CAPSTONE

Company: Computing Vision

The company has seen other big movie companies create original films, and they want to get in on the fun. They have decided to create a new movie studio to create original films.

Computing Vision does not have much experience in original film creation.

- Goal: Provide three business recommendations derived from analyzing several datasets of current box office movies.

> Notebook flow: This jupyter notebook goes as follows, stating each business recomendation with the appropriate code, and conclusions.

Data Cleaning and merging dataframes

In this notebook, we'll work with `movie_basics` and `movie_ratings` tables from `'im.db'`. As well as `'tn.movie_budget.csv'`.

Before we can get going, we'll need to import the relevant packages and conncet to the database.

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.ticker as mtick
%matplotlib inline
import seaborn as sns
import statsmodels.api as sm
from statsmodels.formula.api import ols
from sklearn.linear_model import LinearRegression
from scipy.stats import pearsonr
from scipy import stats
import sqlite3
conn = sqlite3.connect('im.db')
```

In this notebook we'll focus on the `movie_basics` and `movie_ratings`.

Compared to the Individual Tables:

`movie_basics` Table:

In [2]:
```
q = """
SELECT *
```

```
FROM movie_basics
"""
pd.read_sql(q, conn).head(5)
```

Out[2]:

	movie_id	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
3	tt0069204	Sabse Bada Sukh	Sabse Bada Sukh	2018	NaN	Comedy,Drama
4	tt0100275	The Wandering Soap Opera	La Telenovela Errante	2017	80.0	Comedy,Drama,Fantasy

`movie_ratings` Table:

In [3]:

```
q = """
SELECT *
FROM movie_ratings
"""
pd.read_sql(q, conn).head(5)
```

Out[3]:

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

Displaying movie_basics Along with movie_ratings

Since we need to generate a table that includes details about `movie_basics` and `movie_ratings`, we would need to take data from multiple tables in a single statement using a concise way to join the tables, the `USING` clause. Which in this case is `movie_id`. Again, this only works if the column is **identically named** for both tables.

Then we assign the result of the querry to a variable names `df`, which is a dataframe.

In [4]:

```
q = """
SELECT
    movie_id,
    primary_title,
    genres,
    averagerating,
```

```
    numvotes
FROM movie_basics
JOIN movie_ratings
    USING (movie_id)
"""
df = pd.read_sql(q, conn)
```

To get a concise summary of the dataframe, you can use `.info()` :

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 73856 entries, 0 to 73855
Data columns (total 5 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   movie_id       73856 non-null  object 
 1   primary_title  73856 non-null  object 
 2   genres         73052 non-null  object 
 3   averagerating  73856 non-null  float64
 4   numvotes       73856 non-null  int64  
dtypes: float64(1), int64(1), object(3)
memory usage: 2.8+ MB
```

Nan (Not a Number):

When working with datasets, it is common to have missing or `NaN (Not a Number)` values. In order to understand the extent of missing data in a dataset, you can use the `.isna()` method to identify where the `NaN` values are located. Taking the `.sum()` of the `.isna(`) method will return the total number of NaN values in the dataset broken down by column.

In [6]:
```
df.isna().sum()
```

Out[6]:
```
movie_id           0
primary_title      0
genres           804
averagerating      0
numvotes           0
dtype: int64
```

Since the `'genre'` is a categorical data and the missing values accounts for only 1 % of our data we decided to drop the rows that contained null values using the built-in function `.dropna()`. Since we are creating a new df, a new name will be given to it as `df_nonull_genres`.

In [7]:
```
df_nonull_genres = df.dropna(axis = 0)
```

Lets print the shape of our new df. Next, display the total number of `NaN` values in the dataset broken down by column.

In [8]:
```
df_nonull_genres.shape
```

Out[8]:
```
(73052, 5)
```

In [9]:
```
df_nonull_genres.isna().sum()
```

Out[9]:
```
movie_id         0
primary_title    0
genres           0
averagerating    0
numvotes         0
dtype: int64
```

Since, we dont have any other missing data to deal with, lets focus on finding duplicate movie titles using `.dulpicated()` and `.value_counts()`. Lets display the total amount of dulpicated rows. Subsequently, break it down by frequency for each movie title.

In [10]:
```
df_nonull_genres['primary_title'].duplicated().value_counts()
```

Out[10]:
```
False    69248
True      3804
Name: primary_title, dtype: int64
```

In [11]:
```
df_nonull_genres[df_nonull_genres['primary_title'].duplicated()]['primary_title'].valu
```

Out[11]:
```
The Return           10
Broken                9
Lucky                 8
Homecoming            8
Together              8
                     ..
Checkmate             1
Won't Back Down       1
Political Animals     1
Dead Awake            1
Drømmeland            1
Name: primary_title, Length: 2705, dtype: int64
```

The df shows 69248 `non-duplicated` values. A common practice would be to handle them properly but for scope purposes of our study, which means we are time-limited, lets just keep them in mind. Then, lets take a look on the second dataframe.

The second `df` can be found on `'tn.movie_budget.csv'`. Now, let's get started by reading in the data and storing it the DataFrame `movie_budget`. Afterwárds, lets preview the data.

In [12]:
```
movie_budget = pd.read_csv('tn.movie_budgets.csv')
movie_budget.head()
```

Out[12]:

	id	release_date	movie	production_budget	domestic_gross	worldwide_gross
0	1	18-Dec-09	Avatar	$425,000,000	$760,507,625	$2,776,345,279
1	2	20-May-11	Pirates of the Caribbean: On Stranger Tides	$410,600,000	$241,063,875	$1,045,663,875
2	3	7-Jun-19	Dark Phoenix	$350,000,000	$42,762,350	$149,762,350
3	4	1-May-15	Avengers: Age of Ultron	$330,600,000	$459,005,868	$1,403,013,963
4	5	15-Dec-17	Star Wars Ep. VIII: The Last Jedi	$317,000,000	$620,181,382	$1,316,721,747

`production_budget`, `domestic_gross`, and `worldwide_gross` are strings, so we will remove the commas and dollar signs

In [13]:
```
movie_budget['production_budget'] = movie_budget['production_budget'].str.replace('$',
movie_budget['domestic_gross'] = movie_budget['domestic_gross'].str.replace('$', '').s
movie_budget['worldwide_gross'] = movie_budget['worldwide_gross'].str.replace('$', '')
```

```
C:\Users\raguilarsoriano\AppData\Local\Temp\ipykernel_18552\3322509581.py:1: FutureWa
rning: The default value of regex will change from True to False in a future version.
In addition, single character regular expressions will *not* be treated as literal st
rings when regex=True.
  movie_budget['production_budget'] = movie_budget['production_budget'].str.replace
('$', '').str.replace(',', '')
C:\Users\raguilarsoriano\AppData\Local\Temp\ipykernel_18552\3322509581.py:2: FutureWa
rning: The default value of regex will change from True to False in a future version.
In addition, single character regular expressions will *not* be treated as literal st
rings when regex=True.
  movie_budget['domestic_gross'] = movie_budget['domestic_gross'].str.replace('$',
'').str.replace(',', '')
C:\Users\raguilarsoriano\AppData\Local\Temp\ipykernel_18552\3322509581.py:3: FutureWa
rning: The default value of regex will change from True to False in a future version.
In addition, single character regular expressions will *not* be treated as literal st
rings when regex=True.
  movie_budget['worldwide_gross'] = movie_budget['worldwide_gross'].str.replace('$',
'').str.replace(',', '')
```

`production_budget`, `domestic_gross`, and `worldwide_gross` are still strings, so we will change them to integers to be able to perform calculations with those columns

In [14]:
```
movie_budget['production_budget'] = movie_budget['production_budget'].astype('int64')
movie_budget['domestic_gross'] = movie_budget['domestic_gross'].astype('int64')
movie_budget['worldwide_gross'] = movie_budget['worldwide_gross'].astype('int64')
```

A concise summary will be provided using `.info()`.

In [15]:
```
movie_budget.info()
```

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

The above summary contains also the amount of null values brake down by columns. Since we don´t have missing data to deal with, we can move into merging `movie_budget` df and `df_nonull_genres` df.

For doing so, lets print a short preview for both Dataframes. Focus on the column names.

In [16]:
```
df_nonull_genres.head(2)
```

Out[16]:

	movie_id	primary_title	genres	averagerating	numvotes
0	tt0063540	Sunghursh	Action,Crime,Drama	7.0	77
1	tt0066787	One Day Before the Rainy Season	Biography,Drama	7.2	43

`movie_budget` df:

In [17]:
```
movie_budget.head(2)
```

Out[17]:

	id	release_date	movie	production_budget	domestic_gross	worldwide_gross
0	1	18-Dec-09	Avatar	425000000	760507625	2776345279
1	2	20-May-11	Pirates of the Caribbean: On Stranger Tides	410600000	241063875	1045663875

As you can see, the column which displays the movie title is different in both Dataframes. `df_nonull_genres` uses `'primary_title'` while `movie_budget` uses `'movie'`.

We need to set them equal to use the column name as a key for merging both Dataframes into one. Will set both columns names as `'movie'`. In this case, `df_nonull_genres` is the one selected to change its column name.

In [18]:
```
df_nonull_genres.rename(columns={'primary_title' : 'movie'}, inplace = True)
```

```
C:\Users\raguilarsoriano\AppData\Local\Temp\ipykernel_18552\3490391291.py:1: SettingW
ithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  df_nonull_genres.rename(columns={'primary_title' : 'movie'}, inplace = True)
```

Lets view the column labels of the DataFrame `df_nonull_genres`.

In [19]:
```
df_nonull_genres.columns
```

Out[19]:
```
Index(['movie_id', 'movie', 'genres', 'averagerating', 'numvotes'], dtype='object')
```

As you see, now the column name was changed succesfully.

Nan (Not a Number):

The following line of code is performing a merge operation between `df_nonull_genre` s and `movie_budget` .

The merge is performed based on a common column called `'movie'` , specified by the `on` parameter.

The type of merge used is specified by the how parameter, which in this case is set to `'left'` , meaning that all the rows from the `df_nonull_genres dataframe` will be kept and any matching rows from the `movie_budget` dataframe will be included. Any non-matching rows from the movie_budget dataframe will have null values in the resulting dataframe.

Finally, a new column called `'im_and_movie_budget'` is added to the resulting merged dataframe, indicating whether a row is present in both dataframes (i.e., 'both'), only in the left dataframe (i.e., 'left_only'), or only in the right dataframe (i.e., 'right_only'). This is specified by the `indicator` parameter.

The resulting dataframe is assigned to the variable `im_movie_budget` . Subsequently, we use `value_counts()` to return a new Series object with the count of unique values of the new column called `'im_and_movie_budget'` .

In [20]:
```
im_movie_budget = pd.merge(df_nonull_genres ,movie_budget, on='movie', how='left', inc
im_movie_budget['im_and_movie_budget'].value_counts()
```

Out[20]:
```
left_only     70307
both           2867
right_only        0
Name: im_and_movie_budget, dtype: int64
```

A sample was taken from Dataframe `im_movie_budget` by selecting rows that has a string value equal to `both` on column `'im_and_movie_budget'` . That sample name is `cleaned_df` .

In [21]:
```
cleaned_df = im_movie_budget[im_movie_budget['im_and_movie_budget'] == 'both']
cleaned_df.head(4)
```

Out[21]:

	movie_id	movie	genres	averagerating	numvotes	id	release_date	pro
16	tt0249516	Foodfight!	Action,Animation,Comedy	1.9	8248	26.0	31-Dec-12	
36	tt0337692	On the Road	Adventure,Drama,Romance	6.1	37886	17.0	22-Mar-13	
42	tt0359950	The Secret Life of Walter Mitty	Adventure,Comedy,Drama	7.3	275300	37.0	25-Dec-13	
46	tt0365907	A Walk Among the Tombstones	Action,Crime,Drama	6.5	105116	67.0	19-Sep-14	

Lets get a concise summary of the dataframe using `.info()` :

In [22]:
```
cleaned_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2867 entries, 16 to 73164
Data columns (total 11 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   movie_id             2867 non-null   object
 1   movie                2867 non-null   object
 2   genres               2867 non-null   object
 3   averagerating        2867 non-null   float64
 4   numvotes             2867 non-null   int64
 5   id                   2867 non-null   float64
 6   release_date         2867 non-null   object
 7   production_budget    2867 non-null   float64
 8   domestic_gross       2867 non-null   float64
 9   worldwide_gross      2867 non-null   float64
 10  im_and_movie_budget  2867 non-null   category
dtypes: category(1), float64(5), int64(1), object(4)
memory usage: 249.3+ KB
```

floats data types it's possible for columns with integer data types to be converted to floating point data types. This can happen if one of the dataframes has null or missing values in the column being merged.

