IMDb Movies Data Analysis using Python

Content:

- Exploring Dataset:
  1. Importing Libraries
  2. Importing the dataset, loading in dataframe
  3. Studying the structure of the dataset
- Data Preprocessing:
  1. Checking NULL Values
  2. Filling NULL Values
  3. Checking Duplicates
  4. Handling Outliers
  5. Data Normalization
- Data Analysis:
  1. Movies per Genre
  2. Heatmap
  3. Word Cloud for Directors
  4. Scatter Plot of Revenue vs. Rating
  5. Movies per Director
  6. Distribution of Movie Ratings
  7. Word cloud of actors
  8. Genre Combinations Analysis
  9. Revenue Distribution by Genre
  10. Rating Distribution by Year
  11. Top Actors Analysis
- Time Series Analysis:
  1. Times Series Analysis of Movies over Year
  2. Time Series Analysis of Average Rating Over Years
  3. Time Series Analysis of Revenue Over Years
  4. Seasonal Trends in Movie Releases
- Predictive Modeling using Machine Learning Algorithms:
  1. For Movie Revenue (Linear Regression)
  2. For movie rating category- High/Low (Random Forest)
  3. Predicting Movie Success (Binary Classification)
  4. Feature Importance Analysis
  5. Clustering Movies
- Visualizations:
  1. Interactive Dashboards
  2. Sunburst Chart for Genre Distribution
  3. Chord Diagram for Director-Actor Collaborations

Importing Libraries

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import missingno as msno
from wordcloud import WordCloud, STOPWORDS
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score
from sklearn.preprocessing import LabelEncoder
from sklearn.cluster import KMeans
import warnings
warnings.filterwarnings("ignore")
```

Importing the Dataset

In [3]:
```
df = pd.read_csv(r"C:\Users\anisa\IE 6600 Lab\IMDb-Data-Analysis-Python-Tableau-main\IMDb-Data-Analysis-Python-Tableau-main\IM

df.set_index('Title', inplace=True)

df.head(10)
```

Out[3]:

Title	Rank	Genre	Description	Director	Actors	Year	Runtime (Minutes)	Rating	Votes	Revenue (Millions)	M
Guardians of the Galaxy	1	Action,Adventure,Sci-Fi	A group of intergalactic criminals are forced ...	James Gunn	Chris Pratt, Vin Diesel, Bradley Cooper, Zoe S...	2014	121	8.1	757074	333.13	
Prometheus	2	Adventure,Mystery,Sci-Fi	Following clues to the origin of mankind, a te...	Ridley Scott	Noomi Rapace, Logan Marshall-Green, Michael Fa...	2012	124	7.0	485820	126.46	
Split	3	Horror,Thriller	Three girls are kidnapped by a man with a diag...	M. Night Shyamalan	James McAvoy, Anya Taylor-Joy, Haley Lu Richar...	2016	117	7.3	157606	138.12	
Sing	4	Animation,Comedy,Family	In a city of humanoid animals, a hustling thea...	Christophe Lourdelet	Matthew McConaughey,Reese Witherspoon, Seth Ma...	2016	108	7.2	60545	270.32	
Suicide Squad	5	Action,Adventure,Fantasy	A secret government agency recruits some of th...	David Ayer	Will Smith, Jared Leto, Margot Robbie, Viola D...	2016	123	6.2	393727	325.02	
The Great Wall	6	Action,Adventure,Fantasy	European mercenaries searching for black powde...	Yimou Zhang	Matt Damon, Tian Jing, Willem Dafoe, Andy Lau	2016	103	6.1	56036	45.13	
La La Land	7	Comedy,Drama,Music	A jazz pianist falls for an aspiring actress i...	Damien Chazelle	Ryan Gosling, Emma Stone, Rosemarie DeWitt, J....	2016	128	8.3	258682	151.06	
Mindhorn	8	Comedy	A has-been actor best known for playing the ti...	Sean Foley	Essie Davis, Andrea Riseborough, Julian Barrat...	2016	89	6.4	2490	NaN	
The Lost City of Z	9	Action,Adventure,Biography	A true-life drama, centering on British explor...	James Gray	Charlie Hunnam, Robert Pattinson, Sienna Mille...	2016	141	7.1	7188	8.01	
Passengers	10	Adventure,Drama,Romance	A spacecraft traveling to a distant colony pla...	Morten Tyldum	Jennifer Lawrence, Chris Pratt, Michael Sheen,...	2016	116	7.0	192177	100.01	

Structure of the Dataset

In [4]:
```python
print("Number of rows:", df.shape[0])
print("Number of columns:", df.shape[1])
```

```
Number of rows: 1000
Number of columns: 11
```

In [5]:
```python
df.info()

df.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1000 entries, Guardians of the Galaxy to Nine Lives
Data columns (total 11 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   Rank                1000 non-null   int64  
 1   Genre               1000 non-null   object 
 2   Description         1000 non-null   object 
 3   Director            1000 non-null   object 
 4   Actors              1000 non-null   object 
 5   Year                1000 non-null   int64  
 6   Runtime (Minutes)   1000 non-null   int64  
 7   Rating              1000 non-null   float64
 8   Votes               1000 non-null   int64  
 9   Revenue (Millions)  872 non-null    float64
 10  Metascore           936 non-null    float64
dtypes: float64(3), int64(4), object(4)
memory usage: 93.8+ KB
```

Out[5]:

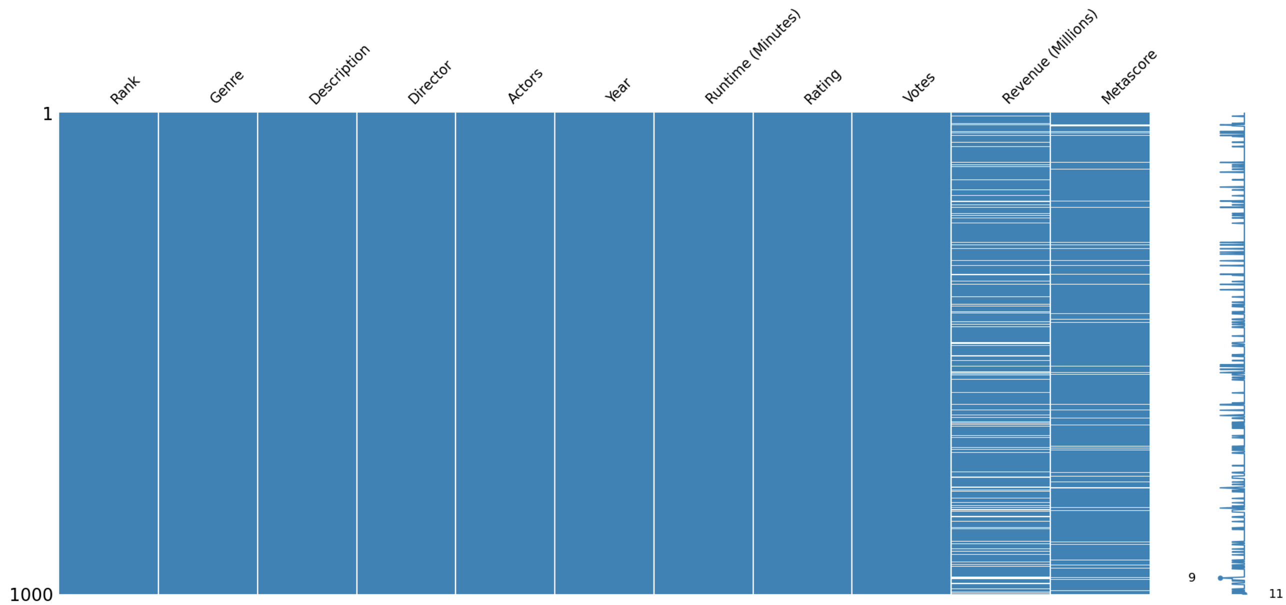
	Rank	Year	Runtime (Minutes)	Rating	Votes	Revenue (Millions)	Metascore
count	1000.000000	1000.000000	1000.000000	1000.000000	1.000000e+03	872.000000	936.000000
mean	500.500000	2012.783000	113.172000	6.723200	1.698083e+05	82.956376	58.985043
std	288.819436	3.205962	18.810908	0.945429	1.887626e+05	103.253540	17.194757
min	1.000000	2006.000000	66.000000	1.900000	6.100000e+01	0.000000	11.000000
25%	250.750000	2010.000000	100.000000	6.200000	3.630900e+04	13.270000	47.000000
50%	500.500000	2014.000000	111.000000	6.800000	1.107990e+05	47.985000	59.500000
75%	750.250000	2016.000000	123.000000	7.400000	2.399098e+05	113.715000	72.000000
max	1000.000000	2016.000000	191.000000	9.000000	1.791916e+06	936.630000	100.000000

Data Preprocessing

1. Checking NULL Values

In [6]:
```python
import missingno as msno
import matplotlib.pyplot as plt

# Print the count of missing values
print(df.isnull().sum())

# Visualize missing data with a colorful matrix
msno.matrix(df, sparkline=True, color=(0.27, 0.52, 0.72))
plt.show()
```

```
Rank                    0
Genre                   0
Description             0
Director                0
Actors                  0
Year                    0
Runtime (Minutes)       0
Rating                  0
Votes                   0
Revenue (Millions)    128
Metascore              64
dtype: int64
```



2. Filling NULL Values

In [7]:
```
df['Revenue (Millions)'].fillna(df['Revenue (Millions)'].mean(), inplace=True)
print(df.isnull().sum())
```

```
Rank                   0
Genre                  0
Description            0
Director               0
Actors                 0
Year                   0
Runtime (Minutes)      0
Rating                 0
Votes                  0
Revenue (Millions)     0
Metascore             64
dtype: int64
```

3. Checking Duplicates

In [8]:
```
duplicates = df[df.duplicated()]
print(duplicates)
```

```
Empty DataFrame
Columns: [Rank, Genre, Description, Director, Actors, Year, Runtime (Minutes), Rating, Votes, Revenue (Millions), Metascore]
Index: []
```

4. Handling Outliers

In [9]:
```
def detect_outliers(data):
    Q1 = data.quantile(0.25)
    Q3 = data.quantile(0.75)
    IQR = Q3 - Q1
    outliers = ((data < (Q1 - 1.5 * IQR)) | (data > (Q3 + 1.5 * IQR)))
    return outliers

outliers = df[['Revenue (Millions)', 'Runtime (Minutes)', 'Rating', 'Votes']].apply(detect_outliers)
print(outliers.sum())
```

