Microsoft Video Content Expansion

by: Lauren Coats Flatiron FT Data Science

Business Problem

This notebook goes over an in-depth analyzation of top 150 movies, based on ROI from the past 10 years. Concluding in Professional recommendations for Microsoft that will aid them in making profitable business decisions regarding the creation of original video content.

Business Value

This project is the next logical step for Microsoft to exand and diverifiy there holdings. If done right this will impress shareholders and drive stock prices up.

Importing Data

The data used for this project was scraped from www.imdb.com, a webside known for producing reviews and giving a few movie metrics as seen below.

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
import os
import sqlite3
import seaborn as sns


from glob import glob
```

In [2]:
```
cd~
```

```
C:\Users\laure
```

In [3]:
```
cd Flatiron\dsc-project-template\dsc-phase-1-project
```

```
C:\Users\laure\Flatiron\dsc-project-template\dsc-phase-1-project
```

In [4]:
```
pwd
```

Out[4]:
```
'C:\\Users\\laure\\Flatiron\\dsc-project-template\\dsc-phase-1-project'
```

In [5]:
```
csv_files = glob("./zippedData/*.csv")
csv_files
```

Out[5]:
```
['./zippedData\\bom.movie_gross.csv',
 './zippedData\\imdb.name.basics.csv',
 './zippedData\\imdb.title.akas.csv',
 './zippedData\\imdb.title.basics.csv',
 './zippedData\\imdb.title.crew.csv',
 './zippedData\\imdb.title.principals.csv',
 './zippedData\\imdb.title.ratings.csv',
```

```
 './zippedData\\tmdb.movies.csv',
 './zippedData\\tn.movie_budgets.csv']
```

In [6]:
```
csv_files[0]
```

Out[6]:
```
'./zippedData\\bom.movie_gross.csv'
```

In [7]:
```
csv_read = {}
```

In [8]:
```
for file in csv_files:
    csv_read[file] = pd.read_csv(file)
```

In [9]:
```
type(csv_read)
```

Out[9]:
```
dict
```

In [10]:
```
csv_read.keys()
```

Out[10]:
```
dict_keys(['./zippedData\\bom.movie_gross.csv', './zippedData\\imdb.name.basics.csv',
'./zippedData\\imdb.title.akas.csv', './zippedData\\imdb.title.basics.csv', './zippedDat
a\\imdb.title.crew.csv', './zippedData\\imdb.title.principals.csv', './zippedData\\imdb.
title.ratings.csv', './zippedData\\tmdb.movies.csv', './zippedData\\tn.movie_budgets.cs
v'])
```

In [11]:
```
csv_files_dict = {}
for filename in csv_files:
    filename_cleaned = os.path.basename(filename).replace('.csv','').replace('.','_')
    filename_df = pd.read_csv(filename,index_col=0)
    csv_files_dict[filename_cleaned] = filename_df
```

In [12]:
```
csv_files_dict.keys()
```

Out[12]:
```
dict_keys(['bom_movie_gross', 'imdb_name_basics', 'imdb_title_akas', 'imdb_title_basic
s', 'imdb_title_crew', 'imdb_title_principals', 'imdb_title_ratings', 'tmdb_movies', 'tn
_movie_budgets'])
```

Cleaning Tables

In order to get useable information from the data it must first be clean, by orginizing it in a way that can be graphed and charted.

Below is a group of functions that was used repeated throughout the cleaning process.

In [13]:
```
#Convert Object into Str
def convert_to_str(df, col):
    df[col] = df[col].astype('str')
    return df.info()
```

In [14]:
```
#Convert object to date YYYY-MM-DD
def convert_dates(df, col):
    df[col] = pd.to_datetime(df[col])
    return df.info()
```

In [15]:
```
#convert NaN and None and none to 0
def convert_0(df, col):
    df[col] = df[col].str.replace('NaN', 0).str.replace('None' , 0)\
        .str.replace('none' , 0).astype('int64')
```

```
    df=df[col].str.fillna(0, inplace = True).astype('int64')
    return df.sort_values(col)
```

In [16]:
```
def convert_to_int(df, col):
    df[col] = df[col].str.replace('$', '').str.replace(',', '').astype('int64')
    return df.info()
```

In [17]:
```
def remove_rows(df, col, index):
    df = df.loc[df[col] != index]
#     inplace=True
    df.info()
    return df
```

In [18]:
```
def sort_values(df, col):
    df.sort_values(col)
```

bom_movie_gross

- Need to change types to reflect what the data type is so that it can be compared with other numbers. Also the NaN with in foreign gross needs to be converted to 0

In [19]:
```
bom_movie_gross_df = csv_files_dict['bom_movie_gross']
bom_movie_gross_df.info()
bom_movie_gross_df.sort_values('foreign_gross')
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 3387 entries, Toy Story 3 to An Actor Prepares
Data columns (total 4 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   studio          3382 non-null   object
 1   domestic_gross  3359 non-null   float64
 2   foreign_gross   2037 non-null   object
 3   year            3387 non-null   int64 
dtypes: float64(1), int64(1), object(2)
memory usage: 132.3+ KB
```

Out[19]:

title	studio	domestic_gross	foreign_gross	year
The Fate of the Furious	Uni.	226000000.0	1,010.0	2017
Jurassic World	Uni.	652300000.0	1,019.4	2015
Star Wars: The Force Awakens	BV	936700000.0	1,131.6	2015
Furious 7	Uni.	353000000.0	1,163.0	2015
Avengers: Infinity War	BV	678800000.0	1,369.5	2018
...	...	...	...	...
The Quake	Magn.	6200.0	NaN	2018
Edward II (2018 re-release)	FM	4800.0	NaN	2018
El Pacto	Sony	2500.0	NaN	2018
The Swan	Synergetic	2400.0	NaN	2018

title	studio	domestic_gross	foreign_gross	year
An Actor Prepares	Grav.	1700.0	NaN	2018

3387 rows × 4 columns

In [20]:
```
bom_movie_gross_df['foreign_gross'] = bom_movie_gross_df['foreign_gross'].fillna(0)
```

imdb_title_akas

-Set original to 1 and get rid of the rest
(use to get the name of movie in another cell)

In [21]:
```
#Creating df for imdb_title_akas
imdb_title_akas_df = csv_files_dict['imdb_title_akas']
imdb_title_akas_df.info()
imdb_title_akas_df['is_original_title'] = imdb_title_akas_df['is_original_title'].filln
imdb_title_akas_df.sort_values('ordering')
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 331703 entries, tt0369610 to tt9880178
Data columns (total 7 columns):
 #   Column             Non-Null Count   Dtype  
---  ------             --------------   -----  
 0   ordering           331703 non-null  int64  
 1   title              331703 non-null  object 
 2   region             278410 non-null  object 
 3   language           41715 non-null   object 
 4   types              168447 non-null  object 
 5   attributes         14925 non-null   object 
 6   is_original_title  331678 non-null  float64
dtypes: float64(1), int64(1), object(5)
memory usage: 20.2+ MB
```

Out[21]:

title_id	ordering	title	region	language	types	attributes	is_original_title
tt1529241	1	Buenavista (Ang kasaysayan ng Lucena)	PH	NaN	NaN	NaN	0.0
tt3497340	1	Tenacity and Gratitude: The Frank Cotolo Story	NaN	NaN	original	NaN	1.0
tt3498716	1	The Art Lives	US	NaN	NaN	NaN	0.0
tt3499896	1	The Philosopher and the Faithful	FR	NaN	NaN	NaN	0.0
tt3500702	1	Excision	GR	NaN	NaN	NaN	0.0
...	...	...	...	...	...	...	...
tt2488496	57	Star Wars: Das Erwachen der Macht	DE	NaN	alternative	NaN	0.0

title_id	ordering	title	region	language	types	attributes	is_original_title
tt2488496	58	Star Wars: O Despertar da Força	BR	NaN	imdbDisplay	NaN	0.0
tt2488496	59	Star Wars: Episódio VII - O Despertar da Força	PT	NaN	imdbDisplay	NaN	0.0
tt2488496	60	Star Wars: Güç Uyaniyor	TR	tr	imdbDisplay	NaN	0.0
tt2488496	61	Žvaigždžiu karai: galia nubunda	LT	NaN	NaN	NaN	0.0

331703 rows × 7 columns

In [22]:
```
def remove_rows(df, col, index):
    df = df.loc[df[col] != index]
#     inplace=True
    df.info()
    return df

# tn_movie_budgets_df = remove_rows(tn_movie_budgets_df, 'worldwide_gross'  , 0)
# tn_movie_budgets_df.sort_values('worldwide_gross')
imdb_title_akas_df = remove_rows(imdb_title_akas_df, 'is_original_title'  , 0.0)
imdb_title_akas_df.sort_values('is_original_title')
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 44700 entries, tt0369610 to tt9880178
Data columns (total 7 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   ordering           44700 non-null  int64  
 1   title              44700 non-null  object 
 2   region             6 non-null      object 
 3   language           4 non-null      object 
 4   types              44700 non-null  object 
 5   attributes         0 non-null      object 
 6   is_original_title  44700 non-null  float64
dtypes: float64(1), int64(1), object(5)
memory usage: 2.7+ MB
```

Out[22]:

title_id	ordering	title	region	language	types	attributes	is_original_title
tt0369610	45	Jurassic World	NaN	NaN	original	NaN	1.0
tt5474656	1	Deti 90-kh	NaN	NaN	original	NaN	1.0
tt5476912	4	Massakren i Dvor	NaN	NaN	original	NaN	1.0
tt5478700	2	Dagenham	NaN	NaN	original	NaN	1.0
tt5487052	3	Den vyborov 2	NaN	NaN	original	NaN	1.0
...	...	...	...	...	...	...	...
tt2265640	1	All Roads Lead	NaN	NaN	original	NaN	1.0
tt2274604	4	Eat with Me	NaN	NaN	original	NaN	1.0

title_id	ordering	title	region	language	types	attributes	is_original_title
tt2275667	1	Ein Versprechen	NaN	NaN	original	NaN	1.0
tt2258664	1	Chehel Salegi	NaN	NaN	original	NaN	1.0
tt9880178	1	La atención	NaN	NaN	original	NaN	1.0

44700 rows × 7 columns

imdb_title_basics

-remove years before 2010
-separate genres by comma

In [23]:
```
#Creating df for imdb_title_basics
imdb_title_basics_df = csv_files_dict['imdb_title_basics']
imdb_title_basics_df.info()
imdb_title_basics_df.sort_values('start_year')
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 146144 entries, tt0063540 to tt9916754
Data columns (total 5 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 
dtypes: float64(1), int64(1), object(3)
memory usage: 6.7+ MB
```

Out[23]:

tconst	primary_title	original_title	start_year	runtime_minutes	genres
tt1566491	Brainiacs in La La Land	Brainiacs in La La Land	2010	NaN	Comedy
tt2578092	Fireplace for your Home: Crackling Fireplace w...	Fireplace for your Home: Crackling Fireplace w...	2010	61.0	Music
tt1634300	Role/Play	Role/Play	2010	85.0	Drama,Romance
tt1634332	Johan1	Johan Primero	2010	78.0	Comedy,Drama,Romance
tt1634334	Hands Up	Les mains en l'air	2010	90.0	Drama
...	...	...	...	...	...
tt10300396	Untitled Star Wars Film	Untitled Star Wars Film	2024	NaN	NaN
tt3095356	Avatar 4	Avatar 4	2025	NaN	Action,Adventure,Fantasy
tt10300398	Untitled Star Wars Film	Untitled Star Wars Film	2026	NaN	Fantasy
tt5637536	Avatar 5	Avatar 5	2027	NaN	Action,Adventure,Fantasy

tconst	primary_title	original_title	start_year	runtime_minutes	genres
tt5174640	100 Years	100 Years	2115	NaN	Drama

146144 rows × 5 columns

In [24]:
```
imdb_title_basics_df['movie']=imdb_title_basics_df['original_title']
```

In [25]:
```
def remove_rows_greater_than(df, col, index):
    df = df.loc[df[col] <= index]
#     inplace=True
    df.info()
    return df

