Download the dataset from: https://github.com/bellawillrise/Introduction-to-Numerical-Computing-in-Python/

Submit a pdf file, which is a rendered saved version of the jupyter notebook. Make sure to execute all the codes so the output can be viewed in the pdf.

Also include the link to the public github repository where the jupyter notebook for the assignment is uploaded.

Link to the github repository: https://github.com/chstrkn/CMSC197

In [3]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [4]:
```
# %matplotlib inline
```

In [5]:
```
data = pd.read_csv("data/movie_metadata_cleaned.csv")
```

In [6]:
```
data.head(2)
```

Out[6]:

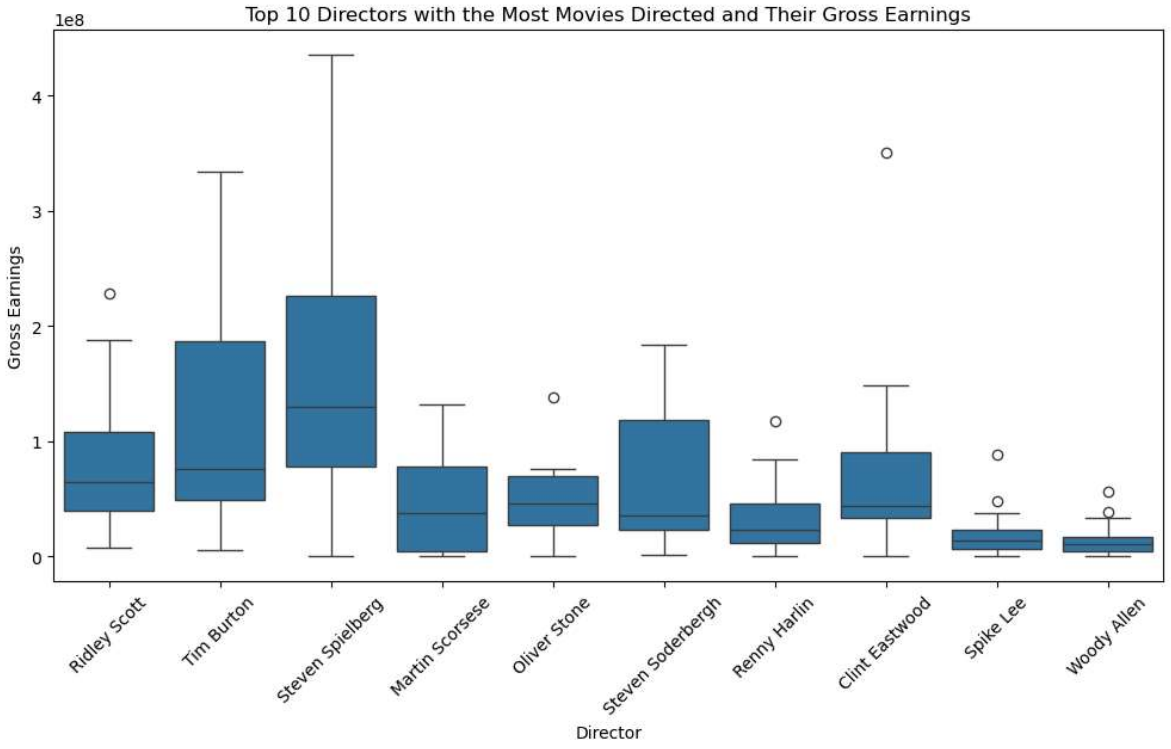
	Unnamed: 0	movie_title	color	director_name	num_critic_for_reviews	duration	direc
0	0	b'Avatar'	Color	James Cameron	723.0	178.0	
1	1	b"Pirates of the Caribbean: At World's End"	Color	Gore Verbinski	302.0	169.0	

2 rows × 29 columns

Get the top 10 directors with most movies directed and use a boxplot for their gross earnings

