soft-movie-analysis-daphine-lucas

February 20, 2024

0.1 MICROSOFT_MOVIE _ANALYSIS

0.1.1 Author: DAPHINE LUCAS

Overview Microsoft has opted to establish a new film studio and seeks deeper insights into the most successful film genres at the box office. This endeavor employs descriptive statistical analysis utilizing data sourced from the IMDb website. The objective is to discern which amalgamation of genres resonates most with audiences across various metrics. To accomplish this, four distinct datasets were utilized, focusing on domestic and foreign gross sales, average ratings, number of votes, and production budgets in relation to gross revenue. The analysis identified the top-performing genre combinations in each category. Notably, the combination of Action, Adventure, and Sci-Fi emerged as the predominant choice across domestic sales, foreign sales, and number of votes, with Adventure featuring prominently in the top 20 across all three categories. Based on these findings, the recommendation for Microsoft's movie production endeavors would be to prioritize films in the Action, Adventure, and Sci-Fi genres, considering their prevalence and success within the dataset, which comprised 322 unique genre combinations post-cleaning. Additionally, Adventure and Action paired with either Animation or Fantasy are deemed viable options, given their demonstrated success. Furthermore, the combination of Adventure, Animation, and Comedy showed promising performance in both domestic and foreign sales, making it a compelling third recommendation. Adventure emerges as a consistent and robust genre choice for producing popular and successful movies.

Business problem Microsoft aims to produce profitable movies and seeks to understand which genres are most successful in achieving this objective. To address this inquiry, an analysis of domestic and foreign sales data, as well as the production budget relative to domestic and worldwide gross, was conducted. This analysis aimed to identify the genres that yield the highest financial returns. Additionally, the average rating and number of votes for each genre were examined to gauge the correlation between popularity and financial success.

Questions to use while analyzing the data sets What are the most voted movies and highly rated movie types?

What movie types have the highest average domestic gross and foreign gross income?

What specific metrics are considered when defining the success of a film genre at the box office?(domestic gross sale,foreign gross sales, worldwide gross sales, production budget, average rating, number of votes, genre specific metrics, rankings)

How do production budgets correlate with gross revenue for different genres, and what implications does this have for investment decisions?

What factors must be considered when determining the ideal combination of genres for Microsoft's film production initiatives?

Data sets used: bom.movie_gross

imdb.title.basics

imdb.title.ratings

tn.movie_budgets

0.1.2 DATA LOADING

Getting a brief description of the data to be analyzed

[1]:
```
# import libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

[2]:
```
# Loading the gross income data
df1 = pd.read_csv ('C:/Users/USER/Downloads/bom.movie_gross.csv')
```

[3]:
```
# data understanding
df1.head()
```

[3]:
```
                                          title studio  domestic_gross  \
0                                  Toy Story 3     BV     415000000.0
1                    Alice in Wonderland (2010)     BV     334200000.0
2  Harry Potter and the Deathly Hallows Part 1     WB     296000000.0
3                                     Inception     WB     292600000.0
4                          Shrek Forever After   P/DW     238700000.0

  foreign_gross  year
0     652000000  2010
1     691300000  2010
2     664300000  2010
3     535700000  2010
4     513900000  2010
```

[4]:
```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3387 entries, 0 to 3386
Data columns (total 5 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   title           3387 non-null   object
 1   studio          3382 non-null   object
```

```
 2   domestic_gross  3359 non-null   float64
 3   foreign_gross   2037 non-null   object
 4   year            3387 non-null   int64
dtypes: float64(1), int64(1), object(3)
memory usage: 132.4+ KB
```

[5]:
```
df1.describe()
```

[5]:
```
       domestic_gross         year
count    3.359000e+03  3387.000000
mean     2.874585e+07  2013.958075
std      6.698250e+07     2.478141
min      1.000000e+02  2010.000000
25%      1.200000e+05  2012.000000
50%      1.400000e+06  2014.000000
75%      2.790000e+07  2016.000000
max      9.367000e+08  2018.000000
```

[6]:
```
# Loading basics csv with information on runtime
df2 = pd.read_csv('C:/Users/USER/Downloads/title.basics.csv')
```

[7]:
```
# Data understanding
df2.head()
```

[7]:
```
      tconst                           title              original_title  \
0  tt0063540                       Sunghursh                   Sunghursh
1  tt0066787  One Day Before the Rainy Season             Ashad Ka Ek Din
2  tt0069049       The Other Side of the Wind  The Other Side of the Wind
3  tt0069204                  Sabse Bada Sukh             Sabse Bada Sukh
4  tt0100275         The Wandering Soap Opera       La Telenovela Errante

   start_year  runtime_minutes                genres
0        2013            175.0    Action,Crime,Drama
1        2019            114.0       Biography,Drama
2        2018            122.0                 Drama
3        2018              NaN          Comedy,Drama
4        2017             80.0  Comedy,Drama,Fantasy
```

[8]:
```
print(df2.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 146144 entries, 0 to 146143
Data columns (total 6 columns):
 #   Column          Non-Null Count   Dtype
---  ------          --------------   -----
 0   tconst          146144 non-null  object
 1   title           146144 non-null  object
 2   original_title  146123 non-null  object
```

```
 3   start_year       146144 non-null  int64
 4   runtime_minutes  114405 non-null  float64
 5   genres           140736 non-null  object
dtypes: float64(1), int64(1), object(4)
memory usage: 6.7+ MB
None
```

[9]:
```
# Loading data on ratings and votes
df3 =pd.read_csv("C:/Users/USER/Downloads/title.ratings.csv")
```

[10]:
```
# Data understanding
df3.head()
```

[10]:

	tconst	averagerating	numvotes
0	tt10356526	8.3	31
1	tt10384606	8.9	559
2	tt1042974	6.4	20
3	tt1043726	4.2	50352
4	tt1060240	6.5	21

[11]:
```
print(df3.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 73856 entries, 0 to 73855
Data columns (total 3 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   tconst         73856 non-null  object
 1   averagerating  73856 non-null  float64
 2   numvotes       73856 non-null  int64
dtypes: float64(1), int64(1), object(1)
memory usage: 1.7+ MB
None
```

[12]:
```
df3.describe()
```

[12]:

	averagerating	numvotes
count	73856.000000	7.385600e+04
mean	6.332729	3.523662e+03
std	1.474978	3.029402e+04
min	1.000000	5.000000e+00
25%	5.500000	1.400000e+01
50%	6.500000	4.900000e+01
75%	7.400000	2.820000e+02
max	10.000000	1.841066e+06

[13]:
```
df4= pd.read_csv('C:/Users/USER/OneDrive/Desktop/Project 1/tn.movie_budgets.
↪csv')
```

[14]:
```
df4.head()
```

[14]:
```
   id  release_date                                    movie  \
0   1  Dec 18, 2009                                   Avatar
1   2  May 20, 2011  Pirates of the Caribbean: On Stranger Tides
2   3   Jun 7, 2019                                Dark Phoenix
3   4   May 1, 2015                  Avengers: Age of Ultron
4   5  Dec 15, 2017        Star Wars Ep. VIII: The Last Jedi

  production_budget domestic_gross worldwide_gross
0      $425,000,000   $760,507,625  $2,776,345,279
1      $410,600,000   $241,063,875  $1,045,663,875
2      $350,000,000    $42,762,350    $149,762,350
3      $330,600,000   $459,005,868  $1,403,013,963
4      $317,000,000   $620,181,382  $1,316,721,747
```

[15]:
```
df4.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5782 entries, 0 to 5781
Data columns (total 6 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   id                 5782 non-null   int64
 1   release_date       5782 non-null   object
 2   movie              5782 non-null   object
 3   production_budget  5782 non-null   object
 4   domestic_gross     5782 non-null   object
 5   worldwide_gross    5782 non-null   object
dtypes: int64(1), object(5)
memory usage: 271.2+ KB
```

merging of datasets df1 and df2

[16]:
```
# df1 and df2 has primary key and foreign key that is similar, which is title
merged_df1 = pd.merge(df1, df2, on='title')
merged_df1.head()
```

[16]:
```
                      title studio  domestic_gross foreign_gross  year  \
0                 Toy Story 3     BV     415000000.0     652000000  2010
1                   Inception     WB     292600000.0     535700000  2010
2         Shrek Forever After   P/DW     238700000.0     513900000  2010
3  The Twilight Saga: Eclipse   Sum.     300500000.0     398000000  2010
4                   Iron Man 2   Par.     312400000.0     311500000  2010

      tconst              original_title  start_year  runtime_minutes  \
0  tt0435761                Toy Story 3        2010            103.0
1  tt1375666                   Inception        2010            148.0
```

```
2  tt0892791         Shrek Forever After        2010             93.0
3  tt1325004  The Twilight Saga: Eclipse        2010            124.0
4  tt1228705                  Iron Man 2        2010            124.0

                       genres
0  Adventure,Animation,Comedy
1     Action,Adventure,Sci-Fi
2  Adventure,Animation,Comedy
3     Adventure,Drama,Fantasy
4     Action,Adventure,Sci-Fi
```

merging the merged data with df3

[17]:
```
#merged_df2 and df3 has primary key and foreign key that is similar, which is␣
 ↪tconst
merged_df2 = pd.merge(merged_df1, df3, on = 'tconst')
merged_df2.head()
```

[17]:
```
                       title studio  domestic_gross foreign_gross  year  \
0                Toy Story 3     BV     415000000.0     652000000  2010
1                  Inception     WB     292600000.0     535700000  2010
2         Shrek Forever After   P/DW     238700000.0     513900000  2010
3  The Twilight Saga: Eclipse   Sum.     300500000.0     398000000  2010
4                  Iron Man 2   Par.     312400000.0     311500000  2010

      tconst              original_title  start_year  runtime_minutes  \
0  tt0435761                Toy Story 3        2010            103.0
1  tt1375666                  Inception        2010            148.0
2  tt0892791         Shrek Forever After        2010             93.0
3  tt1325004  The Twilight Saga: Eclipse        2010            124.0
4  tt1228705                  Iron Man 2        2010            124.0

                       genres  averagerating  numvotes
0  Adventure,Animation,Comedy            8.3    682218
1     Action,Adventure,Sci-Fi            8.8   1841066
2  Adventure,Animation,Comedy            6.3    167532
3     Adventure,Drama,Fantasy            5.0    211733
4     Action,Adventure,Sci-Fi            7.0    657690
```

[18]:
```
# concatenating
# Merging using .merge resulted to an error as the data involoved was float64␣
 ↪and object columns.
# We use pd.concat for concatenating DataFrames along either axis (rows or␣
 ↪columns), we are merging not based on a common key.