imdb_title_basics_df = remove_rows_greater_than(imdb_title_basics_df, 'start_year', 202
imdb_title_basics_df.sort_values('start_year')
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 146018 entries, tt0063540 to tt9916754
Data columns (total 6 columns):
 #   Column           Non-Null Count   Dtype  
---  ------           --------------   -----  
 0   primary_title    146018 non-null  object 
 1   original_title   145997 non-null  object 
 2   start_year       146018 non-null  int64  
 3   runtime_minutes  114398 non-null  float64
 4   genres           140622 non-null  object 
 5   movie            145997 non-null  object 
dtypes: float64(1), int64(1), object(4)
memory usage: 7.8+ MB
```

Out[25]:

tconst	primary_title	original_title	start_year	runtime_minutes	genres	movie
tt1579444	The Song Beneath the Song	To tragoudi kato apo to tragoudi	2010	89.0	Romance	To tragoudi kato apo to tragoudi
tt1543511	Mister Rogers & Me	Mister Rogers & Me	2010	79.0	Documentary	Mister Rogers & Me
tt1543566	New Year	New Year	2010	77.0	Comedy,Drama,Romance	New Year
tt1543597	Off World	Off World	2010	76.0	Drama	Off World
tt1543609	The Sark Case	The Sark Case	2010	75.0	Documentary	The Sark Case
...	...	...	...	...	...	...
tt10328454	Untitled Danny Ramirez Horror	Untitled Danny Ramirez Horror	2020	NaN	Horror	Untitled Danny Ramirez Horror

tconst	primary_title	original_title	start_year	runtime_minutes	genres	movie
tt6723592	Tenet	Tenet	2020	NaN	Action,Drama,Thriller	Tenet
tt10426048	Romeo + Juliet	Romeo + Juliet	2020	NaN	Drama,Romance,Thriller	Romeo + Juliet
tt10328534	Mr. Petty	Mr. Petty	2020	NaN	Comedy	Mr. Petty
tt3661394	The One and Only Ivan	The One and Only Ivan	2020	NaN	Animation,Comedy,Family	The One and Only Ivan

146018 rows × 6 columns

imdb_title_basics

-remove years before 2010
-separate genres by comma

In [26]:
```
#Creating df for imdb_title_principals
imdb_title_principals_df = csv_files_dict['imdb_title_principals']
imdb_title_principals_df.info()
imdb_title_principals_df.head()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1028186 entries, tt0111414 to tt9692684
Data columns (total 5 columns):
 #   Column      Non-Null Count    Dtype 
---  ------      --------------    ----- 
 0   ordering    1028186 non-null  int64 
 1   nconst      1028186 non-null  object
 2   category    1028186 non-null  object
 3   job         177684 non-null   object
 4   characters  393360 non-null   object
dtypes: int64(1), object(4)
memory usage: 47.1+ MB
```

Out[26]:

tconst	ordering	nconst	category	job	characters
tt0111414	1	nm0246005	actor	NaN	["The Man"]
tt0111414	2	nm0398271	director	NaN	NaN
tt0111414	3	nm3739909	producer	producer	NaN
tt0323808	10	nm0059247	editor	NaN	NaN
tt0323808	1	nm3579312	actress	NaN	["Beth Boothby"]

imdb_title_ratings

-get name from other table

In [27]:
```
#Creating df for imdb_title_ratings
imdb_title_ratings_df = csv_files_dict['imdb_title_ratings']
imdb_title_ratings_df.info()
imdb_title_ratings_df.head()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 73856 entries, tt10356526 to tt9894098
Data columns (total 2 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   averagerating  73856 non-null  float64
 1   numvotes       73856 non-null  int64  
dtypes: float64(1), int64(1)
memory usage: 1.7+ MB
```

Out[27]:

tconst	averagerating	numvotes
tt10356526	8.3	31
tt10384606	8.9	559
tt1042974	6.4	20
tt1043726	4.2	50352
tt1060240	6.5	21

tmdb_movies

-Table is fine

In [28]:
```
#Creating df for bom_movie_gross
tmdb_movies_df = csv_files_dict['tmdb_movies']
tmdb_movies_df.info()
tmdb_movies_df.head()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 26517 entries, 0 to 26516
Data columns (total 9 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   genre_ids          26517 non-null  object 
 1   id                 26517 non-null  int64  
 2   original_language  26517 non-null  object 
 3   original_title     26517 non-null  object 
 4   popularity         26517 non-null  float64
 5   release_date       26517 non-null  object 
 6   title              26517 non-null  object 
 7   vote_average       26517 non-null  float64
 8   vote_count         26517 non-null  int64  
dtypes: float64(2), int64(2), object(5)
memory usage: 2.0+ MB
```

Out[28]:

	genre_ids	id	original_language	original_title	popularity	release_date	title	vote_average
0	[12, 14, 10751]	12444	en	Harry Potter and the Deathly Hallows: Part 1	33.533	2010-11-19	Harry Potter and the Deathly Hallows: Part 1	7.7

	genre_ids	id	original_language	original_title	popularity	release_date	title	vote_average
1	[14, 12, 16, 10751]	10191	en	How to Train Your Dragon	28.734	2010-03-26	How to Train Your Dragon	7.7
2	[12, 28, 878]	10138	en	Iron Man 2	28.515	2010-05-07	Iron Man 2	6.8
3	[16, 35, 10751]	862	en	Toy Story	28.005	1995-11-22	Toy Story	7.9
4	[28, 878, 12]	27205	en	Inception	27.920	2010-07-16	Inception	8.3

tn_movie_budgets

```
-Remove all ($0 domestic gross AND $0 world wide)
-Remove Movies before 2010
-Change date to XXXX-MM-DD format
-Create a domestic and world wide profit line
-Change money in to floats (will need to remove $)
```

In [29]:
```
#Creating df for tn_movie_budgets
tn_movie_budgets_df = csv_files_dict['tn_movie_budgets']
tn_movie_budgets_df.info()
tn_movie_budgets_df.sort_values('domestic_gross')
```

```
<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
```

Out[29]:

id	release_date	movie	production_budget	domestic_gross	worldwide_gross
82	Oct 26, 2012	Mientras duermes	$5,000,000	$0	$9,109,597
9	Nov 4, 2005	Wal-Mart: The High Cost of Low Price	$1,500,000	$0	$58,692
8	Jul 20, 2018	Teefa in Trouble	$1,500,000	$0	$98,806
7	Oct 17, 2014	Housebound	$1,500,000	$0	$236,863
58	Dec 31, 2008	Bathory	$15,000,000	$0	$3,436,763
...	...	...	...	...	...
8	Jun 13, 1997	Hercules	$70,000,000	$99,112,101	$250,700,000

id	release_date	movie	production_budget	domestic_gross	worldwide_gross
74	Jun 4, 1999	Desert Blue	$5,000,000	$99,147	$99,147
43	Aug 3, 2018	Christopher Robin	$75,000,000	$99,215,042	$197,504,758
6	Feb 11, 2011	Gnomeo and Juliet	$36,000,000	$99,967,670	$193,737,977
12	Jul 4, 2008	AstÃ©rix aux Jeux Olympiques	$113,500,000	$999,811	$132,999,811

5782 rows × 5 columns

In [30]:
```
convert_to_int(tn_movie_budgets_df, 'production_budget')
```

```
<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   int64 
 3   domestic_gross     5782 non-null   object
 4   worldwide_gross    5782 non-null   object
dtypes: int64(1), object(4)
memory usage: 271.0+ KB
```

In [31]:
```
convert_to_int(tn_movie_budgets_df, 'domestic_gross')
```

```
<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   int64 
 3   domestic_gross     5782 non-null   int64 
 4   worldwide_gross    5782 non-null   object
dtypes: int64(2), object(3)
memory usage: 271.0+ KB
```

In [32]:
```
convert_to_int(tn_movie_budgets_df, 'worldwide_gross')
```

```
<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   int64 
 3   domestic_gross     5782 non-null   int64 
 4   worldwide_gross    5782 non-null   int64 
dtypes: int64(3), object(2)
memory usage: 271.0+ KB
```

In [33]:
```
def remove_rows_less_than(df, col, index):
    df = df.loc[df[col] >= index]
#     inplace=True
```

```
    df.info()
    return df
```

In [34]:
```
convert_to_str(tn_movie_budgets_df, 'movie')
```

```
<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   int64 
 3   domestic_gross     5782 non-null   int64 
 4   worldwide_gross    5782 non-null   int64 
dtypes: int64(3), object(2)
memory usage: 271.0+ KB
```

In [35]:
```
convert_dates(tn_movie_budgets_df, 'release_date')
```

```
<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   int64         
 3   domestic_gross     5782 non-null   int64         
 4   worldwide_gross    5782 non-null   int64         
dtypes: datetime64[ns](1), int64(3), object(1)
memory usage: 271.0+ KB
```

In [36]:
```
tn_movie_budgets_df = remove_rows_greater_than(tn_movie_budgets_df, 'release_date', '20
tn_movie_budgets_df.sort_values('release_date')
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5780 entries, 1 to 82
Data columns (total 5 columns):
 #   Column             Non-Null Count  Dtype         
---  ------             --------------  -----         
 0   release_date       5780 non-null   datetime64[ns]
 1   movie              5780 non-null   object        
 2   production_budget  5780 non-null   int64         
 3   domestic_gross     5780 non-null   int64         
 4   worldwide_gross    5780 non-null   int64         
dtypes: datetime64[ns](1), int64(3), object(1)
memory usage: 270.9+ KB
```

Out[36]:

id	release_date	movie	production_budget	domestic_gross	worldwide_gross
78	1915-02-08	The Birth of a Nation	110000	10000000	11000000
24	1916-09-05	Intolerance	385907	0	0
15	1916-12-24	20,000 Leagues Under the Sea	200000	8000000	8000000
84	1920-09-17	Over the Hill to the Poorhouse	100000	3000000	3000000
7	1925-11-19	The Big Parade	245000	11000000	22000000
...	...	...	...	...	...
13	2019-12-31	Rogue City	13000000	0	0

id	release_date	movie	production_budget	domestic_gross	worldwide_gross
16	2019-12-31	Eli	11000000	0	0
30	2019-12-31	Reagan	25000000	0	0
81	2019-12-31	Army of the Dead	90000000	0	0
36	2020-02-21	Call of the Wild	82000000	0	0

5780 rows × 5 columns

In [37]:
```
tn_movie_budgets_df = remove_rows_less_than(tn_movie_budgets_df, 'release_date', '2010-
tn_movie_budgets_df.sort_values('production_budget')
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2192 entries, 2 to 81
Data columns (total 5 columns):
 #   Column             Non-Null Count  Dtype         
---  ------             --------------  -----         
 0   release_date       2192 non-null   datetime64[ns]
 1   movie              2192 non-null   object        
 2   production_budget  2192 non-null   int64         
 3   domestic_gross     2192 non-null   int64         
 4   worldwide_gross    2192 non-null   int64         
dtypes: datetime64[ns](1), int64(3), object(1)
memory usage: 102.8+ KB
```

Out[37]:

id	release_date	movie	production_budget	domestic_gross	worldwide_gross
81	2015-09-29	A Plague So Pleasant	1400	0	0
78	2018-12-31	Red 11	7000	0	0
73	2012-01-13	Newlyweds	9000	4584	4584
72	2015-05-19	Family Motocross	10000	0	0
61	2010-04-02	Breaking Upwards	15000	115592	115592
...	...	...	...	...	...
6	2015-12-18	Star Wars Ep. VII: The Force Awakens	306000000	936662225	2053311220
5	2017-12-15	Star Wars Ep. VIII: The Last Jedi	317000000	620181382	1316721747
4	2015-05-01	Avengers: Age of Ultron	330600000	459005868	1403013963
3	2019-06-07	Dark Phoenix	350000000	42762350	149762350
2	2011-05-20	Pirates of the Caribbean: On Stranger Tides	410600000	241063875	1045663875

2192 rows × 5 columns

In [38]:
```
def remove_rows(df, col, index):
    df = df.loc[df[col] != index]
#     inplace=True
```

```
    df.info()
    return df

tn_movie_budgets_df = remove_rows(tn_movie_budgets_df, 'worldwide_gross'  , 0)
tn_movie_budgets_df.sort_values('worldwide_gross')
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1922 entries, 2 to 73
Data columns (total 5 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   release_date       1922 non-null   datetime64[ns]
 1   movie              1922 non-null   object
 2   production_budget  1922 non-null   int64
 3   domestic_gross     1922 non-null   int64
 4   worldwide_gross    1922 non-null   int64
dtypes: datetime64[ns](1), int64(3), object(1)
memory usage: 90.1+ KB
```

Out[38]:

id	release_date	movie	production_budget	domestic_gross	worldwide_gross
3	2015-12-11	American Hero	1000000	0	26
36	2014-11-21	Food Chains	913000	0	176
84	2014-12-31	Destiny	750000	0	450
67	2018-05-11	Higher Power	500000	528	528
82	2010-05-21	Perrierâ□□s Bounty	6600000	828	828
...	...	...	...	...	...
27	2012-05-04	The Avengers	225000000	623279547	1517935897
67	2015-04-03	Furious 7	190000000	353007020	1518722794
34	2015-06-12	Jurassic World	215000000	652270625	1648854864
7	2018-04-27	Avengers: Infinity War	300000000	678815482	2048134200
6	2015-12-18	Star Wars Ep. VII: The Force Awakens	306000000	936662225	2053311220

1922 rows × 5 columns

In [39]:

