MOVIE NUMBERS ANALYSIS FOR MICROSOFT STUDIO JUSTIFICATION

OVERVIEW

This project uses descriptive statistics and exploratory data analysis of ratings and title data from iMDb and movie grossing Box Office Mojo and The Numbers to generate insights for a Microsoft company who would like to create a new studio.

BUSINESS PROBLEM

Microsoft sees all the big companies creating original video content and they want to get in on the fun. They have decided to create a new movie studio, but they don’t know anything about creating movies. The main objective of this project is exploring what types of films are currently doing the best at the box office. Findings from this analysis should then be converted into actionable insights that the head of Microsoft's new movie studio can use to help decide what type of films to create.

DATA UNDERSTANDING

In [1]:

```
#Import the necessary libraries under their respective aliases as is the industry standar
d
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

#This line of code allows us to generate plots within this notebook as opposed to generat
ing external plots
%matplotlib inline
```

The first task to read the data files into our working environment then explore the files in order to gain an initial understanding of the data. This also allows us to determine what data wrangling techniques to apply in order to transform the data into a form that can be analysed.

File Parsing

In [2]:

```
# Read the files into the notebook's working memory

gross = pd.read_csv('C:/Users/Hp/Documents/Flatiron/Phase_1_Project/DSF_PTO4_Project_1_So
lution/data files/bom.movie_gross.csv')

titles = pd.read_csv('C:/Users/Hp/Documents/Flatiron/Phase_1_Project/DSF_PTO4_Project_1_S
olution/data files/title.basics.csv')

ratings = pd.read_csv('C:/Users/Hp/Documents/Flatiron/Phase_1_Project/DSF_PTO4_Project_1_
Solution/data files/title.ratings.csv')

budgets = pd.read_csv('C:/Users/Hp/Documents/Flatiron/Phase_1_Project/DSF_PTO4_Project_1_
Solution/data files/tn.movie_budgets.csv', index_col=0)
```

Inspect the DataFrames loaded using df.info() and df.head() methods

In [3]:

```
gross.info()
```

```
gross.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
```

In [4]:

```
gross.head(3)
```

Out[4]:

	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

In [5]:

```
titles.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    146144 non-null  object 
 2   original_title   146123 non-null  object 
 3   start_year       146144 non-null  int64  
 4   runtime_minutes  114405 non-null  float64
 5   genres           140736 non-null  object 
dtypes: float64(1), int64(1), object(4)
memory usage: 6.7+ MB
```

In [6]:

```
titles.head(3)
```

Out[6]:

	tconst	primary_title	original_title	start_year	runtime_minutes	genres
0	tt0063540	Sunghursh	Sunghursh	2013	175.0	Action,Crime,Drama
1	tt0066787	One Day Before the Rainy Season	Ashad Ka Ek Din	2019	114.0	Biography,Drama
2	tt0069049	The Other Side of the Wind	The Other Side of the Wind	2018	122.0	Drama

In [7]:

```
ratings.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)
```

```
dtypes: float64(1), int64(1), object(1)
memory usage: 1.7+ MB
```

In [8]:

```
ratings.head(3)
```

Out[8]:

	tconst	averagerating	numvotes
0	tt10356526	8.3	31
1	tt10384606	8.9	559
2	tt1042974	6.4	20

In [9]:

```
budgets.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5782 entries, 1 to 82
Data columns (total 5 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   release_date       5782 non-null   object
 1   movie              5782 non-null   object
 2   production_budget  5782 non-null   object
 3   domestic_gross     5782 non-null   object
 4   worldwide_gross    5782 non-null   object
dtypes: object(5)
memory usage: 271.0+ KB
```

In [10]:

```
budgets.head()
```

Out[10]:

	release_date	movie	production_budget	domestic_gross	worldwide_gross
id					
1	Dec 18, 2009	Avatar	$425,000,000	$760,507,625	$2,776,345,279
2	May 20, 2011	Pirates of the Caribbean: On Stranger Tides	$410,600,000	$241,063,875	$1,045,663,875
3	Jun 7, 2019	Dark Phoenix	$350,000,000	$42,762,350	$149,762,350
4	May 1, 2015	Avengers: Age of Ultron	$330,600,000	$459,005,868	$1,403,013,963
5	Dec 15, 2017	Star Wars Ep. VIII: The Last Jedi	$317,000,000	$620,181,382	$1,316,721,747

DATA WRANGLING

gross_dataframe

First we convert the 'foreign_gross' column to a numeric type to enable analysis

In [11]:

```
# Uncommenting and running the code below in this cell will yield an error
# ValueError: could not convert string to float: '1,131.6'

#gross['foreign_gross'] = gross['foreign_gross'].astype(float)
```

This error indicates the prescence of ',' within some values in the column. Hence we first eliminate the commas before converting the datatype of the column to float for numerical analysis.