To avoid this type conversion, you can either fill in the missing values before merging the dataframes or use the astype method to convert the column back to an integer after the merge.

In [23]:
```
cleaned_df['production_budget'] = cleaned_df['production_budget'].astype('int64')
cleaned_df['domestic_gross'] = cleaned_df['domestic_gross'].astype('int64')
cleaned_df['worldwide_gross'] = cleaned_df['worldwide_gross'].astype('int64')
```

```
C:\Users\raguilarsoriano\AppData\Local\Temp\ipykernel_18552\2674449468.py:1: SettingW
ithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  cleaned_df['production_budget'] = cleaned_df['production_budget'].astype('int64')
C:\Users\raguilarsoriano\AppData\Local\Temp\ipykernel_18552\2674449468.py:2: SettingW
ithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  cleaned_df['domestic_gross'] = cleaned_df['domestic_gross'].astype('int64')
C:\Users\raguilarsoriano\AppData\Local\Temp\ipykernel_18552\2674449468.py:3: SettingW
ithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  cleaned_df['worldwide_gross'] = cleaned_df['worldwide_gross'].astype('int64')
```

Added net revenue by movie

In [24]:
```
cleaned_df['worlwide_net_revenue'] = cleaned_df['worldwide_gross'] - cleaned_df['produ
cleaned_df[['production_budget', 'worldwide_gross', 'worlwide_net_revenue']]
```

```
C:\Users\raguilarsoriano\AppData\Local\Temp\ipykernel_18552\1701568743.py:1: SettingW
ithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  cleaned_df['worlwide_net_revenue'] = cleaned_df['worldwide_gross'] - cleaned_df['pr
oduction_budget']
```

Out[24]:

	production_budget	worldwide_gross	worlwide_net_revenue
16	45000000	73706	-44926294
36	25000000	9313302	-15686698
42	91000000	187861183	96861183
46	28000000	62108587	34108587
47	215000000	1648854864	1433854864
...	...	...	...
72971	47000000	116773317	69773317
73096	30000000	106030660	76030660
73111	109000000	87683966	-21316034
73151	700000	1110511	410511
73164	95000000	165720921	70720921

2867 rows × 3 columns

At this point, we are all set to begin with the Eploratory Data Analisis.

Business Case 1: Genre based on profits

In the first business case, we need to know based on the `genres` and how much money locally and globaly they generated. So in this case we are analizing which `genres` produce more money.

Identifying genres.

For the beginning, we need to work with the table `movie_basics` of the `im.db` data base,specifically with the columns `primary_title`, `genres` and the `tn.movie_budgets.csv` file with the columns `domestic_gross` and `worldwide_gross`.

We are creating a new dataFrame only using these columns.

In [25]:
```
#Creating a table with Movie Title, genres and profits
case1_df = cleaned_df.loc[:,['movie','genres','worldwide_gross']]
print(case1_df)
```

```
                              movie                  genres  \
16                       Foodfight!  Action,Animation,Comedy
36                      On the Road  Adventure,Drama,Romance
42     The Secret Life of Walter Mitty   Adventure,Comedy,Drama
46         A Walk Among the Tombstones       Action,Crime,Drama
47                   Jurassic World  Action,Adventure,Sci-Fi
...                             ...                     ...
72971                         Earth             Documentary
73096                       Sisters            Action,Drama
73111                           Ali                   Drama
73151                      Columbus                  Comedy
73164                   Unstoppable             Documentary

       worldwide_gross
16               73706
36             9313302
42           187861183
46            62108587
47          1648854864
...                ...
72971        116773317
73096        106030660
73111         87683966
73151          1110511
73164        165720921

[2867 rows x 3 columns]
```

Separating genres and counting them

Checking the DataFrame above we notice that some movie have more than 1 genre, so we need to count each one for knowing the total revenue of each genre.

So we need to serpate eache genera and put it into a new column.

In [26]:
```
#Dividing all of the genres
genres_cols = cleaned_df['genres'].str.split(',', expand=True)
genres_cols.columns = ['genre1','genre2','genre3']
counts1 = genres_cols['genre1'].value_counts()
counts2 = genres_cols['genre2'].value_counts()
counts3 = genres_cols['genre3'].value_counts()
total_counts = counts1.add(counts2, fill_value=0).add(counts3, fill_value=0)
total_counts
```

Out[26]:

```
Action          630.0
Adventure       448.0
Animation       130.0
Biography       195.0
Comedy          758.0
Crime           362.0
Documentary     204.0
Drama          1491.0
Family          144.0
Fantasy         175.0
History          71.0
Horror          360.0
Music            72.0
Musical          22.0
Mystery         223.0
News              3.0
Romance         326.0
Sci-Fi          204.0
Sport            62.0
Thriller        509.0
War              39.0
Western          16.0
dtype: float64
```

Now, we combine the `cleaned_df` with the new three columns for each genre into a new dataFrame called `case1_vs_df`

In [27]:

```
#Creating a new table with number of gender,
case1_vs_df = pd.merge(cleaned_df[['movie','domestic_gross','worldwide_gross','product
case1_vs_df
```

Out[27]:

	movie	domestic_gross	worldwide_gross	production_budget	genre1	genre2
16	Foodfight!	0	73706	45000000	Action	Animation
36	On the Road	720828	9313302	25000000	Adventure	Drama
42	The Secret Life of Walter Mitty	58236838	187861183	91000000	Adventure	Comedy
46	A Walk Among the Tombstones	26017685	62108587	28000000	Action	Crime
47	Jurassic World	652270625	1648854864	215000000	Action	Adventure
...	...	...	...	...	...	...
72971	Earth	32011576	116773317	47000000	Documentary	None
73096	Sisters	87044645	106030660	30000000	Action	Drama
73111	Ali	58183966	87683966	109000000	Drama	None
73151	Columbus	1017107	1110511	700000	Comedy	None
73164	Unstoppable	81562942	165720921	95000000	Documentary	None

2867 rows × 7 columns

To work with the `domestic_gross` and `worldwide_gross` is necessary to take only the numerical part, so we need to delete the "$" and "," simbols

In [28]:

```
#Delete the $ and ,
case1_vs_df['domestic_gross'] = case1_vs_df['domestic_gross'].replace({'\$':'',',':''
case1_vs_df['worldwide_gross'] = case1_vs_df['worldwide_gross'].replace({'\$':'',',':
case1_vs_df['production_budget'] = case1_vs_df['production_budget'].replace({'\$':'',
case1_vs_df
```

Out[28]:

	movie	domestic_gross	worldwide_gross	production_budget	genre1	genre2
16	Foodfight!	0	73706	45000000	Action	Animation
36	On the Road	720828	9313302	25000000	Adventure	Drama
42	The Secret Life of Walter Mitty	58236838	187861183	91000000	Adventure	Comedy
46	A Walk Among the Tombstones	26017685	62108587	28000000	Action	Crime
47	Jurassic World	652270625	1648854864	215000000	Action	Adventure
...	...	...	...	...	...	...
72971	Earth	32011576	116773317	47000000	Documentary	None
73096	Sisters	87044645	106030660	30000000	Action	Drama
73111	Ali	58183966	87683966	109000000	Drama	None
73151	Columbus	1017107	1110511	700000	Comedy	None
73164	Unstoppable	81562942	165720921	95000000	Documentary	None

2867 rows × 7 columns

Having all the totals

We need to convert first the data type of `domestic_gross`, `worldwide_gross` and `production_budget` columns. Then, we need to add the domestic rows values every time that a movie have some genre to have the total of all the profits and all the production budget of every genre.

In [29]:

```
case1_vs_df['domestic_gross'] = case1_vs_df['domestic_gross'].astype(float)
case1_vs_df['worldwide_gross'] = case1_vs_df['worldwide_gross'].astype(float)
case1_vs_df['production_budget'] = case1_vs_df['production_budget'].astype(float)

totals = {}

for index, row in case1_vs_df.iterrows():
    genres = [row['genre1'], row['genre2'], row['genre3']]
    for genre in genres:
        if genre not in totals:
            totals[genre] = {'domestic_gross': 0, 'worldwide_gross': 0, 'production_bu
        totals[genre]['domestic_gross'] += row['domestic_gross']
        totals[genre]['worldwide_gross'] += row['worldwide_gross']
        totals[genre]['production_budget'] += row['production_budget']

new_df = pd.DataFrame(totals).T.reset_index().rename(columns={'index': 'genre'})
new_df['genre'] = new_df['genre'].astype('string')
new_df['genre'].fillna('Other',inplace=True)
```

```
new_df
```

Out[29]:

	genre	domestic_gross	worldwide_gross	production_budget
0	Action	4.468698e+10	1.199860e+11	4.077893e+10
1	Animation	1.505947e+10	4.129402e+10	1.116779e+10
2	Comedy	3.608269e+10	8.045373e+10	2.533618e+10
3	Adventure	4.796407e+10	1.343977e+11	4.090776e+10
4	Drama	4.315093e+10	9.143296e+10	3.513053e+10
5	Romance	9.593389e+09	2.055086e+10	6.616127e+09
6	Crime	1.106037e+10	2.411083e+10	9.946593e+09
7	Sci-Fi	1.860459e+10	5.009867e+10	1.431784e+10
8	Other	6.341734e+10	1.343814e+11	4.873818e+10
9	Family	9.971529e+09	2.342681e+10	7.568903e+09
10	Thriller	1.672351e+10	4.117760e+10	1.401471e+10
11	Horror	1.018044e+10	2.310061e+10	6.522787e+09
12	Mystery	7.057109e+09	1.572666e+10	4.846065e+09
13	Biography	6.765873e+09	1.410100e+10	4.964048e+09
14	History	2.374610e+09	4.997602e+09	2.214910e+09
15	War	7.672892e+08	1.660432e+09	9.010000e+08
16	Fantasy	1.363563e+10	3.763870e+10	1.210108e+10
17	Sport	2.456170e+09	4.823089e+09	1.477125e+09
18	Music	2.142755e+09	4.525251e+09	1.086920e+09
19	Documentary	5.923072e+09	1.175290e+10	4.612943e+09
20	Western	6.032472e+08	1.226694e+09	7.448000e+08
21	Musical	1.723901e+09	3.900989e+09	8.582000e+08
22	News	2.821122e+07	1.100462e+08	4.980000e+07

Finally, we want to select only the top 5 genres with more profits because we have a lot of genres, but we only need the most convenient genres and show them in a bar graph.