```
def remove_rows_between(df, col, index, index2):
    df = df.loc[df[col] >= index]
    df = df.loc[df[col] <= index2]
#     inplace=True
    df.info()
    return df

tn_movie_budgets_df = remove_rows_between(tn_movie_budgets_df, 'release_date'  , '2010-
tn_movie_budgets_df.sort_values('release_date')
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1922 entries, 2 to 73
Data columns (total 5 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   release_date       1922 non-null   datetime64[ns]
 1   movie              1922 non-null   object
```

```
 2   production_budget  1922 non-null   int64
 3   domestic_gross     1922 non-null   int64
 4   worldwide_gross    1922 non-null   int64
dtypes: datetime64[ns](1), int64(3), object(1)
memory usage: 90.1+ KB
```

Out[39]:

id	release_date	movie	production_budget	domestic_gross	worldwide_gross
13	2010-01-08	Youth in Revolt	18000000	15285588	19685588
40	2010-01-08	Leap Year	19000000	25918920	32618920
67	2010-01-08	Daybreakers	20000000	30101577	51445503
36	2010-01-15	Fish Tank	3000000	374675	5922292
25	2010-01-15	The Last Station	18000000	6617867	15696146
...	...	...	...	...	...
3	2019-06-07	Dark Phoenix	350000000	42762350	149762350
81	2019-06-07	The Secret Life of Pets 2	80000000	63795655	113351496
98	2019-06-14	Shaft	30000000	600000	600000
42	2019-06-14	Men in Black: International	110000000	3100000	3100000
39	2019-06-21	Kursk	40000000	0	4212799

1922 rows × 5 columns

In [40]:
```
tn_movie_budgets_df['domestic_gross_profit'] = (0 - tn_movie_budgets_df['production_bud
tn_movie_budgets_df.sort_values('release_date')
```

Out[40]:

id	release_date	movie	production_budget	domestic_gross	worldwide_gross	domestic_gross_prof
13	2010-01-08	Youth in Revolt	18000000	15285588	19685588	-27144
40	2010-01-08	Leap Year	19000000	25918920	32618920	691892
67	2010-01-08	Daybreakers	20000000	30101577	51445503	101015
36	2010-01-15	Fish Tank	3000000	374675	5922292	-26253
25	2010-01-15	The Last Station	18000000	6617867	15696146	-113821
...	...	...	...	...	...	
3	2019-06-07	Dark Phoenix	350000000	42762350	149762350	-3072376
81	2019-06-07	The Secret Life of Pets 2	80000000	63795655	113351496	-162043
98	2019-06-14	Shaft	30000000	600000	600000	-294000

id	release_date	movie	production_budget	domestic_gross	worldwide_gross	domestic_gross_prof
42	2019-06-14	Men in Black: International	110000000	3100000	3100000	-1069000(
39	2019-06-21	Kursk	40000000	0	4212799	-4000000

1922 rows × 6 columns

In [41]:
```
tn_movie_budgets_df['worldwide_gross_profit'] = (0 - tn_movie_budgets_df['production_bu
tn_movie_budgets_df.sort_values('release_date')
```

Out[41]:

id	release_date	movie	production_budget	domestic_gross	worldwide_gross	domestic_gross_prof
13	2010-01-08	Youth in Revolt	18000000	15285588	19685588	-27144
40	2010-01-08	Leap Year	19000000	25918920	32618920	69189
67	2010-01-08	Daybreakers	20000000	30101577	51445503	101015
36	2010-01-15	Fish Tank	3000000	374675	5922292	-26253
25	2010-01-15	The Last Station	18000000	6617867	15696146	-113821
...	...	...	...	...	...	
3	2019-06-07	Dark Phoenix	350000000	42762350	149762350	-3072376
81	2019-06-07	The Secret Life of Pets 2	80000000	63795655	113351496	-162043
98	2019-06-14	Shaft	30000000	600000	600000	-2940000
42	2019-06-14	Men in Black: International	110000000	3100000	3100000	-1069000(
39	2019-06-21	Kursk	40000000	0	4212799	-4000000

1922 rows × 7 columns

In [42]:
```
tn_movie_budgets_df['domestic_roi_percent'] = round(tn_movie_budgets_df['domestic_gross
                                                    tn_movie_budgets_df['production_bud
tn_movie_budgets_df.sort_values('domestic_roi_percent')
```

Out[42]:

id	release_date	movie	production_budget	domestic_gross	worldwide_gross	domestic_gross_profit

id	release_date	movie	production_budget	domestic_gross	worldwide_gross	domestic_gross_profit
39	2017-09-29	Don Gato, el inicio de la pandilla	8000000	0	4604656	-8000000
32	2016-07-13	The Infiltrator	47500000	15436808	20718104	-32063192
12	2016-05-06	Dheepan	9000000	248795	7704357	-8751205
40	2010-06-18	Jonah Hex	47000000	10547117	11022696	-36452883
42	2016-12-23	Silence	46500000	7100177	23726626	-39399823
...	...	...	...	...	...	...
49	2017-02-24	Get Out	5000000	176040665	255367951	171040665
63	2011-04-01	Insidious	1500000	54009150	99870886	52509150
90	2014-03-21	Godâ□□s Not Dead	1150000	60755732	63777092	59605732
12	2012-01-06	The Devil Inside	1000000	53262945	101759490	52262945
80	2015-07-10	The Gallows	100000	22764410	41656474	22664410

1922 rows × 8 columns

In [43]:
```
#Remove max from tn_movie_budgets_df
tn_movie_budgets_df = tn_movie_budgets_df.loc[tn_movie_budgets_df['domestic_roi_percent
                                              ['domestic_roi_percent'].max()]
tn_movie_budgets_df.info()
tn_movie_budgets_df.sort_values('domestic_roi_percent')
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1921 entries, 2 to 73
Data columns (total 8 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   release_date           1921 non-null   datetime64[ns]
 1   movie                  1921 non-null   object
 2   production_budget      1921 non-null   int64
 3   domestic_gross         1921 non-null   int64
 4   worldwide_gross        1921 non-null   int64
 5   domestic_gross_profit  1921 non-null   int64
 6   worldwide_gross_profit 1921 non-null   int64
 7   domestic_roi_percent   1921 non-null   float64
dtypes: datetime64[ns](1), float64(1), int64(5), object(1)
memory usage: 135.1+ KB
```

Out[43]:

id	release_date	movie	production_budget	domestic_gross	worldwide_gross	domestic_gross_profit

id	release_date	movie	production_budget	domestic_gross	worldwide_gross	domestic_gross_profi
id						
39	2017-09-29	Don Gato, el inicio de la pandilla	8000000	0	4604656	-800000(
32	2016-07-13	The Infiltrator	47500000	15436808	20718104	-3206319:
12	2016-05-06	Dheepan	9000000	248795	7704357	-875120:
40	2010-06-18	Jonah Hex	47000000	10547117	11022696	-3645288:
42	2016-12-23	Silence	46500000	7100177	23726626	-3939982:
...	...	...	...	...	...	..
14	2015-04-17	Unfriended	1000000	32789645	64364198	3178964:
49	2017-02-24	Get Out	5000000	176040665	255367951	17104066:
63	2011-04-01	Insidious	1500000	54009150	99870886	5250915(
90	2014-03-21	Godâ□□s Not Dead	1150000	60755732	63777092	5960573:
12	2012-01-06	The Devil Inside	1000000	53262945	101759490	5226294:

1921 rows × 8 columns

In [44]:
```
#Remove max from tn_movie_budgets_df
tn_movie_budgets_df = tn_movie_budgets_df.loc[tn_movie_budgets_df['domestic_gross_profi
                                              ['domestic_gross_profit'].max()]
tn_movie_budgets_df.info()
tn_movie_budgets_df.sort_values('domestic_gross_profit')
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1920 entries, 2 to 73
Data columns (total 8 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   release_date           1920 non-null   datetime64[ns]
 1   movie                  1920 non-null   object
 2   production_budget      1920 non-null   int64
 3   domestic_gross         1920 non-null   int64
 4   worldwide_gross        1920 non-null   int64
 5   domestic_gross_profit  1920 non-null   int64
 6   worldwide_gross_profit 1920 non-null   int64
 7   domestic_roi_percent   1920 non-null   float64
dtypes: datetime64[ns](1), float64(1), int64(5), object(1)
memory usage: 135.0+ KB
```

Out[44]:

id	release_date	movie	production_budget	domestic_gross	worldwide_gross	domestic_gross_profit
3	2019-06-07	Dark Phoenix	350000000	42762350	149762350	-307237650

id	release_date	movie	production_budget	domestic_gross	worldwide_gross	domestic_gross_profit
14	2012-03-09	John Carter	275000000	73058679	282778100	-201941321
13	2013-07-02	The Lone Ranger	275000000	89302115	260002115	-185697885
2	2011-05-20	Pirates of the Caribbean: On Stranger Tides	410600000	241063875	1045663875	-169536125
32	2012-05-18	Battleship	220000000	65233400	313477717	-154766600
...	...	...	...	...	...	...
7	2018-04-27	Avengers: Infinity War	300000000	678815482	2048134200	378815482
27	2012-05-04	The Avengers	225000000	623279547	1517935897	398279547
44	2018-06-15	Incredibles 2	200000000	608581744	1242520711	408581744
34	2015-06-12	Jurassic World	215000000	652270625	1648854864	437270625
42	2018-02-16	Black Panther	200000000	700059566	1348258224	500059566

1920 rows × 8 columns

In [45]:
```
tn_movie_budgets_df['worldwide_roi_percent'] = round(tn_movie_budgets_df['worldwide_gro
                                                     tn_movie_budgets_df['production_budget'
tn_movie_budgets_df.sort_values('worldwide_roi_percent')
```

Out[45]:

id	release_date	movie	production_budget	domestic_gross	worldwide_gross	domestic_gross_profi
21	2015-01-20	Veronika Decides to Die	9000000	0	2243	-900000(
49	2010-04-30	Gunless	9900000	458054	458054	-944194(
5	2015-10-02	Shanghai	50000000	46425	15505922	-4995357
2	2015-04-17	Child 44	50000000	1224330	8004221	-4877567(
81	2016-06-24	Free State of Jones	50000000	20810036	23237252	-2918996
...	...	...	...	...	...	..

id	release_date	movie	production_budget	domestic_gross	worldwide_gross	domestic_gross_profi
50	2017-01-20	Split	5000000	138141585	278964806	13314158
65	2010-10-20	Paranormal Activity 2	3000000	84752907	177512032	8175290
14	2015-04-17	Unfriended	1000000	32789645	64364198	3178964
63	2011-04-01	Insidious	1500000	54009150	99870886	5250915
12	2012-01-06	The Devil Inside	1000000	53262945	101759490	5226294

1920 rows × 9 columns

TOP 150 Moives

We will be evaulating the top 100 movies bases on Domestic ROI

In [46]:
```
def IQR(df, col):
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1    #IQR is interquartile range.

