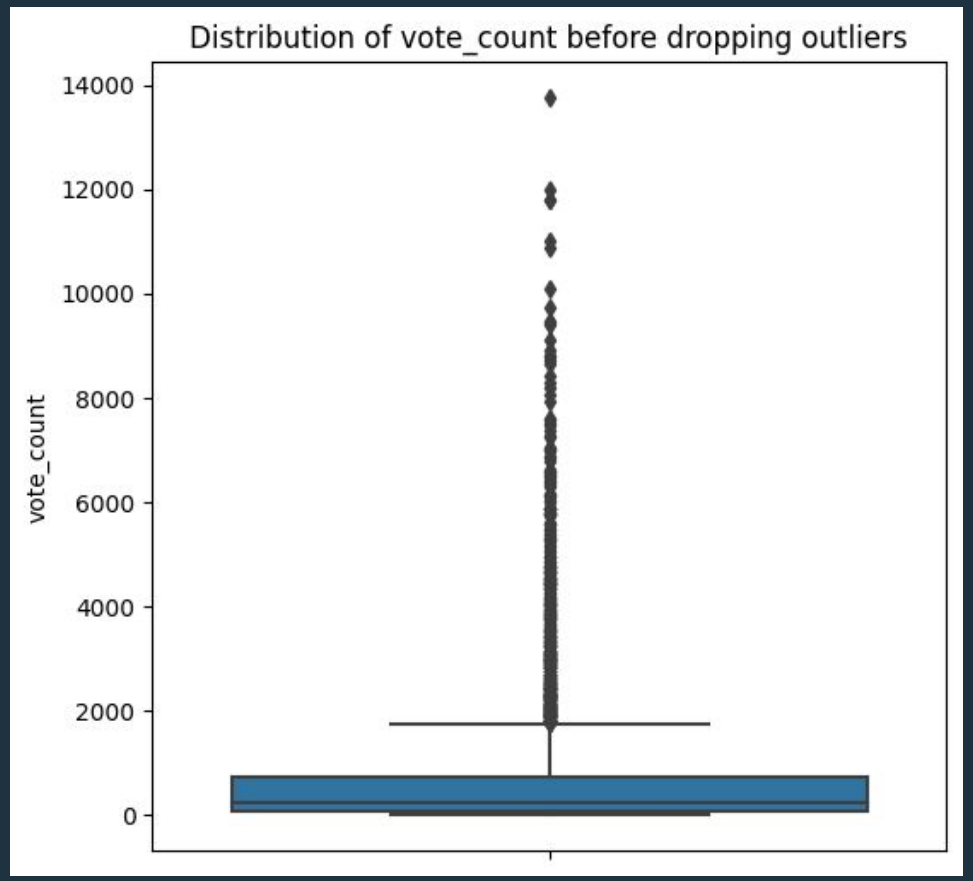


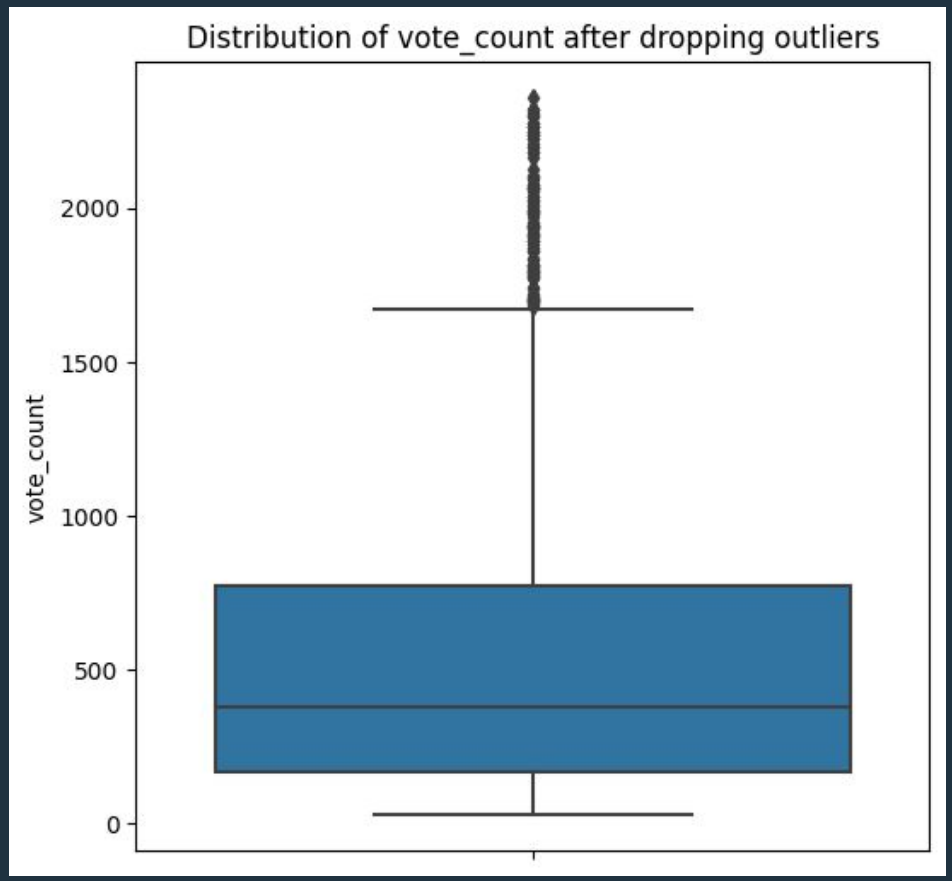
Drop Outliers in Budget





Drop Outliers in Vote_count





Parse Json

Parse Json

- Use library ast to extract information from Json format.
- There are 6 variables which are list type. We defined a extract_name function to extract useful information from these columns.
- There are 2 variables which are dictionary type. We defined two functions to extract 'get_producer' and 'get_director' from the 'crew' column.
- Those information will help us to analyze the prediction of the revenue of movie.

Extract Name

The purpose of this function is to extract name from a list of dictionaries represented as a string.

It will help us on splitting the genres in exploring the shifting trend of genres.

