najma-abdi-student-1

February 18, 2024

0.1 Final Project Submission

Please fill out: * Student name: NAJMA ABDI * Student pace: part time * Scheduled project review date/time: 02/18/2024 * Instructor name: Noah Kandie * Blog post URL:

PROJECT TITLE CRAFTING MICROSOFT'S CINEMATIC SUCCESS STORY

The Business Problem revolves around Microsoft's lack of expertise in movie production and the desire to capitalize on the success of original video content.This Project aims to assist Microsoft in establishing a successful movie studio by analyzing current trends in the film industry to identify lucrative genres. Data from Box Office perfomance,audience demographics and critical reception will be gathered.The result will guide Microsoft's decision making process,focusing on investing in genres with high commercial success potential.

[ ]:
```
# Your code here - remember to use markdown cells for comments as well!
```

1. DATA INSPECTION

1.1 LOADING DATASET

BASICS CSV

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

[2]:
```
#loading data
t = r"C:
 ↪\Users\abdin\OneDrive\Desktop\DSF_P1P\dsc-phase-1-project\zippedData\imdb.
 ↪title.basics.csv.gz"
basics_dt = pd.read_csv(t)
#GETTING FIRST FIVE ROWS
basics_dt.head()
```

[2]:

	tconst	primary_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
```

[3]:
```
#summary of the data
basics_dt.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   primary_title    146143 non-null  object
 2   original_title   146122 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
```

[4]:
```
#accessing the last 5 rows
basics_dt.tail()
```

[4]:
```
           tconst                                primary_title  \
146139  tt9916538                            Kuambil Lagi Hatiku
146140  tt9916622  Rodolpho Teóphilo - O Legado de um Pioneiro
146141  tt9916706                                Dankyavar Danka
146142  tt9916730                                        6 Gunn
146143  tt9916754                Chico Albuquerque - Revelações

                                  original_title  start_year  \
146139                         Kuambil Lagi Hatiku        2019
146140  Rodolpho Teóphilo - O Legado de um Pioneiro        2015
146141                               Dankyavar Danka        2013
146142                                       6 Gunn        2017
146143               Chico Albuquerque - Revelações        2013

        runtime_minutes       genres
146139            123.0        Drama
146140              NaN  Documentary
```

```
146141              NaN       Comedy
146142            116.0          NaN
146143              NaN  Documentary
```

[5]:
```
#checking summary statistics
basics_dt.describe()
```

[5]:
```
          start_year  runtime_minutes
count  146144.000000    114405.000000
mean     2014.621798        86.187247
std         2.733583       166.360590
min      2010.000000         1.000000
25%      2012.000000        70.000000
50%      2015.000000        87.000000
75%      2017.000000        99.000000
max      2115.000000     51420.000000
```

[6]:
```
#finding number of rows & columns
basics_dt.shape
print("Number of rows",basics_dt.shape[0])
print("Number of columns",basics_dt.shape[1])
```

```
Number of rows 146144
Number of columns 6
```

[7]:
```
basics_dt.dtypes
```

[7]:
```
tconst             object
primary_title      object
original_title     object
start_year          int64
runtime_minutes   float64
genres             object
dtype: object
```

[ ]:
```

```

[ ]:
```
RATINGS CSV
```

[31]:
```
#importing and reading the csv
k = r"C:
 ↪\Users\abdin\OneDrive\Desktop\DSF_P1P\dsc-phase-1-project\zippedData\imdb.
 ↪title.ratings.csv.gz"
ratings_data = pd.read_csv(k)
ratings_data.head()
```

[31]:

	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

[12]:
```
#summary of the dataframe
ratings_data.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
```

[13]:
```
#checking the last 5 rows
ratings_data.tail()
```

[13]:

	tconst	averagerating	numvotes
73851	tt9805820	8.1	25
73852	tt9844256	7.5	24
73853	tt9851050	4.7	14
73854	tt9886934	7.0	5
73855	tt9894098	6.3	128

[14]:
```
#summary statistics
ratings_data.describe()
```

[14]:

	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

[15]:
```
#finding number of rows & columns
ratings_data.shape
print("Number of rows",basics_dt.shape[0])
```

```
print("Number of columns",basics_dt.shape[1])
```

```
Number of rows 146144
Number of columns 6
```

[16]:
```
ratings_data.dtypes
```

[16]:
```
tconst            object
averagerating    float64
numvotes           int64
dtype: object
```

[17]:
```
ratings_data.columns
```

[17]:
```
Index(['tconst', 'averagerating', 'numvotes'], dtype='object')
```

[ ]:
```

```

BOM.MOVIE GROSS CSV

[8]:
```
#loading the boom.movie gross csv
f = r"C:
 ↪\Users\abdin\OneDrive\Desktop\DSF_P1P\dsc-phase-1-project\zippedData\bom.
 ↪movie_gross.csv\bom.movie_gross.csv"
movie_gross_dt = pd.read_csv(f)
#first five rows
movie_gross_dt.head()
```

[8]:

	title	studio	domestic_gross	foreign_gross	year
0	Toy Story 3	BV	415000000.0	652000000	2010
1	Alice in Wonderland (2010)	BV	334200000.0	691300000	2010
2	Harry Potter and the Deathly Hallows Part 1	WB	296000000.0	664300000	2010
3	Inception	WB	292600000.0	535700000	2010
4	Shrek Forever After	P/DW	238700000.0	513900000	2010

[9]:
```
#summary information on metadata
movie_gross_dt.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
```

[10]:
```
#last 5 rows
movie_gross_dt.tail()
```

[10]:

	title	studio	domestic_gross	foreign_gross	year
3382	The Quake	Magn.	6200.0	NaN	2018
3383	Edward II (2018 re-release)	FM	4800.0	NaN	2018
3384	El Pacto	Sony	2500.0	NaN	2018
3385	The Swan	Synergetic	2400.0	NaN	2018
3386	An Actor Prepares	Grav.	1700.0	NaN	2018

[11]:
```
movie_gross_dt.shape
print("Number of rows",movie_gross_dt.shape[0])
print("Number of columns",movie_gross_dt.shape[1])
```

```
Number of rows 3387
Number of columns 5
```

[12]:
```
#summary statistics
movie_gross_dt.describe()
```

[12]:

	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

[13]:
```
movie_gross_dt.columns
```

[13]:
```
Index(['title', 'studio', 'domestic_gross', 'foreign_gross', 'year'],
dtype='object')
```

[ ]:

TMDB MOVIES CSV

[14]:
```
d = r"C:
↪\Users\abdin\OneDrive\Desktop\DSF_P1P\dsc-phase-1-project\zippedData\tmdb.
↪movies.csv.gz"
db_movies = pd.read_csv(d)
#check first 5 rows
db_movies.head()
```

[14]:

	Unnamed: 0	genre_ids	id	original_language	original_title	popularity	release_date	title	vote_average	vote_count
0	0	[12, 14, 10751]	12444	en	Harry Potter and the Deathly Hallows: Part 1	33.533	2010-11-19	Harry Potter and the Deathly Hallows: Part 1	7.7	10788
1	1	[14, 12, 16, 10751]	10191	en	How to Train Your Dragon	28.734	2010-03-26	How to Train Your Dragon	7.7	7610
2	2	[12, 28, 878]	10138	en	Iron Man 2	28.515	2010-05-07	Iron Man 2	6.8	12368
3	3	[16, 35, 10751]	862	en	Toy Story	28.005	1995-11-22	Toy Story	7.9	10174
4	4	[28, 878, 12]	27205	en	Inception	27.920	2010-07-16	Inception	8.3	22186

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26517 entries, 0 to 26516
Data columns (total 10 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Unnamed: 0         26517 non-null  int64
 1   genre_ids          26517 non-null  object
 2   id                 26517 non-null  int64
 3   original_language  26517 non-null  object
 4   original_title     26517 non-null  object
 5   popularity         26517 non-null  float64
```

```
 6   release_date       26517 non-null  object
 7   title              26517 non-null  object
 8   vote_average       26517 non-null  float64
 9   vote_count         26517 non-null  int64
dtypes: float64(2), int64(3), object(5)
memory usage: 2.0+ MB
```

[16]:
```
#check for summary statistics
db_movies.describe()
```

[16]:

	Unnamed: 0	id	popularity	vote_average	vote_count
count	26517.00000	26517.000000	26517.000000	26517.000000	26517.000000
mean	13258.00000	295050.153260	3.130912	5.991281	194.224837
std	7654.94288	153661.615648	4.355229	1.852946	960.961095
min	0.00000	27.000000	0.600000	0.000000	1.000000
25%	6629.00000	157851.000000	0.600000	5.000000	2.000000
50%	13258.00000	309581.000000	1.374000	6.000000	5.000000
75%	19887.00000	419542.000000	3.694000	7.000000	28.000000
max	26516.00000	608444.000000	80.773000	10.000000	22186.000000