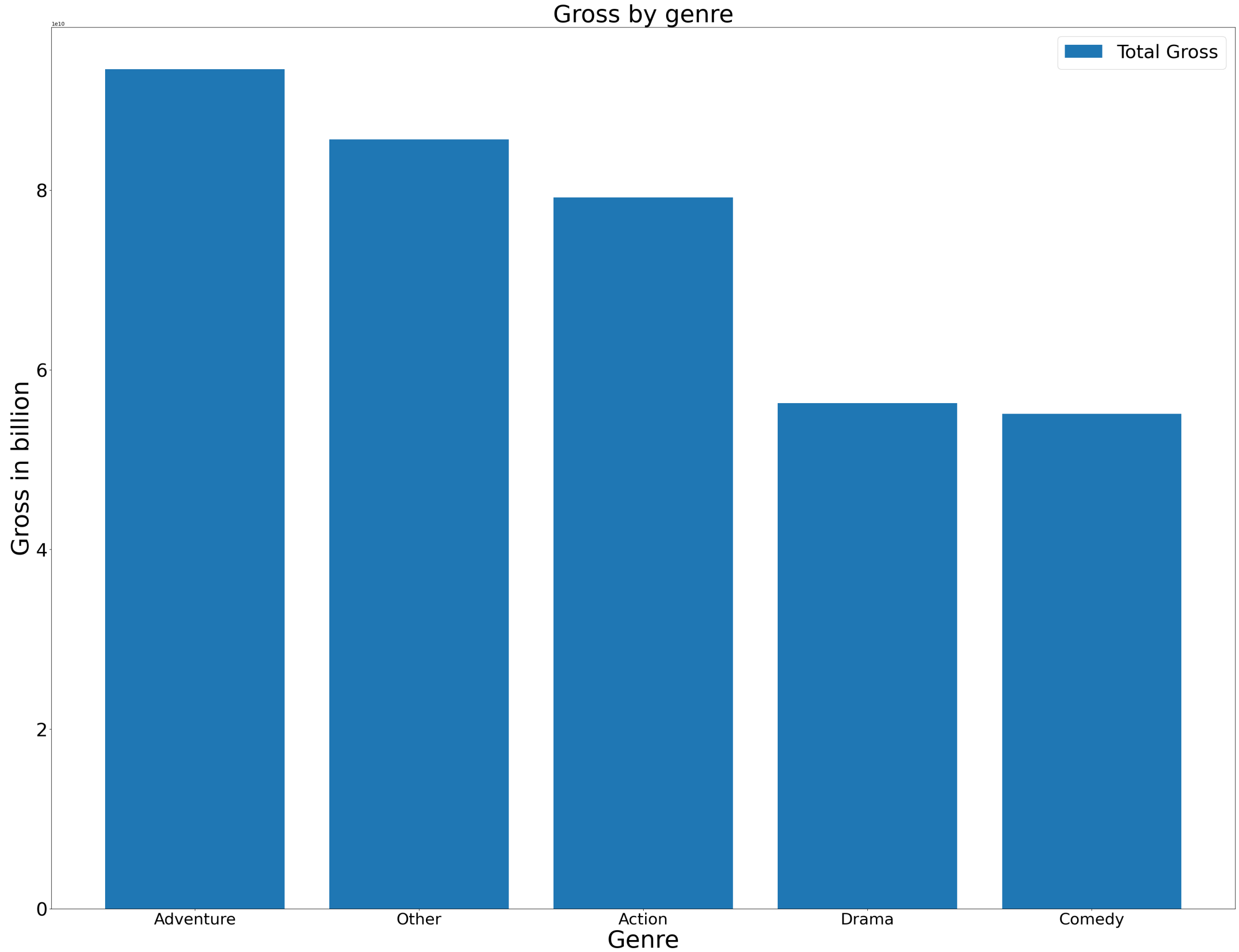
In [30]:
```
final_gross = new_df['worldwide_gross'] - new_df['production_budget']
new_df['total_gross'] = final_gross
new_df = new_df.sort_values(by='total_gross', ascending=False)
top_5 = new_df.head(5)
top_5
```

Out[30]:

	genre	domestic_gross	worldwide_gross	production_budget	total_gross
3	Adventure	4.796407e+10	1.343977e+11	4.090776e+10	9.348997e+10
8	Other	6.341734e+10	1.343814e+11	4.873818e+10	8.564318e+10
0	Action	4.468698e+10	1.199860e+11	4.077893e+10	7.920712e+10
4	Drama	4.315093e+10	9.143296e+10	3.513053e+10	5.630243e+10
2	Comedy	3.608269e+10	8.045373e+10	2.533618e+10	5.511755e+10

In [31]:

```
#Creating a bar graph for the case1_vs_df table

#Defining the columns of the x and y axis
new_df = new_df[new_df.genre !='Other']
x = top_5['genre']
y = top_5['total_gross']

#Defining the name of each bar
plt.figure(figsize=(40,30))
plt.xticks(fontsize=30)
plt.yticks(fontsize=35)
plt.bar(x,y,label='Total Gross')

#Defining name of the labels and title of the bar graph
plt.title('Gross by genre', fontsize=45)
plt.xlabel('Genre', fontsize=45)
plt.ylabel('Gross in billion', fontsize=45)

plt.legend(fontsize=35)
plt.show()
```



Business Case 1: Conclusion

Checking the bar graph noticing that the genres that produce more money are `Adventure`, `Action` and `Drama`. These three genres generate more money that the other ones.

The conclusion of this business case is that if you want to create a movie that could produce a lot of money are Adventure, Action and Drama genres.

Business Case 2: Difference in budget by genres

In this business case, we want to know if higher the `production_budget` the higher the `worlwide_net_revenue`.

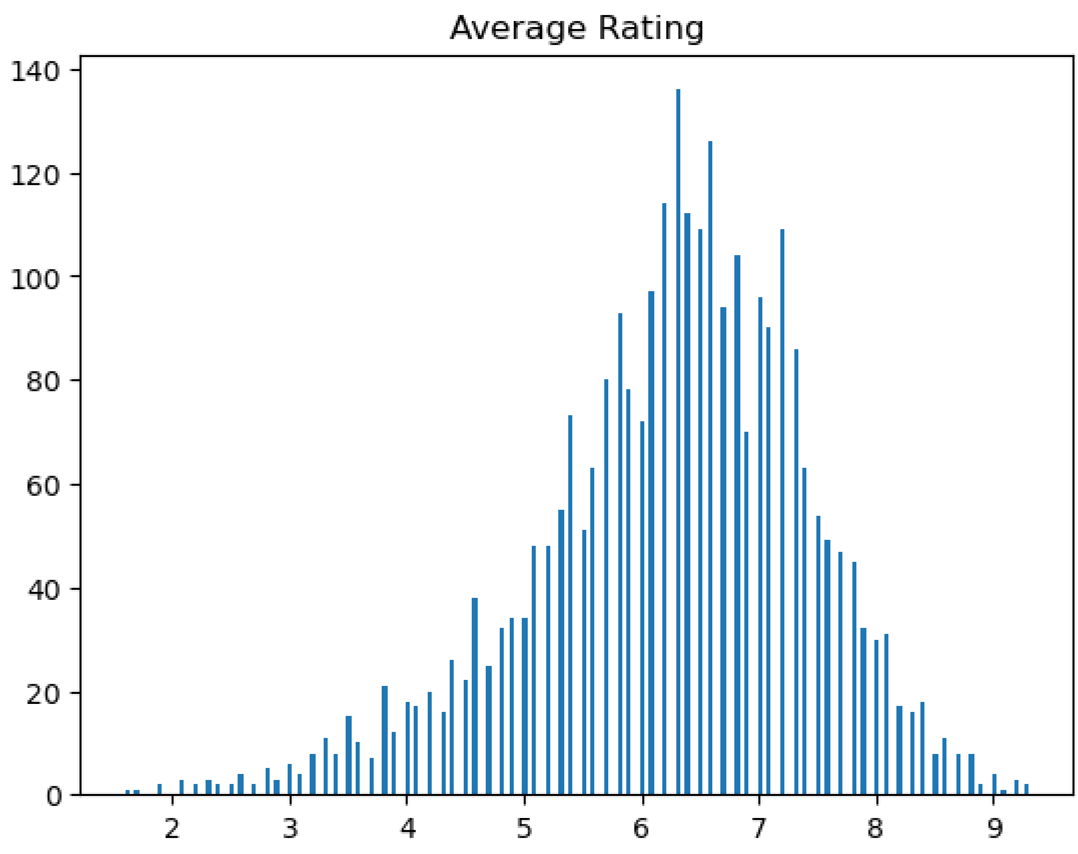
To test this hypothesis, we would need to conduct regression analysis, to examine the strength and direction of the relationship between these two variables. First for the whole population and then across genres.

Befor jumping into the complex part, lets define a function named `stats` to define the `mean`, `median`, and `standard deviation` of a given column of the dataframe to better understand the distribution of some variables/columns.

In [32]:
```
def stats(column):
    """
    This function takes the name of a column as inputs, and returns its mean, median a
    deviation.

    Args:
        column_name (str): The name of the column to get their stats.

    Returns:
        pandas.DataFrame: A new DataFrame with the specified column sorted in descendi
    """
    mean = cleaned_df[column].mean()
    median = cleaned_df[column].median()
    std = cleaned_df[column].std()
    return print(f' Mean:{mean}\n Median:{median}\n Std:{std}\n')
```

We will produce a histogram for `averagerating`

In [33]:
```
fig, ax = plt.subplots()
plt.hist(cleaned_df["averagerating"], bins=200)
ax.set_title("Average Rating");
```



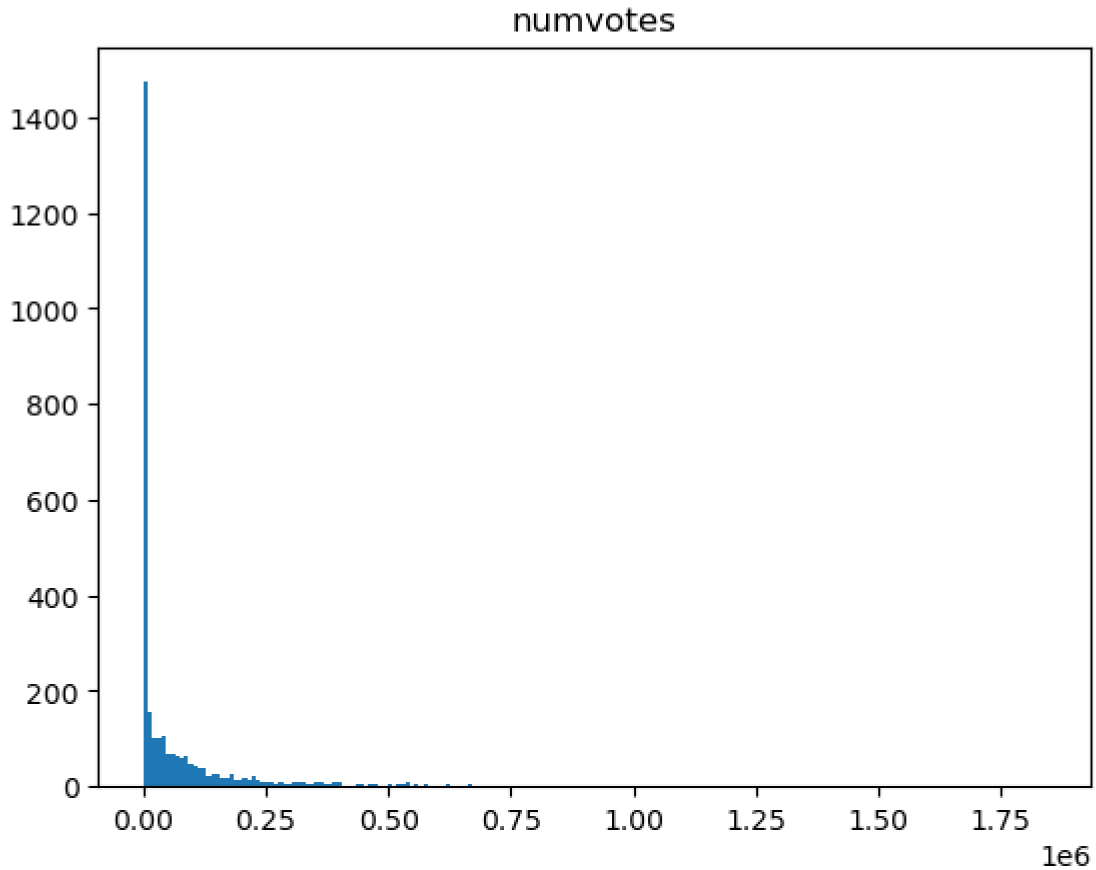
We will produce the `stats` function for `averagerating`

In [34]:
```
stats('averagerating')
```

```
 Mean:6.249110568538549
 Median:6.4
 Std:1.1859530886691918
```

We will produce a histogram for `numvotes`

In [35]:
```
fig, ax = plt.subplots()
plt.hist(cleaned_df["numvotes"], bins=200)
ax.set_title("numvotes");
```