We will also apply this function to other 6 variables.

```
merged_df['genres'].head(1)
```

```
97    [{"id": 28, "name": "Action"}, {"id": 12, "name": "Adventure"},
{"id": 18, "name": "Drama"}, {"id": 27, "name": "Horror"}, {"id": 878,
"name": "Science Fiction"}]
```

```
def extract_name(data, attr = ['name']):
    result = []
    for i in ast.literal_eval(data):
        for a in attr:
            result.append(i[a])
    return result
```

```
merged_df['genres'].head(1)
```

```
97    [Action, Adventure, Drama, Horror, Science Fiction]
Name: genres, dtype: object
```

Extracted Merged Data Frame

	genres	keywords	production_companies	production_countries	spoken_languages	cast
97	[Action, Adventure, Drama, Horror, Science Fiction]	[monster, godzilla, giant monster, destruction, kaiju, toyko]	[Cine Bazar, Toho Pictures]	[Japan]	[Italiano, Deutsch, English, 日本語]	[Hiroki Hasegawa, Yutaka Takenouchi, Satomi Ishihara, Kengo Kora, Matsuo Satoru, Mikako Ichikawa...
151	[Adventure, Action, Animation]	[denmark, nordic mythology, lie, pride and vanity, folk hero, human weakness, viking, alienation...	[Paramount Pictures, Shangri-La Entertainment, ImageMovers, Paramount Animation]	[United States of America]	[English]	[Ray Winstone, Angelina Jolie, Anthony Hopkins, Robin Wright, John Malkovich, Brendan Gleeson, C...
207	[Action, Adventure, Science Fiction]	[oxygen, falsely accused, resistance, mars, double life, telepathy, mutant, hologram, space colo...	[TriStar Pictures, Carolco Pictures, Carolco International N.V.]	[United States of America]	[English]	[Arnold Schwarzenegger, Sharon Stone, Rachel Ticotin, Ronny Cox, Michael Ironside, Marshall Bell...
235	[Fantasy, Adventure, Comedy, Family]	[competition, greece, colosseum, olympic games, emperor, magic, horse, roman, wild boar, governa...	[Constantin Film, TF1 Films Productions, Pathé Renn Productions, La Petite Reine, Tri Pictures, ...	[Belgium, France, Germany, Italy, Spain]	[Français, Português]	[Clovis Cornillac, Gérard Depardieu, Franck Dubosc, José Garcia, Stéphane Rousseau, Jean-Pierre ...

Parse ‘crew’

```
def director(data):
    for i in ast.literal_eval(data):
        if i['department'] == 'Directing':
            return i['name']

def producer(data):
    for i in ast.literal_eval(data):
        if i['job'] == 'Producer':
            return i['name']
```

We extract director and producer from ‘crew’ column.

	Director	Producer
97	Hideaki Anno	Kazutoshi Wadakura
151	Robert Zemeckis	Robert Zemeckis
207	Paul Verhoeven	Ronald Shusett
235	Thomas Langmann	Thomas Langmann
273	Dominic Sena	Jerry Bruckheimer
...	...	...
4773	Kevin Smith	Kevin Smith
4788	John Waters	John Waters
4792	Kiyoshi Kurosawa	None
4796	Shane Carruth	Shane Carruth
4798	Robert Rodriguez	Robert Rodriguez

Data visualization

Release Date Analysis

Analysis of Movie Release Patterns by Day, Month, and Year

Release Date Analysis

First of all, we convert the ‘release_date’ column into a datetime object, format as %Y-%m-%d.

Then, extract the information about release year, month, and day of week from ‘release_date_obj’

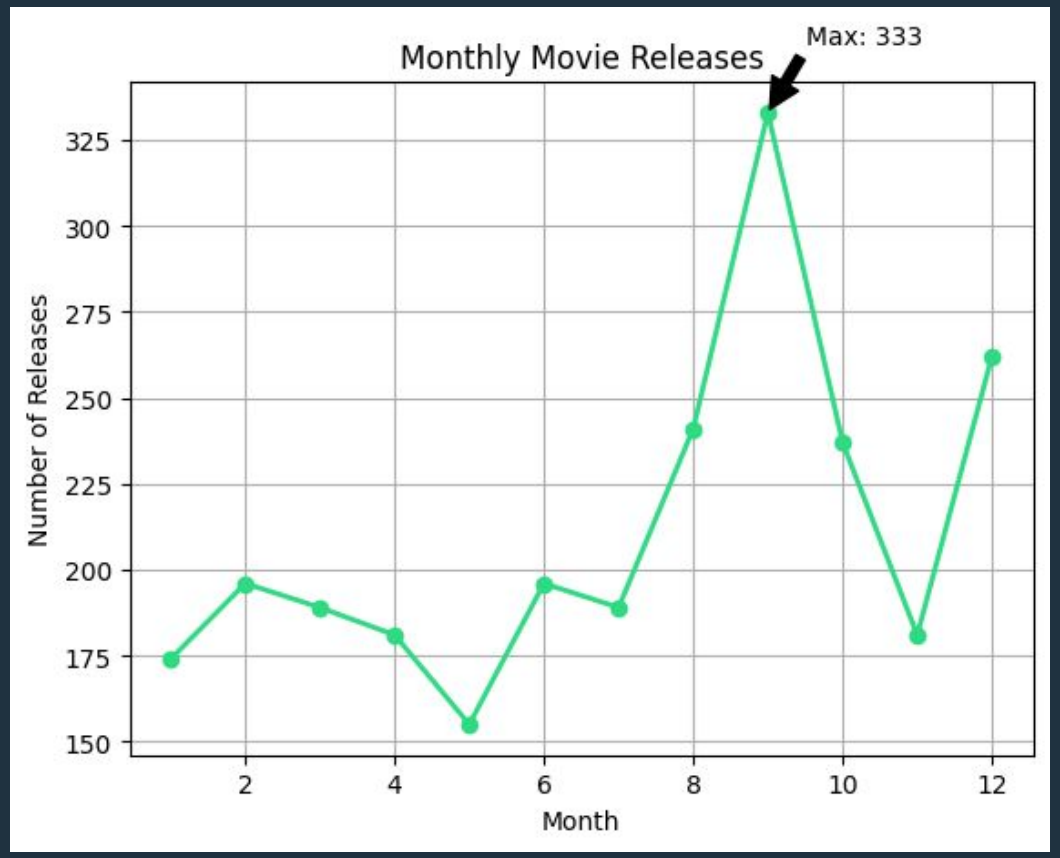
Finally, display and double-check the date after we processed.

```
merged_df["release_date_obj"] = pd.to_datetime(merged_df['release_date'],␣
↪format='%Y-%m-%d')
```

```
# Extract information about release year, month, and day of week
merged_df["release_month"] = merged_df['release_date_obj'].dt.month
merged_df["release_year"] = merged_df['release_date_obj'].dt.year
merged_df['release_day_of_week'] = merged_df['release_date_obj'].dt.day_name()
```

	release_month	release_year	release_day_of_week
97	7	2016	Friday
151	11	2007	Monday
207	6	1990	Friday
235	1	2008	Sunday
273	6	2000	Friday

Monthly Movie Releases



To better understand how the data evolves over time and identify peak points, we organized the release data into lists for each month and plotted it using line charts. After analyzing the data in this way, we discovered that the month of September had the highest number of movie releases, with a total of 333 movies. This approach allowed us to easily identify the highest data points for each month and visualize the changes in the data over time.