```
Revenue (Millions)    82
Runtime (Minutes)     30
Rating                19
Votes                 45
dtype: int64
```

5. Data Normalization

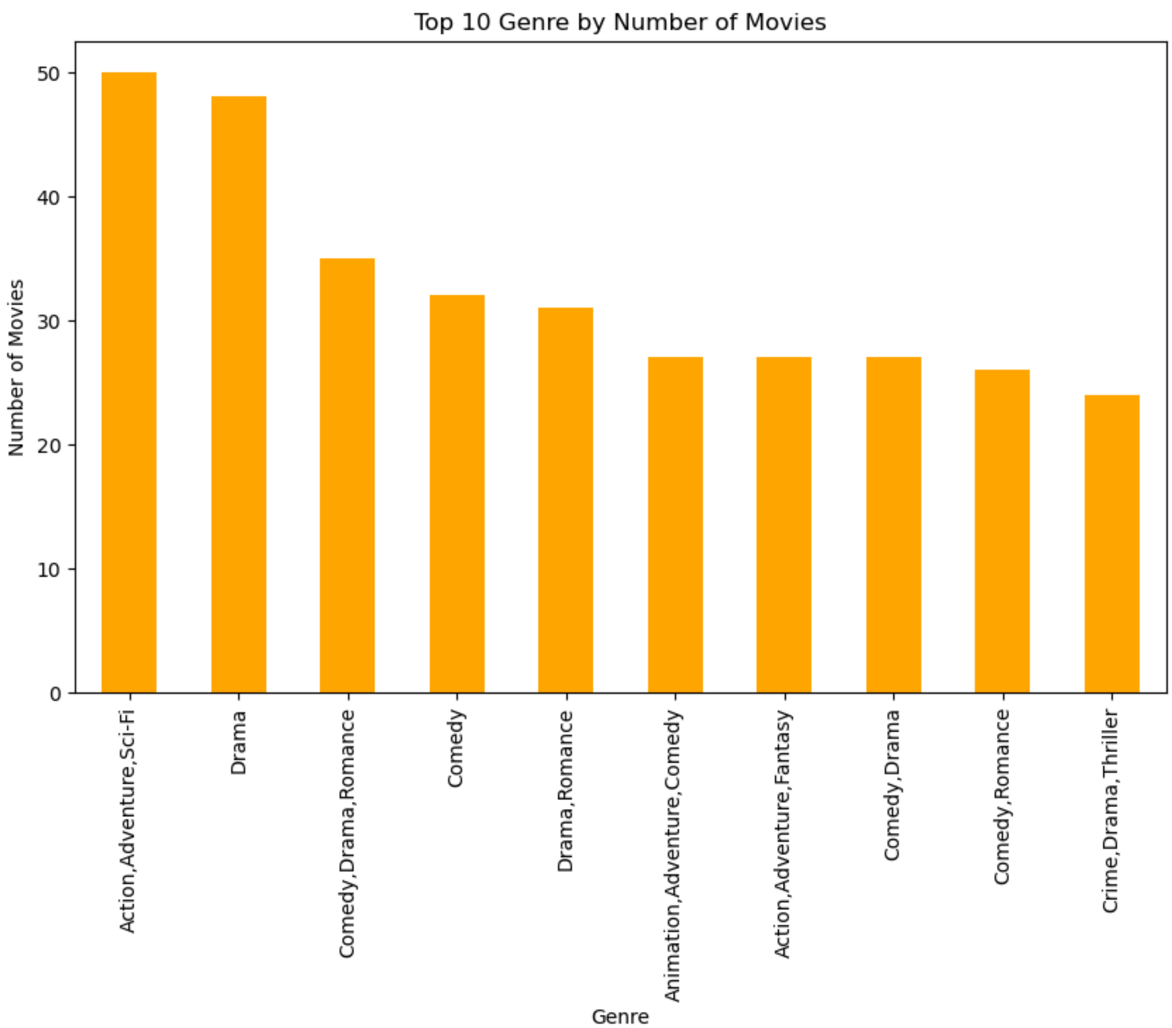
In [10]:
```
from sklearn.preprocessing import MinMaxScaler

# Normalizing numerical features
```

```
scaler = MinMaxScaler()
df[['Revenue (Millions)', 'Runtime (Minutes)', 'Rating', 'Votes']] = scaler.fit_transform(df[['Revenue (Millions)', 'Runtime
```

Data Analysis

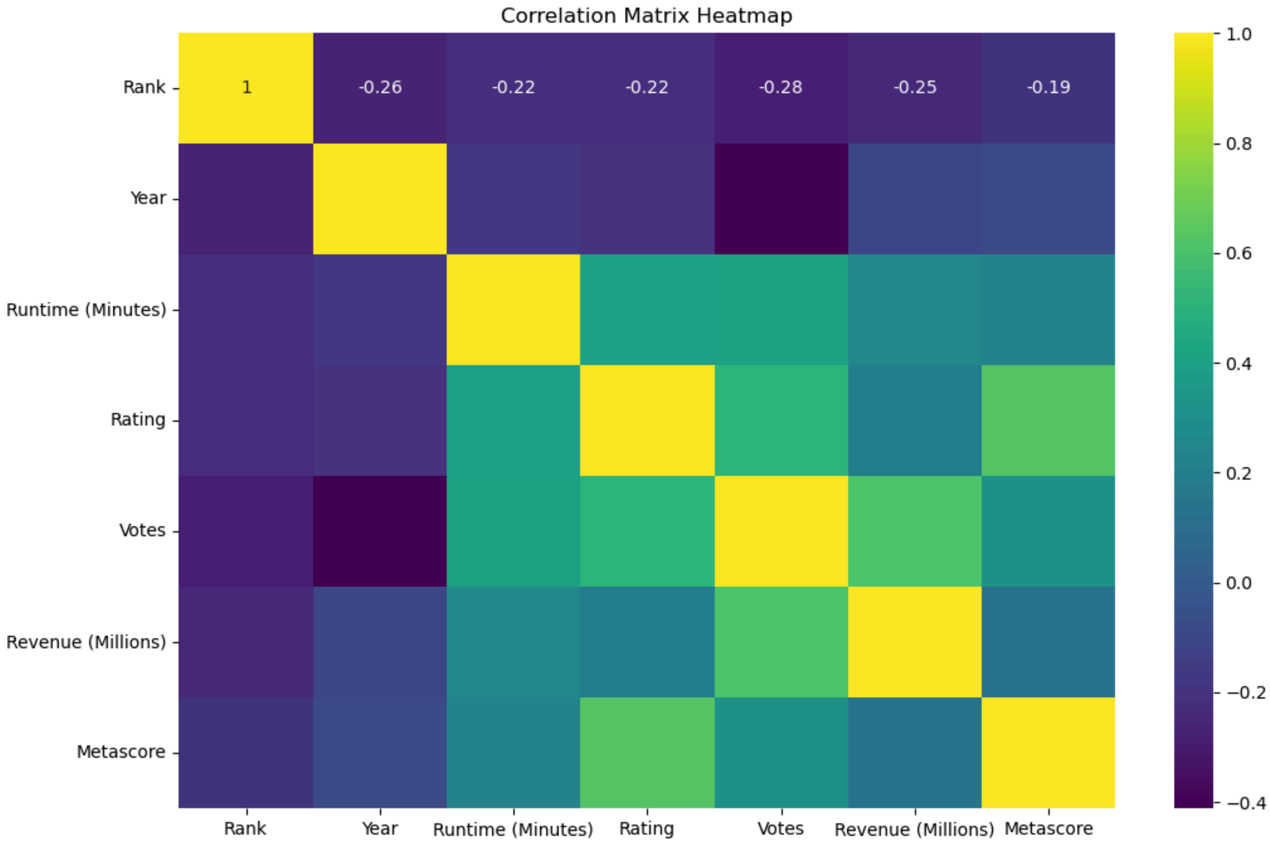
1. Movies per Genre

In [11]:
```
top_10_genres = df['Genre'].value_counts().nlargest(10)
plt.figure(figsize=(10, 6))
top_10_genres.plot(kind='bar', color='Orange')
plt.title('Top 10 Genre by Number of Movies')
plt.xlabel('Genre')
plt.ylabel('Number of Movies')
plt.xticks(rotation=90)
plt.show()
```



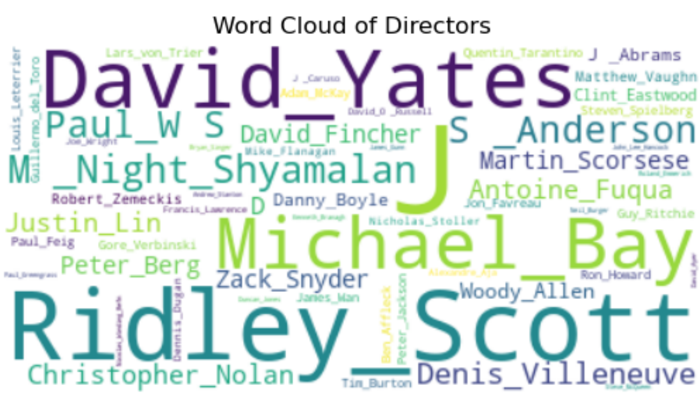
2. Heatmap

In [12]:
```
numeric_df = df.select_dtypes(include=['float64', 'int64'])
plt.figure(figsize=(12, 8))
sns.heatmap(numeric_df.corr(), annot=True, cmap='viridis')
plt.title('Correlation Matrix Heatmap')
plt.show()
```



3. Word Cloud for Directors

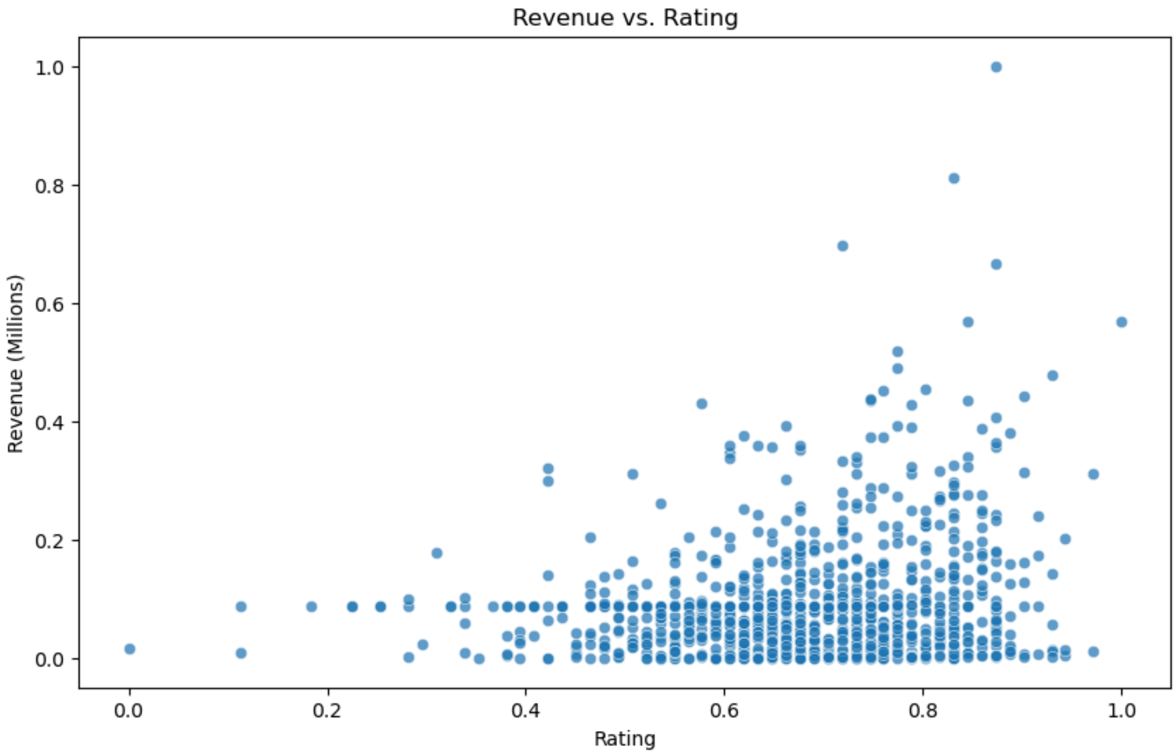
In [13]:
```
text = ''
for i in df['Director']:
    value = i.strip()
    value = value.replace(" ", "_")
    text = text + " " + value

