Project 4: Visualization with Matplotlib

John Wesley Mathis

Dr. Anthony Choi

June 15, 2024

ECE/SSE 591, Summer 2024

Table of Contents:

Table of Figures

Deliverable Table

The purpose of this table is to provide a complete view of the concepts covered in chapter 4 of *"Python Data Science Handbook"* (VanderPlas, 2016) and provide a general page location for where the topic was demonstrated.

Deliverables	Location
Simple Line Plots	p. 9-10
Simple Scatter Plots	p. 11-13
Visualizing Errors	p.34
Density and Contour Plots	p. 16
Histograms, Binnings, and Density	p. 23
Customizing Plot Legends	p. 17-18
Customizing Colorbars	p. 36
Multiple Subplots	p. 37-38
Text and Annotation	p. 11
Customizing Ticks	p. 26
Customizing Matplotlib: Configurations and Stylesheets	p. 8
Three-Dimensional Plotting in Matplotlib	p. 14-15
Geographic Data with Basemap	p. 36
Visualization with Seaborn	p. 40-41

Additionally, here is a link to my GitHub were the datasets and the Jupyter Notebook for the project can be downloaded: https://github.com/jwmathis/SSE591_Project4. In order to run the file, Python and other dependencies must be installed.

1. Introduction

Matplotlib provides a comprhensive and flexible interface for creating static, animated, and interactive visualizations in Python. While libraries like Pandas and NumPy are essential for data manipulation and numerical computations, Matplotlib excels at presenting this data in a visual format that can uncover insights and trends. Because of its wide range of plotting functions and customization options, it makes it an invaluable tool for data scientists who aim to present their data clearly and effectively. Additionally, its integration with Pandas and NumPy allows for seamless data visualization directly from the libraries respective data structures.

This report aims to demonstrate my proficiency in Python data visualization techniques as covered in Chapter 4 of the "Python Data Science Handbook" by Jake VanderPlas (2016). This report attempts to illustrate the core concepts and functionalities of the Pandas library by implementing the concepts into practical examples. The code presented in this report was developed using Visual Studio Code with Jupyter Notebook extensions. I will provide detailed explanations, highlighting key features and operations that make Matplotlib an essential tool for data analysis.

2. Adapting SIR Model for Visualization

Before beginning any of the projects, I first imported the necessary libraries that I would. Figure 1 shows the libraries I used for the entire project. Additionally, I used `*plt.style.use()*` to customize Matplotlib to my likening. For this entire project I used *ggplot*. Along with the necessary libraries, I also imported a couple of toolkits.

Import Libraries

```
%matplotlib inline
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
plt.style.use('ggplot')
from mpl_toolkits.mplot3d import Axes3D
from mpl_toolkits.basemap import Basemap
```

[1] ✓ 1.6s

Figure 1: Importing Libraries and Customizing Matplotlib

Using my previous SIR model from Project 2 that covered using the NumPy library, I decided to revisit the project code to construct graphs for various scenarios using Matplotlib. I first began by transferring the necessary code from my modeling infection spread to recreate the simulated data. I also transferred the Monte Carlo simulation code to include in the visualization. Figure 2 below shows the code. I began constructing simple line plots. The first line plot demonstrates how to plot multiple sets of data on the same graph and how to annotate the graph and change the line styles and colors using appropriate parameters. Figures 3 and 4 below shows the code and the output graph.

1. Adapting Infection Spread Model for Visualization

Modeling infection spread using SIR model

```python
def SIR_model(S0, I0, R0, beta, gamma, days):
    dtype = [('day', int), ('susceptible', float), ('infected', float), ('recovered', float)]
    #Initialize arrays
    data = np.zeros(days, dtype=dtype)
    data['day'] = np.arange(days)
    data['susceptible'][0] = S0
    data['infected'][0] = I0
    data['recovered'][0] = R0

    #Total population
    N = S0 + I0 + R0

    #Calculate values
    for t in range(1, days):
        data['susceptible'][t] = data['susceptible'][t-1] - beta * data['susceptible'][t-1] * data['infected'][t-1] / N
        data['infected'][t] = data['infected'][t-1] + beta * data['susceptible'][t-1] * data['infected'][t-1] / N - gamma * data['infected'][t-1]
        data['recovered'][t] = data['recovered'][t-1] + gamma * data['infected'][t-1]

    return data

# -------------------------
# Main Program
# -------------------------

# Initial values
S0 = 990 #Number of susceptible individuals
I0 = 10 #Number of infected individuals
R0 = 0 #Number of recovered individuals
beta = 0.3 #infection rate
gamma = 0.1 #recovery rate
days = 160 #Number of days to simulate

SIR_data = SIR_model(S0, I0, R0, beta, gamma, days)

# Find peak number of infections
peak_infections = np.max(SIR_data['infected'])
peak_day = np.argmax(SIR_data['infected'])
```

Figure 2: SIR Model Code

Line Plots of SIR Model Dynamics

```python
plt.figure(figsize=(8,4))
plt.plot(SIR_data['day'], SIR_data['susceptible'], '--k', label='Susceptible')
plt.plot(SIR_data['day'], SIR_data['infected'], '-.', label='Infected', color='orange')
plt.plot(SIR_data['day'], SIR_data['recovered'], label='Recovered', color='c')
#plt.axvline(peak_day, color='black', linestyle='--')
plt.plot([peak_day, peak_day], [0, peak_infections], '.')
plt.text(peak_day, peak_infections, f'Peak: {peak_infections:.0f}',
         verticalalignment='bottom', horizontalalignment='left', color='black')
plt.xlabel('Days')
plt.ylabel('Population')
plt.title('SIR Model Dynamics')
plt.legend();
```

Figure 3: SIR Model Dynamics Line Plot Code

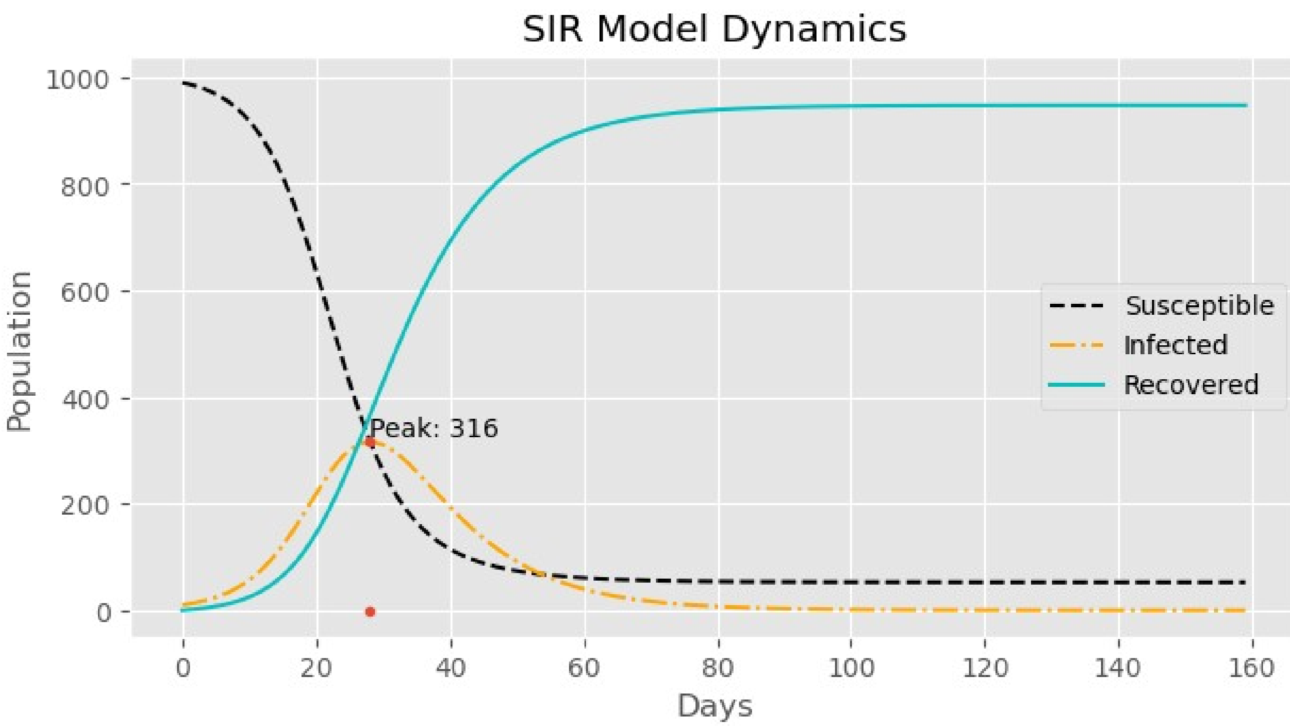

Figure 4: SIR Model Dynamics Line Plot

To further demonstrate how to annotate a graph, I isolated the line plot for infected. Then I added a dashed vertical line along with the peak value to represent on the graph the peak infection day. Figures 5 and 6 show the code and output plot.

Figure and Code

```
plt.figure(figsize=(10, 6))
plt.plot(SIR_data['day'], SIR_data['infected'], label='Infected', color='red')
plt.axvline(peak_day, color='black', linestyle='--', label='Peak Infection Day')
plt.text(peak_day, peak_infections, f'Peak: {peak_infections:.0f}',
         verticalalignment='bottom', horizontalalignment='right', color='black')
plt.xlabel('Days')
plt.ylabel('Infected Individuals')
plt.title('Peak Infection Day Annotation')
plt.legend()
```

Figure 5: Peak Infection Day Annotation Code

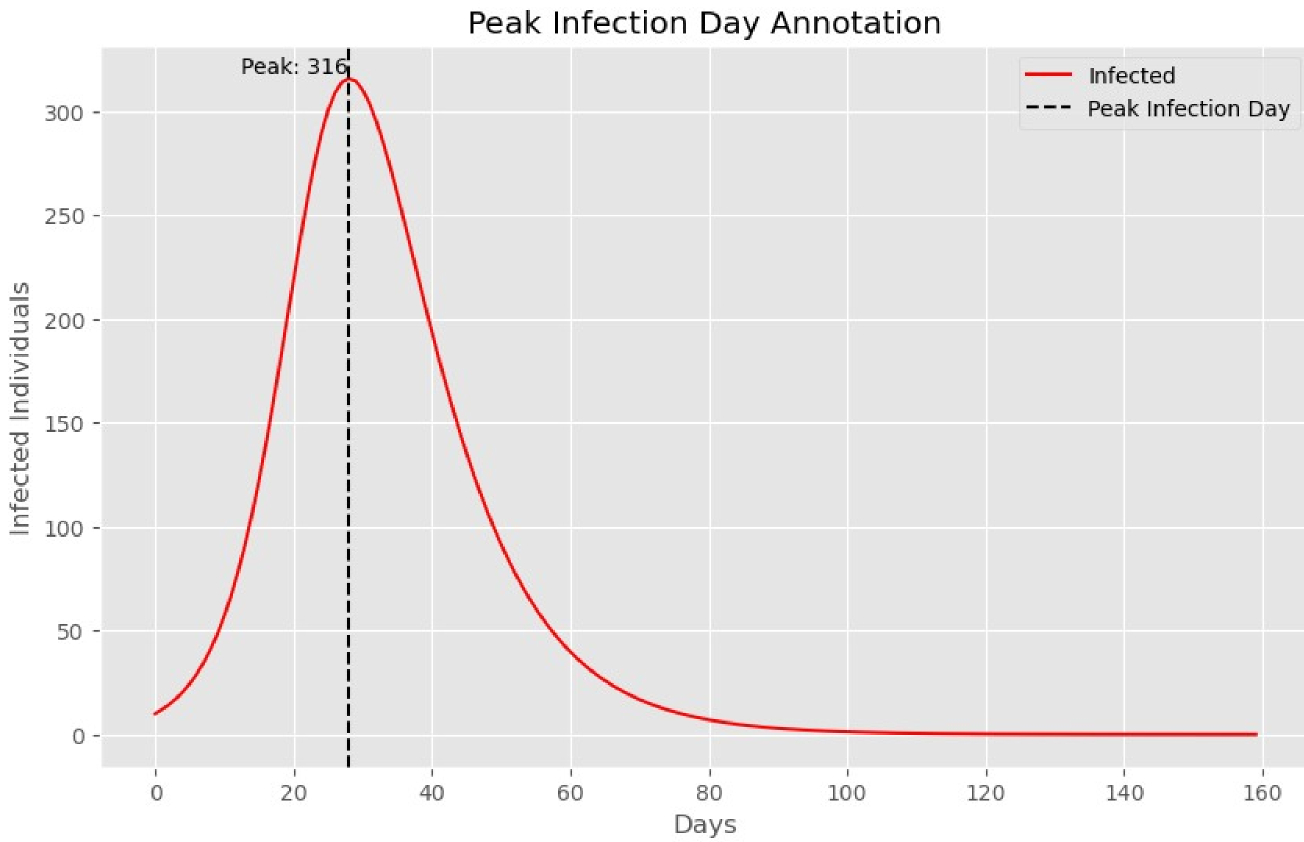

Figure 6: Peak Infection Day Annotation Plot

Next I ran the Monte Carlo simulation (Fig. 7). I made a scatter plot for this data to show the number of final infected individuals for each simulation number. Figures 8 and 9 show the code along with the resulting plot.

