Experiment report: Data Visualization

Done by student: **Bylkova Kristina (伯汀娜), 1820249049**
Course: **Big Data Analysis Technology**
Date: **October 2024**

Task 1

Types of data analysis and what kind of Data Visualization tools can support which type of data analysis?

1. Descriptive analysis: deals with what happened in the Past (Microsoft Power BI, Tableau, Matplotlib & Seaborn);
2. Diagnostic analysis: deals with why it happened in the Past (Tableau, Power BI, D3.js, Matplotlib & Seaborn);
3. Predictive analysis: deals with what will happen in the Future (Tableau, Power BI, Plotly, Matplotlib & Seaborn);
4. Perspective analysis: deals with what step to take to avoid a problem in the future (Power BI, Tableau, D3.js, Plotly).

Task 2

Data Visualization tools:

1. Microsoft Power BI: provides real-time data insights through dashboards;
2. Tableau: a tool for creating interactive data visualizations;
3. D3.js (Data-Driven Documents): an open-source library written in javascript, creates dashboards in web browsers;
4. Plotly: a graphing library used for creating interactive, web-based data visualizations, including charts, graphs, and dashboards;
5. Matplotlib & Seaborn: Matplotlib is a Python library for creating visualizations and Seaborn is a Python data visualization library built on top of Matplotlib (it also makes visualizations).

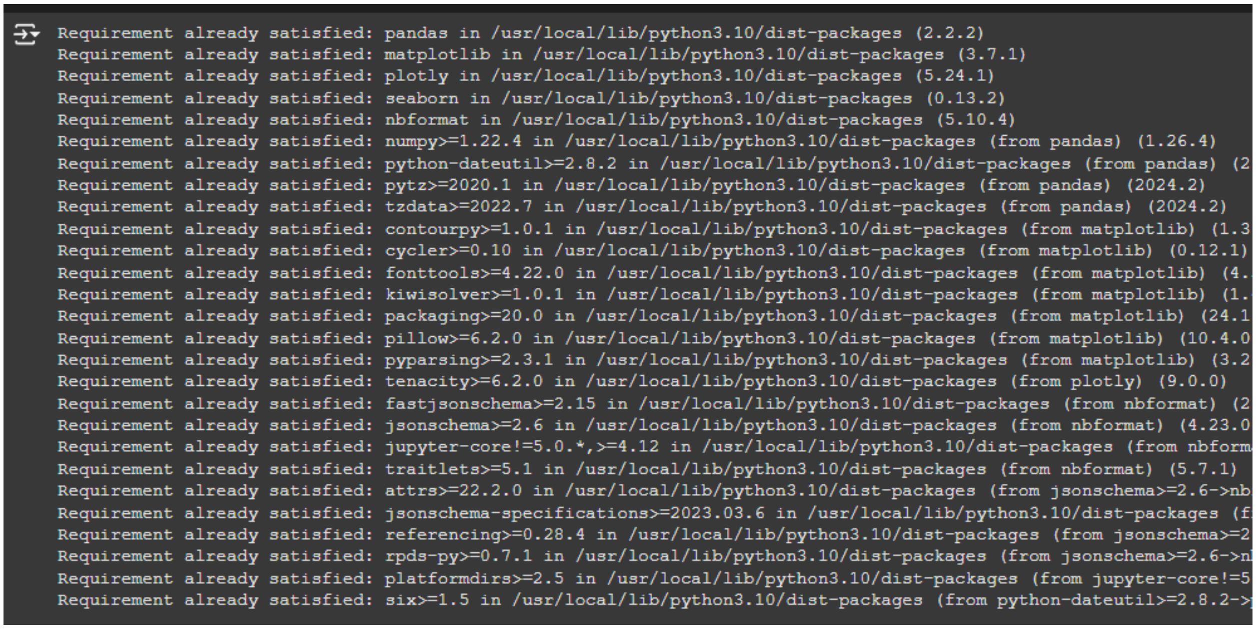
Experiment 1: Statistical Analysis

Here we dwelve in to statistical information, where we filter data through an attribute.