text = text.strip()

wordcloud = WordCloud(stopwords=STOPWORDS, background_color="white").generate(text)
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.title('Word Cloud of Directors')
plt.show()
```



4. Scatter Plot of Revenue vs. Rating

In [14]:
```
plt.figure(figsize=(10, 6))
sns.scatterplot(x='Rating', y='Revenue (Millions)', data=df, alpha=0.7)
plt.title('Revenue vs. Rating')
plt.xlabel('Rating')
```

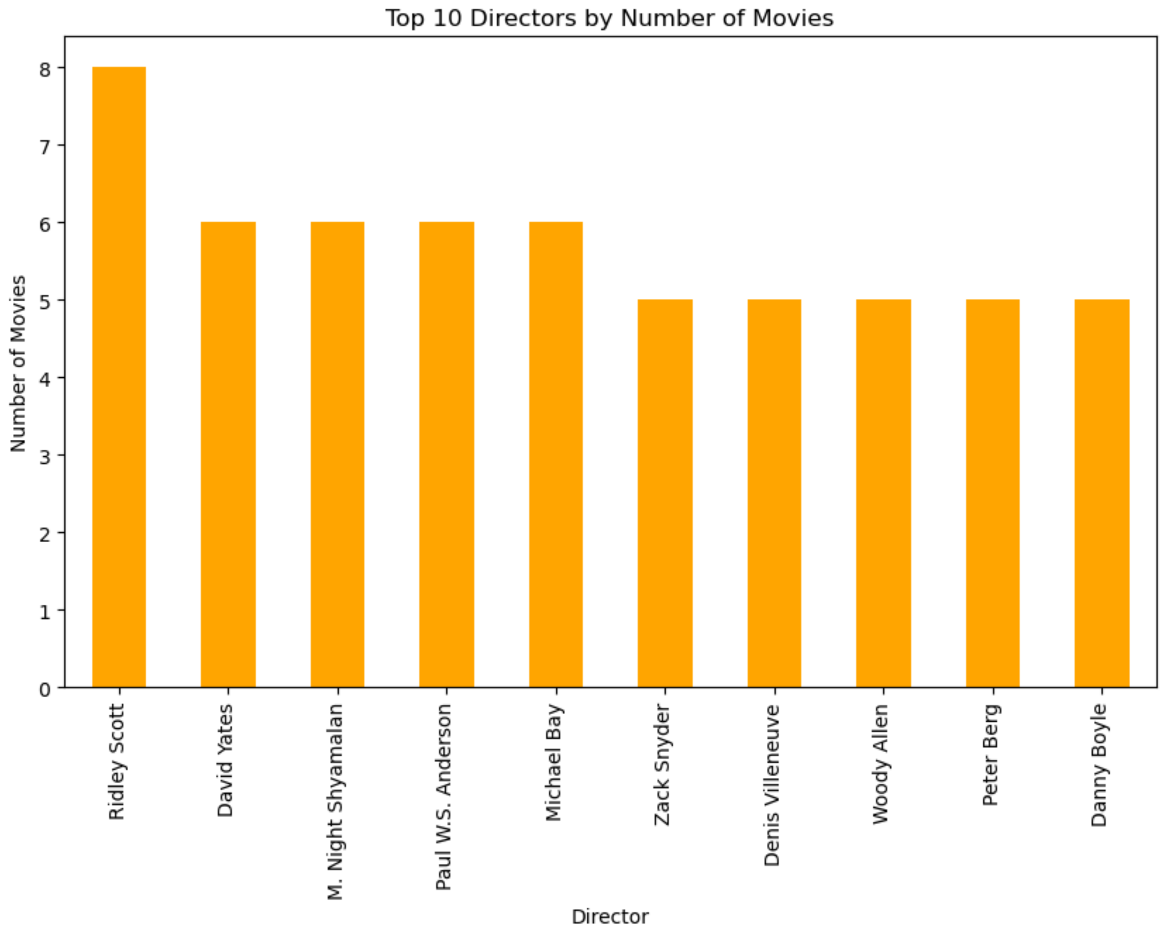
```
plt.ylabel('Revenue (Millions)')
plt.show()
```



5. Movies per Director

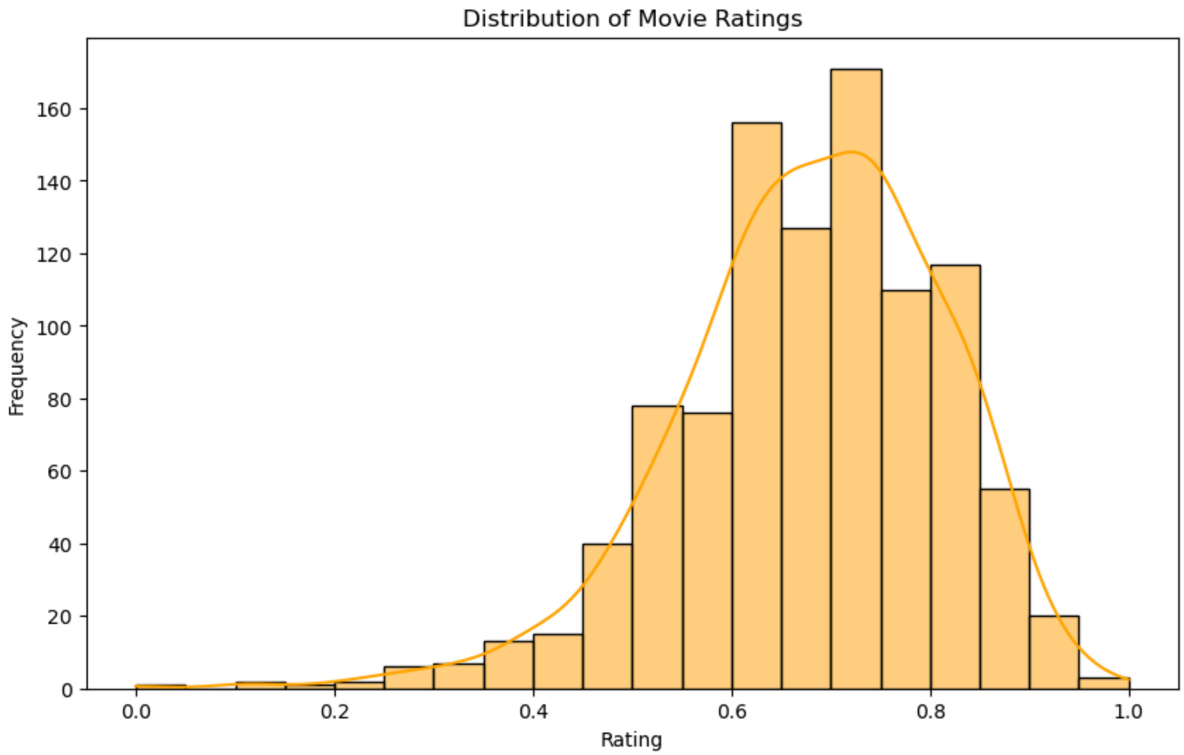
In [15]:

```
top_directors = df['Director'].value_counts().nlargest(10)
plt.figure(figsize=(10, 6))
top_directors.plot(kind='bar', color='Orange')
plt.title('Top 10 Directors by Number of Movies')
plt.xlabel('Director')
plt.ylabel('Number of Movies')
plt.xticks(rotation=90)
plt.show()
```



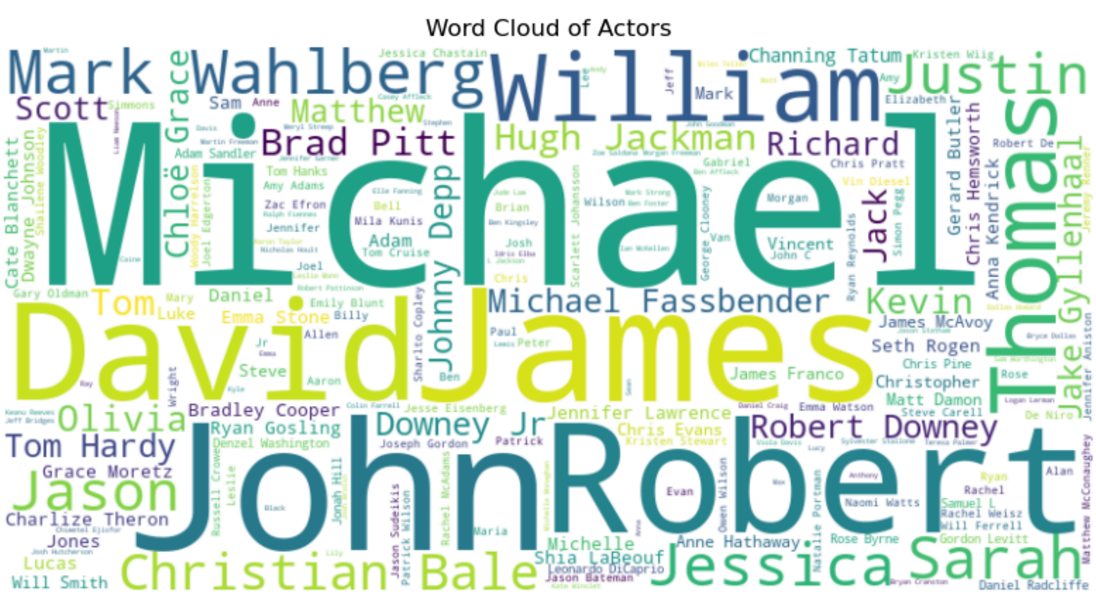
6. Distribution of Movie Ratings

In [16]:
```
plt.figure(figsize=(10, 6))
sns.histplot(df['Rating'], bins=20, kde=True, color='Orange')
plt.title('Distribution of Movie Ratings')
plt.xlabel('Rating')
plt.ylabel('Frequency')
plt.show()
```



7. Word Cloud of Actors

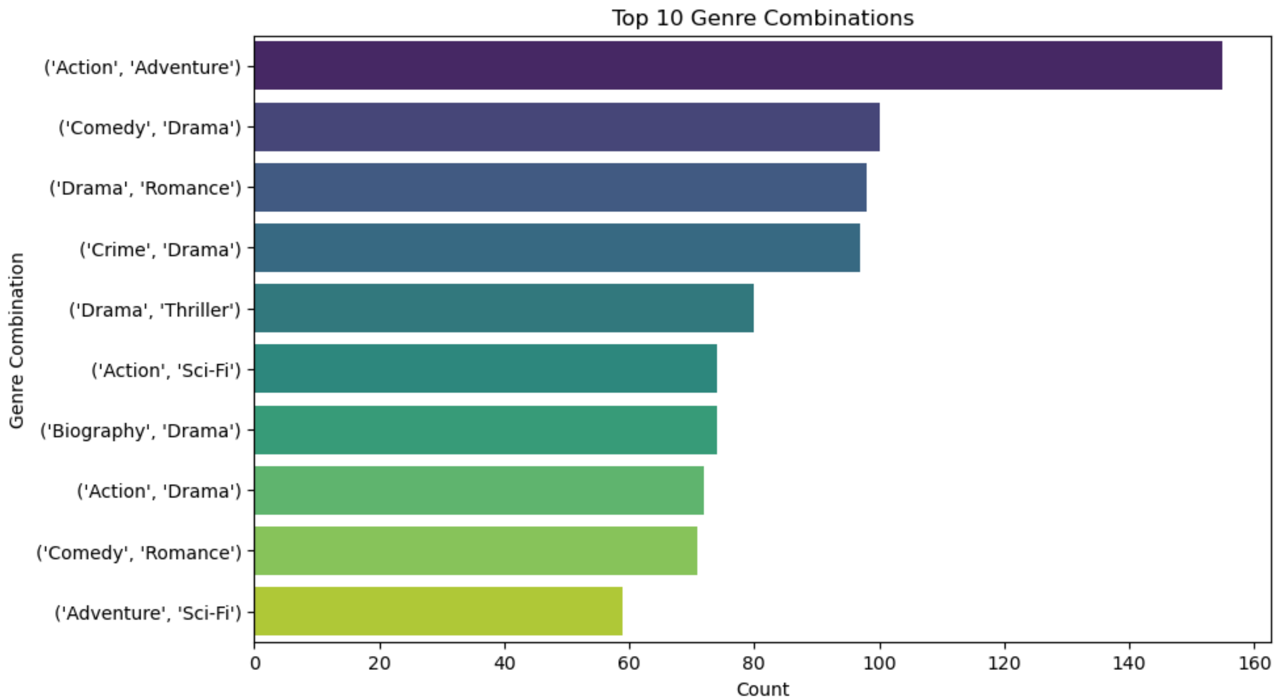
In [17]:
```
actors_text = ' '.join(df['Actors'].str.replace(',', '').values)
wordcloud = WordCloud(width=800, height=400, background_color='white').generate(actors_text)

plt.figure(figsize=(10, 6))
plt.imshow(wordcloud, interpolation='bilinear')
plt.title('Word Cloud of Actors')
plt.axis('off')
plt.show()
```



8. Genre Combinations Analysis

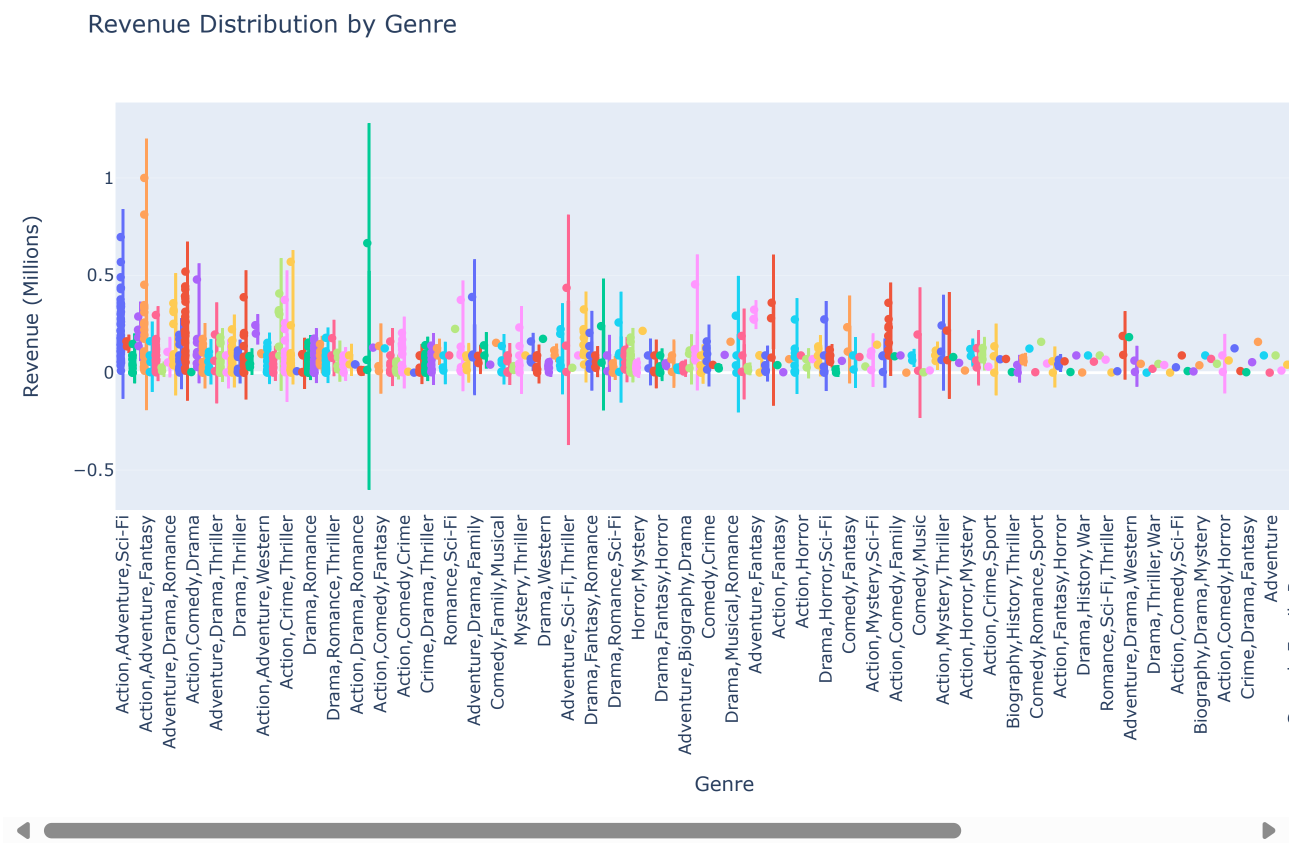
In [18]:
```
from itertools import combinations
from collections import Counter

genre_combinations = df['Genre'].str.split(',').apply(lambda x: list(combinations(x, 2)))
genre_combinations_flat = [item for sublist in genre_combinations for item in sublist]
genre_combinations_counter = Counter(genre_combinations_flat)
```