Yearly Movie Releases

In the yearly movie releases, we first create an empty dictionary called "year_dic" to store the count of movies released each year. Then we loop through each year in the "release_year" column of the merged_df dataframe, incrementing the count of the corresponding year in the "year_dic" dictionary.

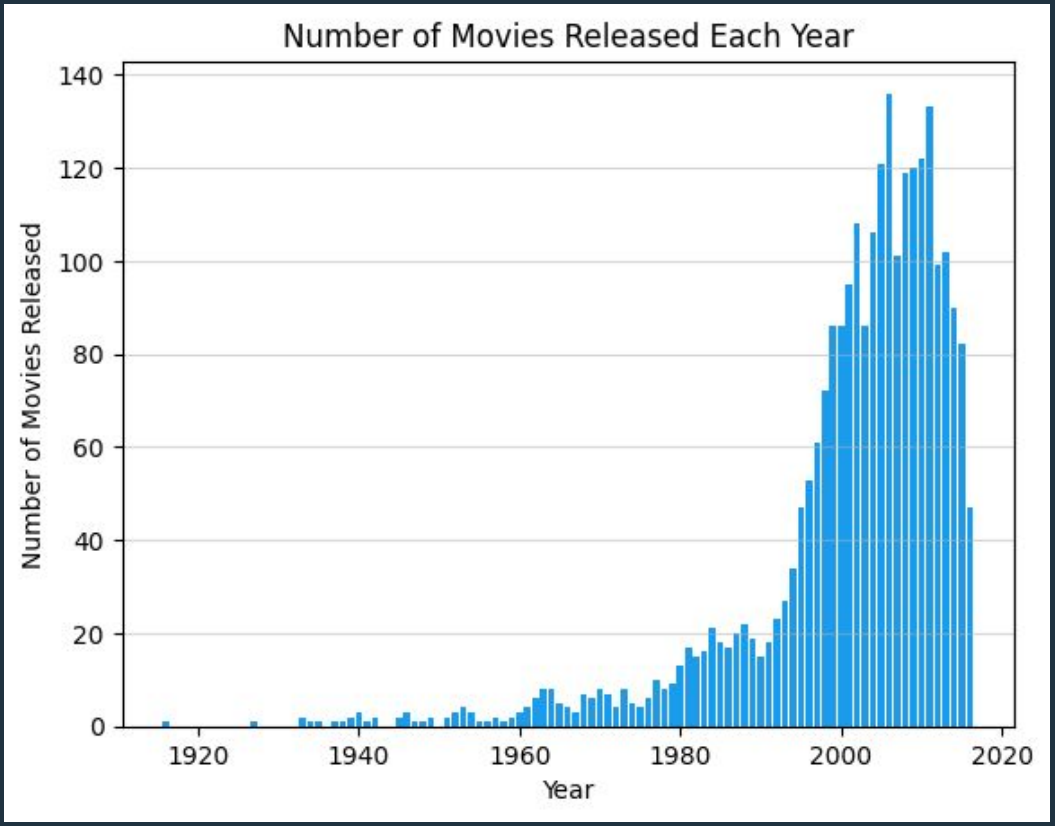
Next, we sort the dictionary items in ascending order based on the keys (years) using the "sorted" function. Then we create a bar chart and passing the years as the x-axis values and the movie count for each year as the y-axis values.

To make the plot more readable, we also add a grid using the "plt.grid" function. Finally, we display the plot using the "plt.show" function.

```
[40] year_dic = {}
     for year in merged_df["release_year"]:
         if year in year_dic:
             year_dic[year] += 1
         else:
             year_dic[year] = 1


     year_dic = dict(sorted(year_dic.items()))
```

```
# Create a bar chart of the number of movies released each year
plt.bar(year_dic.keys(), year_dic.values(), color="#1a9bec")

# Add axis labels and a title to the plot
plt.xlabel("Year")
plt.ylabel("Number of Movies Released")
plt.title("Number of Movies Released Each Year")

# Increase the font size of the axis labels and title
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)
plt.title("Number of Movies Released Each Year", fontsize=12)

# Add a grid to the plot
plt.grid(axis='y', alpha=0.5)

# Display the plot
plt.show()
```

Yearly Movie Releases



The bar chart shows that the number of movies released each year has increased steadily from the early 2000s, with a peak of 136 movies released in 2006. After this peak, there appears to be a plateau in the number of movies released, followed by a slight decrease in more recent years.

Day of The Week Movie Releases

We first group the merged_df dataframe by the "release_day_of_week" column using the "groupby" method, and then count the number of movies released on each day of the week using the "count" method.