In [8]:
```
# Calculate the number of movies each director (excluding "0") has directed
director_movie_count = data[data['director_name'] != "0"]['director_name'].value

# Select the top 10 directors with the most movies directed
top_directors = director_movie_count.head(10).index

# Filter the data to include only these top 10 directors
top_directors_data = data[data['director_name'].isin(top_directors)]

# Plot a boxplot of the gross earnings for the top 10 directors
plt.figure(figsize=(12, 6))
```

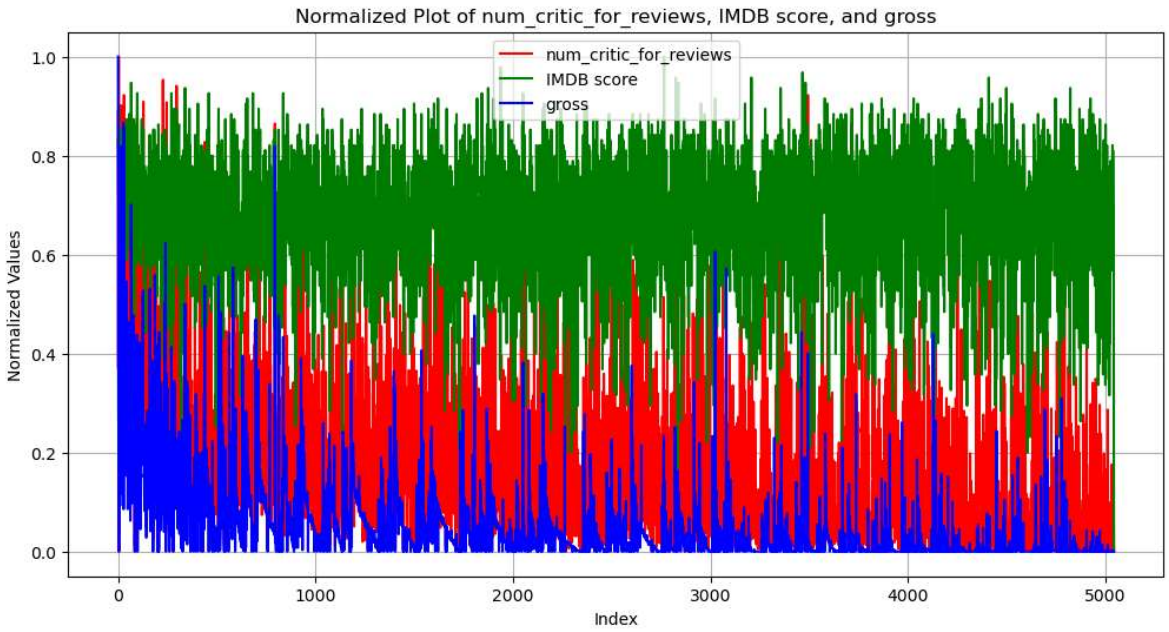
```
sns.boxplot(x='director_name', y='gross', data=top_directors_data)
plt.xticks(rotation=45)
plt.title('Top 10 Directors with the Most Movies Directed and Their Gross Earnin
plt.xlabel('Director')
plt.ylabel('Gross Earnings')
plt.show()
```



Plot the following variables in one graph:

- num_critic_for_reviews
- IMDB score
- gross

In [41]:
```
# Specify the columns to normalize
columns = ['num_critic_for_reviews', 'imdb_score', 'gross']

# Normalize the specified columns
normalized_data = (data[columns] - data[columns].min()) / (data[columns].max() -

# Plot the normalized data
plt.figure(figsize=(12, 6))
plt.plot(normalized_data.index, normalized_data['num_critic_for_reviews'], label
plt.plot(normalized_data.index, normalized_data['imdb_score'], label='IMDB score
plt.plot(normalized_data.index, normalized_data['gross'], label='gross', color='

plt.xlabel('Index')
plt.ylabel('Normalized Values')
plt.title('Normalized Plot of num_critic_for_reviews, IMDB score, and gross')

plt.legend()
plt.grid(True)
plt.show()
```



Compute Sales (Gross - Budget), add it as another column

In [12]:
```
# Subtract budget from gross
data['sales'] = data['gross'] - data['budget']

# Display the data
data['sales']
```

Out[12]:
```
0       523505847.0
1         9404152.0
2       -44925825.0
3       198130642.0
4               0.0
           ...
5039            0.0
5040        -1400.0
5041        10443.0
5042        84122.0
5043            0.0
Name: sales, Length: 5044, dtype: float64
```

Which directors garnered the most total sales?