In [12]:

```
# Remove any existing ',' in the DataFrame

gross['foreign_gross'] = gross['foreign_gross'].str.replace(',','')
```

In [13]:

```
# Convert the datatype of 'foreign_gross' from string to float

gross['foreign_gross'] = gross['foreign_gross'].astype(float)
```

In [14]:

```
# Convert the 'year' column to datetime format
gross['year'] = pd.to_datetime(gross['year'])
```

In [15]:

```
# Next we check that the datatype for the 'foreign_gross' column has been converted succe
ssfully.

gross.dtypes
```

Out[15]:

```
title                     object
studio                    object
domestic_gross           float64
foreign_gross            float64
year              datetime64[ns]
dtype: object
```

budgets_dataframe

In this DataFrame, there are several columns that require cleaning as well as formatting to numerical datatypes i.e. 'release_date', 'domestic_gross', 'production_budget' & 'worlwide_gross'.

In [16]:

```
# Eliminate ',' from all data values in the column

budgets['release_date'] = budgets['release_date'].str.replace(',','')
```

In [17]:

```
# Convert 'relesae_date' column to datetime format

budgets['release_date'] = pd.to_datetime(budgets['release_date'])
```

In [18]:

```
# Ensure the datatype conversion was successful

budgets.dtypes
```

Out[18]:

```
release_date         datetime64[ns]
movie                        object
production_budget            object
domestic_gross               object
worldwide_gross              object
dtype: object
```

Just as we did for the 'foreign_gross' column from the 'gross' DataFrame, we first eliminate any punctuation(',' & '$') from 'production_budget', 'domestic_gross', 'worldwide_gross' columns before performing the datatype conversion to avoid the error encountered earlier.

In [19]:

```
# Eliminate ',' and '&' from 'production_budget' column

budgets['production_budget'] = budgets['production_budget'].str.replace(',','')
budgets['production_budget'] = budgets['production_budget'].str.replace('$','')
```

In [20]:

```
# Convert 'production_budget' from string to float

budgets['production_budget'] = budgets['production_budget'].astype(float)
```

In [21]:

```
# Eliminate ',' and '&' from 'domestic_gross' column

budgets['domestic_gross'] = budgets['domestic_gross'].str.replace(',','')
budgets['domestic_gross'] = budgets['domestic_gross'].str.replace('$','')
```

In [22]:

```
# Convert 'domestic_gross' from string to float

budgets['domestic_gross'] = budgets['domestic_gross'].astype(float)
```

In [23]:

```
# Eliminate ',' and '&' from 'worldwide_gross' column

budgets['worldwide_gross'] = budgets['worldwide_gross'].str.replace(',','')
budgets['worldwide_gross'] = budgets['worldwide_gross'].str.replace('$','')
```

In [24]:

```
# Convert 'worldwide_gross' from string to float

budgets['worldwide_gross'] = budgets['worldwide_gross'].astype(float)
```

In [25]:

```
# Ensure the datatype conversions were successful

budgets.dtypes
```

Out[25]:

```
release_date         datetime64[ns]
movie                        object
production_budget           float64
domestic_gross              float64
worldwide_gross             float64
dtype: object
```

DATA ANALYSIS

Feature Engineering & Data Visualization

gross_dataframe

In [26]:

```
# Engineering a new feature 'total_gross from existing columns

gross['total_gross'] = gross['foreign_gross'] + gross['domestic_gross']
```

In [27]:

```
# Ensure that the new column has been added to the DataFrame by inspecting using .info()

gross.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3387 entries, 0 to 3386
Data columns (total 6 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   float64
 4   year            3387 non-null   datetime64[ns]
 5   total_gross     2009 non-null   float64
dtypes: datetime64[ns](1), float64(3), object(2)
memory usage: 158.9+ KB
```

In [28]:

```
# Statistical summary of numerical data in the dataset

gross.describe()
```

Out[28]:

	domestic_gross	foreign_gross	total_gross
count	3.359000e+03	2.037000e+03	2.009000e+03
mean	2.874585e+07	7.487281e+07	1.226913e+08
std	6.698250e+07	1.374106e+08	2.074870e+08
min	1.000000e+02	6.000000e+02	4.900000e+03
25%	1.200000e+05	3.700000e+06	8.141000e+06
50%	1.400000e+06	1.870000e+07	4.230000e+07
75%	2.790000e+07	7.490000e+07	1.337000e+08
max	9.367000e+08	9.605000e+08	1.518900e+09