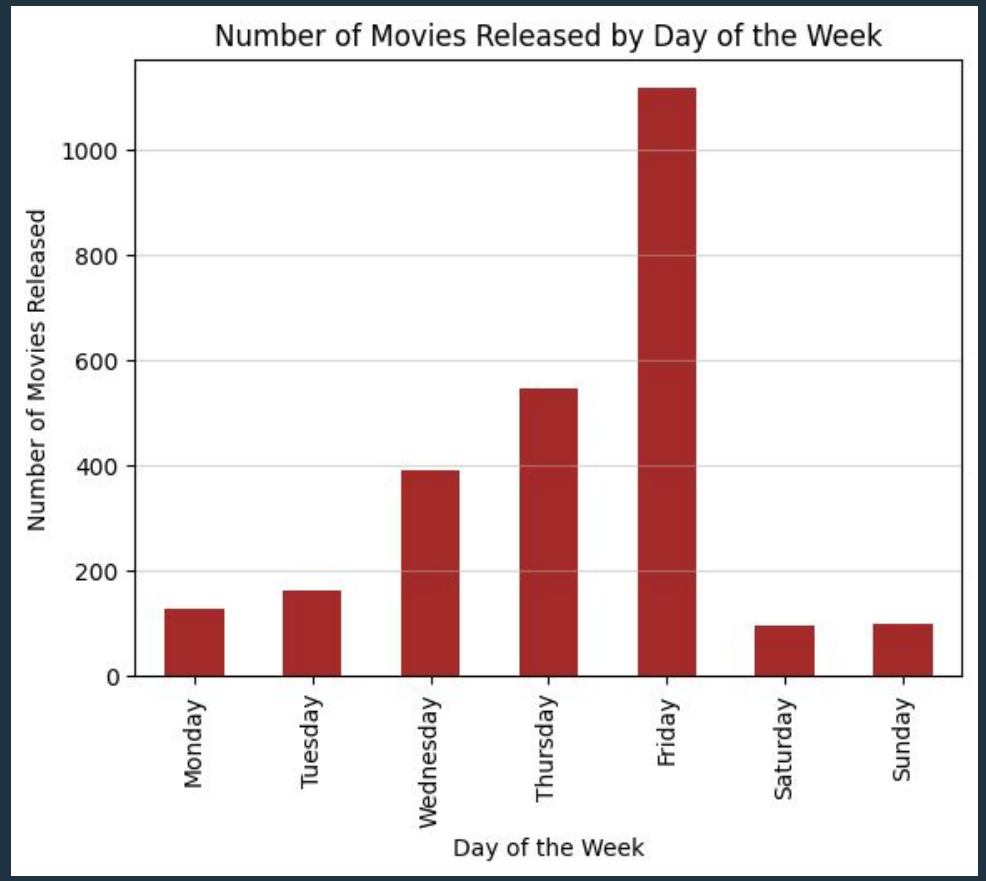
Then we sort the days of the week in ascending order using the "reindex" method, and create a bar chart of the number of movies released on each day of the week using the "plot.bar" method.

```
# Group the DataFrame by day of week and count the number of movies released on each day
day_of_week_counts = merged_df.groupby("release_day_of_week").count()["id"]

# Sort the days of the week in ascending order
day_of_week_counts = day_of_week_counts.reindex(
    ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"])

# Create a bar chart of the number of movies released on each day of the week
day_of_week_counts.plot.bar(color = "brown")

# Add axis labels and a title to the plot
plt.xlabel("Day of the Week")
plt.ylabel("Number of Movies Released")
plt.title("Number of Movies Released by Day of the Week")

# Increase the font size of the axis labels and title
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)
plt.title("Number of Movies Released by Day of the Week", fontsize=12)

# Add a grid to the plot
plt.grid(axis='y', alpha=0.5)

# Display the plot
plt.show()
```

Day of The Week Movie Releases

The resulting plot shows that the number of movies released varies by day of the week, with a peak on Fridays and a gradual decrease in the number of releases on weekends. The lowest number of releases occurs on Mondays.

This trend could be due to a variety of factors, including consumer behavior and preferences, competition for screen time and box office revenue, and the way movies are marketed and distributed.



Movie Genre Analysis

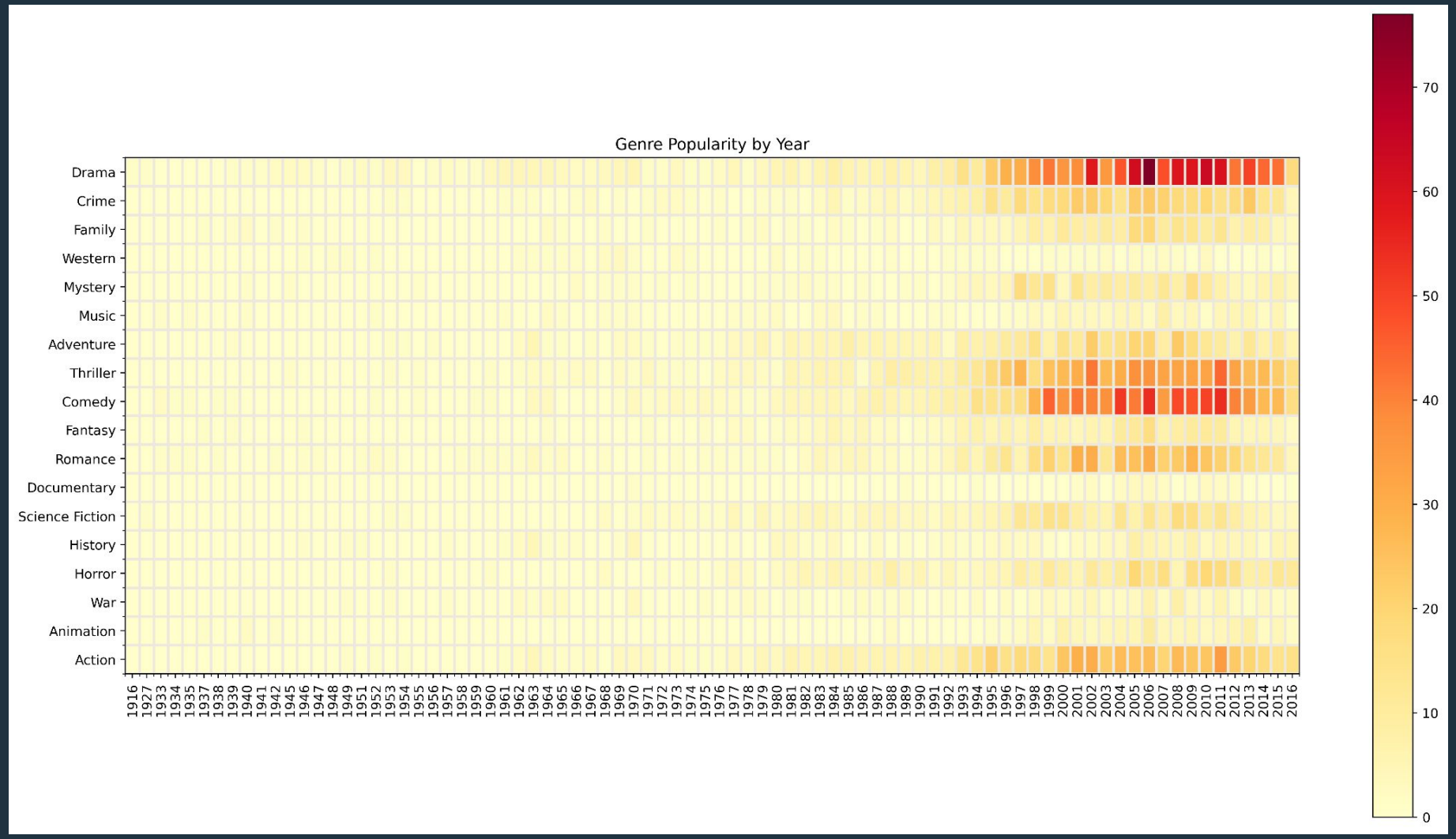
Identifying Genre Trend Shifting Patterns