[17]:
```
#check for data types
db_movies.dtypes
```

[17]:
```
Unnamed: 0            int64
genre_ids            object
id                    int64
original_language    object
original_title       object
popularity          float64
release_date         object
title                object
vote_average        float64
vote_count            int64
dtype: object
```

[18]:
```
#check for rows & columns
db_movies.shape
```

[18]:
```
(26517, 10)
```

[ ]:
```

```

2.DATA CLEANING

HANDLING MISSING VALUES IN EACH DATASET

1.0 TITLE BASICS CSV

[19]:
```
#Checking out for missing values
basics_dt.isnull().sum()
```

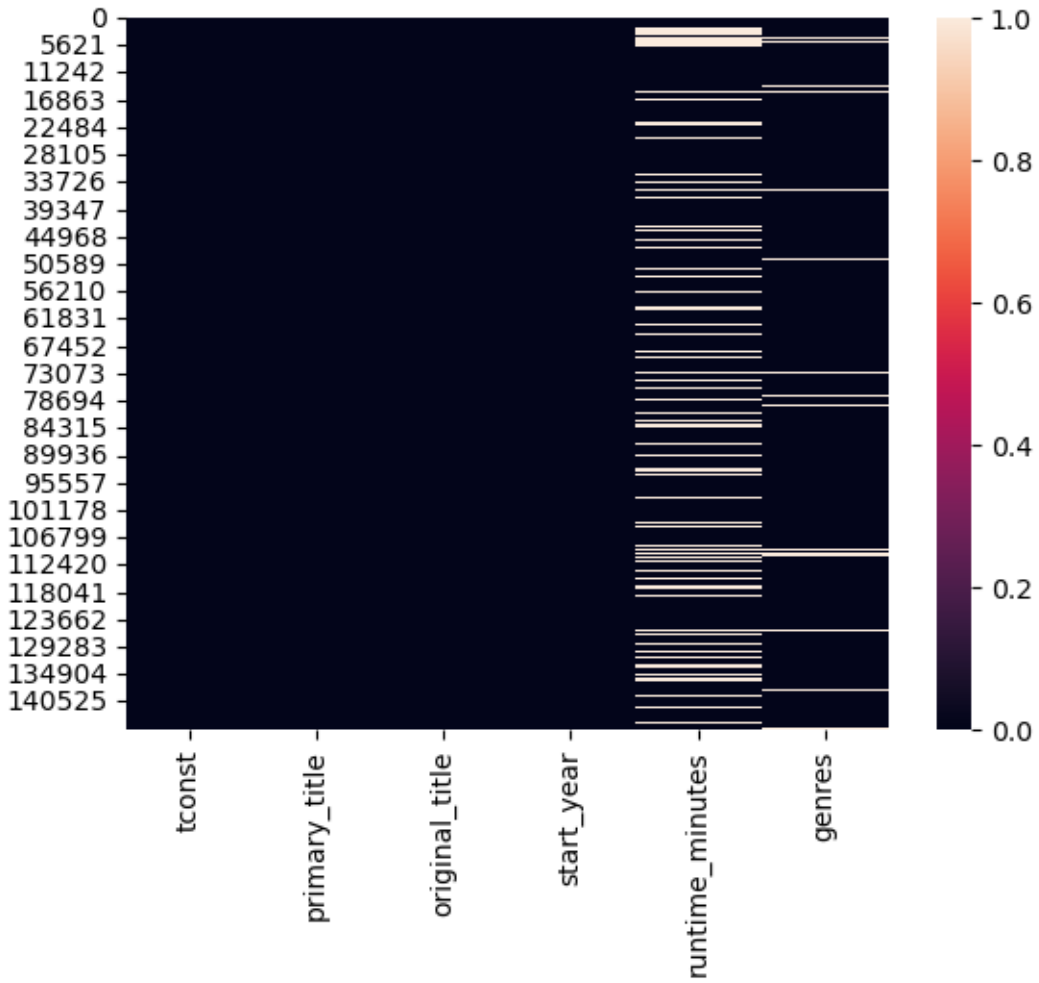
[19]:
```
tconst                0
primary_title         1
original_title       22
start_year            0
runtime_minutes    31739
genres              5408
dtype: int64
```

[20]:
```
basics_dt.isnull().mean()
```

[20]:
```
tconst             0.000000
primary_title      0.000007
original_title     0.000151
start_year         0.000000
runtime_minutes    0.217176
genres             0.037005
dtype: float64
```

[21]:
```
#visualize missing values in basics_dt csv
sns.heatmap(basics_dt.isnull())
```

[21]:
```
<Axes: >
```



[22]:
```
#genres and original_title missing values are few hence drop them without␣
↪causing any effect on data
null_genres = basics_dt.genres.isna()
basics_dt = basics_dt[~null_genres]
```

[23]:
```
null_title = basics_dt.original_title.isna()
basics_dt = basics_dt[~null_title]
```

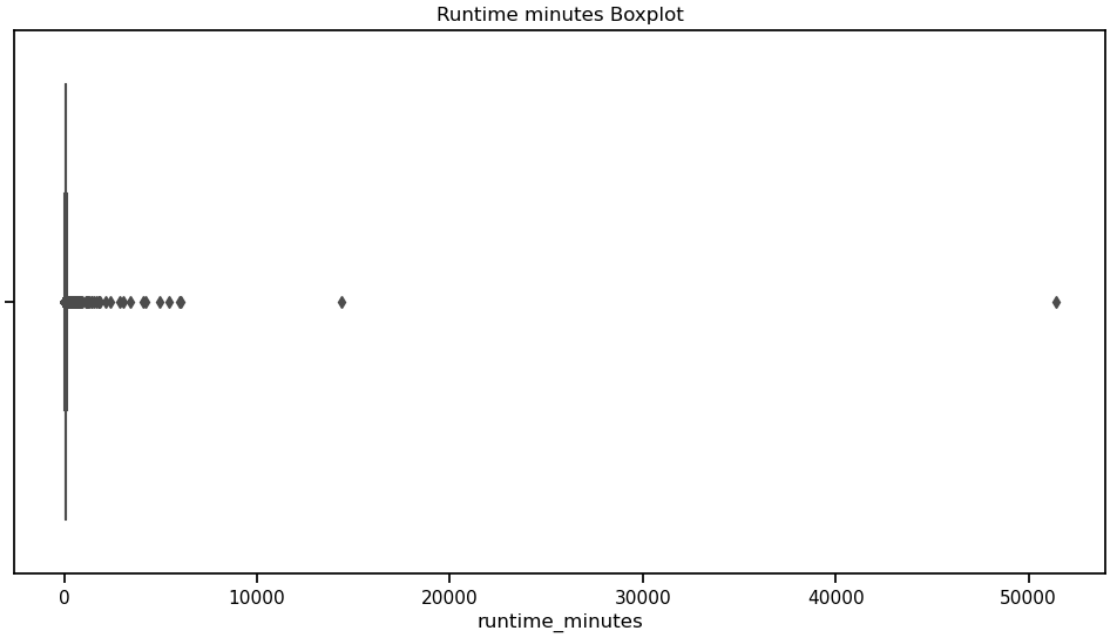
[24]:
```
#Confirming whether null values in original_title and genres columns have all␣
↪dropped
basics_dt.isnull().sum()
```

[24]:
```
tconst              0
primary_title       0
original_title      0
start_year          0
```

```
runtime_minutes    28501
genres                 0
dtype: int64
```

[26]:
```
#checking for outliers and handling them
#Visualization of runtime minutes

runtimes = basics_dt.runtime_minutes

#finding max and mim runtime
min_runtime = runtimes.min()
max_runtime = runtimes.max()
mean_runtime = runtimes.mean()
print(f"minimum runtime: {min_runtime}")
print(f"Maximum runtime: {max_runtime}")
print(f"Mean runtime: {mean_runtime}")

#Choosing boxplot column
col_data = basics_dt.runtime_minutes

# Creating boxplot
plt.figure(figsize=(12,6))
sns.set_context('notebook')
sns.boxplot(x= col_data, color= "yellow")
plt.title('Runtime minutes Boxplot');
```

```
minimum runtime: 1.0
Maximum runtime: 51420.0
Mean runtime: 86.26155641884668
```



[27]:
```
#Checking for the highest runtime
basics_dt.loc[basics_dt.runtime_minutes == max_runtime]
```

[27]:
```
           tconst primary_title original_title  start_year  runtime_minutes  \
132389  tt8273150     Logistics      Logistics        2012          51420.0

             genres
132389  Documentary
```

[28]:
```
#replacing the missing values with  median runtime
basics_dt.runtime_minutes.fillna(basics_dt.runtime_minutes.median(),␣
 ↪inplace=True)
```

[29]:
```
basics_dt.isnull().sum()
```

[29]:
```
tconst             0
primary_title      0
original_title     0
start_year         0
runtime_minutes    0
genres             0
dtype: int64
```