In [14]:
```
# Calculate the total sales for each director
total_sales = data.groupby('director_name')['sales'].sum().reset_index()

# Sort in descending order
total_sales = total_sales.sort_values(by='sales', ascending=False)

# Filter the data to include only the top 10 directors
top_directors_sales = total_sales[:10]

# Display the data
top_directors_sales
```

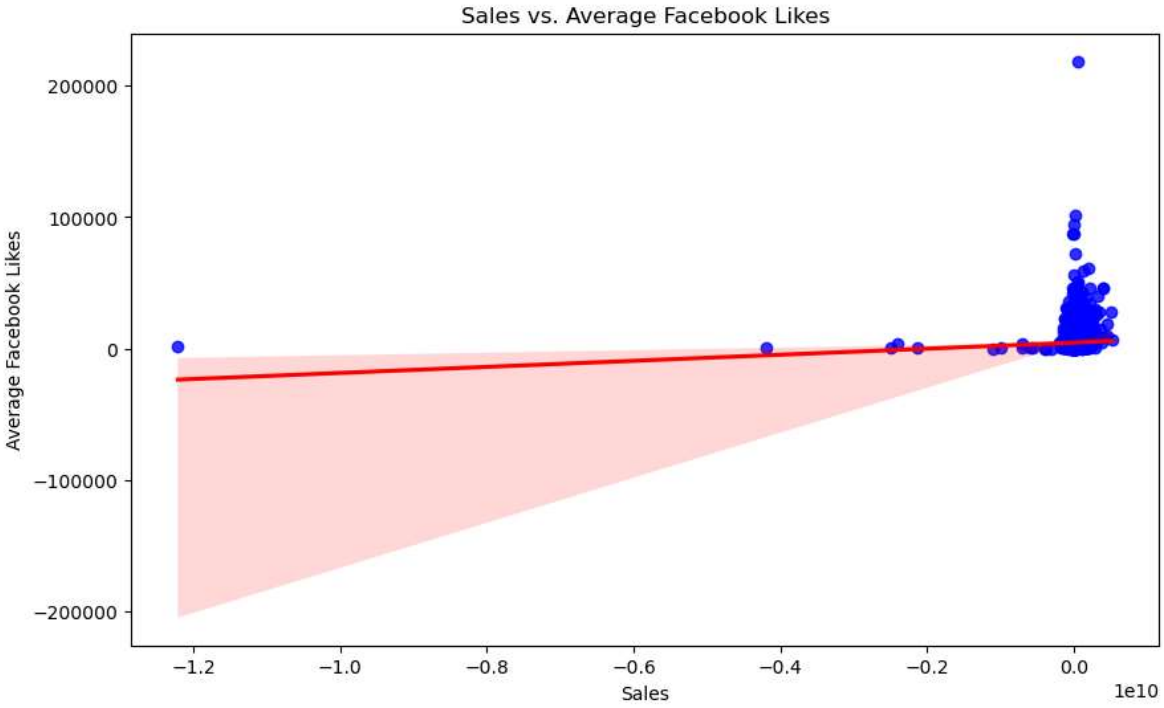
Out[14]:

	director_name	sales
2159	Steven Spielberg	2.451332e+09
765	George Lucas	1.386641e+09
923	James Cameron	1.199626e+09
1219	Joss Whedon	1.000887e+09
335	Chris Columbus	9.417076e+08
1787	Peter Jackson	9.009693e+08
2221	Tim Burton	8.242755e+08
374	Christopher Nolan	8.082276e+08
1158	Jon Favreau	7.693815e+08
695	Francis Lawrence	7.555020e+08

Plot sales and average likes as a scatterplot. Fit it with a line.