    filter = (df[col] >= Q1 - 1.5 * IQR) & (df[col] <= Q3 + 1.5 *IQR)
    df = df.loc[filter]

    return df
```

Determining the Budget

Using the top 150 movies to come up with metrics for the movie such as Domestic Gross, Domestic Profit, Worldwide Gross, Worldwide profit, Domestic ROI, and Worldwide ROI. Once the mean of those numbers are found I can use them to determine the budget along with the metrics that would qualify this project as successful.

In [47]:
```
top150_gd = tn_movie_budgets_df.sort_values('domestic_gross').tail(150)
top150_gd = IQR(top150_gd, 'domestic_gross')
```

In [48]:
```
top150_wg = tn_movie_budgets_df.sort_values('worldwide_gross').tail(150)
top150_wg = IQR(top150_wg, 'worldwide_gross')
```

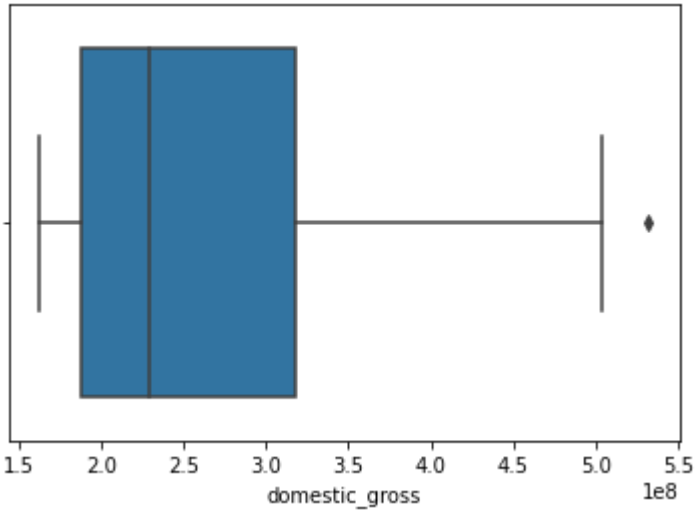
In [49]:
```
top150_dgp = tn_movie_budgets_df.sort_values('domestic_gross_profit').tail(150)
top150_dgp = IQR(top150_dgp, 'domestic_gross_profit')
```

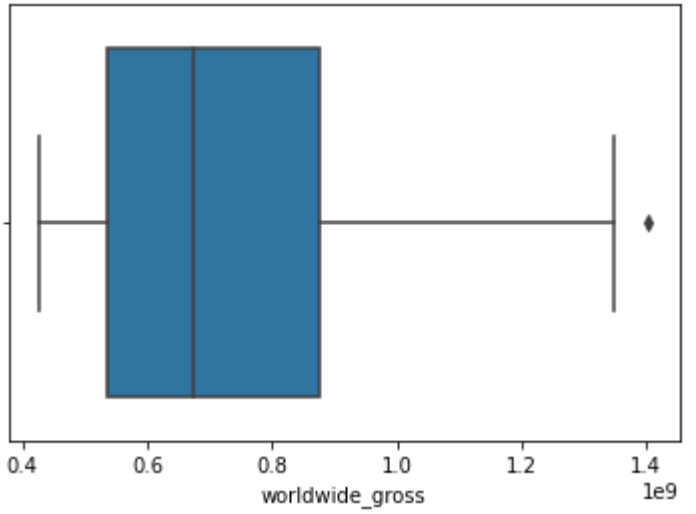
In [50]:
```
top150_wgp = tn_movie_budgets_df.sort_values('worldwide_gross_profit').tail(150)
top150_wgp = IQR(top150_wgp, 'worldwide_gross_profit')
```

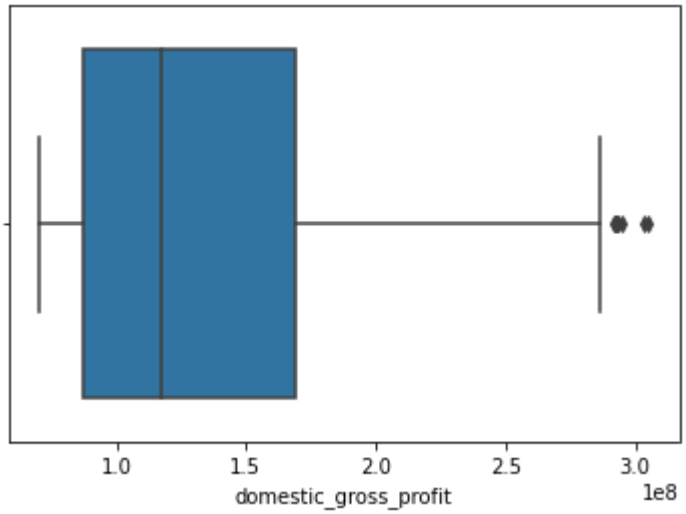
In [51]:
```
top150_drp = tn_movie_budgets_df.sort_values('domestic_roi_percent').tail(150)
top150_drp = IQR(top150_drp, 'domestic_roi_percent')
```

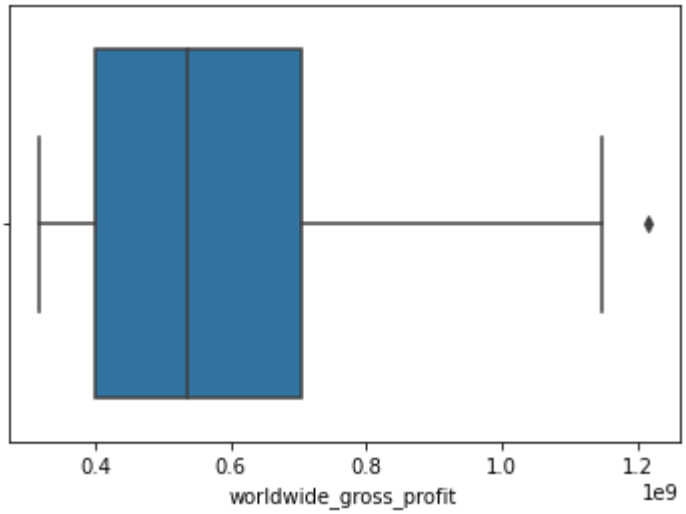
In [52]:
```
top150_wrp = tn_movie_budgets_df.sort_values('worldwide_roi_percent').tail(150)
top150_wrp = IQR(top150_wrp, 'worldwide_roi_percent')
```

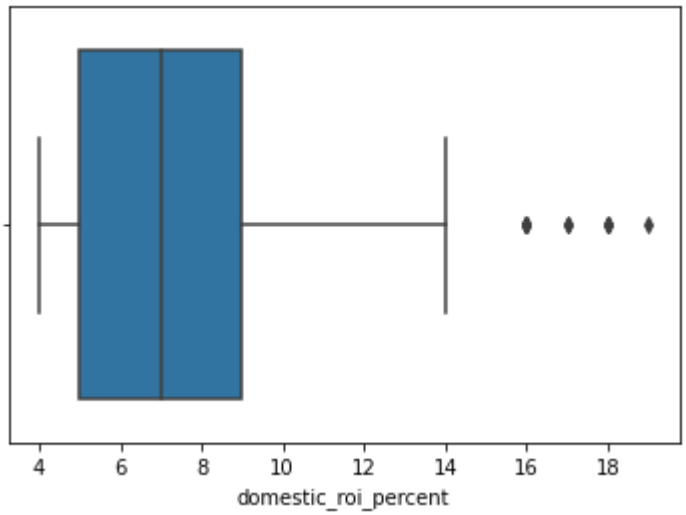
In [53]:
```
sns.boxplot(x=top150_gd['domestic_gross'])
plt.show()
sns.boxplot(x=top150_wg['worldwide_gross'])
plt.show()
sns.boxplot(x=top150_dgp['domestic_gross_profit'])
plt.show()
sns.boxplot(x=top150_wgp['worldwide_gross_profit'])
plt.show()
sns.boxplot(x=top150_drp['domestic_roi_percent'])
plt.show()
sns.boxplot(x=top150_wrp['worldwide_roi_percent'])
plt.show()
```

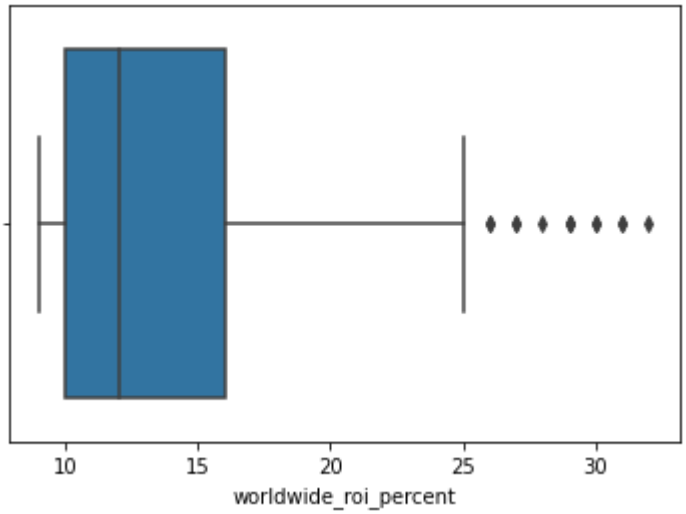





1.0
1.5
2.0
2.5
3.0
domestic_gross_profit
1e8


0.4
0.6
0.8
1.0
1.2
worldwide_gross_profit
1e9


4
6
8
10
12
14
16
18
domestic_roi_percent



In [54]:
```
print('top150_gd[domestic_gross]')
print(round(top150_gd['domestic_gross'].mean(),2))
print('top150_wg[worldwide_gross]')
print(round(top150_wg['worldwide_gross'].mean(),2))
print('top150_dgp[domestic_gross_profit]')
print(round(top150_dgp['domestic_gross_profit'].mean(),2))
print('top150_wgp[worldwide_gross_profit]')
print(round(top150_wgp['worldwide_gross_profit'].mean(),2))
print('top150_drp[domestic_roi_percent]')
print(round(top150_drp['domestic_roi_percent'].mean(),2))
print('top150_wrp[worldwide_roi_percent]')
print(round(top150_wrp['worldwide_roi_percent'].mean(),2))
```

```
top150_gd[domestic_gross]
257607969.4
top150_wg[worldwide_gross]
737928247.17
top150_dgp[domestic_gross_profit]
137324039.1
top150_wgp[worldwide_gross_profit]
587494350.54
top150_drp[domestic_roi_percent]
8.0
top150_wrp[worldwide_roi_percent]
14.42
```

Based on the above means the Budget was determine by subtracting the Domestic Gross Profit from the Domestic Gross.

In [55]:
```
round(top150_gd['domestic_gross'].mean(),2)\
 - round(top150_dgp['domestic_gross_profit'].mean(),2)
```

Out[55]:
```
120283930.30000001
```

Genere

In [56]:
```
top_150 = tn_movie_budgets_df.sort_values('domestic_roi_percent').tail(150)
top_150['year'] = pd.DatetimeIndex(top_150['release_date']).year
```

In [57]:
```
movie_details_df = pd.merge(top_150, imdb_title_basics_df, left_on = ['movie'], right_o
```