In [29]:

```
# Inspecting the dataframe

gross.head()
```

Out[29]:

	title	studio	domestic_gross	foreign_gross	year	total_gross
0	Toy Story 3	BV	415000000.0	652000000.0	1970-01-01 00:00:00.000002010	1.067000e+09
1	Alice in Wonderland (2010)	BV	334200000.0	691300000.0	1970-01-01 00:00:00.000002010	1.025500e+09
2	Harry Potter and the Deathly Hallows Part 1	WB	296000000.0	664300000.0	1970-01-01 00:00:00.000002010	9.603000e+08
3	Inception	WB	292600000.0	535700000.0	1970-01-01 00:00:00.000002010	8.283000e+08
4	Shrek Forever After	P/DW	238700000.0	513900000.0	1970-01-01 00:00:00.000002010	7.526000e+08

In [30]:

```
# Number of unique studios contained in the dataset

gross['studio'].nunique()
```

Out[30]:

```
257
```

In [31]:

```
#Total number of movies by each studio

movie_studio_count = gross['studio'].value_counts()
```

In [32]:

```
# Silcing the top ten most productive studios of the time period

movie_studio_count[0:10]
```

Out[32]:

```
IFC      166
Uni.     147
WB       140
Magn.    136
Fox      136
SPC      123
Sony     110
BV       106
LGF      103
Par.     101
Name: studio, dtype: int64
```

In [33]:

```
# Determining the period of analysis

gross['year'].nunique()
```

Out[33]:

```
9
```

Considering the above information, we can deduce that to be top studio the number of movie projects output should be around 10-15 movies.

In [34]:

```
gross['year'].unique()
```

Out[34]:

```
array(['1970-01-01T00:00:00.000002010', '1970-01-01T00:00:00.000002011',
       '1970-01-01T00:00:00.000002012', '1970-01-01T00:00:00.000002013',
       '1970-01-01T00:00:00.000002014', '1970-01-01T00:00:00.000002015',
       '1970-01-01T00:00:00.000002016', '1970-01-01T00:00:00.000002017',
       '1970-01-01T00:00:00.000002018'], dtype='datetime64[ns]')
```

In [35]:

```
# Enginnering a new feature that groups average total gross by studio

mean_total_gross = gross.groupby(['studio'])['total_gross'].mean()
```

In [36]:

```
# Statistical summary of mean total gross

mean_total_gross.describe()
```

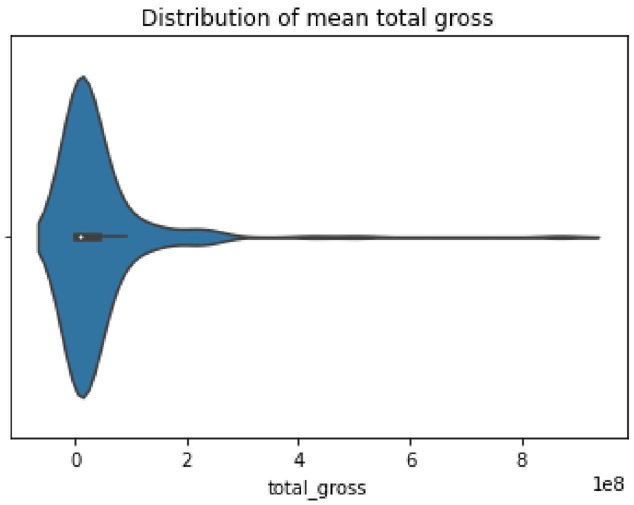
Out[36]:

```
count    1.720000e+02
mean     4.123883e+07
std      9.474540e+07
min      3.830000e+04
```

```
25%      1.934200e+06
50%      8.751622e+06
75%      3.855026e+07
max      8.703000e+08
Name: total_gross, dtype: float64
```

In [37]:

```
# Generating a violin plot to understand the distribution of this feature

sns.violinplot(x=mean_total_gross)
plt.title("Distribution of mean total gross");
```



In [38]:

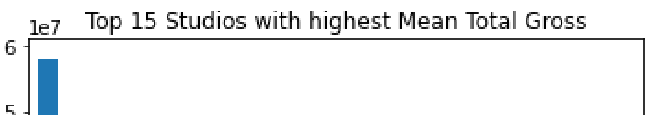
```
# Retrieving the studios with the highest average total gross

mean_total_gross.dropna().head(20).sort_values(ascending=False)
```