Monte Carlo Simulation

```python
# Monte Carlo simulation to estimate the impact of random initial infected
def monte_carlo_simulation(S0, I0_range, R0, beta, gamma, days, num_simulations):

    final_infected = np.zeros(num_simulations)

    for i in range(num_simulations):
        # Randomize the initial number of infected individuals within the specified range
        I0 = np.random.randint(I0_range[0], I0_range[1])

        # Run the SIR model
        SIR_data = SIR_model(S0, I0, R0, beta, gamma, days)

        # Store the final number of infected individuals
        final_infected[i] = SIR_data['infected'][-1]

    return final_infected

S0 = 990
I0_range = (5, 15)
R0 = 0
beta = 0.3
gamma = 0.1
days = 55
num_simulations = 100

# Run Monte Carlo Simulation
final_infected_results = monte_carlo_simulation(S0, I0_range, R0, beta, gamma, days, num_simulations)
#print("Final infected individuals form each simulation\n: ", final_infected_results)

# Analyze the results
mean_final_infected = np.mean(final_infected_results)
std_final_infected = np.std(final_infected_results)
```

Figure 7: Monte Carlo Simulation Code

```python
plt.figure(figsize=(10,6))
plt.plot(SIR_data['day'], SIR_data['infected'], label='Infected', color='red')
plt.axvline(peak_day, color='black', linestyle='--', label='Peak Infection Day')
plt.text(peak_day, peak_infections, f'Peak: {peak_infections:.0f}',
         verticalalignment='bottom', horizontalalignment='right', color='black')
plt.xlabel('Days')
plt.ylabel('Infected Individuals')
plt.title('Peak Infection Day Annotation')
plt.legend()
```
[5]

Figure 8: Monte Carlo Simulation Results Code

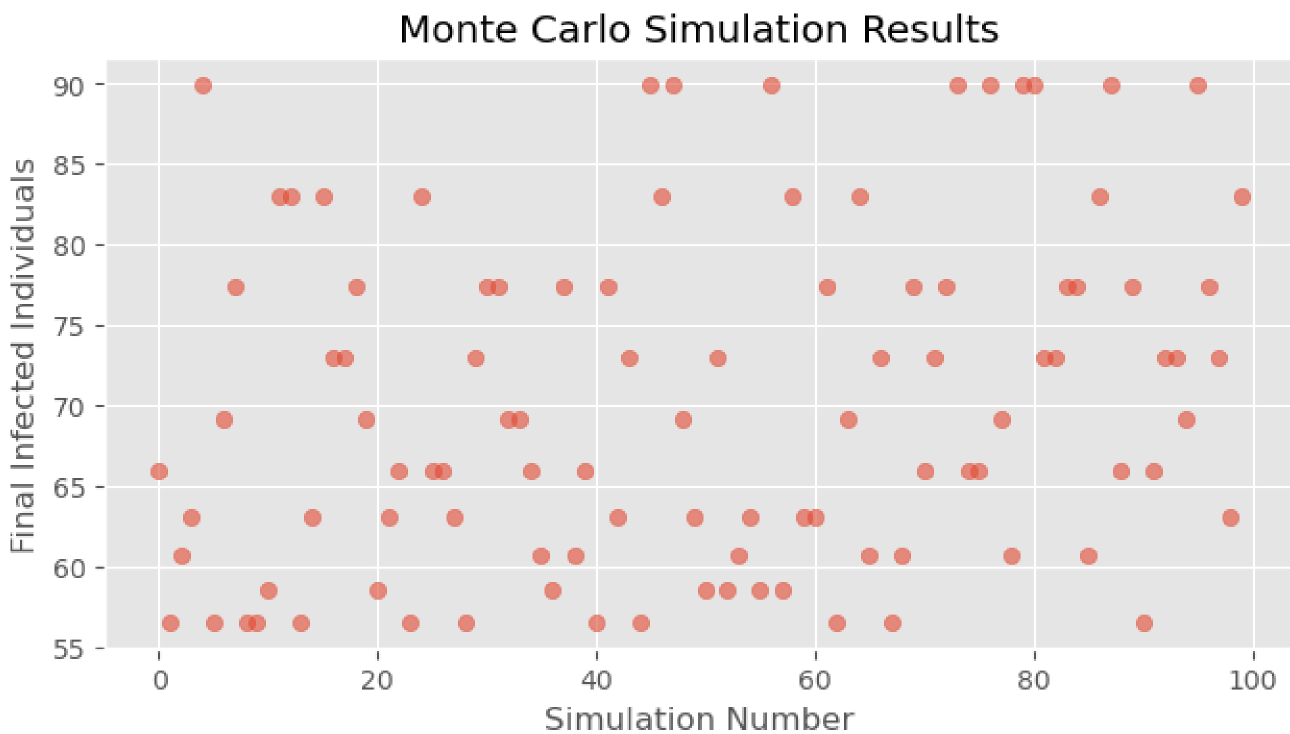

Figure 9: Monte Carlo Simulation Results Scatter Plot

Though the scatter plot was a good representation of the data, I felt that a histogram would be a better visual to display the frequency of the final infected individuals after each simulation. So used *`plt.hist()`* to construct a histogram with 20 bins. Figures 10 and 11 show the code and the output.

Histogram of Final Infected Individuals from Monte Carlo Simulation

```
plt.figure(figsize=(8,4))
plt.hist(final_infected_results, bins=20, color='purple', alpha=0.7)
plt.xlabel('Final Infected Individuals')
plt.ylabel('Frequency')
plt.title('Histogram of Final Infected Individuals from Monte Carlo Simulations');
```

[7]

Figure 10: Final Infected Individuals Histogram Code

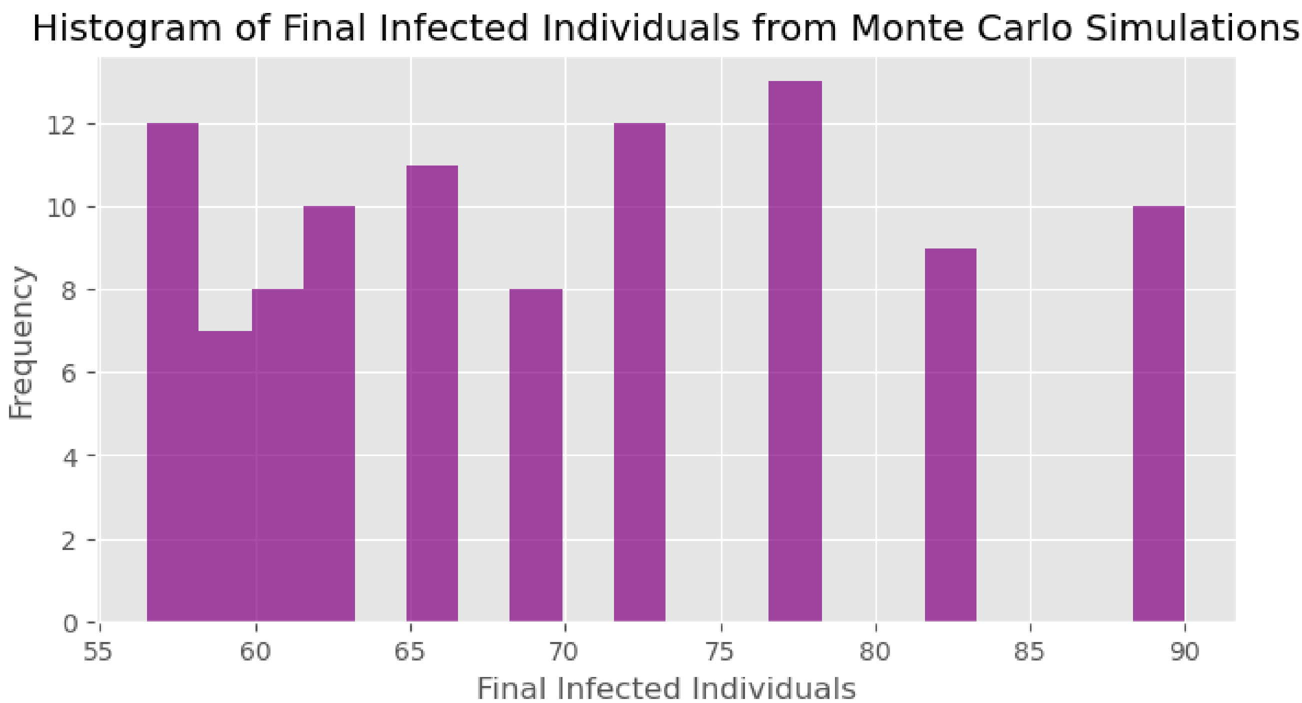

Figure 11: Final Infected Individuals Histogram Plot

To demonstrate working with a 3D contour plot, I use the original SIR model and plot the relationship between the infected and recovered over time. Figures 12 and 13 show the code and the resulting plot. From this plot we are able to visualize the dynamics of disease spread. With this plot we can see how the number of infected and recovered individuals changes over time simultaneously.

3D Plot of SIR Model

```
fig = plt.figure(figsize=(8, 4))
ax = fig.add_subplot(111, projection='3d')
ax.plot3D(SIR_data['day'], SIR_data['infected'], SIR_data['recovered'], color='blue')
ax.set_xlabel('Days')
ax.set_ylabel('Infected')
ax.set_zlabel('Recovered')
ax.set_title('3D Plot of Infection and Recovery over Time');
```

Figure 12: 3D Plot of SIR Model Code

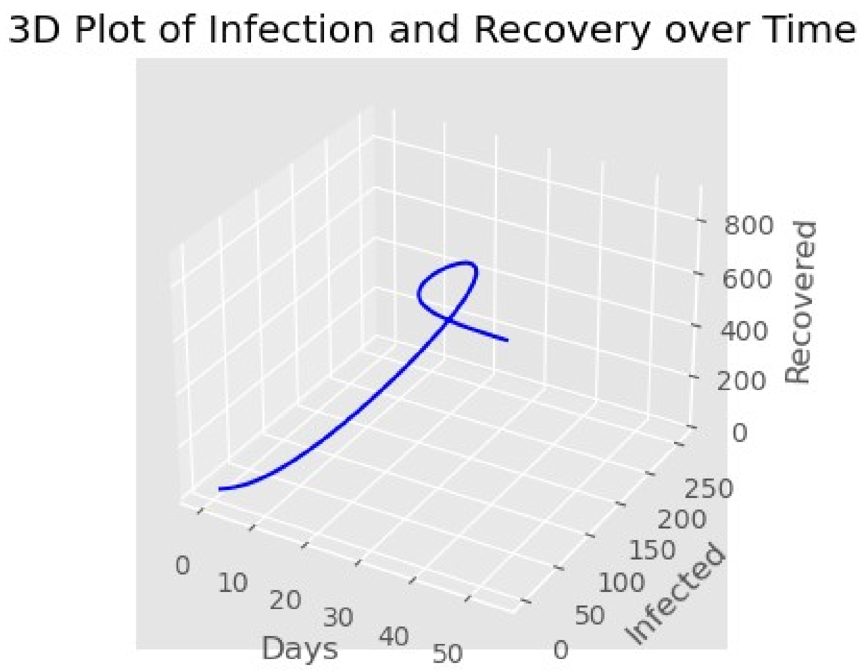

Figure 13: 3D Plot of SIR Model

I coded a contour plot to visualize the final number of infected individuals as a function of two parameters: the infection rate and the recovery rate. This helps to understand the sensitivity of the SIR model with changes in `*beta*` and `*gamma*` parameters. Areas on the graph with higher contours indicate that it will lead to more significant final infected counts. Figures 14 and 15 show the code and the output.