```
movie_details_df.dropna(inplace=True)
```

In [58]:
```
```

In [59]:
```
movie_details_df2 = pd.merge(top_150, imdb_title_basics_df, left_on= ['movie', 'year'],
                right_on = ['primary_title', 'start_year'], how = 'left')
```

In [60]:
```
movie_details_df2.shape
```

Out[60]:
```
(152, 16)
```

In [61]:
```
movie_details_df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 152 entries, 0 to 151
Data columns (total 16 columns):
 #   Column                 Non-Null Count  Dtype         
---  ------                 --------------  -----         
 0   release_date           152 non-null    datetime64[ns]
 1   movie_x                152 non-null    object        
 2   production_budget      152 non-null    int64         
 3   domestic_gross         152 non-null    int64         
 4   worldwide_gross        152 non-null    int64         
 5   domestic_gross_profit  152 non-null    int64         
 6   worldwide_gross_profit 152 non-null    int64         
 7   domestic_roi_percent   152 non-null    float64       
 8   worldwide_roi_percent  152 non-null    float64       
 9   year                   152 non-null    int64         
 10  primary_title          117 non-null    object        
 11  original_title         117 non-null    object        
 12  start_year             117 non-null    float64       
 13  runtime_minutes        117 non-null    float64       
 14  genres                 117 non-null    object        
 15  movie_y                117 non-null    object        
dtypes: datetime64[ns](1), float64(4), int64(6), object(5)
memory usage: 20.2+ KB
```

In [62]:
```
movie_details_df2.dropna(inplace=True)
movie_details_df2
```

Out[62]:

	release_date	movie_x	production_budget	domestic_gross	worldwide_gross	domestic_gross_prof
0	2016-01-22	The Boy	10000000	35819556	68220952	258195
1	2017-10-27	Jigsaw	10000000	38052832	102445196	280528
2	2018-03-16	Love, Simon	10000000	40826341	65520633	308263
3	2014-02-14	About Last Night	13000000	48637684	50445860	356376
4	2014-08-22	If I Stay	11000000	50474843	78356170	394748
...	...	...	...	...	...	
143	2010-08-27	The Last Exorcism	1800000	41034350	70165900	392343
144	2015-08-28	War Room	3000000	67790117	73975239	647901
145	2010-10-20	Paranormal Activity 2	3000000	84752907	177512032	817529
148	2017-02-24	Get Out	5000000	176040665	255367951	1710406

	release_date	movie_x	production_budget	domestic_gross	worldwide_gross	domestic_gross_prof
150	2012-01-06	The Devil Inside	1000000	53262945	101759490	522629

117 rows × 16 columns

In [63]:
```
movie_details_df['genres'] = movie_details_df['genres'].apply(lambda x: x.split(',') if
movie_details_df.head()
```

Out[63]:

	release_date	movie_x	production_budget	domestic_gross	worldwide_gross	domestic_gross_profit	v
0	2016-01-22	The Boy	10000000	35819556	68220952	25819556	
1	2016-01-22	The Boy	10000000	35819556	68220952	25819556	
2	2017-10-27	Jigsaw	10000000	38052832	102445196	28052832	
3	2018-03-16	Love, Simon	10000000	40826341	65520633	30826341	
4	2014-02-14	About Last Night	13000000	48637684	50445860	35637684	

In [64]:
```
movie_details_df2['genres'] = movie_details_df2['genres'].apply(lambda x: x.split(',')
movie_details_df.head()
```

Out[64]:

	release_date	movie_x	production_budget	domestic_gross	worldwide_gross	domestic_gross_profit	v
0	2016-01-22	The Boy	10000000	35819556	68220952	25819556	
1	2016-01-22	The Boy	10000000	35819556	68220952	25819556	
2	2017-10-27	Jigsaw	10000000	38052832	102445196	28052832	
3	2018-03-16	Love, Simon	10000000	40826341	65520633	30826341	
4	2014-02-14	About Last Night	13000000	48637684	50445860	35637684	

In [65]:
```
movie_details_df2['genres'] = movie_details_df2['genres'].astype(str)
movie_details_df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 117 entries, 0 to 150
Data columns (total 16 columns):
 #   Column                  Non-Null Count  Dtype         
---  ------                  --------------  -----         
 0   release_date            117 non-null    datetime64[ns]
 1   movie_x                 117 non-null    object        
 2   production_budget       117 non-null    int64         
 3   domestic_gross          117 non-null    int64         
 4   worldwide_gross         117 non-null    int64         
 5   domestic_gross_profit   117 non-null    int64         
 6   worldwide_gross_profit  117 non-null    int64         
 7   domestic_roi_percent    117 non-null    float64       
 8   worldwide_roi_percent   117 non-null    float64       
 9   year                    117 non-null    int64         
 10  primary_title           117 non-null    object        
 11  original_title          117 non-null    object        
 12  start_year              117 non-null    float64       
 13  runtime_minutes         117 non-null    float64       
 14  genres                  117 non-null    object        
 15  movie_y                 117 non-null    object        
dtypes: datetime64[ns](1), float64(4), int64(6), object(5)
memory usage: 15.5+ KB
```

In [66]:
```
#movie_details_df['genres'] = movie_details_df['genres'].astype(str)
```

In [67]:
```
all_genres = set()
for genres in movie_details_df['genres']:
    if genres:
        all_genres.update(genres)
```

In [68]:
```
all_genres
```

Out[68]:
```
{'Action',
 'Adventure',
 'Animation',
 'Biography',
 'Comedy',
 'Crime',
 'Documentary',
 'Drama',
 'Family',
 'Fantasy',
 'History',
 'Horror',
 'Music',
 'Mystery',
 'Romance',
 'Sci-Fi',
 'Sport',
 'Thriller',
 'Western'}
```

In [69]:
```
movie_details_df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
Int64Index: 117 entries, 0 to 150
Data columns (total 16 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   release_date           117 non-null    datetime64[ns]
 1   movie_x                117 non-null    object
 2   production_budget      117 non-null    int64
 3   domestic_gross         117 non-null    int64
 4   worldwide_gross        117 non-null    int64
 5   domestic_gross_profit  117 non-null    int64
 6   worldwide_gross_profit 117 non-null    int64
 7   domestic_roi_percent   117 non-null    float64
 8   worldwide_roi_percent  117 non-null    float64
 9   year                   117 non-null    int64
 10  primary_title          117 non-null    object
 11  original_title         117 non-null    object
 12  start_year             117 non-null    float64
 13  runtime_minutes        117 non-null    float64
 14  genres                 117 non-null    object
 15  movie_y                117 non-null    object
dtypes: datetime64[ns](1), float64(4), int64(6), object(5)
memory usage: 15.5+ KB
```

In [70]:
```
for genre in all_genres:
    movie_details_df2[genre] = np.zeros(shape=movie_details_df2.shape[0])

movie_details_df2.head()
```

Out[70]:

	release_date	movie_x	production_budget	domestic_gross	worldwide_gross	domestic_gross_profit	v
0	2016-01-22	The Boy	10000000	35819556	68220952	25819556	
1	2017-10-27	Jigsaw	10000000	38052832	102445196	28052832	
2	2018-03-16	Love, Simon	10000000	40826341	65520633	30826341	
3	2014-02-14	About Last Night	13000000	48637684	50445860	35637684	
4	2014-08-22	If I Stay	11000000	50474843	78356170	39474843	

5 rows × 35 columns

In [71]:
```
for index, row in movie_details_df2.iterrows():
    if row['genres']:
        for genre in row['genres']:
            movie_details_df2.loc[index,genre] = 1

movie_details_df2.head()
```

Out[71]:

	release_date	movie_x	production_budget	domestic_gross	worldwide_gross	domestic_gross_profit	v
0	2016-01-22	The Boy	10000000	35819556	68220952	25819556	
1	2017-10-27	Jigsaw	10000000	38052832	102445196	28052832	

	release_date	movie_x	production_budget	domestic_gross	worldwide_gross	domestic_gross_profit	v
2	2018-03-16	Love, Simon	10000000	40826341	65520633	30826341	
3	2014-02-14	About Last Night	13000000	48637684	50445860	35637684	
4	2014-08-22	If I Stay	11000000	50474843	78356170	39474843	

5 rows × 70 columns

In [72]:
```
movie_details_df_modified = movie_details_df.drop(columns = 'genres')
```

In [73]:
```
for col in movie_details_df_modified:
    print(f'Viewing values in col: {col}')
    print(f'Top 5 values:\n{movie_details_df_modified[col].value_counts()}')
```

```
Viewing values in col: release_date
Top 5 values:
2015-08-07    9
2018-04-13    6
2013-01-18    6
2017-01-20    5
2015-09-11    4
             ..
2018-10-05    1
2018-07-04    1
2010-10-15    1
2016-12-09    1
2018-05-11    1
Name: release_date, Length: 130, dtype: int64
Viewing values in col: movie_x
Top 5 values:
The Gift                           9
Truth or Dare                      6
Mama                               6
Split                              5
Neighbors                          4
                                 ..
The Best Exotic Marigold Hotel     1
Sausage Party                      1
The Conjuring                      1
Safety Not Guaranteed              1
Justin Bieber: Never Say Never     1
Name: movie_x, Length: 134, dtype: int64
Viewing values in col: production_budget
Top 5 values:
5000000     36
20000000    12
10000000    11
1000000     11
3500000     10
15000000     8
12000000     6
2000000      5
13000000     5
1500000      4
3000000      4
```

```
7000000      4
18000000     4
25000000     4
4000000      3
250000       3
17000000     2
1800000      2
900000       2
19000000     2
2500000      2
2300000      2
58000000     2
11000000     2
750000       2
35000000     2
225000       2
200000       2
6500000      1
175000       1
80000000     1
75000000     1
28000000     1
14500000     1
32500000     1
21000000     1
30000        1
6000000      1
36000000     1
8000000      1
6900000      1
100000       1
50000        1
74000000     1
8500000      1
120000       1
9000000      1
76000000     1
916000       1
30000000     1
3200000      1
29000000     1
Name: production_budget, dtype: int64
Viewing values in col: domestic_gross
Top 5 values:
43787265     9
41411015     6
71628180     6
138141585    5
65206105     4
             ..
36006065     1
67790117     1
144822       1
215288866    1
169607287    1
Name: domestic_gross, Length: 134, dtype: int64
Viewing values in col: worldwide_gross
Top 5 values:
58978477     9
95127344     6
148095566    6
278964806    5
98677816     4
             ..
304604712    1
```

```
4422318      1
7712436      1
144822       1
16376066     1
Name: worldwide_gross, Length: 134, dtype: int64
Viewing values in col: domestic_gross_profit
Top 5 values:
38787265     9
37911015     6
56628180     6
133141585    5
91035005     4
            ..
29744051     1
35637684     1
45086903     1
50401464     1
38729221     1
Name: domestic_gross_profit, Length: 134, dtype: int64
Viewing values in col: worldwide_gross_profit
Top 5 values:
53978477     9
133095566    6
91627344     6
273964806    5
93677816     4
            ..
1927779      1
743025593    1
8617402      1
258625468    1
217276928    1
Name: worldwide_gross_profit, Length: 134, dtype: int64
Viewing values in col: domestic_roi_percent
Top 5 values:
5.0     45
6.0     25
7.0     18
9.0     17
8.0     12
13.0    10
12.0     8
28.0     6
16.0     6
4.0      5
10.0     5
11.0     4
14.0     3
18.0     3
19.0     2
21.0     2
23.0     2
36.0     1
17.0     1
33.0     1
35.0     1
53.0     1
Name: domestic_roi_percent, dtype: int64
Viewing values in col: worldwide_roi_percent
Top 5 values:
12.0     25
9.0      19
10.0     13
13.0     11
6.0      11
```

```
7.0      10
8.0       8
15.0      8
20.0      8
5.0       7
27.0      7
56.0      5
11.0      5
25.0      4
16.0      4
30.0      4
18.0      2
14.0      2
17.0      2
26.0      2
21.0      2
29.0      2
24.0      2
43.0      2
41.0      1
51.0      1
4.0       1
64.0      1
59.0      1
39.0      1
19.0      1
67.0      1
42.0      1
28.0      1
22.0      1
40.0      1
102.0     1
Name: worldwide_roi_percent, dtype: int64
Viewing values in col: year
Top 5 values:
2015    28
2018    25
2016    21
2012    21
2017    18
2014    18
2013    15
2011    14
2010    10
2019     8
Name: year, dtype: int64
Viewing values in col: primary_title
Top 5 values:
The Gift                          9
Truth or Dare                     6
Mama                              6
Split                             5
Neighbors                         4
                                 ..
The Best Exotic Marigold Hotel    1
Sausage Party                     1
The Conjuring                     1
Safety Not Guaranteed             1
Justin Bieber: Never Say Never    1
Name: primary_title, Length: 134, dtype: int64
Viewing values in col: original_title
Top 5 values:
The Gift                              8
Mama                                  6
Split                                 5
```

```
The Visit                              4
Neighbors                              4
                                      ..
The VVitch: A New-England Folktale     1
The Conjuring                          1
Safety Not Guaranteed                  1
Your Sister's Sister                   1
Justin Bieber: Never Say Never         1
Name: original_title, Length: 140, dtype: int64
Viewing values in col: start_year
Top 5 values:
2016.0    29
2018.0    21
2011.0    20
2015.0    20
2012.0    19
2010.0    18
2014.0    16
2013.0    15
2017.0    15
2019.0     5
Name: start_year, dtype: int64
Viewing values in col: runtime_minutes
Top 5 values:
90.0     10
92.0      9
86.0      7
110.0     7
100.0     7
         ..
27.0      1
114.0     1
137.0     1
77.0      1
123.0     1
Name: runtime_minutes, Length: 67, dtype: int64
Viewing values in col: movie_y
Top 5 values:
The Gift                               8
Mama                                   6
Split                                  5
The Visit                              4
Neighbors                              4
                                      ..
The VVitch: A New-England Folktale     1
The Conjuring                          1
Safety Not Guaranteed                  1
Your Sister's Sister                   1
Justin Bieber: Never Say Never         1
Name: movie_y, Length: 140, dtype: int64
```

In [74]:
```
cols = list(movie_details_df2.columns)
```

In [75]:
```
genre_cols = cols[11:]
```

In [76]:
```
genre_count = {}
for col in genre_cols:
    count = np.sum(movie_details_df2[col] == 1).sum()
    genre_count[col] = count
```

In [77]:
```
genre_count
```