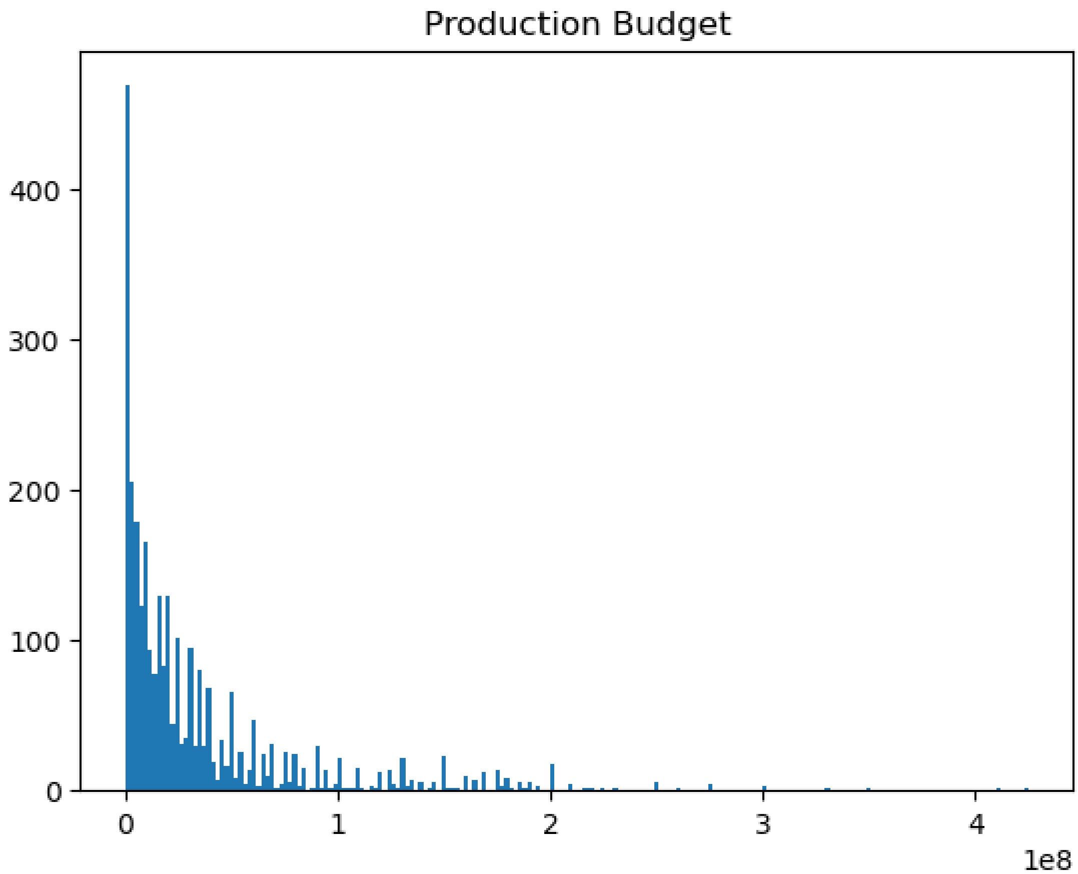
We will produce the `stats` function for `numvotes`

In [36]:
```
stats('numvotes')
```

```
Mean:66465.27659574468
Median:7999.0
Std:134449.32806695305
```

We will produce a histogram for `production_budget`

In [37]:
```
fig, ax = plt.subplots()
plt.hist(cleaned_df["production_budget"], bins=200)
ax.set_title("Production Budget");
```



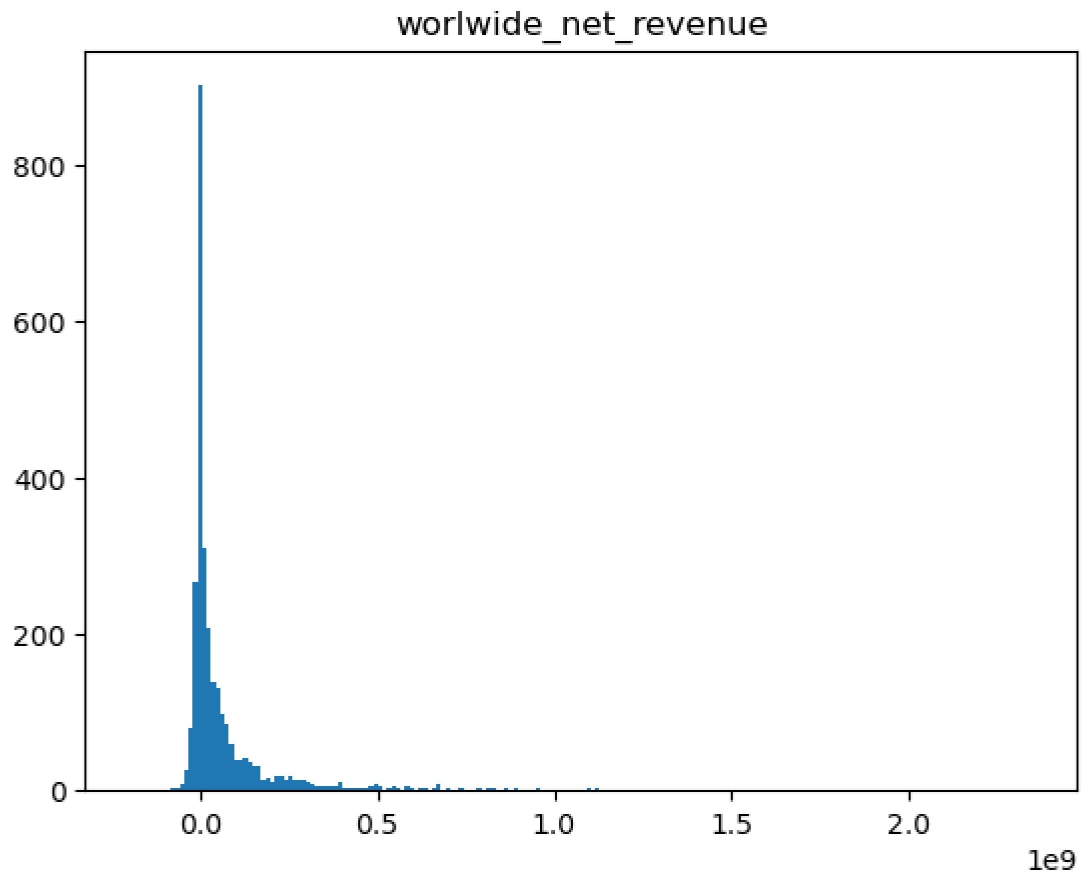
We will produce the `stats` function for `production_budget`

In [38]:
```
stats('production_budget')
```

```
Mean:34287085.19637252
Median:17000000.0
Std:47672750.903373405
```

We will produce a histogram for `worlwide_net_revenue`

In [39]:
```
fig, ax = plt.subplots()
plt.hist(cleaned_df["worlwide_net_revenue"], bins=200)
ax.set_title("worlwide_net_revenue");
```



We will produce the `stats` function for `worlwide_net_revenue`

In [40]:
```
stats('worlwide_net_revenue')
```

```
Mean:68593345.74363446
Median:9596747.0
Std:169464167.71571285
```

Let's create a new column named `'by_genre'` that performs an individual `COUNT` for each genre. Since some films have multi-categorical genres, we need to break them down into single categories, resulting in the film appearing as many times as the number of genres it has. This will allow us to group the data later by genre using `.GROUPBY()`.

In [41]:
```
# split values by commas from column genres and explode the column
b2_explode_cleaned_df = cleaned_df.assign(by_genre = cleaned_df['genres'].str.split(',
b2_explode_cleaned_df.head(6)
```

Out[41]:

	movie_id	movie	genres	averagerating	numvotes	id	release_date	proc
16	tt0249516	Foodfight!	Action,Animation,Comedy	1.9	8248	26.0	31-Dec-12	
16	tt0249516	Foodfight!	Action,Animation,Comedy	1.9	8248	26.0	31-Dec-12	
16	tt0249516	Foodfight!	Action,Animation,Comedy	1.9	8248	26.0	31-Dec-12	
36	tt0337692	On the Road	Adventure,Drama,Romance	6.1	37886	17.0	22-Mar-13	
36	tt0337692	On the Road	Adventure,Drama,Romance	6.1	37886	17.0	22-Mar-13	
36	tt0337692	On the Road	Adventure,Drama,Romance	6.1	37886	17.0	22-Mar-13	

In [42]:

```
# Count appearances of each genre.
b2_explode_cleaned_df['by_genre'].value_counts()
```

Out[42]:

```
Drama          1491
Comedy          758
Action          630
Thriller        509
Adventure       448
Crime           362
Horror          360
Romance         326
Mystery         223
Sci-Fi          204
Documentary     204
Biography       195
Fantasy         175
Family          144
Animation       130
Music            72
History          71
Sport            62
War              39
Musical          22
Western          16
News              3
Name: by_genre, dtype: int64
```

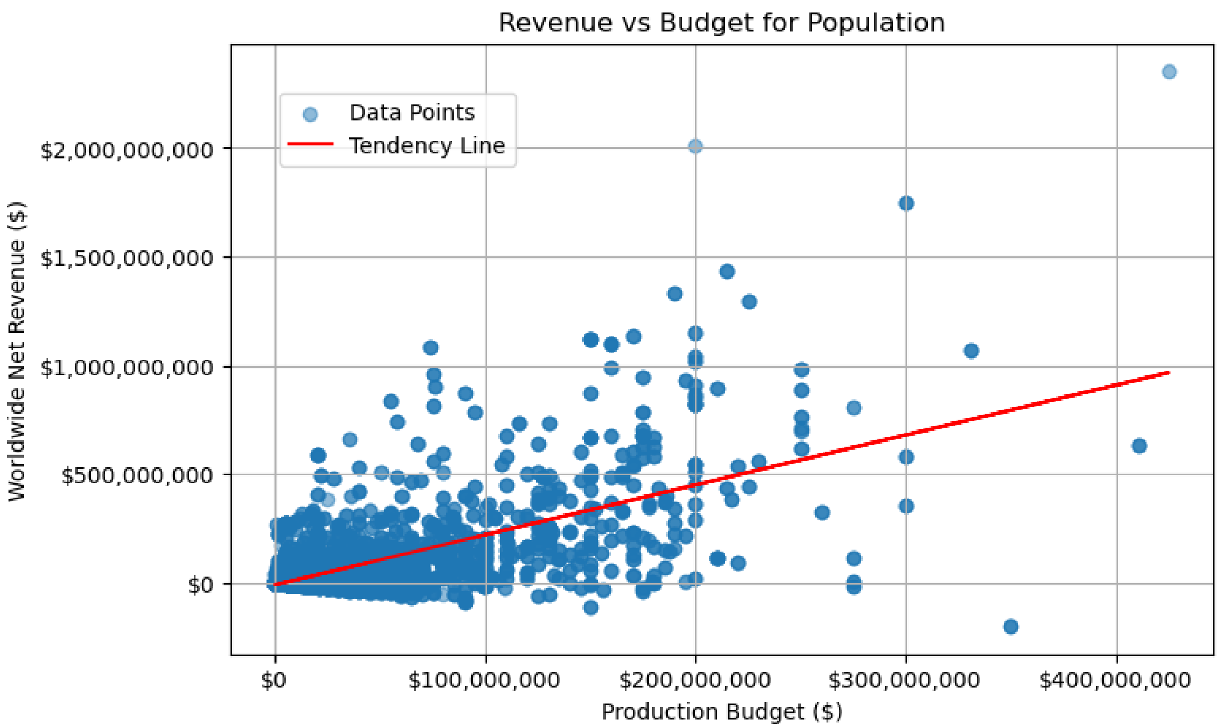
Finding the population correlation between `production_budget` and `worlwide_net_revenue`.

Lets add the correlation level and the R-squared coefficient. Lets use the regresión line as tendency line just to ilustrate the positive correlation.