[ ]:
```

```

1.1 TITLE RATINGS CSV

[32]:
```
#Checking out for missing values
ratings_data.isnull().sum()
```

[32]:
```
tconst           0
averagerating    0
numvotes         0
dtype: int64
```

1.2 BOM GROSS MOVIES CSV

[33]:
```
#Checking out for missing values
movie_gross_dt.isnull().sum()
```

[33]:
```
title              0
studio             5
domestic_gross    28
foreign_gross   1350
year               0
dtype: int64
```

[34]:
```
#Filtering dataframe to remove null values
null_studio = movie_gross_dt.studio.isna()
movie_gross_dt = movie_gross_dt[~null_studio]
```

[35]:
```
#checking for mean null values
movie_gross_dt.isnull().mean()
```

[35]:
```
title             0.000000
studio            0.000000
domestic_gross    0.007688
foreign_gross     0.398876
year              0.000000
dtype: float64
```

[36]:
```
#removing unwanted characters like commas etc
movie_gross_dt.foreign_gross.replace(',','', inplace=True, regex=True)
```

[37]:
```
#checking for data type
movie_gross_dt.dtypes
```

[37]:
```
title              object
studio             object
domestic_gross    float64
foreign_gross      object
year                int64
dtype: object
```

[38]:
```
#convert foreign_gross data type to float
#column_name = 'foreign_gross'

#[column_name] = pd.to_numeric(movie_gross_dt[column_name], errors='coerce')

movie_gross_dt.foreign_gross = movie_gross_dt.foreign_gross.astype('float64')
```

[39]:
```
movie_gross_dt.dtypes
```

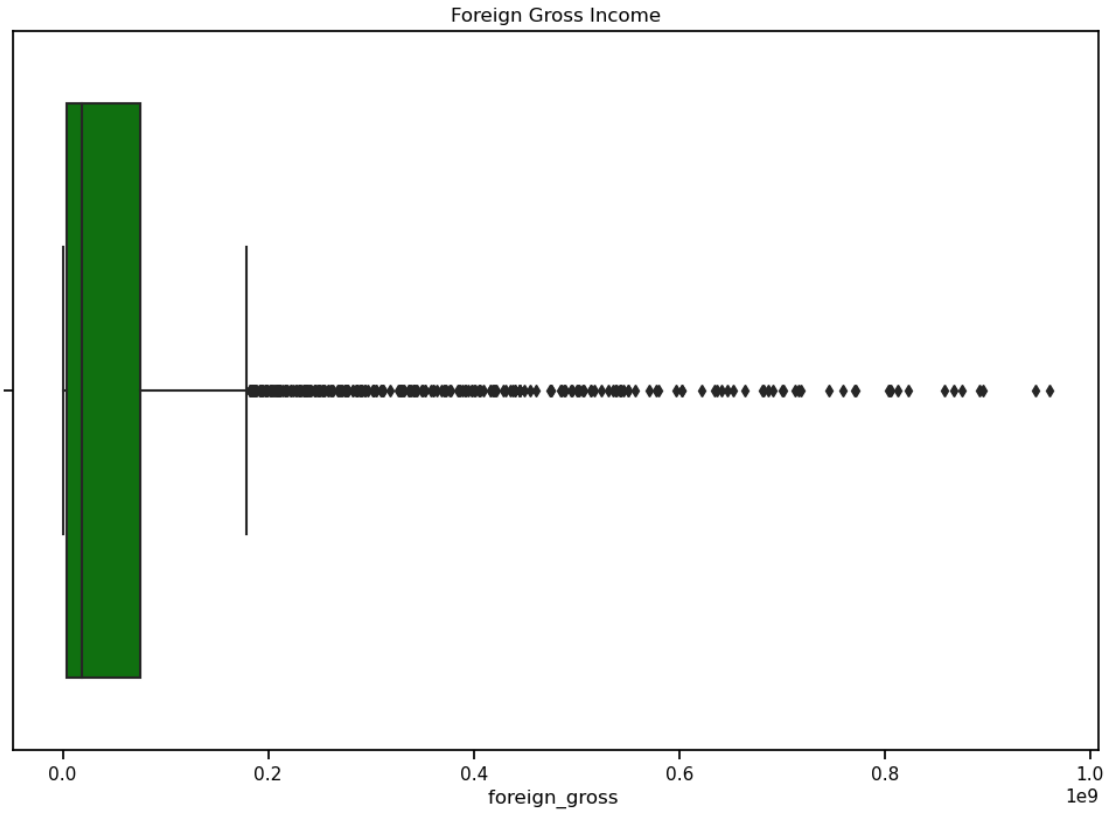
[39]:
```
title              object
studio             object
domestic_gross    float64
foreign_gross     float64
year                int64
dtype: object
```

[40]:
```
#checking for outliers and handling them
#Visualization of movie_gross dt foreign gross
income_foreign = movie_gross_dt.foreign_gross
# find minimum and maximum values in foreign gross column
```

```
min_foreign = income_foreign.min()
max_foreign = income_foreign.max()
mean_foreign = income_foreign.mean()
print(f"minimum foreign: {min_foreign}")
print(f"Maximum foreign: {max_foreign}")
print(f"Mean foreign: {mean_foreign}")
# selecting the column for the boxplot
col_data= movie_gross_dt.foreign_gross
# Creating boxplot using Seaborn Library
plt.figure(figsize= (12,8))
sns.boxplot(x= col_data, color= "green")
plt.title('Foreign Gross Income')
plt.xlabel('foreign_gross ')
```

```
minimum foreign: 600.0
Maximum foreign: 960500000.0
Mean foreign: 74954901.2673389
```

[40]: Text(0.5, 0, 'foreign_gross ')

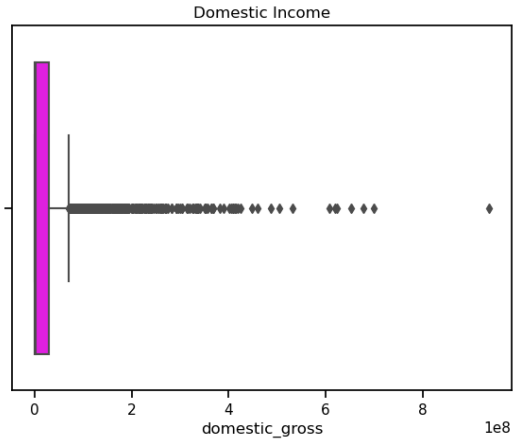


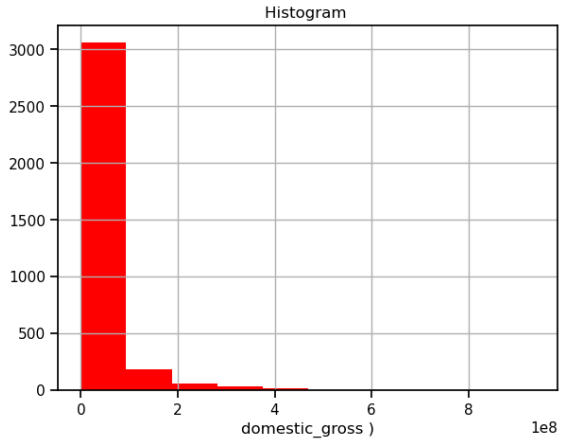
[41]:
```
#filling the missing value with median
movie_gross_dt.foreign_gross.fillna(movie_gross_dt.foreign_gross.isna().median,␣
↪inplace=True)
movie_gross_dt.isnull().sum()
```

[41]:
```
title              0
studio             0
domestic_gross    26
foreign_gross      0
year               0
dtype: int64
```

[42]:
```
#Visualize distribution of domestic_gross
income_domestic = movie_gross_dt.domestic_gross

#extract the min and max values in our domestic gross column
min_domestic = income_domestic.min()
max_domestic = income_domestic.max()
mean_domestic = income_domestic.mean()
print(f"minimum domestic: {min_domestic}")
print(f"Maximum domestic: {max_domestic}")
print(f"Mean domestic: {mean_domestic}")

# selecting the column for the boxplot
col_data = movie_gross_dt.domestic_gross

# Creating boxplot & histogram using Seaborn Library
fig, ax = plt.subplots(ncols=2, nrows=1, figsize= (15,5))
sns.boxplot(x=col_data, ax=ax[0], color='magenta')
ax[0].set_title('Domestic Income')
ax[0].set_xlabel('domestic_gross')
movie_gross_dt.domestic_gross.hist(ax=ax[1], color='red')
ax[1].set_title('Histogram ')
ax[1].set_xlabel('domestic_gross )')
plt.tight_layout;
```

```
minimum domestic: 100.0
Maximum domestic: 936700000.0
Mean domestic: 28771489.56495828
```