In [16]:

```
# Calculate the average number of Facebook likes
data['average_facebook_likes'] = data[['director_facebook_likes',
                                       'actor_1_facebook_likes',
                                       'actor_2_facebook_likes',
                                       'actor_3_facebook_likes',
                                       'cast_total_facebook_likes',
                                       'movie_facebook_likes']].mean(axis=1)

# Plot a scatterplot with a regression line
plt.figure(figsize=(10, 6))
sns.regplot(
    x=data['sales'],
    y=data['average_facebook_likes'],
    scatter_kws={'color': 'blue'},
    line_kws={'color': 'red'}
)

# Display the plot
plt.xlabel('Sales')
plt.ylabel('Average Facebook Likes')
plt.title('Sales vs. Average Facebook Likes')
plt.show()
```

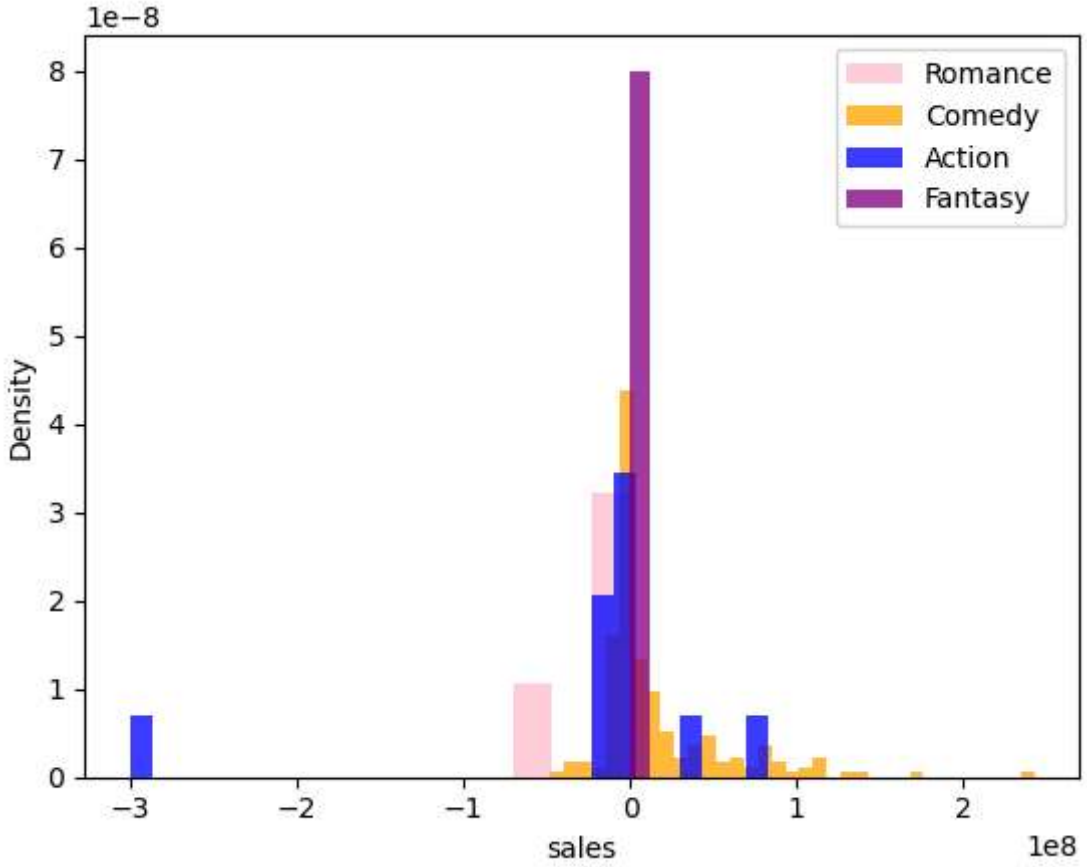


Which of these genres are the most profitable? Plot their sales using different histograms, superimposed in the same axis.