merged_df3 = pd.concat([merged_df2, df4],axis=0)
merged_df3.head()
```

[18]:

	title	studio	domestic_gross	foreign_gross	year	\
0	Toy Story 3	BV	4.15e+08	652000000	2010.0	
1	Inception	WB	2.926e+08	535700000	2010.0	
2	Shrek Forever After	P/DW	2.387e+08	513900000	2010.0	
3	The Twilight Saga: Eclipse	Sum.	3.005e+08	398000000	2010.0	
4	Iron Man 2	Par.	3.124e+08	311500000	2010.0	

	tconst	original_title	start_year	runtime_minutes	\
0	tt0435761	Toy Story 3	2010.0	103.0	
1	tt1375666	Inception	2010.0	148.0	
2	tt0892791	Shrek Forever After	2010.0	93.0	
3	tt1325004	The Twilight Saga: Eclipse	2010.0	124.0	
4	tt1228705	Iron Man 2	2010.0	124.0	

	genres	averagerating	numvotes	id	release_date	\
0	Adventure,Animation,Comedy	8.3	682218.0	NaN	NaN	
1	Action,Adventure,Sci-Fi	8.8	1841066.0	NaN	NaN	
2	Adventure,Animation,Comedy	6.3	167532.0	NaN	NaN	
3	Adventure,Drama,Fantasy	5.0	211733.0	NaN	NaN	
4	Action,Adventure,Sci-Fi	7.0	657690.0	NaN	NaN	

	movie	production_budget	worldwide_gross
0	NaN	NaN	NaN
1	NaN	NaN	NaN
2	NaN	NaN	NaN
3	NaN	NaN	NaN
4	NaN	NaN	NaN

[19]:
```
# Understand the merged data
merged_df3.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 8807 entries, 0 to 5781
Data columns (total 17 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   title              3025 non-null   object
 1   studio             3022 non-null   object
 2   domestic_gross     8785 non-null   object
 3   foreign_gross      1831 non-null   object
 4   year               3025 non-null   float64
 5   tconst             3025 non-null   object
 6   original_title     3025 non-null   object
 7   start_year         3025 non-null   float64
 8   runtime_minutes    2978 non-null   float64
 9   genres             3018 non-null   object
 10  averagerating      3025 non-null   float64
```

```
 11  numvotes           3025 non-null   float64
 12  id                 5782 non-null   float64
 13  release_date       5782 non-null   object
 14  movie              5782 non-null   object
 15  production_budget  5782 non-null   object
 16  worldwide_gross    5782 non-null   object
dtypes: float64(6), object(11)
memory usage: 1.2+ MB
```

[20]:
```
merged_df3.describe(include='all')
```

[20]:

	title	studio	domestic_gross	foreign_gross	year	tconst
count	3025	3022	8785	1831	3025.000000	3025
unique	2596	215	6686	1006	NaN	3023
top	Gold	Uni.	$0	1200000	NaN	tt2442772
freq	6	156	548	17	NaN	2
mean	NaN	NaN	NaN	NaN	2014.077686	NaN
std	NaN	NaN	NaN	NaN	2.441833	NaN
min	NaN	NaN	NaN	NaN	2010.000000	NaN
25%	NaN	NaN	NaN	NaN	2012.000000	NaN
50%	NaN	NaN	NaN	NaN	2014.000000	NaN
75%	NaN	NaN	NaN	NaN	2016.000000	NaN
max	NaN	NaN	NaN	NaN	2018.000000	NaN

	original_title	start_year	runtime_minutes	genres	averagerating
count	3025	3025.000000	2978.000000	3018	3025.000000
unique	2725	NaN	NaN	322	NaN
top	Eden	NaN	NaN	Drama	NaN
freq	6	NaN	NaN	317	NaN
mean	NaN	2013.783140	107.225991	NaN	6.458612
std	NaN	2.466558	20.077436	NaN	1.011553
min	NaN	2010.000000	3.000000	NaN	1.600000
25%	NaN	2012.000000	94.000000	NaN	5.900000
50%	NaN	2014.000000	105.000000	NaN	6.600000
75%	NaN	2016.000000	118.000000	NaN	7.100000
max	NaN	2019.000000	272.000000	NaN	9.200000

	numvotes	id	release_date	movie	production_budget
count	3.025000e+03	5782.000000	5782	5782	5782
unique	NaN	NaN	2418	5698	509
top	NaN	NaN	Dec 31, 2014	Home	$20,000,000
freq	NaN	NaN	24	3	231
mean	6.173183e+04	50.372363	NaN	NaN	NaN
std	1.255487e+05	28.821076	NaN	NaN	NaN
min	5.000000e+00	1.000000	NaN	NaN	NaN
25%	2.113000e+03	25.000000	NaN	NaN	NaN
50%	1.310900e+04	50.000000	NaN	NaN	NaN

```
75%     6.294200e+04    75.000000           NaN   NaN              NaN
max     1.841066e+06   100.000000           NaN   NaN              NaN

       worldwide_gross
count             5782
unique            5356
top                 $0
freq               367
mean               NaN
std                NaN
min                NaN
25%                NaN
50%                NaN
75%                NaN
max                NaN
```

[21]:
```
# Make the merged data to a dataframe for easier analysis
df = merged_df3
```

[22]:
```
df.shape
```

[22]:
```
(8807, 17)
```

[23]:
```
df
```

[23]:
```
                          title studio domestic_gross foreign_gross    year  \
0                   Toy Story 3     BV       4.15e+08     652000000  2010.0
1                     Inception     WB      2.926e+08     535700000  2010.0
2            Shrek Forever After   P/DW      2.387e+08     513900000  2010.0
3     The Twilight Saga: Eclipse   Sum.      3.005e+08     398000000  2010.0
4                     Iron Man 2   Par.      3.124e+08     311500000  2010.0
…                            …    …            …            …      …
5777                        NaN    NaN            $0           NaN     NaN
5778                        NaN    NaN       $48,482           NaN     NaN
5779                        NaN    NaN        $1,338           NaN     NaN
5780                        NaN    NaN            $0           NaN     NaN
5781                        NaN    NaN      $181,041           NaN     NaN

         tconst              original_title  start_year  runtime_minutes  \
0     tt0435761                Toy Story 3      2010.0            103.0
1     tt1375666                   Inception      2010.0            148.0
2     tt0892791         Shrek Forever After      2010.0             93.0
3     tt1325004  The Twilight Saga: Eclipse      2010.0            124.0
4     tt1228705                  Iron Man 2      2010.0            124.0
…         …                           …         …               …
5777        NaN                        NaN         NaN             NaN
5778        NaN                        NaN         NaN             NaN
```

```
5779        NaN                          NaN         NaN              NaN
5780        NaN                          NaN         NaN              NaN
5781        NaN                          NaN         NaN              NaN

                          genres  averagerating   numvotes    id  \
0     Adventure,Animation,Comedy            8.3   682218.0   NaN
1        Action,Adventure,Sci-Fi            8.8  1841066.0   NaN
2     Adventure,Animation,Comedy            6.3   167532.0   NaN
3        Adventure,Drama,Fantasy            5.0   211733.0   NaN
4        Action,Adventure,Sci-Fi            7.0   657690.0   NaN
…                           …            …         …   …
5777                        NaN            NaN        NaN  78.0
5778                        NaN            NaN        NaN  79.0
5779                        NaN            NaN        NaN  80.0
5780                        NaN            NaN        NaN  81.0
5781                        NaN            NaN        NaN  82.0

      release_date                          movie production_budget  \
0              NaN                            NaN               NaN
1              NaN                            NaN               NaN
2              NaN                            NaN               NaN
3              NaN                            NaN               NaN
4              NaN                            NaN               NaN
…              …                              …                 …
5777  Dec 31, 2018                        Red 11            $7,000
5778   Apr 2, 1999                     Following            $6,000
5779  Jul 13, 2005  Return to the Land of Wonders            $5,000
5780  Sep 29, 2015           A Plague So Pleasant            $1,400
5781   Aug 5, 2005              My Date With Drew            $1,100

     worldwide_gross
0                NaN
1                NaN
2                NaN
3                NaN
4                NaN
…                …
5777              $0
5778        $240,495
5779          $1,338
5780              $0
5781        $181,041

[8807 rows x 17 columns]
```

1 DATA CLEANING

[24]:
```
# Check for missing values
print(df.isnull())# or df.isna()
```

```
      title  studio  domestic_gross  foreign_gross   year  tconst  \
0     False   False           False          False  False   False
1     False   False           False          False  False   False
2     False   False           False          False  False   False
3     False   False           False          False  False   False
4     False   False           False          False  False   False
…     …      …               …              …      …       …
5777   True    True           False           True   True    True
5778   True    True           False           True   True    True
5779   True    True           False           True   True    True
5780   True    True           False           True   True    True
5781   True    True           False           True   True    True

      original_title  start_year  runtime_minutes  genres  averagerating  \
0              False       False            False   False          False
1              False       False            False   False          False
2              False       False            False   False          False
3              False       False            False   False          False
4              False       False            False   False          False
…              …          …                …       …              …
5777            True        True             True    True           True
5778            True        True             True    True           True
5779            True        True             True    True           True
5780            True        True             True    True           True
5781            True        True             True    True           True

      numvotes     id  release_date  movie  production_budget  worldwide_gross
0        False   True          True   True               True             True
1        False   True          True   True               True             True
2        False   True          True   True               True             True
3        False   True          True   True               True             True
4        False   True          True   True               True             True
…        …      …              …      …                  …                …
5777      True  False         False  False              False            False
5778      True  False         False  False              False            False
5779      True  False         False  False              False            False
5780      True  False         False  False              False            False
5781      True  False         False  False              False            False

[8807 rows x 17 columns]
```

[25]:
```
# summation of missing values
print(df.isnull().sum())
```