Density and Contour Plots for SIR Model Parameters

```python
beta_values = np.linspace(0.1, 0.5, 50)
gamma_values = np.linspace(0.05, 0.2, 50)
final_infected = np.zeros((len(beta_values), len(gamma_values)))

for i, beta in enumerate(beta_values):
    for j, gamma in enumerate(gamma_values):
        SIR_data = SIR_model(S0, I0, R0, beta, gamma, days)
        final_infected[i,j] = SIR_data['infected'][-1]

plt.figure(figsize=(8,4))
plt.contour(beta_values, gamma_values, final_infected.T, cmap='viridis')
plt.colorbar(label='Final Infected Individuals')
plt.xlabel('Beta (Infection Rate)')
plt.ylabel('Gamma (Recovery Rate)')
plt.title('Contour Plot of Final Infected Individuals');
```

✓ 0.3s

Figure 14: Contour Plot of Final Infected Individuals Code

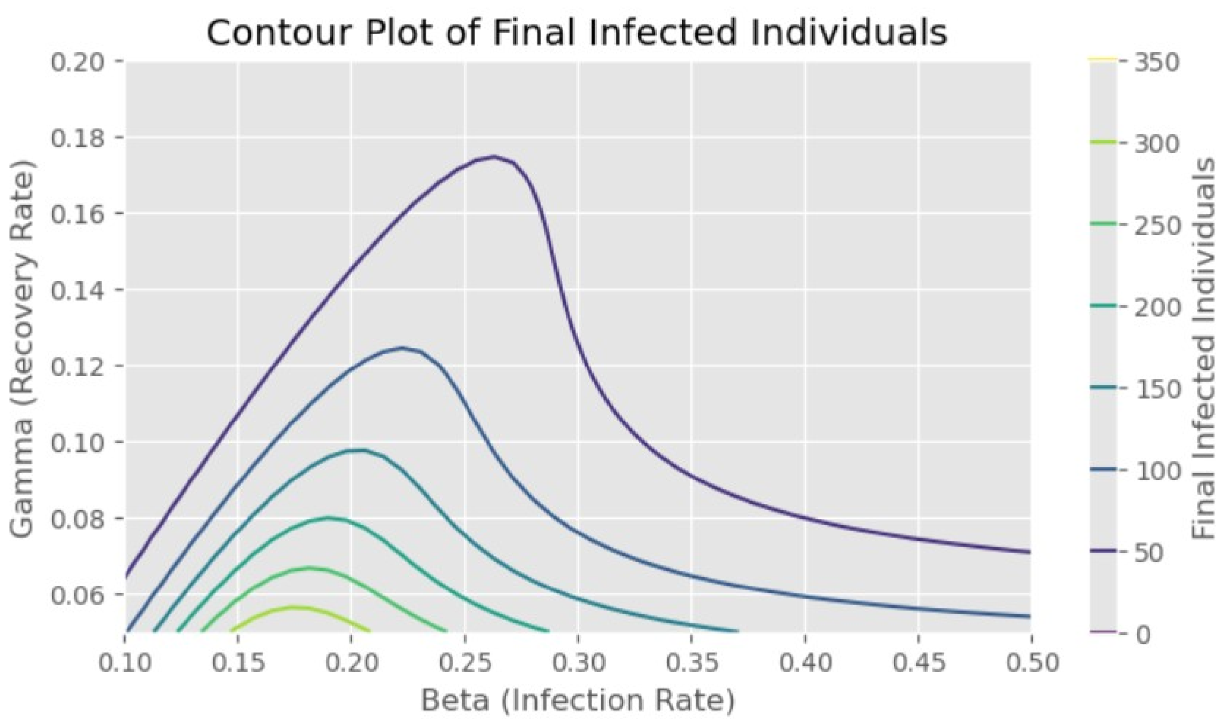

Figure 15: Contour Plot of Final Infected Individuals

The SIR model is an excellent tool to simulate and learn about disease spread. Though it is limited, it is still widely used in the real world to model infections and in classroom environments to teach differential equations. For a fun experiment, I wanted to create a more interactive graph that would allow the user to have more control to explore the different model parameters (susceptible, infected, recovered, beta, gamma) and observe the result. For this, I leveraged *`ipywidgets`* and imported the interact, Float slider, IntSlider, and BoundedIntText functions. I constructed a new function that would be used to call the SIR model and update its values as parameters change. Figures 16 and 17 show the code and the output.

Interactive Variables

```
from ipywidgets import interact, FloatSlider, IntSlider, BoundedIntText, jslink
# interactive plotting for SIR model
def plot_SIR(S0, I0, R0, beta, gamma):
    days=100
    SIR_data = SIR_model(S0, I0, R0, beta, gamma, days)

    peak_infections = np.max(SIR_data['infected'])
    peak_day = np.argmax(SIR_data['infected'])

    plt.figure(figsize=(8,4))
    plt.plot(SIR_data['day'], SIR_data['susceptible'], '--k', label='Susceptible')
    plt.plot(SIR_data['day'], SIR_data['infected'], '-.', label='Infected', color='orange')
    plt.plot(SIR_data['day'], SIR_data['recovered'], label='Recovered', color='c')
    #plt.axvline(peak_day, color='black', linestyle='--')
    plt.plot([peak_day, peak_day], [0, peak_infections], '.')
    plt.text(peak_day, peak_infections, f'Peak Infections: {peak_infections:.0f}',
            verticalalignment='bottom', horizontalalignment='left', color='black')
    plt.xlabel('Days')
    plt.ylabel('Population')
    plt.title('Interactive SIR Model Dynamics')
    plt.legend(fancybox=True, frameon=True, framealpha=1, facecolor='white', edgecolor='black', title='Legend:', borderpad=1);

susceptible_text = BoundedIntText(value=990, min=0, max=2000, step=1, description='Susceptible:')
susceptible_slider = IntSlider(value=990, min=0, max=2000, step=1, description='Susceptible:')
infected_text = BoundedIntText(value=10, min=0, max=2000, step=1, description='Infected:')
recovered_text = BoundedIntText(value=0, min=0, max=2000, step=1, description='Recovered:')
beta_slider = FloatSlider(value=0.3, min=0.0, max=1.0, step=0.01, description='Beta:')
gamma_slider = FloatSlider(value=0.1, min=0.0, max=1.0, step=0.01, description='Gamma:')
#day_slider = FloatSlider(value=160, min=0, max=500, step=1, description='Days')

interact(plot_SIR, S0=susceptible_text, I0=infected_text, R0=recovered_text, beta=beta_slider, gamma=gamma_slider);
```

Figure 16: Interactive SIR Model Dynamics Code

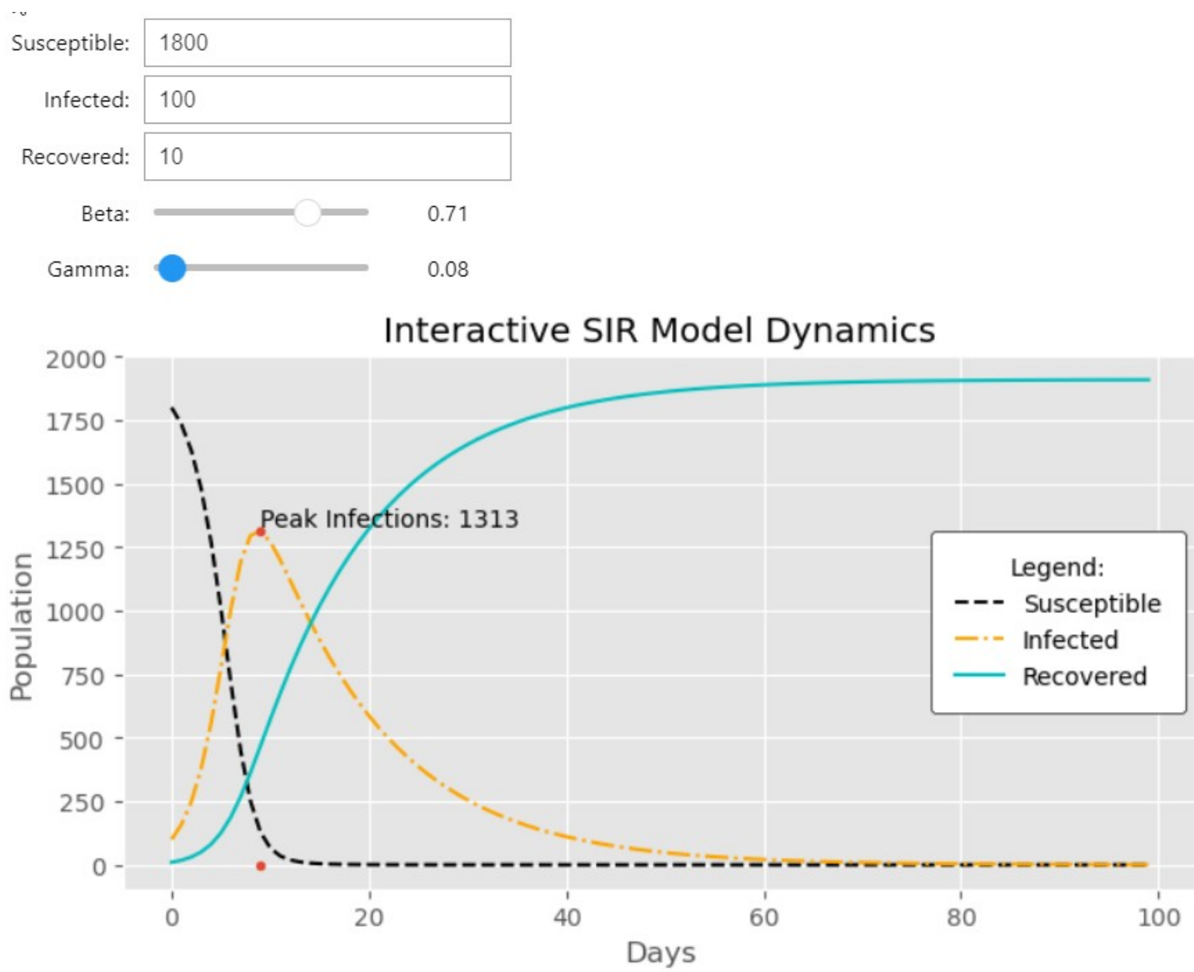

Figure 17: Interactive SIR Model Dynamics Plot and Layout

Another important concept in studying parameter relationships in differential equations is the trajectory plot also known as the phase plot plane. A phase plane plot visualizes the trajectory of the susceptible and infected populations in the SIR model. This code shown below in Figure 18 calculates this relationship over time. The purpose of this graph is to help understand the dynamic interactions between susceptible and infected individuals. There are better ways to produce this plot, however I wanted to attempt to recreate this plot using Matplotlib. Figure 19-20 shows the ouptut.

Trajectory Plot (Phase Plane Plot)

```python
S_vals = np.linspace(0, 1, 100)
I_vals = np.linspace(0, 1, 100)
S_grid, I_grid = np.meshgrid(S_vals, I_vals)
R_grid = 1 - S_grid - I_grid  # Since S + I + R = 1
contour = plt.contour(S_grid, I_grid, R_grid, levels=10, cmap='viridis')
plt.colorbar(contour, label='Recovered (R)')
plt.figure(figsize=(8,4))
plt.plot(SIR_data['susceptible'], SIR_data['infected'], label='Trajectory in phase plane', color='red')
plt.xlabel('Susceptible (S)')
plt.ylabel('Infectious (I)')
plt.title('Phase Plane of the SIR Model')
plt.legend();
```

Figure 18: Trajectory Plot Code

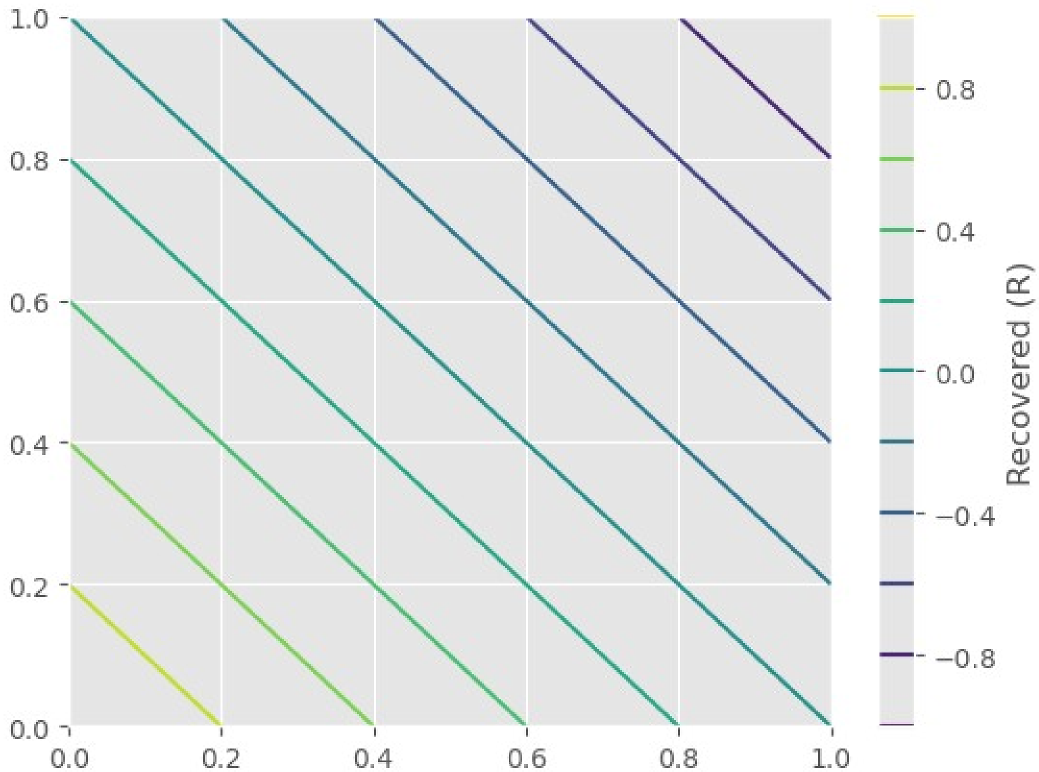

Figure 19: Trajectory Plot (a)

