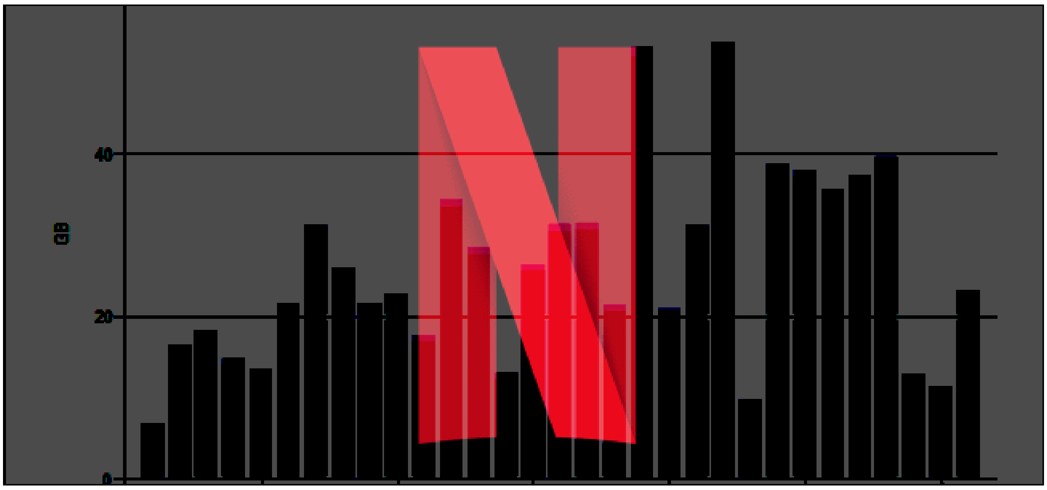
Candidate Numbers :

346039, 762519, 820229, 671096



Big Data Coursework - Netflix Dataset

This notebook is a Data Preprocessing and Exploratory Data Analysis report on the Netflix Price Data dataset.

Table of Contents

0.0 - Importing Libraries and Preparing Environment

In [1]:

```
#Base Libraries
import re
import time
import numpy as np
import pandas as pd

#Library for Google Colab
from google.colab import drive

#Library for Plotting
import seaborn as sns
sns.set(style="darkgrid")
import matplotlib.pyplot as plt
%matplotlib inline

#Library for Data Preprocessing and Cleaning
from sklearn.model_selection import train_test_split
from sklearn.impute import IterativeImputer, KNNImputer
from sklearn.base import TransformerMixin, BaseEstimator
from sklearn.experimental import enable_iterative_imputer
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: FutureWarning:
pandas.util.testing is deprecated. Use the functions in the public API at pandas.testing instead.
pandas.util.testing is deprecated. Use the functions in the public API at pandas.testing instead.
  import pandas.util.testing as tm
```

In [0]:

```
Start_time = time.time()
```

In [3]:

```
#Mount Google Drive on kernel
drive.mount('/content/gdrive')
path = "/content/gdrive/My Drive"
```

```
Drive already mounted at /content/gdrive; to attempt to forcibly remount, call
drive.mount("/content/gdrive", force_remount=True).
```

The decorator below is used for the whole notebook.

It constantly monitors how long each step of the ML Pipeline takes to run.

In [0]:

```
def time_step(fn):
  def wrapper(*args, **kwargs):
    start_time = time.time()
    result = fn(*args, **kwargs)
    end_time = time.time()
    time_taken = round(end_time - start_time, 4)
    if time_taken < 60:
      print(f'{fn.__name__} took {time_taken} seconds to run')
    else :
      print(f'{fn.__name__} took {time_taken/60} minutes to run')
    return result
  return wrapper
```

1.0 - Business Objective

Licensing agreements are made between Netflix and the bodies that produce TV shows and movies. These agreements give Netflix the right to stream the content. In 2019, Netflix spent 15.3 billion dollars on these agreements and this spending is forecast to reach 17 billion dollars in 2020.

We want to build a model that predicts the Netflix rating of a movie/TV show. This would allow Netflix to determine which new content to obtain the license for and how much they should spend on it. If predicted ratings for a new movie are high enough, then it is worth Netflix paying for the license. If not, it is not worth it. These decisions are even more important, when they relate to *exclusive content* as obtaining the license for these is more expensive than non-exclusive content.

The predictive model will use average Netflix rating as the dependent variable and IMDB data about the specific TV show/movie for the independent variables.

2.0 - Data Preparation

2.1 - Loading Data - Netflix Movie Titles Dataset and IMDB Dataset

The data from these two datasets is loaded into 2 dataframes before being converted into a third Movie info dataframe.

In [0]:

```
# The Netflix dataset includes movieID and name.  The name will be used when merging with the IMDB
dataset.
# The IMDB dataset includes all the data that we will use for the independent variables.
netflix_movie_df = pd.read_csv("/content/gdrive/My Drive/Colab Notebooks/netflix-prize-
data/movie_titles.csv", encoding = "ISO-8859-1", header = None, names = ['Movie_Id', 'Year_of_relea
se', 'Movie_Name', 'Alternate_Name'])
imdb_df = pd.read_csv("/content/gdrive/My Drive/Colab Notebooks/netflix-prize-
data/movie_metadata.csv", encoding = "utf_8", header = 'infer')
```

In [6]:

```
#Validating Netflix dataframe
netflix_movie_df.head()
```

Out[6]:

	Movie_Id	Year_of_release	Movie_Name	Alternate_Name
0	1	2003.0	Dinosaur Planet	NaN
1	2	2004.0	Isle of Man TT 2004 Review	NaN
2	3	1997.0	Character	NaN
3	4	1994.0	Paula Abdul's Get Up & Dance	NaN
4	5	2004.0	The Rise and Fall of ECW	NaN

In [7]:

```
#Validating shape of Netflix dataframe
netflix_movie_df.shape
```

Out[7]:

```
(17770, 4)
```

There are 17770 movies in the Netflix dataframe. For each, the movieID, title, year of release and alternative name are given.

In [8]:

```
#Validating IMDB dataframe
imdb_df.head()
```

Out[8]:

	color	director_name	num_critic_for_reviews	duration	director_facebook_likes	actor_3_facebook_likes	actor_2_name	actor_1_faceb
0	Color	James Cameron	723.0	178.0	0.0	855.0	Joel David Moore	
1	Color	Gore Verbinski	302.0	169.0	563.0	1000.0	Orlando Bloom	
2	Color	Sam Mendes	602.0	148.0	0.0	161.0	Rory Kinnear	
3	Color	Christopher Nolan	813.0	164.0	22000.0	23000.0	Christian Bale	
4	NaN	Doug Walker	NaN	NaN	131.0	NaN	Rob Walker	

In [9]:

```
#Validating shape of IMDB dataframe
imdb_df.shape
```

Out[9]:

```
(5043, 28)
```

There are 5043 movies in the IMDB dataframe. For each, the movie title is given as well as 27 *other* variables.

In [0]:

```
# '\xa0' is removed from movie title column in IMDB dataframe
imdb_df.loc[:,'movie_title'] = imdb_df.loc[:,'movie_title'].apply(lambda x : x.strip('\xa0'))
# The movie title variable from the IMDB dataframe and the movie name from the Netflix dataframe a
re used to merge the two dataframes together
movie_info_df = netflix_movie_df.merge(imdb_df, left_on='Movie_Name', right_on='movie_title')
```

In [0]:

```
# The columns of interest are then pulled into a new dataframe called Movie Info
movie_info_df = movie_info_df.loc[:,
['Movie_Id','movie_title','Movie_Name','Alternate_Name','title_year','Year_of_release','actor_1_nam
e','actor_2_name','actor_3_name',

'director_name','actor_1_facebook_likes','actor_2_facebook_likes','actor_3_facebook_likes','directo
r_facebook_likes',
                                     'cast_total_facebook_likes', 'movie_facebook_likes','gross','b
dget','facenumber_in_poster','num_voted_users','num_user_for_reviews',
                                     'num_critic_for_reviews','imdb_score','plot_keywords','genres'
'language','country','aspect_ratio', 'content_rating','color','duration','movie_imdb_link']]
# The duplicates are removed from Movie Info dataframe
movie_info_df = movie_info_df.drop_duplicates()
```

In [12]:

```
#Validating new Movie Info dataframe
movie_info_df.head()
```

Out[12]:

	Movie_Id	movie_title	Movie_Name	Alternate_Name	title_year	Year_of_release	actor_1_name	actor_2_name	actor_3_name	direct
0	3	Character	Character	NaN	1997.0	1997.0	Jan Decleir	Fedja van Huêt	Hans Kesting	Mike
1	24	My Bloody Valentine	My Bloody Valentine	NaN	2009.0	1981.0	Jensen Ackles	Jaime King	Edi Gathegi	Patric
2	30	Something's Gotta Give	Something's Gotta Give	NaN	2003.0	2003.0	Keanu Reeves	Jon Favreau	Paul Michael Glaser	Nan
3	45	The Love Letter	The Love Letter	NaN	1998.0	1999.0	Jennifer Jason Leigh	Campbell Scott	Estelle Parsons	D
5	17721	The Love Letter	The Love Letter	NaN	1998.0	1998.0	Jennifer Jason Leigh	Campbell Scott	Estelle Parsons	D

2.2 - Variable Notes - Netflix Movie Titles dataframe and IMDB dataframe

Variables are categorised into multiple types : Nominal, Ordinal, Interval, Ratio.

To simplify, the types are narrowed down into 3 main types:

- Numeric : Variables containing numeric values.
- Categorical : Variables containing text data / each unique value indicates a *category*.
- Boolean : Categorical Variables with two categories

Unless otherwise stated, the variable comes from the IMDB dataframe.

Input data	Definition	Category
Movie_Id	Number of movie (from Netflix Movie Titles dataframe)	Numeric
movie_title	Name of the movie (from Netflix Movie Titles dataframe)	Categorical
Movie_Name	Name of the movie	Categorical
Alternate_Name	Another name of the movie (from Netflix Movie Titles dataframe)	Categorical
title_year	Production Year	Numeric
Year_of_release	Production Year (from Netflix Movie Titles dataframe)	Numeric
actor_1_name	Primary actor starring in the movie	Categorical

Input data	Definition	Category
actor_2_name	Secondary actor starring in the movie	Categorical
actor_3_name	Tertiary actor starring in the movie	Categorical
director_name	Name of the Director of the Movie	Categorical
actor_1_facebook_likes	Number of likes of the actor_1 on his/her Facebook Page	Numeric
actor_2_facebook_likes	Number of likes of the actor_2 on his/her Facebook Page	Numeric
actor_3_facebook_likes	Number of likes of the actor_3 on his/her Facebook Page	Numeric
director_facebook_likes	Number of likes of the director on his/her Facebook Page	Numeric
cast_total_facebook_likes	Number of all actors likes on Facebook	Numeric
movie_facebook_likes	Number of movie likes on Facebook	Numeric
gross	Earnings of movie in U.S. dollars	Numeric
budget	Amount of cost of production in U.S. dollars	Numeric
facenumber_in_poster	Number of actor faces on movie poster	Numeric
num_voted_users	Number of people voting on movie	Numeric
num_user_for_reviews	Number of people reviewing movie	Numeric
num_critic_for_reviews	Number of critical evaluation reviews on movie	Numeric
imdb_score	Average score collected by IMDB	Numeric
plot_keywords	Words or phrases that are searchable for specific movie	Categorical
genres	Classifications of the movie	Categorical
language	Movie language	Categorical
country	Production Countries	Categorical
aspect_ratio	Ratio between the width and the height of the movie display	Numeric
content_rating	MPAA movie ratings	Categorical
color	Boolean Variable of color used	Boolean
duration	Length of the movie	Numeric
movie_imdb_link	Link to the movie on IMDB website	Categorical

2.3 - Loading Data - Ratings Dataset

The Ratings data is in text file format. These text files are made up of lists of movieIDs. For each movieID, there is a sub-list of customerID's that watched that movie, their rating and the date. The Ratings dataframe is developed by producing a dictionary of movieIDs, with a list of ratings as the value for each movieID. This dictionary is converted into the final dataframe which is grouped by movieID to give the average rating for each movie.

a. Upload of the first dataset and transform it into a dictionary

In [0]:

```
f1="/content/gdrive/My Drive/Colab Notebooks/netflix-prize-data/combined_data_1.txt"

#Creation of an empty dictionary
D1={}

# at the beginning, no list and no dictionary
key=None
for line in open(f1):
    # remove the leading and trailing spaces after each line
    line=line.strip()
    # search for lines starting with movie_id:
    if re.compile(r"\d+:").search(line):
        if key is not None:
            # After the keys have been created...
            D1[key]=list1
            # ...then the value of these keys are lists
        key=line.strip(":")
        # If the key is None then the key needs to be created by removing the :
        # creation of an empty list for the values of the keys
        list1=[]
    else:
      # Search for the pattern of the ratings, which is a digit between 2 commas...,
        p = re.compile("\,\d\,")
        for r in p.findall(line):
```

```
        for m in p.findall(line):
          # ... in each line
            m = m.replace(',', '')
            # Delete the commas by replacing them with nothing
            list1.append(m)
        # append the ratings to the list in the dictionary
    D1[key] = list1
```

In [14]:

```
# Transform the dictionary into a dataframe.
# Concatenate all the keys (k) with a dataframe that uses the dictionary values as its argument
# Keep default so that axis=0.  This means movie ID (key) will go in the index instead of the header
df1=pd.concat({k: pd.DataFrame(v) for k, v in D1.items()})

# Reset the index in the current dataframe. Like when ungrouping a pivot table, the former index (movie_ ID and index number) will be put in new columns
# and copied for each row/rating
df1.reset_index(inplace=True)

# The movieId value is in a new column 'level_0'.
# Let's rename it
df1.rename(columns={'level_0':'Movie_Id'}, inplace=True)
# Transform the ratings into numerical values, while renaming the column with the ratings
df1.loc[:,'rating']=pd.to_numeric(df1[0])

# Keep only the columns we need
df1 = df1.loc[:,['Movie_Id','rating']]

# Datatypes
pd.DataFrame(df1.dtypes, columns = ['Data Type'])

# Check for missing values
df1.loc[:,['Movie_Id','rating']].isna().sum()
```

Out[14]:

```
Movie_Id    0
rating      0
dtype: int64
```

In [0]:

```
# drop missing values
df1.dropna(inplace=True)
```

In [0]:

```
# Create a new column with the average ratings
df1.loc[:,'avg_rating'] = df1.loc[:,'rating'].groupby(df1.loc[:,'Movie_Id']).transform('mean')
```

In [17]:

```
# remove the column 'rating'
df1.drop('rating', axis=1,inplace=True)

# remove the duplicates
df1.drop_duplicates(inplace=True)

# reset the index
df1.reset_index(inplace=True)

# Keep only the columns we need
df1 = df1.loc[:,['Movie_Id','avg_rating']]
df1.head()
```

Out[17]:

	Movie_Id	avg_rating
0	1	3.749543

	Movie_Id	avg_rating
1	2	3.558621
2	3	3.641153
3	4	2.739437
4	5	3.919298

b. Upload of the 2nd dataset

In [0]:

```
f2="/content/gdrive/My Drive/Colab Notebooks/netflix-prize-data/combined_data_2.txt"

# at the beginning, no list and no dictionary
D2={}
key=None
for line in open(f2):
    # remove the leading and trailing spaces after each line
    line=line.strip()
    # search for the lines starting with a movie_id and a :
    if re.compile(r"\d+:").search(line):
        if key is not None:
            # After the keys have been created...
            D2[key]=list2
            # ...then the value of these keys are lists
        key=line.strip(":")
        # If the key is None then the key needs to be created by removing the :
        # creation of an empty list for the values of the keys
        list2=[]
    else:
 # Search for the pattern of the ratings, which is a digit between 2 commas...,
        p = re.compile("\,\d\,")
        for m in p.findall(line):
          # ... in each line
            m = m.replace(',', '')
            #  Delete the commas by replacing them with nothing
            list2.append(m)
            # append the ratings to the list in the dictionary
    D2[key] = list2
```

In [19]:

```
# Transform the dictionary into a dataframe.
# Concatenate all the keys (k) with a dataframe that uses the dictionary values as its argument
# Keep default so that axis=0.  This means movie ID (key) will go in the index instead of the
header
df2=pd.concat({k: pd.DataFrame(v) for k, v in D2.items()}, axis=0)

# Reset the index in the current dataframe. Like when ungrouping a pivot table, the former index (
movie_ ID and index number) will be put in new columns
# and copied for each row/rating
df2.reset_index(inplace=True)

# The movieId value is in a new column 'level_0'.
# Let's rename it
df2.rename(columns={'level_0':'Movie_Id'}, inplace=True)

# Transform the ratings into numerical values, while renaming the column with the ratings
df2.loc[:,'rating']=pd.to_numeric(df2[0])

# Keep only the columns we need
df2 = df2.loc[:,['Movie_Id','rating']]

# check for missing values
df2.loc[:,['Movie_Id','rating']].isna().sum()
```

Out[19]:

```
Movie_Id    0
rating      0
dtype: int64
```

In [0]:

In [0]:

```
# drop missing values
df2.dropna(inplace=True)
```

In [0]:

```
# Create a new column with the average ratings
df2.loc[:,'avg_rating'] = df2.loc[:,'rating'].groupby(df2.loc[:,'Movie_Id']).transform('mean')
```

In [22]:

```
# remove the column 'rating'
df2.drop('rating', axis=1,inplace=True)

# remove the duplicates
df2.drop_duplicates(inplace=True)

# reset the index
df2.reset_index(inplace=True)

# keep only the columns we need
df2 = df2.loc[:,['Movie_Id','avg_rating']]
df2.head()
```

Out[22]:

	Movie_Id	avg_rating
0	4500	3.329457
1	4501	3.043697
2	4502	2.144231
3	4503	3.213603
4	4504	3.153409

c. Upload of the 3rd dataset

In [0]:

```
f3="/content/gdrive/My Drive/Colab Notebooks/netflix-prize-data/combined_data_3.txt"

#Creation of an empty dictionary
D3={}

# at the beginning, no list and no dictionary

key=None
for line in open(f3):
    # remove the leading and trailing spaces after each line
    line=line.strip()
    # search for the lines starting with a movie_id and a :
    if re.compile(r"\d+:").search(line):
        if key is not None:
            # After the keys have been created...
            D3[key]=list3
            # ...then the value of these keys are lists
        key=line.strip(":")
        # If the key is None then the key needs to be created by removing the :
        # creation of an empty list for the values of the keys
        list3=[]
    else:
        # Search for the pattern of the ratings, which is a digit between 2 commas...,
        p = re.compile("\,\d\,")
        for m in p.findall(line):
           # ... in each line
            m = m.replace(',', '')
            #  Delete the commas by replacing them with nothing
            list3.append(m)
           # append the ratings to the list in the dictionary
    D3[key] = list3
```

In [24]:

```
# Transform the dictionary into a dataframe.
# Concatenate all the keys (k) with a dataframe that uses the dictionary values as its argument
# Keep default so that axis=0.  This means movie ID (key) will go in the index instead of the
header
df3=pd.concat({k: pd.DataFrame(v) for k, v in D3.items()}, axis=0)

# Reset the index in the current dataframe. Like when ungrouping a pivot table, the former index (
movie_ ID and index number) will be put in new columns
# and copied for each row/rating
df3.reset_index(inplace=True)

# The movieId value is in a new column 'level_0'.
# Let's rename it
df3.rename(columns={'level_0':'Movie_Id'}, inplace=True)

# Transform the ratings into numerical values, while renaming the column with the ratings
df3.loc[:,'rating']=pd.to_numeric(df3[0])

# Keep only the columns we need
df3 = df3.loc[:,['Movie_Id','rating']]

# check for missing values
df3.loc[:,['Movie_Id','rating']].isna().sum()
```

Out[24]:

```
Movie_Id    0
rating      0
dtype: int64
```

In [0]:

```
# drop missing values
df3.dropna(inplace=True)
```

In [0]:

```
# Create a new column with the average ratings
df3.loc[:,'avg_rating'] = df3.loc[:,'rating'].groupby(df3.loc[:,'Movie_Id']).transform('mean')
```

In [27]:

```
# remove the column rating
df3.drop('rating', axis=1,inplace=True)

# remove the duplicates
df3.drop_duplicates(inplace=True)

# reset the index
df3.reset_index(inplace=True)

# keep only the columns we need
df3 = df3.loc[:,['Movie_Id','avg_rating']]
df3.head()
```

Out[27]:

	Movie_Id	avg_rating
0	9211	2.369048
1	9212	3.706302
2	9213	3.125874
3	9214	2.403846
4	9215	3.482667

d. Upload of the 4th dataset

In [0]:

```
f4="/content/gdrive/My Drive/Colab Notebooks/netflix-prize-data/combined_data_4.txt"

#Creation of an empty dictionary
D4={}

# at the beginning, no list and no dictionary

key=None
for line in open(f4):
    # remove the leading and trailing spaces after each line
    line=line.strip()
    # search for the lines starting with a movie_id and a :
    if re.compile(r"\d+:").search(line):
        if key is not None:
            # After the keys have been created...
            D4[key]=list4
            # ...then the value of these keys are lists
        key=line.strip(":")
        # If the key is None then the key needs to be created by removing the :
        # creation of an empty list for the values of the keys
        list4=[]
    else:
        # Search for the pattern of the ratings, which is a digit between two commas...,
        p = re.compile("\,\d\,")
        for m in p.findall(line):
            # ...in each line
            m = m.replace(',', '')
            # Delete the commas by replacing them with nothing
            list4.append(m)
        # append the ratings to the list in the dictionary
    D4[key] = list4
```

In [29]:

```
# check the last key (to check that the 4th dictionary ends with the 17770 th movie)
last_key4=[]
for k in D4.keys():
    last_key4.append(k)

last_key4[-1]
```

Out[29]:

```
'17770'
```

In [30]:

```
# Transform the dictionary into a dataframe.
# Concatenate all the keys (k) with a dataframe that uses the dictionary values as its argument
# Keep default so that axis=0.  This means movie ID (key) will go in the index instead of the
header
df4=pd.concat({k: pd.DataFrame(v) for k, v in D4.items()}, axis=0)

# Reset the index in the current dataframe. Like when ungrouping a pivot table,
# the former index (movie_ ID and index number) will be put in new columns
# and copied for each row/rating
df4.reset_index(inplace=True)

# The movieId value is in a new column 'level_0'.
# Let's rename it
df4.rename(columns={'level_0':'Movie_Id'}, inplace=True)

# Transform the ratings into numerical values, while renaming the column with the ratings
df4.loc[:,'rating']=pd.to_numeric(df4[0])

# Keep only the columns we need
df4 = df4.loc[:,['Movie_Id','rating']]

# check for missing values
df4.loc[:,['Movie_Id','rating']].isna().sum()
```

Out[30]:

Out[30]:

```
Movie_Id    0
rating      0
dtype: int64
```

In [0]:

```
# drop missing values
df4.dropna(inplace=True)
```

In [0]:

```
# Create a new column with the average ratings
df4.loc[:,'avg_rating'] = df4.loc[:,'rating'].groupby(df4.loc[:,'Movie_Id']).transform('mean')
```

In [33]:

```
# remove the column rating
df4.drop('rating', axis=1,inplace=True)

# remove the duplicates
df4.drop_duplicates(inplace=True)

# reset the index
df4.reset_index(inplace=True)

# keep only the columns we need
df4 = df4.loc[:,['Movie_Id','avg_rating']]
df4.head()
```

Out[33]:

	Movie_Id	avg_rating
0	13368	3.548387
1	13369	3.526667
2	13370	3.760958
3	13371	2.669643
4	13372	2.703911

Combine the 4 Rating dataframes.

In [34]:

```
# ignore_index=True to ignore initial indexes of each DF
rating_df = pd.concat([df1, df2, df3, df4], ignore_index=True)
rating_df.shape
```

Out[34]:

```
(17770, 2)
```

There are 17,770 movies in the final Rating dataframe.

In [35]:

```
#Validating Rating dataframe
rating_df.tail()
```

Out[35]:

	Movie_Id	avg_rating
17765	17766	3.193388
17766	17767	3.671569

	Movie_Id	avg_rating
17767	17768	2.839207
17768	17769	2.498592
17769	17770	2.816504

This 'tail' function confirms that the last movie in the Ratings dataframe is the 17770 th movie.

2.4 - Variable Notes - Ratings dataframe

In [36]:

```
pd.DataFrame(rating_df.dtypes, columns = ['Data Type'])
```

Out[36]:

	Data Type
Movie_Id	object
avg_rating	float64

Movie_Id is stored as a string object (not a numerical value).

avg_rating is stored as a float64 which is a numerical value with a decimal point.

3.0 - Creating the new Feature dataframe

3.1 - Adding features to this dataframe

Create a new dataframe (Feature dataframe) containing features extracted from Movie info dataframe.

In [37]:

```
# Feature dataframe pulls relevant columns from the Movie info dataframe
features_df = movie_info_df.loc[:,
['Movie_Id','Year_of_release','color','director_name','num_critic_for_reviews','duration','director
_facebook_likes',

'actor_1_name','actor_2_name','actor_3_name','actor_1_facebook_likes','actor_2_facebook_likes','act
or_3_facebook_likes',
                                   'cast_total_facebook_likes','movie_facebook_likes','gross','num_
oted_users','facenumber_in_poster','num_user_for_reviews','language','country',
                                   'content_rating','budget','imdb_score']]
#Validating Features dataframe
features_df.head()
```

Out[37]:

	Movie_Id	Year_of_release	color	director_name	num_critic_for_reviews	duration	director_facebook_likes	actor_1_name	actor_2_n
0	3	1997.0	Color	Mike van Diem	54.0	122.0	4.0	Jan Decleir	Fedja
1	24	1981.0	Color	Patrick Lussier	264.0	101.0	71.0	Jensen Ackles	Jaime
2	30	2003.0	Color	Nancy Meyers	145.0	128.0	278.0	Keanu Reeves	Jon Fav
3	45	1999.0	Color	Dan Curtis	NaN	99.0	45.0	Jennifer Jason Leigh	Camp S
5	17721	1998.0	Color	Dan Curtis	NaN	99.0	45.0	Jennifer Jason Leigh	Camp S

As shown in the variable notes for the Movie info dataframe, each movie has been given several 'genres' or 'classifications'. These are given in a single column and are broken up by |

These different 'genres' can be split up into individual columns.

The code below produces a new dataframe which is made up of these new individual 'genre' columns and then joins this to the existing Feature dataframe.

In [38]:

```
# Create a new dataframe with the 'genres' column split by the different genre categories, by spli
tting at and removing the '|' character.
genres_df = movie_info_df.loc[:,'genres'].str.split('|', expand=True)
# Name the new columns "genres"
genres_df.columns = ['genres_1','genres_2','genres_3','genres_4','genres_5','genres_6','genres_7',
'genres_8']
#Join the "genres" dataframe to the features dataframe
features_df = features_df.join(genres_df)
features_df.head()
```

Out[38]:

	Movie_Id	Year_of_release	color	director_name	num_critic_for_reviews	duration	director_facebook_likes	actor_1_name	actor_2_n
0	3	1997.0	Color	Mike van Diem	54.0	122.0	4.0	Jan Decleir	Fedja
1	24	1981.0	Color	Patrick Lussier	264.0	101.0	71.0	Jensen Ackles	Jaime
2	30	2003.0	Color	Nancy Meyers	145.0	128.0	278.0	Keanu Reeves	Jon Fav
3	45	1999.0	Color	Dan Curtis	NaN	99.0	45.0	Jennifer Jason Leigh	Camp
5	17721	1998.0	Color	Dan Curtis	NaN	99.0	45.0	Jennifer Jason Leigh	Camp

As shown in the variable notes for the Movie info dataframe, each movie has been given several 'plot_keywords'. These are given in a single column and are broken up by |

These different 'plot_keywords' can be split up into individual columns.

The code below produces a new dataframe which is made up of these new individual 'plot_keywords' columns and then joins this to the existing Feature dataframe.

In [39]:

```
# Create a new dataframe with the'plot_keywords' column split by the different words, by splitting
at and removing the '|' character.
plot_keywords_df = movie_info_df.loc[:,'plot_keywords'].str.split('|', expand=True)
# Name the new columns "plot_keywords"
plot_keywords_df.columns =
['plot_keywords_1','plot_keywords_2','plot_keywords_3','plot_keywords_4','plot_keywords_5']
#Join the "plot_keywords" dataframe to the features dataframe
features_df = features_df.join(plot_keywords_df)
features_df.head()
```

Out[39]:

	Movie_Id	Year_of_release	color	director_name	num_critic_for_reviews	duration	director_facebook_likes	actor_1_name	actor_2_n
0	3	1997.0	Color	Mike van Diem	54.0	122.0	4.0	Jan Decleir	Fedja
1	24	1981.0	Color	Patrick Lussier	264.0	101.0	71.0	Jensen Ackles	Jaime
2	30	2003.0	Color	Nancy Meyers	145.0	128.0	278.0	Keanu Reeves	Jon Fav
3	45	1999.0	Color	Dan Curtis	NaN	99.0	45.0	Jennifer Jason Leigh	Camp
5	17721	1998.0	Color	Dan Curtis	NaN	99.0	45.0	Jennifer Jason Leigh	Camp

3.2 - Variable Notes - Feature dataframe

As before, types are narrowed down to Numeric, Categorical and Boolean.

As before, types are narrowed down to Numeric, Categorical and Boolean.

Input data	Definition	Category
Movie_Id	Number of movie	Numeric
Year_of_release	Production Year	Numeric
color	Boolean Variable of color used	Boolean
director_name	Name of the Director of the Movie	Categorical
num_critic_for_reviews	Number of critical evaluation reviews on movie	Numeric
duration	Length of the movie	Numeric
director_facebook_likes	Number of likes of the director on his/her Facebook Page	Numeric
actor_1_name	Primary actor starring in the movie	Categorical
actor_2_name	Secondary actor starring in the movie	Categorical
actor_3_name	Tertiary actor starring in the movie	Categorical
actor_1_facebook_likes	Number of likes of the actor_1 on his/her Facebook Page	Numeric
actor_2_facebook_likes	Number of likes of the actor_2 on his/her Facebook Page	Numeric
actor_3_facebook_likes	Number of likes of the actor_3 on his/her Facebook Page	Numeric
cast_total_facebook_likes	Number of all actor likes on Facebook	Numeric
movie_facebook_likes	Number of movie likes on Facebook	Numeric
gross	Earnings of movie in U.S. dollars	Numeric
num_voted_users	Number of people voting on movie	Numeric
facenumber_in_poster	Number of actor faces on movie poster	Numeric
num_user_for_reviews	Number of people reviewing movie	Numeric
language	Movie language	Categorical
country	Production Countries	Categorical
content_rating	MPAA movie ratings	Categorical
budget	Amount of cost of production in U.S. dollars	Numeric
imdb_score	Average score collected by IMDB	Numeric
genres	Classifications of the movie	Categorical
plot_keywords	Words or phrases that are searchable for specific movie	Categorical

3.3 - Joining the Feature dataframe with Ratings dataframe

These dataframes are joined to eliminate any movies that aren't common to both. The X and y dataframes (which will be used for the predictive analysis) are developed from this joint dataframe.