```
title                5782
studio               5785
domestic_gross         22
foreign_gross        6976
year                 5782
tconst               5782
original_title       5782
start_year           5782
runtime_minutes      5829
genres               5789
averagerating        5782
numvotes             5782
id                   3025
release_date         3025
movie                3025
production_budget    3025
worldwide_gross      3025
dtype: int64
```

[26]:
```
# create a function to check the percentage of missing values
def missing_values(data):
    miss = data.isnull().sum().sort_values(ascending = False)
    percentage_miss = (data.isnull().sum() / len(data)).sort_values(ascending =␣
 ↪False)
    missing = pd.DataFrame({"Missing Values": miss, "Percentage":␣
 ↪percentage_miss}).reset_index()
    missing.drop(missing[missing["Percentage"] == 0].index, inplace = True)
    return missing

missing_data = missing_values(df)
missing_data
```

[26]:

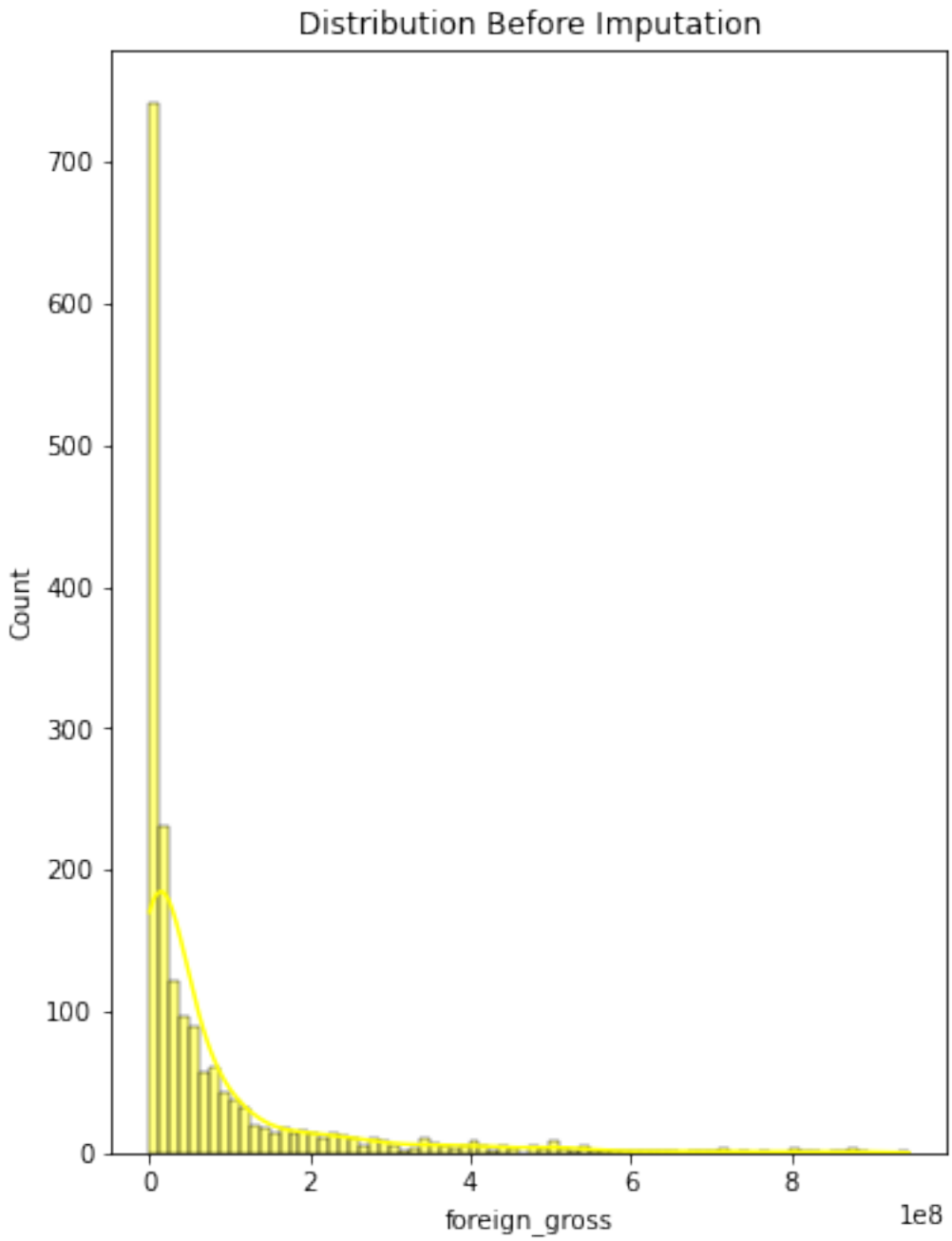
	index	Missing Values	Percentage
0	foreign_gross	6976	0.792097
1	runtime_minutes	5829	0.661860
2	genres	5789	0.657318
3	studio	5785	0.656864
4	start_year	5782	0.656523
5	year	5782	0.656523
6	tconst	5782	0.656523
7	original_title	5782	0.656523
8	title	5782	0.656523
9	averagerating	5782	0.656523
10	numvotes	5782	0.656523

```
11  production_budget            3025    0.343477
12                 id            3025    0.343477
13       release_date            3025    0.343477
14              movie            3025    0.343477
15    worldwide_gross            3025    0.343477
16     domestic_gross              22    0.002498
```

[27]:
```
# Convert 'foreign_gross' column to numeric, ignoring errors to handle␣
 ↪non-numeric values
df['foreign_gross'] = pd.to_numeric(df['foreign_gross'], errors='coerce')
```

[28]:
```
# Information in the foreign column is usefull, instead of dropping it we will␣
 ↪have to impute
# imputing can be done using the mean or median depending on the distribution
# If it's a normal distribution, use the mean
# If it's a skewed distribution, use the median

# Create a figure with dimensions 13x8 inches
plt.figure(figsize=(13, 8))

# Create the first subplot (1 row, 2 columns, first plot)
plt.subplot(1, 2, 1)

# Plot the histogram with KDE of the 'bmi' column from df, dropping missing␣
 ↪values
sns.histplot(df['foreign_gross'], kde=True, color='Yellow')

# Set title for the subplot
plt.title('Distribution Before Imputation')
```

[28]:
```
Text(0.5, 1.0, 'Distribution Before Imputation')
```



[29]:
```
# Affirming this is a skewed distribution
df['foreign_gross'].fillna(df['foreign_gross'].median(), inplace = True)
```

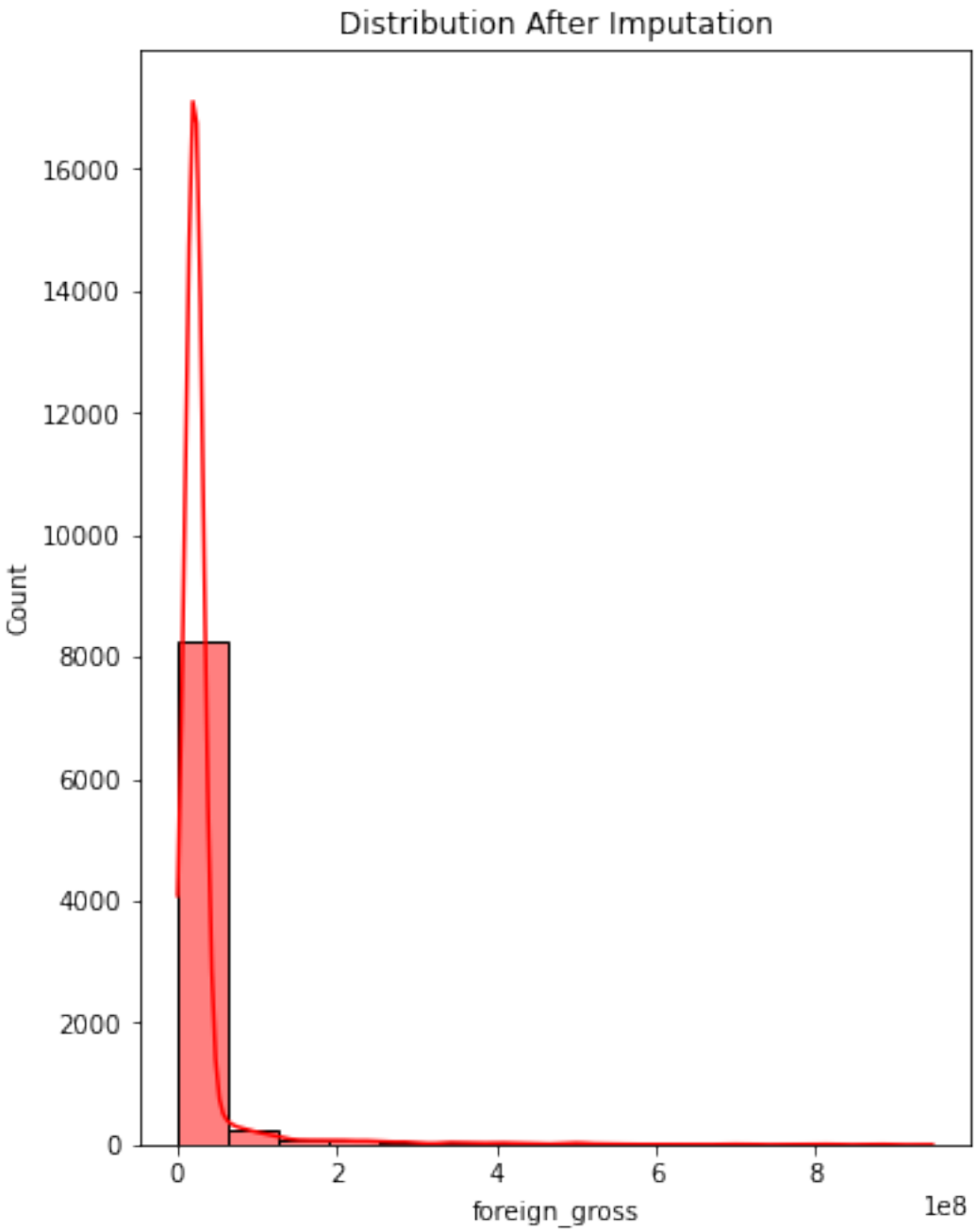
[30]:
```
#check for missing values after imputation
print(df.isnull().sum())
```

```
title                5782
studio               5785
```

```
domestic_gross          22
foreign_gross            0
year                  5782
tconst                5782
original_title        5782
start_year            5782
runtime_minutes       5829
genres                5789
averagerating         5782
numvotes              5782
id                    3025
release_date          3025
movie                 3025
production_budget     3025
worldwide_gross       3025
dtype: int64
```

[31]:
```
# Create a figure with dimensions 13x8 inches
plt.figure(figsize=(13, 8))