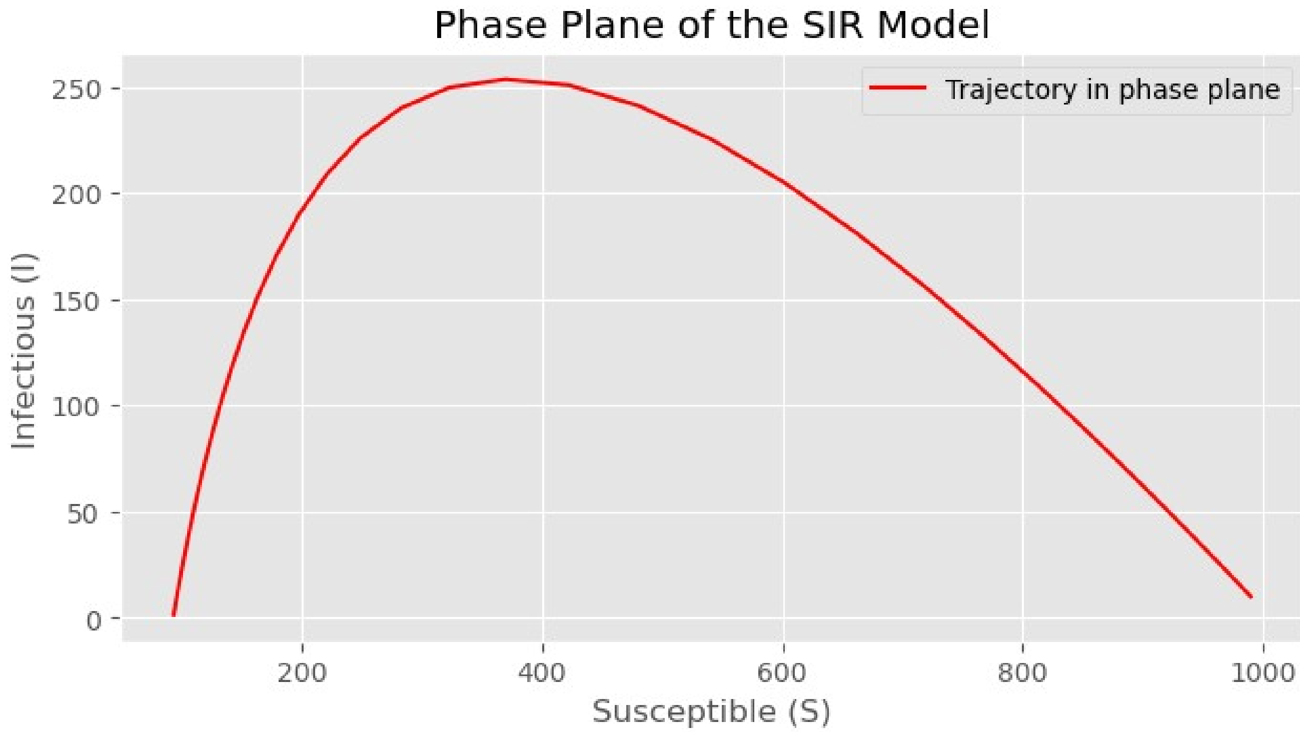

Figure 20: Trajectory Plot (b)

3. Adapting Movie Analysis Model for Visualization

To continue showing how Matplotlib can be used to visualize data and provide a more comprehensive understanding of data, I used my previous project that demonstrated using Pandas by analyzing a movie data and converting it to a dataframe for analysis. I first began by reading in the CSV and of top 1000 movies. Using my understanding of Pandas, I cleaned the data up by reordering columns, reducing the types of genres to make them a little more general, dropping any unused columns, removing rows containing null data. Further, I converted the data that should be a numerical datatype, sorted the data by ratings, and finally adding a rank column. The code and the result are shown below in Figures 21-23.

2. Adapting Movie Data Analysis for Visualization

Load Dataset and clean it up

```python
# Load the dataset
movies_df = pd.read_csv("CSV Files//imdb_top_1000.csv")

# Add a 'Rank' column
movies_df['Rank'] = range(1, len(movies_df) + 1)

# Reorder the DataFrame Columns
new_columns = list(movies_df.columns)
new_columns.remove('Rank')
new_columns.insert(0, 'Rank')
movies_df = movies_df[new_columns]

# reducing types of genres
movies_df['Genre'] = movies_df['Genre'].str.replace(r'Crime.*', 'Crime', regex=True)
movies_df['Genre'] = movies_df['Genre'].str.replace(r'Action.*', 'Action', regex=True)
movies_df['Genre'] = movies_df['Genre'].str.replace(r'Biography.*', 'Biography', regex=True)
movies_df['Genre'] = movies_df['Genre'].str.replace(r'Animation.*', 'Animation', regex=True)
movies_df['Genre'] = movies_df['Genre'].str.replace(r'Mystery.*', 'Mystery', regex=True)
movies_df['Genre'] = movies_df['Genre'].str.replace(r'Drama.*', 'Drama', regex=True)
movies_df['Genre'] = movies_df['Genre'].str.replace(r'Adventure.*', 'Adventure', regex=True)
movies_df['Genre'] = movies_df['Genre'].str.replace(r'Romance$', 'Romance', regex=True)
movies_df['Genre'] = movies_df['Genre'].str.replace(r'Comedy.*', 'Comedy', regex=True)

# drop unused columns
movies_df.drop(['Certificate', 'Poster_Link'], axis=1, inplace=True)

# remove rows with null data
movies_df.dropna(inplace=True)

# Convert 'Gross' to numbers
movies_df['Gross'] = movies_df['Gross'].replace(r'[\$,]', '', regex=True).str.strip()
movies_df['Gross'] = pd.to_numeric(movies_df['Gross'])

# Convert 'Runtime' to numbers
movies_df['Runtime'] = movies_df['Runtime'].str.replace(' min', '')
movies_df['Runtime'] = pd.to_numeric(movies_df['Runtime'])
```

Figure 21: Movie Analysis Code (a)

```python
# Convert 'Released_Year' to datetime
valid_year_mask = movies_df['Released_Year'].str.match(r'^\d{4}$')
movies_df = movies_df.loc[valid_year_mask].copy()
movies_df['Released_Year'] = pd.to_datetime(movies_df['Released_Year'] + '-01-01')
movies_df['Released_Year'] = movies_df['Released_Year'].dt.year

# Clean up 'Genre' strings
movies_df['Genre'] = movies_df['Genre'].str.strip().str.lower()

# Sorting the data by ratings
movies_df = movies_df.sort_values(by='IMDB_Rating', ascending=False)

# Resetting the index
movies_df.reset_index(drop=True, inplace=True)

# Updating the 'Rank' column
movies_df['Rank'] = movies_df.index + 1

# Setting the 'Rank' column as the index
movies_df.set_index('Rank', inplace=True)

# Print first few rows of dataset
movies_df.head()
```

Figure 22: Movie Analysis Code (b)

Rank	Series_Title	Released_Year	Runtime	Genre	IMDB_Rating	Overview	Meta_score	Director	Star1	Star2	Star3	Star4	No_of_Votes	Gross
1	The Shawshank Redemption	1994	142	drama	9.3	Two imprisoned men bond over a number of years...	80.0	Frank Darabont	Tim Robbins	Morgan Freeman	Bob Gunton	William Sadler	2343110	28341469
2	The Godfather	1972	175	crime	9.2	An organized crime dynasty's aging patriarch t...	100.0	Francis Ford Coppola	Marlon Brando	Al Pacino	James Caan	Diane Keaton	1620367	134966411
3	The Dark Knight	2008	152	action	9.0	When the menace known as the Joker wreaks havo...	84.0	Christopher Nolan	Christian Bale	Heath Ledger	Aaron Eckhart	Michael Caine	2303232	534858444
4	The Godfather: Part II	1974	202	crime	9.0	The early life and career of Vito Corleone in ...	90.0	Francis Ford Coppola	Al Pacino	Robert De Niro	Robert Duvall	Diane Keaton	1129952	57300000
5	12 Angry Men	1957	96	crime	9.0	A jury holdout attempts to prevent a miscarria...	96.0	Sidney Lumet	Henry Fonda	Lee J. Cobb	Martin Balsam	John Fiedler	689845	4360000

Figure 23: Movie Analysis Code (c)

To begin to better understand the data, I plotted the IMDB_Rating in a histogram to better understand the distribution of ratings. Here we see that the majority of top movies tend to have a rating of 7.6 to 8.20. It is rare to have a movie with a rating above a 9. Figures 24 and 25 shows the code and the results.

Distribution of IMDB Ratings

```
plt.figure(figsize=(8,4))
sns.histplot(movies_df['IMDB_Rating'], kde=True, bins=20)
plt.title('Distribution of IMDB Ratings')
plt.xlabel('IMDB Rating')
plt.ylabel('Frequency');
```

Figure 24: Distribution of IMDB Ratings Code

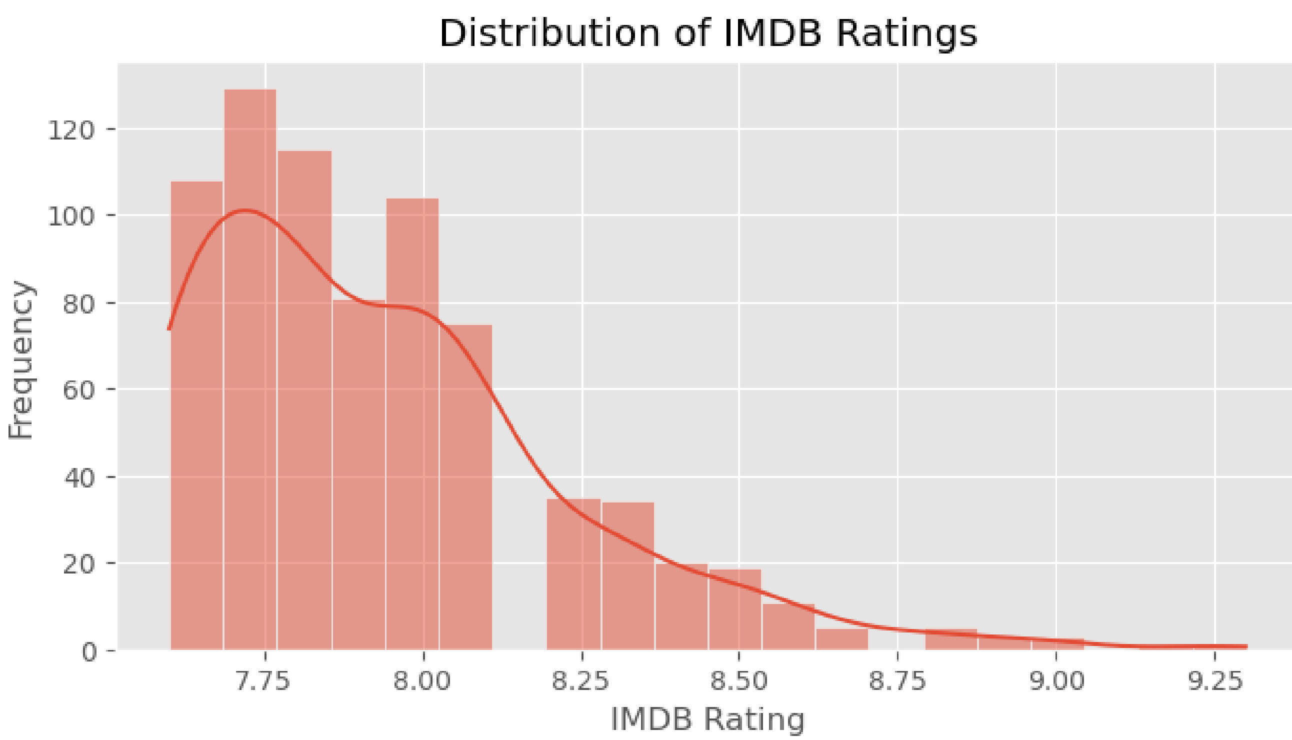

Figure 25: Distribution of IMDB Ratings Histogram Plot

Next I used a similar histogram plot to show the distribution of gross earnings, along with showing the distribution of runtimes. Figures 26-29 shows the code and output. The majority of the top movies tend to stick around a runtime of 100 to 125 minutes.

Distribution of Gross Earnings

```python
plt.figure(figsize=(8,4))
sns.histplot(movies_df['Gross'], kde=True, bins=20)
plt.title('Distribution of Gross Earnings')
plt.xlabel('Gross Earnings (in dollars)')
plt.ylabel('Frequency');
```

✓ 0.0s

Figure 26: Distribution of Gross Earnings Code

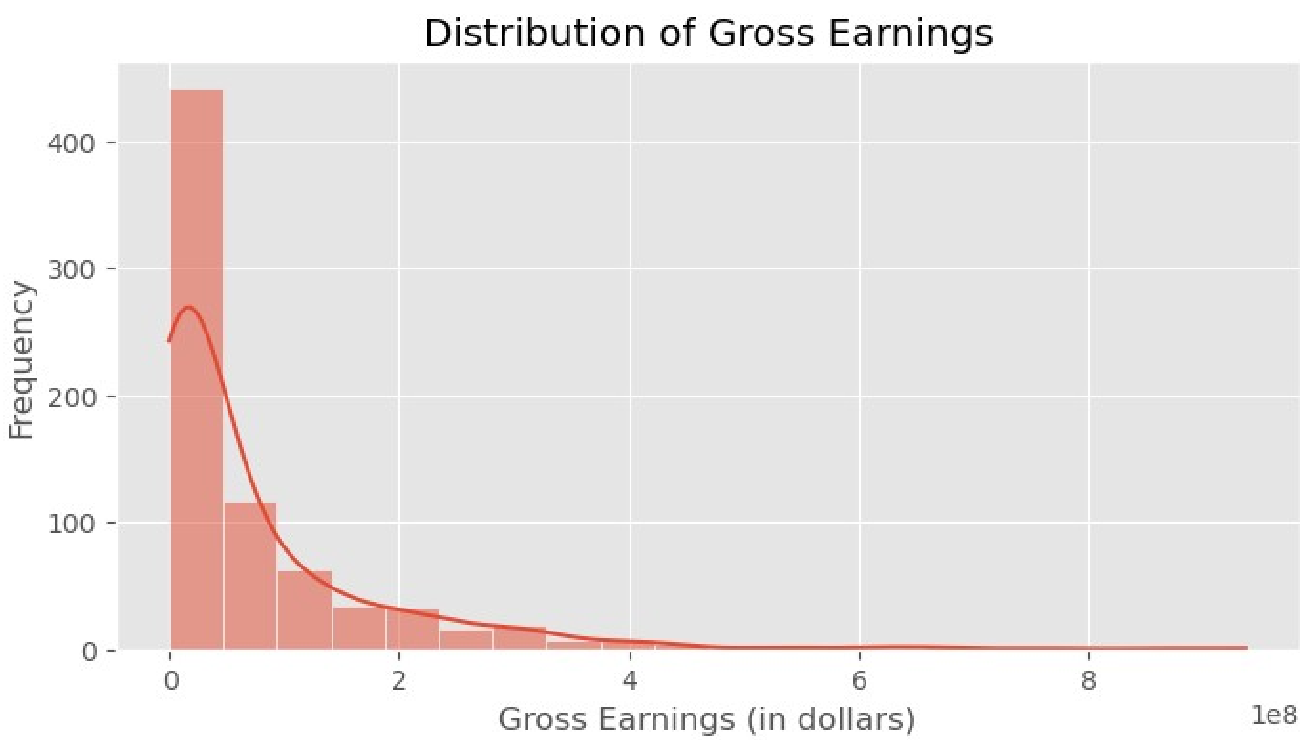

Figure 27: Distribution of Gross Earnings

Distribution of Movie Runtimes

```python
plt.figure(figsize=(8,4))
sns.histplot(movies_df['Runtime'], kde=True, bins=20)
plt.title('Distribution of Movie Runtimes')
plt.xlabel('Runtime (minutes)')
plt.ylabel('Frequency');
```