Code:

```python
# Importing all necessary libraries for data manipulation, analysis and visualization
import sys
!{sys.executable} -m pip install pandas matplotlib plotly seaborn nbformat

import plotly.express as px
import plotly.graph_objects as go
import pandas as pd
import sqlite3
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns

# Creating a connection to the database and load the data
conn = sqlite3.connect('imdb_games.db')
df_cleaned = pd.read_sql_query("SELECT * FROM games", conn)
```

Result:



```
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (2.2.2)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (3.7.1)
Requirement already satisfied: plotly in /usr/local/lib/python3.10/dist-packages (5.24.1)
Requirement already satisfied: seaborn in /usr/local/lib/python3.10/dist-packages (0.13.2)
Requirement already satisfied: nbformat in /usr/local/lib/python3.10/dist-packages (5.10.4)
Requirement already satisfied: numpy>=1.22.4 in /usr/local/lib/python3.10/dist-packages (from pandas) (1.26.4)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas) (2
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages (from pandas) (2024.2)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (1.3
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (4.
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (1.
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (24.1
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (10.4.0
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (3.2
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from plotly) (9.0.0)
Requirement already satisfied: fastjsonschema>=2.15 in /usr/local/lib/python3.10/dist-packages (from nbformat) (2
Requirement already satisfied: jsonschema>=2.6 in /usr/local/lib/python3.10/dist-packages (from nbformat) (4.23.0
Requirement already satisfied: jupyter-core!=5.0.*,>=4.12 in /usr/local/lib/python3.10/dist-packages (from nbform
Requirement already satisfied: traitlets>=5.1 in /usr/local/lib/python3.10/dist-packages (from nbformat) (5.7.1)
Requirement already satisfied: attrs>=22.2.0 in /usr/local/lib/python3.10/dist-packages (from jsonschema>=2.6->nb
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in /usr/local/lib/python3.10/dist-packages (f
Requirement already satisfied: referencing>=0.28.4 in /usr/local/lib/python3.10/dist-packages (from jsonschema>=2
Requirement already satisfied: rpds-py>=0.7.1 in /usr/local/lib/python3.10/dist-packages (from jsonschema>=2.6->n
Requirement already satisfied: platformdirs>=2.5 in /usr/local/lib/python3.10/dist-packages (from jupyter-core!=5
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.2->
```