Let us have a look first at the shapes of the "features" and "rating" dataframe before joining them.

In [40]:

```
#shape of the 2 dataframes
print(features_df.shape)
print(rating_df.shape)
```

```
(2299, 37)
(17770, 2)
```

Since "Movie_ID" is an object in the Ratings dataframe and a float in the Feature dataframe we need to convert both into integers (int64).

In [0]:

```
rating_df['Movie_Id']=rating_df['Movie_Id'].astype('int64')
features_df['Movie_Id']=features_df['Movie_Id'].astype('int64')
```

In [0]:

```
# Rating and Features dataframes are merged on movieID
# Inner join is used to ensure that only movies that have their movieID in BOTH dataframes are included in final dataframe
# Final dataframe is called joined_DF
joined_DF = features_df.merge(rating_df, how='inner', on='Movie_Id')

# X dataframe includes all columns EXCEPT for avg_rating
X = joined_DF.loc[:,[col for col in joined_DF.columns if col != 'avg_rating']]
# y dataframe ONLY includes avg_rating
y = joined_DF.loc[:,'avg_rating']
```

In [43]:

```
# check the shape of the 2 dataframes
print(X.shape)
print(y.shape)
```

```
(2299, 37)
(2299,)
```

In the final X and y dataframes there are 2299 movies.

3.4 - Removing Duplicate Rows

Because duplicate rows (identical rows) will introduce bias when analysing the data, they must be deleted.

In [44]:

```
# Look at the shape before dropping
print(X.shape)
# Drop duplicate rows
X.drop_duplicates(keep = 'first', inplace = True)
#Looking at the shape after dropping
print(X.shape)
```

```
(2299, 37)
(2299, 37)
```

From the above result, we can see there are no duplicate rows in the dataset.

3.5 - Removing Missing Values in the y Dataset

We shouldn't impute values for a missing dependent variable because we are using this as the target in the predictive model.

Therefore, we need to check the y dataframe for missing values and if there are any, we must remove the value PLUS the corresponding row in the X dataframe.

In [45]:

```
# Count number of missing values
'Number of Missing Values : ' + str(y.isna().sum())
```

Out[45]:

```
'Number of Missing Values : 0'
```

There are no missing values in the y dataframe.

4.0 - Data Splitting

The X and y dataframes need to be split into train and test data (for training and testing the predictive model).

In [0]:

```
# Defining a random_state will ensure we have the same split and the same result every time we run the notebook
# we want the test set to be 30% of the total (and train to be 70%)
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=2020)
```

In [47]:

```
#shape of the 4 dataframes
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(1609, 37)
(690, 37)
(1609,)
(690,)
```

5.0 - Descriptive Stats on Training Datasets

In this section, we investigate the variables of the dataframes one at a time. The analysis is separated into dependent variable, numeric independent variables and categorical independent variables. A selection of these variables will be used in the predictive model so the descriptive statistics are an important way to improve our understanding of them.

Histograms and box plots are given for each of the numerical variables to illustrate their skewness and to visualise any outliers. These outliers will be dealt with later in the assignment.

Bar charts are given for each of the categorical variables to illustrate their value counts.

We do not want to do any 'data snooping' as this would cause bias in model development. Therefore, all analysis in this section is done on the x_train and y_train dataframes.

5.1 - Dependent Variable

Visualise the first few rows of the dependent variable (avg_rating).

In [48]:

```
pd.DataFrame(y_train.head(3))
```

Out[48]:

	avg_rating
1612	3.308080
1463	3.078547
2268	3.357953

In [49]:

```
# descriptive stats
pd.DataFrame(y_train.describe())
```

Out[49]:

	avg_rating
count	1609.000000
mean	3.336607
std	0.418645

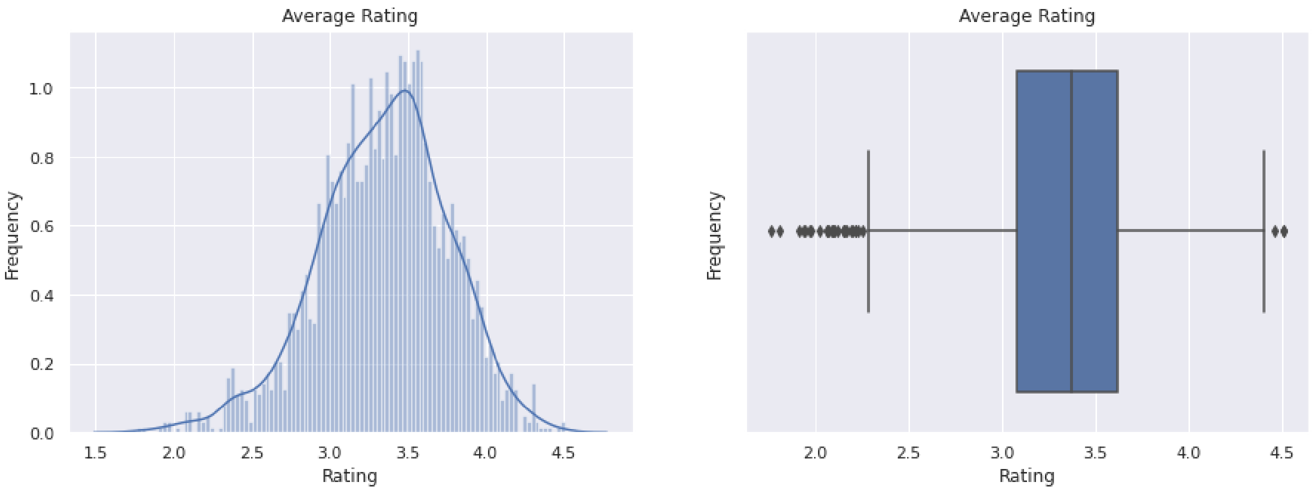
	avg_rating
min	1.762083
25%	3.073668
50%	3.360873
75%	3.609756
max	4.504035

The mean average rating is 3.34, with a standard deviation of 0.42. The range of the average rating is between 1.76 and 4.50 (this inevitably falls between the actual range of netflix ratings of 1 to 5).

To illustrate the skewness of the dependent variable and identify outliers, we plot the distribution with a histogram and box-plot.

In [50]:

```
fig = plt.figure(figsize = (15,5))

#Histogram
plt.subplot(1,2,1)
#Define plot object
hist = sns.distplot(y, bins = 100)
#Setting graph title
hist.set_title('Average Rating')
hist.set(xlabel = 'Rating', ylabel = 'Frequency')

#Boxplot
plt.subplot(1,2,2)
#Define plot object
box = sns.boxplot(y)
#Setting graph title
box.set_title('Average Rating')
box.set(xlabel = 'Rating', ylabel = 'Frequency')

#Showing the plot
plt.show()
```



The variable is fairly normally distributed. The outliers are at the low end of the box plot.

5.2 - Numeric Independent Variables

The following columns are Numeric Independent Variables and are investigated below.

- Year_of_release
- num_critic_for_reviews
- duration
- director_facebook_likes
- actor_1_facebook_likes
- actor_2_facebook_likes
- actor_3_facebook_likes

- cast_total_facebook_likes
- movie_facebook_likes
- gross
- num_voted_users
- facenumber_in_poster
- num_user_for_reviews
- budget
- imdb_score

5.2.1 - Year of release

Descriptive statistics for Year of release.

In [51]:

```
pd.DataFrame(x_train.loc[:,'Year_of_release'].describe())
```

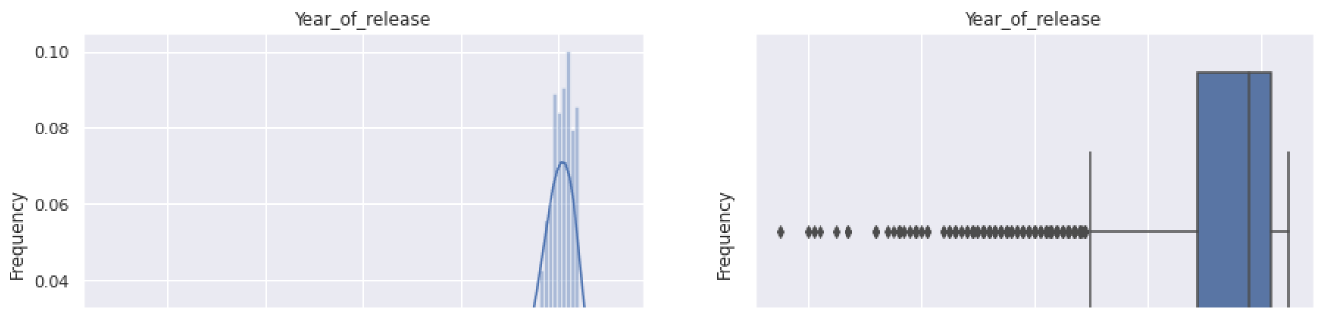
Out[51]:

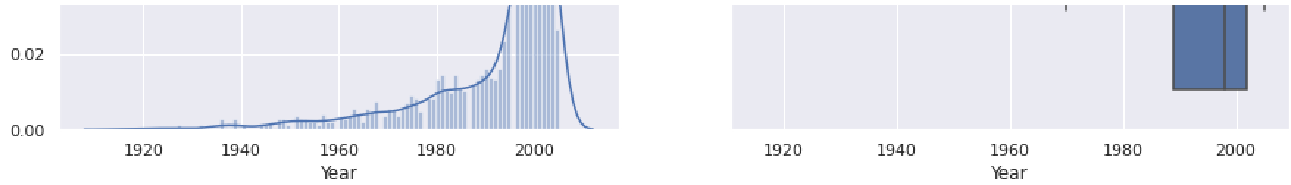
	Year_of_release
count	1609.000000
mean	1992.630205
std	14.472245
min	1915.000000
25%	1989.000000
50%	1998.000000
75%	2002.000000
max	2005.000000

The distribution and skewness of this variable are illustrated by the histogram and box plot below.

In [52]:

```
fig = plt.figure(figsize = (15,5))

#Histogram
plt.subplot(1,2,1)
#Define plot object
hist = sns.distplot(x_train.loc[:,'Year_of_release'].astype(float), bins = 100)
#Setting graph title
hist.set_title( 'Year_of_release')
hist.set(xlabel = 'Year', ylabel = 'Frequency')

#Boxplot
plt.subplot(1,2,2)
#Define plot object
box = sns.boxplot(x_train.loc[:,'Year_of_release'].astype(float))
#Setting graph title
box.set_title( 'Year_of_release')
box.set(xlabel = 'Year', ylabel = 'Frequency')

#Showing the plot
plt.show()
```





The majority of films in this dataset were released in the 1990s. This is demonstrated by the mean and median values which are 1993 and 1998 respectively. The earliest and latest release dates are 1915 and 2005 respectively. The outliers are all the movies that were released before 1970.

5.2.2 - Number of critic reviews

Descriptive statistics for Number of critic reviews.

In [53]:

```
pd.DataFrame(x_train.loc[:,'num_critic_for_reviews'].describe())
```

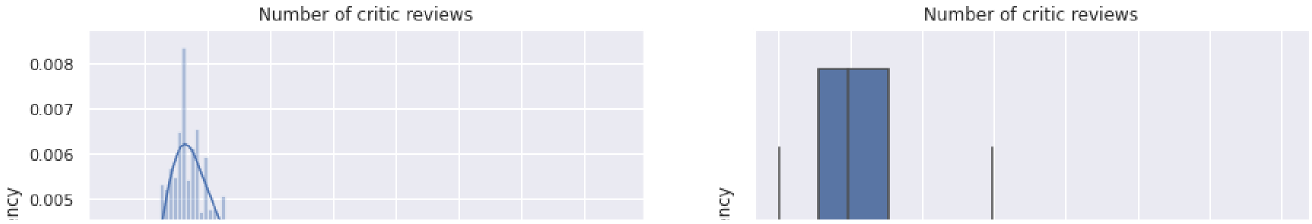
Out[53]:

	num_critic_for_reviews
count	1606.000000
mean	116.898506
std	87.063018
min	1.000000
25%	56.000000
50%	96.000000
75%	153.000000
max	703.000000

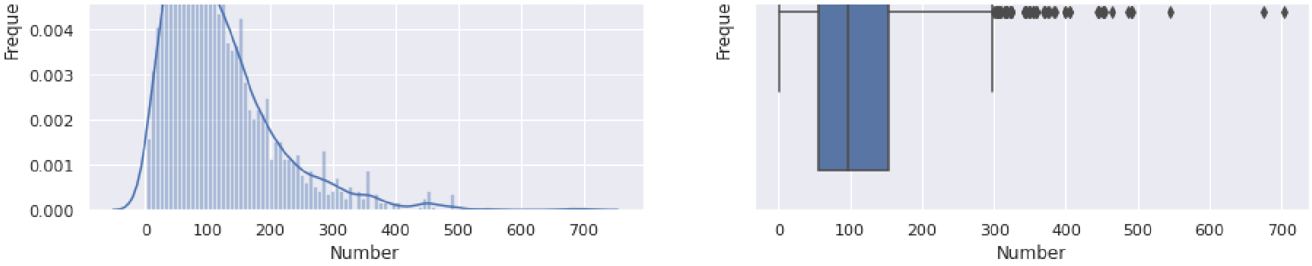
The distribution and skewness of this variable are illustrated by the histogram and box plot below.

In [54]:

```
fig = plt.figure(figsize = (15,5))

#Histogram
plt.subplot(1,2,1)
#Define plot object
hist = sns.distplot(x_train.loc[:,'num_critic_for_reviews'].astype(float), bins = 100)
#Setting graph title
hist.set_title( 'Number of critic reviews')
hist.set(xlabel = 'Number', ylabel = 'Frequency')

#Boxplot
plt.subplot(1,2,2)
#Define plot object
box = sns.boxplot(x_train.loc[:,'num_critic_for_reviews'].astype(float))
#Setting graph title
box.set_title( 'Number of critic reviews')
box.set(xlabel = 'Number', ylabel = 'Frequency')

#Showing the plot
plt.show()
```


Number of critic reviews

Number of critic reviews



The negative skew of the histogram and box plot shows that the majority of movies have received between 1 and 150 critical evaluation reviews. The mean number of reviews is 116.90 and the most reviews for a single movie is 703. The outliers are movies that received over 300 reviews.

5.2.3 - Duration

The descriptive statistics for Duration.