[43]:
```
#Replacing the missing values with Median
movie_gross_dt.domestic_gross.fillna(movie_gross_dt.domestic_gross.isna().
 ↪median(), inplace=True)
movie_gross_dt.isnull().sum()
```

[43]:
```
title             0
studio            0
domestic_gross    0
foreign_gross     0
year              0
dtype: int64
```

[ ]:
```

```

1.3 DB MOVIES CSV

[44]:
```
#Checking out for missing values
db_movies.isnull().sum()
```

[44]:
```
Unnamed: 0          0
genre_ids           0
id                  0
original_language   0
original_title      0
popularity          0
release_date        0
title               0
vote_average        0
vote_count          0
dtype: int64
```

[45]:
```
#checking for leading & trailing whitespaces and removing them in all datasets
[col.strip() for col in basics_dt.columns]
```

[45]:
```
['tconst',
 'primary_title',
 'original_title',
 'start_year',
 'runtime_minutes',
 'genres']
```

[46]:
```
[col.strip() for col in ratings_data.columns]
```

[46]:
```
['tconst', 'averagerating', 'numvotes']
```

[47]:
```
[col.strip() for col in movie_gross_dt.columns]
```

[47]:
```
['title', 'studio', 'domestic_gross', 'foreign_gross', 'year']
```

[48]:
```
[col.strip() for col in db_movies.columns]
```

[48]:
```
['Unnamed: 0',
 'genre_ids',
 'id',
 'original_language',
 'original_title',
 'popularity',
 'release_date',
 'title',
 'vote_average',
 'vote_count']
```

[49]:
```
#checking for duplicates in all dataset
```

[50]:
```
basics_dt.duplicated()
```

[50]:
```
0         False
1         False
2         False
3         False
4         False
          …
146138    False
146139    False
146140    False
146141    False
146143    False
Length: 140733, dtype: bool
```

[51]:
```
ratings_data.duplicated()
```

[51]:
```
0        False
1        False
2        False
3        False
4        False
         …
73851    False
73852    False
73853    False
73854    False
73855    False
Length: 73856, dtype: bool
```

[52]:
```
db_movies.duplicated()
```

[52]:
```
0        False
1        False
2        False
3        False
4        False
         …
26512    False
26513    False
26514    False
26515    False
26516    False
Length: 26517, dtype: bool
```

[53]:
```
movie_gross_dt.duplicated()
```

[53]:
```
0       False
1       False
2       False
3       False
4       False
        …
3382    False
3383    False
3384    False
3385    False
3386    False
Length: 3382, dtype: bool
```

3.FEATURE ENGINEERING

[54]:
```
#Merging my datasets in a single dataset
final_data = movie_gross_dt.merge(basics_dt, left_on='title',␣
 ↪right_on='original_title', how='left')
final_data = final_data.merge(ratings_data, left_on='tconst',␣
 ↪right_on='tconst', how='left')
final_data.head()
```

[54]:

	title	studio	domestic_gross	foreign_gross	year	tconst	primary_title	original_title	start_year	runtime_minutes	genres	averagerating	numvotes
0	Toy Story 3	BV	415000000.0	652000000.0	2010	tt0435761	Toy Story 3	Toy Story 3	2010.0	103.0	Adventure,Animation,Comedy	8.3	682218.0
1	Alice in Wonderland (2010)	BV	334200000.0	691300000.0	2010	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	Harry Potter and the Deathly Hallows Part 1	WB	296000000.0	664300000.0	2010	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	Inception	WB	292600000.0	535700000.0	2010	tt1375666	Inception	Inception	2010.0	148.0	Action,Adventure,Sci-Fi	8.8	1841066.0
4	Shrek Forever After	P/DW	238700000.0	513900000.0	2010	tt0892791	Shrek Forever After	Shrek Forever After	2010.0	93.0	Adventure,Animation,Comedy	6.3	167532.0

[55]:
```
final_data.shape
print("Number of rows",final_data.shape[0])
print("Number of columns",final_data.shape[1])
```

```
Number of rows 3960
Number of columns 13
```

[56]:
```
#check for missing values
final_data.isnull().mean()
```

[56]:
```
title             0.000000
studio            0.000000
```

```
domestic_gross     0.000000
foreign_gross      0.000000
year               0.000000
tconst             0.308838
primary_title      0.308838
original_title     0.308838
start_year         0.308838
runtime_minutes    0.308838
genres             0.308838
averagerating      0.383838
numvotes           0.383838
dtype: float64
```

[57]:
```
#Accessing Columns that fit analysis and reassigning to db movies dataset
db_movies = db_movies.loc[:, ['original_title', 'vote_average', 'vote_count',␣
 ↪'release_date']]
```

[58]:
```
#Checking whether the needed data is successfully extracted
db_movies.head()
```

[58]:
```
                                 original_title  vote_average  vote_count  \
0  Harry Potter and the Deathly Hallows: Part 1           7.7       10788
1                      How to Train Your Dragon           7.7        7610
2                                     Iron Man 2           6.8       12368
3                                      Toy Story           7.9       10174
4                                      Inception           8.3       22186

  release_date
0   2010-11-19
1   2010-03-26
2   2010-05-07
3   1995-11-22
4   2010-07-16
```

[59]:
```
# Convert 'release_date' column to date_time if it's not already in datetime␣
 ↪format
db_movies['release_date'] = pd.to_datetime(db_movies['release_date'])

# Extract the year from 'release_date' and create a new column 'release_year'
db_movies['release_year'] = db_movies['release_date'].dt.year

# Drop the 'release_date' column to remain with year
db_movies.drop('release_date', axis=1, inplace=True)

# # Display the first few rows of the modified DataFrame
print(db_movies.head())
```

```
                                 original_title  vote_average  vote_count  \
0  Harry Potter and the Deathly Hallows: Part 1           7.7       10788
1                      How to Train Your Dragon           7.7        7610
2                                  Iron Man 2           6.8       12368
3                                   Toy Story           7.9       10174
4                                   Inception           8.3       22186

   release_year
0          2010
1          2010
2          2010
3          1995
4          2010
```

[60]:
```
#retain only movies that appear in both tables using inner join
final_data = final_data.merge(db_movies, left_on= 'title', right_on =␣
 ↪'original_title', how="inner")
final_data.head()
```

[60]:
```
                      title studio  domestic_gross foreign_gross  year  \
0                 Toy Story 3     BV     415000000.0   652000000.0  2010
1                   Inception     WB     292600000.0   535700000.0  2010
2         Shrek Forever After   P/DW     238700000.0   513900000.0  2010
3  The Twilight Saga: Eclipse   Sum.     300500000.0   398000000.0  2010
4                  Iron Man 2   Par.     312400000.0   311500000.0  2010

      tconst               primary_title            original_title_x  \
0  tt0435761                Toy Story 3                 Toy Story 3
1  tt1375666                  Inception                   Inception
2  tt0892791         Shrek Forever After         Shrek Forever After
3  tt1325004  The Twilight Saga: Eclipse  The Twilight Saga: Eclipse
4  tt1228705                  Iron Man 2                  Iron Man 2

   start_year  runtime_minutes                      genres  averagerating  \
0      2010.0            103.0  Adventure,Animation,Comedy            8.3
1      2010.0            148.0     Action,Adventure,Sci-Fi            8.8
2      2010.0             93.0  Adventure,Animation,Comedy            6.3
3      2010.0            124.0     Adventure,Drama,Fantasy            5.0
4      2010.0            124.0     Action,Adventure,Sci-Fi            7.0

    numvotes            original_title_y  vote_average  vote_count  \
0   682218.0                 Toy Story 3           7.7        8340
1  1841066.0                   Inception           8.3       22186
2   167532.0         Shrek Forever After           6.1        3843
3   211733.0  The Twilight Saga: Eclipse           6.0        4909
4   657690.0                  Iron Man 2           6.8       12368
```

```
   release_year
0          2010
1          2010
2          2010
3          2010
4          2010
```

[61]:
```
final_data.columns
```

[61]:
```
Index(['title', 'studio', 'domestic_gross', 'foreign_gross', 'year', 'tconst',
       'primary_title', 'original_title_x', 'start_year', 'runtime_minutes',
       'genres', 'averagerating', 'numvotes', 'original_title_y',
       'vote_average', 'vote_count', 'release_year'],
      dtype='object')
```

[62]:
```
final_data.isnull().mean()
```

[62]:
```
title               0.000000
studio              0.000000
domestic_gross      0.000000
foreign_gross       0.000000
year                0.000000
tconst              0.026863
primary_title       0.026863
original_title_x    0.026863
start_year          0.026863
runtime_minutes     0.026863
genres              0.026863
averagerating       0.141669
numvotes            0.141669
original_title_y    0.000000
vote_average        0.000000
vote_count          0.000000
release_year        0.000000
dtype: float64
```