Figure 28: Distribution of Movie Runtimes Code

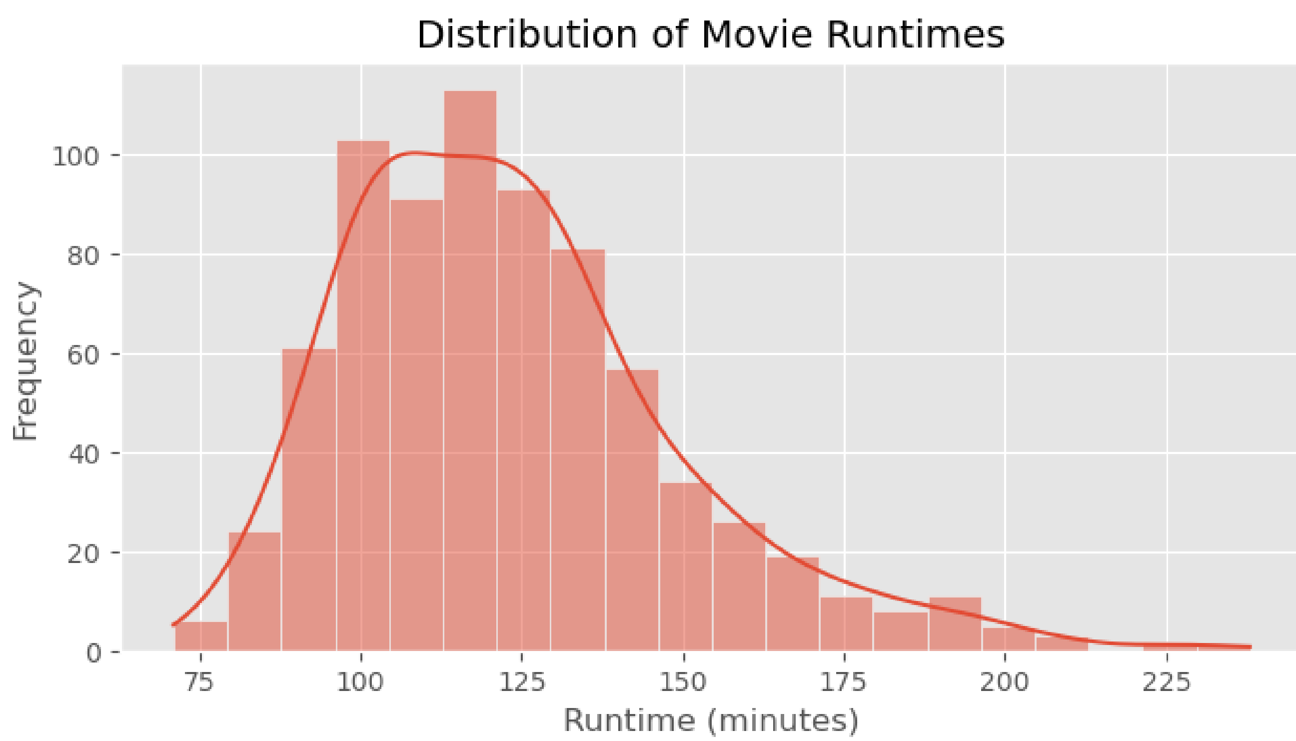

Figure 29: Distribution of Movie Runtimes

The next bar plot displays the number of movies that are considered the top movies for each year. The code processes the dataset to extract release years, handling missing and incorrect data, and the plots the counts using *`seaborn`*. This graph provides insight into trends and patterns in movie production over time. Figures 30 and 31 show the code and output.

Number of Movies Released per Year

```
plt.figure(figsize=(10,6))
sns.countplot(data=movies_df, x='Released_Year')
plt.title('Number of Movies Released per Year')
plt.xlabel('Year')
plt.ylabel('Number of Movies')
plt.xticks(rotation=45, fontsize=10)

plt.gca().xaxis.set_major_locator(plt.MaxNLocator(nbins=20));
```

Figure 30: Number of Movies Released per Year Code

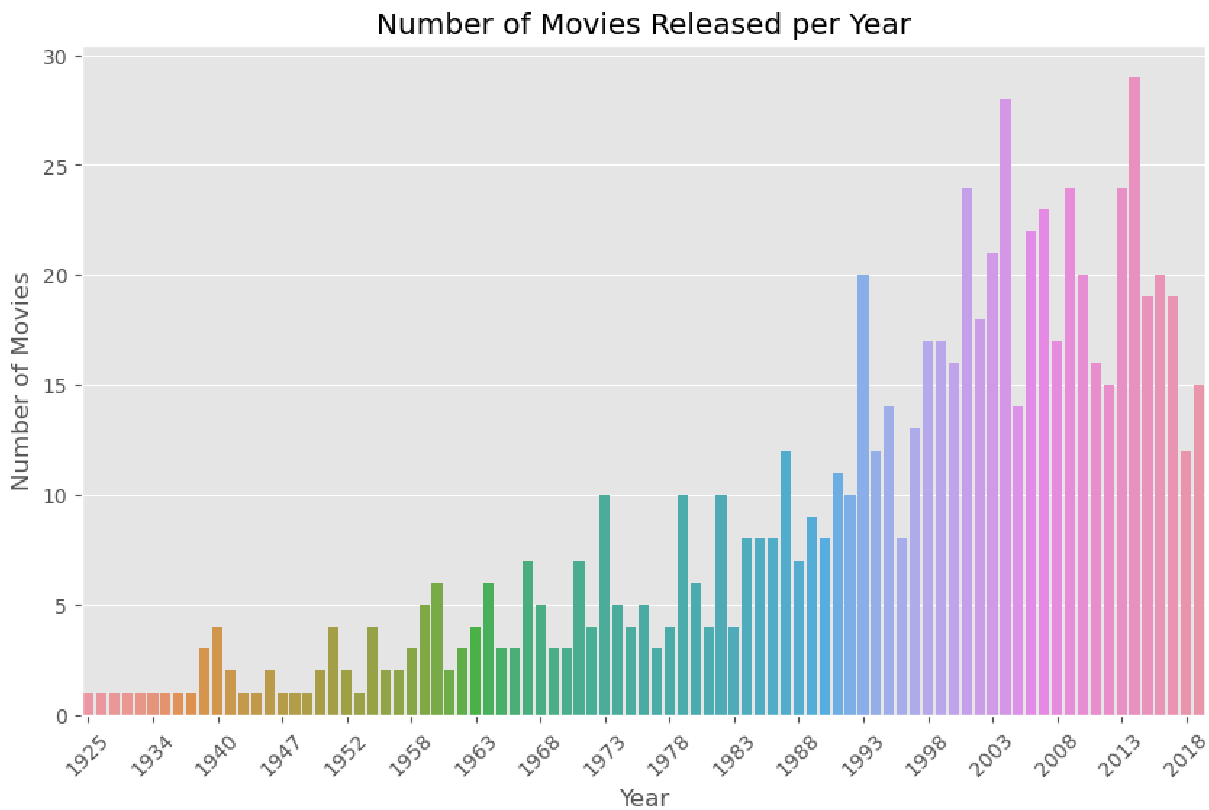

Figure 31: Number of Movies Released per Year Plot

The next graph is another bar plot that shows the average IMDB ratings for the different genres. The code simplifies genre names using regex, groups the data by genre, and calculates the average rating for each. This visualization highlights differences in audience reception across various genres. Figures 32 and 33 show the code and output.

Average IMDB Rating by Genre

```python
plt.figure(figsize=(10,6))
top_genres = movies_df['Genre'].value_counts().head(20).index
top_genre_ratings = movies_df[movies_df['Genre'].isin(top_genres)].groupby('Genre')['IMDB_Rating'].mean().sort_values()
sns.barplot(x=top_genre_ratings, y=top_genre_ratings.index)
plt.title('Average IMDB Rating by Top 20 Genres')
plt.xlabel('Average IMDB Rating')
plt.ylabel('Genre');
```

Figure 32: Average IMDB Rating by Genre Code

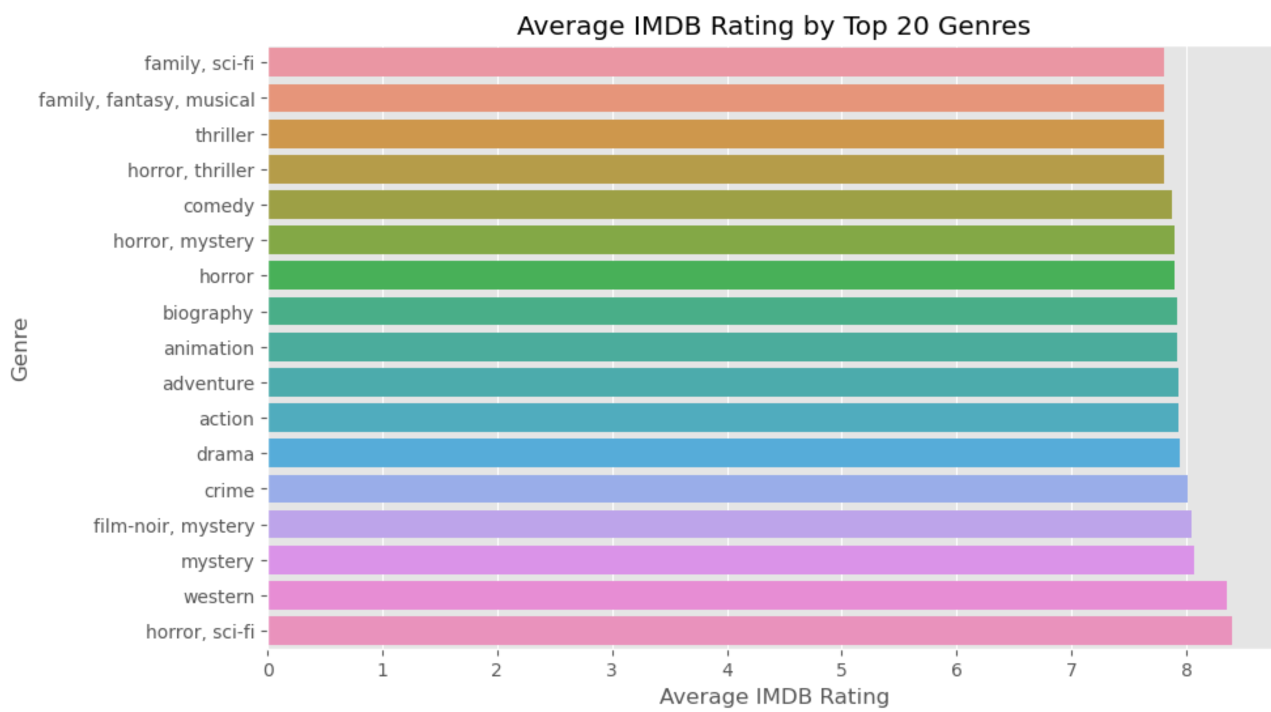

Figure 33: Average IMDB Rating by Genre Plot

The next graph is a scatter plot to examine the relationship between gross earnings and IMDB ratings. The code plots these two variables, showing how movie earnings correlate with their ratings to identify potential patterns in the data. Figure 34 and 35 show the output and code.