```python
# Plot the most common genre combinations
common_combinations = pd.DataFrame(genre_combinations_counter.most_common(10), columns=['Combination', 'Count'])
plt.figure(figsize=(10, 6))
sns.barplot(x='Count', y='Combination', data=common_combinations, palette='viridis')
plt.title('Top 10 Genre Combinations')
plt.xlabel('Count')
plt.ylabel('Genre Combination')
plt.show()
```



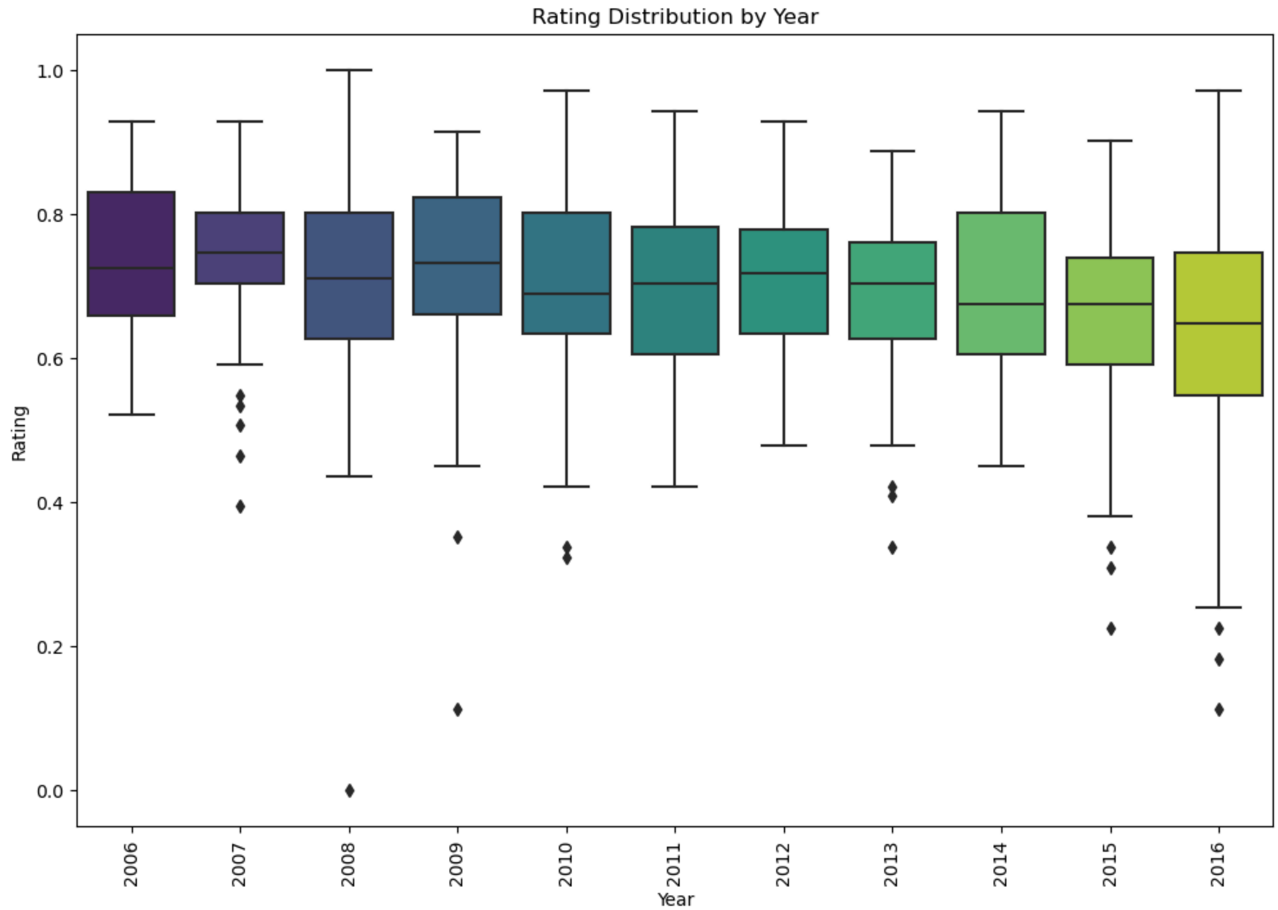
9. Revenue Distribution by Genre

In [19]:
```python
import plotly.express as px