Figure 1: Importing libraries

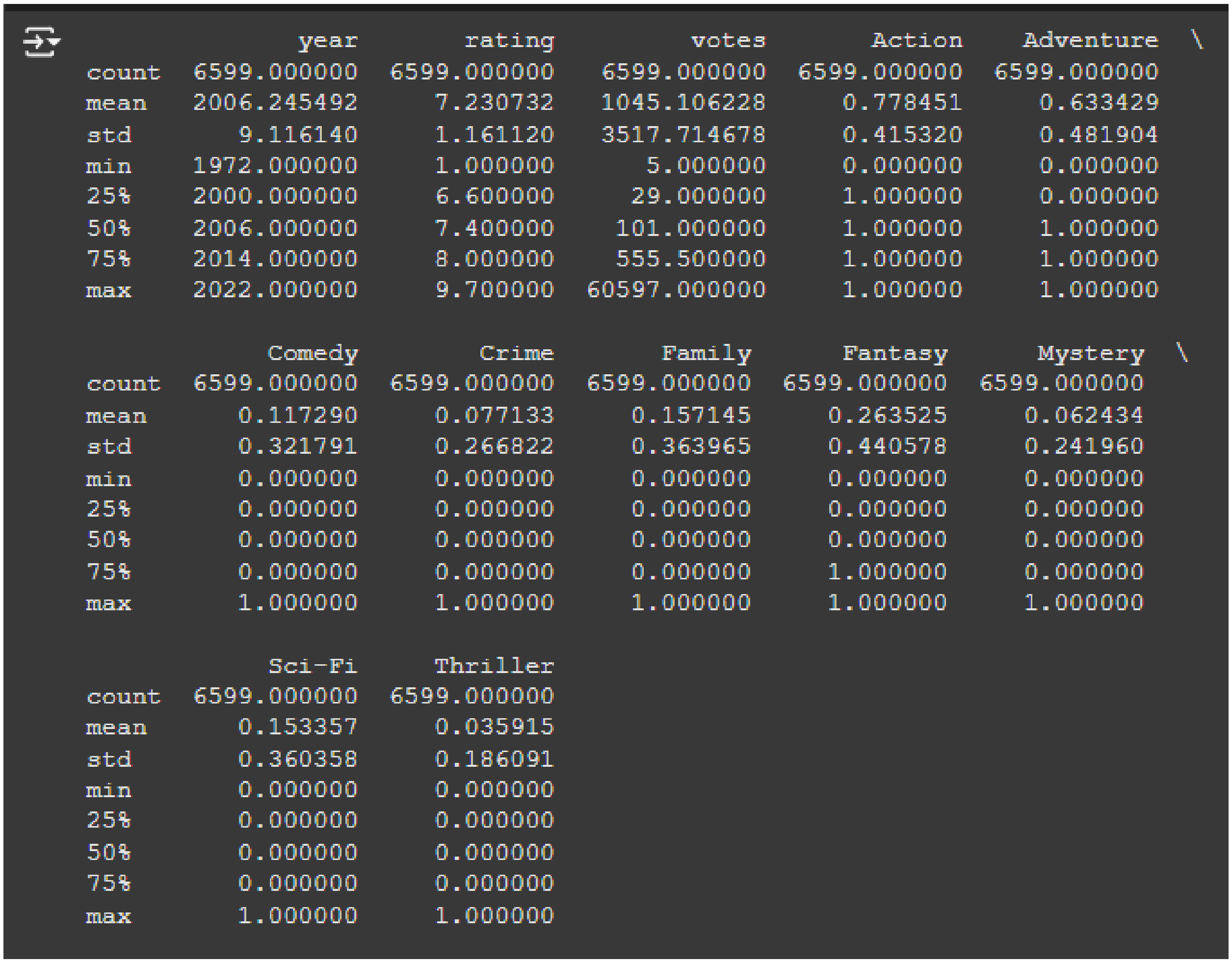
Experiment 1.1: Descriptive Statistics

Getting the descriptive statistics, that is the Count, Mean, Standard Deviation, Percentiles (25%, 50%, 75%), minimum and maximum vaues of each column of the cleaned data from the dataset.

Code:

```python
summary_stats = df_cleaned.describe()
print(summary_stats)
```

Result:



```
              year       rating         votes       Action    Adventure  \
count  6599.000000  6599.000000   6599.000000  6599.000000  6599.000000
mean   2006.245492     7.230732   1045.106228     0.778451     0.633429
std       9.116140     1.161120   3517.714678     0.415320     0.481904
min    1972.000000     1.000000      5.000000     0.000000     0.000000
25%    2000.000000     6.600000     29.000000     1.000000     0.000000
50%    2006.000000     7.400000    101.000000     1.000000     1.000000
75%    2014.000000     8.000000    555.500000     1.000000     1.000000
max    2022.000000     9.700000  60597.000000     1.000000     1.000000

            Comedy        Crime       Family      Fantasy      Mystery  \
count  6599.000000  6599.000000  6599.000000  6599.000000  6599.000000
mean      0.117290     0.077133     0.157145     0.263525     0.062434
std       0.321791     0.266822     0.363965     0.440578     0.241960
min       0.000000     0.000000     0.000000     0.000000     0.000000
25%       0.000000     0.000000     0.000000     0.000000     0.000000
50%       0.000000     0.000000     0.000000     0.000000     0.000000
75%       0.000000     0.000000     0.000000     1.000000     0.000000
max       1.000000     1.000000     1.000000     1.000000     1.000000

            Sci-Fi     Thriller
count  6599.000000  6599.000000
mean      0.153357     0.035915
std       0.360358     0.186091
min       0.000000     0.000000
25%       0.000000     0.000000
50%       0.000000     0.000000
75%       0.000000     0.000000
max       1.000000     1.000000
```