Movie Genre Analysis

We creates a 2D dictionary where the outer dictionary has the year as its key and the inner dictionary has the genre as its key, storing the frequency of each genre that occurs in the dataset for that year.

```
genre_dic = {}
for index, row in merged_df.iterrows():
    if row['release_year'] not in genre_dic:
        genre_dic[row['release_year']] = {}

    for genre in row['genres']:
        if genre in genre_dic[row['release_year']]:
            genre_dic[row['release_year']][genre] += 1
        else:
            genre_dic[row['release_year']][genre] = 1
```

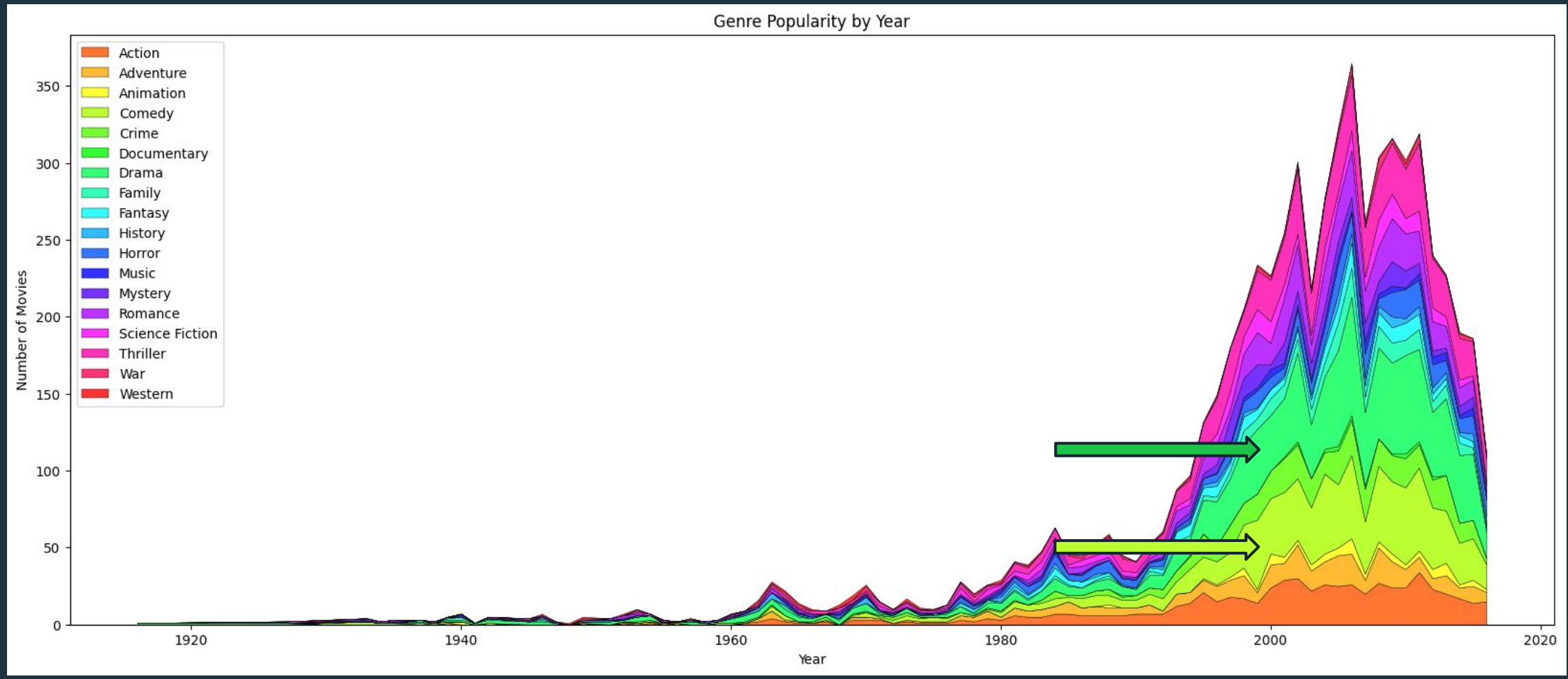
```
all_genres = set()
for year in genre_dic:
    for genre in genre_dic[year]:
        if genre not in all_genres:
            all_genres.add(genre)

# fill zeros
for year in genre_dic:
    for genre in all_genres:
        if genre not in genre_dic[year]:
            genre_dic[year][genre] = 0
```



Movie Genre Analysis

This heatmap that shows the popularity of each movie genre over time. The x-axis represents the years, and the y-axis represents the movie genres. The intensity of each color represents the popularity of each genre in a particular year, with brighter colors indicating more popular genres.