```
{'original_title': 0,
```

Out[77]:
```
'start_year': 0,
'runtime_minutes': 0,
'genres': 0,
'movie_y': 0,
'Drama': 0,
'Horror': 0,
'Fantasy': 0,
'Sport': 0,
'Family': 0,
'Biography': 0,
'Crime': 0,
'Mystery': 0,
'Documentary': 0,
'Music': 0,
'History': 0,
'Sci-Fi': 0,
'Adventure': 0,
'Thriller': 0,
'Western': 0,
'Romance': 0,
'Action': 0,
'Animation': 0,
'Comedy': 0,
'[': 117,
"'": 117,
'H': 40,
'o': 101,
'r': 106,
',': 102,
' ': 102,
'M': 29,
'y': 79,
's': 37,
't': 53,
'e': 93,
'T': 30,
'h': 40,
'i': 67,
'l': 33,
']': 117,
'C': 44,
'm': 86,
'd': 44,
'D': 65,
'a': 76,
'R': 23,
'n': 49,
'c': 49,
'F': 19,
'u': 20,
'A': 17,
'v': 9,
'S': 12,
'-': 11,
'B': 11,
'g': 11,
'p': 11,
'W': 1}
```

Is the overseas market worth it?

In [78]:
```
df3 = pd.merge(imdb_title_basics_df,tn_movie_budgets_df)
df3
```

Out[78]:

	primary_title	original_title	start_year	runtime_minutes	genres	movie	rel
0	Foodfight!	Foodfight!	2012	91.0	Action,Animation,Comedy	Foodfight!	2
1	The Overnight	The Overnight	2010	88.0	NaN	The Overnight	2
2	The Overnight	The Overnight	2015	79.0	Comedy,Mystery	The Overnight	2
3	On the Road	On the Road	2012	124.0	Adventure,Drama,Romance	On the Road	2
4	On the Road	On the Road	2011	90.0	Drama	On the Road	2
...	...	...	...	...	...	...	
2219	Chloe	Chloe	2017	6.0	Documentary,Family	Chloe	2
2220	Happy Death Day 2U	Happy Death Day 2U	2019	100.0	Drama,Horror,Mystery	Happy Death Day 2U	2
2221	Fahrenheit 11/9	Fahrenheit 11/9	2018	128.0	Documentary	Fahrenheit 11/9	2
2222	The Witch	The Witch	2018	NaN	Horror	The Witch	2
2223	Unplanned	Unplanned	2019	106.0	Biography,Drama	Unplanned	2

2224 rows × 14 columns

In [79]:
```
df3
```

Out[79]:

	primary_title	original_title	start_year	runtime_minutes	genres	movie	rel
0	Foodfight!	Foodfight!	2012	91.0	Action,Animation,Comedy	Foodfight!	2
1	The Overnight	The Overnight	2010	88.0	NaN	The Overnight	2
2	The Overnight	The Overnight	2015	79.0	Comedy,Mystery	The Overnight	2
3	On the Road	On the Road	2012	124.0	Adventure,Drama,Romance	On the Road	2
4	On the Road	On the Road	2011	90.0	Drama	On the Road	2
...	...	...	...	...	...	...	
2219	Chloe	Chloe	2017	6.0	Documentary,Family	Chloe	2
2220	Happy Death Day 2U	Happy Death Day 2U	2019	100.0	Drama,Horror,Mystery	Happy Death Day 2U	2
2221	Fahrenheit 11/9	Fahrenheit 11/9	2018	128.0	Documentary	Fahrenheit 11/9	2

	primary_title	original_title	start_year	runtime_minutes	genres	movie	rel
2222	The Witch	The Witch	2018	NaN	Horror	The Witch	2
2223	Unplanned	Unplanned	2019	106.0	Biography,Drama	Unplanned	2

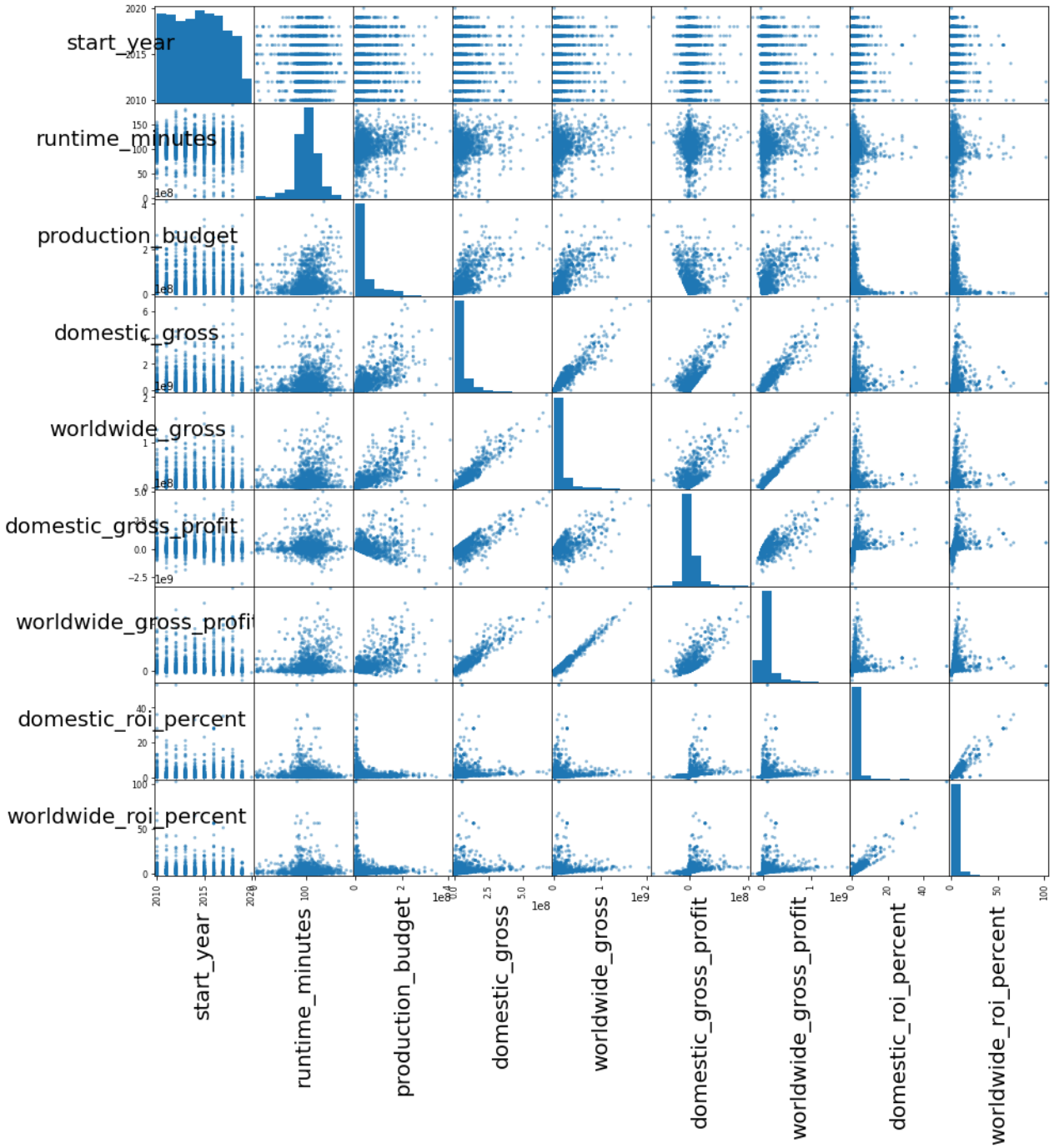
2224 rows × 14 columns

In [80]:
```
df4 = df3.sort_values('worldwide_roi_percent')
```

This scatter plot was created to check and see if there where any notable correlations. There are some strong correlations between Domestic ROI Percent and Worldwide ROI Percent, which is worth looking into. Another interesting correlation to look at further is the bell-shaped relation between Runtime Minutes and Domestic/Worldwide ROI Percent. Since there is a time restraint on this project we will only be looking at the Domestic ROI Percent verses the Runtime.

In [81]:
```
scatter_matrix = pd.plotting.scatter_matrix(
    df4,
    figsize  = [15, 15]#,
#     marker   = ".",
#     s        = 0.2,
#     diagonal = "kde"
)

for ax in scatter_matrix.ravel():
    ax.set_xlabel(ax.get_xlabel(), fontsize = 20, rotation = 90)
    ax.set_ylabel(ax.get_ylabel(), fontsize = 20, rotation = 0)
```



To drill down further on the relationship between Runtime Minutes and Domestic ROI Percent. The minutes were grouped by year and rounded to the 0 places since typically movies are only described in whole minutes

In [82]:
```
df6 = df4.tail(150)
df6 = df6.groupby(['start_year']).mean()
df6['runtime_minutes']=round(df6['runtime_minutes'],0)
df6
```

Out[82]:

start_year	runtime_minutes	production_budget	domestic_gross	worldwide_gross	domestic_gross_profit
2010	94.0	4.888203e+06	5.034187e+07	9.461879e+07	4.545366e+07
2011	97.0	4.773761e+06	3.608828e+07	7.179414e+07	3.131452e+07

start_year	runtime_minutes	production_budget	domestic_gross	worldwide_gross	domestic_gross_profit
2012	94.0	1.066429e+07	7.778303e+07	1.575851e+08	6.711874e+07
2013	83.0	1.077857e+07	7.183005e+07	1.770090e+08	6.105148e+07
2014	102.0	8.207510e+06	5.458619e+07	1.356763e+08	4.637868e+07
2015	97.0	1.432500e+07	8.356906e+07	2.085943e+08	6.924406e+07
2016	94.0	1.539783e+07	1.173078e+08	2.777675e+08	1.019099e+08
2017	101.0	2.000667e+07	1.234879e+08	2.942634e+08	1.034813e+08
2018	103.0	1.066944e+07	7.549884e+07	1.723262e+08	6.482940e+07
2019	107.0	1.140000e+07	8.386914e+07	1.395950e+08	7.246914e+07

The above table shows that movie times for the top 150 movies range from 99 minutes to 105 minutes. Meaning that all the 150 movies are withing 6 minutes of the others. Seems like a small range for such a diverse group of elements. Below er look to see if this trend is reflective of all the movies in our data set.

In [83]:
```
df5 = df4.groupby(['start_year']).mean()
df5['runtime_minutes']=round(df5['runtime_minutes'],0)