In [55]:

```
pd.DataFrame(x_train.loc[:,'duration'].describe())
```

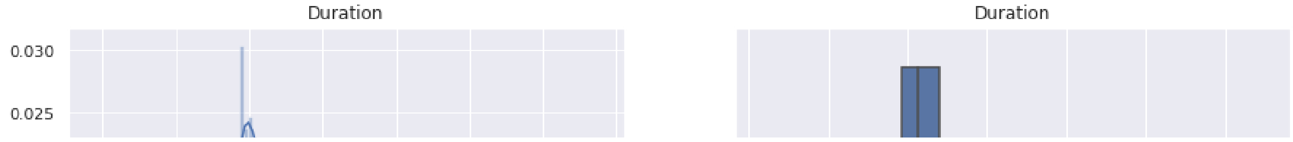
Out[55]:

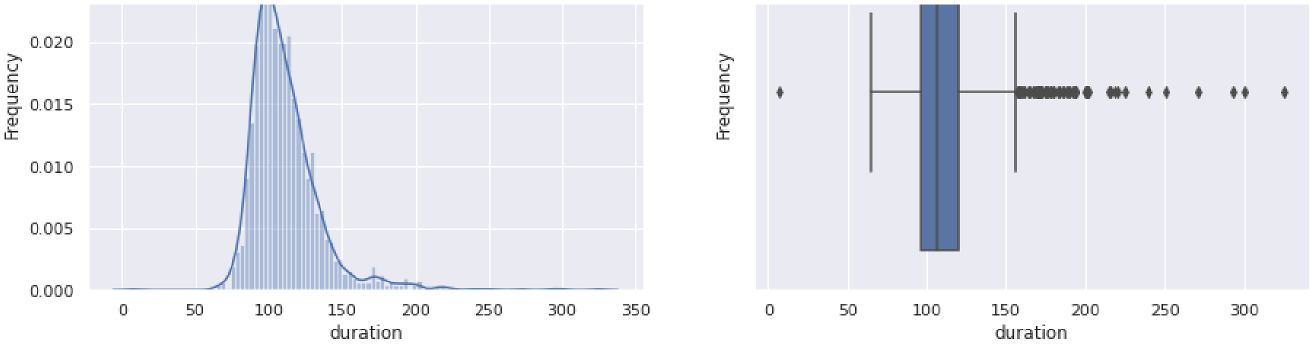
	duration
count	1608.000000
mean	110.949627
std	23.916143
min	7.000000
25%	96.000000
50%	106.000000
75%	120.000000
max	325.000000

The distribution and skewness of this variable are illustrated by the histogram and box plot below.

In [56]:

```
fig = plt.figure(figsize = (15,5))

#Histogram
plt.subplot(1,2,1)
#Define plot object
hist = sns.distplot(x_train.loc[:,'duration'].astype(float), bins = 100)
#Setting graph title
hist.set_title( 'Duration')
hist.set(xlabel = 'duration', ylabel = 'Frequency')

#Boxplot
plt.subplot(1,2,2)
#Define plot object
box = sns.boxplot(x_train.loc[:,'duration'].astype(float))
#Setting graph title
box.set_title( 'Duration')
box.set(xlabel = 'duration', ylabel = 'Frequency')

#Showing the plot
plt.show()
```





On average, the movies are 111 minutes long and only 25% of movies have a duration longer than 120 minutes. The outliers have a duration longer than roughly 155 minutes.

5.2.4 - Director' and actors' Likes

The descriptive statistics for director_facebook_likes.

In [57]:

```
pd.DataFrame(x_train.loc[:,'director_facebook_likes'].describe())
```

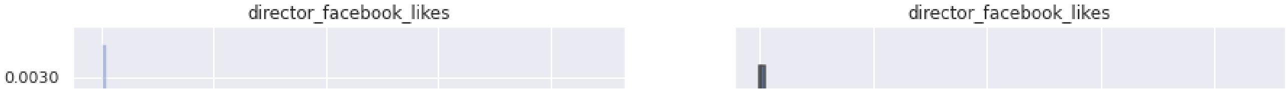
Out[57]:

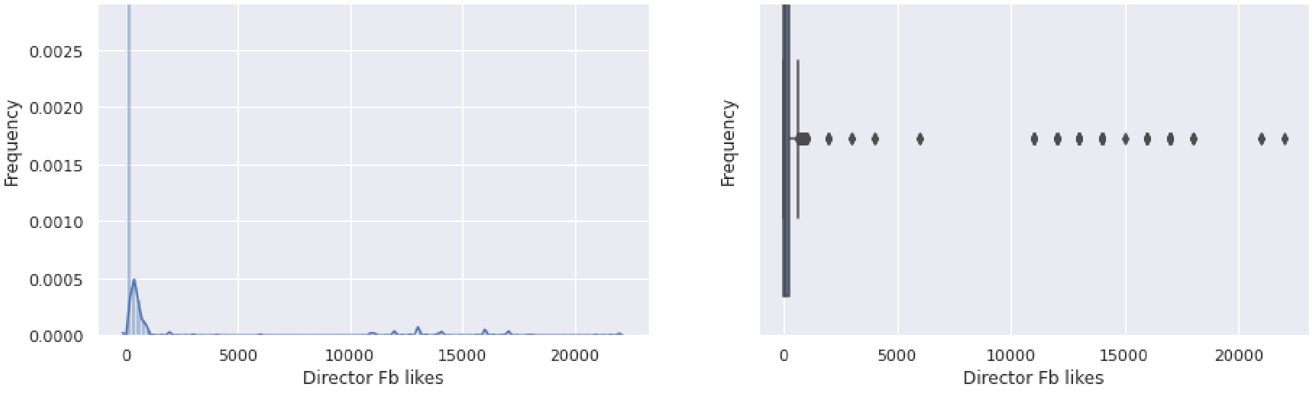
	director_facebook_likes
count	1609.000000
mean	848.326911
std	3099.060854
min	0.000000
25%	10.000000
50%	58.000000
75%	272.000000
max	22000.000000

The distribution and skewness of this variable are illustrated by the histogram and box plot below.

In [58]:

```
fig = plt.figure(figsize = (15,5))

#Histogram
plt.subplot(1,2,1)
#Define plot object
hist = sns.distplot(x_train.loc[:,'director_facebook_likes'].astype(float), bins = 100)
#Setting graph title
hist.set_title('director_facebook_likes')
hist.set(xlabel = 'Director Fb likes', ylabel = 'Frequency')

#Boxplot
plt.subplot(1,2,2)
#Define plot object
box = sns.boxplot(x_train.loc[:,'director_facebook_likes'].astype(float))
#Setting graph title
box.set_title( 'director_facebook_likes')
box.set(xlabel = 'Director Fb likes', ylabel = 'Frequency')

#Showing the plot
plt.show()
```





Most Directors have a low number of Facebook likes and we have several outliers. It makes sense because not many people know who directed a movie and there are only a few very famous directors.

The descriptive stats for actor_1_facebook_likes.

In [59]:

```
pd.DataFrame(x_train.loc[:,'actor_1_facebook_likes'].describe())
```

Out[59]:

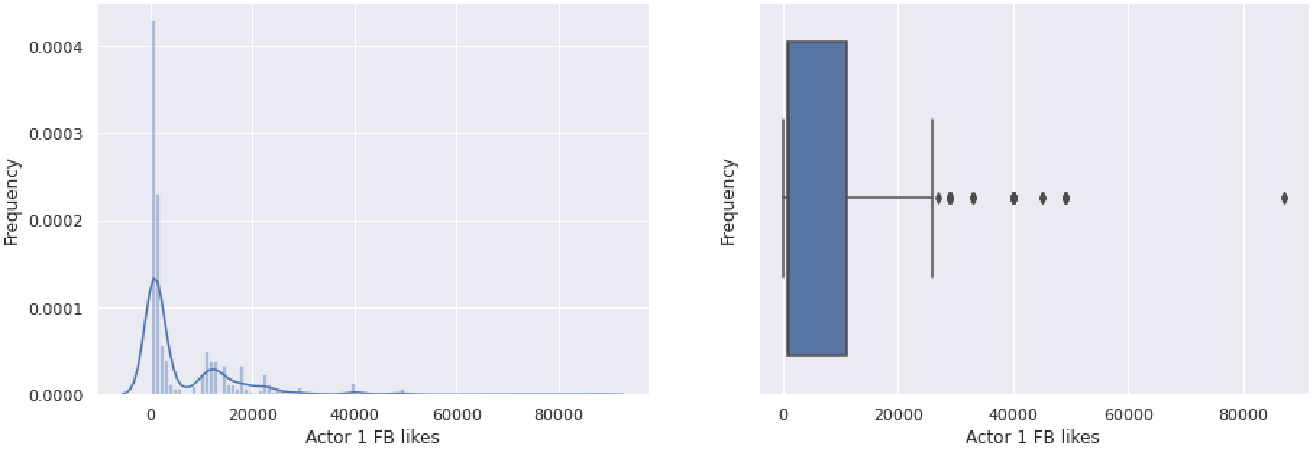
	actor_1_facebook_likes
count	1608.000000
mean	6256.871891
std	9350.890571
min	0.000000
25%	696.500000
50%	1000.000000
75%	11000.000000
max	87000.000000

The first actor with the biggest number of Facebook likes has "only" 87,000 likes. Since our data dates back from the 2000's and Facebook was launched in 2004, this makes sense.

The distribution and skewness of this variable are illustrated by the histogram and box plot below.

In [60]:

```
fig = plt.figure(figsize = (15,5))

#Histogram
plt.subplot(1,2,1)
#Define plot object
hist = sns.distplot(x_train.loc[:,'actor_1_facebook_likes'].astype(float), bins = 100)
#Setting graph title
hist.set_title( 'actor_1_facebook_likes')
hist.set(xlabel = 'Actor 1 FB likes', ylabel = 'Frequency')

#Boxplot
plt.subplot(1,2,2)
#Define plot object
box = sns.boxplot(x_train.loc[:,'actor_1_facebook_likes'].astype(float))
#Setting graph title
box.set_title( 'actor_1_facebook_likes')
box.set(xlabel = 'Actor 1 FB likes', ylabel = 'Frequency')

#Showing the plot
plt.show()
```



There are a few outliers, which represent the biggest movie stars, who set up a Facebook account in the early 2000's.

The descriptive stats for actor_2_facebook_likes.

```
In [61]:
```

```
pd.DataFrame(x_train.loc[:,'actor_2_facebook_likes'].describe())
```

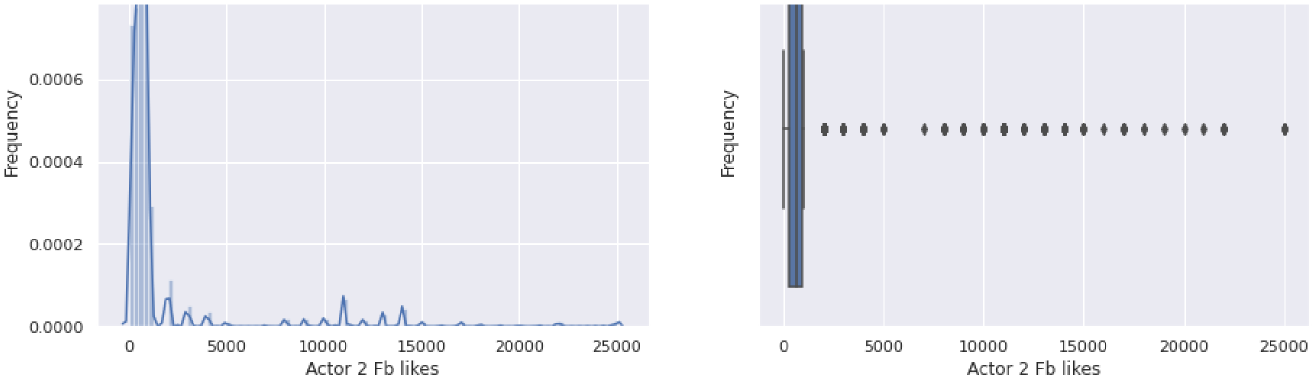
```
Out[61]:
```

	actor_2_facebook_likes
count	1608.000000
mean	1519.351368
std	3375.600693
min	0.000000
25%	329.750000
50%	629.000000
75%	918.250000
max	25000.000000

The distribution and skewness of this variable are illustrated by the histogram and box plot below.

```
In [62]:
```

```
fig = plt.figure(figsize = (15,5))

#Histogram
plt.subplot(1,2,1)
#Define plot object
hist = sns.distplot(x_train.loc[:,'actor_2_facebook_likes'].astype(float), bins = 100)
#Setting graph title
hist.set_title( 'actor_2_facebook_likes')
hist.set(xlabel = 'Actor 2 Fb likes', ylabel = 'Frequency')

#Boxplot
plt.subplot(1,2,2)
#Define plot object
box = sns.boxplot(x_train.loc[:,'actor_2_facebook_likes'].astype(float))
#Setting graph title
box.set_title( 'actor_2_facebook_likes')
box.set(xlabel = 'Actor 2 Fb likes', ylabel = 'Frequency')

#Showing the plot
plt.show()
```



Most second lead actors have low numbers of facebook likes. Their average number of likes is lower than the lead actors' because, most of the time, they are less famous. There are more outliers than for the first lead, which suggests that a few famous actors, with many Facebook likes, played second roles. This may have been done to increase the 'attractiveness' of the movie.

The descriptive stats for actor_3_facebook_likes.

In [63]:

```
pd.DataFrame(x_train.loc[:,'actor_3_facebook_likes'].describe())
```

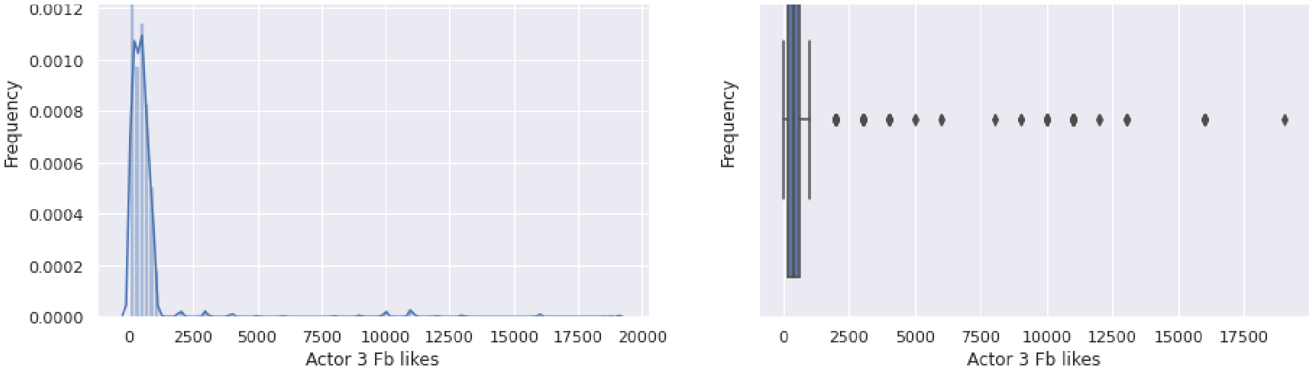
Out[63]:

	actor_3_facebook_likes
count	1605.000000
mean	624.240498
std	1498.974782
min	0.000000
25%	167.000000
50%	407.000000
75%	636.000000
max	19000.000000

The distribution and skewness of this variable are illustrated by the histogram and box plot below.