Figure 2: Descriptive Statistics

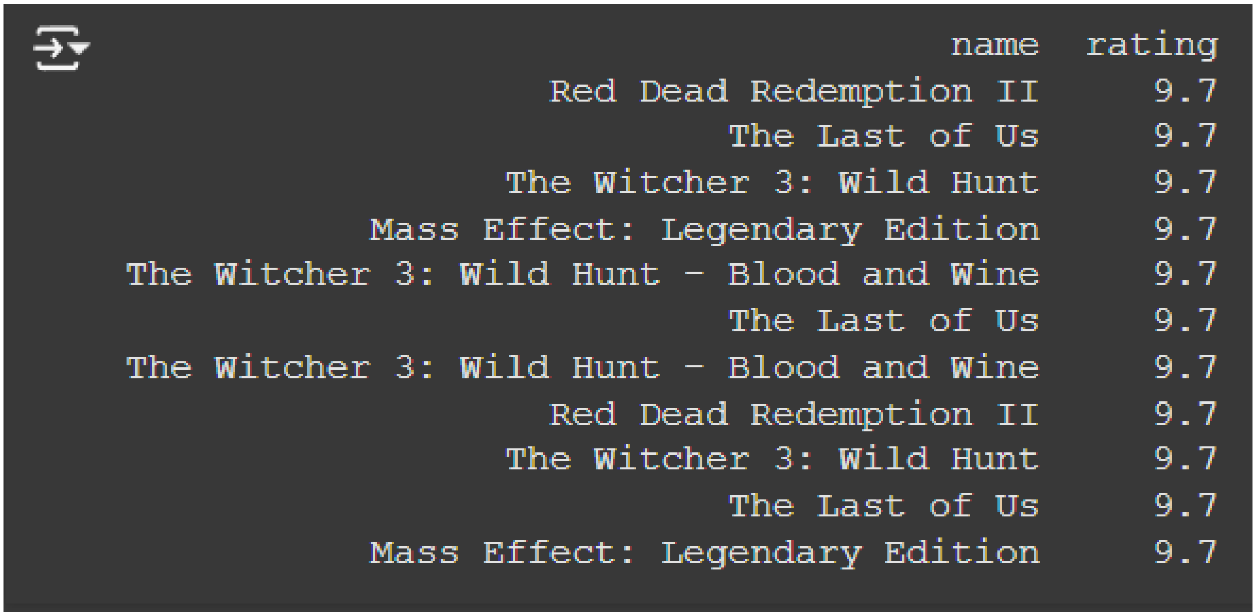
Experiment 1.2: Top 11 games by ratings

Discovering the top 11 games by ratings from the cleaned dataset.

Code:

```python
conn = sqlite3.connect('imdb_games.db')

query = '''
SELECT name, rating
FROM games
ORDER BY rating DESC
LIMIT 11
'''
top_games = pd.read_sql(query, conn)

conn.close()

print(top_games.to_string(index=False))
```

Result:



```
                                     name rating
                   Red Dead Redemption II    9.7
                           The Last of Us    9.7
                 The Witcher 3: Wild Hunt    9.7
           Mass Effect: Legendary Edition    9.7
The Witcher 3: Wild Hunt - Blood and Wine    9.7
                           The Last of Us    9.7
The Witcher 3: Wild Hunt - Blood and Wine    9.7
                   Red Dead Redemption II    9.7
                 The Witcher 3: Wild Hunt    9.7
                           The Last of Us    9.7
           Mass Effect: Legendary Edition    9.7
```