# Revenue Distribution by Genre using Plotly
fig = px.violin(df, x='Genre', y='Revenue (Millions)', color='Genre', box=True, points='all',
                title='Revenue Distribution by Genre')
fig.update_layout(
    xaxis_title='Genre',
    yaxis_title='Revenue (Millions)',
    xaxis_tickangle=-90,
    width=1200,  # Adjust the width
    height=600   # Adjust the height
)
fig.show()
```



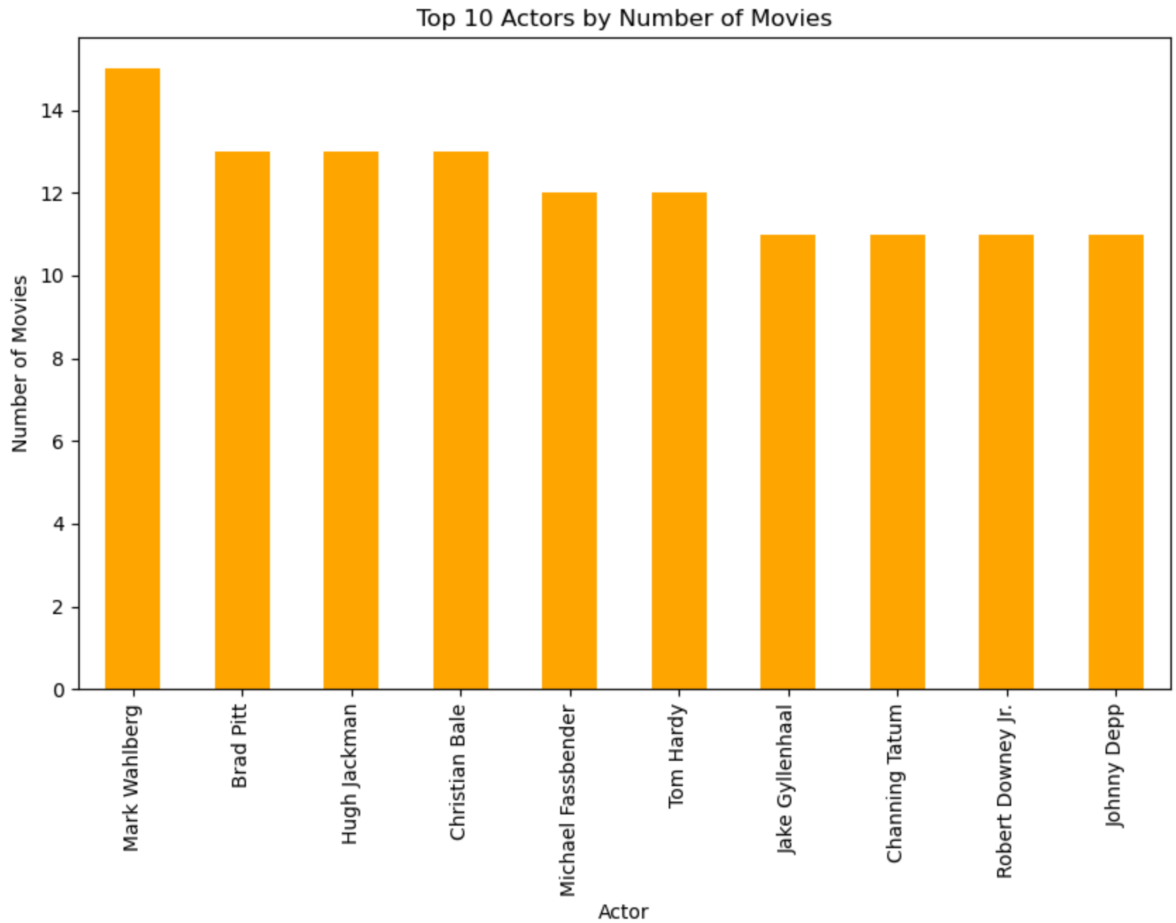
10. Rating Distribution by Year

In [20]:
```
plt.figure(figsize=(12, 8))
sns.boxplot(x='Year', y='Rating', data=df, palette='viridis')
plt.title('Rating Distribution by Year')
plt.xlabel('Year')
plt.ylabel('Rating')
plt.xticks(rotation=90)
plt.show()
```



11. Top Actors Analysis

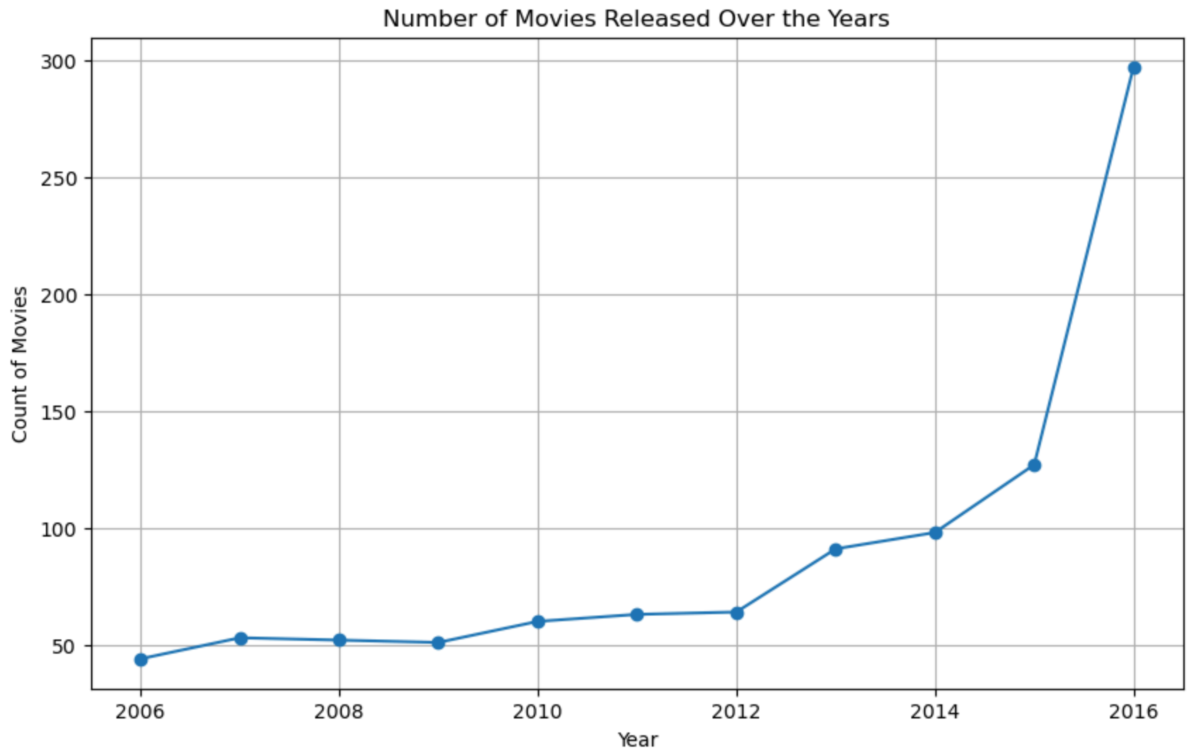
In [21]:
```
actors_list = df['Actors'].str.split(', ').explode()
top_actors = actors_list.value_counts().nlargest(10)
plt.figure(figsize=(10, 6))
top_actors.plot(kind='bar', color='Orange')
plt.title('Top 10 Actors by Number of Movies')
plt.xlabel('Actor')
plt.ylabel('Number of Movies')
plt.xticks(rotation=90)
plt.show()
```



Time Series Analysis

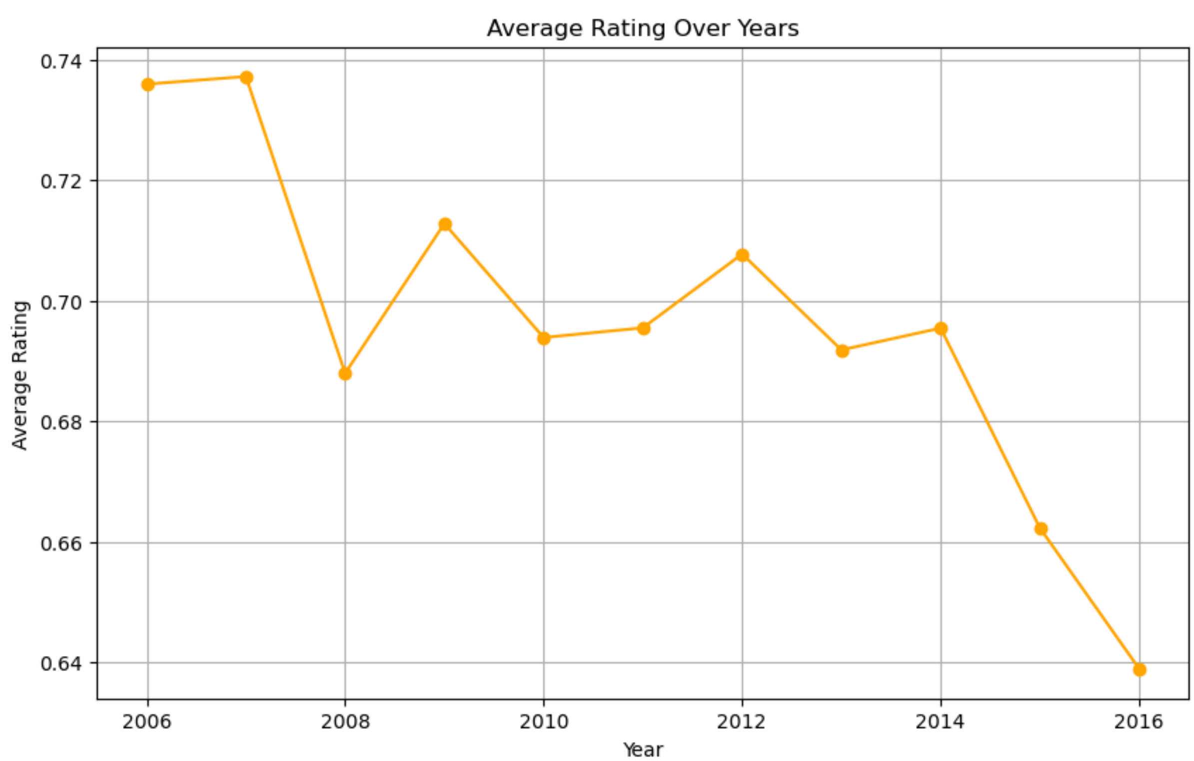
1. Time Series Analysis of Movies over Year

In [22]:
```
movie_over_years = df['Year'].value_counts().sort_index()
plt.figure(figsize=(10, 6))
plt.plot(movie_over_years, marker='o')
plt.xlabel('Year')
plt.ylabel('Count of Movies')
plt.grid(True)
plt.title('Number of Movies Released Over the Years')
plt.show()
```



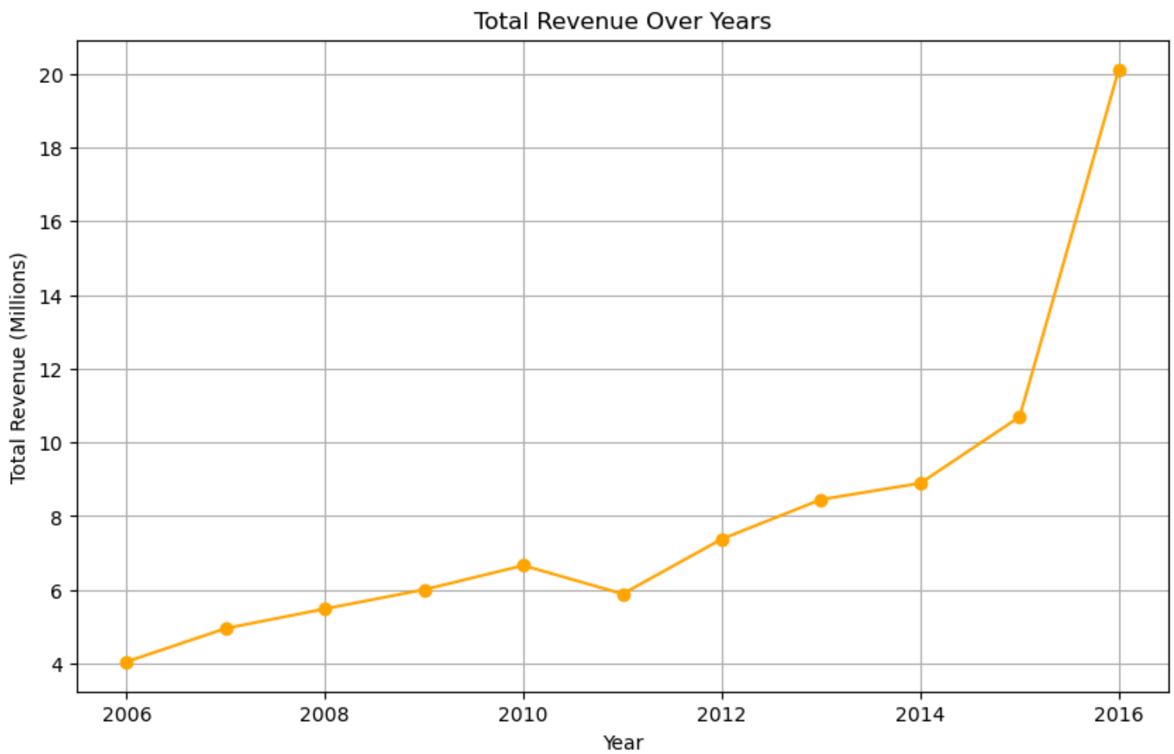
2. Time Series Analysis of Average Rating Over Years

In [23]:
```
ratings_over_years = df.groupby('Year')['Rating'].mean()
plt.figure(figsize=(10, 6))
ratings_over_years.plot(marker='o', color='orange')
plt.title('Average Rating Over Years')
plt.xlabel('Year')
plt.ylabel('Average Rating')
plt.grid(True)
plt.show()
```



3. Time Series Analysis of Revenue Over Years

In [24]:
```
revenue_over_years = df.groupby('Year')['Revenue (Millions)'].sum()
plt.figure(figsize=(10, 6))
revenue_over_years.plot(marker='o', color='orange')
plt.title('Total Revenue Over Years')
plt.xlabel('Year')
plt.ylabel('Total Revenue (Millions)')
plt.grid(True)
plt.show()
```



Predictive Modeling using Machine Learning Algorithms

1. For Movie Revenue (Linear Regression)

In [25]:
```
features = ['Rating', 'Runtime (Minutes)', 'Metascore', 'Year']
X = df[features].fillna(0)
y = df['Revenue (Millions)'].fillna(0)

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Predictions using the model
y_pred_train = model.predict(X_train)
y_pred_test = model.predict(X_test)

# Evaluation of the model
train_rmse = np.sqrt(mean_squared_error(y_train, y_pred_train))
test_rmse = np.sqrt(mean_squared_error(y_test, y_pred_test))

print("Train RMSE:", train_rmse)
print("Test RMSE:", test_rmse)
```

```
Train RMSE: 0.09896805404834702
Test RMSE: 0.09808285077921733
```

2. For Movie Rating Category- High/Low (Random Forest)

In [26]:
```
rating_threshold = df['Rating'].quantile(0.75)
df['Rating_Category'] = df['Rating'].apply(lambda x: 'High' if x >= rating_threshold else 'Low')

# Feature Engineering
features = ['Genre', 'Director', 'Actors', 'Runtime (Minutes)', 'Year']
X = df[features]
y = df['Rating_Category']

# Encode categorical features
label_encoders = {}
for column in ['Genre', 'Director', 'Actors']:
    label_encoders[column] = LabelEncoder()
    X[column] = label_encoders[column].fit_transform(X[column])

# Handle missing values
X.fillna(0, inplace=True)

# Model Training
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train, y_train)

# Make predictions
y_pred = clf.predict(X_test)

# Model Evaluation
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
```

```
Accuracy: 0.81
Classification Report:
               precision    recall  f1-score   support