df5
```

Out[83]:

start_year	runtime_minutes	production_budget	domestic_gross	worldwide_gross	domestic_gross_profit
2010	102.0	3.575972e+07	4.447180e+07	1.038875e+08	8.712087e+06
2011	102.0	3.833374e+07	4.471338e+07	1.111158e+08	6.379640e+06
2012	101.0	4.110708e+07	4.991368e+07	1.233575e+08	8.806601e+06
2013	104.0	4.076439e+07	4.711385e+07	1.151156e+08	6.349458e+06
2014	103.0	3.783038e+07	4.894848e+07	1.221308e+08	1.111810e+07
2015	100.0	3.377672e+07	4.161903e+07	1.013513e+08	7.842309e+06
2016	103.0	4.042475e+07	5.418752e+07	1.306297e+08	1.376277e+07
2017	99.0	4.956400e+07	5.771735e+07	1.523517e+08	8.153353e+06
2018	105.0	4.550791e+07	6.632132e+07	1.633607e+08	2.081341e+07
2019	105.0	4.627909e+07	4.302174e+07	1.016307e+08	-3.257351e+06
2020	NaN	3.500000e+07	1.681600e+04	7.865320e+05	-3.498318e+07

When we look at the avrages for all the data from the past 10 years we see an even smaller range of 6 minutes. Futher below you will see graphes that dive deeper in to these findings.

Below begins to look at the correlation between the Domestic ROI Percent and Worldwide ROI Percent. Since looking at the at the top 150 based on Domestic ROI Percent and comparing it to all the above was useful the following tables will continue that type of comparison.

In [84]:
```
#Round percent 3 decimal points
df6 = df4.tail(150).groupby(['start_year']).mean()
df6['domestic_roi_percent']=round(df6['domestic_roi_percent'],3)
df6['worldwide_roi_percent']=round(df6['worldwide_roi_percent'],3)
df6
```

Out[84]:

start_year	runtime_minutes	production_budget	domestic_gross	worldwide_gross	domestic_gross_profit
2010	93.928571	4.888203e+06	5.034187e+07	9.461879e+07	4.545366e+07
2011	96.923077	4.773761e+06	3.608828e+07	7.179414e+07	3.131452e+07
2012	93.750000	1.066429e+07	7.778303e+07	1.575851e+08	6.711874e+07
2013	82.833333	1.077857e+07	7.183005e+07	1.770090e+08	6.105148e+07
2014	102.357143	8.207510e+06	5.458619e+07	1.356763e+08	4.637868e+07
2015	96.923077	1.432500e+07	8.356906e+07	2.085943e+08	6.924406e+07
2016	94.136364	1.539783e+07	1.173078e+08	2.777675e+08	1.019099e+08
2017	100.846154	2.000667e+07	1.234879e+08	2.942634e+08	1.034813e+08
2018	102.875000	1.066944e+07	7.549884e+07	1.723262e+08	6.482940e+07
2019	107.200000	1.140000e+07	8.386914e+07	1.395950e+08	7.246914e+07

In [85]:
```
#Round percent 3 decimal points
df5 = df4.groupby(['start_year']).mean()
df5['domestic_roi_percent']=round(df5['domestic_roi_percent'],3)
df5['worldwide_roi_percent']=round(df5['worldwide_roi_percent'],3)
df5
```

Out[85]:

start_year	runtime_minutes	production_budget	domestic_gross	worldwide_gross	domestic_gross_profit
2010	101.528926	3.575972e+07	4.447180e+07	1.038875e+08	8.712087e+06
2011	101.950617	3.833374e+07	4.471338e+07	1.111158e+08	6.379640e+06
2012	101.479452	4.110708e+07	4.991368e+07	1.233575e+08	8.806601e+06
2013	104.379913	4.076439e+07	4.711385e+07	1.151156e+08	6.349458e+06
2014	102.791165	3.783038e+07	4.894848e+07	1.221308e+08	1.111810e+07
2015	100.165323	3.377672e+07	4.161903e+07	1.013513e+08	7.842309e+06
2016	103.296137	4.042475e+07	5.418752e+07	1.306297e+08	1.376277e+07
2017	99.243094	4.956400e+07	5.771735e+07	1.523517e+08	8.153353e+06
2018	104.530488	4.550791e+07	6.632132e+07	1.633607e+08	2.081341e+07

start_year	runtime_minutes	production_budget	domestic_gross	worldwide_gross	domestic_gross_profit
2019	105.294118	4.627909e+07	4.302174e+07	1.016307e+08	-3.257351e+06
2020	NaN	3.500000e+07	1.681600e+04	7.865320e+05	-3.498318e+07

In [86]:
```
df5['year']= ['2010', '2011', '2012', '2013', '2014', '2015', '2016', '2017', '2018', '
df5
```

Out[86]:

start_year	runtime_minutes	production_budget	domestic_gross	worldwide_gross	domestic_gross_profit
2010	101.528926	3.575972e+07	4.447180e+07	1.038875e+08	8.712087e+06
2011	101.950617	3.833374e+07	4.471338e+07	1.111158e+08	6.379640e+06
2012	101.479452	4.110708e+07	4.991368e+07	1.233575e+08	8.806601e+06
2013	104.379913	4.076439e+07	4.711385e+07	1.151156e+08	6.349458e+06
2014	102.791165	3.783038e+07	4.894848e+07	1.221308e+08	1.111810e+07
2015	100.165323	3.377672e+07	4.161903e+07	1.013513e+08	7.842309e+06
2016	103.296137	4.042475e+07	5.418752e+07	1.306297e+08	1.376277e+07
2017	99.243094	4.956400e+07	5.771735e+07	1.523517e+08	8.153353e+06
2018	104.530488	4.550791e+07	6.632132e+07	1.633607e+08	2.081341e+07
2019	105.294118	4.627909e+07	4.302174e+07	1.016307e+08	-3.257351e+06
2020	NaN	3.500000e+07	1.681600e+04	7.865320e+05	-3.498318e+07

In [87]:
```
df6['year']= ['2010', '2011', '2012', '2013', '2014', '2015', '2016', '2017', '2018', '
df6
```

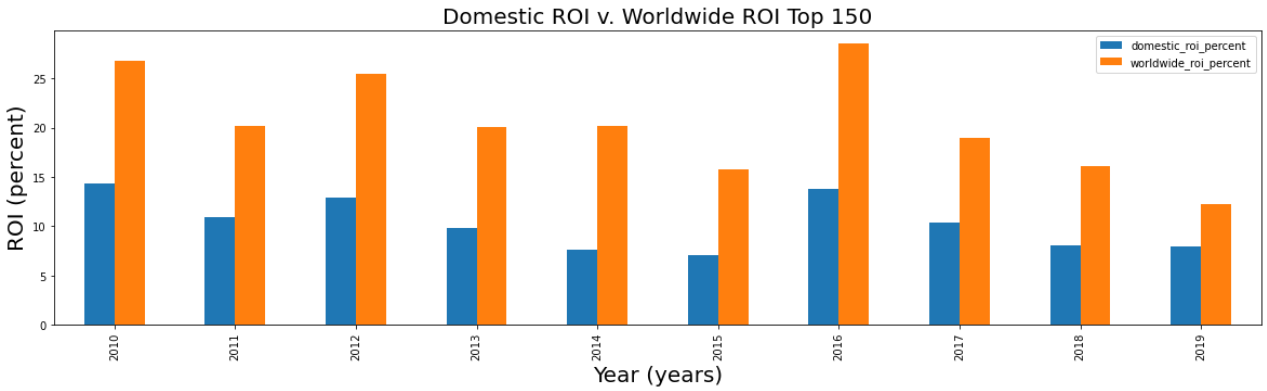
Out[87]:

start_year	runtime_minutes	production_budget	domestic_gross	worldwide_gross	domestic_gross_profit
2010	93.928571	4.888203e+06	5.034187e+07	9.461879e+07	4.545366e+07
2011	96.923077	4.773761e+06	3.608828e+07	7.179414e+07	3.131452e+07
2012	93.750000	1.066429e+07	7.778303e+07	1.575851e+08	6.711874e+07
2013	82.833333	1.077857e+07	7.183005e+07	1.770090e+08	6.105148e+07
2014	102.357143	8.207510e+06	5.458619e+07	1.356763e+08	4.637868e+07
2015	96.923077	1.432500e+07	8.356906e+07	2.085943e+08	6.924406e+07
2016	94.136364	1.539783e+07	1.173078e+08	2.777675e+08	1.019099e+08
2017	100.846154	2.000667e+07	1.234879e+08	2.942634e+08	1.034813e+08

start_year	runtime_minutes	production_budget	domestic_gross	worldwide_gross	domestic_gross_profit
2018	102.875000	1.066944e+07	7.549884e+07	1.723262e+08	6.482940e+07
2019	107.200000	1.140000e+07	8.386914e+07	1.395950e+08	7.246914e+07

In [88]:
```
df6.reset_index().plot(
    x="year", y= ["domestic_roi_percent", "worldwide_roi_percent"], kind="bar", figsize
)
plt.title("Domestic ROI v. Worldwide ROI Top 150",  fontsize = 20)
plt.xlabel("Year (years)", fontsize = 20)
plt.ylabel("ROI (percent)", fontsize = 20,)
```

Out[88]:
```
Text(0, 0.5, 'ROI (percent)')
```

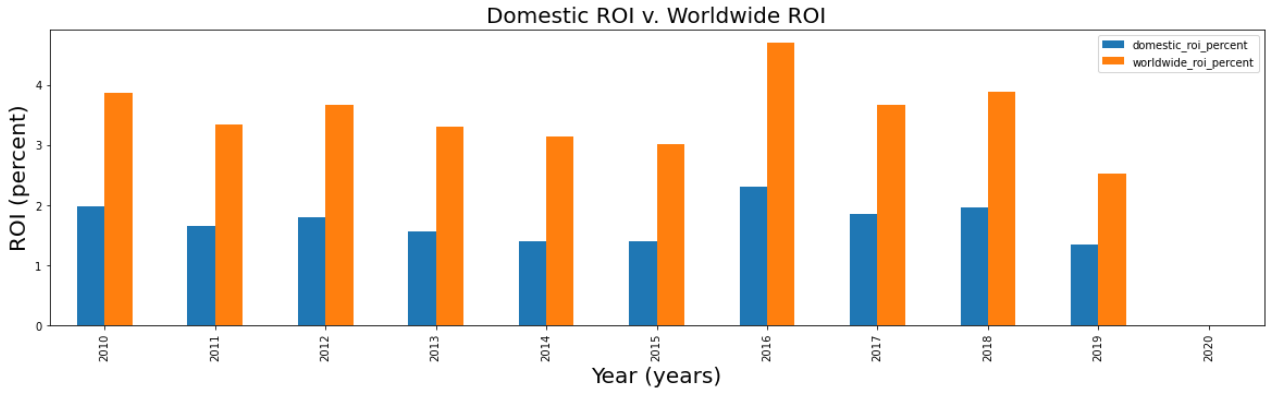


Above you can see that for the top 150 movies, for their respective year, had almost a 1:2 ratio for Domestic ROI Percent to Worldwide ROI Percent. letting up know that if a movie is doing well domestically it would do just as well worldwide. Increasing profits with a lot less effort that doing a whole another movie.

Below you can see that that that trend is also represented with in all the data. Which shows that even if you do put out a poor performing movie you should still be able to double you ROI by making it available worldwide.

In [89]:
```
#df5[[df5['year'], "domestic_roi_percent", "worldwide_roi_percent"]].plot(x=df5['year']
df5.reset_index().plot(
    x="year", y= ["domestic_roi_percent", "worldwide_roi_percent"], kind="bar", figsize
)
plt.title("Domestic ROI v. Worldwide ROI",  fontsize = 20)
plt.xlabel("Year (years)", fontsize = 20)
plt.ylabel("ROI (percent)", fontsize = 20)
```

Out[89]:
```
Text(0, 0.5, 'ROI (percent)')
```



How long should the movies be?

In [90]:
```
x = df5['runtime_minutes'].mean()
print('The average movie time is ' + 'x')
```