# Create the first subplot (1 row, 2 columns, first plot)
plt.subplot(1, 2, 1)

# Plot the histogram with KDE of the 'bmi' column from stroke_data, dropping␣
 ↪missing values
sns.histplot(df['foreign_gross'], kde=True, color='red')

# Set title for the subplot
plt.title('Distribution After Imputation')
```

[31]:
```
Text(0.5, 1.0, 'Distribution After Imputation')
```



[32]:
```
df.info()
```

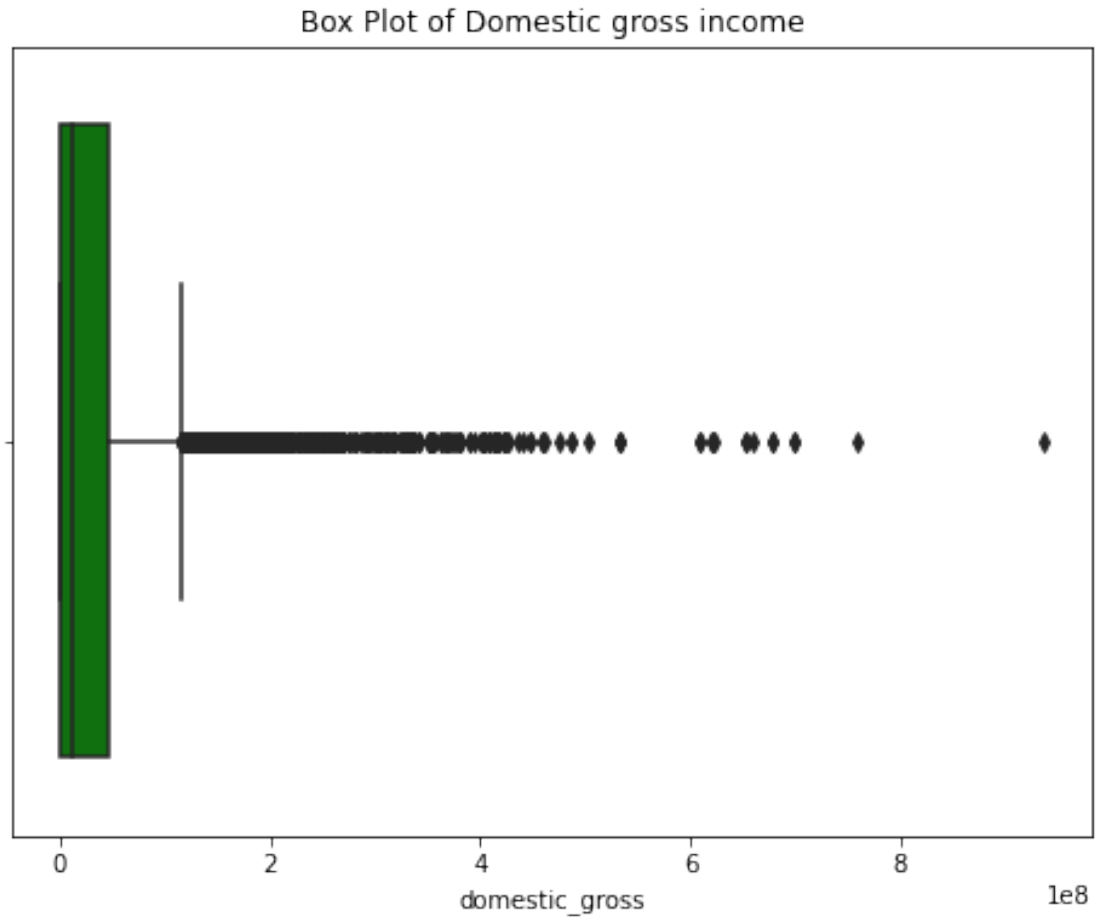
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 8807 entries, 0 to 5781
Data columns (total 17 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   title              3025 non-null   object
```

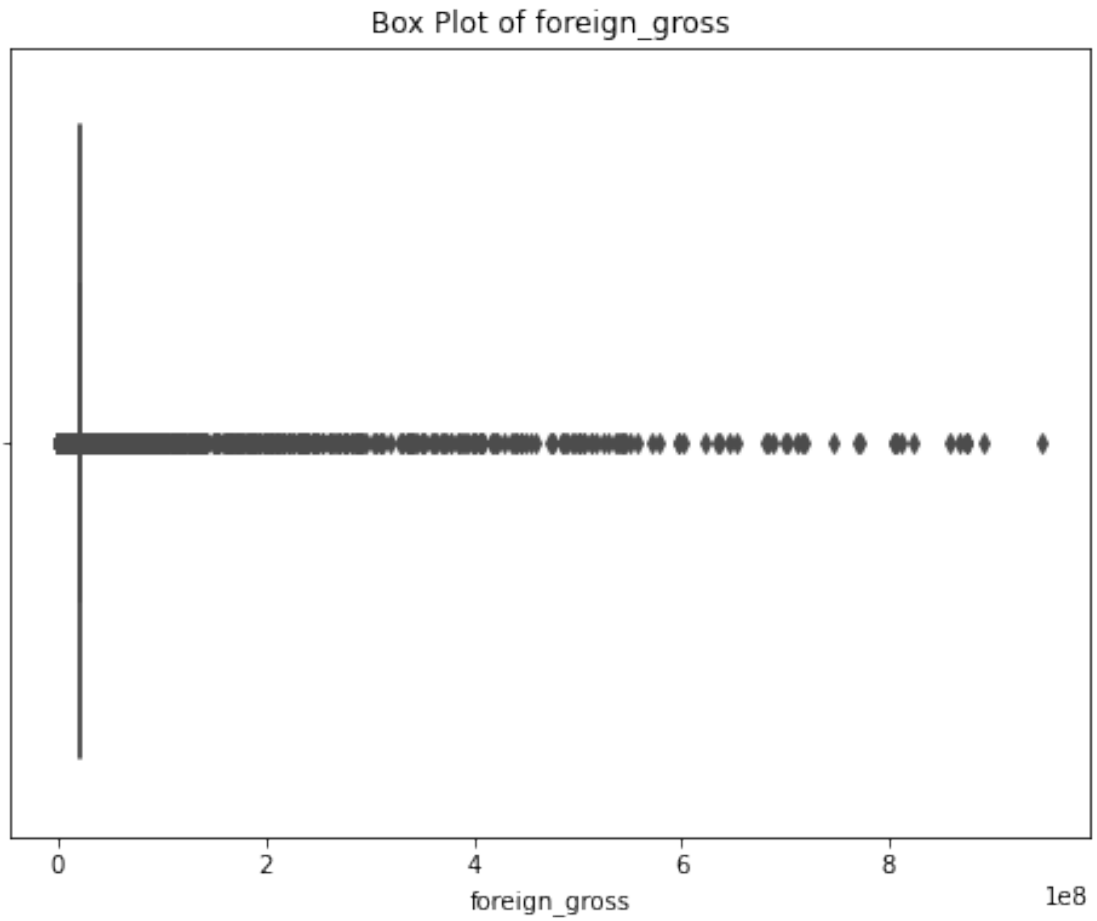
```
 1   studio              3022 non-null   object
 2   domestic_gross      8785 non-null   object
 3   foreign_gross       8807 non-null   float64
 4   year                3025 non-null   float64
 5   tconst              3025 non-null   object
 6   original_title      3025 non-null   object
 7   start_year          3025 non-null   float64
 8   runtime_minutes     2978 non-null   float64
 9   genres              3018 non-null   object
 10  averagerating       3025 non-null   float64
 11  numvotes            3025 non-null   float64
 12  id                  5782 non-null   float64
 13  release_date        5782 non-null   object
 14  movie               5782 non-null   object
 15  production_budget   5782 non-null   object
 16  worldwide_gross     5782 non-null   object
dtypes: float64(7), object(10)
memory usage: 1.2+ MB
```

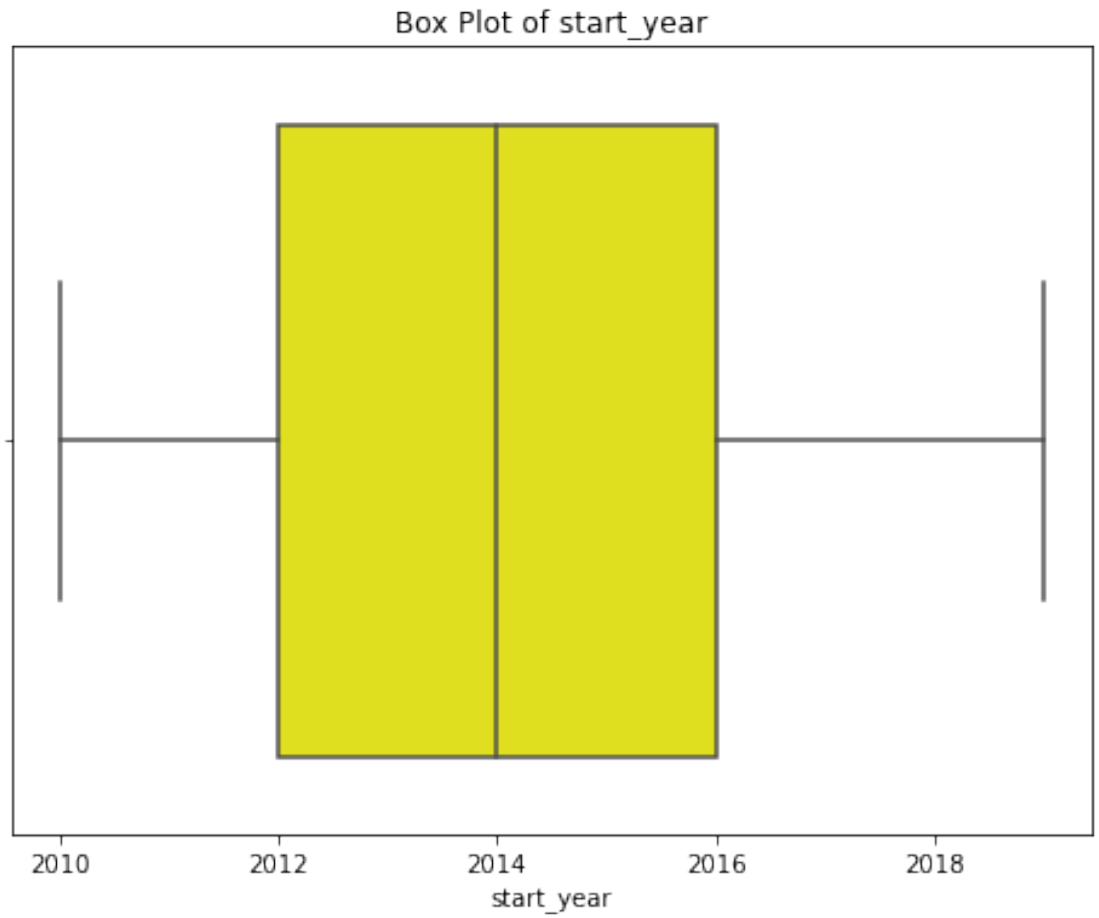
[33]:
```
# Remove non-numeric characters from 'domestic_gross' column
df['domestic_gross'] = df['domestic_gross'].replace('[\$,]', '', regex=True)

# Convert 'domestic_gross' column to float
df['domestic_gross'] = pd.to_numeric(df['domestic_gross'], errors='coerce')
```