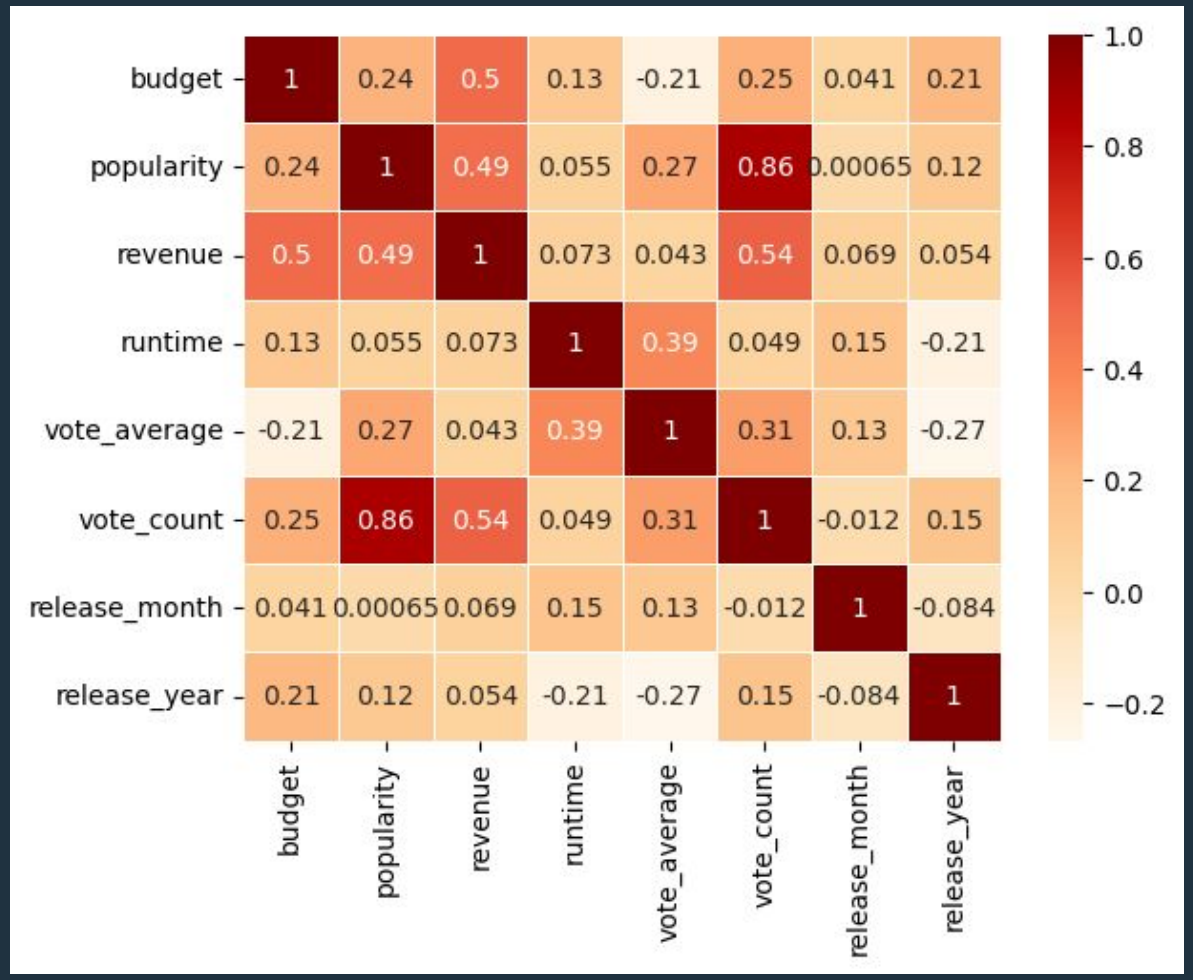
From the heatmap, we can see that the popularity of some genres has remained consistent over time, such as drama, comedy, and thriller. Other genres, such as fantasy and science fiction, have become more popular over the years. We can also see some fluctuations in popularity for certain genres, such as horror and romance.



Movie Genre Analysis

The stacked area plot shows the popularity of movie genres over time. Each color represents a different genre and the height of the colored area at any given year represents the number of movies released in that genre in that year. The x-axis represents the years and the y-axis represents the number of movies. The legend shows which color corresponds to which genre. From the plot, we can observe the rise and fall in popularity of various genres over time, and also see which genres have consistently remained popular throughout the years.

Modelling



The heatmap analysis shows that budget, popularity, and vote_count have strong positive correlation with revenue, while vote_average has weak correlation. Therefore, we will focus on these three features to build a predictive model for the dataset. Higher budgets, more popular, and more vote_count tend to generate higher revenues, but the average rating of a movie is not a strong predictor of its revenue.

Linear Regression

Linear Regression

We creates a new pandas DataFrame named x that includes three new columns: "log_bud", "log_pop", and "log_vc". These columns are created by applying the NumPy log function to three columns from the merged_df DataFrame: "budget", "popularity", and "vote_count", respectively.

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
```

```
x = pd.DataFrame()
x["log_bud"] = merged_df["budget"].map(lambda x:np.log(x+1))
x["log_pop"] = merged_df["popularity"].map(lambda x:np.log(x+1))
x["log_vc"] = merged_df["vote_count"].map(lambda x:np.log(x+1))

y = merged_df["revenue"].map(lambda x:np.log(x+1))
```

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)
```

```
linear_regression_model = LinearRegression()
linear_regression_model.fit(x_train, y_train)
prediction = linear_regression_model.predict(x_test)
```

Linear Regression

RSE: 1.288
RMSE: 1.284

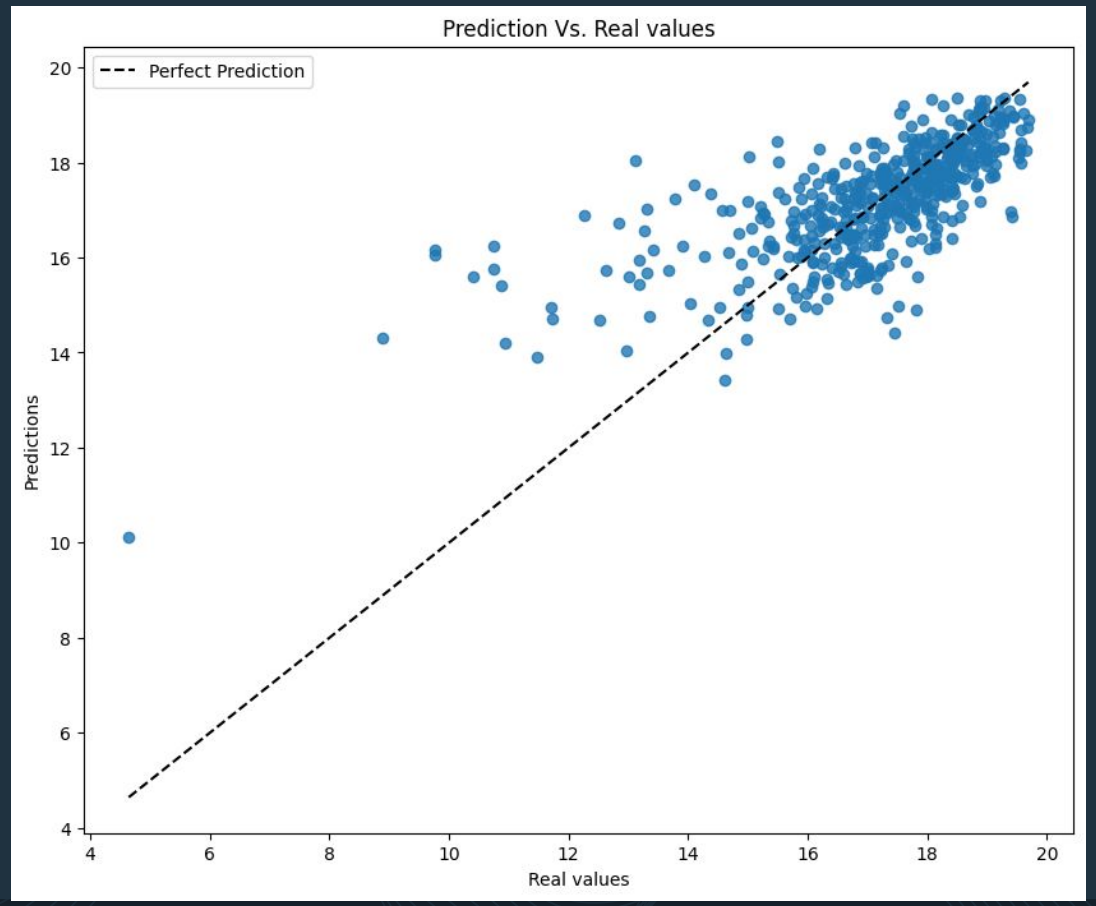
```python
from sklearn.metrics import mean_squared_error

# Calculate the mean squared error of the model on the testing set
mse_test = mean_squared_error(y_test, prediction)

# Calculate the residual standard error of the model on the testing set
rse_test = np.sqrt(mse_test * (len(y_test) - 1) / (len(y_test) - x_test.shape[1] - 1))

print("Residual Standard Error (RSE) on testing set:", rse_test)
```

```
root mean squared error:  1.2840663439782913
Residual Standard Error (RSE) on testing set: 1.287889875053391
```