In [43]:

```
# Fit linear regression model
model = LinearRegression().fit(b2_explode_cleaned_df[['production_budget']], b2_exploc

# Create scatter plot with regression line
fig, ax = plt.subplots(figsize=(8,5))
ax.scatter(b2_explode_cleaned_df['production_budget'], b2_explode_cleaned_df['worlwide
```

```
ax.plot(b2_explode_cleaned_df['production_budget'], model.predict(b2_explode_cleaned_c

# Set axis labels and title
ax.set_xlabel('Production Budget ($)')
ax.set_ylabel('Worldwide Net Revenue ($)')
ax.set_title('Revenue vs Budget for Population')

# Add a legend
ax.legend(["Data Points", "Tendency Line"], loc=(.05, .80))

# Add gridlines
ax.grid()

# Format tick labels
ax.get_xaxis().set_major_formatter(plt.FuncFormatter(lambda x, loc: "${:,.0f}".format(
ax.get_yaxis().set_major_formatter(plt.FuncFormatter(lambda x, loc: "${:,.0f}".format(

# Save and show plot
plt.savefig('plot.png', dpi=300, bbox_inches='tight')
plt.show()
```



In [44]:
```
# Imort library again, for some reason the first time was not enough
from scipy import stats

# R-squared coeficient
# calculate the slope, the intercept and the correlation coefficient
slope, intercept, r_value, p_value, std_err = stats.linregress(b2_explode_cleaned_df['

# calculating the coefficient of determination (R-squared)
r_squared = r_value**2

# print r-squared coeficient
print(f"R-squared coeficient: {r_squared}")

# Pearson Coeficient:
```

```
corr_coef, p_value = pearsonr(b2_explode_cleaned_df["production_budget"], b2_explode_c
print(f'Pearson coeficient: {corr_coef}\nThe p-value is: {p_value}')
```

```
R-squared coeficient: 0.42798445595949036
Pearson coeficient: 0.6542052093643786
The p-value is: 0.0
```

The pearson coeficients indicate a moderate-strong postive correlation between `production_budget` and `worlwide_net_revenue`. The R-squared suggests that less than half of the observed variation of `worlwide_net_revenue` can be explained by `production_budget`. This two points are not conclusive, but gives us an idea that overall there is no reason to believe that the higher the budget the higher the revenue.

Next steps requires identifying the correlation level across genres and creating a graph to better understand their level.

In [45]:
```
def get_rsquared_by_genre(df):
    """
    Iterates over the distinct genres in the 'by_genre' column of a pandas DataFrame
    and calculates the R-squared coefficient for each genre using linear regression.

    Parameters:
        df (pandas.DataFrame): The DataFrame to iterate over.

    Returns:
        A pandas DataFrame with two columns: 'Genre' and 'R-squared', containing the
        calculated R-squared coefficient for each distinct genre in the 'by_genre' col
    """
    # get list of distinct genres
    genres = df['by_genre'].unique()

    # initialize empty lists to store results
    genre_list = []
    rsquared_list = []

    # iterate over genres and calculate R-squared for each
    for genre in genres:
        # subset DataFrame for current genre
        genre_df = df[df['by_genre'] == genre]

        # calculate R-squared using linear regression
        slope, intercept, r_value, p_value, std_err = stats.linregress(genre_df['produ
        r_squared = r_value**2

        # append results to lists
        genre_list.append(genre)
        rsquared_list.append(r_squared)

    # create and return a new DataFrame with results
    result_df = pd.DataFrame({'Genre': genre_list, 'R-squared': rsquared_list}).sort_v
    return result_df
```

In [46]:
```
# Assign function ouput to 'dataframe'
rsquare_df = get_rsquared_by_genre(b2_explode_cleaned_df)

# Change column name from 'Genre' to 'by_genre'
rsquare_df.rename(columns={'Genre' : 'by_genre'}, inplace = True)
```

```
# Print df
rsquare_df
```

Out[46]:

	by_genre	R-squared
20	Musical	0.604143
16	Sport	0.522832
10	Horror	0.502762
7	Sci-Fi	0.485556
14	War	0.449724
6	Crime	0.438581
0	Action	0.425931
2	Comedy	0.396817
8	Family	0.387774
9	Thriller	0.384037
15	Fantasy	0.372668
3	Adventure	0.360849
21	News	0.349186
1	Animation	0.341073
13	History	0.322123
5	Romance	0.302557
12	Biography	0.256764
4	Drama	0.239826
18	Documentary	0.238663
17	Music	0.223204
11	Mystery	0.122329
19	Western	0.024072

In [47]:

```
# Dataframe containing pearson coeficient by movie genre
b2_corr_table = b2_explode_cleaned_df.groupby('by_genre')[['production_budget', 'worlw

# Drop column name 'level_1'
b2_corr_table.drop(columns = 'level_1', inplace = True)

# Change column name from 'Genre' to 'by_genre'
b2_corr_table.rename(columns={'worlwide_net_revenue' : 'Pearson_coef'}, inplace = True

# Print df
b2_corr_table
```

Out[47]:

	by_genre	Pearson_coef
0	Musical	0.777266
1	Sport	0.723071
2	Horror	0.709057
3	Sci-Fi	0.696818
4	War	0.670615
5	Crime	0.662254
6	Action	0.652634
7	Comedy	0.629934
8	Family	0.622715
9	Thriller	0.619707
10	Fantasy	0.610466
11	Adventure	0.600707
12	News	0.590920
13	Animation	0.584015
14	History	0.567559
15	Romance	0.550052
16	Biography	0.506719
17	Drama	0.489720
18	Documentary	0.488532
19	Music	0.472445
20	Mystery	0.349756
21	Western	0.155151

In [48]:

```
# Merge two dataframes
r_pear_coef_df = pd.merge(b2_corr_table ,rsquare_df, on='by_genre', how='left')
r_pear_coef_df
```

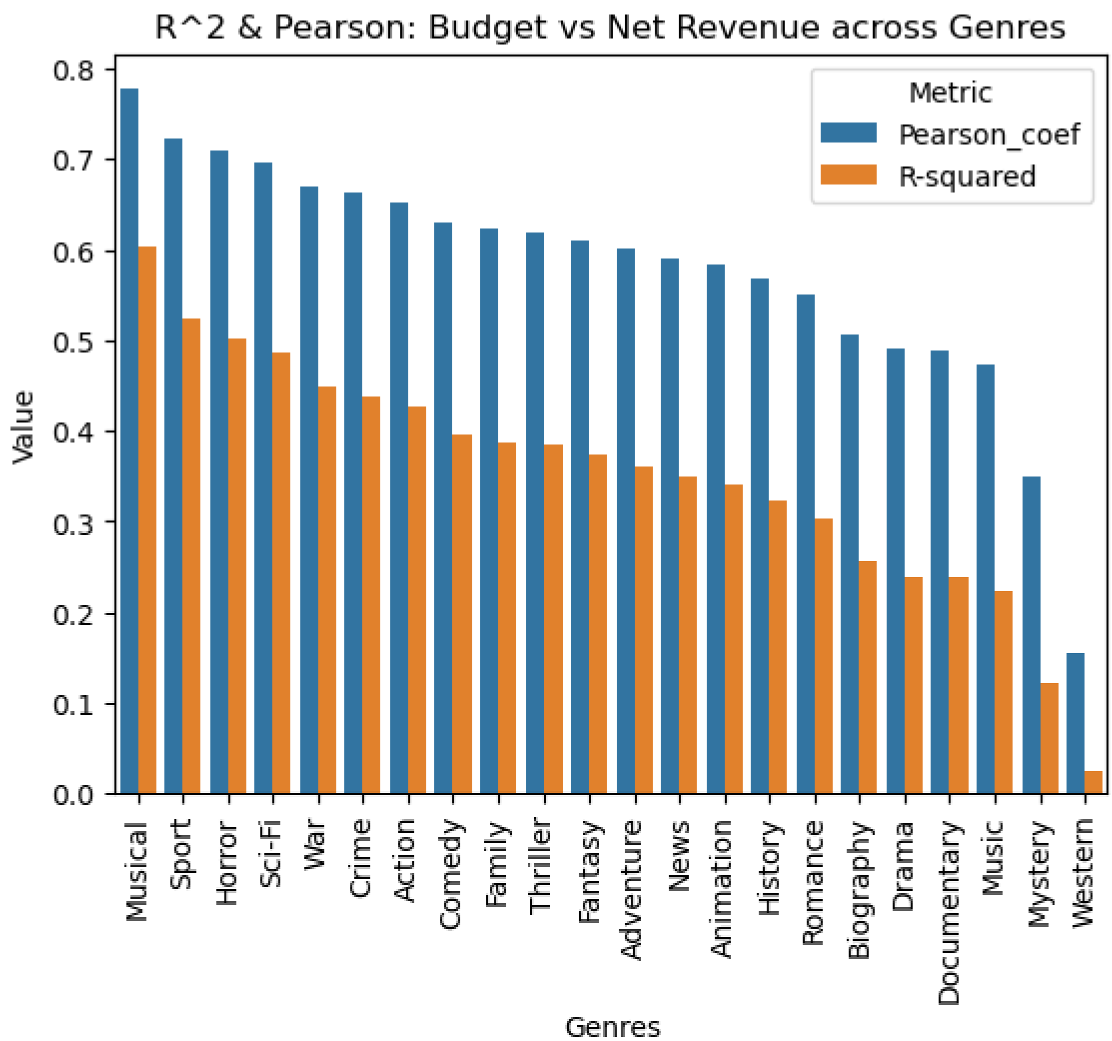
Out[48]:

	by_genre	Pearson_coef	R-squared
0	Musical	0.777266	0.604143
1	Sport	0.723071	0.522832
2	Horror	0.709057	0.502762
3	Sci-Fi	0.696818	0.485556
4	War	0.670615	0.449724
5	Crime	0.662254	0.438581
6	Action	0.652634	0.425931
7	Comedy	0.629934	0.396817
8	Family	0.622715	0.387774
9	Thriller	0.619707	0.384037
10	Fantasy	0.610466	0.372668
11	Adventure	0.600707	0.360849
12	News	0.590920	0.349186
13	Animation	0.584015	0.341073
14	History	0.567559	0.322123
15	Romance	0.550052	0.302557
16	Biography	0.506719	0.256764
17	Drama	0.489720	0.239826
18	Documentary	0.488532	0.238663
19	Music	0.472445	0.223204
20	Mystery	0.349756	0.122329
21	Western	0.155151	0.024072

In [49]:

```
# melt the DataFrame to create a tidy format
df_melt = pd.melt(r_pear_coef_df, id_vars='by_genre', var_name='metric', value_name='v

# create the bar graph
sns.barplot(x='by_genre', y = 'value', hue = 'metric', data = df_melt)

# add labels and legends to the graph
plt.xlabel('by_genre')
plt.ylabel('Value')
plt.legend(title='Metric', loc='upper right')

# Set axis labels and title
plt.xlabel('Genres')
# plt.ylabel('Correlation')
plt.title('R^2 & Pearson: Budget vs Net Revenue across Genres')

# Rotate xticks 90 degrees
plt.xticks(rotation=90)
```

```python
# Save and show plot
plt.savefig('plot_corre.png', dpi=300, bbox_inches='tight')
plt.show();
```



A moderate-strong positive correlation was observed between `production_budget` and `worldwide_net_revenue`. However, this correlation decreases significantly across different genres, indicating that the relationship is not universal. The r-squared coefficient is also significantly lower across genres, indicating that there is not a strong relation between `production_budget` and `worldwide_net_revenue`. This suggests that higher budgets do not translate to higher revenues, neither for the population nor across genres.

Business Case 2: Conclusion

The conclusion of this business case is that Computer Vision should be carefull when thinking that a bigger budget will translate into higher revenue. So to speak, there is an adequate budget. Unfortunatedly, the data provided does not contain more data that could help us understand what other factors may contribute to high revenue.

Business Case #3: Release Date(Season/Month) And Correlation With Popularity/Profit

In this business Case we are analyzing the correlation between `Release Date` of Movies from `Seasons` and `Months` to `Popularity` and `Profit`.