        High       0.66      0.44      0.53        48
         Low       0.84      0.93      0.88       152

    accuracy                           0.81       200
   macro avg       0.75      0.68      0.70       200
weighted avg       0.80      0.81      0.80       200
```

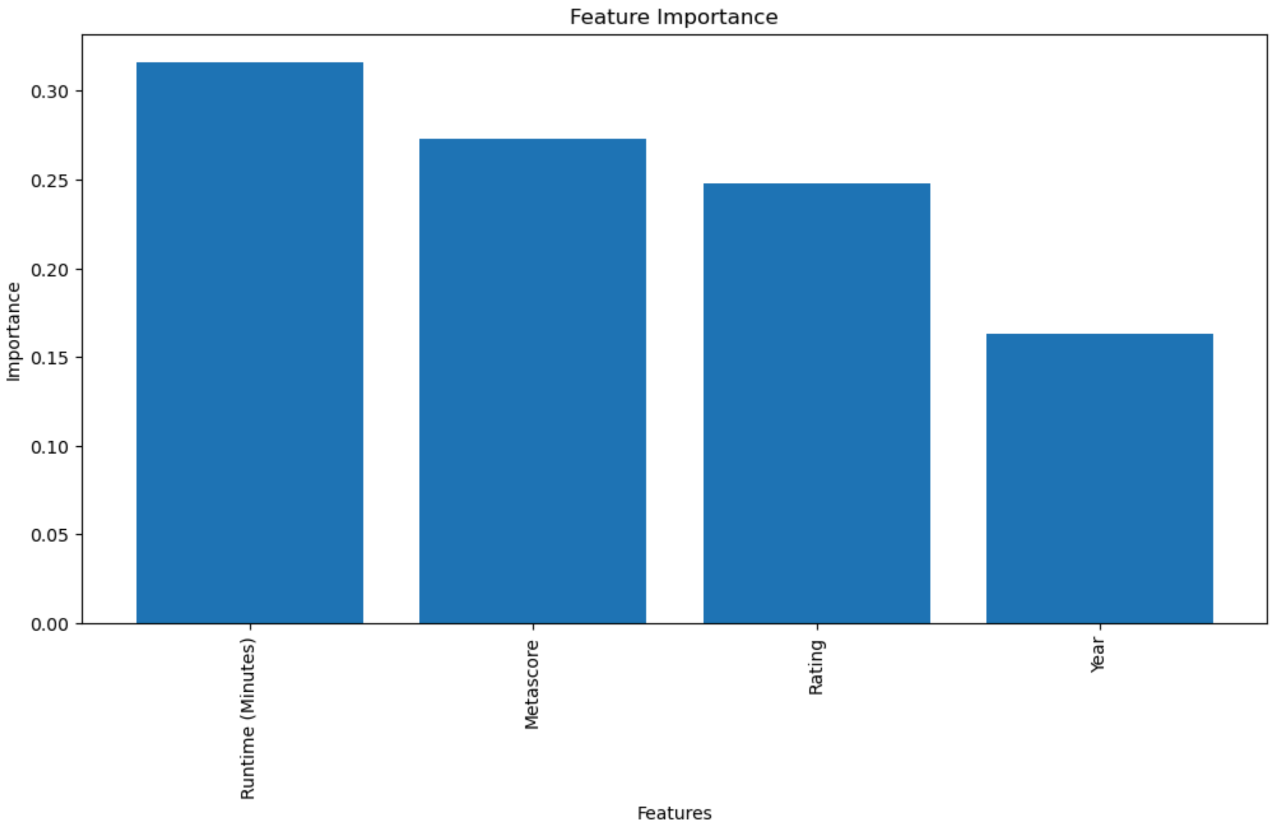
3. Predicting Movie Success (Binary Classification)

In [27]:
```
success_threshold = df['Revenue (Millions)'].quantile(0.75)
df['Success'] = df['Revenue (Millions)'].apply(lambda x: 1 if x >= success_threshold else 0)

# Feature Engineering
features = ['Rating', 'Runtime (Minutes)', 'Metascore', 'Year']
X = df[features].fillna(0)
y = df['Success']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train logistic regression model
from sklearn.linear_model import LogisticRegression
log_reg = LogisticRegression()
log_reg.fit(X_train, y_train)

# Make predictions
y_pred = log_reg.predict(X_test)

# Model Evaluation
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
```

```
Accuracy: 0.76
Classification Report:
               precision    recall  f1-score   support

           0       0.76      1.00      0.86       152
           1       0.00      0.00      0.00        48

    accuracy                           0.76       200
   macro avg       0.38      0.50      0.43       200
weighted avg       0.58      0.76      0.66       200
```

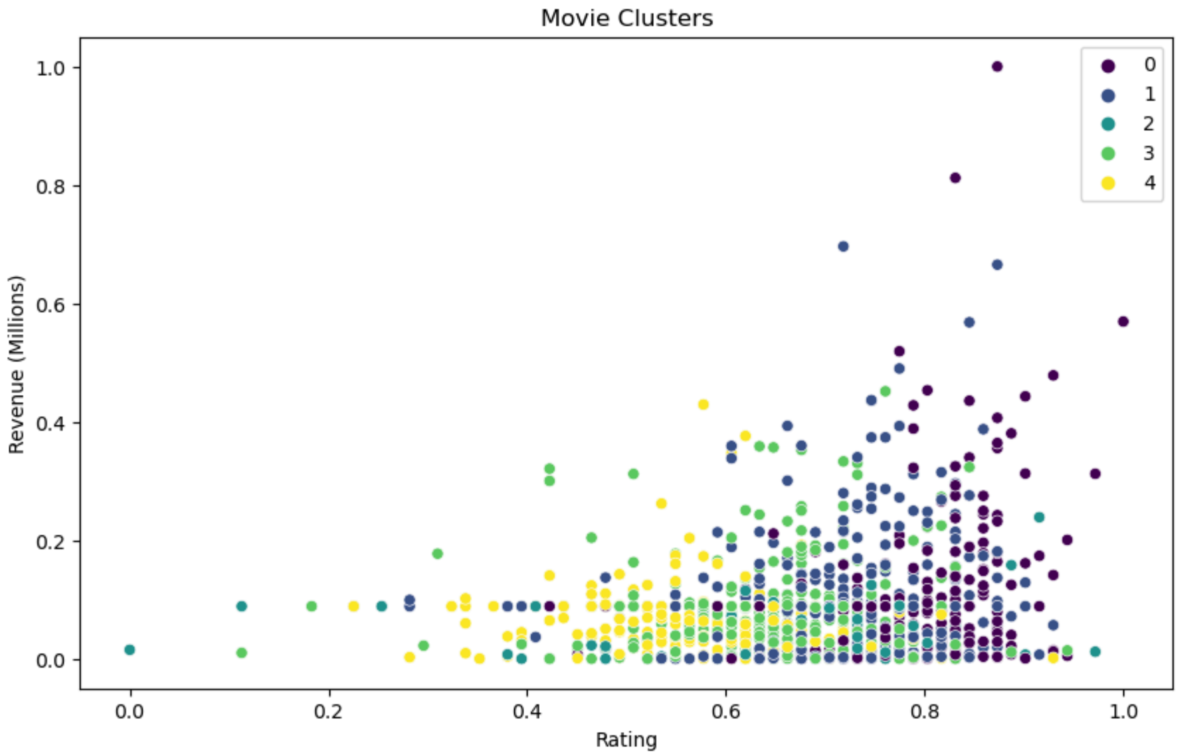
4. Feature Importance Analysis

In [28]:
```
clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train, y_train)

# Plot feature importance
importances = clf.feature_importances_
indices = np.argsort(importances)[::-1]
feature_names = [features[i] for i in indices]

plt.figure(figsize=(12, 6))
plt.title('Feature Importance')
plt.bar(range(X_train.shape[1]), importances[indices], align='center')
plt.xticks(range(X_train.shape[1]), feature_names, rotation=90)
plt.xlabel('Features')
plt.ylabel('Importance')
plt.show()
```



5. Clustering Movies

In [29]:
```
kmeans = KMeans(n_clusters=5, random_state=42)
df['Cluster'] = kmeans.fit_predict(X)

# Visualize clusters
plt.figure(figsize=(10, 6))
sns.scatterplot(x='Rating', y='Revenue (Millions)', hue='Cluster', data=df, palette='viridis')
plt.title('Movie Clusters')
plt.xlabel('Rating')
plt.ylabel('Revenue (Millions)')
plt.legend()
plt.show()
```



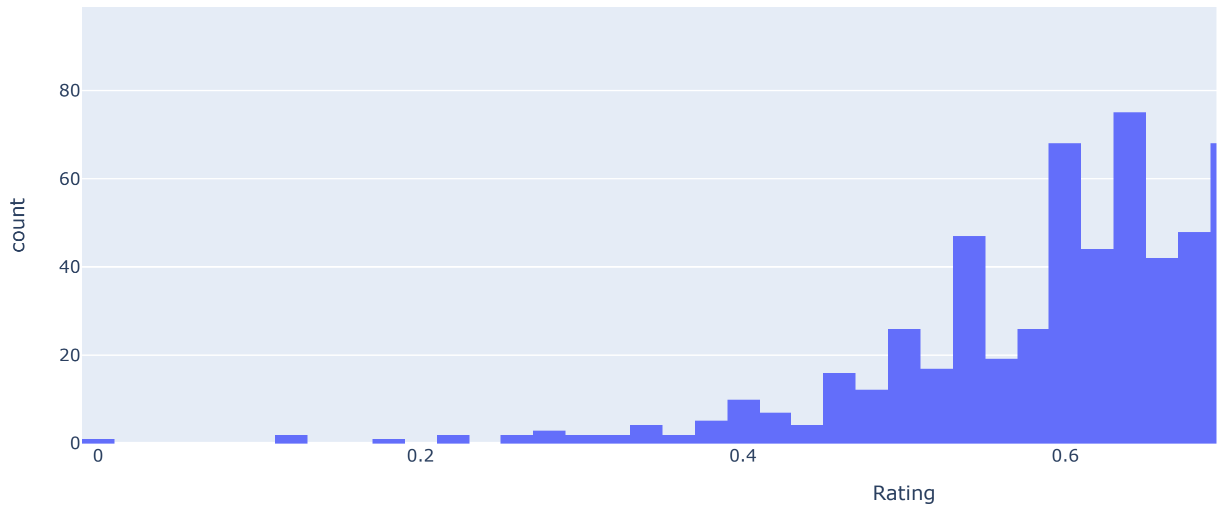
Visualizations

1. Interactive Dashboards

In [30]:
```python
# Example using Plotly Dash (Please run this as a separate script)
import dash
import dash_core_components as dcc
import dash_html_components as html
from dash.dependencies import Input, Output

app = dash.Dash(__name__)

app.layout = html.Div([
    html.H1("IMDb Movies Analysis Dashboard"),
    dcc.Dropdown(id='feature-dropdown', options=[
        {'label': 'Rating', 'value': 'Rating'},
        {'label': 'Revenue (Millions)', 'value': 'Revenue (Millions)'},
        {'label': 'Runtime (Minutes)', 'value': 'Runtime (Minutes)'}
    ], value='Rating'),
    dcc.Graph(id='feature-graph')
])