Scatter Plot of Gross Earnings vs IMDB Rating

```python
plt.figure(figsize=(10, 6))
sns.scatterplot(data=movies_df, x='IMDB_Rating', y='Gross')
plt.title('Gross Earnings vs. IMDB Rating')
plt.xlabel('IMDB Rating')
plt.ylabel('Gross Earnings (in dollars)')
plt.show()
```

Figure 34: Scatter Plot of Gross Earnings vs IMDB Rating Code

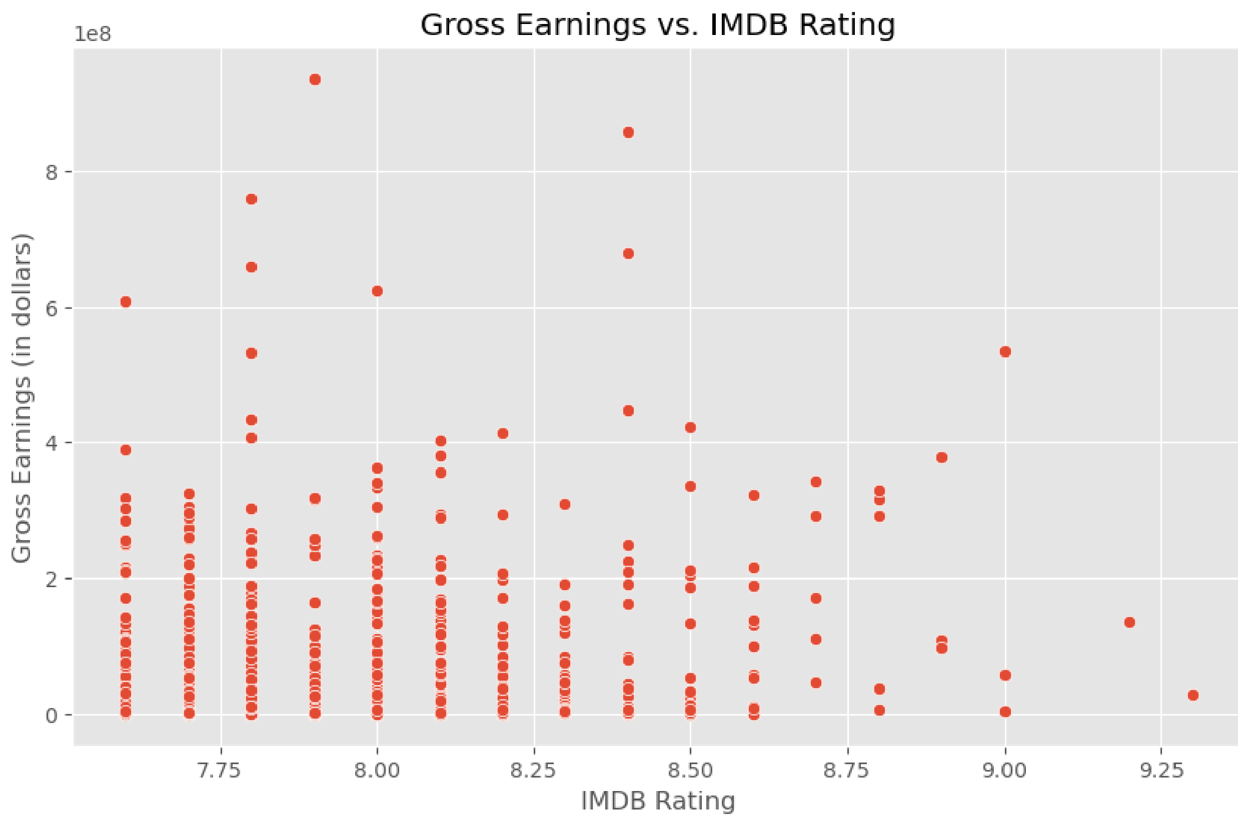

Figure 35: Scatter Plot of Gross Earnings vs IMDB Rating

Next I used a pie chart to represent the top 10 directors by the number of movies directed. The code counts the occurrences of each director in the dataset and visualizes the top 10. This graph gives a view of which directors have the most significant presence in the top 1000 movies (Fig. 36-37).

Visualizing top directors

```python
director_counts = movies_df['Director'].value_counts()

top_directors = director_counts.head(10)

plt.figure(figsize=(8,8))
plt.pie(top_directors, labels=top_directors.index, autopct='%1.1f%%', startangle=140)
plt.title('Top 10 Directors by Number of Movies Directed')
plt.axis('equal');
```

Figure 36: Visualizing Top Directors Code

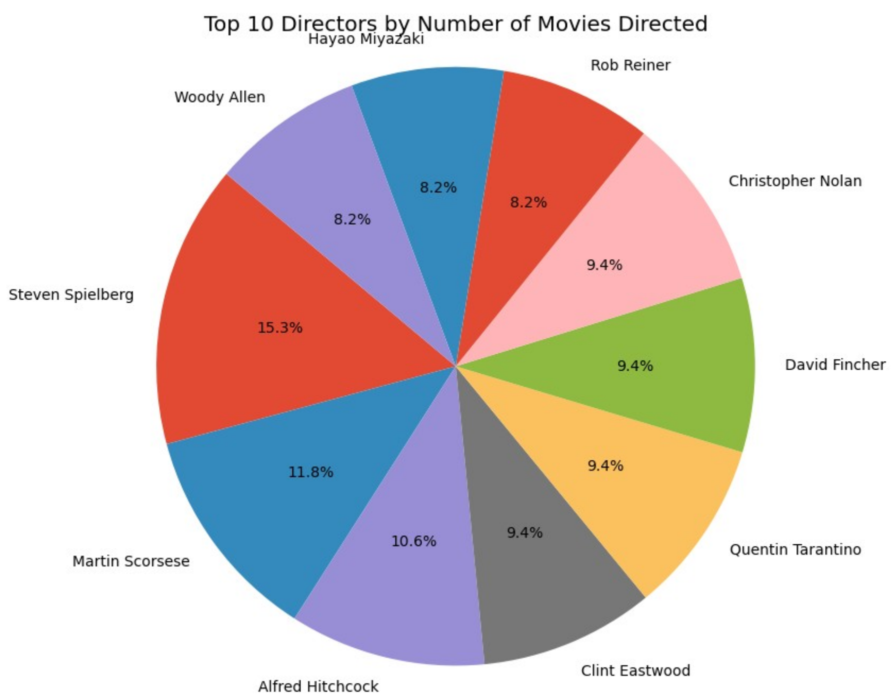

Figure 37: Visualizing Top Directors Pie Chart

The next graph shown below was a box plot constructed using `*seaborn*` to visualize the distribution of IMDB ratings across different movie genres. This graph provides insight into how ratings vary within each genre and to compare the central tendency and spread across genres. Figure 38 and 39 shows the output and code.

Box Plot of IMDB Ratings by Genre

```
plt.figure(figsize=(21,7))
sns.boxplot(data=movies_df, x='IMDB_Rating', y='Genre')
plt.title('IMDB Ratings by Genre')
plt.ylabel('Genre');
```

Figure 38: Box Plot of IMDB Ratings by Genre Code

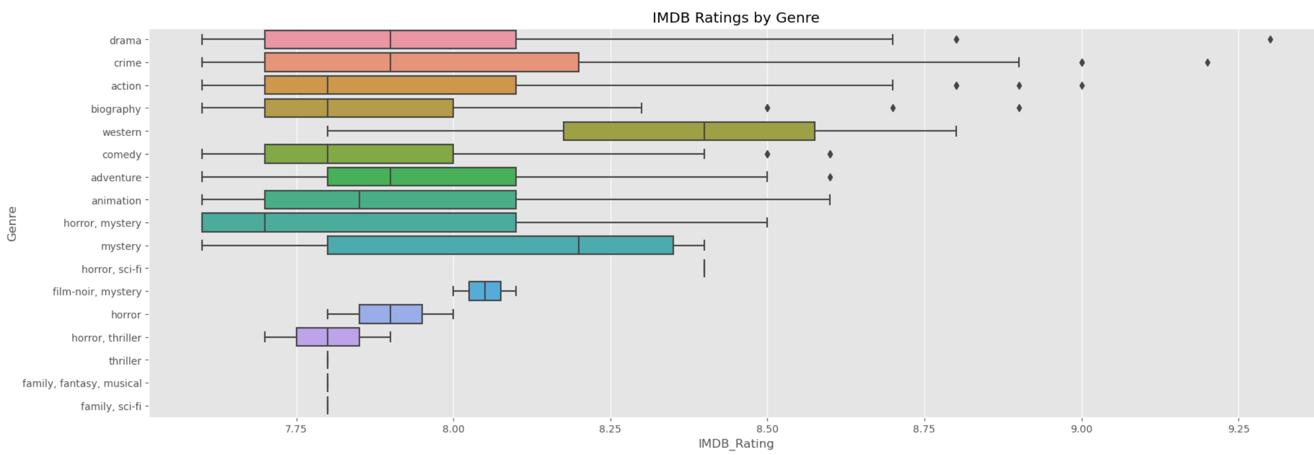

Figure 39: Box Plot of IMDB Ratings by Genre

Using `*seaborn*` again, a pairplot was used to visualize and asses the relationship between the rating, gross earnings, and runtime. Figure 40 and 41 shows the output and code.

Pairplot of Ratings, Gross, and Runtime

```python
sns.pairplot(movies_df[['IMDB_Rating', 'Gross', 'Runtime']]);
```

✓ 0.7s

Figure 40: Pairplot of Ratings, Gross Earnings, and Runtime Code

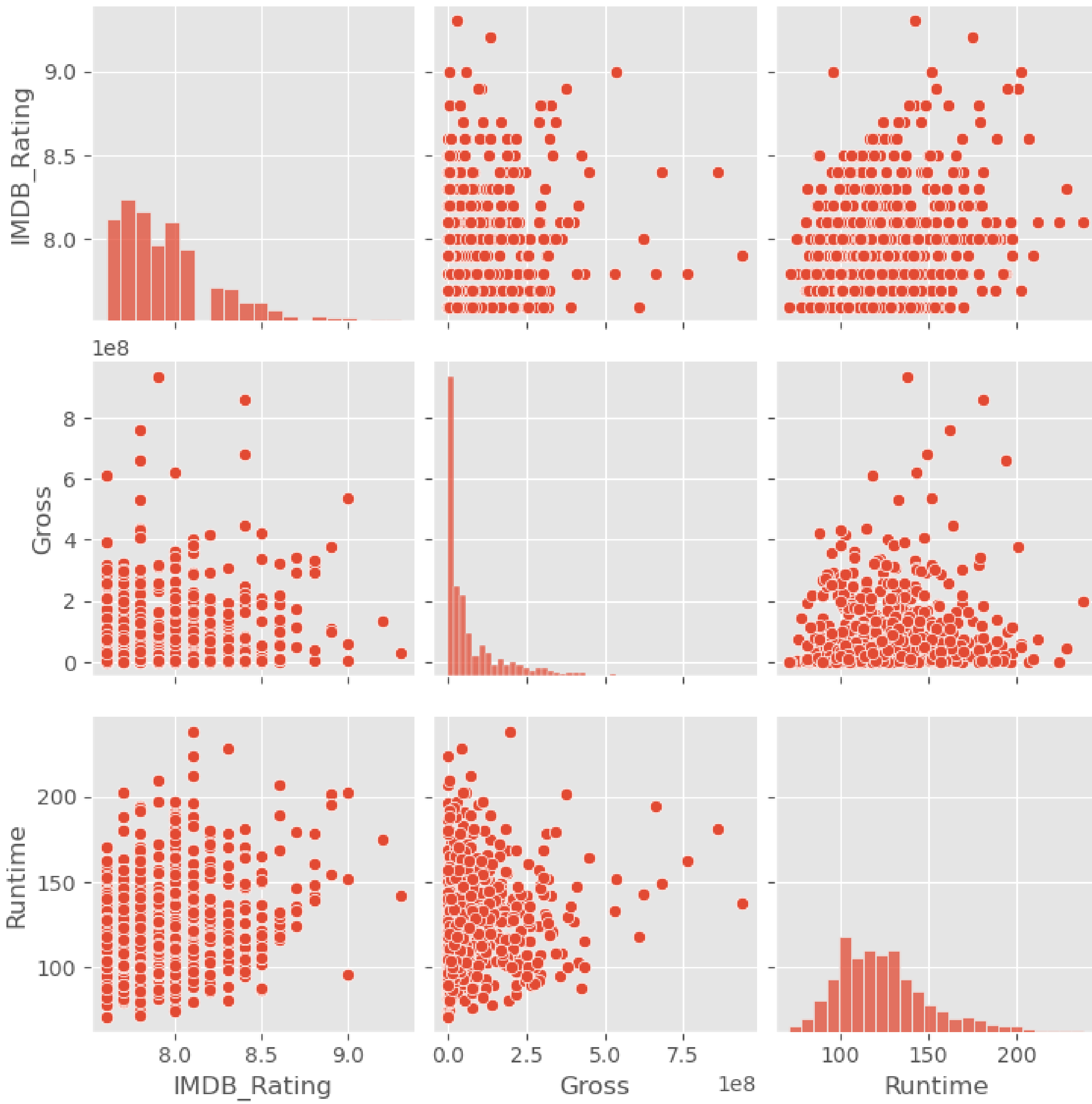

Figure 41: Pairplot of Ratings, Gross Earnings, and Runtime

The next graph is an area plot that shows the popularity of different genres over time. The code groups the data by release year and genre, then plots the number of movies in each genre per year to reveal any trends in genre popularity over the years. Figure 42 and 43 show the code and output.