[63]:
```
# dropping the column start_year
final_data.drop('start_year', inplace=True, axis=1)

#Filling missing values in averagerating column with vote_ average column values
final_data.averagerating.fillna(final_data.vote_average, inplace=True)

#dropping vote_average columns
final_data.drop('vote_average', axis=1, inplace=True)

#dropping vote_count columns and refilling the null values in num_votes with␣
 ↪median
```

```
final_data.drop('vote_count', inplace=True, axis=1)

#checking whether the vote_count colum has dropped successfully
final_data.head()
```

[63]:

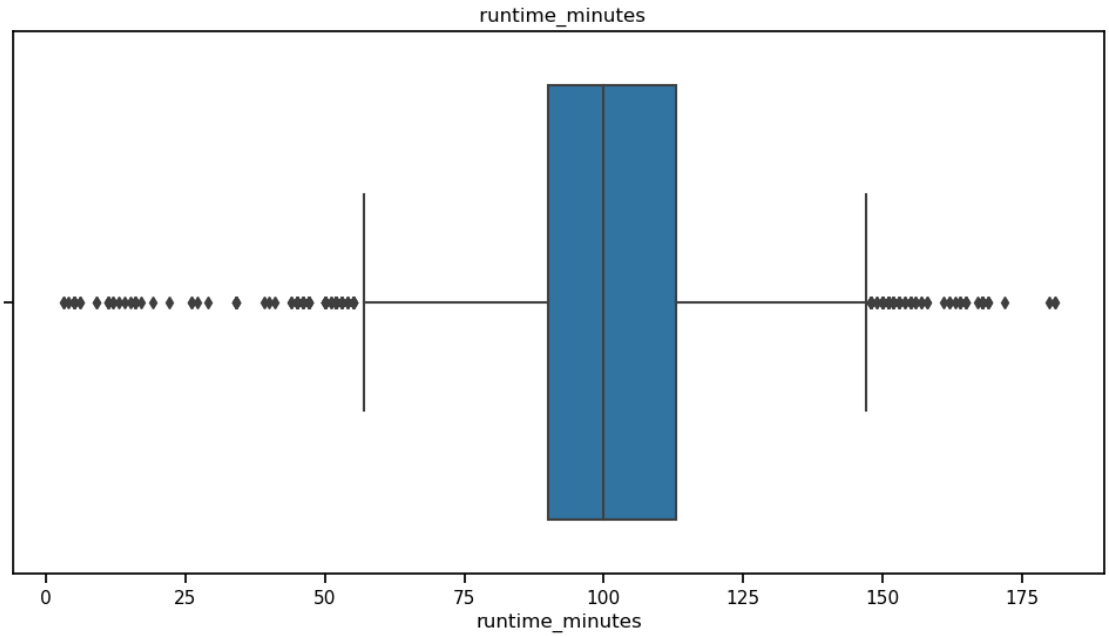
	title	studio	domestic_gross	foreign_gross	year	tconst	primary_title	original_title_x	runtime_minutes	genres	averagerating	numvotes	original_title_y	release_year
0	Toy Story 3	BV	415000000.0	652000000.0	2010	tt0435761	Toy Story 3	Toy Story 3	103.0	Adventure,Animation,Comedy	8.3	682218.0	Toy Story 3	2010
1	Inception	WB	292600000.0	535700000.0	2010	tt1375666	Inception	Inception	148.0	Action,Adventure,Sci-Fi	8.8	1841066.0	Inception	2010
2	Shrek Forever After	P/DW	238700000.0	513900000.0	2010	tt0892791	Shrek Forever After	Shrek Forever After	93.0	Adventure,Animation,Comedy	6.3	167532.0	Shrek Forever After	2010
3	The Twilight Saga: Eclipse	Sum.	300500000.0	398000000.0	2010	tt1325004	The Twilight Saga: Eclipse	The Twilight Saga: Eclipse	124.0	Adventure,Drama,Fantasy	5.0	211733.0	The Twilight Saga: Eclipse	2010
4	Iron Man 2	Par.	312400000.0	311500000.0	2010	tt1228705	Iron Man 2	Iron Man 2	124.0	Action,Adventure,Sci-Fi	7.0	657690.0	Iron Man 2	2010

[65]:
```
final_data.isnull().mean()
```

[65]:
```
title              0.000000
studio             0.000000
domestic_gross     0.000000
foreign_gross      0.000000
year               0.000000
tconst             0.026863
primary_title      0.026863
original_title_x   0.026863
runtime_minutes    0.026863
genres             0.026863
averagerating      0.000000
```

```
numvotes            0.141669
original_title_y    0.000000
release_year        0.000000
dtype: float64
```

[66]:
```
#Visualizing the distribution in final dataframe for runtime_mintues
col_data = final_data.runtime_minutes
plt.figure(figsize=(12,6))
sns.boxplot(x=col_data)
plt.title(' runtime_minutes');
```



[67]:
```
#from the visualization the data contains outliers
#lets use imputation method to replace the missing values

final_data.runtime_minutes.fillna(final_data.runtime_minutes.
 ↪mean(),inplace=True)
```

[68]:
```
#now lets check if we succeeded
final_data.isnull().sum()
```

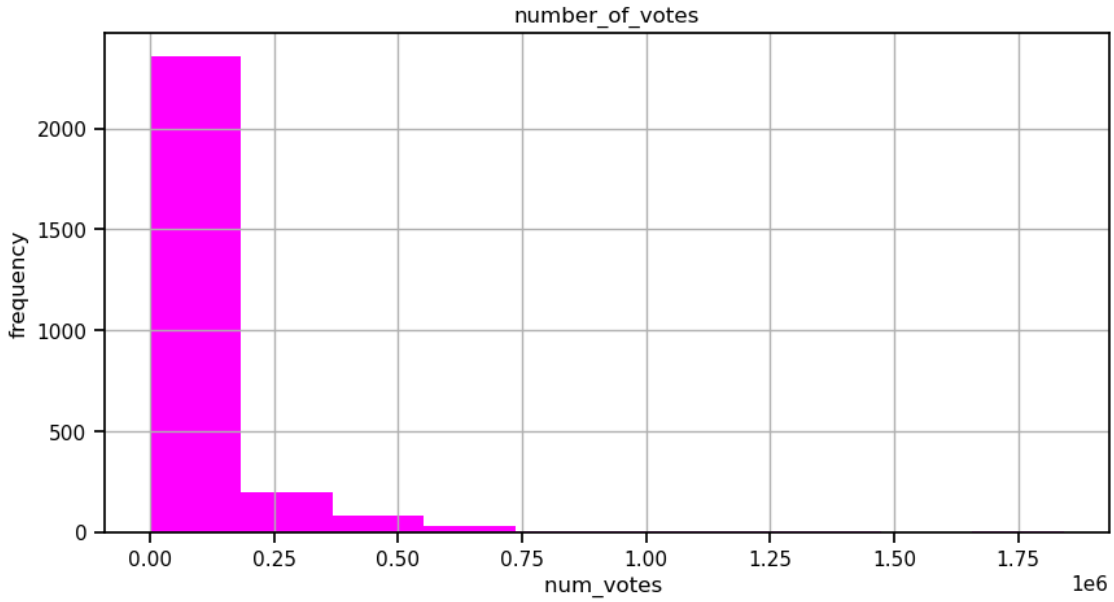
[68]:
```
title                0
studio               0
domestic_gross       0
foreign_gross        0
year                 0
tconst              84
```

```
primary_title        84
original_title_x     84
runtime_minutes       0
genres               84
averagerating         0
numvotes            443
original_title_y      0
release_year          0
dtype: int64
```

[69]:
```
#visualizing the distribution of numvotes
print(final_data.numvotes.agg(['mean', 'std', 'min', 'max']))
final_data.numvotes.hist(color='magenta', figsize=(10, 5))
plt.title("number_of_votes")
plt.xlabel('num_votes ')
plt.ylabel('frequency')
```

```
mean    7.731767e+04
std     1.362478e+05
min     5.000000e+00
max     1.841066e+06
Name: numvotes, dtype: float64
```

[69]:
```
Text(0, 0.5, 'frequency')
```



[70]:
```
#Filling the skewed data using imputation median
final_data.numvotes.fillna(final_data.numvotes.median(), inplace=True)
```

[71]:
```
#now lets check if we succeeded
final_data.isnull().sum()
```

[71]:
```
title               0
studio              0
domestic_gross      0
foreign_gross       0
year                0
tconst             84
primary_title      84
original_title_x   84
runtime_minutes     0
genres             84
averagerating       0
numvotes            0
original_title_y    0
release_year        0
dtype: int64
```

[72]:
```
#Checking the most dominant genre using the process

#Calling the value_counts method to find the mode genre
top_genre = final_data.genres.value_counts().head(1)
print(f'The most common genre in our data is: {top_genre}')
```

```
The most common genre in our data is: genres
Drama    323
Name: count, dtype: int64
```

[74]:
```
#Replacing the missing values with mode imputation
final_data.genres.fillna(final_data.genres.mode().iloc[0], inplace=True)
```