In [64]:

```
fig = plt.figure(figsize = (15,5))

#Histogram
plt.subplot(1,2,1)
#Define plot object
hist = sns.distplot(x_train.loc[:,'actor_3_facebook_likes'].astype(float), bins = 100)
#Setting graph title
hist.set_title( 'actor_3_facebook_likes')
hist.set(xlabel = 'Actor 3 Fb likes', ylabel = 'Frequency')

#Boxplot
plt.subplot(1,2,2)
#Define plot object
box = sns.boxplot(x_train.loc[:,'actor_3_facebook_likes'].astype(float))
#Setting graph title
box.set_title( 'actor_3_facebook_likes')
box.set(xlabel = 'Actor 3 Fb likes', ylabel = 'Frequency')

#Showing the plot
plt.show()
```



The third lead actors have even lower Facebook likes, which makes sense because they are less well-known than actor_1 or actor_2. There are a few outliers which suggests that a few famous actors, with many Facebook likes, played third roles.

5.2.5 - Cast total Facebook Likes

Descriptive statistics for Cast total Facebook likes.

In [65]:

```
pd.DataFrame(x_train.loc[:,'cast_total_facebook_likes'].describe())
```

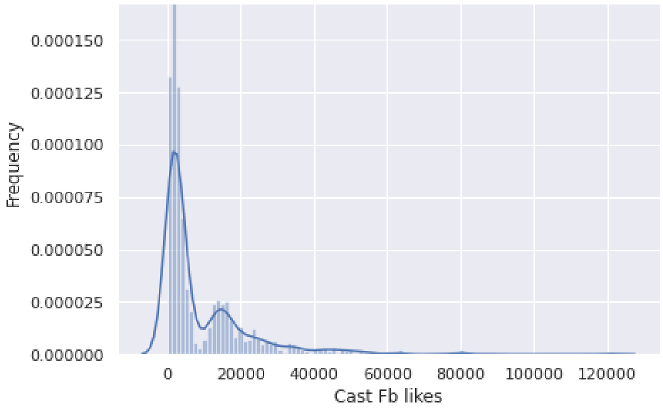
Out[65]:

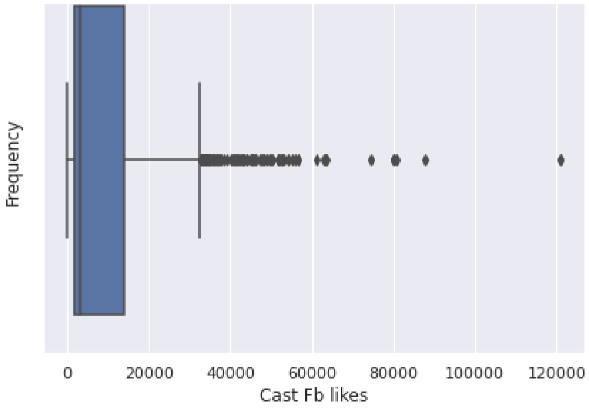
	cast_total_facebook_likes
count	1609.000000
mean	9275.150404
std	12842.900112
min	0.000000
25%	1679.000000
50%	3306.000000
75%	14087.000000
max	120797.000000

The distribution and skewness of this variable are illustrated by the histogram and box plot below.

In [66]:

```
fig = plt.figure(figsize = (15,5))

#Histogram
plt.subplot(1,2,1)
#Define plot object
hist = sns.distplot(x_train.loc[:,'cast_total_facebook_likes'].astype(float), bins = 100)
#Setting graph title
hist.set_title('cast_total_facebook_likes')
hist.set(xlabel = 'Cast Fb likes', ylabel = 'Frequency')

#Boxplot
plt.subplot(1,2,2)
#Define plot object
box = sns.boxplot(x_train.loc[:,'cast_total_facebook_likes'].astype(float))
#Setting graph title
box.set_title('cast_total_facebook_likes')
box.set(xlabel = 'Cast Fb likes', ylabel = 'Frequency')

#Showing the plot
plt.show()
```





There seems to be three types of movies - (1) movies with a cast with low Facebook likes, (2) movies with a cast with an average number of Facebook likes (mean is 9275 likes) and (3) a few movies with a famous cast, that have many Facebook likes (these are the outliers).

The more famous actors there are in a movie, the higher its budget. That is why there are not a lot of movies with a high number of Facebook likes.

Our Netflix dataset dates back to 2005 with many movies from the 2000's and 1900's. Facebook was only launched in 2004, which explains why the Facebook likes of the actors and director are low. Nowadays, 15 years later, the number of Facebooks likes for actors and directors would be much higher.

5.2.6 - Movie Facebook Likes

Descriptive statistics for Movie Facebook Likes.

In [67]:

```
pd.DataFrame(x_train.loc[:,'movie_facebook_likes'].describe())
```

Out[67]:

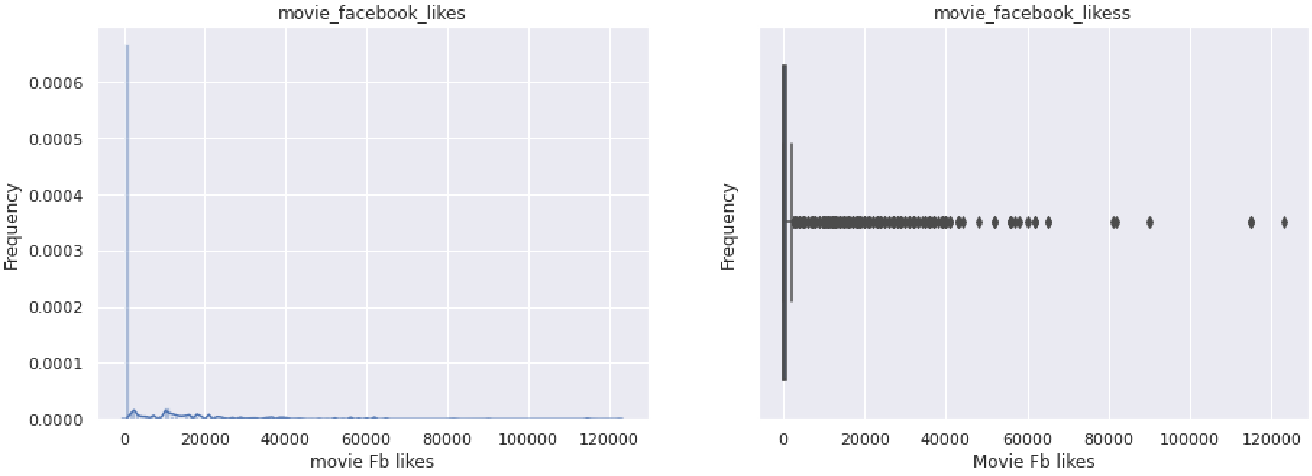
	movie_facebook_likes
count	1609.000000
mean	3647.137352
std	10744.745716
min	0.000000
25%	0.000000
50%	0.000000
75%	877.000000
max	123000.000000

The distribution and skewness of this variable are illustrated by the histogram and box plot below.

In [68]:

```
fig = plt.figure(figsize = (15,5))

#Histogram
plt.subplot(1,2,1)
#Define plot object
hist = sns.distplot(x_train.loc[:,'movie_facebook_likes'].astype(float), bins = 100)
#Setting graph title
hist.set_title('movie_facebook_likes')
hist.set(xlabel = 'movie Fb likes', ylabel = 'Frequency')

#Boxplot
plt.subplot(1,2,2)
#Define plot object
box = sns.boxplot(x_train.loc[:,'movie_facebook_likes'].astype(float))
```

```
#Setting graph title
box.set_title('movie_facebook_likess')
box.set(xlabel = 'Movie Fb likes', ylabel = 'Frequency')

#Showing the plot
plt.show()
```



Most movies have either no or a very low number of Facebook likes because Facebook was new in the 2000's. There are a few movies (outliers) with a higher number of likes.

5.2.7 - Gross

Descriptive statistics of Gross.

In [69]:

```
pd.DataFrame(x_train.loc[:,'gross'].describe())
```

Out[69]:

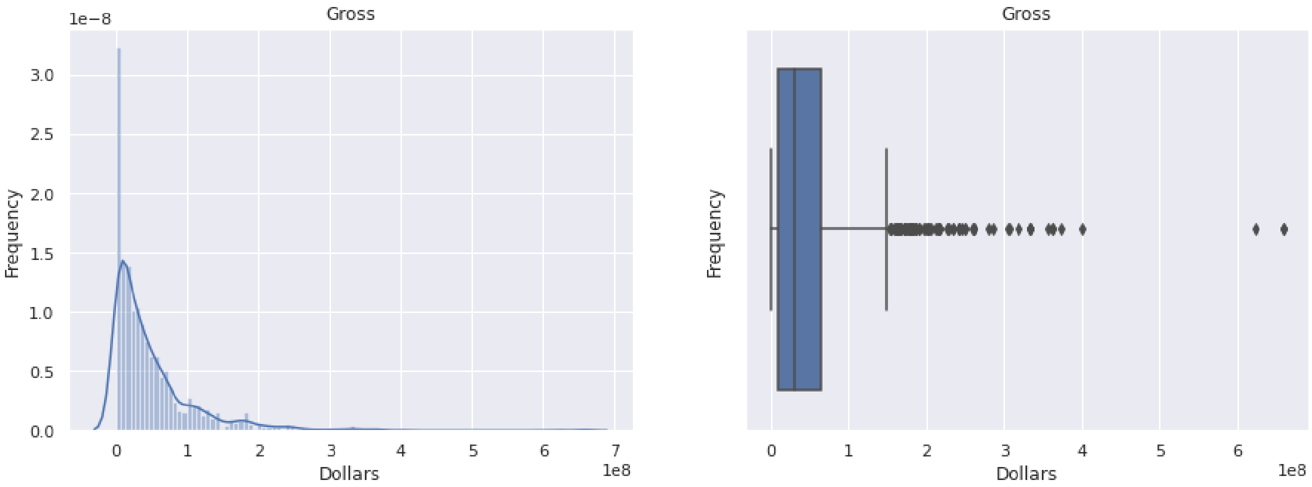
	gross
count	1.440000e+03
mean	4.942434e+07
std	6.376728e+07
min	7.030000e+02
25%	9.158819e+06
50%	3.003169e+07
75%	6.500000e+07
max	6.586723e+08

The distribution and skewness of this variable are illustrated by the histogram and box plot below.

In [70]:

```
fig = plt.figure(figsize = (15,5))

#Histogram
plt.subplot(1,2,1)
#Define plot object
hist = sns.distplot(x_train.loc[:,'gross'].astype(float), bins = 100)
#Setting graph title
hist.set_title('Gross')
hist.set(xlabel = 'Dollars', ylabel = 'Frequency')

#Boxplot
plt.subplot(1,2,2)
#Define plot object
```

```
#Define plot object
box = sns.boxplot(x_train.loc[:,'gross'].astype(float))
#Setting graph title
box.set_title('Gross')
box.set(xlabel = 'Dollars', ylabel = 'Frequency')

#Showing the plot
plt.show()
```



The distribution of Gross is positively skewed but we cannot gain any insight. This is because there are very extreme values (outliers) in the dataset, which heavily skew any exploration or prediction.

5.2.8 - Number of Voted Users (voters)

Descriptive statistics.

In [71]:

```
pd.DataFrame(x_train.loc[:,'num_voted_users'].describe())
```

Out[71]:

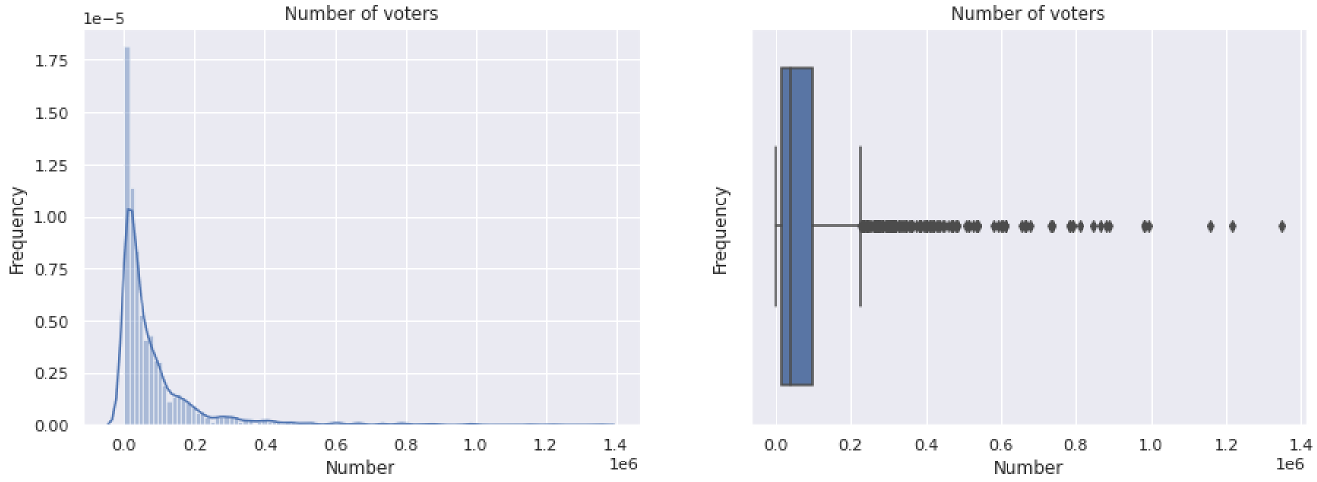
	num_voted_users
count	1.609000e+03
mean	8.637662e+04
std	1.359876e+05
min	1.180000e+02
25%	1.419300e+04
50%	3.965900e+04
75%	9.898900e+04
max	1.347461e+06

The distribution and skewness of this variable are illustrated by the histogram and box plot below.

In [72]:

```
fig = plt.figure(figsize = (15,5))

#Histogram
plt.subplot(1,2,1)
#Define plot object
hist = sns.distplot(x_train.loc[:,'num_voted_users'].astype(float), bins = 100)
#Setting graph title
hist.set_title('Number of voters')
hist.set(xlabel = 'Number', ylabel = 'Frequency')

#Boxplot
```

```
#Boxplot
plt.subplot(1,2,2)
#Define plot object
box = sns.boxplot(x_train.loc[:,'num_voted_users'].astype(float))
#Setting graph title
box.set_title('Number of voters')
box.set(xlabel = 'Number', ylabel = 'Frequency')

#Showing the plot
plt.show()
```



The distribution of number of voters is positively skewed. On average, the movie received 8.637662e+04 votes by different users. The maximum number of voters was 1.347461e+06 and there are several large outliers.

5.2.9 - Number of faces in poster

Descriptive statistics of Number of faces in poster.