```
The average movie time is x
```

In [91]:
```
df5
```

Out[91]:

start_year	runtime_minutes	production_budget	domestic_gross	worldwide_gross	domestic_gross_profit
2010	101.528926	3.575972e+07	4.447180e+07	1.038875e+08	8.712087e+06
2011	101.950617	3.833374e+07	4.471338e+07	1.111158e+08	6.379640e+06
2012	101.479452	4.110708e+07	4.991368e+07	1.233575e+08	8.806601e+06
2013	104.379913	4.076439e+07	4.711385e+07	1.151156e+08	6.349458e+06
2014	102.791165	3.783038e+07	4.894848e+07	1.221308e+08	1.111810e+07
2015	100.165323	3.377672e+07	4.161903e+07	1.013513e+08	7.842309e+06
2016	103.296137	4.042475e+07	5.418752e+07	1.306297e+08	1.376277e+07
2017	99.243094	4.956400e+07	5.771735e+07	1.523517e+08	8.153353e+06
2018	104.530488	4.550791e+07	6.632132e+07	1.633607e+08	2.081341e+07
2019	105.294118	4.627909e+07	4.302174e+07	1.016307e+08	-3.257351e+06
2020	NaN	3.500000e+07	1.681600e+04	7.865320e+05	-3.498318e+07

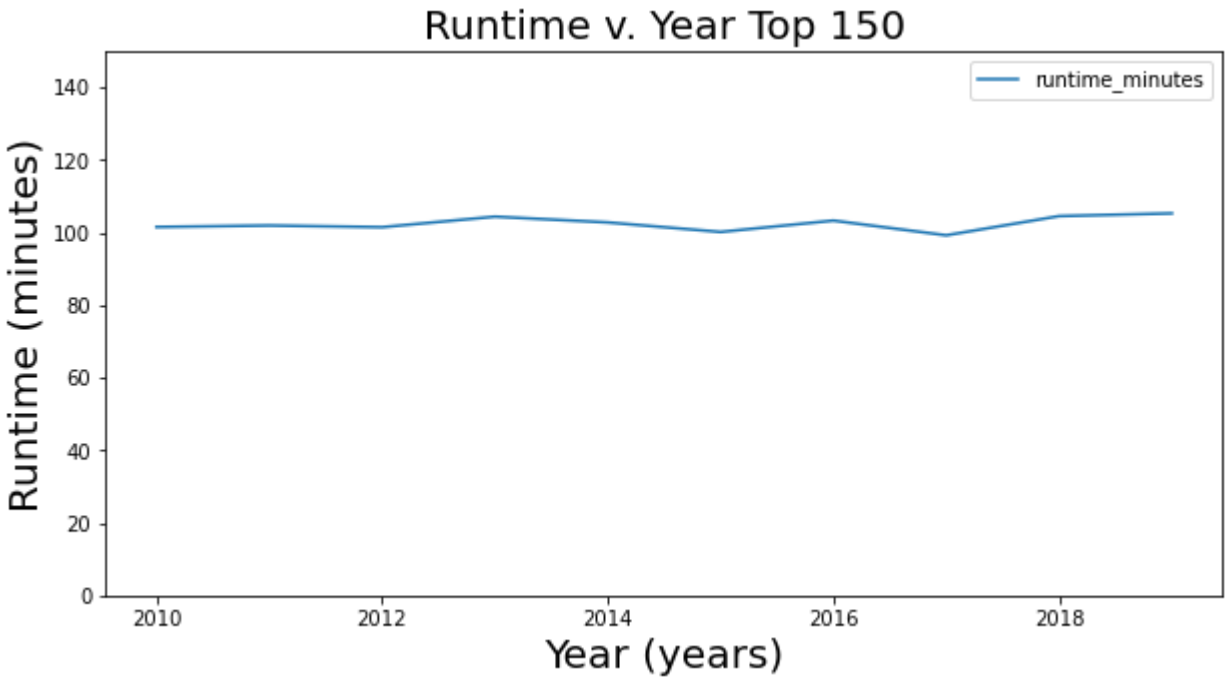
The two graph below show an intresting runtime to Domestic ROI Percent trend. With a Avrage of 102 it is clear that movie maker have a "rule" about lenght of movies, that it should be as close to 102 minutes as possible.

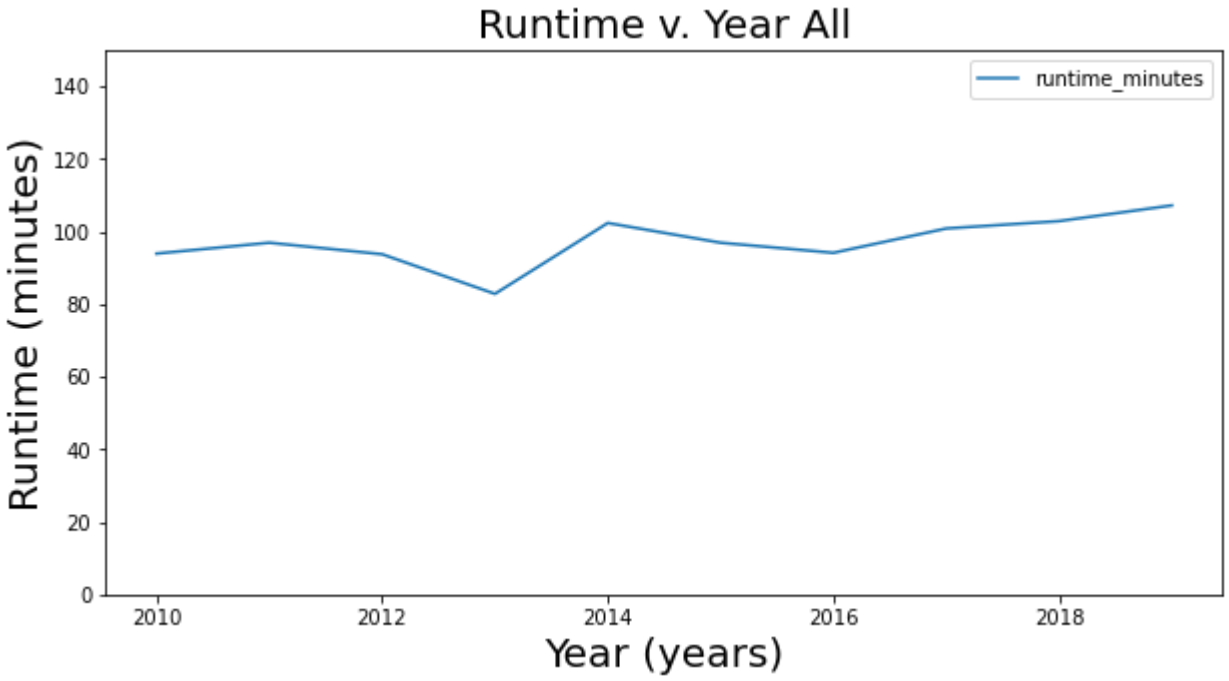
In [92]:
```
ax1 = df5.plot.line(x='year',
                    y='runtime_minutes', figsize  = [10, 5])
plt.title("Runtime v. Year Top 150",  fontsize = 20)
plt.xlabel("Year (years)", fontsize = 20)
plt.ylabel("Runtime (minutes)", fontsize = 20)
plt.ylim([0, 150])
```

```
ax2 = df6.plot.line(x='year',
                    y='runtime_minutes', figsize  = [10, 5])
plt.title("Runtime v. Year All",  fontsize = 20)
plt.xlabel("Year (years)", fontsize = 20)
plt.ylabel("Runtime (minutes)", fontsize = 20)

plt.ylim([0, 150])
```

Out[92]: (0.0, 150.0)





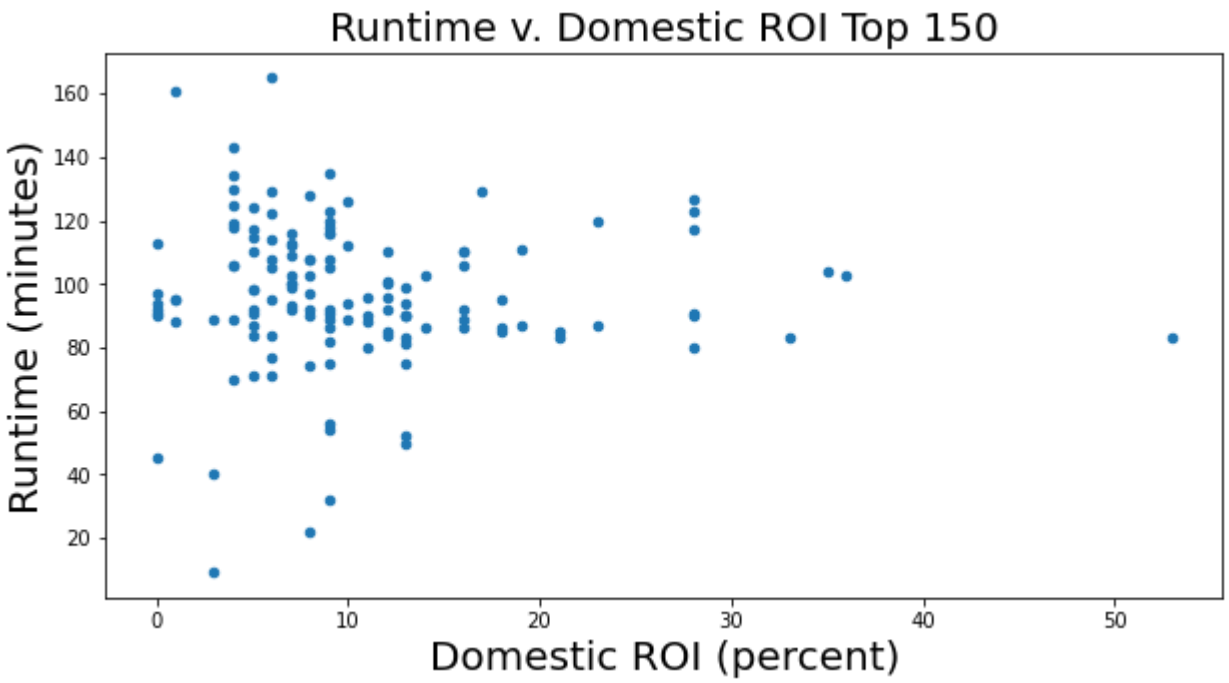
The Below scatter plot reinforces the "rule" around time. The movies that stayed around the 102 had ROI that ranged from 0 to above 50. whereas the further you got away from 102 minutes the lower the ROI was. It is safest to stick to a time of 102 and hope for a hit than to think you are the expectation to the rule because even when looking at all the data there were no excessively long or short movies that had a return grater that 10%.

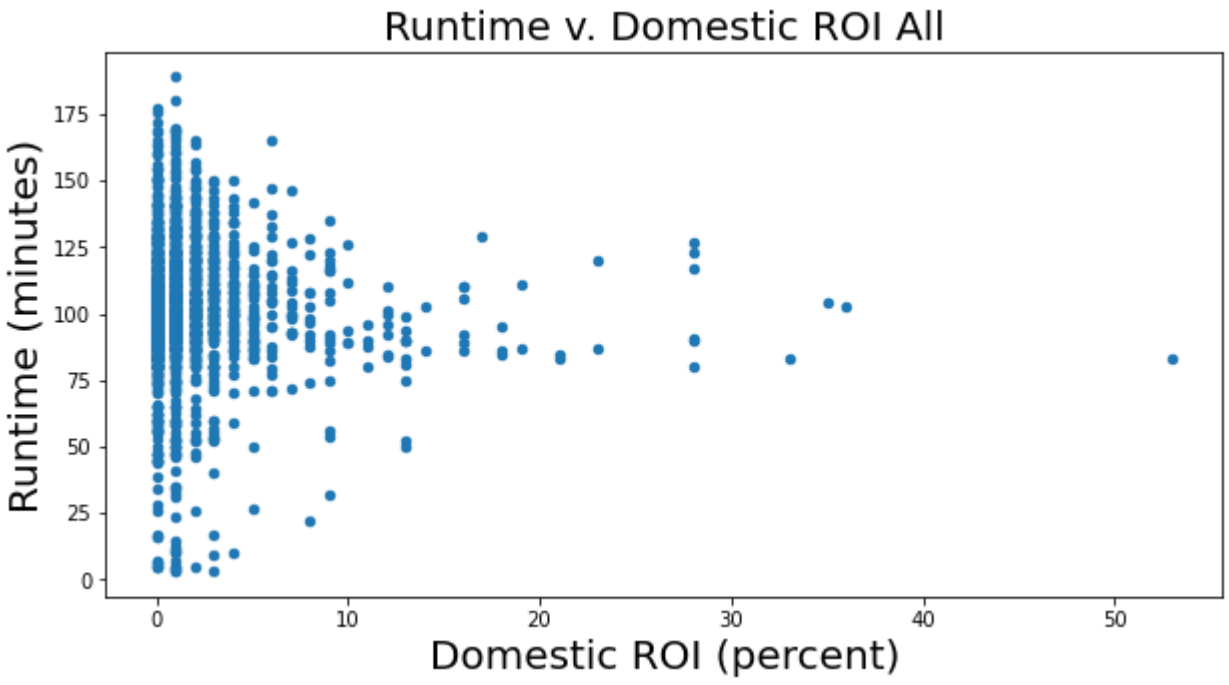
In [93]:
```
df44 = df4.tail(150)
```

In [94]:
```
ax3 = df44.plot.scatter(x='domestic_roi_percent',
                        y='runtime_minutes', figsize  = [10, 5])
plt.title("Runtime v. Domestic ROI Top 150",  fontsize = 20)
plt.xlabel("Domestic ROI (percent)", fontsize = 20)
plt.ylabel("Runtime (minutes)", fontsize = 20)

ax3 = df4.plot.scatter(x='domestic_roi_percent',
                        y='runtime_minutes', figsize  = [10, 5])
plt.title("Runtime v. Domestic ROI All",  fontsize = 20)
plt.xlabel("Domestic ROI (percent)", fontsize = 20)
plt.ylabel("Runtime (minutes)", fontsize = 20)
```

Out[94]:
```
Text(0, 0.5, 'Runtime (minutes)')
```





In [95]:
```
df4['runtime_minutes'].mean()
```

Out[95]:
```
102.21029626032055
```

In [96]:
```
x=0
y=0
for i in list(df4['runtime_minutes'] < 102):
    if i == True:
        x=x+1
    else:
        y=y+1

print(x,y)
```

```
995 1229
```

Concludsion/Recommendations

Production Budget should be $120,283,930

Based on the expected Gross and Profit it is suggested that Microsoft plans to spend the above amount on a new movie.(Thought this number does not include start up cost.)

Movie Length Should be 102 Minutes

Movies whose length is above and below have a lower ROI with no outliers with in thoughs groups.

Create a Movie That Can Easily be Translated and Marketed in Several

Countries When looking at the data from the top 150 and all data the ROI on averages doubles when the make for the movie was expanded world wide

Future Work

The following will help to improve findings and find more insightful findings: Scape and use various movie websites and compare information.

- Sign up for a benchmark to get data directly from other studios.
- Research changes to the movie industry since as a result of COVID(ie When will movies attendance be back to normal? How has movies going straight to digital effected the marker?)
- Investigate weather how many non-domestic movies where not translate and see how that effect ROI.
- Explore what makes extremely profitable movies so successful.
- Drill down on which genres in other countries are most popular.