[75]:
```
#confirming that we have no missing values in our dataset
final_data.isnull().sum()
```

[75]:
```
title              0
studio             0
domestic_gross     0
foreign_gross      0
year               0
tconst            84
primary_title     84
original_title_x  84
runtime_minutes    0
genres             0
averagerating      0
numvotes           0
```

```
original_title_y     0
release_year         0
dtype: int64
```

[76]:
```
#Confirming the dtypes of final dataset
final_data.dtypes
```

[76]:
```
title                object
studio               object
domestic_gross      float64
foreign_gross        object
year                  int64
tconst               object
primary_title        object
original_title_x     object
runtime_minutes     float64
genres               object
averagerating       float64
numvotes            float64
original_title_y     object
release_year          int32
dtype: object
```

[77]:
```
movie_gross_dt['foreign_gross'] = movie_gross_dt['foreign_gross'].astype(str).
 ↪str.replace('.','',regex=False)
```

[78]:
```
#creating a new column that we will use to determine the financial success of␣
 ↪the company
final_data['total_gross'] = final_data.domestic_gross.astype(str) + final_data.
 ↪foreign_gross.astype(str)

#determinng whether new column was created successfully
final_data.head()
```

[78]:

	title	studio	domestic_gross	foreign_gross	year	\
0	Toy Story 3	BV	415000000.0	652000000.0	2010	
1	Inception	WB	292600000.0	535700000.0	2010	
2	Shrek Forever After	P/DW	238700000.0	513900000.0	2010	
3	The Twilight Saga: Eclipse	Sum.	300500000.0	398000000.0	2010	
4	Iron Man 2	Par.	312400000.0	311500000.0	2010	

	tconst	primary_title	original_title_x	\
0	tt0435761	Toy Story 3	Toy Story 3	
1	tt1375666	Inception	Inception	
2	tt0892791	Shrek Forever After	Shrek Forever After	
3	tt1325004	The Twilight Saga: Eclipse	The Twilight Saga: Eclipse	
4	tt1228705	Iron Man 2	Iron Man 2	

```
   runtime_minutes                      genres  averagerating   numvotes  \
0            103.0  Adventure,Animation,Comedy            8.3   682218.0
1            148.0     Action,Adventure,Sci-Fi            8.8  1841066.0
2             93.0  Adventure,Animation,Comedy            6.3   167532.0
3            124.0     Adventure,Drama,Fantasy            5.0   211733.0
4            124.0     Action,Adventure,Sci-Fi            7.0   657690.0

            original_title_y  release_year             total_gross
0                Toy Story 3          2010  415000000.0652000000.0
1                  Inception          2010  292600000.0535700000.0
2        Shrek Forever After          2010  238700000.0513900000.0
3  The Twilight Saga: Eclipse          2010  300500000.0398000000.0
4                 Iron Man 2          2010  312400000.0311500000.0
```

[79]:
```
final_data['total_gross']
```

[79]:
```
0                                415000000.0652000000.0
1                                292600000.0535700000.0
2                                238700000.0513900000.0
3                                300500000.0398000000.0
4                                312400000.0311500000.0
                              …
3122    14000.0<bound method NDFrame._add_numeric_oper…
3123    11400.0<bound method NDFrame._add_numeric_oper…
3124    11400.0<bound method NDFrame._add_numeric_oper…
3125    11400.0<bound method NDFrame._add_numeric_oper…
3126    1700.0<bound method NDFrame._add_numeric_opera…
Name: total_gross, Length: 3127, dtype: object
```

[80]:
```
#Splitting movie genres and exploded to allow for genre-specific analyses
final_data.genres = final_data.genres.str.split(',')
final_data.head()
```

[80]:
```
                     title studio  domestic_gross foreign_gross  year  \
0                Toy Story 3     BV     415000000.0   652000000.0  2010
1                  Inception     WB     292600000.0   535700000.0  2010
2        Shrek Forever After   P/DW     238700000.0   513900000.0  2010
3  The Twilight Saga: Eclipse   Sum.     300500000.0   398000000.0  2010
4                 Iron Man 2   Par.     312400000.0   311500000.0  2010

      tconst               primary_title            original_title_x  \
0  tt0435761                 Toy Story 3                 Toy Story 3
1  tt1375666                   Inception                   Inception
2  tt0892791         Shrek Forever After         Shrek Forever After
3  tt1325004  The Twilight Saga: Eclipse  The Twilight Saga: Eclipse
4  tt1228705                   Iron Man 2                  Iron Man 2
```

```
   runtime_minutes                          genres  averagerating   numvotes  \
0            103.0  [Adventure, Animation, Comedy]            8.3   682218.0
1            148.0     [Action, Adventure, Sci-Fi]            8.8  1841066.0
2             93.0  [Adventure, Animation, Comedy]            6.3   167532.0
3            124.0     [Adventure, Drama, Fantasy]            5.0   211733.0
4            124.0     [Action, Adventure, Sci-Fi]            7.0   657690.0

             original_title_y  release_year            total_gross
0                  Toy Story 3          2010  415000000.0652000000.0
1                    Inception          2010  292600000.0535700000.0
2          Shrek Forever After          2010  238700000.0513900000.0
3  The Twilight Saga: Eclipse          2010  300500000.0398000000.0
4                   Iron Man 2          2010  312400000.0311500000.0
```

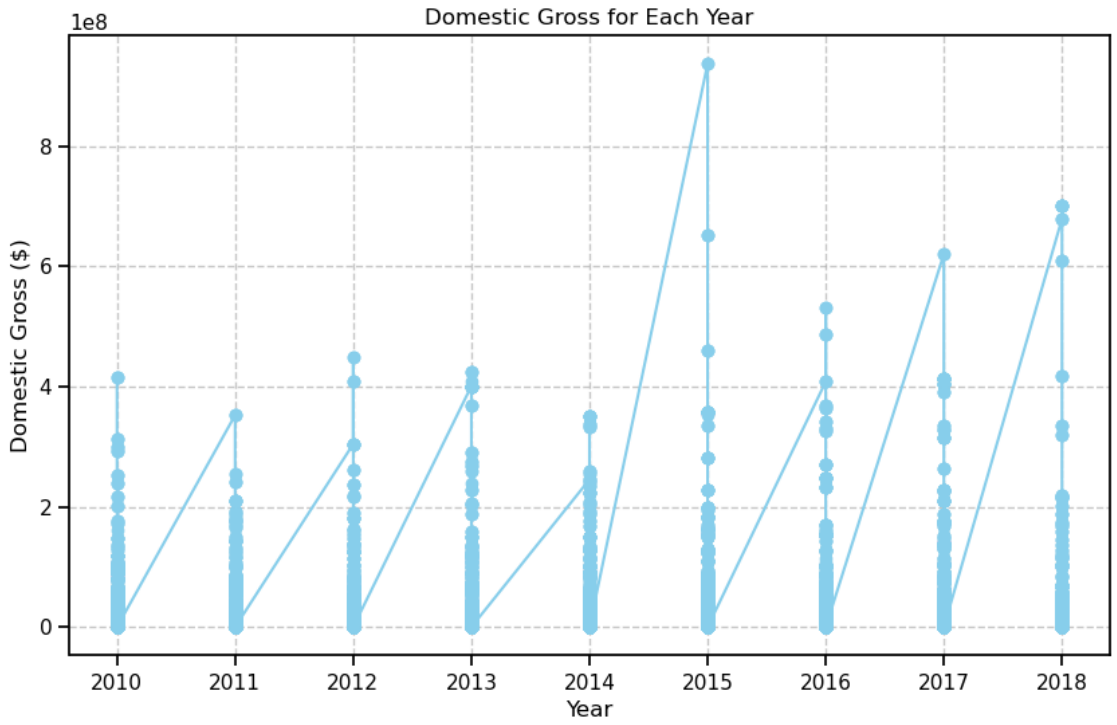
[81]:
```
#Exploding the genres
final_data = final_data.explode('genres')
```

[82]:
```
final_data.head()
```

[82]:
```
          title studio  domestic_gross foreign_gross  year     tconst  \
0  Toy Story 3     BV     415000000.0   652000000.0  2010  tt0435761
0  Toy Story 3     BV     415000000.0   652000000.0  2010  tt0435761
0  Toy Story 3     BV     415000000.0   652000000.0  2010  tt0435761
1    Inception     WB     292600000.0   535700000.0  2010  tt1375666
1    Inception     WB     292600000.0   535700000.0  2010  tt1375666

  primary_title original_title_x  runtime_minutes     genres  averagerating  \
0   Toy Story 3      Toy Story 3            103.0  Adventure            8.3
0   Toy Story 3      Toy Story 3            103.0  Animation            8.3
0   Toy Story 3      Toy Story 3            103.0     Comedy            8.3
1     Inception        Inception            148.0     Action            8.8
1     Inception        Inception            148.0  Adventure            8.8

    numvotes original_title_y  release_year            total_gross
0   682218.0      Toy Story 3          2010  415000000.0652000000.0
0   682218.0      Toy Story 3          2010  415000000.0652000000.0
0   682218.0      Toy Story 3          2010  415000000.0652000000.0
1  1841066.0        Inception          2010  292600000.0535700000.0
1  1841066.0        Inception          2010  292600000.0535700000.0
```