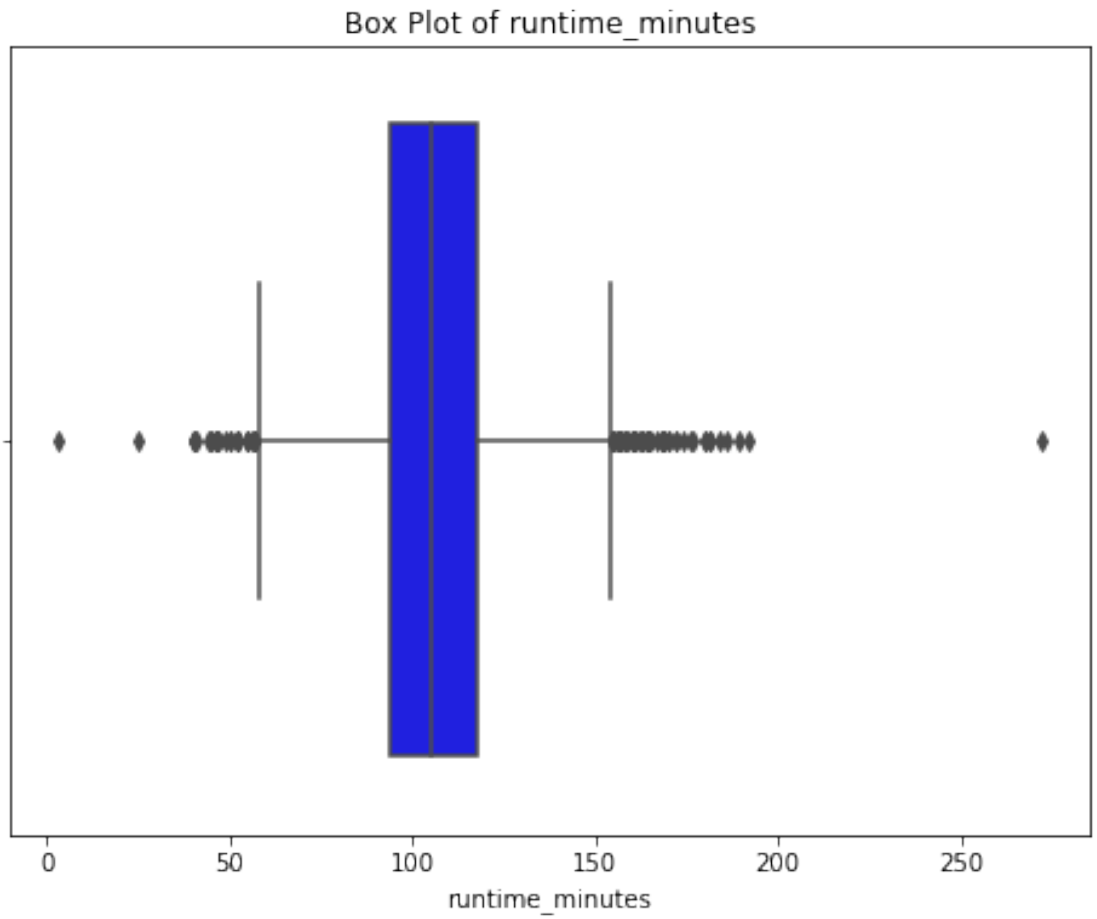
[34]:
```
# Check for outliers

# Box plots are good for identify the outliers
# domestic_gross column
plt.figure(figsize=(8, 6))
sns.boxplot(x=df['domestic_gross'], color='green')
plt.title('Box Plot of Domestic gross income')
plt.show()

# foreign_gross column
plt.figure(figsize=(8, 6))
sns.boxplot(x=df['foreign_gross'], color='red')
plt.title('Box Plot of foreign_gross')
plt.show()

# start_year column
plt.figure(figsize=(8, 6))
sns.boxplot(x=df['start_year'], color='yellow')
plt.title('Box Plot of start_year')
plt.show()

# runtime_minutes column
```

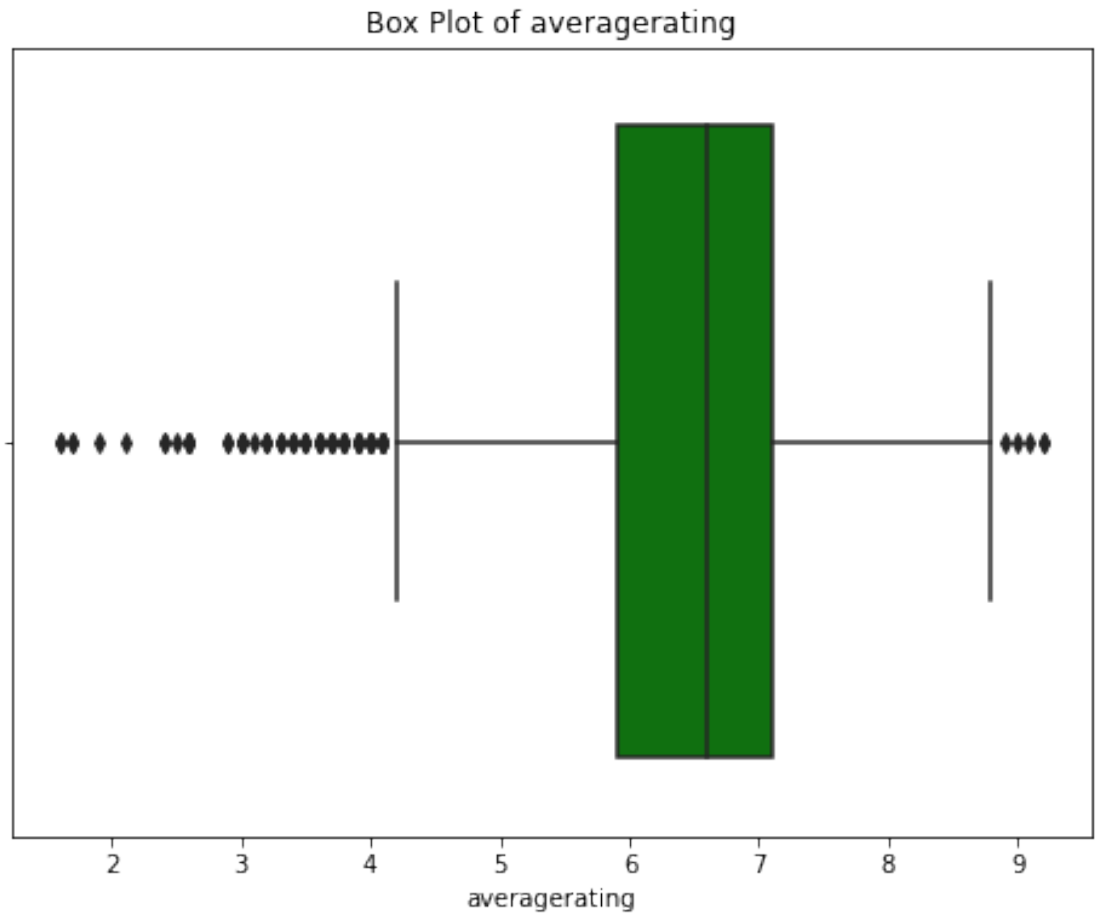
```
plt.figure(figsize=(8, 6))
sns.boxplot(x=df['runtime_minutes'], color='blue')
plt.title('Box Plot of runtime_minutes')
plt.show()

# averagerating column
plt.figure(figsize=(8, 6))
sns.boxplot(x=df['averagerating'], color='green')
plt.title('Box Plot of averagerating')
plt.show()
```