Calculate the root mean squared error and residual standard error, the linear regression model appears to have decent predictive power on the testing set.

The result indicates that the model has relatively low errors and can be used to predict the revenue of movies in the dataset with a reasonable degree of accuracy.

Linear Regression



The scatter plot of real values versus predictions shows that the model is able to predict revenue reasonably well, with most of the points falling near the line of perfect prediction.

Logistic Regression

Logistic Regression

We create a new column called ‘target’ which is based on a binary classification of movies into two categories based on their revenue and budget: if the revenue is less than the budget, the movie is classified as a "loss" (0), and if the revenue is greater than or equal to the budget, the movie is classified as a "success" (1)

```
x = pd.DataFrame()
x["log_bud"] = merged_df["budget"].map(lambda x:np.log(x+1))
x["log_pop"] = merged_df["popularity"].map(lambda x:np.log(x+1))
x["log_vc"] = merged_df["vote_count"].map(lambda x:np.log(x+1))


log_rev = merged_df["revenue"].map(lambda x:np.log(x+1))
x["target"] = [0 if z[0] < z[1] else 1 for z in zip(log_rev, x["log_bud"])]
```

Balance the class size

```
# Balancing the class size for model performance
copied_data = x[x["target"] == 0]
x = x.append(copied_data)
```

Balancing the class size is necessary in machine learning to address class imbalance. Class imbalance occurs when the number of instances in different classes is significantly different, which can lead to biased models and poor performance on the minority class. By balancing the class size, we ensure that the model is exposed to sufficient samples from all classes, allowing it to learn patterns and make accurate predictions for both majority and minority classes.

F1: 0.778
RSE: 0.524

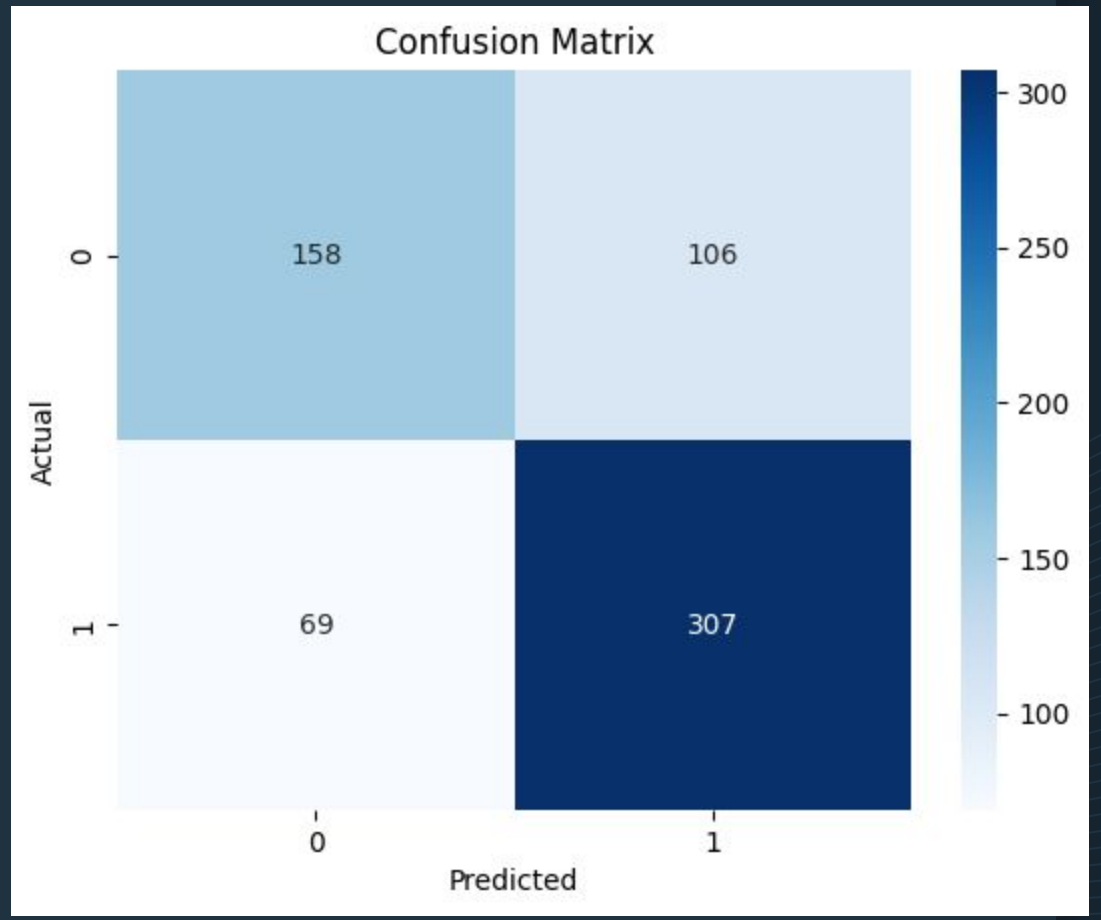
```
# Calculate the mean squared error of the model on the testing set
mse_test = mean_squared_error(y_test, prediction)

# Calculate the residual standard error of the model on the testing set
rse_test = np.sqrt(mse_test * (len(y_test) - 1) / (len(y_test) - x_test.shape[1] - 1))

print("Residual Standard Error (RSE) on testing set:", rse_test)
```

```
Residual Standard Error (RSE) on testing set: 0.5241443498963829
```

The logistic regression model achieved an F1 score of 0.778, indicating that it can accurately predict whether a movie will earn above or below its budget based on the selected features. The RSE value of 0.524 suggests that there is some level of error in the model's predictions. Overall, the model's performance is decent, but there is still room for improvement.