Genre Popularity Over Time with Streamgraph

```python
genre_year_count = movies_df.groupby(['Released_Year', 'Genre']).size().unstack().fillna(0)

genre_year_count.plot(kind='area', stacked=True, figsize=(14,8), cmap='tab20')
plt.title('Popularity of Genres Over Time')
plt.xlabel('Year')
plt.ylabel('Number of Movies')
plt.legend(loc='upper left', bbox_to_anchor=(1,1));
```

Figure 42: Genre Popularity over Time with Area Plot Code

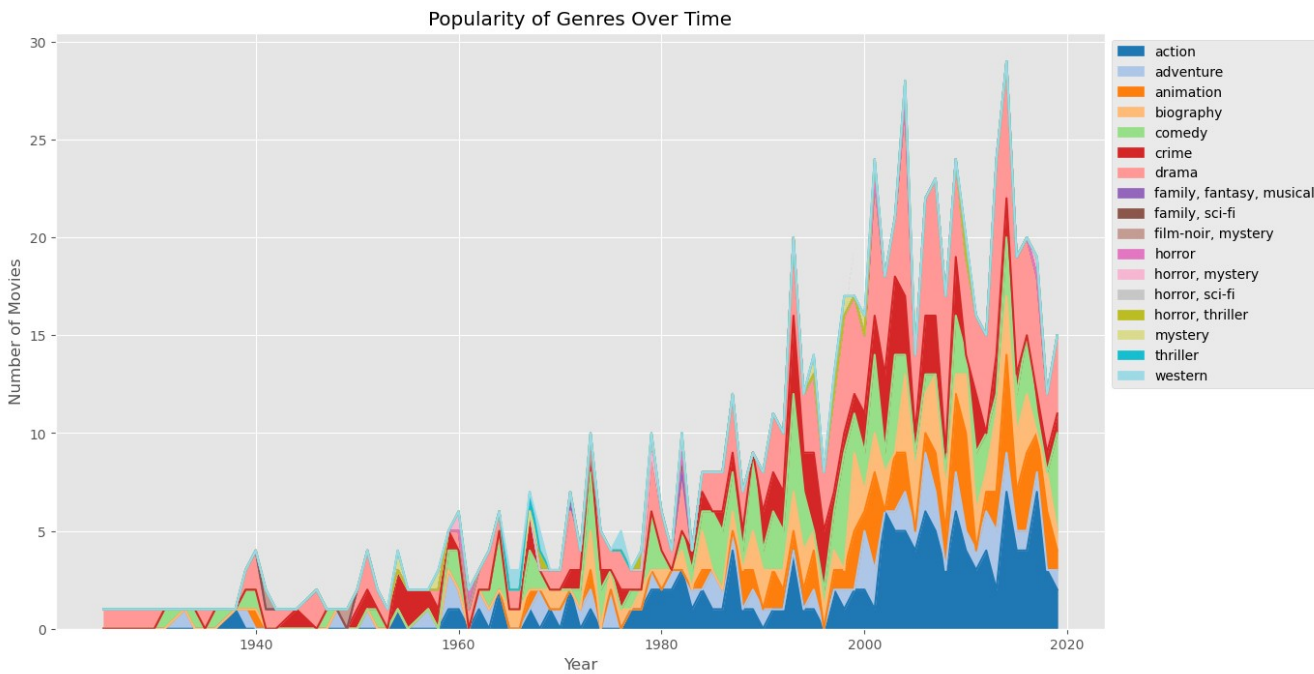

Figure 43: Genre Popularity over Time with Area Plot

The last graph is a bar plot with error bars to visualize the average IMDB rating for each genre. The error bars represent the standard error of the mean, which help to provide a visual indication of the variability around the mean rating for each genre. Figure 44 and 45 shows the code and output.

```python
# Plotting the bar plot with error bars
genre_stats = movies_df.groupby('Genre')['IMDB_Rating'].agg(['mean', 'sem']).reset_index()
plt.figure(figsize=(12, 6))
plt.bar(genre_stats['Genre'], genre_stats['mean'], yerr=genre_stats['sem'], capsize=5, color='skyblue', edgecolor='black')
plt.xlabel('Genre')
plt.ylabel('Average IMDB Rating')
plt.title('Average IMDB Rating by Genre with Error Bars')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```

Figure 44: Average IMDB Rating by Genre with Error Bars Code

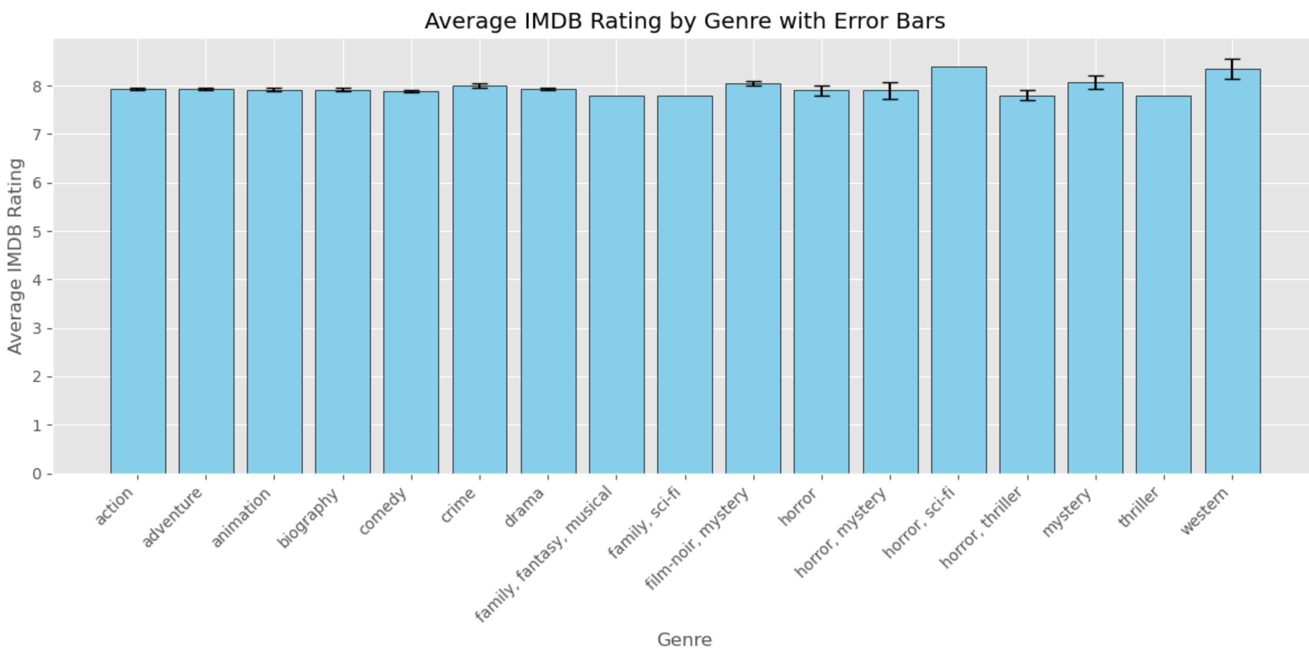

Figure 45: Average IMDB Rating by Genre with Error Bars

4. Other Models

To demonstrate the ability of using the Basemap toolkit, I decided to visualize COVID-19 cases across the US. I downloaded a dataset from John Hopkins University GitHuub. I cleaned the data up by dropping rows that did not contain any information about the latitude and longitude, and filling specific columns that contain missing data with zero. Figure 46 and 47 shows the code and the output. I used the Basemap toolkit to plot the geographic data, which helps provide a spatial understanding of COVID-19's impact. Figure 48 and 49 shows the code and output. Additionally, I constructed subplots using `*sns.barplot*` to visualize the average number of confirmed COVID-19 cases and deaths in the US. Figure 50 and 51 shows the code and the results.

Covid-19 Data

Loading the data

```python
covid_df = pd.read_csv("CSV Files//JHU-03-08-2023.csv")
covid_df = covid_df.dropna(subset=['Lat', 'Long_'])
# Fill missing values in 'Deaths', 'Recovered', and 'Confirmed' columns with zeros
covid_df['Deaths'] = covid_df['Deaths'].fillna(0)
covid_df['Recovered'] = covid_df['Recovered'].fillna(0)
covid_df['Confirmed'] = covid_df['Confirmed'].fillna(0)
covid_df.head()


```

25] ✓ 0.0s

Figure 46: Importing and Cleaning COVID-19 Dataset Code

	Province_State	Country_Region	Last_Update	Lat	Long_	Confirmed	Deaths	Recovered	Active	FIPS	Incident_Rate
0	Alabama	US	2023-03-09 04:32:54	32.3182	-86.9023	1644533	21032	0.0	NaN	1.0	33540.096896
1	Alaska	US	2023-03-09 04:32:54	61.3707	-152.4044	307655	1486	0.0	NaN	2.0	42055.512648
2	American Samoa	US	2023-03-09 04:32:54	-14.2710	-170.1320	8320	34	0.0	NaN	60.0	14953.002282
3	Arizona	US	2023-03-09 04:32:54	33.7298	-111.4312	2443514	33102	0.0	NaN	4.0	33570.669117
4	Arkansas	US	2023-03-09 04:32:54	34.9697	-92.3731	1006622	13015	0.0	NaN	5.0	33356.109277

Figure 47: Importing and Cleaning COVID-19 Dataset Output

Making the map

```
plt.figure(figsize=(15,10))    (parameter) llcrnrlat: Any | None
#m = Basemap(projection='lcc',                                 :6, lat_0=37.5, lon_0=-97)
m = Basemap(projection='merc', llcrnrlat=20, urcrnrlat=50, llcrnrlon=-130, urcrnrlon=-60, resolution='i')
#m.drawmapboundary(fill_color='aqua')
#m.fillcontinents(color='w', lake_color='aqua')
m.drawcoastlines()
m.drawstates()
m.drawcountries();

x, y = m(covid_df['Long_'].values, covid_df['Lat'].values)
m.scatter(x, y, s=covid_df['Confirmed']/500, c=covid_df['Confirmed'], cmap=plt.cm.get_cmap('Reds', 4), alpha=0.6, edgecolors='w', linewidth=0.5, zorder=2)
plt.colorbar(label='Number of Confirmed Cases', extend='both')
plt.title('Number of Covid-19 Cases by State in the US');
```

Figure 48: Visualizing COVID-19 Cases in the US with Basemap Code

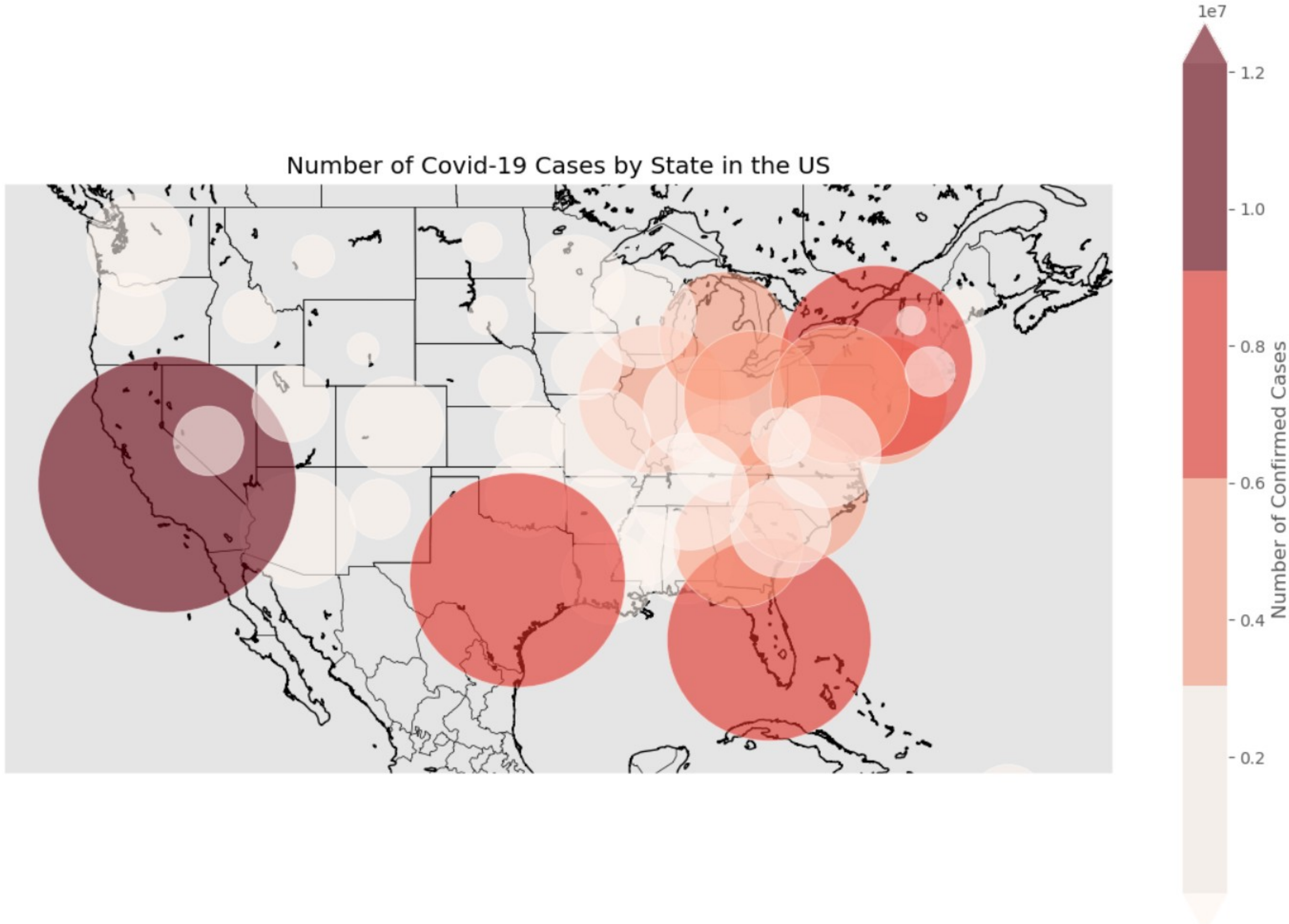

Figure 49: Visualizing COVID-19 Cases in the US with Basemap