Box Plot of foreign_gross
0
2
4
6
8
foreign_gross
1e8


Box Plot of start_year
2010
2012
2014
2016
2018
start_year


Box Plot of runtime_minutes
0
50
100
150
200
250
runtime_minutes

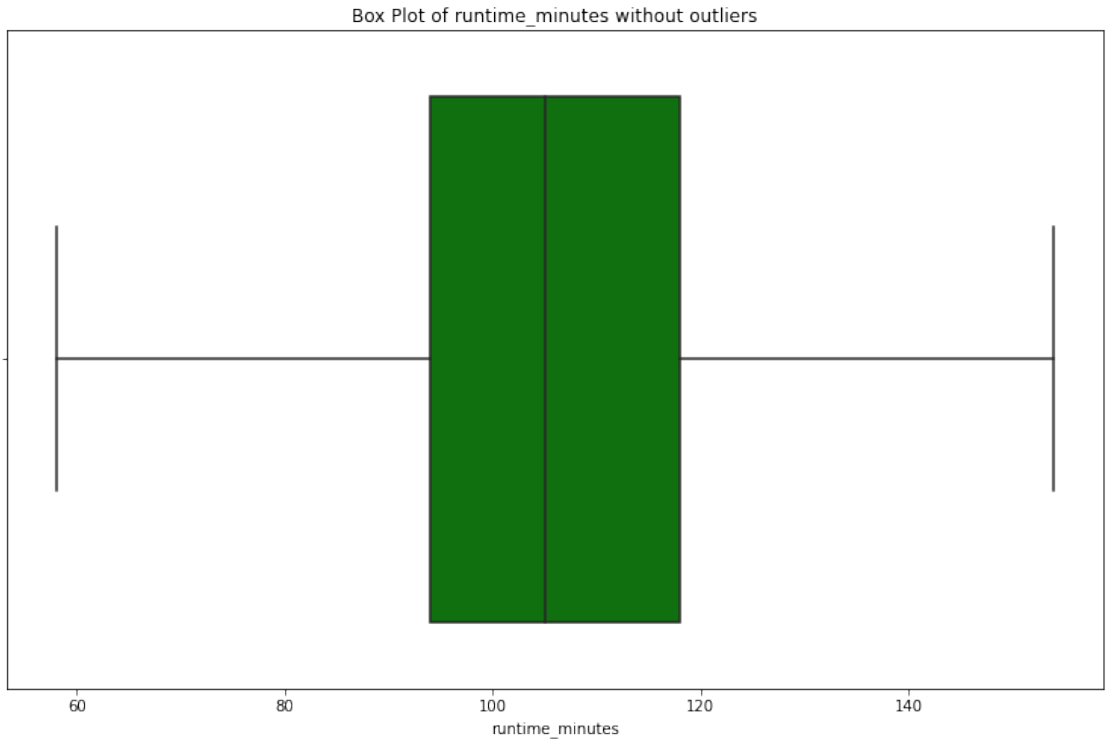


Dealing with outliers

[35]:
```
# cap run time outliers

# Calculate IQR for the 'run time' column
Q1 = df['runtime_minutes'].quantile(0.25)
Q3 = df['runtime_minutes'].quantile(0.75)
IQR = Q3 - Q1

# Define the upper and lower bounds to identify outliers
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Identify outliers
outliers = df[(df['runtime_minutes'] < lower_bound) | (df['runtime_minutes'] >␣
 ↪upper_bound)]

# Capping outliers to the upper and lower bounds
```

```
df['runtime_minutes'] = df['runtime_minutes'].clip(lower=lower_bound,␣
 ↪upper=upper_bound)
```

[36]:
```
# runtime_minutes column after removing outliers
plt.figure(figsize=(13, 8))
sns.boxplot(x=df['runtime_minutes'], color='green')
plt.title('Box Plot of runtime_minutes without outliers')
plt.show()
```



1.0.1 DEALING WITH DUPLICTAES

[37]:
```
# Check for duplicate rows
print(df[df.duplicated()])
```

```
Empty DataFrame
Columns: [title, studio, domestic_gross, foreign_gross, year, tconst,
original_title, start_year, runtime_minutes, genres, averagerating, numvotes,
id, release_date, movie, production_budget, worldwide_gross]
Index: []
```

The output you provided indicates that there are no duplicate rows in your DataFrame. This means that each row is unique based on all columns or the subset of columns you checked for duplicates.

1.1 EXPLARATORY DATA ANALYSIS

[38]:
```
df.describe()
```

[38]:

	domestic_gross	foreign_gross	year	start_year
count	8.785000e+03	8.807000e+03	3025.000000	3025.000000
mean	3.804049e+07	3.315999e+07	2014.077686	2013.783140
std	6.793403e+07	6.729405e+07	2.441833	2.466558
min	0.000000e+00	6.000000e+02	2010.000000	2010.000000
25%	4.028580e+05	2.130000e+07	2012.000000	2012.000000
50%	1.149484e+07	2.130000e+07	2014.000000	2014.000000
75%	4.640000e+07	2.130000e+07	2016.000000	2016.000000
max	9.366622e+08	9.464000e+08	2018.000000	2019.000000

	runtime_minutes	averagerating	numvotes	id
count	2978.000000	3025.000000	3.025000e+03	5782.000000
mean	107.060107	6.458612	6.173183e+04	50.372363
std	18.855284	1.011553	1.255487e+05	28.821076
min	58.000000	1.600000	5.000000e+00	1.000000
25%	94.000000	5.900000	2.113000e+03	25.000000
50%	105.000000	6.600000	1.310900e+04	50.000000
75%	118.000000	7.100000	6.294200e+04	75.000000
max	154.000000	9.200000	1.841066e+06	100.000000

[39]:
```
# Count the number of unique genres
num_genres = df['genres'].nunique()

print("Number of genres:", num_genres)
```

```
Number of genres: 322
```

Grouping by genre and doing the descriptive analysis

[40]:
```
print(df.dtypes)
```

```
title                 object
studio                object
domestic_gross       float64
foreign_gross        float64
year                 float64
tconst                object
original_title        object
start_year           float64
runtime_minutes      float64
genres                object
averagerating        float64
numvotes             float64
id                   float64
release_date          object
```

```
movie                 object
production_budget     object
worldwide_gross       object
dtype: object
```

[41]:
```
# Remove non-numeric characters from 'production_budget' and 'worldwide_gross' columns
df['production_budget'] = df['production_budget'].astype(str).str.replace('$','').str.replace(',', '').astype(float)
df['worldwide_gross'] = df['worldwide_gross'].astype(str).str.replace('$', '').str.replace(',', '').astype(float)
```

[42]:
```
# Group by 'genres' and calculate various statistics
genre_stats = df.groupby('genres').agg({
    'averagerating': 'mean',                # Mean rating
    'domestic_gross': 'mean',               # Mean domestic gross income
    'foreign_gross': 'mean',                # Mean foreign gross
    'runtime_minutes': 'mean',              # Mean runtime
    'numvotes': 'mean',                     # Mean numvotes

})

print(genre_stats)
```

```
                           averagerating  domestic_gross  foreign_gross  \
genres
Action                          6.116667    1.032559e+07   3.917778e+07
Action,Adventure                5.866667    5.408333e+04   1.487450e+07
Action,Adventure,Animation      7.354545    9.930275e+07   1.989091e+08
Action,Adventure,Biography      7.000000    6.005725e+07   1.470250e+08
Action,Adventure,Comedy         6.271875    9.913976e+07   2.076031e+08
…                                     …               …              …
Romance,Thriller                5.850000    2.736500e+05   6.560500e+06
Sci-Fi                          5.050000    2.063390e+08   2.153000e+08
Sport                           7.900000    5.300000e+06   2.130000e+07
Thriller                        5.728000    2.097900e+07   3.998409e+07
Thriller,Western                6.400000    2.110000e+04   3.000000e+05

                           runtime_minutes       numvotes
genres
Action                          117.066667    6956.000000
Action,Adventure                113.666667    4892.333333
Action,Adventure,Animation      100.227273  124986.818182
Action,Adventure,Biography      128.250000  191598.000000
Action,Adventure,Comedy         111.093750  181259.937500
…                                      …             …
Romance,Thriller                108.500000   14547.000000
Sci-Fi                           74.500000    1760.500000
```

```
Sport                          114.000000      77.000000
Thriller                        97.227273    1191.280000
Thriller,Western                95.000000    7874.000000

[322 rows x 5 columns]
```

[43]:
```python
# Find the genre with the highest and lowest values for each category
highest_ratings = genre_stats['averagerating'].idxmax()
lowest_ratings = genre_stats['averagerating'].idxmin()

highest_domestic_gross = genre_stats['domestic_gross'].idxmax()
lowest_domestic_gross = genre_stats['domestic_gross'].idxmin()

highest_foreign_gross = genre_stats['foreign_gross'].idxmax()
lowest_foreign_gross = genre_stats['foreign_gross'].idxmin()

longest_runtime = genre_stats['runtime_minutes'].idxmax()
shortest_runtime = genre_stats['runtime_minutes'].idxmin()

most_numvotes = genre_stats['numvotes'].idxmax()
least_numvotes = genre_stats['numvotes'].idxmin()


# Print the results
print("Highest average rating genre:", highest_ratings)
print("Lowest average rating genre:", lowest_ratings)
print("Genre with highest foreign gross:", highest_foreign_gross)
print("Genre with lowest foreign gross:", lowest_foreign_gross)
print("Genre with longest runtime:", longest_runtime)
print("Genre with shortest runtime:", shortest_runtime)
print("Genre with most numvotes:", most_numvotes)
print("Genre with least numvotes:", least_numvotes)
```

```
Highest average rating genre: Adventure
Lowest average rating genre: Comedy,Thriller
Genre with highest foreign gross: Adventure,Drama,Sport
Genre with lowest foreign gross: Biography,Documentary,Thriller
Genre with longest runtime: Drama,History,Sport
Genre with shortest runtime: Action,Sport
Genre with most numvotes: Adventure,Drama,Sci-Fi
Genre with least numvotes: Documentary,Drama,Romance
```

[44]:
```python
# Find the top 5 and bottom 5 genres for each category
top_bottom_genres = {}

for column in genre_stats.columns:
    top_bottom_genres[column] = {
```

```
        'top_5': genre_stats[column].nlargest(5),
        'bottom_5': genre_stats[column].nsmallest(5)
    }

# Print the results
for category, values in top_bottom_genres.items():
    print(f"Category: {category}")
    print("Top 5 Genres:")
    print(values['top_5'])
    print("\nBottom 5 Genres:")
    print(values['bottom_5'])
    print("\n")
```