To begin our analysis of the correlation between seasons, we will have to organize our dataset by Movies Released per seasonn and storing it in a new list.

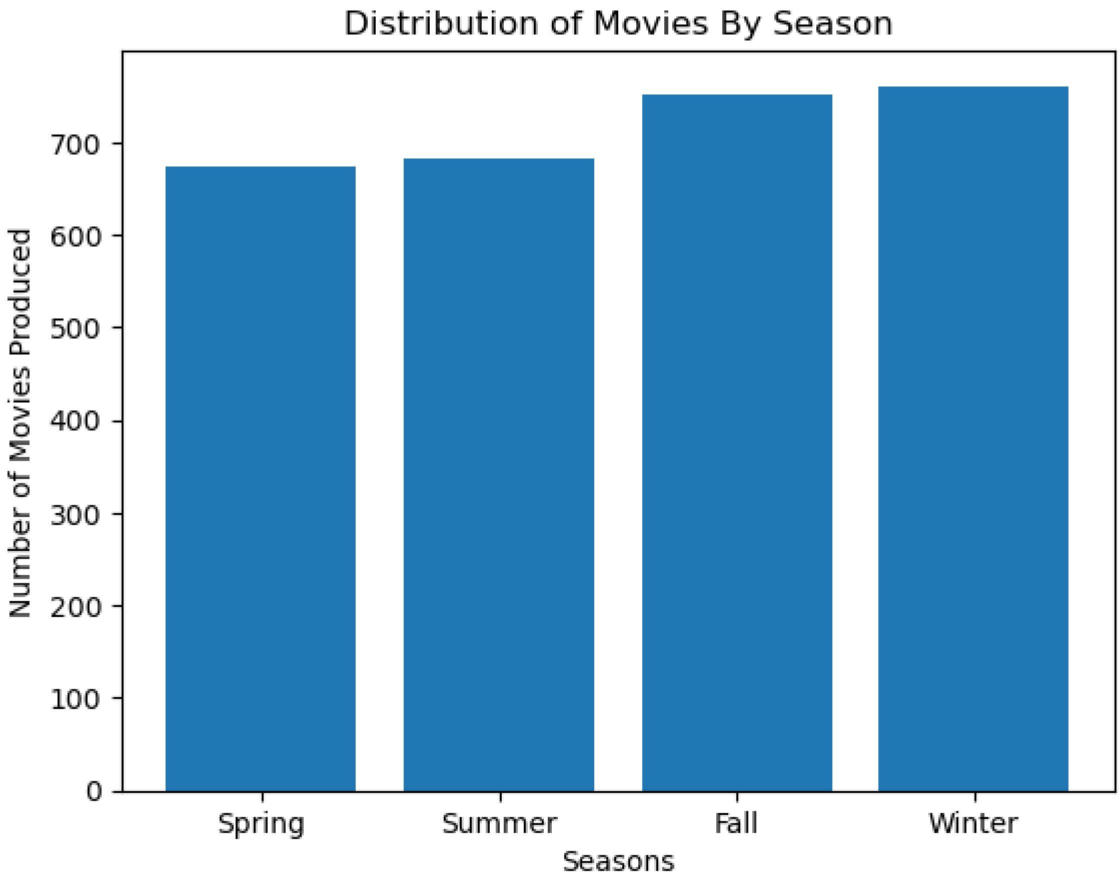
In [50]:
```
#Count total movies in each season
seasons = {'Spring': ['Mar', 'Apr', 'May'], 'Summer': ['Jun', 'Jul', 'Aug'], 'Fall': [

cleaned_df['release_date']

season_dict = {'Spring':0,'Summer':0,'Fall':0,'Winter':0,}

"""
    This function takes in the release date in our dataframe and returns which season

    Args:
        Release Date within the dataframe

    Returns:
        String: Season Name
"""
def get_season(x):
    for key,val in seasons.items():
        if x in val:
            return key

'''
    This Function that counts up the number of movies release in each season

    Returns:
        New appended dictionary of Movies Produced per season
'''
def season_count():
    for i in cleaned_df['release_date']:
        x = i.split('-')
        key = get_season(x[1])
        season_dict[key]+=1
    return season_dict

season_count()
```

Out[50]:
```
{'Spring': 673, 'Summer': 682, 'Fall': 751, 'Winter': 761}
```

To visualize the dictionary, a bar graph is created and showb below

In [51]:
```
#Bar Chart for movies released per season
fig, ax = plt.subplots()
```

```
ax.bar(season_dict.keys(),season_dict.values())
ax.set_xlabel('Seasons')
ax.set_ylabel('Number of Movies Produced')
ax.set_title('Distribution of Movies By Season')

plt.savefig("figure.png")
```



We can see from the result that Fall-Winter has the most movie released

We will now Count the VoteCount Popularity Between Each Season to and storing it in a new list to analyze any correlation between number of votes to seasons

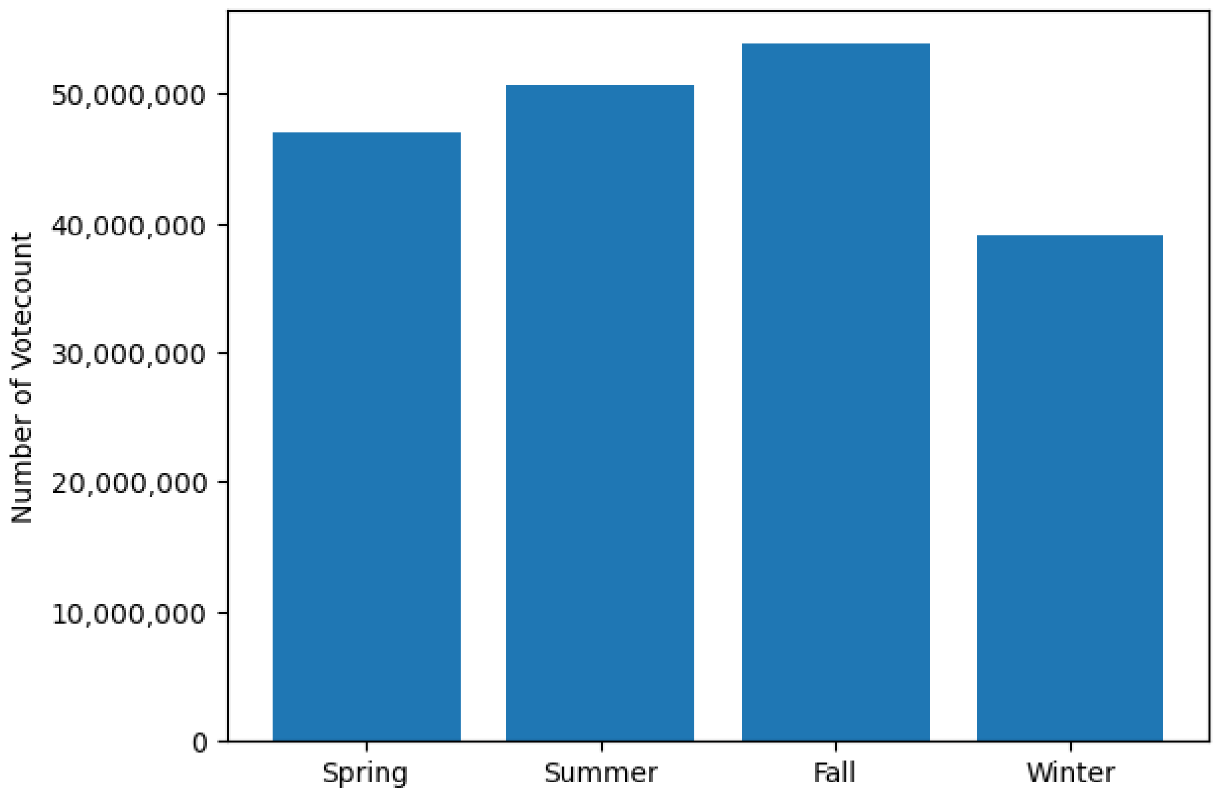
In [52]:
```
#Count of Popularity between each season

cleaned_df['numvotes']

popularity_dict={'Spring':0,'Summer':0,'Fall':0,'Winter':0,}

length = cleaned_df.shape[0]

'''
    This Function creates a new list and adds total number of votes to each season

    Returns:
        New appended dictionary of number of votes per season
'''

def popularity_count():
    for i in cleaned_df.itertuples():
        numvote= i[5]
```

```
        x=i[7].split('-')
        y=get_season(x[1])
        popularity_dict[y]+=numvote
    return popularity_dict

popularity_count()
```

Out[52]:
```
{'Spring': 46994534, 'Summer': 50685780, 'Fall': 53826543, 'Winter': 39049091}
```

Bar Chart for visualization

In [53]:
```
#Bar Chart
import matplotlib as mpl
fig, ax = plt.subplots()

ax.bar(popularity_dict.keys(),popularity_dict.values())
#ax.set_xlabel('Season')
ax.set_ylabel('Number of Votecount')
#ax.set_title('Distribution of Votecount By Season')
ax.yaxis.set_major_formatter(mpl.ticker.StrMethodFormatter('{x:,.0f}'))

plt.savefig("figure4.png")
```



We can see that Summer-Fall is the most popular season for movies

To find an accurate rating to number of votes for each season, we will need to perform a weighted analysis for each season. We will average the ratings per season and store in a list

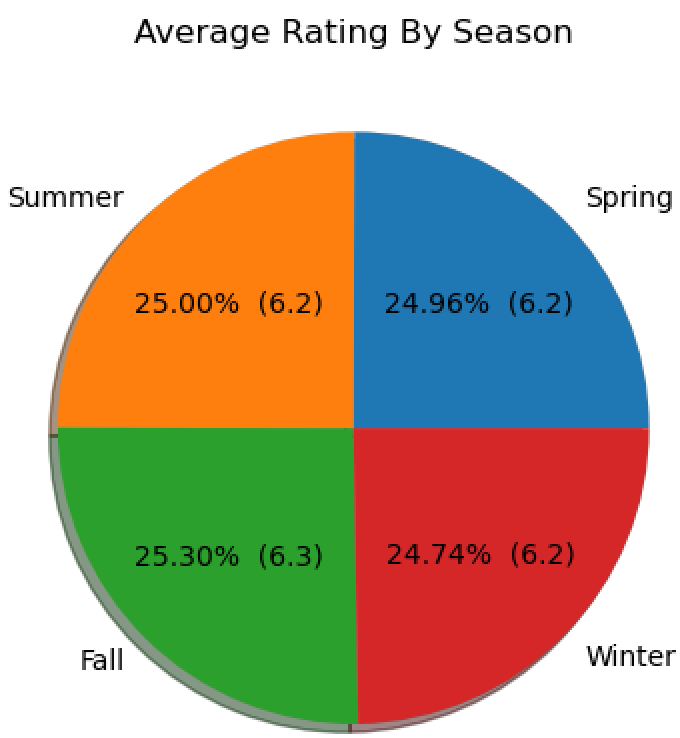
In [54]:
```
#Ratio of Rating to Numvotes of each season List
spring_averagerating_list=[]
summer_averagerating_list=[]
fall_averagerating_list=[]
```

```
winter_averagerating_list=[]

#average numvotes
spring_numvote_list=[]
summer_numvote_list=[]
fall_numvote_list=[]
winter_numvote_list=[]

#Average Rating of each Season List
spring_rating_list=[]
summer_rating_list=[]
fall_rating_list=[]
winter_rating_list=[]

#Loops through DataFrame and adds up rating for each season
for i in cleaned_df.itertuples():
    rating = i[4]
    numvotes = i[5]
    avg = rating/ numvotes *100
    x=i[7].split('-')
    y=get_season(x[1])
    if y == 'Spring':
        spring_averagerating_list.append(avg)
        spring_rating_list.append(rating)
        spring_numvote_list.append(numvotes)
    if y == 'Summer':
        summer_averagerating_list.append(avg)
        summer_rating_list.append(rating)
        summer_numvote_list.append(numvotes)
    if y == 'Fall':
        fall_averagerating_list.append(avg)
        fall_rating_list.append(rating)
        fall_numvote_list.append(numvotes)
    if y == 'Winter':
        winter_averagerating_list.append(avg)
        winter_rating_list.append(rating)
        winter_numvote_list.append(numvotes)

sp_mean = np.mean(spring_averagerating_list)
su_mean = np.mean(summer_averagerating_list)
f_mean = np.mean(fall_averagerating_list)
w_mean = np.mean(winter_averagerating_list)

sp_mean2 = np.mean(spring_rating_list)
su_mean2 = np.mean(summer_rating_list)
f_mean2 = np.mean(fall_rating_list)
w_mean2 = np.mean(winter_rating_list)

print("Average Spring Rating:",sp_mean2)
print("Average Summer Rating:",su_mean2)
print("Average Fall Rating:", f_mean2)
print("Average Winter Rating:", w_mean2,"\n")
```

```
Average Spring Rating: 6.238335809806834
Average Summer Rating: 6.2497067448680355
Average Fall Rating: 6.323035952063916
Average Winter Rating: 6.1851511169513795
```