Figure 3: Top 11 games by ratings

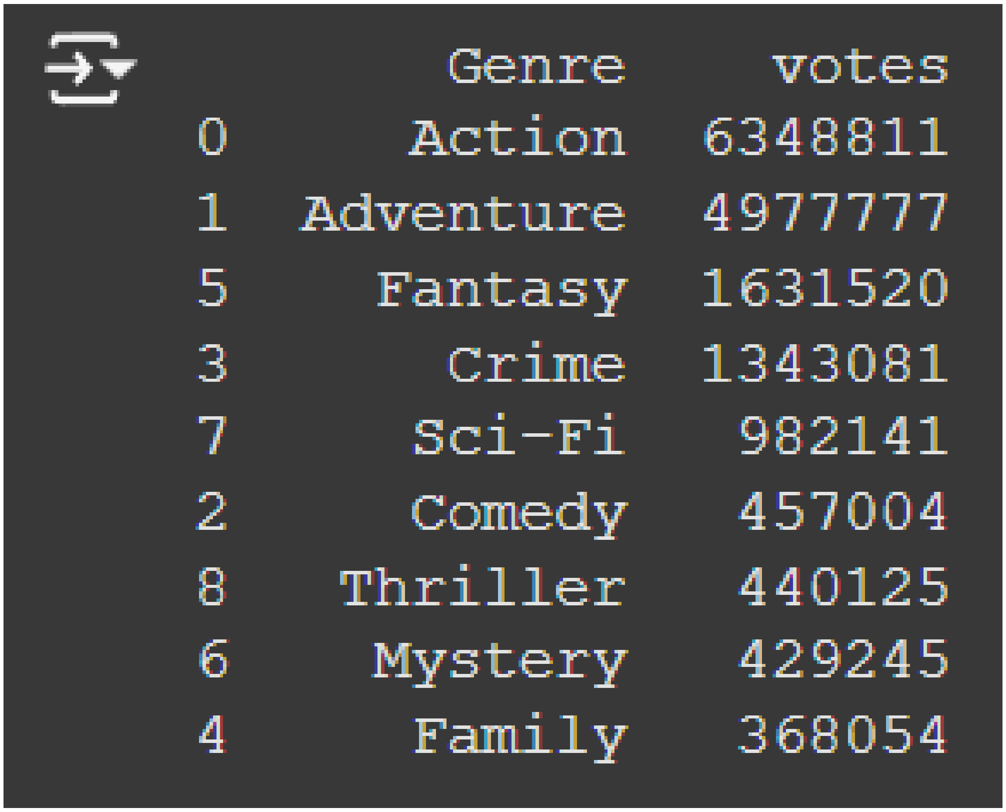
Experiment 1.3: Total Votes for Each Game

Discovering the total votes for each game genre.

Code:

```python
# Defining the genre columns
genre_columns = ['Action', 'Adventure', 'Comedy', 'Crime', 'Family',
                 'Fantasy', 'Mystery', 'Sci-Fi', 'Thriller']

melted_df = df_cleaned.melt(id_vars=['votes'], value_vars=genre_columns,
                            var_name='Genre', value_name='Is_Genre')

# Filtering only applicable genres
filtered_df = melted_df[melted_df['Is_Genre'] == 1]

# Group by Genre and sum votes
total_votes_by_genre = filtered_df.groupby('Genre')['votes'].sum().reset_index()

# Sort by total votes in descending order
total_votes_by_genre = total_votes_by_genre.sort_values(by='votes', ascending=False)

print(total_votes_by_genre)
```

Result:



```
       Genre    votes
0     Action  6348811
1  Adventure  4977777
5    Fantasy  1631520
3      Crime  1343081
7     Sci-Fi   982141
2     Comedy   457004
8   Thriller   440125
6    Mystery   429245
4     Family   368054
```

Figure 4: Total votes for each genre

Experiment 2: Visualization

Experiment 2.1: Distribution of ratings

Getting the distribution of ratings with the use of a histogram.

Code:

```python
mean_rating = df_cleaned['rating'].mean()

fig = px.histogram(df_cleaned, x='rating', nbins=90,
                   title='Distribution of Video Game Ratings',
                   labels={'rating': 'Rating'},
                   color_discrete_sequence=['purple'])

fig.add_vline(x=mean_rating, line_color='red', line_dash='dash',
              annotation_text=f'Mean: {mean_rating:.2f}',
              annotation_position='top right')

fig.update_layout(xaxis_title='Rating', yaxis_title='Frequency')

fig.show()
```

Result:

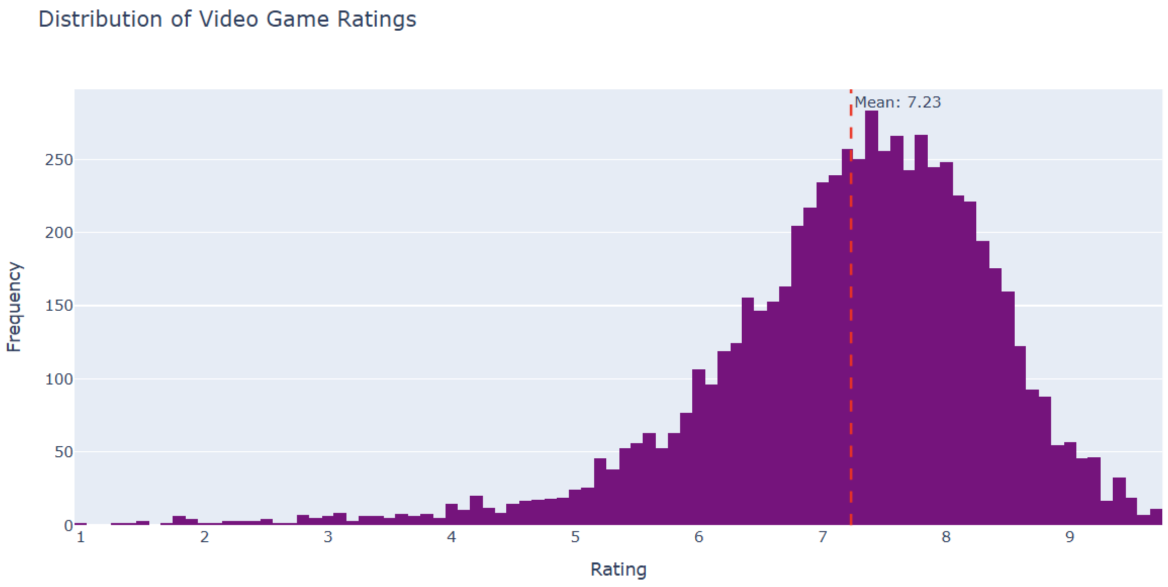


Figure 5: Distribution of ratings

Experiment 2.2: Ratings over the years with trendline

Getting the ratings over the years using a line plot aided with a trendline.

Code:

```python
average_ratings = df_cleaned.groupby('year')['rating'].mean().reset_index()

fig = px.line(average_ratings, x='year', y='rating',
              title='Average Ratings of Video Games Over the Years',
              labels={'year': 'Year', 'rating': 'Average Rating'},
              markers=True)

trendline = go.Scatter(x=average_ratings['year'], y=average_ratings['rating'],
                        mode='lines',
                        name='Trendline',
                        line=dict(color='red'))

fig.add_trace(trendline)

fig.show()
```

Result:

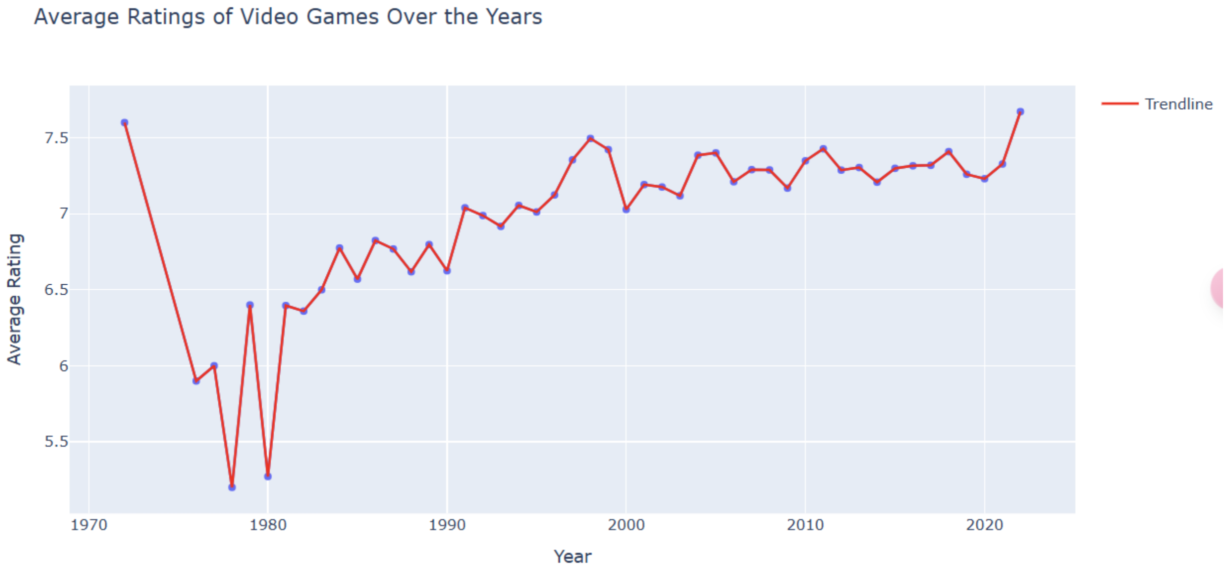


Figure 6: Ratings over the years with trendline

Experiment 3: Optional Experiment

Experiment 3.1: How are certificates related to different genres?

Exploring the relationship between game certificates and all the genres using a grouped bar chart to visualize.

Code:

```python
melted_df = df_cleaned.melt(id_vars=['certificate', 'rating'], value_vars=genre_columns,
                             var_name='Genre', value_name='Is_Genre')

filtered_df = melted_df[melted_df['Is_Genre'] == 1]

# Grouping by certificate and genre to calculate the average rating
cert_genre_relation = filtered_df.groupby(['certificate', 'Genre'])['rating'].mean().reset_index()

fig = px.bar(cert_genre_relation, x='certificate', y='rating', color='Genre',
             title='Average Genre Ratings by Certificate',
             labels={'certificate': 'Certificate', 'rating': 'Average Rating'},
             barmode='group')

fig.update_layout(xaxis_title='Certificate',
                  yaxis_title='Average Rating',
                  legend_title='Genre')

fig.show()
```

Result:

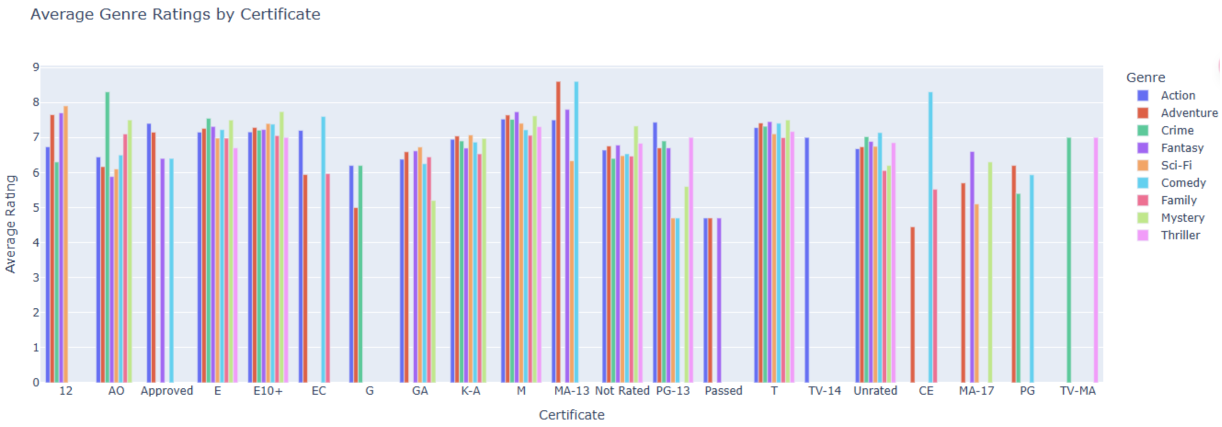


Figure 7: Relation of certificates

Experiment 3.2: What is the relationship between the Genres, ratings, release year and votes?

Making a correlation heatmap.