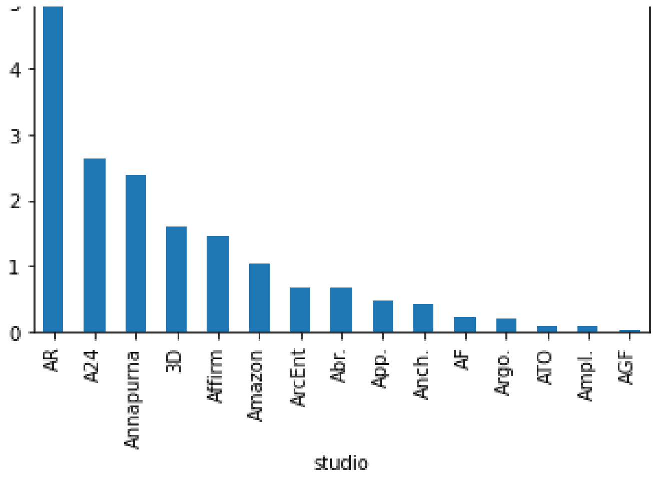
Out[38]:

```
studio
AR            5.805000e+07
A24           2.625889e+07
Annapurna     2.380000e+07
3D            1.600000e+07
BG            1.530033e+07
Affirm        1.452000e+07
BBC           1.168130e+07
Amazon        1.036000e+07
ArcEnt        6.813000e+06
Abr.          6.723500e+06
App.          4.700000e+06
Anch.         4.122950e+06
Aviron        3.800000e+06
AF            2.327500e+06
Argo.         1.829300e+06
ATO           9.241000e+05
Ampl.         7.710000e+05
BGP           3.051000e+05
Arth.         2.767000e+05
AGF           1.768000e+05
Name: total_gross, dtype: float64
```

In [39]:

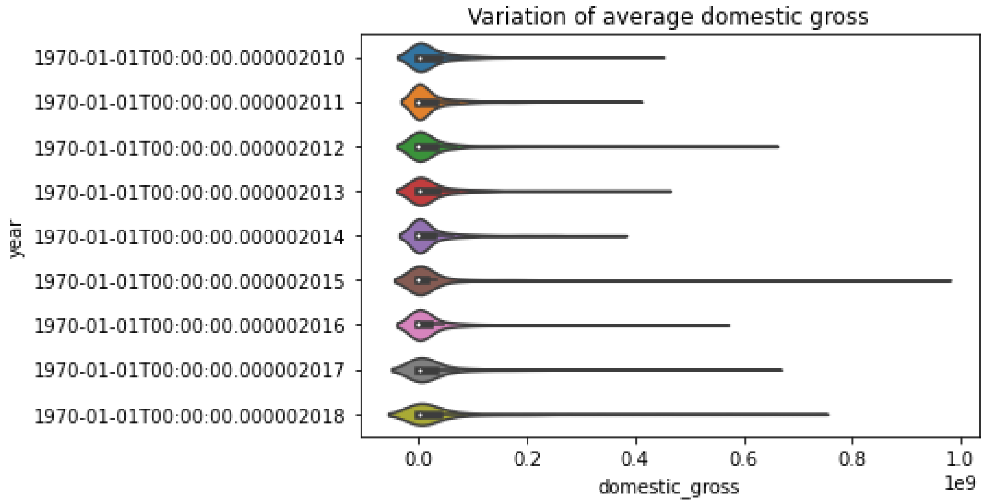
```
# Visualizing the studios with the highest average total gross

mean_total_gross.dropna().head(15).sort_values(ascending=False).plot(kind='bar')
plt.title("Top 15 Studios with highest Mean Total Gross");
```





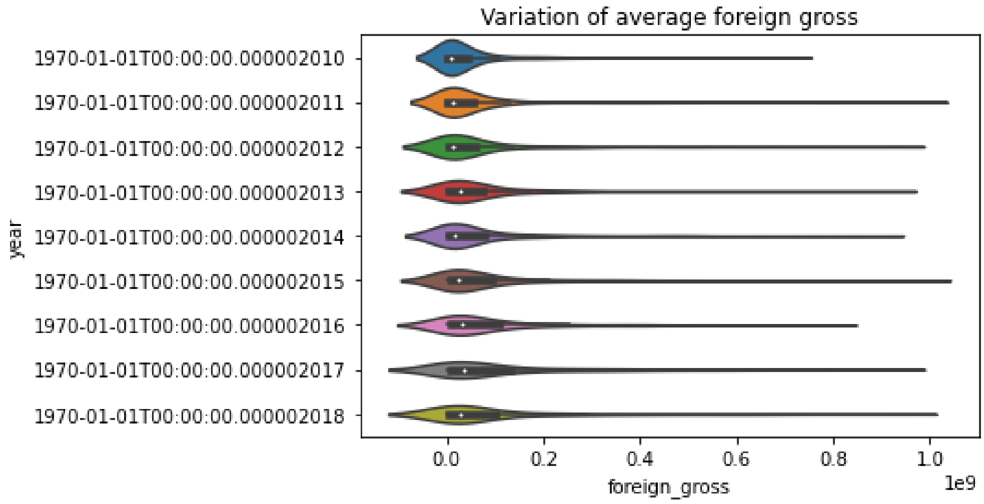
In [40]:

```
# Generate a plot to show how domestic gross varies over the years

sns.violinplot(data=gross, x=gross['domestic_gross'], y=gross['year'] , inner='box')
plt.title("Variation of average domestic gross");
```