Logistic Regression

This confusion matrix give an idea of how well the logistic regression model is performing. By looking at the matrix, we can see how many true positives, true negatives, false positives, and false negatives the model produced.



KNN Classifier

KNN Classifier

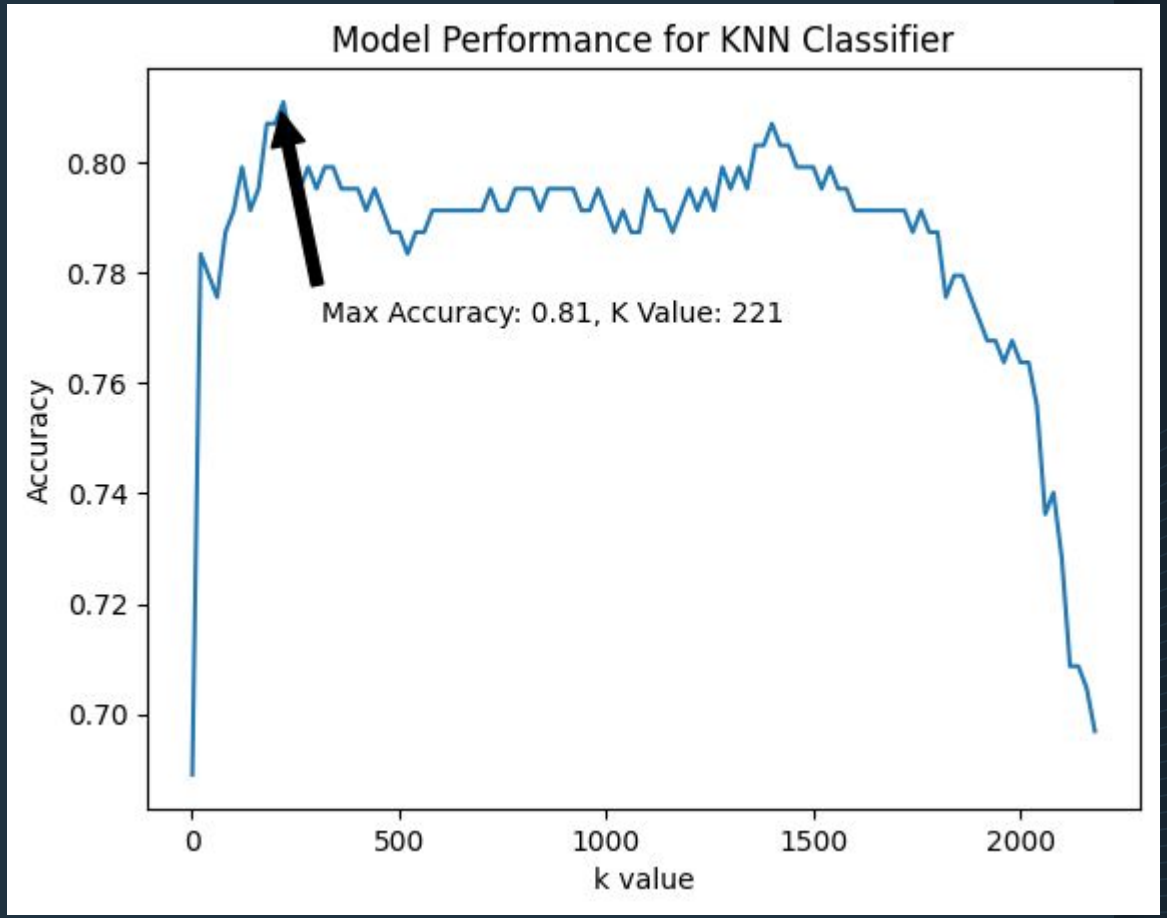
First of all, the revenue column labels the data as high-revenue or low-revenue based on its percentile rank.

Next, trains a KNN model with different values of k and evaluates its accuracy using the accuracy_score function. The results are plotted on a graph to determine the optimal value of k for the model.

```
y = merged_df["revenue"]

# get y label (high-revenue and low revenue)
y_label = [1 if percentileofscore(y, i) > 50 else 0 for i in y]

# split data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(x, y_label, test_size=0.1, random_state=42)

# Initialize empty lists for k values and accuracy
k_values = []
accuracies = []

for i in range(1, 2200, 20):
    # instantiate KNN model with k=5
    knn = KNeighborsClassifier(n_neighbors=i)

    # fit the model on training data
    knn.fit(x_train, y_train)

    # make predictions on test data
    prediction = knn.predict(x_test)

    # evaluate the model's accuracy
    accuracy = accuracy_score(y_test, prediction)
    k_values.append(i)
    accuracies.append(accuracy)
```

KNN Classifier

Performing K-Nearest Neighbors (KNN) classification on a dataset containing information about movie budgets, popularity, and vote count to predict whether a movie will have high or low revenue.



KNN Classifier

F1: 0.808
RSE:0.437

```
score = f1_score(y_test, prediction)
print("F1 score:", score)


# Calculate the mean squared error of the model on the testing set
mse_test = mean_squared_error(y_test, prediction)

# Calculate the residual standard error of the model on the testing set
rse_test = np.sqrt(mse_test * (len(y_test) - 1) / (len(y_test) - x_test.shape[1] - 1))

print("Residual Standard Error (RSE) on testing set:", rse_test)
```

```
F1 score: 0.808
Residual Standard Error (RSE) on testing set: 0.4373146401484736
```

The linear regression model has an F1 score of 0.808 and RSE of 0.437, indicating that the model has a relatively good fit to the data.

Conclusion

Comparing Performance - RSE

1.288

Linear Regression

0.524

Logistic Regression

0.437

KNN Classifier

Comparing Performance - Cross Validation Score

0.835

Linear Regression

0.723

Logistic Regression

0.780

KNN Classifier

Models

Linear Regression Model - The model is a regression model that utilizes budget, popularity, and vote count as input features to predict movie revenue. It learns the patterns and correlations between these features and revenue from a training dataset, and can then generate revenue predictions for new movies based on these learned relationships.

Logistic Regression Model - The logistic regression model takes budget, popularity, and vote count as input features to classify movies as either profitable or not. It learns the relationship between these features and the binary outcome of profitability from a training dataset, using a logistic function to model the probability of a movie being profitable.

KNN Model - The model takes into consideration the budget, popularity, and vote count of movies to classify them as high-revenue or low-revenue. It does this by comparing the features of a new movie to the K number of nearest neighbors in the training dataset. The label assigned to the new movie is determined by the majority class of its K nearest neighbors.

Thank You