Graphing Average Rating per Season

In [55]:
```
#Pie Chart for Average Rating
fig,ax = plt.subplots()

x = {'Spring': sp_mean, 'Summer': su_mean, 'Fall': f_mean, 'Winter': w_mean}
xx = {'Spring': sp_mean2, 'Summer': su_mean2, 'Fall': f_mean2, 'Winter': w_mean2}

#Function to format chart
def make_autopct(values):
    def my_autopct(pct):
        total = sum(values)
        val = round((pct*total/100.0),2)
        return ('{p:.2f}%  ({v:.1f})'.format(p=pct,v=val))
    return my_autopct

ax.pie(xx.values(),labels=xx.keys(),autopct = make_autopct(xx.values()), shadow=True)
ax.set_title('Average Rating By Season')

plt.show()
```



Based off of the pie chart, we can see that the rating for each season are pretty similar but Fall and Summer has the greatest Ratings

Within our given dataset, we can see that there may be some rating:numvotes ratio that can be inaccurate which will skew our analysis. We will now perform a weighted rating calculation to find accurate rating to number of votes.

In [56]:
```
#weighted rating (WR) = (v ÷ (v+m)) × R + (m ÷ (v+m)) × C , where:

#* R = average for the movie (mean) = (Rating)
```

```
#* v = number of votes for the movie = (votes)
#* m = minimum votes required to be listed in the Top 250 (currently 3000)
#* C = the mean vote across the whole report (currently 6.9)

#list of each season
spring_list=[]
summer_list=[]
fall_list=[]
winter_list=[]

#median of each season
med_sp = np.median(spring_numvote_list)
med_su = np.median(summer_numvote_list)
med_f = np.median(fall_numvote_list)
med_w = np.median(winter_numvote_list)

#Iterates through dataframe to perform calculations and append to a new list
for i in cleaned_df.itertuples():
    rating = i[4]
    numvotes = i[5]
    x=i[7].split('-')
    y=get_season(x[1])

    if y == 'Spring':
        wr = (numvotes / (numvotes+med_sp)) * rating + (med_sp / (numvotes + med_sp))
        spring_list.append(wr)
    elif y == 'Summer':
        wr = (numvotes / (numvotes+med_su)) * rating + (med_su / (numvotes + med_su))
        summer_list.append(wr)
    elif y == 'Fall':
        wr = (numvotes / (numvotes+med_f)) * rating + (med_f / (numvotes + med_f)) * f
        fall_list.append(wr)
    elif y == 'Winter':
        wr = (numvotes / (numvotes+med_w)) * rating + (med_w / (numvotes + med_w)) * w
        winter_list.append(wr)

#average list
avg_sp_mean = np.mean(spring_list)
avg_su_mean = np.mean(summer_list)
avg_f_mean = np.mean(fall_list)
avg_w_mean= np.mean(winter_list)

print('Weighted Average Spring Rating: ',avg_sp_mean )
print('Weighted Average Summer Rating: ',avg_su_mean )
print('Weighted Average Fall Rating: ',avg_f_mean )
print('Weighted Average Winter Rating: ',avg_w_mean )
```

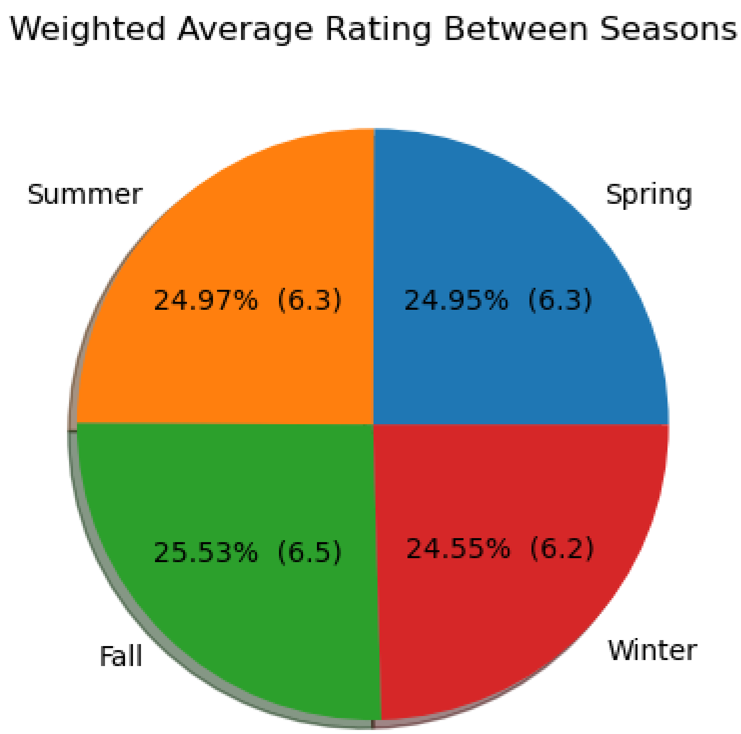
```
Weighted Average Spring Rating:  6.32352831735592
Weighted Average Summer Rating:  6.327367181515572
Weighted Average Fall Rating:  6.470400097919406
Weighted Average Winter Rating:  6.221610798601266
```

Graph of Average Weight

In [57]:

```
#Pie Chart for weighed average
xxx = {'Spring': avg_sp_mean, 'Summer': avg_su_mean, 'Fall': avg_f_mean, 'Winter': avg

fig3,ax3 = plt.subplots()

ax3.pie(xxx.values(),labels=xxx.keys(),autopct = make_autopct(xxx.values()), shadow=Tr
```

```
ax3.set_title('Weighted Average Rating Between Seasons')

plt.show()
```



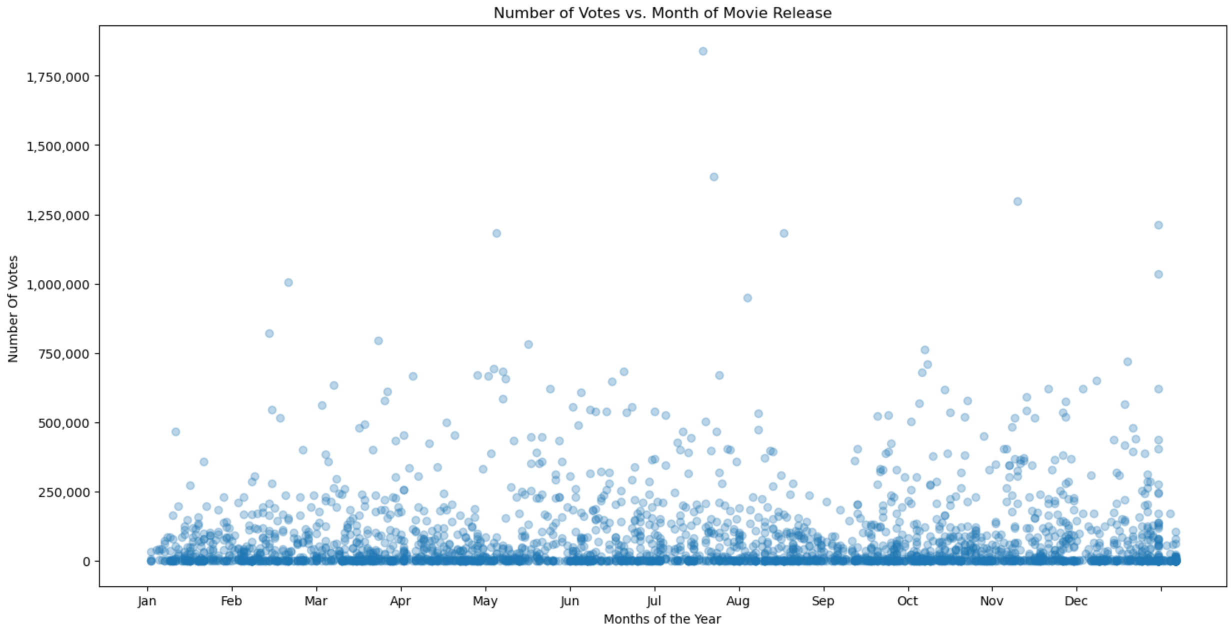
With the weighted rating calculations, we can see that Fall and Summer is the most popular

To understand correlation between seasons and release Date we will analyze the specific month of release date and its correlation with popularity.

In [58]:
```
#correlation between release date vs. numvotes
import calendar
from datetime import datetime
from matplotlib.ticker import (MultipleLocator,
                               FormatStrFormatter,
                               AutoMinorLocator)
fig, ax = plt.subplots(figsize=(16, 8))

month_list=[]
month_value = {'Jan':1,'Feb':32,'Mar': 60,'Apr':91, 'May':121, 'Jun':152, 'Jul':182, '

#Convert Date to Int
for i in cleaned_df['release_date']:
        x = i.split('-')
        val = month_value[x[1]]
        y = x[0]
        month_list.append(val + int(y) - 1)

#Plot
label = ['','Jan', 'Feb', 'Mar','Apr','May','Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov',

ax.scatter(month_list,cleaned_df['numvotes'],alpha=0.3)
ax.set_xlabel('Months of the Year')
```

```
ax.set_ylabel('Number Of Votes ')
ax.set_title('Number of Votes vs. Month of Movie Release')

ax.xaxis.set_major_locator(MultipleLocator(30))
ax.yaxis.set_major_formatter(mpl.ticker.StrMethodFormatter('{x:,.0f}'))
ax.set_xticklabels(label)


plt.savefig('plot.png', dpi=300, bbox_inches='tight')
plt.show()
```

```
C:\Users\raguilarsoriano\AppData\Local\Temp\ipykernel_18552\4095425253.py:29: UserWar
ning: FixedFormatter should only be used together with FixedLocator
  ax.set_xticklabels(label)
```



The correlation between number of vote and release month, has weak correlation but the highest popularity of the movie based on our limited datset, is around Summer Time

Here we are trying to find the `Total Profit` for both domestic and worldwide related to each season

In [59]:
```
#Business case #3
#I'm trying to count the total profit, both domestic and worldwide, for each season(s)
#Mercedez & Jordan

seasons = {'Spring': ['Mar','Apr','May'], 'Summer': ['Jun','Jul','Aug',], 'Fall': ['Se

test = {'Spring':0,'Summer':0,'Fall':0,'Winter':0,}

"""
    This function takes in the release date in our dataframe and returns which season

    Args:
        Release Date within the dataframe

    Returns:
        String: Season Name
"""
```