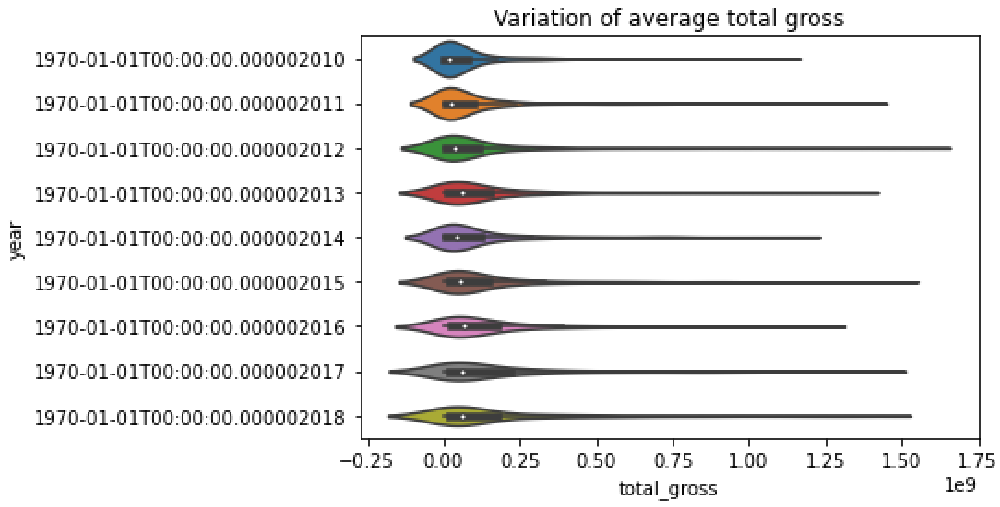
In [41]:

```
# Generate a plot to show how foreign gross varies over the years

sns.violinplot(x=gross['foreign_gross'], y=gross['year'] , inner='box')
plt.title("Variation of average foreign gross");
```



In [42]:

```
# Generate a plot to show how total gross varies over the years

sns.violinplot(x=gross['total_gross'], y=gross['year'] , inner='box')
plt.title("Variation of average total gross");
```



Key Insight(s)

- **Top studios have an average movie output of 10 - 15 project every years**
- **The top 15 studios have annual average gross of over one million dollars**
- **Over the years, the average total gross of movies has been increasing.**

titles_df & ratings_df

We can observe from initial review of the dataframes above that there is a common column ('tconst') between these datasets, hence, we can perform a JOIN opertaion in order to consolidate the data into one DataFrame for easier analysis.

In [43]:

```
# Set the index of the DataFrame to 'tconst'

titles.set_index('tconst', inplace=True)
```

In [44]:

```
# Set the index of the DataFrame to 'tconst'

ratings.set_index('tconst', inplace=True)
```

In [45]:

```
# Join the 2 DataFrames

titles_and_ratings = titles.join(ratings, how = 'left')
```

In [46]:

```
#Inspect the new DataFrame to ensure the join was successful
titles_and_ratings.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 146144 entries, tt0063540 to tt9916754
Data columns (total 7 columns):
 #   Column           Non-Null Count   Dtype  
---  ------           --------------   -----  
 0   primary_title    146144 non-null  object 
 1   original_title   146123 non-null  object 
 2   start_year       146144 non-null  int64  
 3   runtime_minutes  114405 non-null  float64
 4   genres           140736 non-null  object 
 5   averagerating    73856 non-null   float64
 6   numvotes         73856 non-null   float64
dtypes: float64(3), int64(1), object(3)
memory usage: 13.9+ MB
```

In [47]:

```
titles_and_ratings.head(3)
```

Out[47]:

	primary_title	original_title	start_year	runtime_minutes	genres	averagerating	numvotes
tconst							
tt0063540	Sunghursh	Sunghursh	2013	175.0	Action,Crime,Drama	7.0	77.0
tt0066787	One Day Before the Rainy Season	Ashad Ka Ek Din	2019	114.0	Biography,Drama	7.2	43.0
tt0069049	The Other Side of the Wind	The Other Side of the Wind	2018	122.0	Drama	6.9	4517.0