In [73]:

```
pd.DataFrame(x_train.loc[:,'facenumber_in_poster'].describe())
```

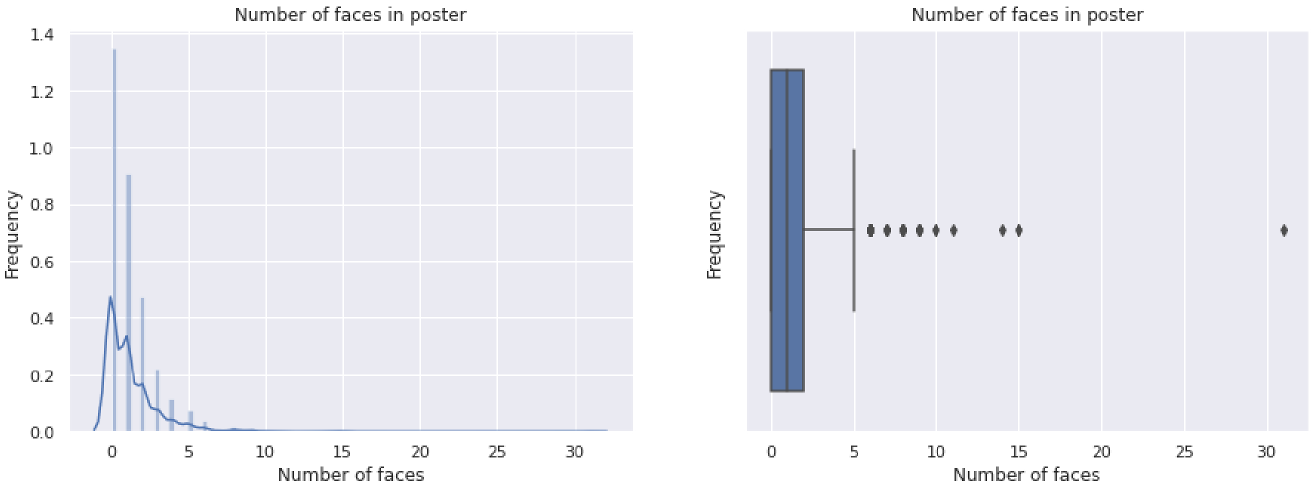
Out[73]:

	facenumber_in_poster
count	1608.000000
mean	1.262438
std	1.841114
min	0.000000
25%	0.000000
50%	1.000000
75%	2.000000
max	31.000000

The distribution and skewness of this variable are illustrated by the histogram and box plot below.

In [74]:

```
fig = plt.figure(figsize = (15,5))

#Histogram
plt.subplot(1,2,1)
#Define plot object
hist = sns.distplot(x_train.loc[:,'facenumber_in_poster'].astype(float), bins = 100)
#Setting graph title
hist.set_title('Number of faces in poster')
hist.set(xlabel = 'Number of faces', ylabel = 'Frequency')
hist.set(xlabel = 'Number of faces', ylabel = 'Frequency')

#Boxplot
plt.subplot(1,2,2)
#Define plot object
box = sns.boxplot(x_train.loc[:,'facenumber_in_poster'].astype(float))
#Setting graph title
box.set_title('Number of faces in poster')
box.set(xlabel = 'Number of faces', ylabel = 'Frequency')

#Showing the plot
plt.show()
```



The distribution of number of faces is positively skewed. Most movies have a face count of around 1 (and the mean is 1).

The maximum face count on a poster is 31. The outliers are movies that have a face count more than 5.

5.2.10 - Number of Reviewers

Descriptive statistics of Number of reviewers.

In [75]:

```
pd.DataFrame(x_train.loc[:,'num_user_for_reviews'].describe())
```

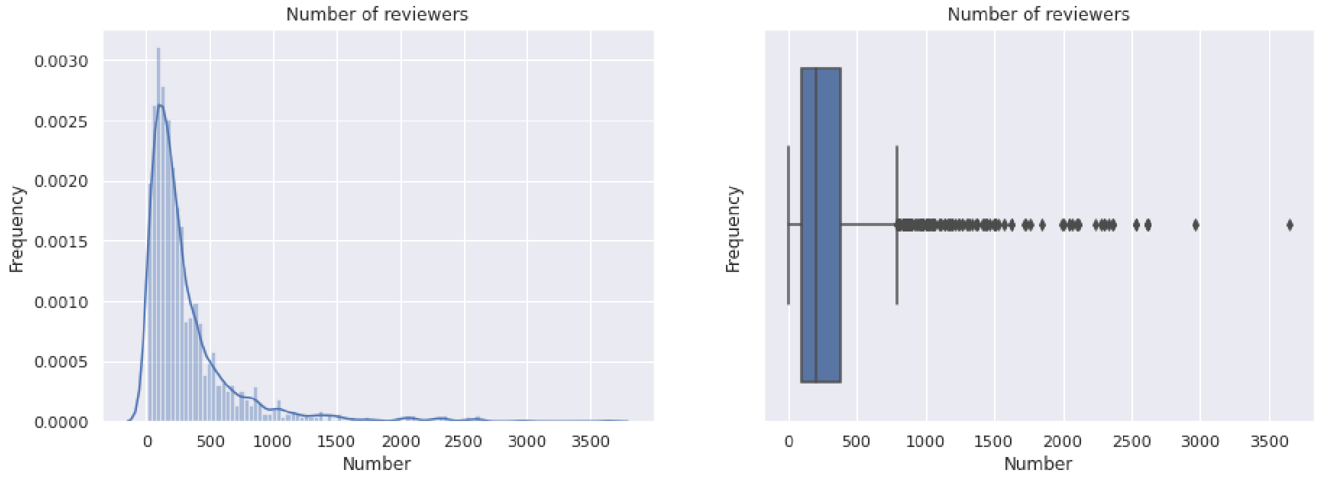
Out[75]:

	num_user_for_reviews
count	1608.000000
mean	316.224502
std	376.239011
min	3.000000
25%	101.000000
50%	199.000000
75%	380.000000
max	3646.000000

The distribution and skewness of this variable are illustrated by the histogram and box plot below.

In [76]:

```
fig = plt.figure(figsize = (15,5))

#Histogram
plt.subplot(1,2,1)
#Define plot object
hist = sns.distplot(x_train.loc[:,'num_user_for_reviews'].astype(float), bins = 100)
hist = sns.distplot(x_train.loc[:,'num_user_for_reviews'].astype(float), bins = 100)
#Setting graph title
hist.set_title('Number of reviewers')
hist.set(xlabel = 'Number', ylabel = 'Frequency')

#Boxplot
plt.subplot(1,2,2)
#Define plot object
box = sns.boxplot(x_train.loc[:,'num_user_for_reviews'].astype(float))
#Setting graph title
box.set_title('Number of reviewers')
box.set(xlabel = 'Number', ylabel = 'Frequency')

#Showing the plot
plt.show()
```



The distribution shows that (1) the number of reviewers is positively skewed and (2) the majority of movies have received between 50 and 300 reviews. The mean number is 316 and the maximum is 3646 (there are several large outliers).

5.2.11 - Budget

Descriptive statistics of Budget.

In [77]:

```
pd.DataFrame(x_train.loc[:,'budget'].describe())
```

Out[77]:

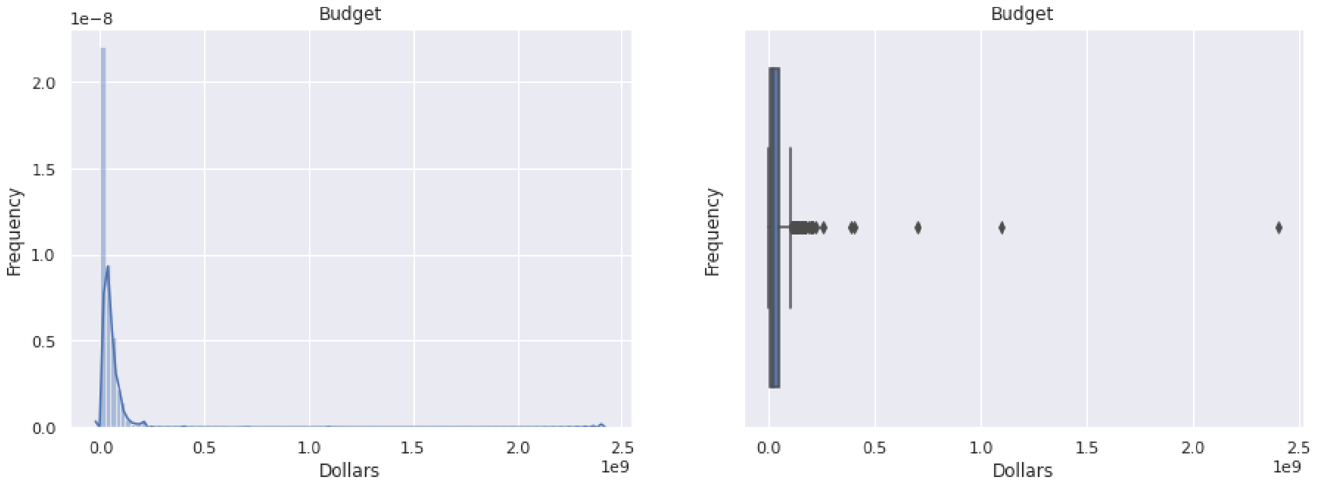
	budget
count	1.534000e+03
mean	3.629642e+07
std	7.831891e+07
min	2.180000e+02
25%	8.000000e+06
50%	2.200000e+07
75%	4.800000e+07
max	2.400000e+09

The distribution and skewness of this variable are illustrated by the histogram and box plot below.

In [78]:

```
fig = plt.figure(figsize = (15,5))

#Histogram
plt.subplot(1,2,1)
```

```
#Define plot object
hist = sns.distplot(x_train.loc[:,'budget'].astype(float), bins = 100)
#Setting graph title
hist.set_title( 'Budget')
hist.set(xlabel = 'Dollars', ylabel = 'Frequency')

#Boxplot
plt.subplot(1,2,2)
#Define plot object
box = sns.boxplot(x_train.loc[:,'budget'].astype(float))
#Setting graph title
box.set_title( 'Budget')
box.set(xlabel = 'Dollars', ylabel = 'Frequency')

#Showing the plot
plt.show()
```



From the above graphs, we cannot gain any insight. This is because there are very extreme values (outliers) in the dataset, which heavily skew any exploration or prediction.

5.2.12 - IMDB Score

Descriptive statistics of IMDB Score.

In [79]:

```
pd.DataFrame(x_train.loc[:,'imdb_score'].describe())
```

Out[79]:

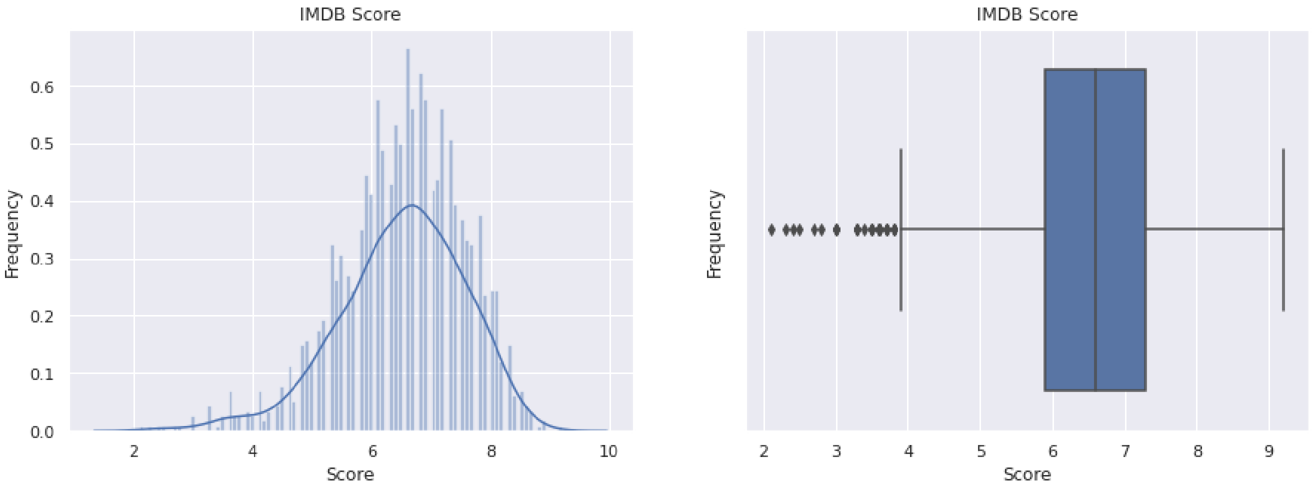
	imdb_score
count	1609.000000
mean	6.511871
std	1.056490
min	2.100000
25%	5.900000
50%	6.600000
75%	7.300000
max	9.200000

The distribution and skewness of this variable are illustrated by the histogram and box plot below.

In [80]:

```
fig = plt.figure(figsize = (15,5))
```

```
#Histogram
plt.subplot(1,2,1)
#Define plot object
hist = sns.distplot(x_train.loc[:,'imdb_score'].astype(float), bins = 100)
#Setting graph title
hist.set_title('IMDB Score')
hist.set(xlabel = 'Score', ylabel = 'Frequency')

#Boxplot
plt.subplot(1,2,2)
#Define plot object
box = sns.boxplot(x_train.loc[:,'imdb_score'].astype(float))
#Setting graph title
box.set_title('IMDB Score')
box.set(xlabel = 'Score', ylabel = 'Frequency')

#Showing the plot
plt.show()
```



A quick glance at graphs reveals that they are similar to the graphs of avg_rating, our dependent variable. This makes sense because both scores are derived by public votes and therefore they should have the same properties, for example a tail at the lower end. The are a few low outliers.

The outliers identified in these descriptive statistics will be dealt with via transformation in Section 6. This is done prior to their use in any predictive modelling to prevent bias being introduced.

5.3 - Categorical Independent Variables

The following coloumns are Categorical Independent Variables and are investigated below.

- color
- director_name
- actor_1_name
- actor_2_name
- actor_3_name
- language
- country
- content_rating
- genres
- plot_keywords

5.3.1 - Color

Descriptive statistics and value counts for Color variable.

In [81]:

```
pd.DataFrame(x_train.loc[:,'color'].describe())
```

Out[81]:

	color
count	1609
unique	2
top	Color
freq	1507

In [82]:

```
#Getting the value counts of the color variable
values = pd.DataFrame(x_train.loc[:,'color'].value_counts())
values.columns = ['Color Count']

#The normalize attribute in the value_counts method computes the percentage for each category
percentages = pd.DataFrame(round(x_train.loc[:,'color'].value_counts(normalize=True)*100,2).map(lambda x : str(x)+'%'))
percentages.columns = ['Color %']

values.join(percentages)
```

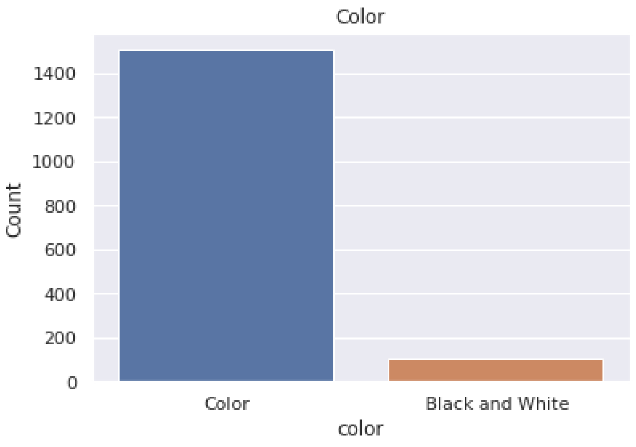
Out[82]:

	Color Count	Color %
Color	1507	93.66%
Black and White	102	6.34%

The plot below illustrates the 'counts' for this variable.