[83]:
```
# Plotting the domestic gross for each year
plt.figure(figsize=(10, 6))
plt.plot(final_data['year'], final_data['domestic_gross'], marker='o',␣
 ↪color='skyblue', linestyle='-')

# Adding labels and title
```

```
plt.xlabel('Year')
plt.ylabel('Domestic Gross ($)')
plt.title('Domestic Gross for Each Year')

# Adding grid
plt.grid(True, linestyle='--', alpha=0.7)

# Show plot
plt.show()
```



[84]:
```
final_data.dtypes
```

[84]:
```
title                object
studio               object
domestic_gross      float64
foreign_gross        object
year                  int64
tconst               object
primary_title        object
original_title_x     object
runtime_minutes     float64
genres               object
averagerating       float64
```

```
numvotes            float64
original_title_y     object
release_year          int32
total_gross          object
dtype: object
```

[85]:
```
final_data['title'].unique()
```

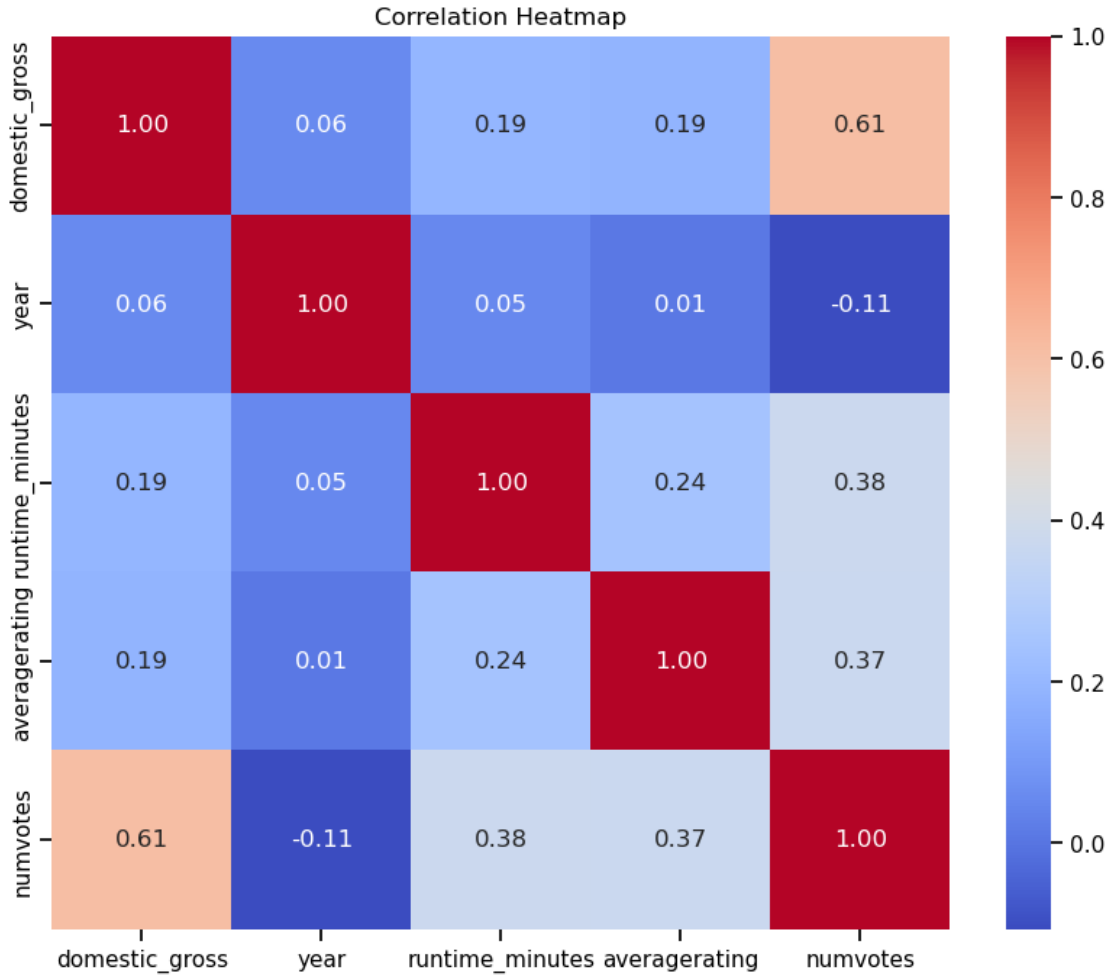
[85]:
```
array(['Toy Story 3', 'Inception', 'Shrek Forever After', …,
       'The Escape', 'Souvenir', 'An Actor Prepares'], dtype=object)
```

[86]:
```
#checking for leading & trailing whitespaces and removing them in all datasets
[col.strip() for col in movie_gross_dt.columns]
```

[86]:
```
['title', 'studio', 'domestic_gross', 'foreign_gross', 'year']
```

[88]:
```
# Creating a heatmap to visualize the correlation matrix
# Plotting the correlation heatmap
columns_of_interest =␣
 ↪['domestic_gross','year','runtime_minutes','averagerating','numvotes']
selected_column_df = final_data[columns_of_interest]
correlation_matrix = selected_column_df.corr()
# Plotting the correlation heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Heatmap')
plt.show
```

[88]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```

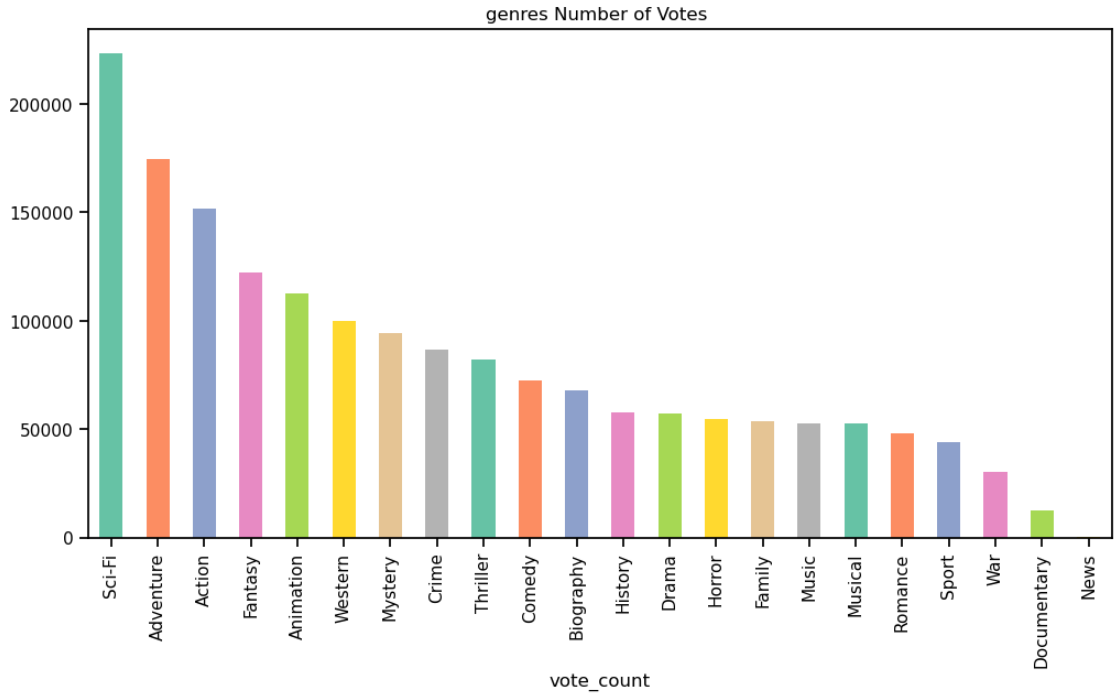


1.FROM THE ABOVE CORRELATION VISUALIZATION,DOMESTIC GROSS AND NUMBER OF VOTES HAVE A MODERATELY POSITIVE CORRELATION WHICH SUGGESTS THAT MOVIES WITH HIGHER DOMESTIC GROSS TEND TO ATTRACT MORE VIEWER ENGAGEMENT AND VOTES. 2.FROM THE ABOVE CORRELATION HEATMAP,THERE IS A WEAK POSITIVE CORRELATION BETWEEN AVERAGE RATING AND NUMBER OF VOTES WHICH INDICATES THAT MOVIES WITH HIGHER AVERAGE RATING MAY ATTRACT SLIGHTLY MORE VIEWER ENGAGEMENT. 3.FROM THE ABOVE,THERE IS A WEAK POSITIVE CORELLATION BETWEEN AVERAGE RATING AND DOMESTIC GROSS,THIS SUGGESTS THAT MOVIES WITH HIGHER AVERAGE RATINGS TEND TO PERFORM SLIGHTLY BETTER IN TERMS OF REVENUE. 4.NO SIGNIFICANT CORRELATION BETWEEN YEAR OF RELEASE AND DOMESTIC GROSS OR NUMBER OF VOTES 5.NO SIGNIFICANT CORRELATION BETWEEN YEAR OF RELEASE AND AVERAGE RATING.