In [48]:

```
type(titles_and_ratings['genres'].iloc[0])
```

Out[48]:

```
str
```

In [49]:

```
titles_and_ratings['genres'].nunique()
```

Out[49]:

```
1085
```

In [50]:

```
t_r = titles_and_ratings.unstack()
```

In [51]:

```
t_r.head()
```

Out[51]:

```
               tconst
primary_title  tt0063540                          Sunghursh
               tt0066787    One Day Before the Rainy Season
               tt0069049         The Other Side of the Wind
               tt0069204                     Sabse Bada Sukh
               tt0100275          The Wandering Soap Opera
dtype: object
```

In [52]:

```
t_r.groupby(['start_year']).head()
```

Out[52]:

```
               tconst
primary_title  tt0063540                          Sunghursh
               tt0066787    One Day Before the Rainy Season
               tt0069049         The Other Side of the Wind
               tt0069204                     Sabse Bada Sukh
               tt0100275          The Wandering Soap Opera
dtype: object
```

In [53]:

```
titles_and_ratings.groupby(['start_year']).mean().sort_values(by='start_year')
#plot(kind='bar')
```

Out[53]:

	runtime_minutes	averagerating	numvotes
start_year			
2010	85.495694	6.259585	4488.480418
2011	86.410106	6.290134	4431.113953
2012	89.208856	6.297057	4261.238932
2013	84.931670	6.287259	4460.397622
2014	84.541500	6.319806	4107.310238
2015	85.407108	6.265894	3080.688721
2016	84.974249	6.347300	3052.597523
2017	85.732214	6.397624	2513.674280
2018	87.661099	6.415599	2193.447914
2019	90.887358	6.703578	1408.505046
2020	91.280488	NaN	NaN
2021	101.750000	NaN	NaN
2022	109.666667	NaN	NaN
2023	NaN	NaN	NaN
2024	NaN	NaN	NaN
2025	NaN	NaN	NaN
2026	NaN	NaN	NaN
2027	NaN	NaN	NaN
2115	NaN	NaN	NaN

Key Insight(s)

We can discern from the above dataframe that the runtime for movies has been staedily increasing whereas despite the avrage ratings obtained for films has been relatively constant over the years. We can also notice that the number of critics voting for films has reduced.

budgets_df

In [54]:

```
budgets.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5782 entries, 1 to 82
Data columns (total 5 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   release_date       5782 non-null   datetime64[ns]
 1   movie              5782 non-null   object
 2   production_budget  5782 non-null   float64
 3   domestic_gross     5782 non-null   float64
 4   worldwide_gross    5782 non-null   float64
dtypes: datetime64[ns](1), float64(3), object(1)
memory usage: 271.0+ KB
```

In [55]:

```
# Engineering a new feature 'roi'(Return on Investment)
budgets['roi'] = round((budgets['worldwide_gross'] - budgets['production_budget']) / bud
gets['production_budget'] ,2)
```

In [56]:

```
budgets.head()
```

Out[56]:

	release_date	movie	production_budget	domestic_gross	worldwide_gross	roi
id						
1	2009-12-18	Avatar	425000000.0	760507625.0	2.776345e+09	5.53
2	2011-05-20	Pirates of the Caribbean: On Stranger Tides	410600000.0	241063875.0	1.045664e+09	1.55
3	2019-06-07	Dark Phoenix	350000000.0	42762350.0	1.497624e+08	-0.57
4	2015-05-01	Avengers: Age of Ultron	330600000.0	459005868.0	1.403014e+09	3.24
5	2017-12-15	Star Wars Ep. VIII: The Last Jedi	317000000.0	620181382.0	1.316722e+09	3.15

In [57]:

```
# Engineering a new feature 'release_year' to extract the year from each date variable pr
ovided into a new column

budgets['release_year'] = budgets['release_date'].dt.year
```

In [58]:

```
# Inspectring the dataframe to ensure the new column has been added

budgets.head()
```

Out[58]:

	release_date	movie	production_budget	domestic_gross	worldwide_gross	roi	release_year
id							
1	2009-12-18	Avatar	425000000.0	760507625.0	2.776345e+09	5.53	2009
2	2011-05-20	Pirates of the Caribbean: On Stranger Tides	410600000.0	241063875.0	1.045664e+09	1.55	2011
3	2019-06-07	Dark Phoenix	350000000.0	42762350.0	1.497624e+08	-0.57	2019
4	2015-05-01	Avengers: Age of Ultron	330600000.0	459005868.0	1.403014e+09	3.24	2015
5	2017-12-15	Star Wars Ep. VIII: The Last Jedi	317000000.0	620181382.0	1.316722e+09	3.15	2017

In [59]:

```
# Number of unique years in the dataframe

budgets['release_year'].nunique()
```

Out[59]:

```
96
```

We can see the that this dataset has 5782 recorded movies scattered over a period of 96 years. Plotting return on investment on a year by year basis would not be meaningful due to the large number of datapoints hence we consider the avreage return on investment over the entire period.