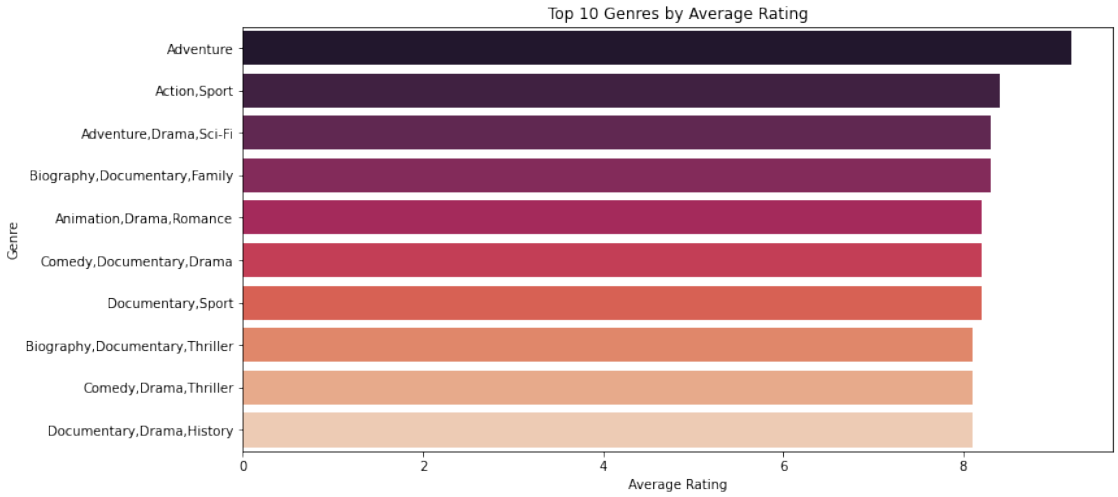
```
Category: averagerating
Top 5 Genres:
genres
Adventure                       9.2
Action,Sport                    8.4
Adventure,Drama,Sci-Fi          8.3
Biography,Documentary,Family    8.3
Animation,Drama,Romance         8.2
Name: averagerating, dtype: float64

Bottom 5 Genres:
genres
Comedy,Thriller          2.1
Comedy,Family,Sci-Fi     2.6
Action,Drama,Music       3.4
Drama,Mystery,Western    3.4
Fantasy,Horror           3.8
Name: averagerating, dtype: float64


Category: domestic_gross
Top 5 Genres:
genres
Adventure,Drama,Sport      4.007000e+08
Action,Adventure,Sci-Fi    2.345681e+08
Adventure,Drama,Sci-Fi     2.082000e+08
Documentary,Drama,Sport    2.067250e+08
Sci-Fi                     2.063390e+08
Name: domestic_gross, dtype: float64

Bottom 5 Genres:
genres
Comedy,Thriller           800.0
Fantasy,Thriller         1400.0
```

```
Action,Horror,Mystery    2800.0
Biography                4300.0
Comedy,Crime,History     4800.0
Name: domestic_gross, dtype: float64


Category: foreign_gross
Top 5 Genres:
genres
Adventure,Drama,Sport     8.757000e+08
Action,Comedy,Mystery     5.421000e+08
Adventure,Fantasy         5.111333e+08
Fantasy,Romance           4.585000e+08
Adventure,Drama,Sci-Fi    4.455500e+08
Name: foreign_gross, dtype: float64

Bottom 5 Genres:
genres
Biography,Documentary,Thriller    202000.0
Documentary,Drama,Mystery         242000.0
Thriller,Western                  300000.0
Animation,Drama,Sci-Fi            318000.0
Comedy,Mystery,Romance            421000.0
Name: foreign_gross, dtype: float64


Category: runtime_minutes
Top 5 Genres:
genres
Drama,History,Sport       151.0
Adventure,Drama,Sci-Fi    149.0
Action,Romance            146.0
Action,Comedy,Musical     145.0
Biography                 142.0
Name: runtime_minutes, dtype: float64

Bottom 5 Genres:
genres
Action,Sport                 58.000000
Adventure,Comedy,Horror      58.000000
Documentary,Drama,Romance    58.000000
Documentary,News             65.000000
Documentary,Drama,Family     70.333333
Name: runtime_minutes, dtype: float64


Category: numvotes
Top 5 Genres:
```

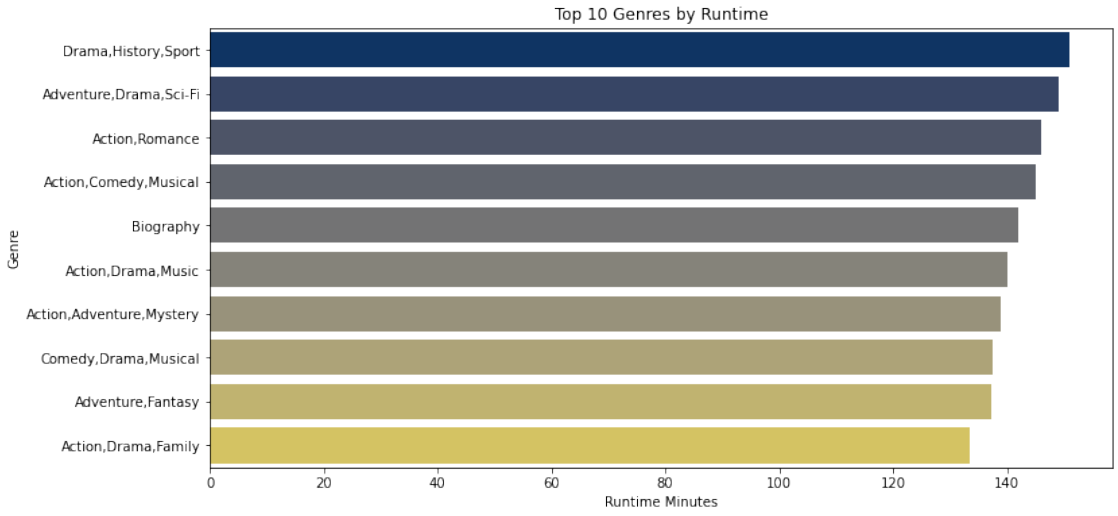
```
genres
Adventure,Drama,Sci-Fi       989725.000000
Adventure,Mystery,Sci-Fi     538720.000000
Action,Adventure,Sci-Fi      419616.851064
Mystery,Sci-Fi,Thriller      406532.500000
Action,Adventure,Mystery     399703.000000
Name: numvotes, dtype: float64

Bottom 5 Genres:
genres
Documentary,Drama,Romance        5.0
Action,Sport                     8.0
Family                          12.0
Mystery                         16.0
Biography,Documentary,Family    18.0
Name: numvotes, dtype: float64
```

1.2 DATA VISUALIZATION

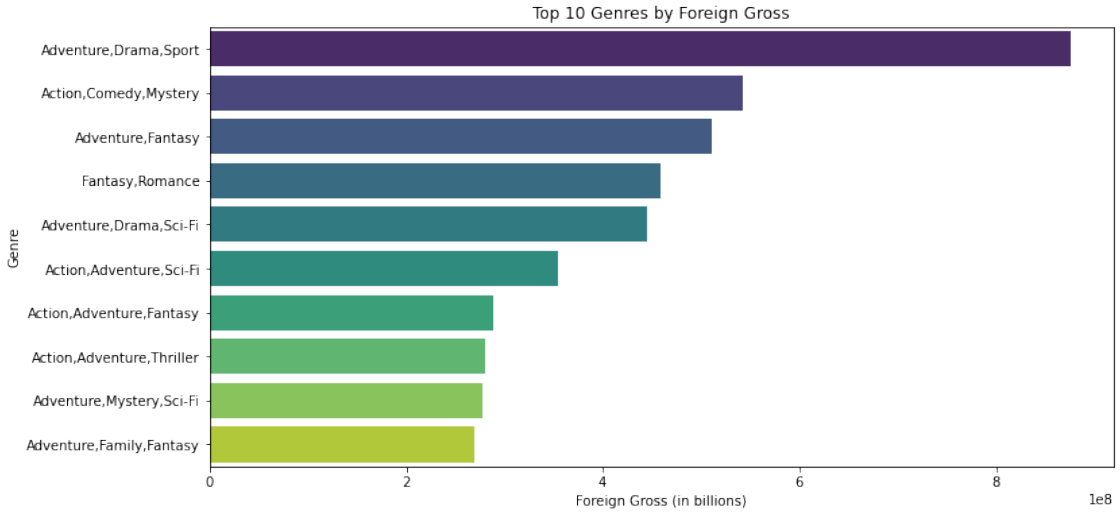
[45]:
```
# A bar plot for average rating
plt.figure(figsize=(12, 6))
sns.barplot(x=genre_stats['averagerating'].nlargest(10),␣
 ↪y=genre_stats['averagerating'].nlargest(10).index, palette='rocket')
plt.xlabel('Average Rating')
plt.ylabel('Genre')
plt.title('Top 10 Genres by Average Rating')
plt.show()
```



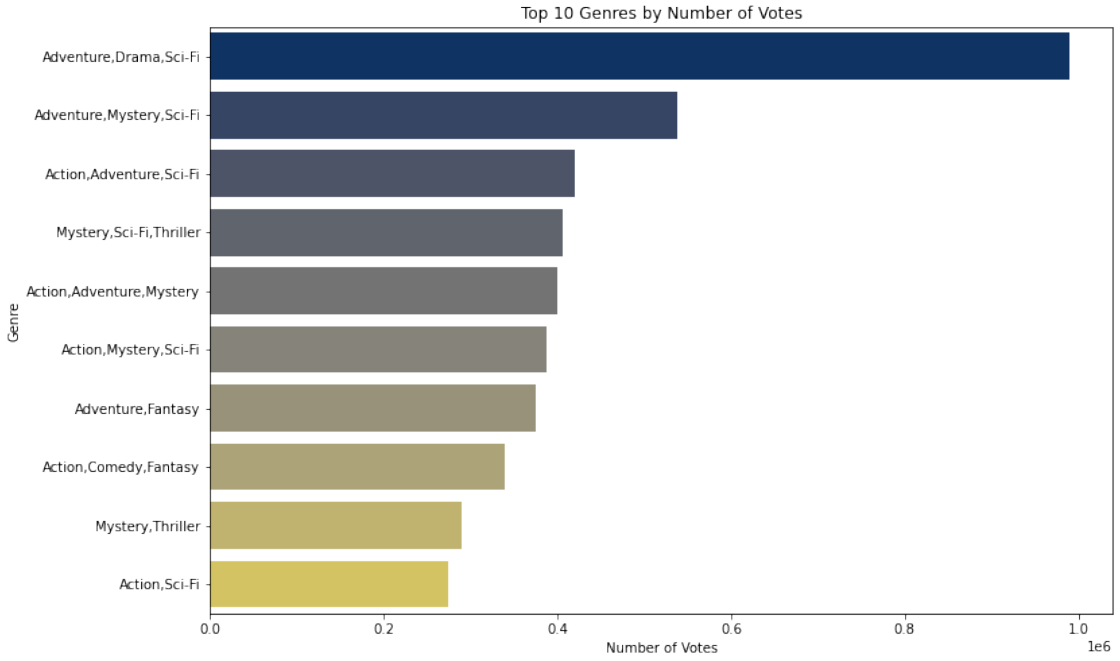
[46]:
```
# a # A bar plot for runtime minutes
plt.figure(figsize=(12, 6))
sns.barplot(x=genre_stats['runtime_minutes'].nlargest(10),␣
 ↪y=genre_stats['runtime_minutes'].nlargest(10).index, palette='cividis')
plt.xlabel('Runtime Minutes')
plt.ylabel('Genre')
plt.title('Top 10 Genres by Runtime')
plt.show()
```