[89]:
```
# Visualizing the relationship between genres and number of votes
plt.figure(figsize=(12, 6))
genre_num_votes = final_data.groupby('genres')['numvotes'].mean()
```

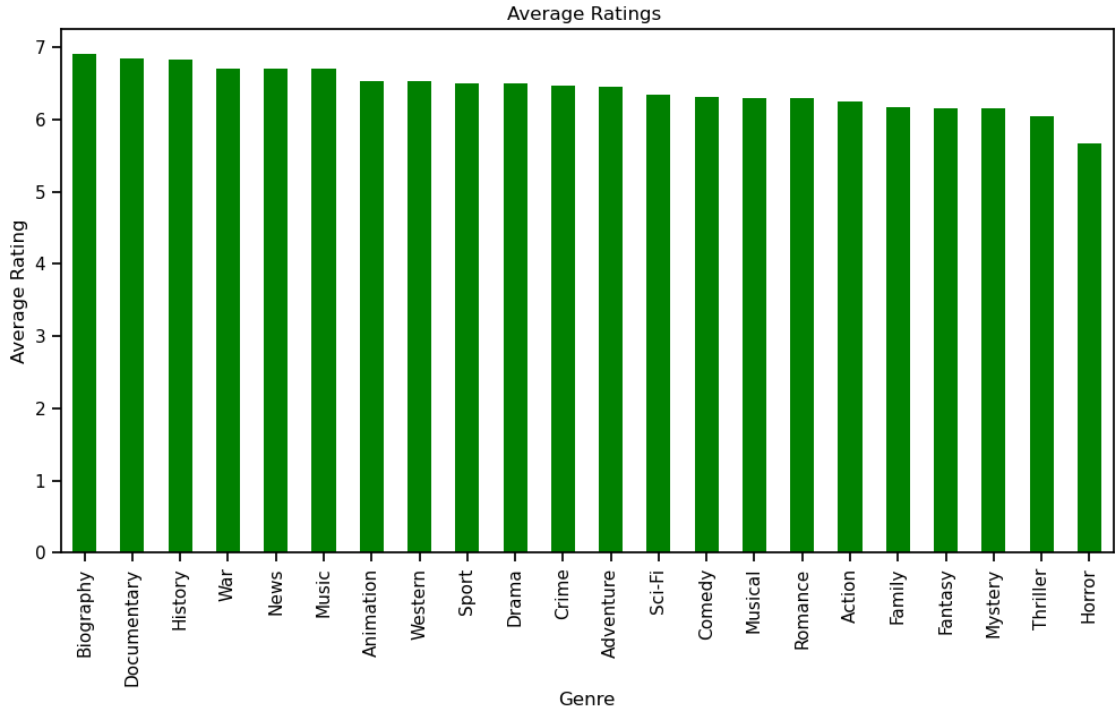
```
sorted_data = genre_num_votes.sort_values(ascending=False)
colors = sns.color_palette("Set2", n_colors=len(sorted_data))
sorted_data.plot(kind='bar', color=colors)
plt.title("genres Number of Votes ")
plt.xlabel("vote_count")
```

[89]: Text(0.5, 0, 'vote_count')



FROM THE ABOVE BAR PLOT,SCI FI GENRE HAS HIGHER NUMBER OF VOTES SUGGESTING HIGHEST NUMBER OF VIEWERS WHICH INDICATES BROADER AUDIENCE APPEAL AND POTENTIALLY HIGHER MARKET POTENTIAL. SC FI IS FOLLOWED BY ADVENTURE AND ACTION.

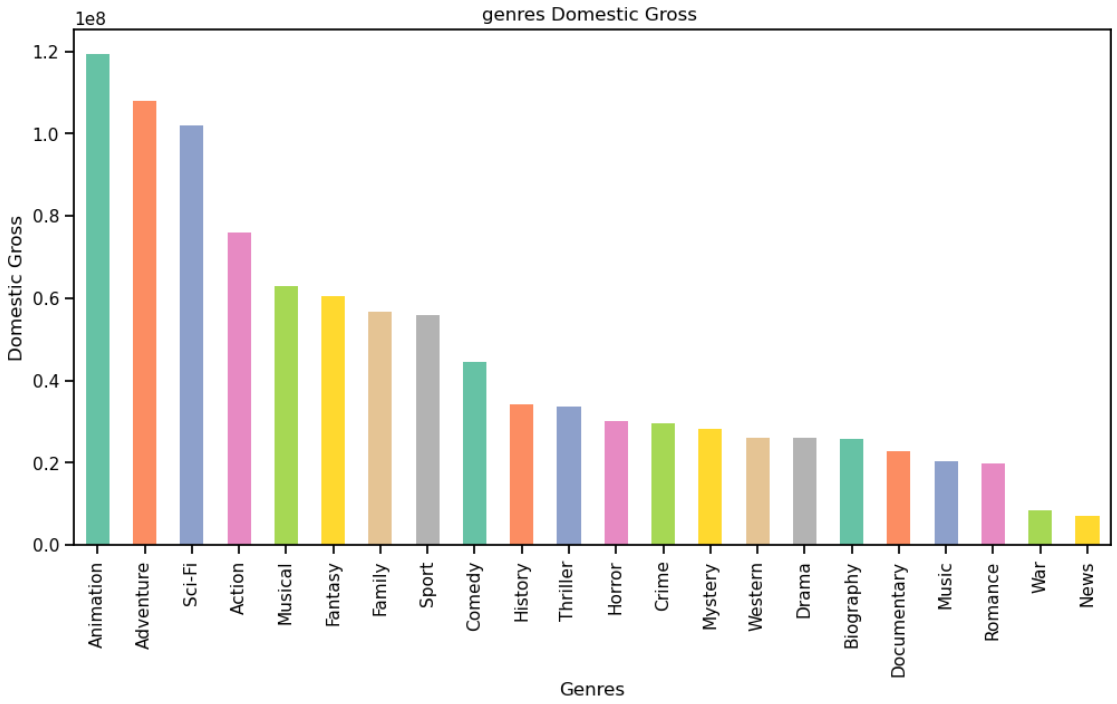
[90]:
```
# Visualizing the relationship between genre and average ratings
plt.figure(figsize=(12, 6))
genre_avg_ratings = final_data.groupby('genres')['averagerating'].mean()
sorted_data = genre_avg_ratings.sort_values(ascending=False)
sorted_data.plot(kind='bar', color='green')
plt.title("Average Ratings ")
plt.xticks(rotation=90)
plt.xlabel("Genre")
plt.ylabel("Average Rating");
```



FROM ABOVE BAR PLOT BIOGRAPHY,DOCUMENTARY AND HISTORY HAVE HIGHEST AVERAGE RATING INDICATING BETTER RECEPTION AND SATISFACTION AMONG THE AUDIENCE.INVESTING IN GENRES WITH CONSISTENTLY HIGH AVARAGE RATINGS CAN LEAD TO POSITIVE CRITICAL RECEPTION,WHICH IS ESSENTIAL FOR BUILDING A REPUTABLE BRAND AND ATTRACTING TALENT IN THE INDUSTRY.INVESTING IN MOVIES WITHIN THESE CATEGORIES COULD ENHANCE THE STUDIO'S REPUTATION FOR DELIVERING HIGH QUALITY CONTENT AND ATTRACT VIEWERS WHO VALUE ENGAGING STORYTELLING.

[94]:
```
# Visualizing the relationship between genres and domestic gross
plt.figure(figsize=(12, 6))
genre_gross = final_data.groupby('genres')['domestic_gross'].mean()
sorted_data = genre_gross.sort_values(ascending=False)
colors = sns.color_palette("Set2", n_colors=len(sorted_data))
sorted_data.plot(kind='bar', color=colors)
plt.title("genres Domestic Gross ")
plt.xlabel("Genres")
plt.ylabel("Domestic Gross")
plt.show
```

[94]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



[ ]:
```
RECOMMENDATION
```

From the above barplots,Genres such as Animation,Adventure,Sci Fi ,Action tend to have higher average domestic gross and moderately higher average rating compared to others.Investing in content production within these genres could potentially lead to higher revenue generation at the box office

Additionally, Microsoft could explore opportunities to blend these successful genres with innovative storytelling and production techniques to differentiate their content and attract a broader audience.