Code:

```python
numeric_df = df_cleaned.select_dtypes(include=[np.number])

plt.figure(figsize=(10, 8))
correlation_matrix = numeric_df.corr()  # Calculating correlation on numeric DataFrame
sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```

Result:

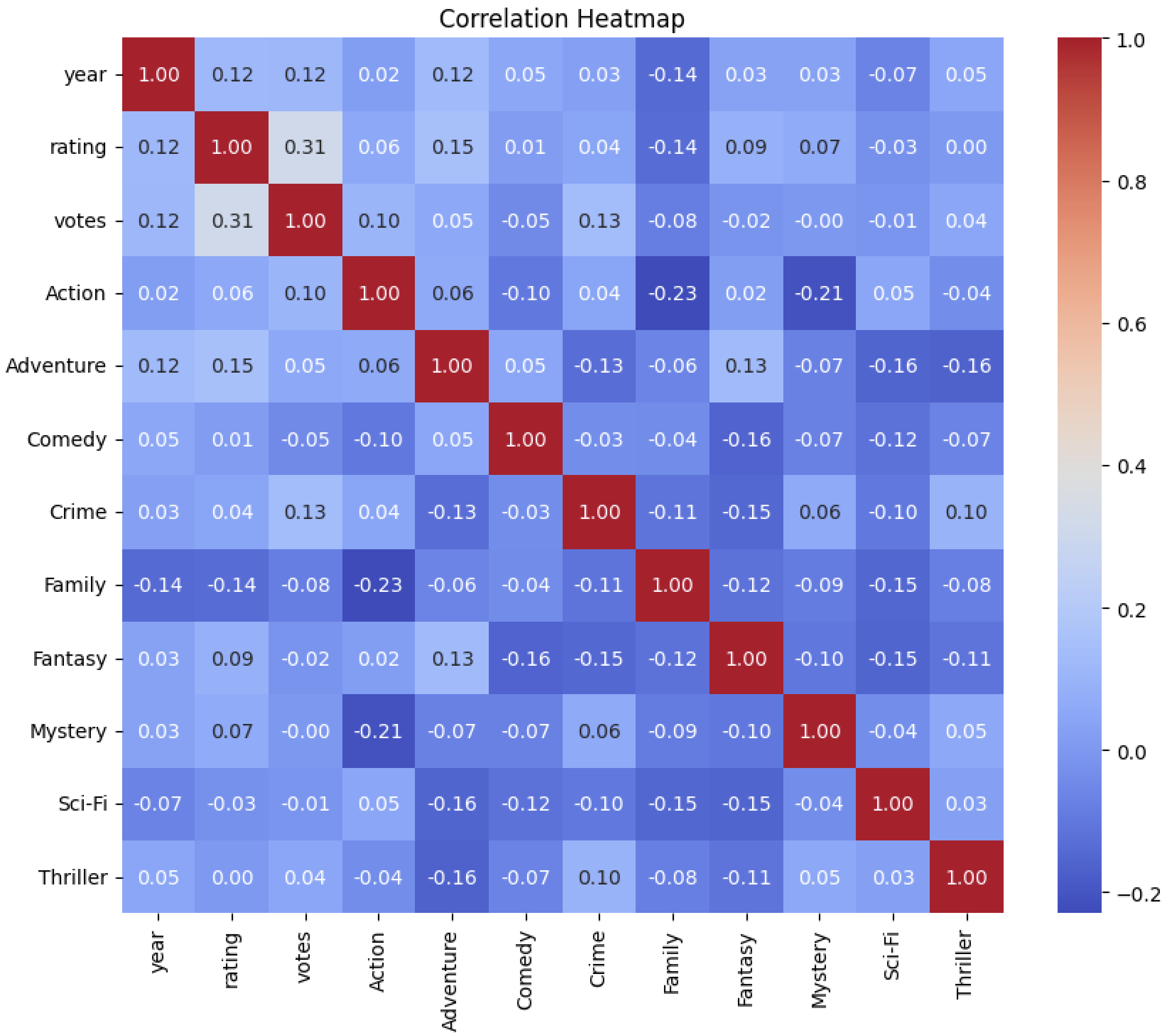


Figure 8: Correlation Heatmap