```python
# Aggregating data for visualization
top_states = covid_df.groupby('Province_State').agg({
    'Confirmed': 'mean',
    'Deaths': 'mean',
    'Recovered': 'mean'
}).reset_index()

# Selecting top states by confirmed cases
top_states = top_states.sort_values(by='Confirmed', ascending=False)

# Create subplots
fig, ax = plt.subplots(2, 1, figsize=(14, 18))

# Plotting the bar plot for confirmed cases
sns.barplot(x='Province_State', y='Confirmed', data=top_states, ax=ax[0], palette='Blues_d')
ax[0].set_xlabel('State')
ax[0].set_ylabel('Number of Confirmed Cases')
ax[0].set_title('Average Number of Confirmed COVID-19 Cases by State')
ax[0].tick_params(axis='x', rotation=45)
ax[0].set_xticks(range(len(top_states['Province_State'])))  # Set x-ticks manually
ax[0].set_xticklabels(top_states['Province_State'], rotation=45, ha='right')  # Set tick labels with rotation

# Plotting the bar plot for deaths
sns.barplot(x='Province_State', y='Deaths', data=top_states, ax=ax[1], palette='Reds_d')
ax[1].set_xlabel('State')
ax[1].set_ylabel('Number of Deaths')
ax[1].set_title('Average Number of COVID-19 Deaths by State')
ax[1].tick_params(axis='x', rotation=45)
ax[1].set_xticks(range(len(top_states['Province_State'])))
ax[1].set_xticklabels(top_states['Province_State'], rotation=45, ha='right')

plt.tight_layout();
```

✓ 0.7s

Figure 50: Subplots of Average Number of Confirmed COVID-19 Cases and Deaths by State Code

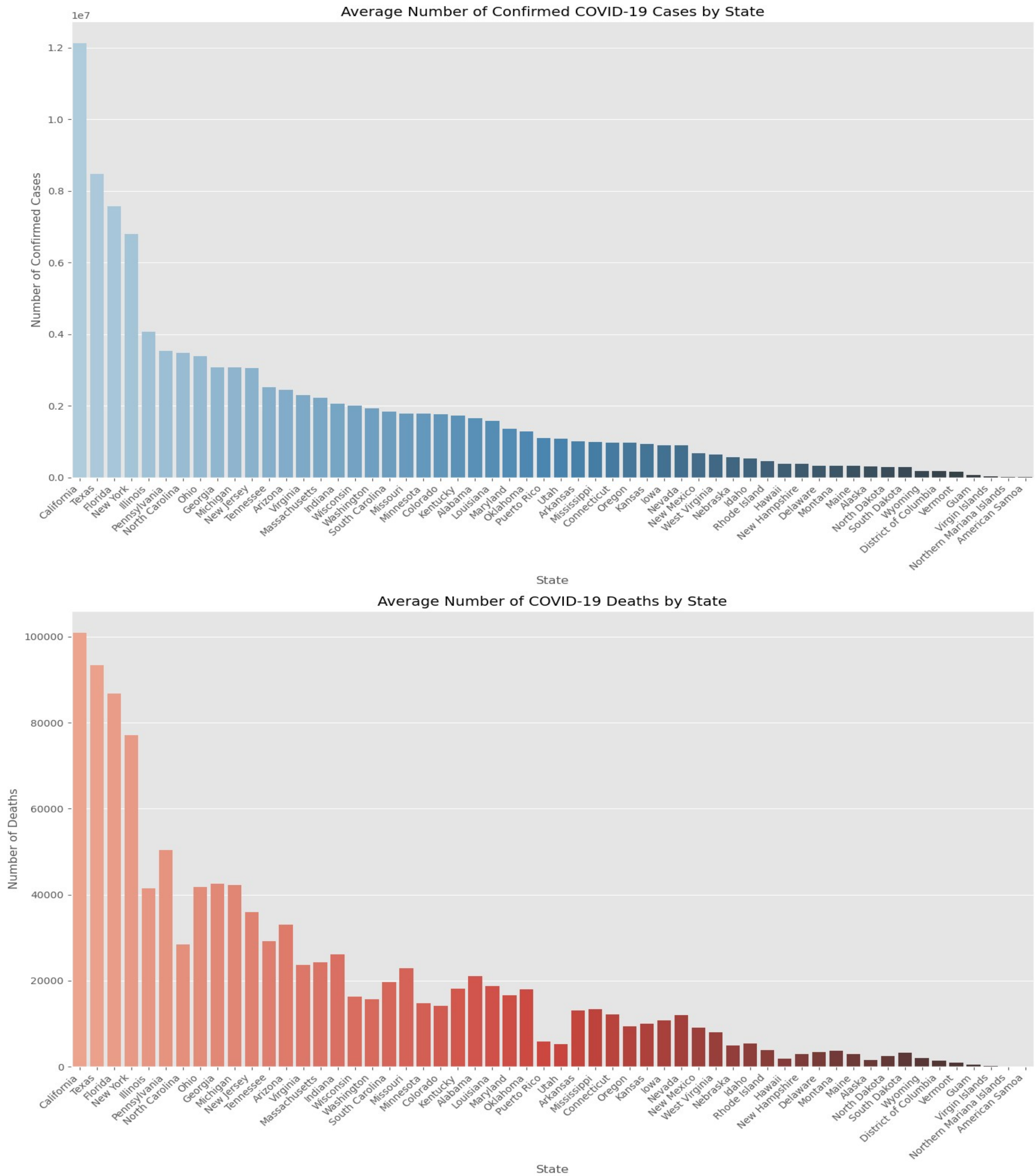

Figure 51: Subplots of Average Number of Confirmed COVID-19 Cases and Deaths by State

I also imported a Pokemon dataset that contains basic information such as the Pokemon name, Type, HP, Attack, and other stats (Fig 52-53). The next graph is a violin plot that displays the distribution of

HP values for Pokemon, separated by generation. It uses `*seaborn*` to create the plot which helps provide a view of the data distribution for each generation. Figure 54 and 55 shows the code and output.

```
### Pokemon Data Analysis and Visualization
#### Import libraries
%matplotlib inline
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

#### Read in dataset
poke_df = pd.read_csv('CSV FIles//pokemon.csv')

display(poke_df)
poke_df.info()
```

✓ 0.0s

Figure 52: Importing Pokemon dataset code

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary
0	1	Bulbasaur	Grass	Poison	45	49	49	65	65	45	1	False
1	2	Ivysaur	Grass	Poison	60	62	63	80	80	60	1	False
2	3	Venusaur	Grass	Poison	80	82	83	100	100	80	1	False
3	4	Mega Venusaur	Grass	Poison	80	100	123	122	120	80	1	False
4	5	Charmander	Fire	NaN	39	52	43	60	50	65	1	False
...	...	...	...	...	...	...	...	...	...	...	...	...
795	796	Diancie	Rock	Fairy	50	100	150	100	150	50	6	True
796	797	Mega Diancie	Rock	Fairy	50	160	110	160	110	110	6	True
797	798	Hoopa Confined	Psychic	Ghost	80	110	60	150	130	70	6	True
798	799	Hoopa Unbound	Psychic	Dark	80	160	60	170	130	80	6	True
799	800	Volcanion	Fire	Water	80	110	120	130	90	70	6	True

800 rows × 12 columns

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 800 entries, 0 to 799
Data columns (total 12 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   #           800 non-null    int64
 1   Name        799 non-null    object
 2   Type 1      800 non-null    object
 3   Type 2      414 non-null    object
 4   HP          800 non-null    int64
 5   Attack      800 non-null    int64
 6   Defense     800 non-null    int64
 7   Sp. Atk     800 non-null    int64
 8   Sp. Def     800 non-null    int64
 9   Speed       800 non-null    int64
 10  Generation  800 non-null    int64
 11  Legendary   800 non-null    bool
dtypes: bool(1), int64(8), object(3)
memory usage: 69.7+ KB
```

Figure 53: Output of Pokemon dataset

Violin Plot with Pokemon Data

```
plt.figure(figsize=(10, 6))
sns.violinplot(x='Generation', y='HP', data=poke_df)
plt.xlabel('Generation')
plt.ylabel('HP')
plt.title('Distribution of HP Across Generations')
plt.show()
```

Figure 54: Violin Plot with Pokemon Data Code

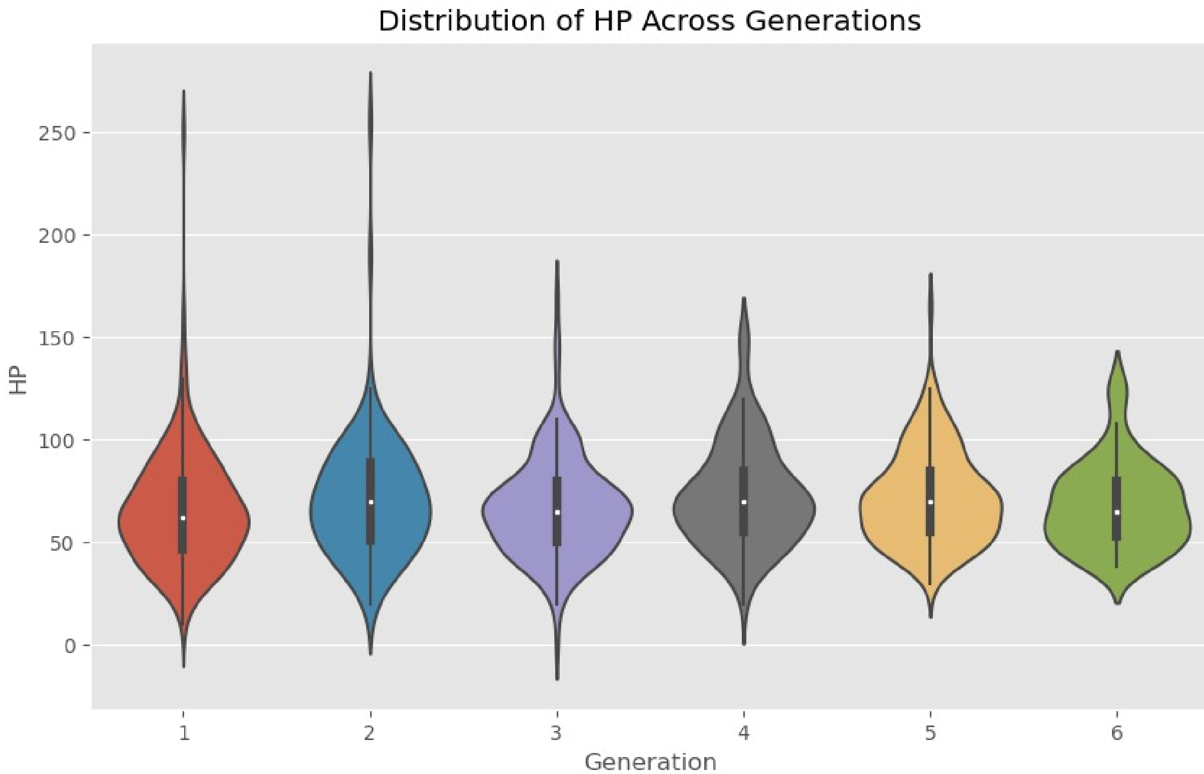

Figure 55: Violin Plot with Pokemon Data

5. Conclusion

This report documents my journey in learning Matplotlib. Matplotlib is a versatile plotting library in Python. I explored concepts that include fundamental plotting techniques essential for visualizing data and customizing the titles, labels, legends and colors of charts and graphs to enhance clarity and aesthetics. Key concepts covered include creating various types of plots such as line plots, bar charts, scatter plots, histograms, and more. By using real data and previous projects, I was able to explore how to go about creating graphs and charts.

References

1. Harshit Shankhdhar. (2021). IMDB Dataset of Top 1000 Movies and TV Shows. Retrieved from https://www.kaggle.com/datasets/harshitshankhdhar/IMDB-dataset-of-top-1000-movies-and-tv-shows

2. Johns Hopkins University Center for Systems Science and Engineering (2024). COVID-19 Data Repository by the Center for Systems Science and Engineering (CSSE) at Johns Hopkins University. GitHub. Retrieved Jun 14, 2024, from https://github.com/CSSEGISandData/COVID-19/tree/master/csse_covid_19_data/csse_covid_19_daily_reports_us

3. Terminus7. (2024). Pokemon Challenge dataset. Kaggle. Retrieved June 14, 2024, from https://www.kaggle.com/datasets/terminus7/pokemon-challenge?select=pokemon.csv

4. VanderPlas, J. (*2016*). *Python Data Science Handbook*. O'Reilly Media. Retrieved from https://jakevdp.github.io/PythonDataScienceHandbook/index.html