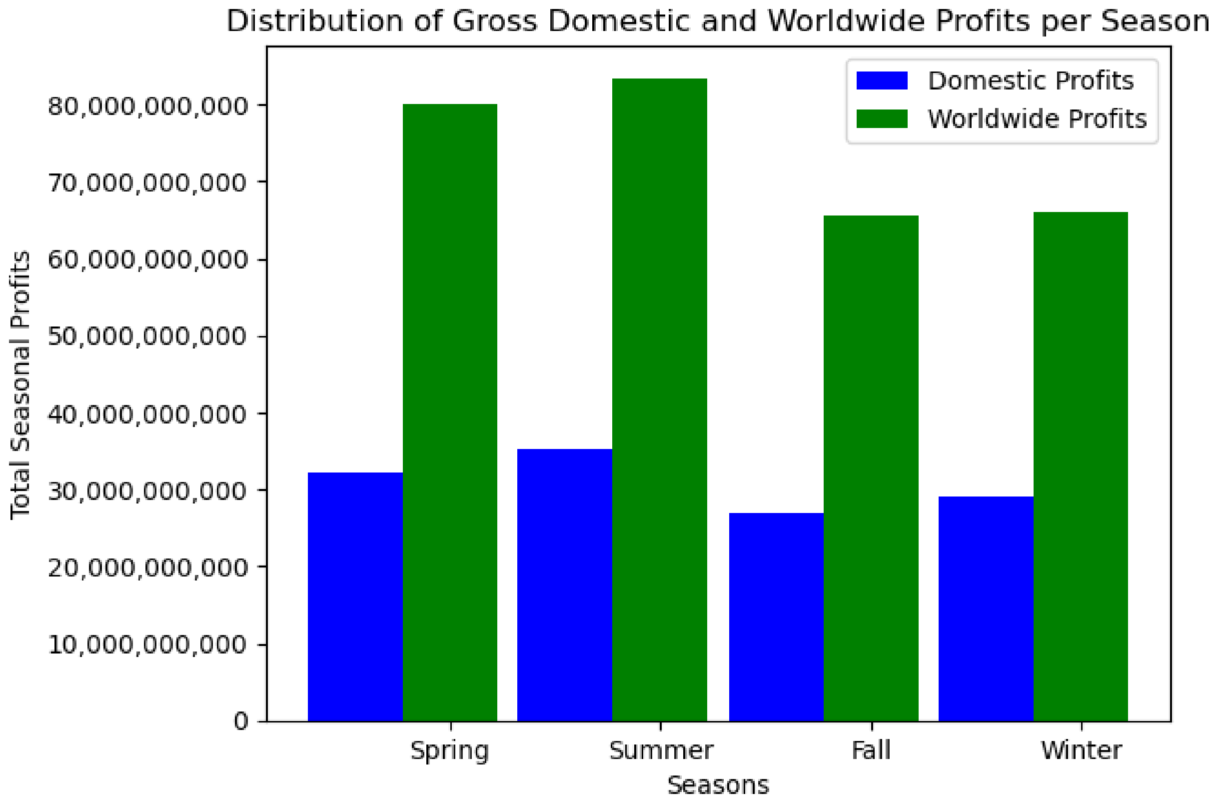
```python
def get_season(x):
    for key,val in seasons.items():
        if x in val:
            return key
#This is for gross domestic
for i in cleaned_df.itertuples():
    date = i[7]
    x=i[7].split('-')
    y=get_season(x[1])
    num=i[9]
    test[y]+=int(num)
print('gross domestic', test)

gross_worldwide = {'Spring':0,'Summer':0,'Fall':0,'Winter':0,}
#This is for gross worldwide
for i in cleaned_df.itertuples():
    date = i[7]
    x=i[7].split('-')
    y=get_season(x[1])
    num = i[10]
    gross_worldwide[y]+=int(num)
print('gross worldwide', gross_worldwide)
```

```
gross domestic {'Spring': 32267117335, 'Summer': 35185851273, 'Fall': 26847802292, 'W
inter': 29023623325}
gross worldwide {'Spring': 79989060868, 'Summer': 83425075374, 'Fall': 65576756077,
'Winter': 65967303186}
```

This is displaying the data we found above for both gross domestic and gross worldwide total profits for all seasons. This is displaying a bar chart.

In [60]:
```python
#Business Case #3
#Bar Chart for the categorical data of seasons and profits
#Mercedez & Jordan

import matplotlib as mpl
fig, ax = plt.subplots()
X = ['Spring','Summer','Fall','Winter'] #x values manually entered

ind = np.arange(len(season_dict.keys()))
N = 2 #num of bars
width = 0.45 # the width of the bars

rects1 = ax.bar(ind,test.values(), width, color='b')
rects2 = ax.bar(ind+width,gross_worldwide.values(), width, color='g')

ax.set_xlabel('Seasons')
ax.set_ylabel('Total Seasonal Profits')
ax.set_title('Distribution of Gross Domestic and Worldwide Profits per Season')
ax.yaxis.set_major_formatter(mpl.ticker.StrMethodFormatter('{x:,.0f}'))
ax.set_xticks(ind+width, ('Spring', 'Summer', 'Fall', 'Winter') )
ax.legend( (rects1[0], rects2[0]), ('Domestic Profits', 'Worldwide Profits') )
```

Out[60]:
```
<matplotlib.legend.Legend at 0x1f63333d5e0>
```



Full statistical analysis on release date and its correlation to worldwide profits to seasons

In [61]:
```
#Business case #3
#Full Statistical Analysis
#Mercedez
```

Statistical Analysis

```
- Null Hypothesis: There is no difference between the seasonal
release date and gross worldwide profits
- Alternative Hypothesis: There is a difference between the seasonal
release date and gross worldwide profits, the sample mean for gross
worldwide profits is higher in the Spring & Summer seasons than Fall
& Winter seasons
- Conducting a one-tailed Z-Test to calculate the statistical
significance for this one direction (greater than value)
- The significance level (alpha) is 0.05, indicating a 95% confidence
level that gross worldwide profits are higher during the Spring &
Summer seasons
- The Z-Test score is 1.50, p-value= 0.065, meaning movies released
in Spring & Summer are in the 93.94th percentile
- Recommendation: Since the z-test score revealed a p-value of 0.065,
bigger than our alpha 0.05, we fail to reject the null hypothesis,
and therefore we cannot recommend (with a 95% confidence level),  a
specific season to release a movie
 - Speculation: Spring & Summer seasonal release dates for a movie
will yield higher gross worldwide profits, being that people have
more free time and therefore plan for more entertainment, as well as
```

```
seek indoor venues to escape "warmer" temperatures
 - Potential limitation: Given more time for analysis, a test for
independence between the variables being examined could be conducted
using a Chi-squared test
```

In [62]:
```
#Code for z-test statistical analysis
#Mercedez

import scipy.stats as stats
from math import sqrt

print('Total profits:',gross_worldwide['Summer'] + gross_worldwide['Spring'])

summer_list = []
winter_list=[]
fall_list=[]
spring_list=[]

for i in cleaned_df.itertuples():
    x= i[7].split('-')
    key = get_season(x[1])
    num = i[9]
    if key == 'Summer':
        summer_list.append(int(num))
    if key == 'Winter':
        winter_list.append(int(num))
    if key == 'Fall':
        fall_list.append(int(num))
    if key == 'Spring':
        spring_list.append(int(num))

print('Sample Mean:', np.mean(summer_list) + np.mean(spring_list))
#print('Sample Number:', len(summer_list))

x_bar = np.mean(summer_list) + np.mean(spring_list) #sample mean
n = 4 # number of seasons in a year
sigma = np.std(summer_list + winter_list + fall_list + spring_list) #sd of population
mu = (np.mean(summer_list) + np.mean(winter_list) + np.mean(fall_list) + np.mean(sprir
print('Population Mean:',mu)

z = (x_bar - mu)/(sigma/(sqrt(n)))

print('Z-Score:',z)
```

```
Total profits: 163414136242
Sample Mean: 99537353.92626137
Population Mean: 43356388.302691706
Z-Score: 1.5062667651495287
```

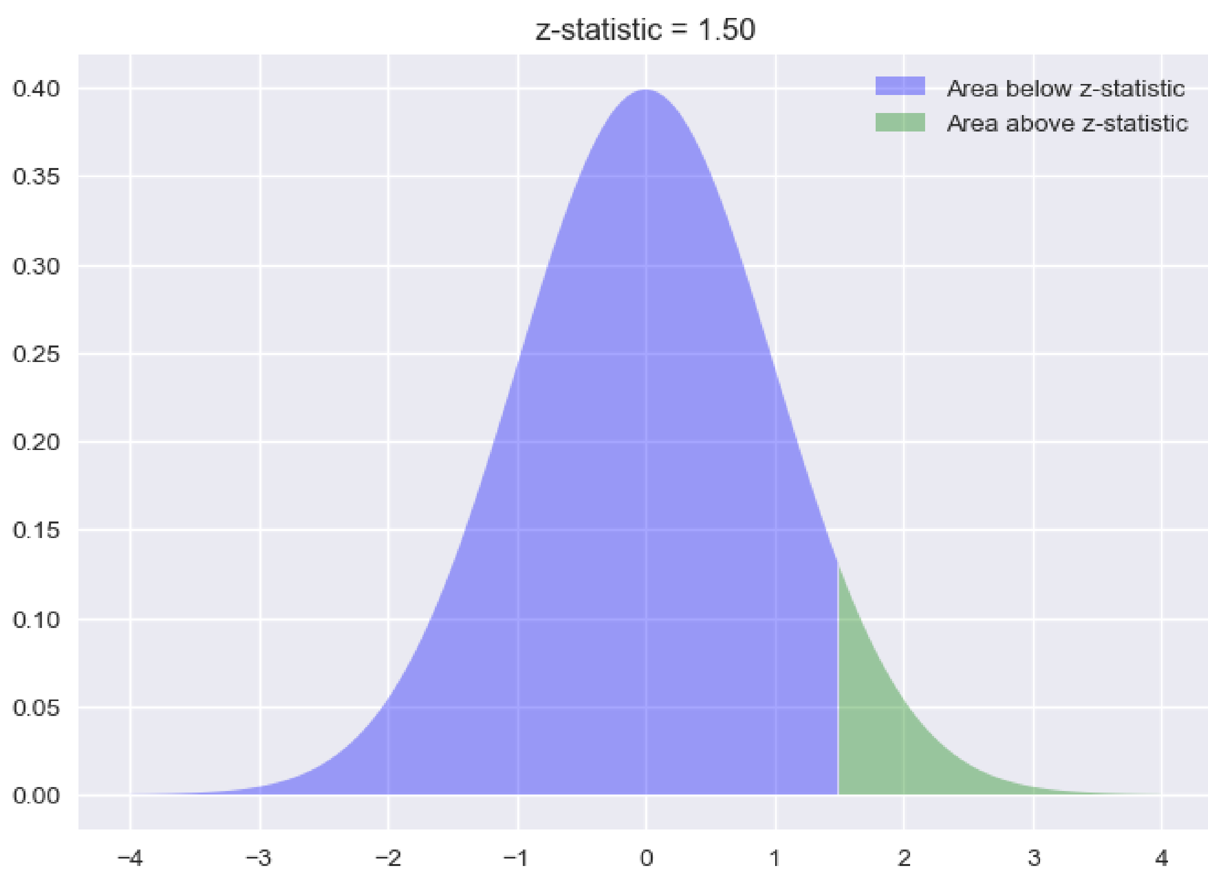
This is to run our z-score to find our p-value

In [63]:
```
pval = 1 - stats.norm.cdf(z)
pval
```

Out[63]:
```
0.06599935310883931
```

This is to plot our z-statistic following a normal distribution, one tailed z-test

In [64]:
```
#plot of value on a standard normal distribution
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

plt.style.use('seaborn')
plt.fill_between(x=np.arange(-4,1.50,0.01),
                 y1= stats.norm.pdf(np.arange(-4,1.50,0.01)) ,
                 facecolor='blue',
                 alpha=0.35,
                 label= 'Area below z-statistic'
                 )

plt.fill_between(x=np.arange(1.50,4,0.01),
                 y1= stats.norm.pdf(np.arange(1.50,4,0.01)) ,
                 facecolor='green',
                 alpha=0.35,
                 label= 'Area above z-statistic')
plt.legend()
plt.title ('z-statistic = 1.50');
```



Business Case 3: Conclusion

Even though there was no statistical significance between release date and gross worldwide profits, there is however a close proximity between our p-value of 0.065 to our alpha value of 0.05, showing that there is a close relationship between `Spring` & `Summer` release dates yielding higher gross profits than Fall & Winter release dates. Further analysis can be conducted given more time.