- Romance
- Comedy
- Action
- Fantasy

In [18]:
```
# Create a histogram for the 'Romance' genre sales
ax = sns.histplot(data[data['genres'] == 'Romance']["sales"], color="pink", labe

# Create a histogram for the 'Comedy' genre sales
sns.histplot(data[data['genres'] == 'Comedy']["sales"], color="orange", label="C

# Create a histogram for the 'Action' genre sales
sns.histplot(data[data['genres'] == 'Action']["sales"], color="blue", label="Act

# Create a histogram for the 'Fantasy' genre sales
sns.histplot(data[data['genres'] == 'Fantasy']["sales"], color="purple", label="

# Add legend to identify the genres
ax.legend()
```

Out[18]:
```
<matplotlib.legend.Legend at 0x1f54ce7b650>
```



For each of movie, compute average likes of the three actors and store it as a new variable

Read up on the mean function.

Store it as a new column, average_actor_likes.

In [20]:
```
# Calculate the average Facebook likes of the three actors
data['average_actor_likes'] = data[['actor_1_facebook_likes', 'actor_2_facebook_

# Display the data
data['average_actor_likes']
```

Out[20]:
```
0         930.333333
1       15333.333333
2        3851.333333
3       24333.333333
4          47.666667
             ...
5039      584.333333
5040        0.000000
5041      718.000000
5042       41.666667
5043        0.000000
Name: average_actor_likes, Length: 5044, dtype: float64
```

Copying the whole dataframe

In [22]:
```
df = data.copy()
df.head()
```

Out[22]:

	Unnamed: 0	movie_title	color	director_name	num_critic_for_reviews	duration	direc
0	0	b'Avatar'	Color	James Cameron	723.0	178.0	
1	1	b"Pirates of the Caribbean: At World's End"	Color	Gore Verbinski	302.0	169.0	
2	2	b'Spectre'	Color	Sam Mendes	602.0	148.0	
3	3	b'The Dark Knight Rises'	Color	Christopher Nolan	813.0	164.0	
4	4	b'Star Wars: Episode VII - The Force Awakens ...	0	Doug Walker	0.0	0.0	

5 rows × 32 columns

Min-Max Normalization

Normalization is a technique often applied as part of data preparation for machine learning. The goal of normalization is to change the values of numeric columns in the dataset to a common scale, without distorting differences in the ranges of values. For machine learning, every dataset does not require normalization. It is required only when features have different ranges.

The min-max approach (often called normalization) rescales the feature to a hard and fast range of [0,1] by subtracting the minimum value of the feature then dividing by the range. We can apply the min-max scaling in Pandas using the .min() and .max() methods.

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

Normalize each numeric column (those that have types integer or float) of the copied dataframe (df)

In [25]:
```
# Select the numeric columns
numeric_columns = df.select_dtypes(include=['float64', 'int64'])

# Apply Min-Max normalization
normalized_data = (numeric_columns - numeric_columns.min()) / (numeric_columns.m
```

```
# Display the normalized data
normalized_data
```

Out[25]:

	Unnamed: 0	num_critic_for_reviews	duration	director_facebook_likes	actor_3_faceb
0	0.000000	0.889299	0.941799	0.000000	
1	0.000198	0.371464	0.894180	0.024478	
2	0.000397	0.740467	0.783069	0.000000	
3	0.000595	1.000000	0.867725	0.956522	
4	0.000793	0.000000	0.000000	0.005696	
...	...	...	...	...	
5039	0.999207	0.052891	0.227513	0.000000	
5040	0.999405	0.015990	0.402116	0.000000	
5041	0.999603	0.017220	0.529101	0.000000	
5042	0.999802	0.052891	0.476190	0.000696	
5043	1.000000	0.000000	0.000000	0.000000	

5044 rows × 20 columns