In [83]:

```
#Define plot object
count = sns.countplot(x_train.loc[:,'color'])
#Setting graph title
count.set_title('Color')
count.set(xlabel = 'color', ylabel = 'Count')
plt.xticks(rotation=0)
#Showing the plot
plt.show()
```



There are far fewer Black and White movies compared with Color movies.

5.3.2 - Director Name

Descriptive statistics of Director Name.

In [84]:

```
pd.DataFrame(x_train.loc[:,'director_name'].describe())
```

Out[84]:

	director_name
count	1609
unique	909
top	Steven Spielberg
freq	12

To expand upon the descriptive statistics, we find the 10 directors with the highest counts.

In [85]:

```
#Getting the value counts of the director_name variable
values = pd.DataFrame(x_train.loc[:,'director_name'].value_counts()[0:10])
values.columns = ['Director Count']

#The normalize attribute in the value_counts method computes the percentage of each
percentages = pd.DataFrame(round(x_train.loc[:,'director_name'].value_counts(normalize=True)*100,2)
.map(lambda x : str(x)+'%'))
percentages.columns = ['Director %']

values.join(percentages)
```

Out[85]:

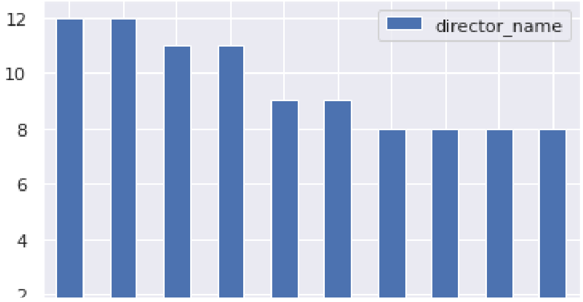
	Director Count	Director %
Steven Spielberg	12	0.75%
Joel Schumacher	12	0.75%
Tim Burton	11	0.68%
Martin Scorsese	11	0.68%
Kenneth Branagh	9	0.56%
Barry Levinson	9	0.56%
Clint Eastwood	8	0.5%
Steven Soderbergh	8	0.5%
John Carpenter	8	0.5%
John McTiernan	8	0.5%

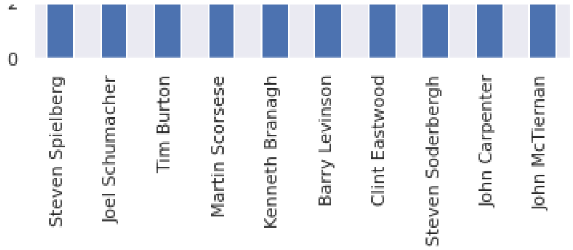
In [86]:

```
# bar chart of top 10 directors
pd.DataFrame(x_train.loc[:,'director_name'].value_counts()[0:10]).plot(kind='bar',)
```

Out[86]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f61e596f940>
```





The value counts and the bar chart illustrate the top 10 most common directors with Steven Spielberg and Joel Schumacher being most frequent.

5.3.3 - Actor Name

Descriptive statistics of Actor 1 Name.

In [87]:

```
pd.DataFrame(x_train.loc[:,'actor_1_name'].describe())
```

Out[87]:

	actor_1_name
count	1608
unique	797
top	Robert De Niro
freq	25

To expand upon the descriptive statistics, we find the 10 primary actors with the highest counts.

In [88]:

```
#Getting the value counts of the actor_1 variable
values = pd.DataFrame(x_train.loc[:,'actor_1_name'].value_counts()[0:10])
values.columns = ['Actor 1 Count']

#The normalize attribute in the value_counts method computes the percentage of each
percentages = pd.DataFrame(round(x_train.loc[:,'actor_1_name'].value_counts(normalize=True)*100,2).
map(lambda x : str(x)+'%'))
percentages.columns = ['Actor 1 %']

values.join(percentages)
```

Out[88]:

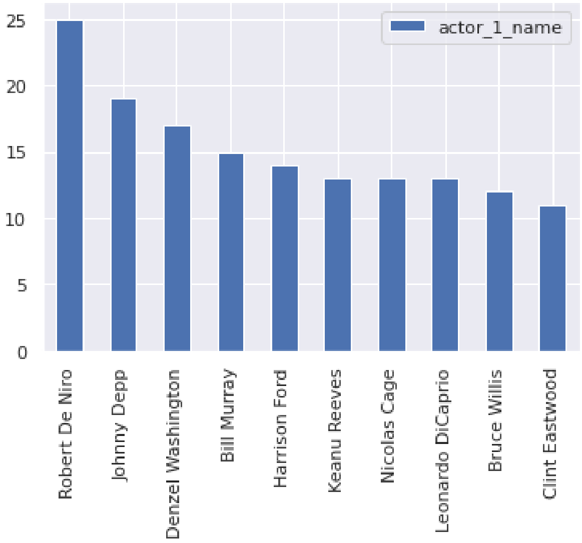
	Actor 1 Count	Actor 1 %
Robert De Niro	25	1.55%
Johnny Depp	19	1.18%
Denzel Washington	17	1.06%
Bill Murray	15	0.93%
Harrison Ford	14	0.87%
Keanu Reeves	13	0.81%
Nicolas Cage	13	0.81%
Leonardo DiCaprio	13	0.81%
Bruce Willis	12	0.75%
Clint Eastwood	11	0.68%

In [89]:

```
# bar chart of top 10 primary actors
pd.DataFrame(x_train.loc[:,'actor_1_name'].value_counts()[0:10]).plot(kind='bar',)
```

Out[89]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f61e5021d30>
```



The value counts and the bar chart illustrate the top 10 most common primary actors with Robert De Niro being most frequent.

Descriptive statistics of Actor 2 Name.

In [90]:

```
pd.DataFrame(x_train.loc[:,'actor_2_name'].describe())
```

Out[90]:

	actor_2_name
count	1608
unique	1098
top	Judy Greer
freq	9

To expand upon the descriptive statistics, we find the 10 secondary actors with the highest counts.

In [91]:

```
#Getting the value counts of the actor_2 variable
values = pd.DataFrame(x_train.loc[:,'actor_2_name'].value_counts()[0:10])
values.columns = ['Actor 2 Count']

#The normalize attribute in the value_counts method computes the percentage of each
percentages = pd.DataFrame(round(x_train.loc[:,'actor_2_name'].value_counts(normalize=True)*100,2).
map(lambda x : str(x)+'%'))
percentages.columns = ['Actor 2 %']

values.join(percentages)
```

Out[91]:

	Actor 2 Count	Actor 2 %
Judy Greer	9	0.56%
Morgan Freeman	8	0.5%
Charlize Theron	7	0.44%

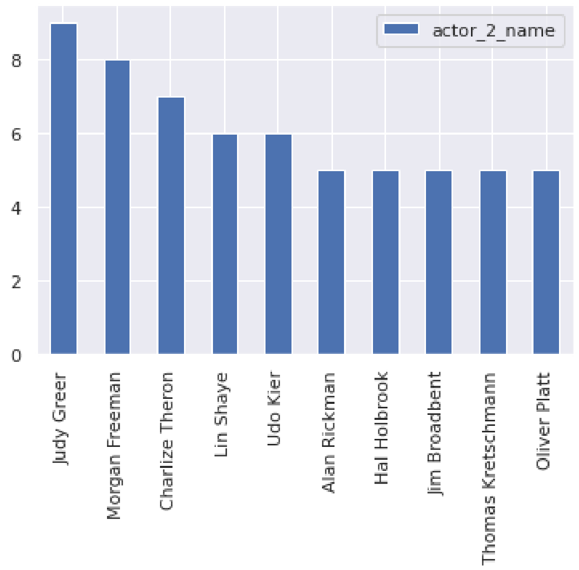
	Actor 2 Count	Actor 2 %
Lin Shaye	6	0.37%
Udo Kier	6	0.37%
Alan Rickman	5	0.31%
Hal Holbrook	5	0.31%
Jim Broadbent	5	0.31%
Thomas Kretschmann	5	0.31%
Oliver Platt	5	0.31%

In [92]:

```
# bar chart of top 10 secondary actors
pd.DataFrame(x_train.loc[:,'actor_2_name'].value_counts()[0:10]).plot(kind='bar',)
```

Out[92]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f61e57d2400>
```



The value counts and the bar chart illustrate the top 10 most common secondary actors with Judy Greer being most frequent.

Descriptive statistics of Actor 3 Name.

In [93]:

```
pd.DataFrame(x_train.loc[:,'actor_3_name'].describe())
```

Out[93]:

	actor_3_name
count	1605
unique	1208
top	Tom Wilkinson
freq	6

To expand upon the descriptive statistics, we find the 10 tertiary actors with the highest counts.

In [94]:

```
#Getting the value counts of the actor_3 variable
values = pd.DataFrame(x_train.loc[:,'actor_3_name'].value_counts()[0:10])
values.columns = ['Actor 3 Count']

#The normalize attribute in the value counts method computes the percentage of each
```

```
percentages = pd.DataFrame(round(x_train.loc[:,'actor_3_name'].value_counts(normalize=True)*100,2).
map(lambda x : str(x)+'%'))
percentages.columns = ['Actor 3 %']

values.join(percentages)
```

Out[94]:

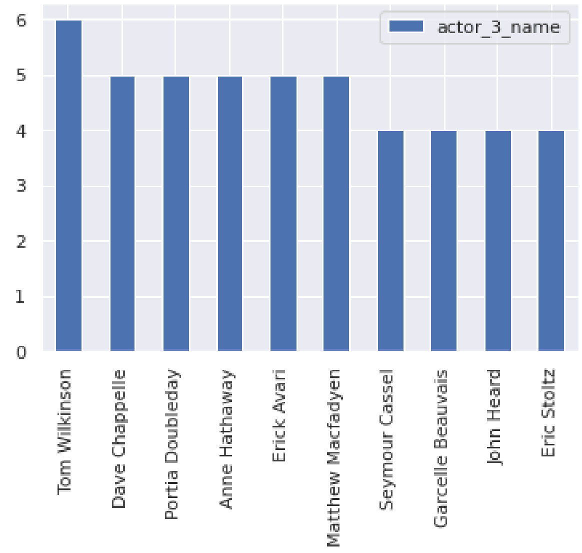
	Actor 3 Count	Actor 3 %
Tom Wilkinson	6	0.37%
Dave Chappelle	5	0.31%
Portia Doubleday	5	0.31%
Anne Hathaway	5	0.31%
Erick Avari	5	0.31%
Matthew Macfadyen	5	0.31%
Seymour Cassel	4	0.25%
Garcelle Beauvais	4	0.25%
John Heard	4	0.25%
Eric Stoltz	4	0.25%

In [95]:

```
# bar chart of top 10 tertiary actors
pd.DataFrame(x_train.loc[:,'actor_3_name'].value_counts()[0:10]).plot(kind='bar',)
```

Out[95]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f61e50dc0f0>
```



The value counts and the bar chart illustrate the top 10 most common tertiary actors with Tom Wilkinson being most frequent.

5.3.4 - Language

Descriptive statistics and value counts for Language variable.

In [96]:

```
pd.DataFrame(x_train.loc[:,'language'].describe())
```

Out[96]:

language

	language
count	1609
unique	19
top	English
freq	1549

In [97]:

```
#Getting the value counts of the language variable
values = pd.DataFrame(x_train.loc[:,'language'].value_counts())
values.columns = ['Language Count']

#The normalize attribute in the value_counts method computes the percentage of each category
percentages = pd.DataFrame(round(x_train.loc[:,'language'].value_counts(normalize=True)*100,2).map(
lambda x : str(x)+'%'))
percentages.columns = ['Language %']

values.join(percentages)
```

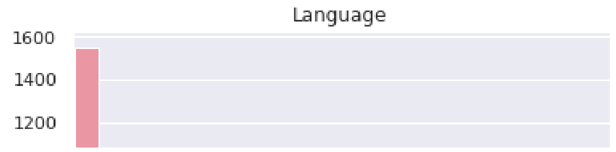
Out[97]:

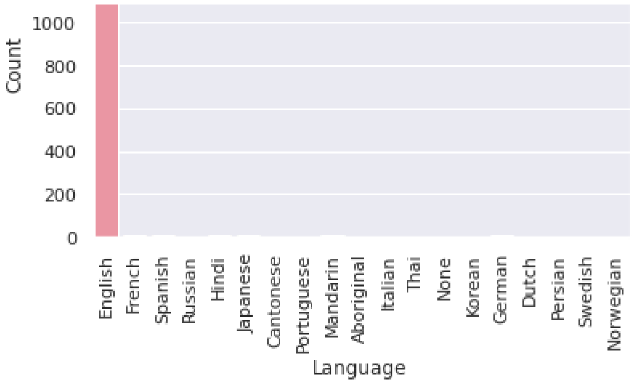
	Language Count	Language %
English	1549	96.27%
French	8	0.5%
Spanish	7	0.44%
Japanese	6	0.37%
Hindi	6	0.37%
Mandarin	5	0.31%
German	5	0.31%
Italian	3	0.19%
Cantonese	3	0.19%
Portuguese	3	0.19%
Russian	3	0.19%
Aboriginal	2	0.12%
Persian	2	0.12%
Korean	2	0.12%
Dutch	1	0.06%
Norwegian	1	0.06%
Thai	1	0.06%
Swedish	1	0.06%
None	1	0.06%

The plot below illustrates the 'counts' for this variable.

In [98]:

```
#Define plot object
count = sns.countplot(x_train.loc[:,'language'])
#Setting graph title
count.set_title('Language')
count.set(xlabel = 'Language', ylabel = 'Count')
plt.xticks(rotation=90)
#Showing the plot
plt.show()
```





From the graph above, you can see that the majority of movies on Netflix are in English but some other languages are also included. Increasing other language movies may be beneficial for Netflix.

5.3.5 - Country

Descriptive statistics and value counts for Country variable.

In [99]:

```
pd.DataFrame(x_train.loc[:,'country'].describe())
```

Out[99]:

	country
count	1609
unique	31
top	USA
freq	1273

In [100]:

```
#Getting the value counts of the country variable
values = pd.DataFrame(x_train.loc[:,'country'].value_counts())
values.columns = ['Country Count']

#The normalize attribute in the value_counts method computes the percentage of each country
percentages = pd.DataFrame(round(x_train.loc[:,'country'].value_counts(normalize=True)*100,2).map(lambda x : str(x)+'%'))
percentages.columns = ['Country %']

values.join(percentages)
```

Out[100]:

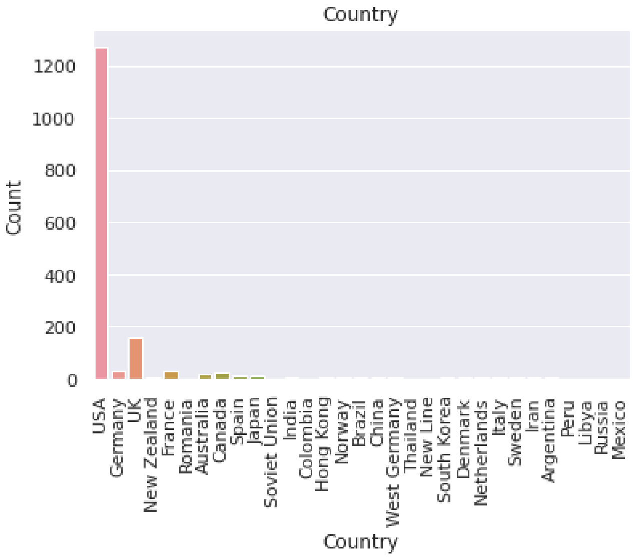
	Country Count	Country %
USA	1273	79.12%
UK	157	9.76%
France	32	1.99%
Germany	32	1.99%
Canada	26	1.62%
Australia	16	0.99%
Japan	10	0.62%
Spain	8	0.5%
New Zealand	6	0.37%
India	6	0.37%
China	6	0.37%

	Country Count	Country %
Italy	5	0.31%
Hong Kong	4	0.25%
Brazil	3	0.19%
Denmark	2	0.12%
Argentina	2	0.12%
Sweden	2	0.12%
South Korea	2	0.12%
Norway	2	0.12%
West Germany	2	0.12%
Netherlands	2	0.12%
Iran	2	0.12%
Colombia	1	0.06%
Libya	1	0.06%
Thailand	1	0.06%
Peru	1	0.06%
New Line	1	0.06%
Russia	1	0.06%
Soviet Union	1	0.06%
Romania	1	0.06%
Mexico	1	0.06%

The plot below illustrates the 'counts' for this variable.