[47]:
```
# A bar plot for foreign gross
plt.figure(figsize=(12, 6))
sns.barplot(x=genre_stats['foreign_gross'].nlargest(10),␣
 ↪y=genre_stats['foreign_gross'].nlargest(10).index, palette='viridis')
plt.xlabel('Foreign Gross (in billions)')
plt.ylabel('Genre')
plt.title('Top 10 Genres by Foreign Gross')
plt.show()
```



[48]:
```
# A bar plot for number of votes
plt.figure(figsize=(12, 8))
sns.barplot(x=genre_stats['numvotes'].nlargest(10), y=genre_stats['numvotes'].
 ↪nlargest(10).index, palette='cividis')
plt.xlabel('Number of Votes')
plt.ylabel('Genre')
plt.title('Top 10 Genres by Number of Votes')
plt.show()
```



[49]:
```
# Finding out the best performing movie overall
# Assign equal weight to each category
weight_rating = 0.25
weight_gross = 0.25
weight_runtime = 0.25
weight_votes = 0.25

# Calculate performance metric for each movie
df['Performance_Metric'] = (weight_rating * df['averagerating'] +
                            weight_gross * (df['domestic_gross'] +␣
 ↪df['foreign_gross']) / 2 +
                            weight_runtime * df['runtime_minutes'] +
                            weight_votes * df['numvotes'])

# Rank movies based on performance metric
ranked_movies = df.sort_values(by='Performance_Metric', ascending=False)

# Select the top-performing movie
best_performing_movie = ranked_movies.iloc[0]

# Print the best-performing movie
# Print the best-performing movie's performance metric
print("Best Performing Movie:")
print(best_performing_movie[['title', 'Performance_Metric']])
```

```
Best Performing Movie:
title                 Avengers: Age of Ultron
Performance_Metric                  1.75841e+08
Name: 1617, dtype: object
```

[50]:
```
# Group by genre and calculate the mean performance metric for each genre
genre_performance = df.groupby('genres')['Performance_Metric'].mean()

# Find the genre with the highest mean performance metric
best_genre = genre_performance.idxmax()
best_genre_performance = genre_performance.max()

# Print the best performing genre and its performance metric
print(f"Best Performing Genre: {best_genre}")
print(f"Performance Metric: {best_genre_performance}")
```

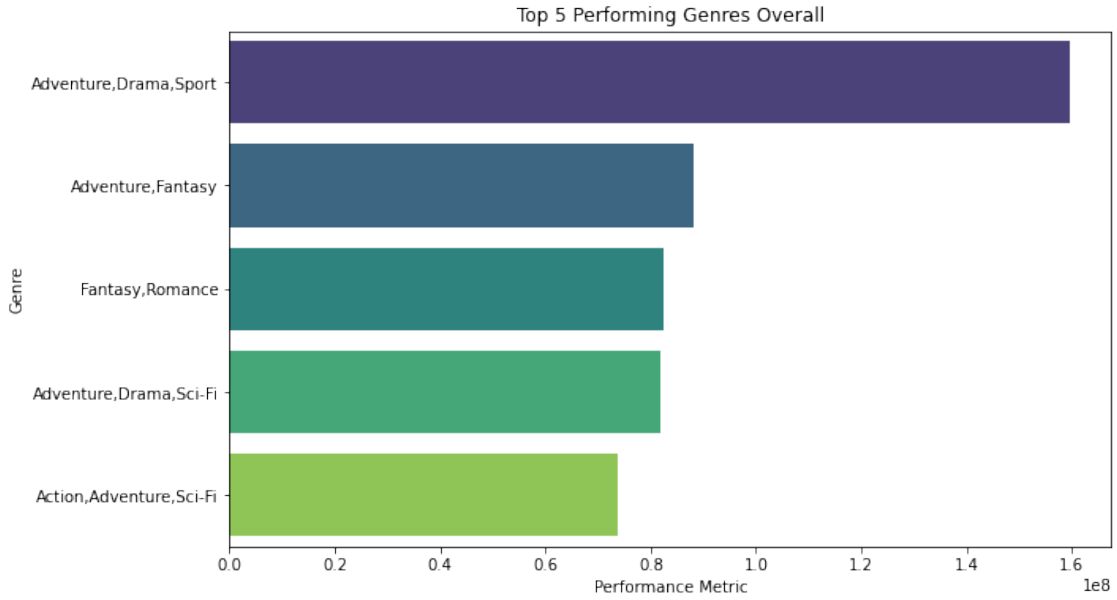
```
Best Performing Genre: Adventure,Drama,Sport
Performance Metric: 159565602.55
```

[51]:
```
#Finding the top 5 performing genres overall
# Sort the genres by their mean performance metric in descending order and␣
 ↪select the top 5
```

```
top_5_genres = genre_performance.nlargest(5)

# Print the top 5 performing genres and their metrics
print("Top 5 Performing Genres:")
for genre, metric in top_5_genres.items():
    print(f"Genre: {genre}, Performance Metric: {metric}")
```

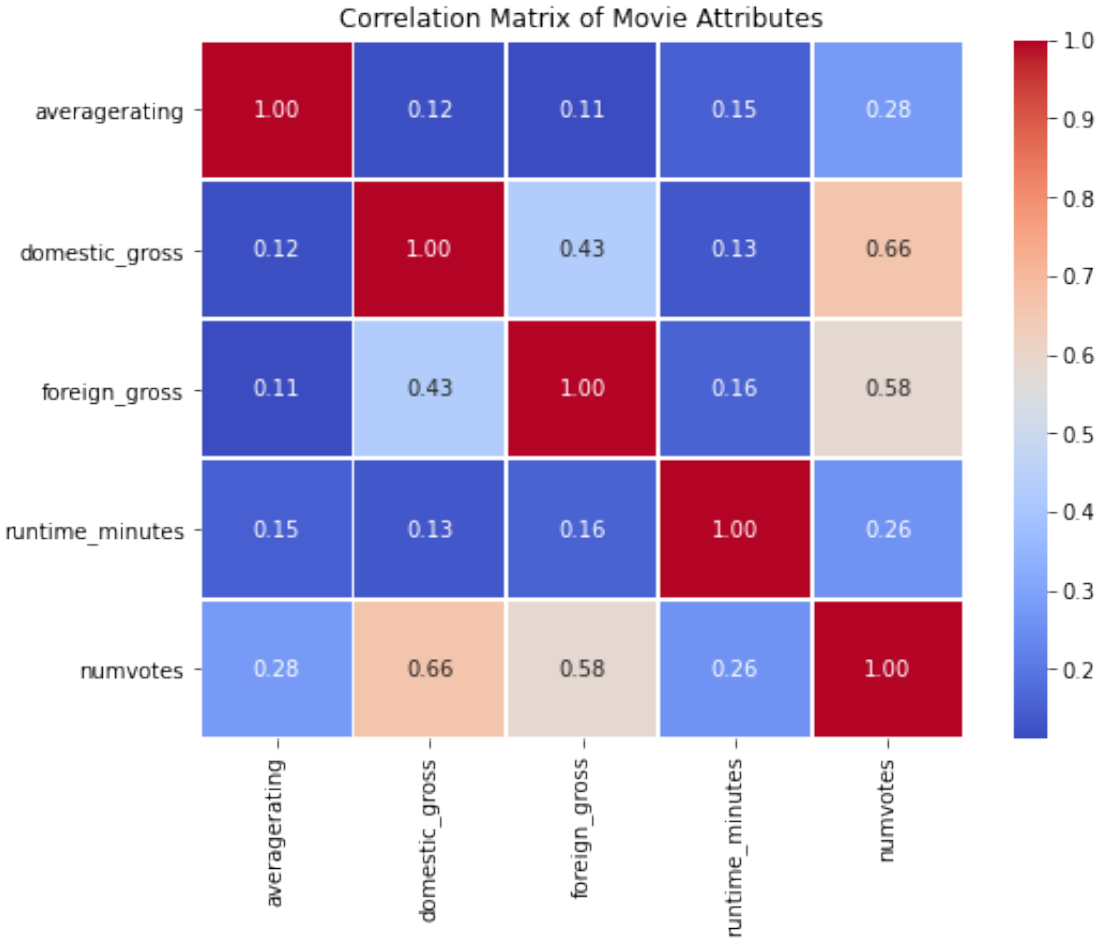
```
Top 5 Performing Genres:
Genre: Adventure,Drama,Sport, Performance Metric: 159565602.55
Genre: Adventure,Fantasy, Performance Metric: 88098145.33333333
Genre: Fantasy,Romance, Performance Metric: 82378432.35
Genre: Adventure,Drama,Sci-Fi, Performance Metric: 81966220.575
Genre: Action,Adventure,Sci-Fi, Performance Metric: 73775151.26595746
```

[52]:
```
# Create a bar plot for the top 5 performing genres
plt.figure(figsize=(10, 6))
sns.barplot(x=top_5_genres.values, y=top_5_genres.index, palette='viridis')
plt.xlabel('Performance Metric')
plt.ylabel('Genre')
plt.title('Top 5 Performing Genres Overall')
plt.show()
```



1.3 CORRELATION ANALYSIS

[53]:
```
# Select numerical columns for correlation analysis
numerical_cols = ['averagerating', 'domestic_gross', 'foreign_gross',␣
 ↪'runtime_minutes', 'numvotes']

# Calculate correlation matrix
correlation_matrix = df[numerical_cols].corr()

# Create a heatmap to visualize the correlation matrix
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f",␣
 ↪linewidths=1)
plt.title('Correlation Matrix of Movie Attributes')
plt.show()
```



2 DATA ANALYSIS SUMMARY

Number of genres was 322

Highest average rating genre: Adventure

Lowest average rating genre: Comedy,Thriller

Genre with highest domestic gross: Adventure,Drama,Sport

Genre with lowest domestic gross: Comedy,

Genre with highest foreign gross: Adventure,Drama,Sport

Genre with lowest foreign gross: Biography,Documentary,Thriller

Genre with longest runtime: Drama,History,

Genre with shortest runtime: Action,Sport

Genre with most numvotes: Adventure,Drama,Sci-Fi

Genre with least numvotes: Documentary,Drama,Romance

Best Performing Genre: Adventure,Drama,Sport

2.1 Insights

Microsoft shouldput its focus on adventure, drama and sport genres.

2.1.1 Recommendations

Implement strategies to continually monitor and adjust to evolving audience preferences and market trends in the film industry.