In [60]:

```
# Statistical summary of return on invest ment over the entire period

budgets['roi'].describe()
```

Out[60]:

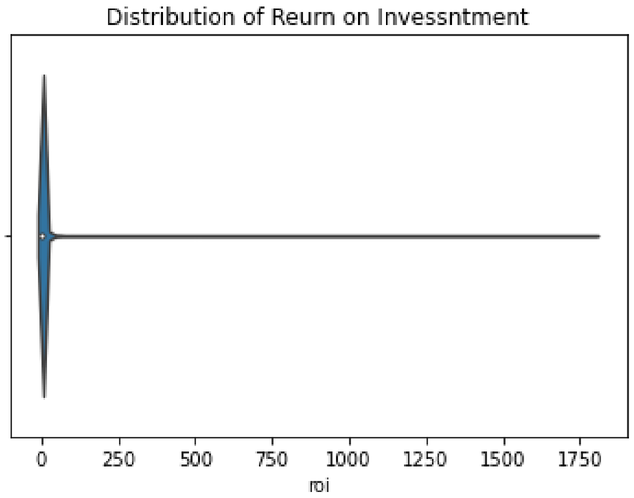
```
count    5782.000000
mean        3.800126
std        29.530251
min        -1.000000
```

```
25%          -0.510000
50%           0.710000
75%           2.760000
max        1799.000000
Name: roi, dtype: float64
```

From the above statistical summary we can observe that the mean return on investment expected from movies as a product is 3.8 (aprrox 4x). Most movies have a return varying grom 2.8 to -0.5 which shows that even if most movies are profitable, it is possible to make losses from the investment.

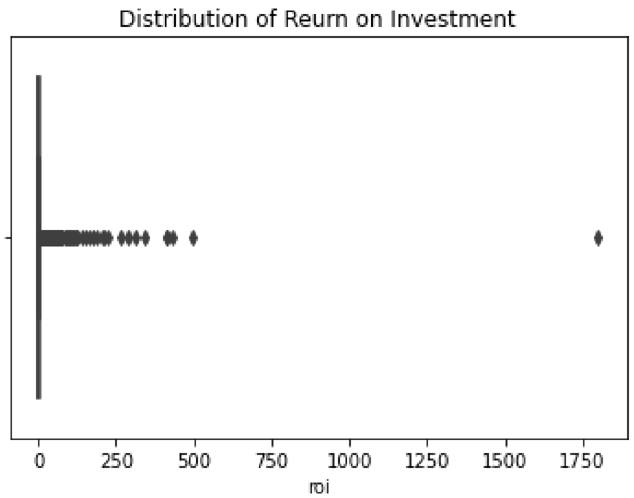
In [61]:

```
# Create a violin plot to visualize the distribution of data

sns.violinplot(x=budgets['roi'])
plt.title("Distribution of Reurn on Invessntment");
```



In [62]:

```
# Create a boxplot to idetify outliers in the data

sns.boxplot(x=budgets['roi'])
plt.title("Distribution of Reurn on Investment");
```



From the above we can see that even if majority of the data is loted around the mean, there are also alot of outlier points that exist in the dataset indicating that movies generating supernormal profits if not the norm but is possible.

In [63]:

```
# Group the data by year and obtain averages over nueriacal variables present

yearly_mean = budgets.groupby('release_year').mean()
```

In [64]:

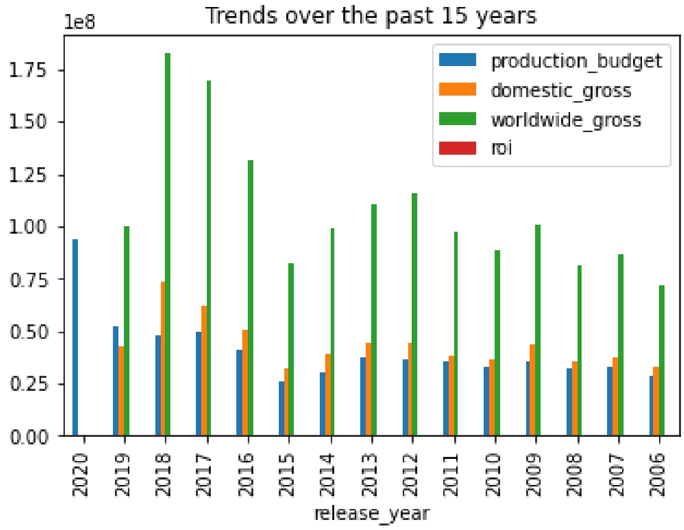
In [64]:

```
yearly_mean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 96 entries, 1915 to 2020
Data columns (total 4 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   production_budget  96 non-null     float64
 1   domestic_gross     96 non-null     float64
 2   worldwide_gross    96 non-null     float64
 3   roi                96 non-null     float64
dtypes: float64(4)
memory usage: 3.8 KB
```

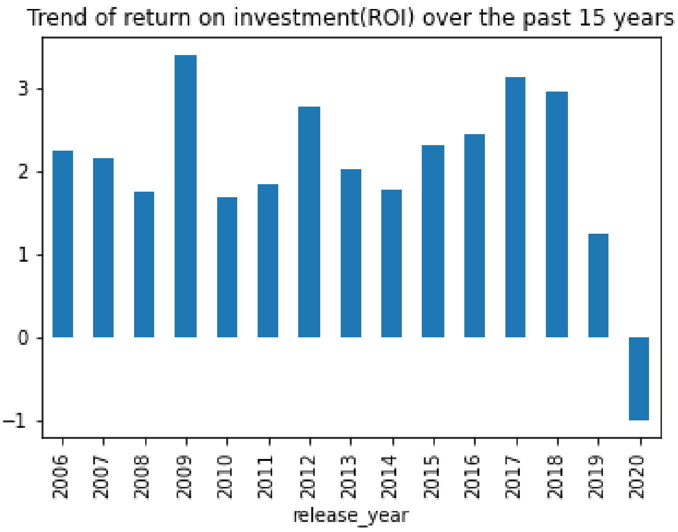
In [65]:

```
# Create a plot to see at the trends of averages for the past 15 years

budgets.groupby('release_year').mean().sort_values(by='release_year', ascending=False).h
ead(15).plot(kind='bar')
plt.title("Trends over the past 15 years");
```



In [66]:

```
#Create a plot to visualize how the average return on investment has varied over the past


yearly_mean['roi'].iloc[81:96].plot(kind = 'bar')
plt.title("Trend of return on investment(ROI) over the past 15 years");
```



Key Insight(s)

We can see that with the exception of 2020, return on investments for movies average above 1.5x which is very

We can see that with the exception of 2020, return on investments for movies average above 1.5x which is very decent return on investment on a yearly basis. The negative ROI observed in 2020 can be attributed to the adverse effects of the coronavirus pandemic on the moviemaking industry and world at large at the time.

We can observe from the visualizations that production budgets have been increasing over the years, but so has both worldwide and domestic gross. We can also see that worldwide gross for movies has been increasing steadily, then quite sharply in recent years. This is a good indicator of growth in market size and consumer apetite for movies around the world, mainly facilitated by the penetration of high quality internet to the masses.

RECOMMENDATIONS

- The gross numbers(domestic, foreign & total) as well as return on investment numbers indicate a market of movies that is on the rise, hence the decision to create astudio company is well supported by the data in this report.
- The studio should aim for an output of upwards of 15 movie projects on an annual basis in order to stay competitive with the most dominant players in the market such as A24 studios, Amazon Studios, Annapurna Studios etc.
- Microsoft movie studio should take advantage of the current state of the market that is still recovering from the effects of the coronavirus pandemic to launch a competitive studio and ride the wave to success as return on investment from movies should return to pre-pandemic margins in the next few years.
- Microsoft should invest in making movies across al genres in order to attract a more diverse audience hence increasing likelihood for higher gross values.
- Content created on this studio platform should have runtimes that average 90 - 100 minutes.

CONCLUSION

Movies enjoy a special place in everyone's memories as both an adult and as a child, hence, as a product it is marketable across all ages. The data analysed in this report has shown that they enjoy steady average return rates of upwards of 1.5x with ever increasing gross both locally and internationally. The product is however unique in that outliers are not uncommon and movies, depending on factors such as popularity, cultural influence, timing, hidden messages and other nuanced intangible factors; can result in a singular movie product generating suprenormal profits of up to 1000x return on investment. Although this should not be the goal, engaging in the industry would be a way to engineer Microsoft's luck towards such a lofty objective.