In [101]:

```
#Define plot object
count = sns.countplot(x_train.loc[:,'country'])
#Setting graph title
count.set_title('Country')
count.set(xlabel = 'Country', ylabel = 'Count')
plt.xticks(rotation=90)
#Showing the plot
plt.show()
```



For the vast majority of movies on Netflix the production country is the USA. This makes sense given that the most frequent language was English.

5.3.6 - Content Rating

Descriptive statistics of Content Rating.

In [102]:

```
pd.DataFrame(x_train.loc[:,'content_rating'].describe())
```

Out[102]:

	content_rating
count	1588
unique	12
top	R
freq	736

Descriptive statistics and value counts for Content Rating variable.

In [103]:

```
#Getting the value counts of the content_rating variable
values = pd.DataFrame(x_train.loc[:,'content_rating'].value_counts())
values.columns = ['Ratings Count']

#The normalize attribute in the value_counts method computes the percentage of each category
percentages = pd.DataFrame(round(x_train.loc[:,'content_rating'].value_counts(normalize=True)*100,2
).map(lambda x : str(x)+'%'))
percentages.columns = ['Ratings %']

values.join(percentages)
```

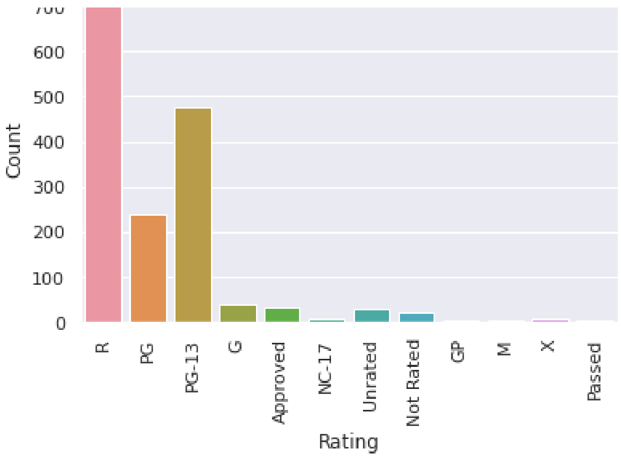
Out[103]:

	Ratings Count	Ratings %
R	736	46.35%
PG-13	477	30.04%
PG	238	14.99%
G	38	2.39%
Approved	31	1.95%
Unrated	27	1.7%
Not Rated	21	1.32%
X	6	0.38%
NC-17	6	0.38%
GP	4	0.25%
Passed	2	0.13%
M	2	0.13%

The plot below illustrates the 'counts' for this variable.

In [104]:

```
#Define plot object
count = sns.countplot(x_train.loc[:,'content_rating'])
#Setting graph title
count.set_title('Content Rating')
count.set(xlabel = 'Rating', ylabel = 'Count')
plt.xticks(rotation=90)
#Showing the plot
plt.show()
```



Most movies are R, PG-13 or PG. R takes the largest share of the movie industry as a whole. We suspect that because R rated content is not as popular in movie theatres, Netflix becomes a popular choice for them.

5.3.7 - Genres

To do descriptive statistics on the Genres, we should consider all related columns together. Hence we will stack all 8 columns together.

In [105]:

```
#Slice out the columns required
genres_df = x_train.loc[:,
['genres_1','genres_2','genres_3','genres_4','genres_5','genres_6','genres_7','genres_8']]

#Validating Data
genres_df.shape
```

Out[105]:

```
(1609, 8)
```

In [106]:

```
#Stacking all the columns
genres_series = pd.Series([i for sublist in genres_df.values.tolist() for i in sublist])

#Validating Data
genres_series.shape
```

Out[106]:

```
(12872,)
```

Descriptive statistics and value counts for Genres variable.

In [107]:

```
pd.DataFrame(genres_series.describe(), columns = ['Genres'])
```

Out[107]:

	Genres
count	4731
unique	23
top	Drama
freq	885

In [108]:

```
#Getting the value counts of the genre variable
values = pd.DataFrame(genres_series.value_counts())
values.columns = ['Genre Count']

#The normalize attribute in the value_counts method computes the percentage of each category
#There are many categories and therefore we only show the top 10 genres
percentages = pd.DataFrame(round(genres_series.value_counts(normalize=True)*100,2).map(lambda x : s
tr(x)+'%'))
percentages.columns = ['Genre %']

values.join(percentages).head(10)
```

Out[108]:

	Genre Count	Genre %
Drama	885	18.71%
Comedy	590	12.47%
Thriller	492	10.4%
Romance	395	8.35%
Action	379	8.01%
Crime	344	7.27%
Adventure	287	6.07%
Fantasy	185	3.91%
Sci-Fi	176	3.72%
Mystery	163	3.45%

The majority of genres are drama, comedy, and thriller. Altogether, these genres account for 40% of all genres.

5.3.8 - Plot Keywords

To do descriptive statistics on the Plot Keywords, we should consider all related columns together. Hence we will stack all 5 columns together.

In [109]:

```
#Slice out the columns required
plot_keywords_df = x_train.loc[:,
['plot_keywords_1','plot_keywords_2','plot_keywords_3','plot_keywords_4','plot_keywords_5']]

#Validating Data
plot_keywords_df.shape
```

Out[109]:

```
(1609, 5)
```

In [110]:

```
#Stacking all the columns
plot_keywords_series = pd.Series([i for sublist in plot_keywords_df.values.tolist() for i in sublis
t])

#Validating Data
plot_keywords_series.shape
```

Out[110]:

```
(8045,)
```

Descriptive statistics and value counts for Plot keywords variable.

In [111]:

In [111]:

```
pd.DataFrame(plot_keywords_series.describe(), columns = ['Plot Keywords'])
```

Out[111]:

	Plot Keywords
count	8016
unique	3580
top	murder
freq	77

In [112]:

```
#Getting the value counts of the plot_keywords variable
values = pd.DataFrame(plot_keywords_series.value_counts())
values.columns = ['Plot Keywords Count']

#The normalize attribute in the value_counts method computes the percentage of each category
#There are many keywords and therefore we only show the top 10
percentages = pd.DataFrame(round(plot_keywords_series.value_counts(normalize=True)*100,2).map(lambd
a x : str(x)+'%'))
percentages.columns = ['Plot Keywords %']

values.join(percentages).head(10)
```

Out[112]:

	Plot Keywords Count	Plot Keywords %
murder	77	0.96%
love	72	0.9%
friend	59	0.74%
death	50	0.62%
high school	40	0.5%
police	38	0.47%
boy	38	0.47%
revenge	31	0.39%
money	30	0.37%
new york city	28	0.35%

Thriller is one of the top 3 genres, which explains why the top keyword is 'murder'.

Any categorical variables required in the predictive model will be converted to dummy variables first via data preprocessing.

6.0 - Data Cleaning

6.1 - Handling Outliers

We identified several variables in the descriptive statistics section (above) that produced outliers. These must be removed so that the variables can be used in the predictive modelling without introducing bias.

The class below will be called during our Individual assignments to transform outliers in both the training and test datasets.

In [0]:

```
@time_step
class OutlierTransformer(TransformerMixin, BaseEstimator):
    """
```

```
  This class transforms all outliers into np.NaN, then the NaNs will be imputed later
  The definition of an outlier is a value smaller than quantile1-(1.5*IQR_Value) or larger than quantile3+(1.5*IQR_Value)
  Note : the fit_transform method is included in the TransformerMixin super class
  """
  def __init__(self):
    self.fitted = False
    return None

  def fit(self, X):
    """
    Compute the upper and lower bound to be used later.
    Parameters
    ----------
    X : {array-like, sparse matrix}, shape [n_samples, n_features]
        The data used to compute the upper and lower bound
        used for later scaling along the features axis.
    """
    #Select and handle outliers for only the columns containing numerical variables
    numerics = ['int16', 'int32', 'int64', 'float16', 'float32', 'float64']
    numeric_X = X.select_dtypes(include=numerics)
    # Get Quantiles and IQR
    self.quantiles = pd.DataFrame(numeric_X.quantile([0.25, 0.75]))
    self.quantiles.loc['IQR',:] = self.quantiles.loc[0.75,:] - self.quantiles.loc[0.25,:]
    #Calculate Upper and Lower Bounds
    #These bounds are calculated on the Train Dataset, then applied to the train and test datasets
    self.quantiles.loc['Lower_Bound',:] = self.quantiles.loc[0.25,:] - 1.5 * self.quantiles.loc['IQR',:]
    self.quantiles.loc['Upper_Bound',:] = self.quantiles.loc[0.75,:] + 1.5 * self.quantiles.loc['IQR',:]

    #Save fitting status
    self.fitted = True

    return self

  def transform(self, X):
      """
      Replaces Outliers with NaNs
      Parameters
      ----------
      X : array-like, shape [n_samples, n_features]
          The data used to scale along the features axis.
      """
      if self.fitted == False:
        print('The transformer must be fitted before transformation.')
        return None

      #Make a copy of X to avoid mutating X
      X_copy = X.copy()

      #Old number of NaNs to count number of outliers
      old_num_na = sum(X_copy.isna().sum())

      for col in self.quantiles.columns:
        #Fill Beyond Lower Bound
        X_copy.loc[X_copy.loc[:,col] < self.quantiles.loc['Lower_Bound',col],col] = np.NaN

        #Fill Beyond Upper Bound
        X_copy.loc[X_copy.loc[:,col] > self.quantiles.loc['Upper_Bound',col],col] = np.NaN

      #Print Number of Outliers
      print(str(sum(X_copy.isna().sum()) - old_num_na) + ' Outliers are Identified.')

      return X_copy
```

6.2 - Imputing Missing Values

After the class above has been called, the class below will be called to impute the NaN values (missing values *and* outliers) in both the training and test datasets.

We will first train an imputer for imputing missing values, then apply this to both the train and test dataset. Hence, if there is a column in the test set containing a missing value that does not exist in the train set, the imputer will fail. Therefore the mean of the column should be imputed instead.

should be imputed instead.

In [0]:

```
class MissingValueTransformer(TransformerMixin, BaseEstimator):
  """
  This class imputes all missing values by IterativeImputer
  This is a strategy for imputing missing values by modeling each feature with missing values as a function
  of other features in a round-robin fashion.
  Note : the fit_transform method is included in the TransformerMixin super class
  """
  def __init__(self):
    self.fitted = False
    return None

  def fit(self, X):
    """
    Train the Missing value imputer
    Parameters
    ----------
    X : {array-like, sparse matrix}, shape [n_samples, n_features]
        The data used to compute the upper and lower bound
        used for later scaling along the features axis.
    """
    """
    Numeric Features
    For Numeric Features, we can use a function to estimate the missing values
    """
    #Select missing values for only the columns containing numerical variables
    numerics = ['int16', 'int32', 'int64', 'float16', 'float32', 'float64']
    numeric_X = X.select_dtypes(include=numerics)
    #Train Imputer
    self.transformer_num = IterativeImputer(random_state = 2020)
    self.transformer_num.fit(numeric_X)

    """
    Categorical Features
    For Categorical Features, we should not use a function to estimate, instead, we could use KNN (k nearest neighbour)
    """
    #Selecting missing values for only non-numeric columns
    #Note that Plot Keywords, Genres, Actor Names and Director Names are
    #not imputed because there can have various numbers of values for them and can be empty,
    #hence we need to be a bit more clever in feature engineering (and do some sort of transformation)
    self.cat_col = ['color','country','language','content_rating']
    cat_X = X.loc[:,self.cat_col]
    #Train Imputer
    self.transformer_cat = KNNImputer(n_neighbors = 10)
    self.transformer_cat.fit(cat_X)

    #Save fitting status
    self.fitted = True

    return self

  def transform(self, X):
      """
      Impute Missing Values
      Parameters
      ----------
      X : array-like, shape [n_samples, n_features]
          The data used to scale along the features axis.
      """
      if self.fitted == False:
        print('The transformer must be fitted before transformation.')
        return None

      #Make a copy of X to avoid mutating X
      X_copy = X.copy()

      #Old number of Missing Values
      old_num_na = sum(X_copy.isna().sum())

      """
      Numeric Features
```

```
        """
        #Select missing values for only the columns containing numerical variables
        numerics = ['int16', 'int32','int64','float16','float32','float64']
        numeric_X = X_copy.select_dtypes(include=numerics)
        numeric_X = self.transformer_num.transform(numeric_X)

        """
        Categorical Features
        """
        #Selecting missing values for only categorical columns
        cat_X = X_copy.loc[:,self.cat_col]
        cat_X = self.transformer_cat.transform(cat_X)

        """
        Merging Columns
        """
        transformed_X = numeric_X.join(cat_X)
        remaining_col = [col for col in X.columns if col not in transformed_X.columns]
        transformed_X = X.copy[:,remaining_col].join(transformed_X)

        #Print Number of Outliers
        print(str(sum(transformed_X.isna().sum()) - old_num_na) + ' Missing Values are Imputed.')

        return transformed_X
```

We are not including the data preprocessing (scaling and normalisation) in the Group assignment because this should be done *after* any feature engineering. We will all be engineering different features in the Individual assignment and therefore we will do the scaling and normalisation after this.

7.0 - Conclusion

In this Group assignment we successfully created a dataframe of movies. For each movie, IMDB metadata and average Netflix rating is given. In the Individual assignment we will be generating a predictive model which predicts average Netflix rating based on independent variables which come from the IMDB metadata. By predicting ratings for new movies, Netflix should be able to determine which licensing agreements to buy and how much to spend on these agreements.

8.0 - Data Exporting

After the conclusion of the Group assignment, we can now pass on the transformed and split datasets to the Individual assignment by exporting them to csv files.

In [0]:

```
x_train.to_csv("/content/gdrive/My Drive/Colab Notebooks/x_train.csv")
y_train.to_csv("/content/gdrive/My Drive/Colab Notebooks/y_train.csv")
x_test.to_csv("/content/gdrive/My Drive/Colab Notebooks/x_test.csv")
y_test.to_csv("/content/gdrive/My Drive/Colab Notebooks/y_test.csv")
```

In [116]:

```
# Finish Timer
"---- The notebook is completed in %s minutes----"%((str(round((time.time() - Start_time)/60,4))))
```

Out[116]:

```
'---- The notebook is completed in 7.067 minutes----'
```