@app.callback(
    Output('feature-graph', 'figure'),
    [Input('feature-dropdown', 'value')]
)
def update_graph(selected_feature):
    fig = px.histogram(df, x=selected_feature)
    return fig

if __name__ == '__main__':
    app.run_server(debug=True)
```

IMDb Movies Analysis Dashboard

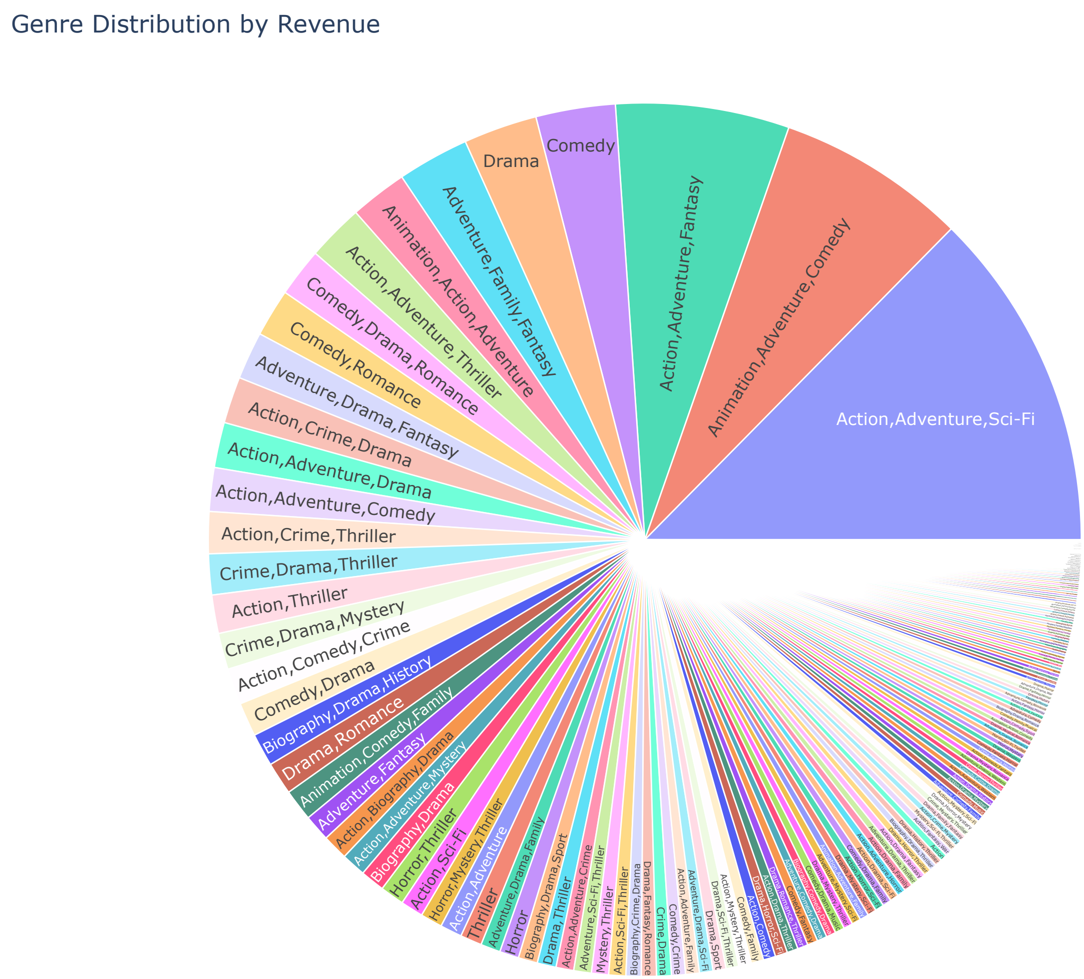
Rating



2. Sunburst Chart for Genre Distribution

In [31]:

```
import plotly.express as px

fig = px.sunburst(df, path=['Genre'], values='Revenue (Millions)', title='Genre Distribution by Revenue')

fig.update_layout(
    width=1000,  # Set the width
    height=800   # Set the height
)


fig.show()
```

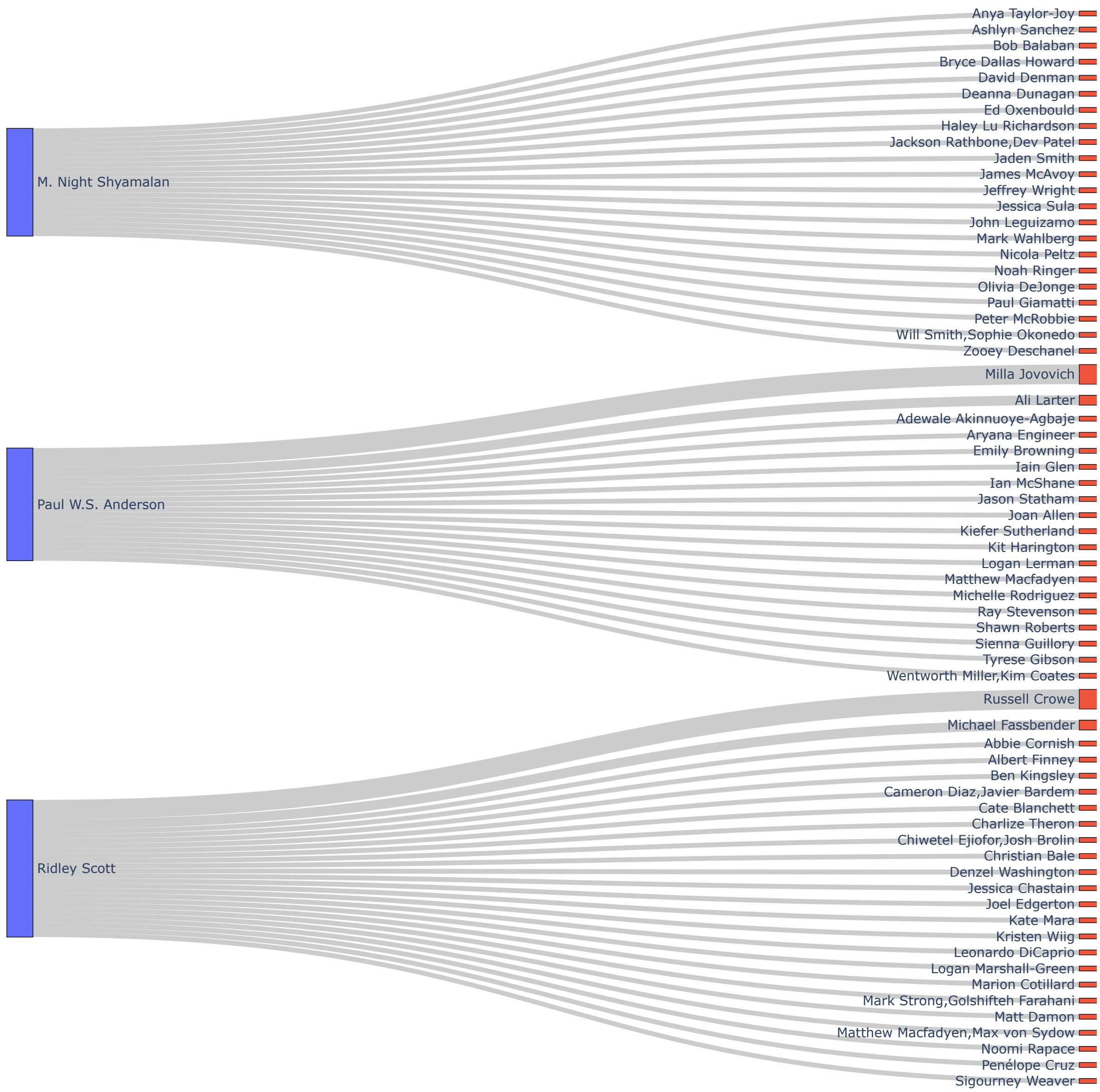


3. Chord Diagram for Director-Actor Collaborations

In [32]:
```
import plotly.graph_objects as go

# Prepare data for chord diagram
director_actor_pairs = df[['Director', 'Actors']].dropna()
director_actor_pairs['Actors'] = director_actor_pairs['Actors'].str.split(', ')
pairs = director_actor_pairs.explode('Actors')

# Count the number of collaborations
collaborations = pairs.groupby(['Director', 'Actors']).size().reset_index(name='Count')

# Get the top 5 directors by the number of collaborations
top_3_directors = collaborations['Director'].value_counts().nlargest(3).index.tolist()
top_director_collaborations = collaborations[collaborations['Director'].isin(top_3_directors)]

# Create a list of unique directors and actors
directors = top_director_collaborations['Director'].unique().tolist()
actors = top_director_collaborations['Actors'].unique().tolist()

# Create a combined list of nodes
nodes = directors + actors

# Create a dictionary to index nodes
node_indices = {node: idx for idx, node in enumerate(nodes)}

# Create the links for the chord diagram
links = []
for _, row in top_director_collaborations.iterrows():
```

```python
    source = node_indices[row['Director']]
    target = node_indices[row['Actors']]
    value = row['Count']
    links.append({'source': source, 'target': target, 'value': value})

# Create the node colors
colors = ['#636EFA'] * len(directors) + ['#EF553B'] * len(actors)

# Create the Plotly figure
fig = go.Figure(data=[go.Sankey(
    node=dict(
        pad=15,
        thickness=20,
        line=dict(color="black", width=0.5),
        label=nodes,
        color=colors
    ),
    link=dict(
        source=[link['source'] for link in links],
        target=[link['target'] for link in links],
        value=[link['value'] for link in links]
    )
)])

# Update the layout to make the figure larger
fig.update_layout(
    title_text="Director-Actor Collaborations (Top 3 Director)",
    font_size=10,
    width=1000,  # Set the width
    height=1000  # Set the height
)

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

Director-Actor Collaborations (Top 3 Director)



In [ ]: