

**project**

**1**

# Factors Contributing to the Success of a Movie Analysis

*Is the movie industry dying? What determines the success of a movie?*

## Goal

*The goal of this project is to identify the factors that contribute to the success of a movie, specifically whether the production company, the movie's budget, or other variables play a significant role in determining a movie's success. By performing a data analysis using Python, this project aims to provide valuable insights that can help movie production companies make more informed decisions about which movies to produce, which companies to partner with, and how to allocate resources to maximize the chances of success.*

# Contributing to the success of a movie

February 24, 2024

## 1 Factors Contributing to the Success of a Movie Analysis

*Is the movie industry dying? What determines the success of a movie?*

### 1.1 Goal

*The goal of this project is to identify the factors that contribute to the success of a movie, specifically whether the production company, the movie's budget, or other variables play a significant role in determining a movie's success. By performing a data analysis using Python, this project aims to provide valuable insights that can help movie production companies make more informed decisions about which movies to produce, which companies to partner with, and how to allocate resources to maximize the chances of success.*

### 1.2 Setup

Importing the necessary libraries I will use for this project

```
[72]: import pandas as pd
import numpy as np
import seaborn as sns

from scipy import stats

import matplotlib.pyplot as plt
import matplotlib.mlab as mlab
import matplotlib
plt.style.use('ggplot')
from matplotlib.pyplot import figure

%matplotlib inline
matplotlib.rcParams['figure.figsize'] = (12,8)

pd.options.mode.chained_assignment = None
```

Reading in the data

```
[2]: df = pd.read_csv(r'C:\Users\parri\Downloads\movies.csv')
```

Looking at the data

```
[3]: df.head()
```

```
[3]:          name rating      genre    year   released    votes \
0     Friday the 13th      R    Horror  1980  1980-09-05  123000.0
1        Raging Bull      R  Biography  1980  1980-12-19  330000.0
2   The Long Riders      R  Biography  1980  1980-05-16  10000.0
3 Any Which Way You Can    PG    Action  1980  1980-12-17  18000.0
4 The Gods Must Be Crazy    PG Adventure  1980  1984-10-26  54000.0

           director        writer        star        country \
0  Sean S. Cunningham  Victor Miller  Betsy Palmer  United States
1  Martin Scorsese     Jake LaMotta  Robert De Niro  United States
2    Walter Hill       Bill Bryden  David Carradine  United States
3  Buddy Van Horn  Stanford Sherman  Clint Eastwood  United States
4        Jamie Uys        Jamie Uys        N!xau  South Africa

      budget      gross            company  runtime
0  5500000.0  39754601.0  Paramount Pictures    95.0
1 18000000.0  23402427.0  Chartoff-Winkler Productions  129.0
2 10000000.0  15795189.0  United Artists    100.0
3 15000000.0  70687344.0  The Malpaso Company   116.0
4  5000000.0  30031783.0  C.A.T. Films    109.0
```

## 1.3 Data Cleaning!

### 1.3.1 Checking for missing data

```
[4]: # Let's loop through the data and see if there is anything missing
```

```
for col in df.columns:
    percent_missing = np.mean(df[col].isnull())
    print('{} - {}%'.format(col, round(percent_missing*100)))
```

```
name - 0%
rating - 1%
genre - 0%
year - 0%
released - 0%
votes - 0%
director - 0%
writer - 0%
star - 0%
country - 0%
budget - 29%
gross - 3%
company - 0%
runtime - 0%
```

## Removing null values from the data

```
[5]: df = df.dropna()
```

## Dropping duplicates

```
[6]: df.drop_duplicates()
```

```
[6]:
```

		name	rating	genre	year	released	\
0		Friday the 13th	R	Horror	1980	1980-09-05	
1		Raging Bull	R	Biography	1980	1980-12-19	
2		The Long Riders	R	Biography	1980	1980-05-16	
3		Any Which Way You Can	PG	Action	1980	1980-12-17	
4		The Gods Must Be Crazy	PG	Adventure	1980	1984-10-26	
...		...	...	...	...	...	
6007		Bad Boys for Life	R	Action	2020	2020-01-17	
6008		Sonic the Hedgehog	PG	Action	2020	2020-02-14	
6009		Dolittle	PG	Adventure	2020	2020-01-17	
6010		The Call of the Wild	PG	Adventure	2020	2020-02-21	
6011		The Eight Hundred	Not Rated	Action	2020	2020-08-28	
		votes	director	writer	star	\	
0	123000.0	Sean S. Cunningham	Victor Miller	Betsy Palmer			
1	330000.0	Martin Scorsese	Jake LaMotta	Robert De Niro			
2	10000.0	Walter Hill	Bill Bryden	David Carradine			
3	18000.0	Buddy Van Horn	Stanford Sherman	Clint Eastwood			
4	54000.0	Jamie Uys	Jamie Uys	N!xau			
...	...	...	...	...	...	...	
6007	140000.0	Adil El Arbi	Peter Craig	Will Smith			
6008	102000.0	Jeff Fowler	Pat Casey	Ben Schwartz			
6009	53000.0	Stephen Gaghan	Stephen Gaghan	Robert Downey Jr.			
6010	42000.0	Chris Sanders	Michael Green	Harrison Ford			
6011	3700.0	Hu Guan	Hu Guan	Zhi-zhong Huang			
		country	budget	gross	\		
0	United States	550000.0	39754601.0				
1	United States	18000000.0	23402427.0				
2	United States	10000000.0	15795189.0				
3	United States	15000000.0	70687344.0				
4	South Africa	5000000.0	30031783.0				
...	...	...	...	...	...	...	
6007	United States	90000000.0	426505244.0				
6008	United States	85000000.0	319715683.0				
6009	United States	175000000.0	245487753.0				
6010	Canada	135000000.0	111105497.0				
6011	China	80000000.0	461421559.0				
		company	runtime				

```

0          Paramount Pictures      95.0
1  Chartoff-Winkler Productions  129.0
2          United Artists       100.0
3      The Malpaso Company     116.0
4          C.A.T. Films        109.0
...
...          ...
6007      Columbia Pictures     124.0
6008      Paramount Pictures     99.0
6009      Universal Pictures    101.0
6010  20th Century Studios    100.0
6011 Beijing Diqi Yinxiang Entertainment 149.0

```

[4208 rows x 14 columns]

### Checking the data type of the columns

[73]: `print(df.dtypes)`

name	object
rating	object
genre	object
year	int64
released	object
votes	float64
director	object
writer	object
star	object
country	object
budget	float64
gross	float64
company	object
runtime	float64
correctyear	object
correctmonth	object
profitability	float64
dtype:	object

### Changing the data type of columns

[8]: `# Some of the year columns and released have different years, we will splice the released date and use that year.`

```
df['correctyear'] = df['released'].astype(str).str[:4]
```

```
df
```

[8]:

	name	rating	genre	year	released
0	Friday the 13th	R	Horror	1980	1980-09-05

1	Raging Bull	R	Biography	1980	1980-12-19
2	The Long Riders	R	Biography	1980	1980-05-16
3	Any Which Way You Can	PG	Action	1980	1980-12-17
4	The Gods Must Be Crazy	PG	Adventure	1980	1984-10-26
...	...	...	...	...	...
6007	Bad Boys for Life	R	Action	2020	2020-01-17
6008	Sonic the Hedgehog	PG	Action	2020	2020-02-14
6009	Dolittle	PG	Adventure	2020	2020-01-17
6010	The Call of the Wild	PG	Adventure	2020	2020-02-21
6011	The Eight Hundred	Not Rated	Action	2020	2020-08-28

	votes	director	writer	star	\
0	1230000.0	Sean S. Cunningham	Victor Miller	Betsy Palmer	
1	3300000.0	Martin Scorsese	Jake LaMotta	Robert De Niro	
2	10000.0	Walter Hill	Bill Bryden	David Carradine	
3	18000.0	Buddy Van Horn	Stanford Sherman	Clint Eastwood	
4	54000.0	Jamie Uys	Jamie Uys	N!xau	
...	...	...	...	...	...
6007	1400000.0	Adil El Arbi	Peter Craig	Will Smith	
6008	102000.0	Jeff Fowler	Pat Casey	Ben Schwartz	
6009	53000.0	Stephen Gaghan	Stephen Gaghan	Robert Downey Jr.	
6010	42000.0	Chris Sanders	Michael Green	Harrison Ford	
6011	3700.0	Hu Guan	Hu Guan	Zhi-zhong Huang	

	country	budget	gross	\
0	United States	550000.0	39754601.0	
1	United States	18000000.0	23402427.0	
2	United States	10000000.0	15795189.0	
3	United States	15000000.0	70687344.0	
4	South Africa	5000000.0	30031783.0	
...	...	...	...	...
6007	United States	90000000.0	426505244.0	
6008	United States	85000000.0	319715683.0	
6009	United States	175000000.0	245487753.0	
6010	Canada	135000000.0	111105497.0	
6011	China	80000000.0	461421559.0	

	company	runtime	correctyear
0	Paramount Pictures	95.0	1980
1	Chartoff-Winkler Productions	129.0	1980
2	United Artists	100.0	1980
3	The Malpaso Company	116.0	1980
4	C.A.T. Films	109.0	1984
...	...	...	...
6007	Columbia Pictures	124.0	2020
6008	Paramount Pictures	99.0	2020
6009	Universal Pictures	101.0	2020

```

6010          20th Century Studios    100.0      2020
6011  Beijing Diqi Yinxiang Entertainment    149.0      2020

```

[4208 rows x 15 columns]

### Adding a Month column

```
[9]: df['correctmonth'] = df['released'].astype(str).str[5:7]
```

```
df
```

```
[9]:          name   rating   genre   year   released \
0     Friday the 13th      R   Horror  1980  1980-09-05
1           Raging Bull      R Biography 1980  1980-12-19
2       The Long Riders      R Biography 1980  1980-05-16
3  Any Which Way You Can    PG Action  1980  1980-12-17
4  The Gods Must Be Crazy    PG Adventure 1980  1984-10-26
...
6007      Bad Boys for Life      R Action  2020  2020-01-17
6008  Sonic the Hedgehog    PG Action  2020  2020-02-14
6009        Dolittle      PG Adventure 2020  2020-01-17
6010  The Call of the Wild    PG Adventure 2020  2020-02-21
6011  The Eight Hundred  Not Rated  Action  2020  2020-08-28

          votes   director   writer   star \
0  1230000.0  Sean S. Cunningham  Victor Miller  Betsy Palmer
1  330000.0  Martin Scorsese  Jake LaMotta  Robert De Niro
2  10000.0  Walter Hill  Bill Bryden  David Carradine
3  18000.0  Buddy Van Horn  Stanford Sherman  Clint Eastwood
4  54000.0  Jamie Uys  Jamie Uys  N!xau
...
6007  140000.0  Adil El Arbi  Peter Craig  Will Smith
6008  102000.0  Jeff Fowler  Pat Casey  Ben Schwartz
6009  53000.0  Stephen Gaghan  Stephen Gaghan  Robert Downey Jr.
6010  42000.0  Chris Sanders  Michael Green  Harrison Ford
6011  3700.0  Hu Guan  Hu Guan  Zhi-zhong Huang

          country   budget   gross \
0  United States  550000.0  39754601.0
1  United States 18000000.0  23402427.0
2  United States 10000000.0  15795189.0
3  United States 15000000.0  70687344.0
4  South Africa  5000000.0  30031783.0
...
6007  United States 90000000.0  426505244.0
6008  United States 85000000.0  319715683.0
6009  United States 175000000.0  245487753.0
```

```

6010          Canada 135000000.0 111105497.0
6011          China  80000000.0 461421559.0

                                company  runtime correctyear correctmonth
0                  Paramount Pictures    95.0      1980          09
1  Chartoff-Winkler Productions   129.0      1980          12
2            United Artists    100.0      1980          05
3        The Malpaso Company   116.0      1980          12
4            C.A.T. Films    109.0      1984          10
...
6007          Columbia Pictures   124.0      2020          01
6008          Paramount Pictures   99.0      2020          02
6009          Universal Pictures  101.0      2020          01
6010        20th Century Studios  100.0      2020          02
6011 Beijing Diqi Yinxiang Entertainment  149.0      2020          08

```

[4208 rows x 16 columns]

## Ordering the Data

```
[10]: df.sort_values(by=['gross'], inplace=False, ascending=False)
```

```

[10]:                                     name  rating  genre  year \
4306                         Avatar  PG-13  Action  2009
5846             Avengers: Endgame  PG-13  Action  2019
2404                     Titanic  PG-13  Drama  1997
5252 Star Wars: Episode VII - The Force Awakens  PG-13  Action  2015
5691             Avengers: Infinity War  PG-13  Action  2018
...
567                 Crimewave  PG-13  Comedy  1985
4459           Tanner Hall     R  Drama  2009
2912           Ginger Snaps  Not Rated  Drama  2000
183                Parasite     R  Horror  1982
2534              Trojan War  PG-13  Comedy  1997

      released      votes      director      writer \
4306 2009-12-18  1100000.0  James Cameron  James Cameron
5846 2019-04-26   903000.0  Anthony Russo  Christopher Markus
2404 1997-12-19  1100000.0  James Cameron  James Cameron
5252 2015-12-18   876000.0       J.J. Abrams  Lawrence Kasdan
5691 2018-04-27   897000.0  Anthony Russo  Christopher Markus
...
567  1986-04-25     5300.0       Sam Raimi  Ethan Coen
4459 2015-01-15     3500.0  Francesca Gregorini  Tatiana von Fürstenberg
2912 2001-11-05     43000.0      John Fawcett  Karen Walton
183  1982-12-03     2300.0      Charles Band  Alan J. Adler
2534 1997-01-10     5800.0      George Huang  Andy Burg

```

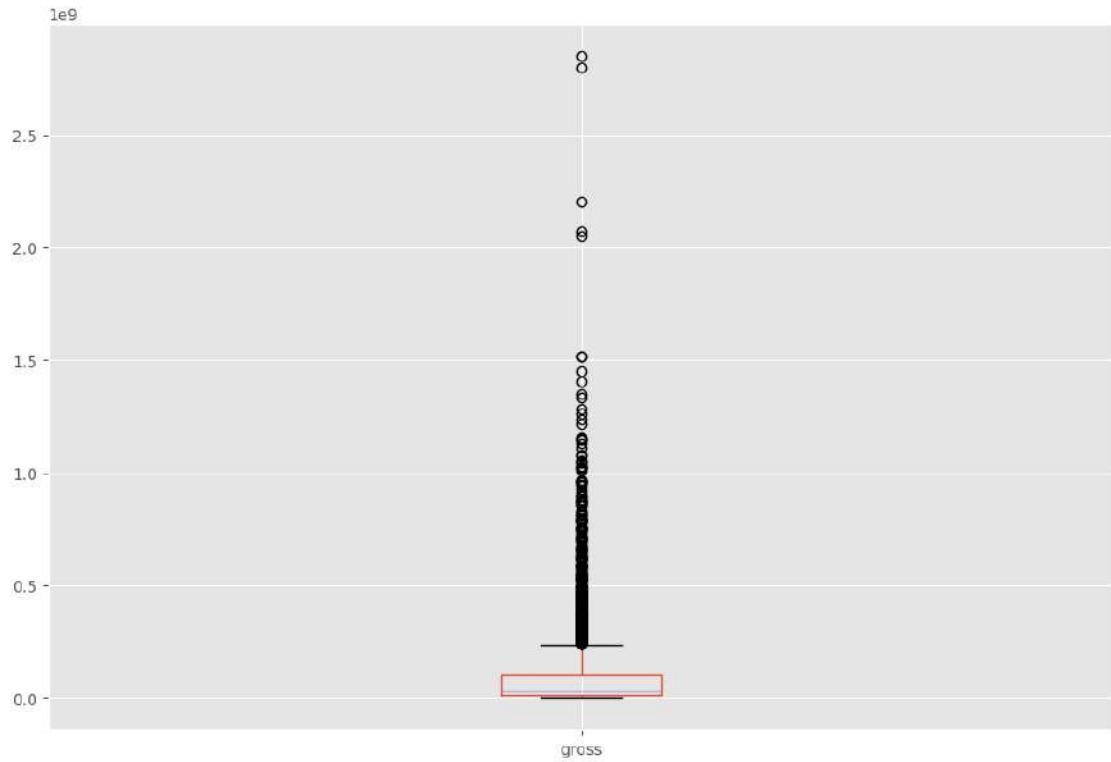
	star	country	budget	gross	\
4306	Sam Worthington	United States	237000000.0	2.847246e+09	
5846	Robert Downey Jr.	United States	356000000.0	2.797501e+09	
2404	Leonardo DiCaprio	United States	200000000.0	2.201647e+09	
5252	Daisy Ridley	United States	245000000.0	2.069522e+09	
5691	Robert Downey Jr.	United States	321000000.0	2.048360e+09	
...	...	...	...	...	
567	Louise Lasser	United States	3000000.0	5.101000e+03	
4459	Rooney Mara	United States	3000000.0	5.073000e+03	
2912	Emily Perkins	Canada	5000000.0	2.554000e+03	
183	Robert Glaudini	United States	800000.0	2.270000e+03	
2534	Will Friedle	United States	15000000.0	3.090000e+02	
		company	runtime	correctyear	correctmonth
4306	Twentieth Century Fox	162.0	2009	12	
5846	Marvel Studios	181.0	2019	04	
2404	Twentieth Century Fox	194.0	1997	12	
5252	Lucasfilm	138.0	2015	12	
5691	Marvel Studios	149.0	2018	04	
...	...	...	...	...	
567	Columbia Pictures	83.0	1986	04	
4459	Two Prong Lesson	96.0	2015	01	
2912	Copperheart Entertainment	108.0	2001	11	
183	Embassy Pictures	85.0	1982	12	
2534	Daybreak	85.0	1997	01	

[4208 rows x 16 columns]

### Checking for Outliers

[11]: df.boxplot(column=['gross'])

[11]: <AxesSubplot:>

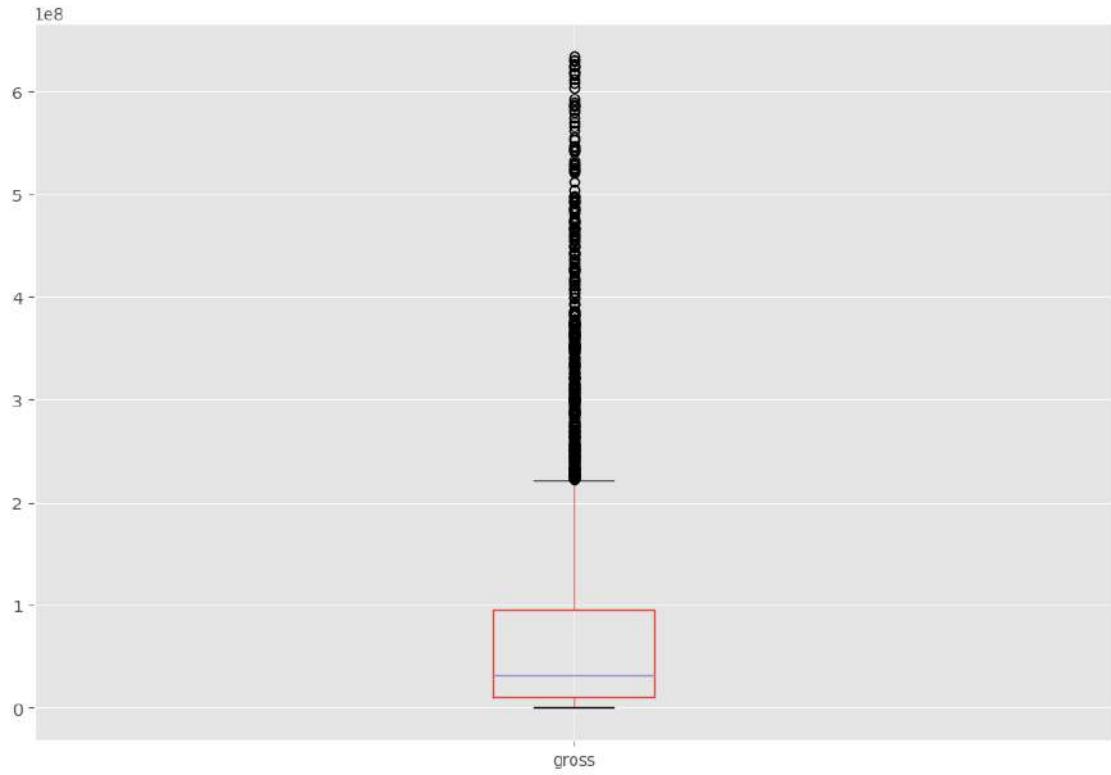


## Removing Outliers

```
[74]: z_scores = stats.zscore(df['gross'])
abs_z_scores = np.abs(z_scores)
filtered_entries = (abs_z_scores < 3)
df = df[filtered_entries]
```

```
[75]: df.boxplot(column=['gross'])
```

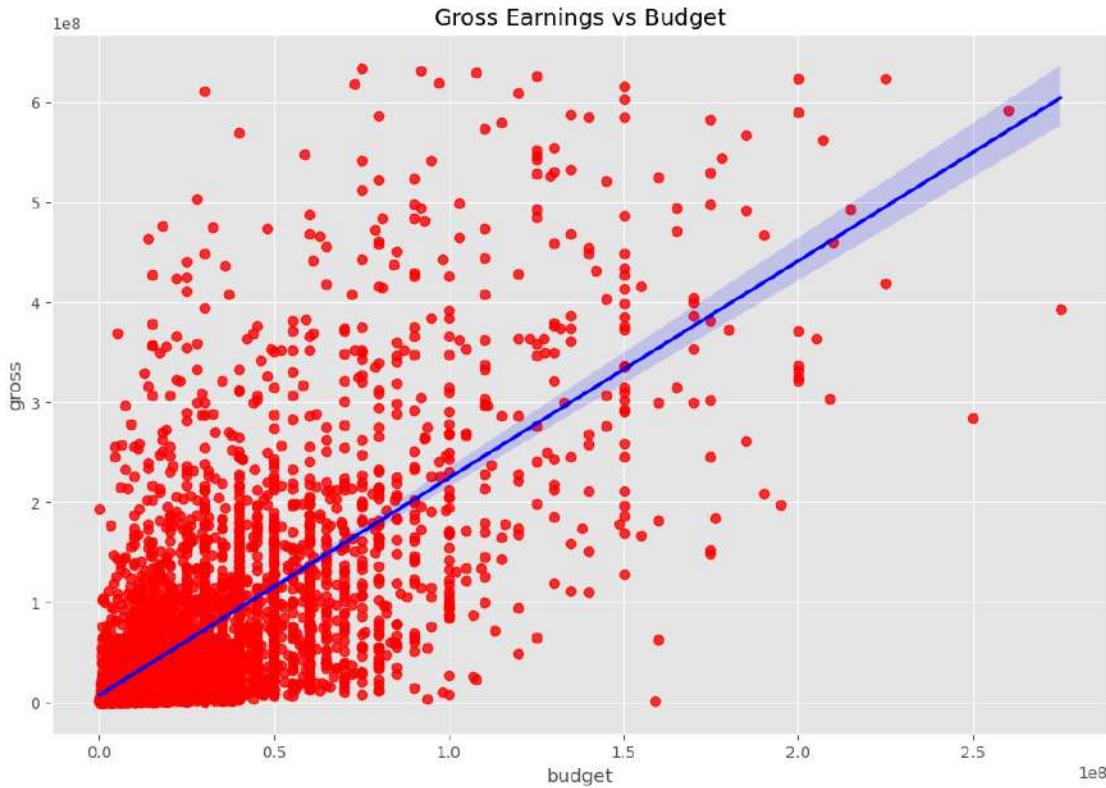
```
[75]: <AxesSubplot:>
```



## 1.4 Data Exploration!

In my analysis, I explored and answered the correlation of a movie's success to key factors in its creation such as budget, company, genre, etc..

```
[93]: # Looking at correlation  
  
sns.regplot(x="budget", y="gross", data=df, scatter_kws = {"color": "red"}, line_kws = {"color": "blue"}).set_title('Gross Earnings vs Budget')  
  
[93]: Text(0.5, 1.0, 'Gross Earnings vs Budget')
```



From this graph, we can determine that there is a significant relationship between the budget and gross earnings of a movie.

#### 1.4.1 Correlation Matrix

Let's take a deeper look at the relationship of all the other variables to see what correlates the most to gross earnings.

**Let's Update all the columns to numeric values** By doing this, we can show the correlation between all the variables and not only the ones with numerical values. This will help us to understand which variables are strongly or weakly correlated with each other.

```
[94]: df_numerized = df

for col_name in df_numerized.columns:
    if(df_numerized[col_name].dtype == 'object'):
        df_numerized[col_name]= df_numerized[col_name].astype('category')
        df_numerized[col_name] = df_numerized[col_name].cat.codes

df_numerized
```

```
[94]:      name  rating  genre  year  released    votes  director  writer  star \
0     1176      5       9  1980        10  123000.0     1519     2572    128
1     2389      5       3  1980        16  330000.0     1093     1047    1251
2     3343      5       3  1980        5  10000.0     1739     235     337
3      268      3       0  1980       15  18000.0     202     2341    277
4     3179      3       1  1980      129  54000.0     698     1085   1093
...
6007    321      5       0  2020      1765  140000.0      15     1992   1548
6008   2729      3       0  2020      1766  102000.0     729     1920    122
6009    895      3       1  2020      1765  53000.0     1562     2355   1252
6010   3028      3       1  2020      1767  42000.0     249     1738    539
6011   3117      2       0  2020      1772  3700.0     636      987   1571

      country      budget      gross  company  runtime  correctyear \
0         43  5500000.0  39754601.0     935     95.0          0
1         43 18000000.0  23402427.0     331    129.0          0
2         43 10000000.0  15795189.0    1182    100.0          0
3         43 15000000.0  70687344.0    1133    116.0          0
4         34  5000000.0  30031783.0     270    109.0          4
...
6007     43 90000000.0  426505244.0    383    124.0         40
6008     43 85000000.0  319715683.0    935    99.0         40
6009     43 175000000.0  245487753.0   1186   101.0         40
6010      6 135000000.0  111105497.0     10    100.0         40
6011      8  80000000.0  461421559.0    190   149.0         40

      correctmonth  profitability
0                  8      71.281093
1                  11      0.300135
2                  4      0.579519
3                  11      3.712490
4                  9      5.006357
...
6007                 0      3.738947
6008                 1      2.761361
6009                 0      0.402787
6010                 1     -0.176996
6011                 7      4.767769
```

[4114 rows x 17 columns]

### Looking at the highest correlation

```
[95]: corr_mat = df_numerized.corr()
corr_pairs = corr_mat.unstack()
```

```

# Sorting the correlation pairs
sorted_pairs = corr_pairs.sort_values()

# Including only pairs that have a correlation greater than 0.5.
# A correlation coefficient of .50 or larger represents a strong or large
# correlation.
high_corr = sorted_pairs[(sorted_pairs) > 0.5]

high_corr

```

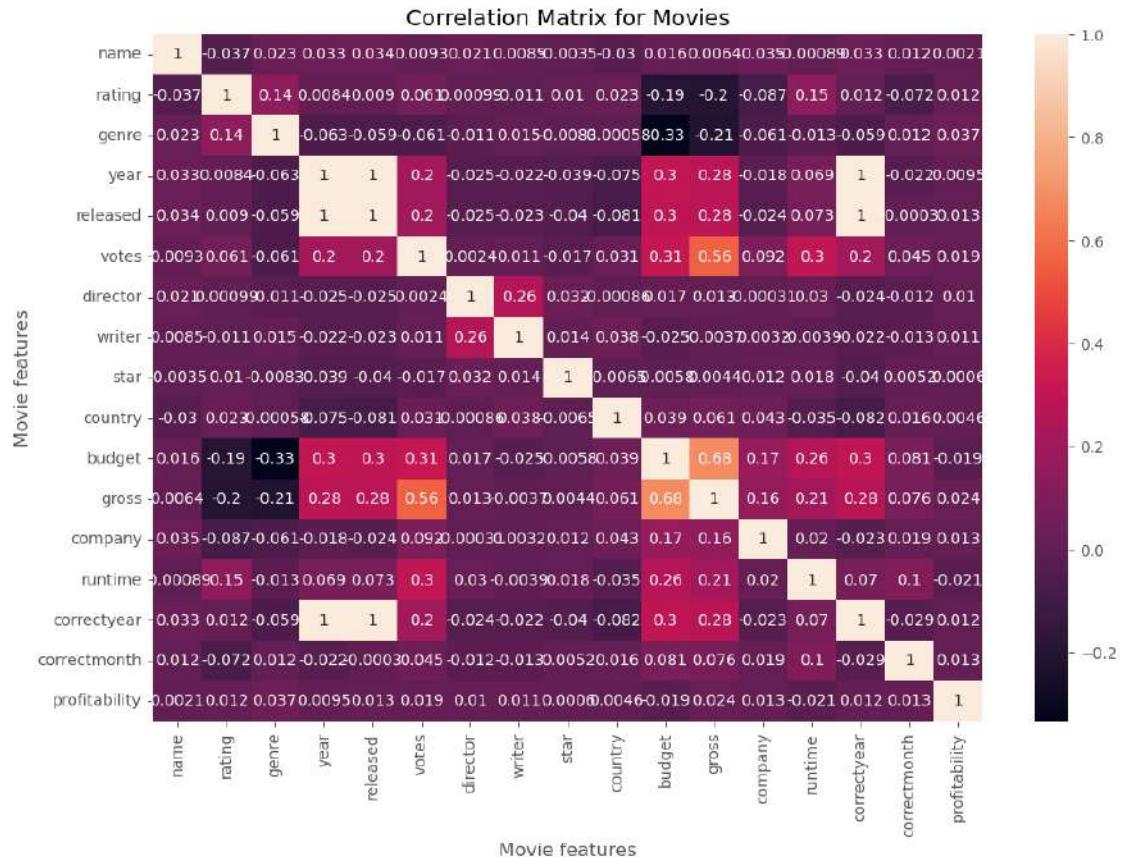
```
[95]: gross           votes          0.557999
       votes           gross          0.557999
       budget          gross          0.682367
       gross           budget          0.682367
       year            released         0.997724
       released         year           0.997724
       correctyear      correctyear    0.998626
       correctyear      released        0.998626
       correctyear      year           0.998858
       year             correctyear    0.998858
       name             name           1.000000
       star             star           1.000000
       correctyear      correctyear    1.000000
       runtime          runtime         1.000000
       company          company         1.000000
       gross            gross           1.000000
       budget           budget          1.000000
       country          country         1.000000
       writer           writer          1.000000
       director         director        1.000000
       votes            votes           1.000000
       released         released        1.000000
       year             year           1.000000
       genre            genre           1.000000
       rating           rating          1.000000
       correctmonth     correctmonth    1.000000
       profitability    profitability   1.000000
       dtype: float64
```

### Visualizing the data

```
[96]: correlation_matrix = df_numerized.corr(method = 'pearson')

sns.heatmap(correlation_matrix, annot = True)
plt.title("Correlation Matrix for Movies")
plt.xlabel("Movie features")
plt.ylabel("Movie features")
```

```
plt.show()
```



Conclusion: The factors contributing to the success of a movie are votes and budget as they have the highest correlation. It was also determined that the Company, Movie name, Genre had no correlation.

## 1.5 Data Exploration Cont'd

While we have determined the factors contributing the most to a movie's success. There are still a few other factors we can explore.

What is the best time of year to release a movie?

What is the ideal runtime for a movie?

What genre of movie had the most success on average?

What genre of movie has the most success overall?

### 1.5.1 What is the best time of year to release a movie?

*By grouping the Gross Earnings by Month, we can answer this question.*

## Tabulating the data

```
[32]: df.groupby(['correctmonth']).mean()
```

```
[32]:      year      votes      budget      gross \
correctmonth
01      2002.330337  87933.896629  2.708138e+07  6.206133e+07
02      2001.828729  101584.546961  3.077599e+07  7.637523e+07
03      2001.750000  107112.392857  3.332767e+07  9.301620e+07
04      2002.091644  95545.517520  3.020464e+07  8.801838e+07
05      2001.805263  147799.647368  4.731544e+07  1.589717e+08
06      2001.769231  119046.853147  3.606901e+07  9.214667e+07
07      2001.324138  103716.510345  3.599994e+07  9.246908e+07
08      2001.997727  84682.472727  2.626852e+07  6.090950e+07
09      2001.320513  99197.269231  3.316988e+07  8.316971e+07
10      2001.800866  105209.038961  2.838029e+07  6.429625e+07
11      2001.417808  119286.984018  4.050880e+07  1.206020e+08
12      2001.500000  143560.812236  4.549368e+07  1.426989e+08

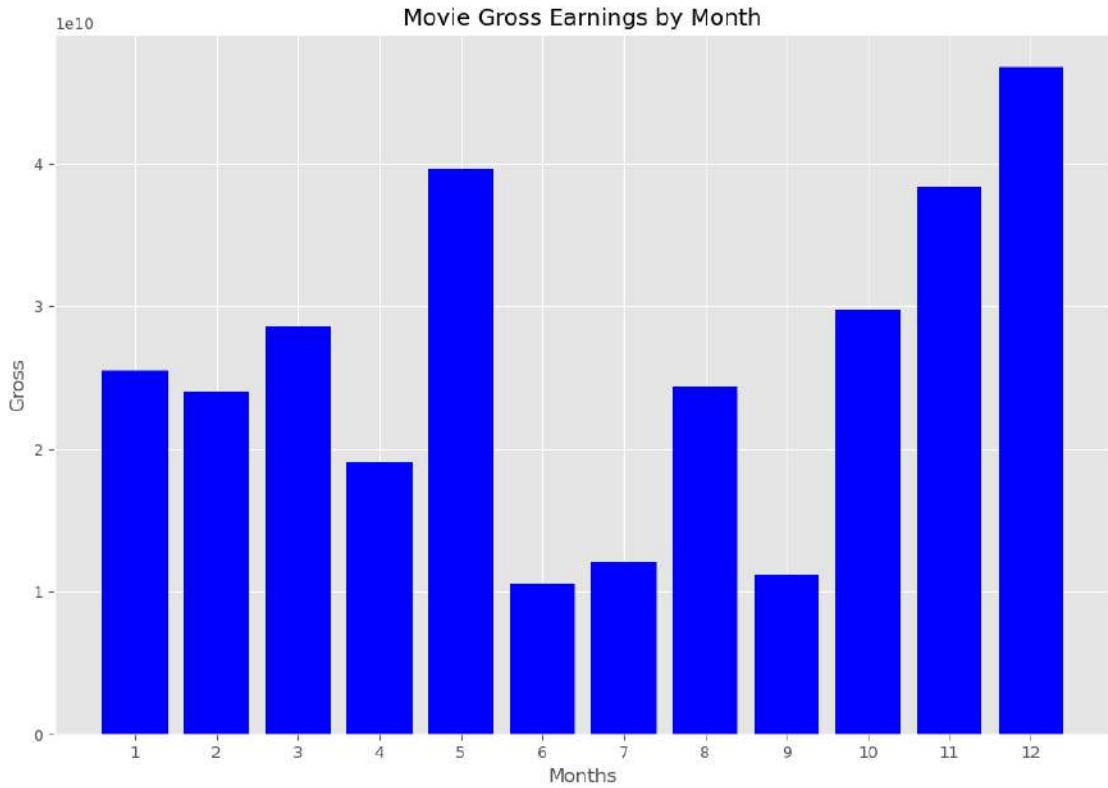
      runtime
correctmonth
01      106.851685
02      105.883978
03      106.206633
04      105.824798
05      109.202632
06      110.293706
07      108.268966
08      105.088636
09      107.096154
10      109.225108
11      108.757991
12      115.715190
```

## Visualizing the data

```
[97]: months = range(1,13)
print(months)

plt.bar(months,df.groupby(['correctmonth']).sum()['gross'], color='blue')
plt.xticks(months)
plt.title('Movie Gross Earnings by Month')
plt.ylabel('Gross')
plt.xlabel('Months')
plt.show()
```

```
range(1, 13)
```



Conclusion: The best time of year to release a movie is December based on the overall gross earnings in this period.

### 1.5.2 What is the ideal runtime for a movie?

#### Tabulating the data

```
[86]: gross_earnings_avg = df.groupby(['runtime'])['gross'].mean()
gross_earnings_avg.sort_values()
```

```
[86]: runtime
209.0      968853.0
242.0     4770222.0
171.0     8064706.5
219.0     8204229.5
192.0    10284493.0
...
149.0    415375961.0
181.0    424208848.0
183.0    449220945.0
163.0    497409852.0
187.0    562363449.0
Name: gross, Length: 121, dtype: float64
```

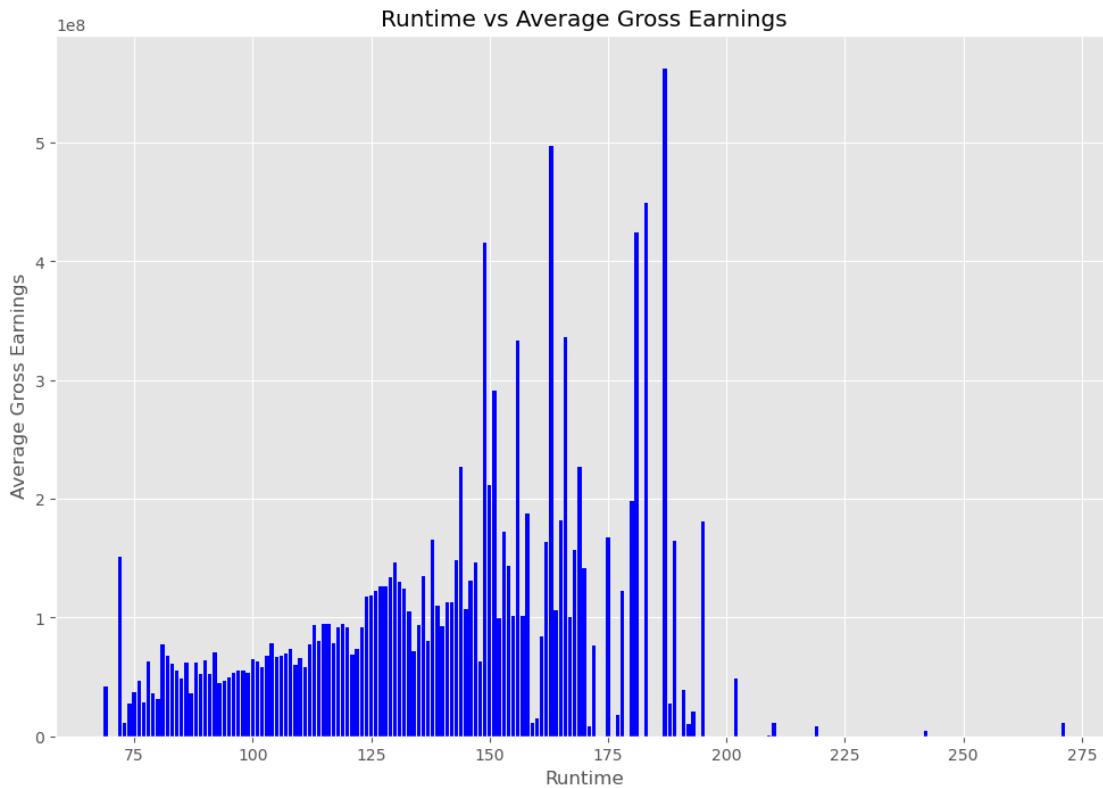
### Visualizing the data

```
[82]: gross_earnings_avg = df.groupby(['runtime'])['gross'].mean()

plt.bar(gross_earnings_avg.index, gross_earnings_avg, color="blue")

plt.xlabel('Runtime')
plt.ylabel('Average Gross Earnings')
plt.title('Runtime vs Average Gross Earnings')

plt.show()
```



Conclusion: The ideal runtime seems to be between 160 to 190 minutes.

### 1.5.3 What genre of movie has the most success overall? What genre of movie had the most success on average?

### Visualizing the data

```
[57]: # Get the sum and mean of gross earnings for each genre
grouped_df = df.groupby(['genre'])['gross'].agg(['sum', 'mean'])
genres = grouped_df.index.tolist()
sum_gross = grouped_df['sum'].tolist()
mean_gross = grouped_df['mean'].tolist()
```

```

# Set up the bar chart
fig, ax1 = plt.subplots()

# Plot the bar chart for sum gross
color = 'tab:blue'
ax1.bar(genres, sum_gross, color=color, label='Sum Gross')

# Add labels and title
ax1.set_xlabel('Genre')
ax1.set_ylabel('Sum Gross', color=color)
ax1.tick_params(axis='y', labelcolor=color)
ax1.set_title('Movie Gross Earnings by Genre')
plt.xticks(rotation=90)

# Create a second y-axis for mean gross and coefficient of variation
ax2 = ax1.twinx()

# Plot the bar chart for mean gross and coefficient of variation
color = 'tab:orange'
ax2.bar(genres, mean_gross, color=color, alpha=0.5, label='Mean Gross')
#ax2.plot(genres, cv_gross, color='tab:red', marker='o', label='Coefficient of Variation')

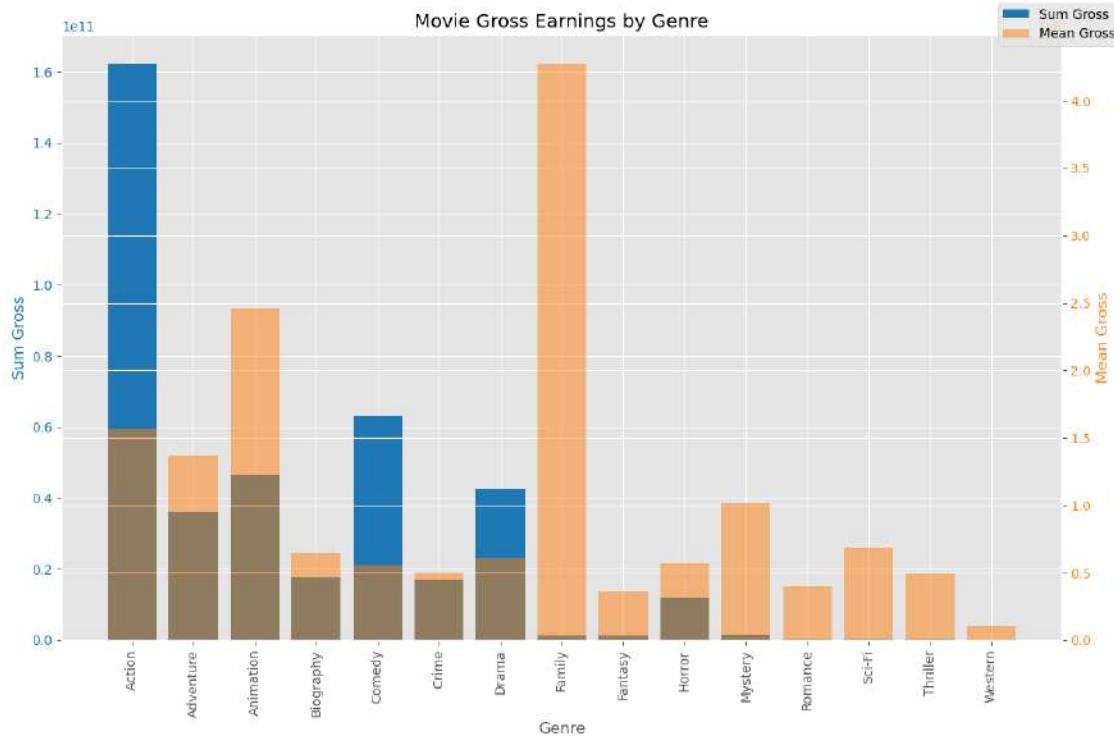
# Add labels and title
ax2.set_ylabel('Mean Gross', color=color)
ax2.tick_params(axis='y', labelcolor=color)

# Add a legend
fig.legend()

# Adjust layout
fig.tight_layout()

plt.show()

```



Conclusion: The Action genre had the highest level of success across all movies, whereas the Family genre had the highest average success rate.

## 1.6 Conclusion:

My analysis revealed that the most significant factors contributing to a movie's success were the production budget, the movie's budget, and the votes. Additionally, certain genres, such as action and family, tended to be more successful than others.

Overall, my Python data analysis project provided valuable insights into the factors contributing to the success of a movie.

**project**

**2**

# tech-store-sales-analysis

February 24, 2024

```
# Tech Store Sales Analysis
```

## 0.1 Importing Necessary Libraries

```
[32]: import os
import pandas as pd
import matplotlib.pyplot as plt
```

Merging the data from each month into one file

```
[25]: path = r"C:\Users\parri\Downloads\SalesData"
files = [file for file in os.listdir(path) if not file.startswith('.')] # Ignore hidden files

all_months_data = pd.DataFrame()

for file in files:
    current_data = pd.read_csv(path+"/"+file)
    all_months_data = pd.concat([all_months_data, current_data])

all_months_data.to_csv("all_data.csv", index=False)
```

Reading in updated dataframe

```
[26]: all_data = pd.read_csv("all_data.csv")
all_data.head()
```

```
[26]:   Order ID          Product  Quantity Ordered  Price Each  \
0    176558  USB-C Charging Cable           2        11.95
1      NaN            NaN           NaN        NaN
2    176559  Bose SoundSport Headphones       1        99.99
3    176560            Google Phone         1        600
4    176560        Wired Headphones         1        11.99

          Order Date          Purchase Address
0  04/19/19 08:46  917 1st St, Dallas, TX 75001
1            NaN            NaN
2  04/07/19 22:30  682 Chestnut St, Boston, MA 02215
```

```
3 04/12/19 14:38 669 Spruce St, Los Angeles, CA 90001
4 04/12/19 14:38 669 Spruce St, Los Angeles, CA 90001
```

## 0.2 Cleaning up the data

```
[27]: # Finding and dropping rows with NaN
```

```
nan_df = all_data[all_data.isna().any(axis=1)]
display(nan_df.head())

all_data = all_data.dropna(how='all')
all_data.head()
```

	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address
1	NaN	NaN	NaN	NaN	NaN	NaN
356	NaN	NaN	NaN	NaN	NaN	NaN
735	NaN	NaN	NaN	NaN	NaN	NaN
1433	NaN	NaN	NaN	NaN	NaN	NaN
1553	NaN	NaN	NaN	NaN	NaN	NaN

```
[27]: Order ID          Product  Quantity Ordered Price Each \
0    176558      USB-C Charging Cable           2        11.95
2    176559  Bose SoundSport Headphones           1       99.99
3    176560            Google Phone             1        600
4    176560            Wired Headphones           1        11.99
5    176561            Wired Headphones           1        11.99
```

	Order Date	Purchase Address
0	04/19/19 08:46	917 1st St, Dallas, TX 75001
2	04/07/19 22:30	682 Chestnut St, Boston, MA 02215
3	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001
4	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001
5	04/30/19 09:27	333 8th St, Los Angeles, CA 90001

```
[28]: # Checking if the NaN were successfully dropped
```

```
nan_df = all_data[all_data.isna().any(axis=1)]
display(nan_df.head())
```

```
Empty DataFrame
Columns: [Order ID, Product, Quantity Ordered, Price Each, Order Date, Purchase
           Address]
Index: []
```

```
[ ]:
```

```
[29]: # Removing text in order date column
```

```
all_data = all_data[all_data['Order Date'].str[0:2]!='Or']
```

```
[30]: # Converting columns to the correct type
```

```
all_data['Quantity Ordered'] = pd.to_numeric(all_data['Quantity Ordered'])
all_data['Price Each'] = pd.to_numeric(all_data['Price Each'])
```

## Augmenting data with additional columns

```
[31]: # Adding month column
```

```
all_data['Month'] = all_data['Order Date'].str[0:2]
all_data['Month'] = all_data['Month'].astype('int32')

all_data.head()
```

```
Order ID          Product  Quantity Ordered  Price Each \
0    176558      USB-C Charging Cable           2        11.95
2    176559  Bose SoundSport Headphones           1       99.99
3    176560            Google Phone             1      600.00
4    176560        Wired Headphones             1        11.99
5    176561        Wired Headphones             1        11.99

Order Date          Purchase Address  Month
0  04/19/19 08:46      917 1st St, Dallas, TX 75001     4
2  04/07/19 22:30      682 Chestnut St, Boston, MA 02215     4
3  04/12/19 14:38      669 Spruce St, Los Angeles, CA 90001     4
4  04/12/19 14:38      669 Spruce St, Los Angeles, CA 90001     4
5  04/30/19 09:27      333 8th St, Los Angeles, CA 90001     4
```

```
[33]: # Adding city column
```

```
def get_city(address):
    return address.split(",")[1].strip(" ")

def get_state(address):
    return address.split(",")[-2].split(" ")[1]

all_data['City'] = all_data['Purchase Address'].apply(lambda x: f'{get_city(x)}\n{get_state(x)}')
all_data.head()
```

```
Order ID          Product  Quantity Ordered  Price Each \
0    176558      USB-C Charging Cable           2        11.95
2    176559  Bose SoundSport Headphones           1       99.99
```

```

3   176560           Google Phone      1       600.00
4   176560           Wired Headphones  1       11.99
5   176561           Wired Headphones  1       11.99

          Order Date             Purchase Address Month \
0  04/19/19 08:46      917 1st St, Dallas, TX 75001    4
2  04/07/19 22:30      682 Chestnut St, Boston, MA 02215    4
3  04/12/19 14:38      669 Spruce St, Los Angeles, CA 90001    4
4  04/12/19 14:38      669 Spruce St, Los Angeles, CA 90001    4
5  04/30/19 09:27      333 8th St, Los Angeles, CA 90001    4

          City
0     Dallas (TX)
2     Boston (MA)
3  Los Angeles (CA)
4  Los Angeles (CA)
5  Los Angeles (CA)

```

### 0.3 Data Exploration!

#### 0.3.1 What was the best month for sales? How much was earned that month?

```
[36]: # Adding sales column

all_data['Sales'] = all_data['Quantity Ordered'].astype('int') * ↴
                  all_data['Price Each'].astype('float')

all_data.head()
```

```
[36]:   Order ID          Product  Quantity Ordered  Price Each \
0   176558  USB-C Charging Cable            2        11.95
2   176559  Bose SoundSport Headphones      1       99.99
3   176560           Google Phone      1       600.00
4   176560           Wired Headphones     1       11.99
5   176561           Wired Headphones     1       11.99

          Order Date             Purchase Address Month \
0  04/19/19 08:46      917 1st St, Dallas, TX 75001    4
2  04/07/19 22:30      682 Chestnut St, Boston, MA 02215    4
3  04/12/19 14:38      669 Spruce St, Los Angeles, CA 90001    4
4  04/12/19 14:38      669 Spruce St, Los Angeles, CA 90001    4
5  04/30/19 09:27      333 8th St, Los Angeles, CA 90001    4

          City  Sales
0     Dallas (TX)  23.90
2     Boston (MA)  99.99
3  Los Angeles (CA) 600.00
```

```
4 Los Angeles (CA) 11.99
5 Los Angeles (CA) 11.99
```

```
[35]: all_data.groupby(['Month']).sum()
```

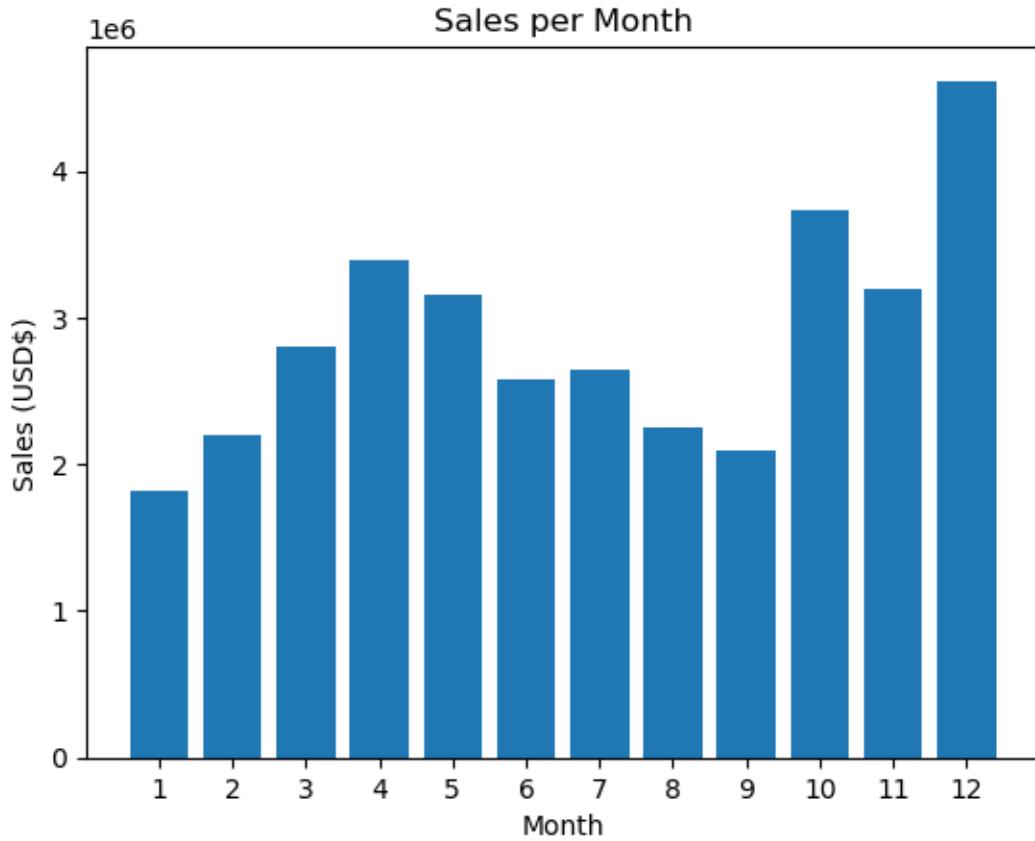
```
[35]:      Quantity Ordered  Price Each        Sales
Month
1                  10903  1811768.38  1822256.73
2                  13449  2188884.72  2202022.42
3                  17005  2791207.83  2807100.38
4                  20558  3367671.02  3390670.24
5                  18667  3135125.13  3152606.75
6                  15253  2562025.61  2577802.26
7                  16072  2632539.56  2647775.76
8                  13448  2230345.42  2244467.88
9                  13109  2084992.09  2097560.13
10                 22703  3715554.83  3736726.88
11                 19798  3180600.68  3199603.20
12                 28114  4588415.41  4613443.34
```

```
[42]: # Visualizing the data for Sales per Month
```

```
months = range(1,13)
print(months)

plt.bar(months,all_data.groupby(['Month']).sum()['Sales'])
plt.xticks(months)
plt.title('Sales per Month')
plt.ylabel('Sales (USD$)')
plt.xlabel('Month')
plt.show()
```

```
range(1, 13)
```



The best month for sales was December where they made a total of \$4,613,443.34.

### 0.3.2 What city sold the most product?

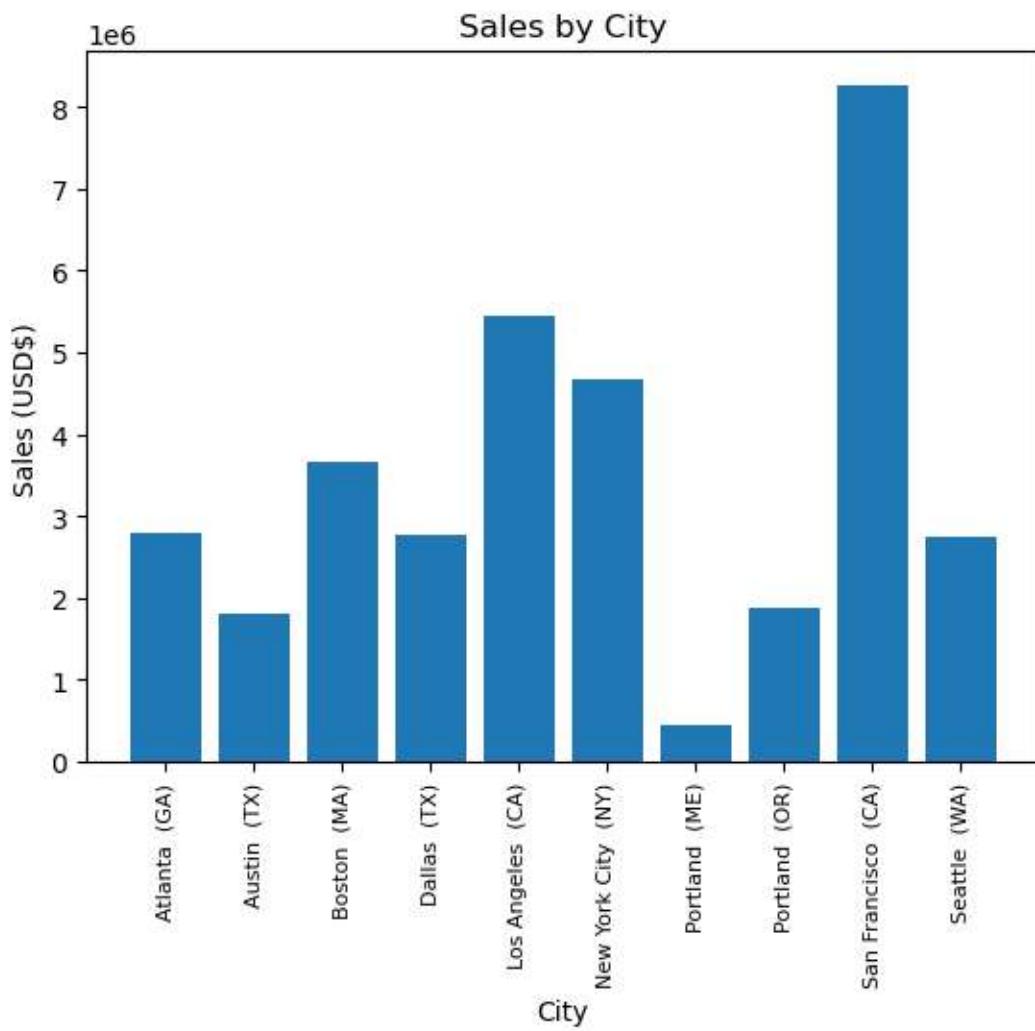
```
[43]: all_data.groupby(['City']).sum()
```

City	Quantity Ordered	Price Each	Month	Sales
Atlanta (GA)	16602	2779908.20	104794	2795498.58
Austin (TX)	11153	1809873.61	69829	1819581.75
Boston (MA)	22528	3637409.77	141112	3661642.01
Dallas (TX)	16730	2752627.82	104620	2767975.40
Los Angeles (CA)	33289	5421435.23	208325	5452570.80
New York City (NY)	27932	4635370.83	175741	4664317.43
Portland (ME)	2750	447189.25	17144	449758.27
Portland (OR)	11303	1860558.22	70621	1870732.34
San Francisco (CA)	50239	8211461.74	315520	8262203.91
Seattle (WA)	16553	2733296.01	104941	2747755.48

```
[46]: # Visualizing the data for Sales per city
```

```
keys = [city for city, df in all_data.groupby(['City'])]

plt.bar(keys,all_data.groupby(['City']).sum()['Sales'])
plt.title('Sales by City')
plt.ylabel('Sales (USD$)')
plt.xlabel('City')
plt.xticks(keys, rotation='vertical', size=8)
plt.show()
```



The city that sold the most products was San Francisco (CA).

### 0.3.3 What time should we display advertisements to maximize likelihood of customer's buying product?

```
[48]: # Adding time columns
```

```
all_data['Hour'] = pd.to_datetime(all_data['Order Date']).dt.hour  
all_data['Minute'] = pd.to_datetime(all_data['Order Date']).dt.minute  
all_data['Count'] = 1  
all_data.head()
```

```
[48]:   Order ID          Product  Quantity Ordered  Price Each  \
```

```
0    176558      USB-C Charging Cable           2       11.95  
2    176559  Bose SoundSport Headphones         1      99.99  
3    176560            Google Phone           1     600.00  
4    176560      Wired Headphones             1       11.99  
5    176561      Wired Headphones             1       11.99
```

```
          Order Date          Purchase Address Month  \
```

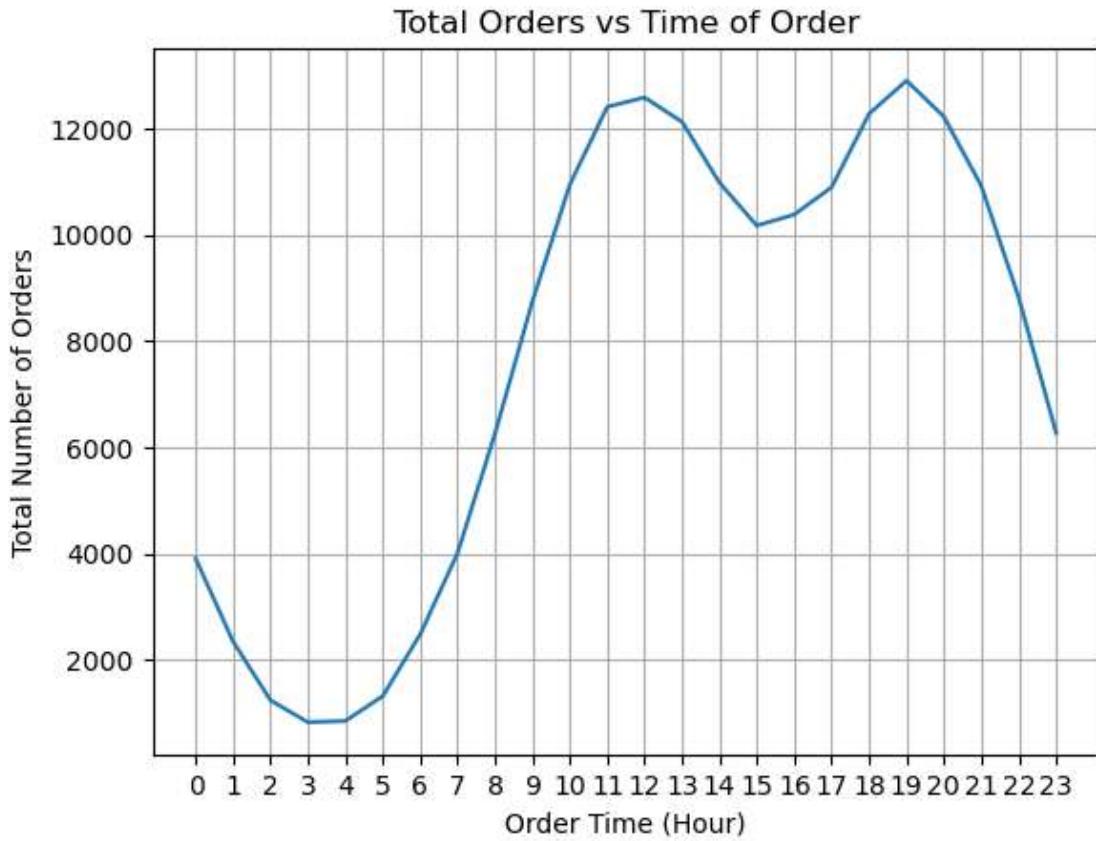
```
0  04/19/19 08:46      917 1st St, Dallas, TX 75001      4  
2  04/07/19 22:30      682 Chestnut St, Boston, MA 02215      4  
3  04/12/19 14:38      669 Spruce St, Los Angeles, CA 90001      4  
4  04/12/19 14:38      669 Spruce St, Los Angeles, CA 90001      4  
5  04/30/19 09:27      333 8th St, Los Angeles, CA 90001      4
```

```
          City  Sales  Hour  Minute  Count
```

```
0    Dallas (TX)  23.90     8      46      1  
2    Boston (MA)  99.99    22      30      1  
3  Los Angeles (CA)  600.00    14      38      1  
4  Los Angeles (CA)  11.99    14      38      1  
5  Los Angeles (CA)  11.99     9      27      1
```

```
[54]: # Using the order time from customers' orders to plot a graph
```

```
keys = [pair for pair, df in all_data.groupby(['Hour'])]  
  
plt.plot(keys, all_data.groupby(['Hour']).count()['Count'])  
plt.xticks(keys)  
plt.title('Total Orders vs Time of Order')  
plt.ylabel('Total Number of Orders')  
plt.xlabel('Order Time (Hour)')  
plt.grid()  
plt.show()
```

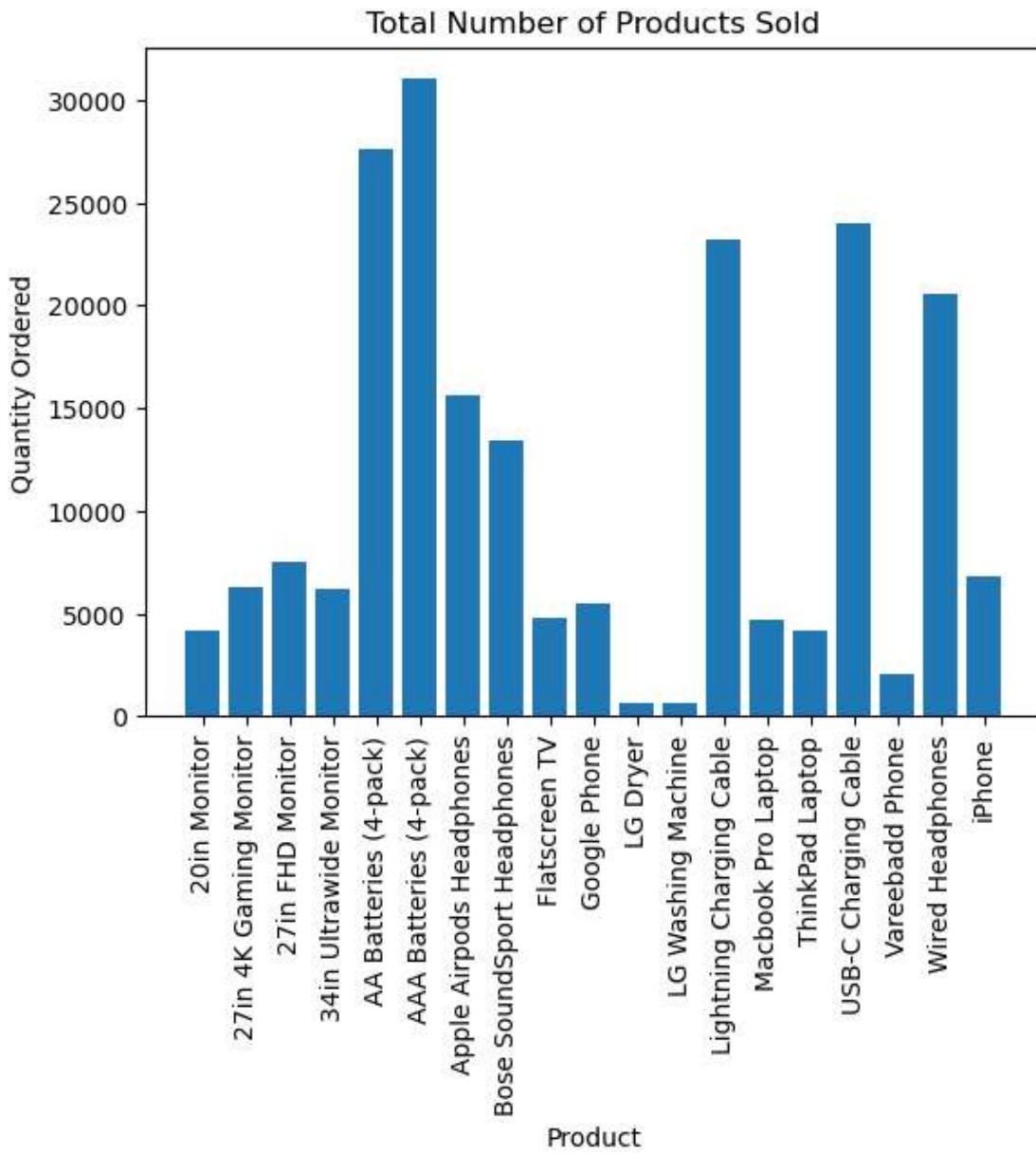


The best time for ads to be aired would be 11AM-12PM or around 7PM.

#### 0.3.4 What product sold the most? Why do you think it sold the most?

```
[72]: product_group = all_data.groupby('Product')
quantity_ordered = product_group.sum()['Quantity Ordered']

keys = [pair for pair, df in product_group]
plt.bar(keys, quantity_ordered)
plt.xticks(keys, rotation='vertical', size=10)
plt.title('Total Number of Products Sold')
plt.ylabel('Quantity Ordered')
plt.xlabel('Product')
plt.show()
```



The product sold the most were AAA Batteries. This may be due to a lot of electronics using them as a power source and its cost in comparison to the other items.

[78]: # Comparing cost of items to the quantity sold.

```
prices = all_data.groupby('Product').mean()['Price Each']

fig, ax1 = plt.subplots()

ax2 = ax1.twinx()
```

```

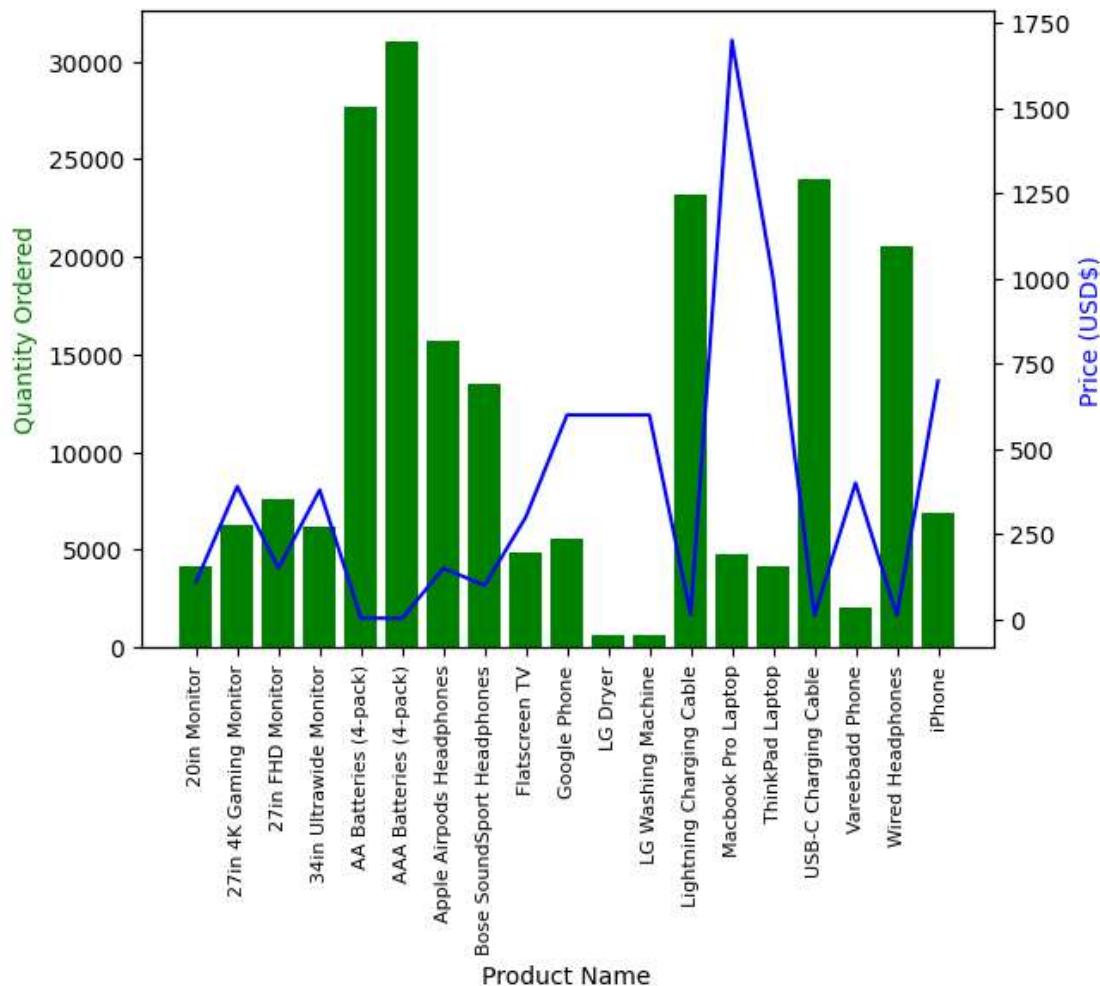
ax1.bar(keys, quantity_ordered, color='g')
ax2.plot(keys, prices, color='b')

ax1.set_xlabel('Product Name')
ax1.set_ylabel('Quantity Ordered', color='g')
ax2.set_ylabel('Price (USD$)', color='b')
ax1.set_xticklabels(keys, rotation='vertical', size=8)

fig.show()

```

C:\Users\parri\AppData\Local\Temp\ipykernel\_20272\205174232.py:14: UserWarning:  
FixedFormatter should only be used together with FixedLocator  
    ax1.set\_xticklabels(keys, rotation='vertical', size=8)  
C:\Users\parri\AppData\Local\Temp\ipykernel\_20272\205174232.py:16: UserWarning:  
Matplotlib is currently using module://matplotlib\_inline.backend\_inline, which  
is a non-GUI backend, so cannot show the figure.  
fig.show()



[ ]:

project

3

# legendary-pok195169mon-analysis

February 24, 2024

## 1 What Makes a Pokémon Legendary?

### 1.1 1. Introduction

In the world of Pokémon academia, one name towers above any other – Professor Samuel Oak. While his colleague Professor Elm specializes in Pokémon evolution, Oak has dedicated his career to understanding the relationship between Pokémon and their human trainers. A former trainer himself, the professor has first-hand experience of how obstinate Pokémon can be – particularly when they hold legendary status.

For his latest research project, Professor Oak has decided to investigate the defining characteristics of legendary Pokémon to improve our understanding of their temperament. Hearing of our expertise in classification problems, he has enlisted us as the lead researchers.

Our journey begins at the professor's research lab in Pallet Town, Kanto. The first step is to open up the Pokédex, an encyclopaedic guide to 801 Pokémon from all seven generations.

Source: bagogames on Flickr

```
[82]: # Load the tidyverse
library(tidyverse)

# Import the dataset and convert variables
pokedex <- read_csv("datasets/pokedex.csv",
                     col_types = cols(name = col_factor(),
                                      type = col_factor(),
                                      is_legendary = col_factor()))

# Look at the first six rows
head(pokedex)

# Examine the structure
str(pokedex)
```

pokedex_number	name	attack	defense	height_m	hp	percentage_male	sp_attack	sp_defense
1	Bulbasaur	49	49	0.7	45	88.1	65	65
2	Ivysaur	62	63	1.0	60	88.1	80	80
3	Venusaur	100	123	2.0	80	88.1	122	120
4	Charmander	52	43	0.6	39	88.1	60	50
5	Charmeleon	64	58	1.1	58	88.1	80	65
6	Charizard	104	78	1.7	78	88.1	159	115

```

spec_tbl_df [801 x 14] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
$ pokedex_number : num [1:801] 1 2 3 4 5 6 7 8 9 10 ...
$ name          : Factor w/ 801 levels "Bulbasaur","Ivysaur",...
8 9 10 ...
$ attack        : num [1:801] 49 62 100 52 64 104 48 63 103 30 ...
$ defense       : num [1:801] 49 63 123 43 58 78 65 80 120 35 ...
$ height_m      : num [1:801] 0.7 1 2 0.6 1.1 1.7 0.5 1 1.6 0.3 ...
$ hp            : num [1:801] 45 60 80 39 58 78 44 59 79 45 ...
$ percentage_male: num [1:801] 88.1 88.1 88.1 88.1 88.1 88.1 88.1 88.1 88.1 50
...
$ sp_attack     : num [1:801] 65 80 122 60 80 159 50 65 135 20 ...
$ sp_defense    : num [1:801] 65 80 120 50 65 115 64 80 115 20 ...
$ speed         : num [1:801] 45 60 80 65 80 100 43 58 78 45 ...
$ type          : Factor w/ 18 levels "grass","fire",...
8 9 10 ...
$ weight_kg     : num [1:801] 6.9 13 100 8.5 19 90.5 9 22.5 85.5 2.9 ...
$ generation    : num [1:801] 1 1 1 1 1 1 1 1 1 1 ...
$ is_legendary   : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
- attr(*, "spec")=
.. cols(
..   pokedex_number = col_double(),
..   name = col_factor(levels = NULL, ordered = FALSE, include_na = FALSE),
..   attack = col_double(),
..   defense = col_double(),
..   height_m = col_double(),
..   hp = col_double(),
..   percentage_male = col_double(),
..   sp_attack = col_double(),
..   sp_defense = col_double(),
..   speed = col_double(),
..   type = col_factor(levels = NULL, ordered = FALSE, include_na = FALSE),
..   weight_kg = col_double(),
..   generation = col_double(),
..   is_legendary = col_factor(levels = NULL, ordered = FALSE, include_na =
FALSE)
.. )
- attr(*, "problems")=<externalptr>

```

## 1.2 2. How many Pokémon are legendary?

After browsing the Pokédex, we can see several variables that could feasibly explain what makes a Pokémon legendary. We have a series of numerical fighter stats – attack, defense, speed and so on – as well as a categorization of Pokémon type (bug, dark, dragon, etc.). `is_legendary` is the binary classification variable we will eventually be predicting, tagged 1 if a Pokémon is legendary and 0 if it is not.

Before we explore these variables in any depth, let's find out how many Pokémon are legendary out of the 801 total, using the handy `count()` function from the `dplyr` package.

```
[84]: # Prepare the data
legendaries_pokemon <- pokedex %>%
  count(is_legendary) %>%
  mutate(prop = n / nrow(pokedex))

# Print the data frame
legendaries_pokemon
```

is_legendary	n	prop
0	731	0.91260924
1	70	0.08739076

## 1.3 3. Legendary Pokémon by height and weight

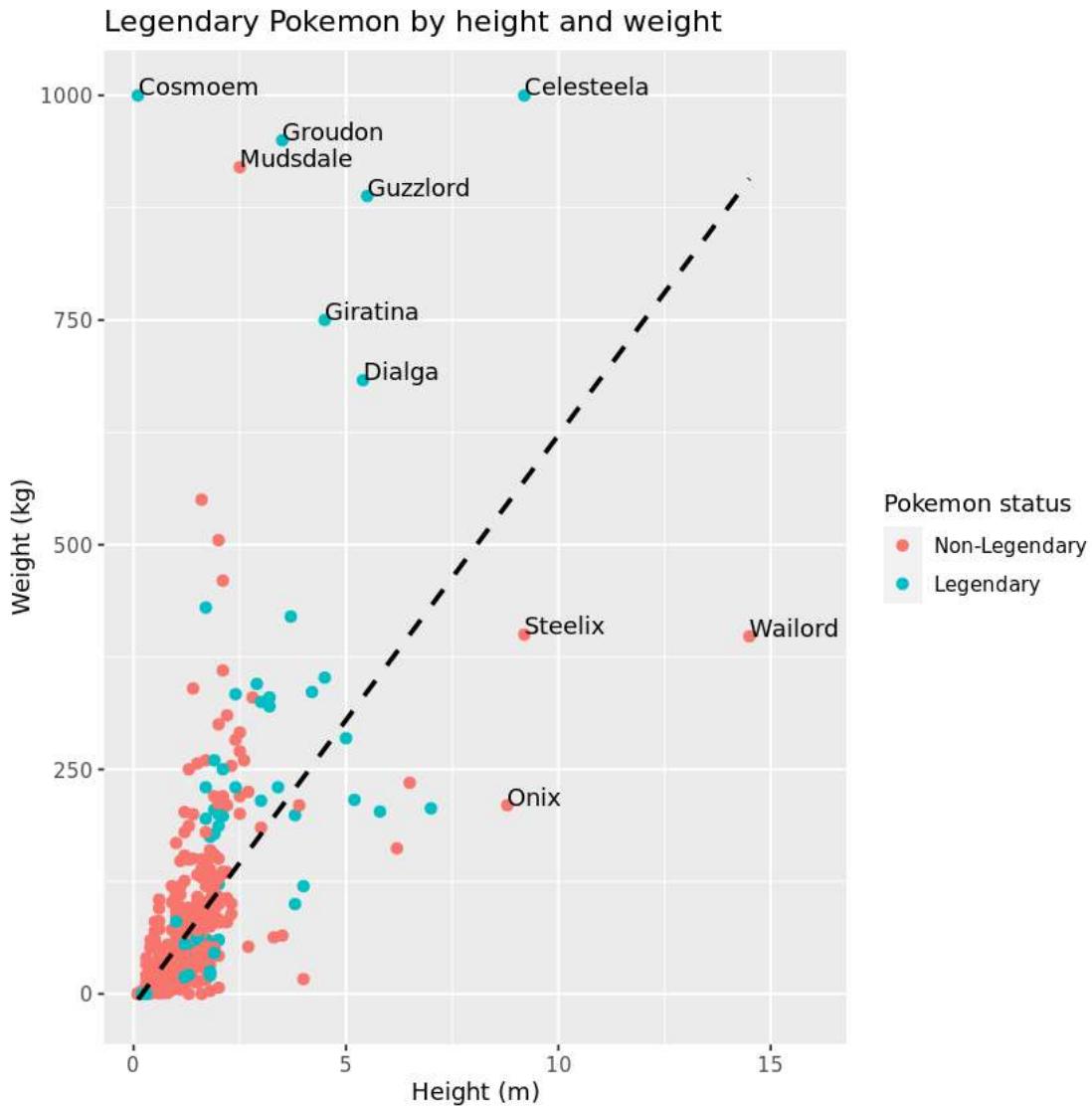
We now know that there are 70 legendary Pokémon – a sizable minority at 9% of the population! Let's start to explore some of their distinguishing characteristics.

First of all, we'll plot the relationship between `height_m` and `weight_kg` for all 801 Pokémon, highlighting those that are classified as legendary. We'll also add conditional labels to the plot, which will only print a Pokémon's name if it is taller than 7.5m or heavier than 600kg.

```
[86]: # Prepare the plot
legend_by_heightweight_plot <- pokedex %>%
  ggplot(aes(x = height_m, y = weight_kg)) +
  geom_point(aes(color = is_legendary), size = 2) +
  geom_text(aes(label = ifelse(height_m > 7.5 | weight_kg > 600, as.character(name), '')),
            vjust = 0, hjust = 0) +
  geom_smooth(method = "lm", se = FALSE, col = "black", linetype = "dashed") +
  expand_limits(x = 16) +
  labs(title = "Legendary Pokémon by height and weight",
       x = "Height (m)",
       y = "Weight (kg)") +
  guides(color = guide_legend(title = "Pokémon status")) +
  scale_color_manual(labels = c("Non-Legendary", "Legendary"),
                     values = c("#F8766D", "#00BFC4"))

# Print the plot
legend_by_heightweight_plot
```

```
`geom_smooth()` using formula 'y ~ x'
Warning message:
"Removed 20 rows containing non-finite values (stat_smooth)." Warning message:
"Removed 20 rows containing missing values (geom_point)." Warning message:
"Removed 20 rows containing missing values (geom_text)."
```



#### 1.4 4. Legendary Pokémon by type

It seems that legendary Pokémon are generally heavier and taller, but with many exceptions. For example, Onix (Gen 1), Steelix (Gen 2) and Wailord (Gen 3) are all extremely tall, but none of them have legendary status. There must be other factors at play.

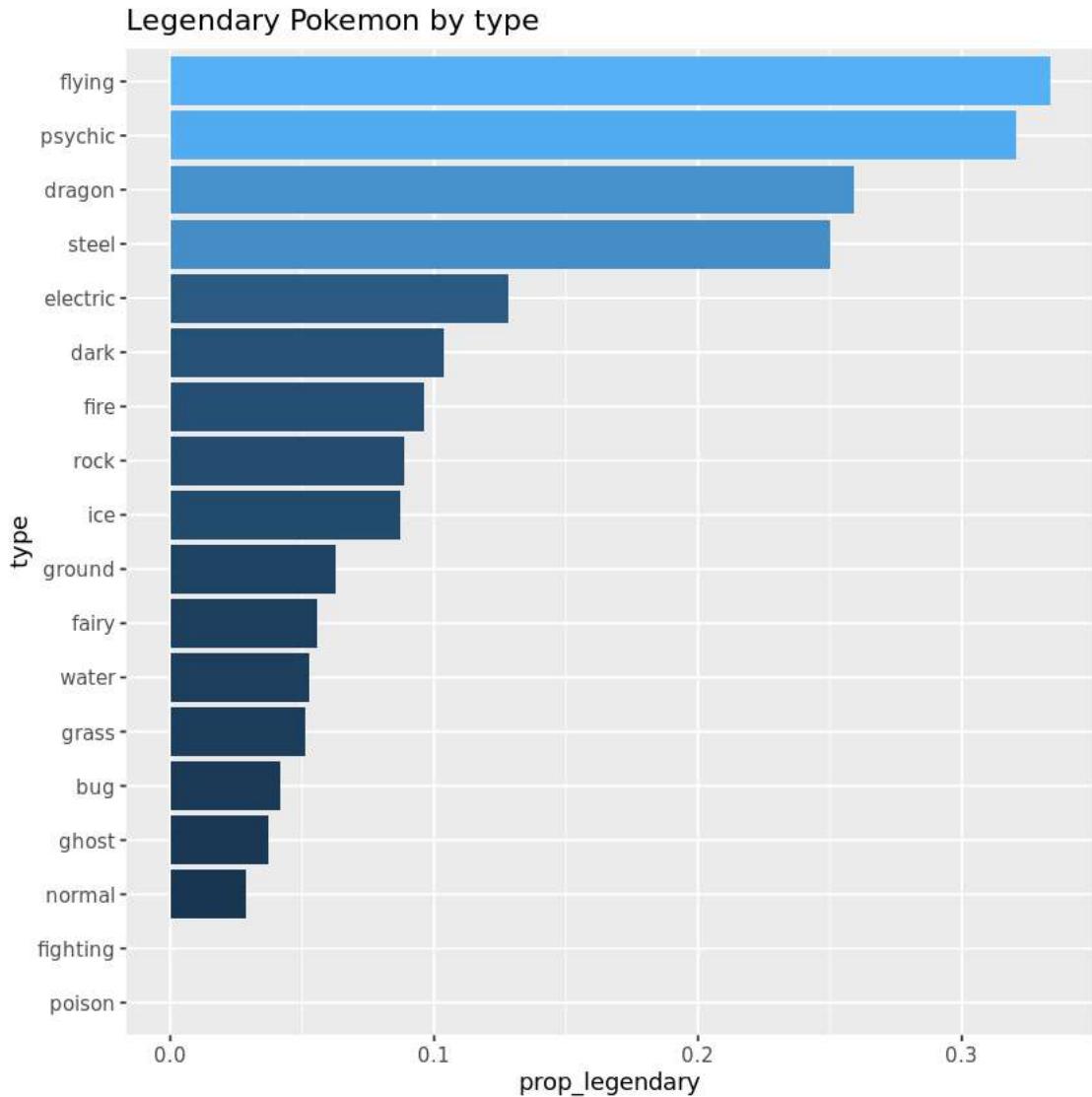
We will now look at the effect of a Pokémon's type on its legendary/non-legends classification. There are 18 possible types, ranging from the common (Grass / Normal / Water) to the rare (Fairy

/ Flying / Ice). We will calculate the proportion of legendary Pokémon within each category, and then plot these proportions using a simple bar chart.

```
[88]: # Prepare the data
legend_by_type <- pokedex %>%
  group_by(type) %>%
  mutate(is_legendary = as.numeric(is_legendary) - 1) %>%
  summarise(prop_legendary = mean(is_legendary)) %>%
  ungroup() %>%
  mutate(type = fct_reorder(type, prop_legendary))

# Prepare the plot
legend_by_type_plot <- legend_by_type %>%
  ggplot(aes(x = type, y = prop_legendary, fill = prop_legendary)) +
  geom_col() +
  labs(title = "Legendary Pokemon by type") +
  coord_flip() +
  guides(fill = "none")

# Print the plot
legend_by_type_plot
```



## 1.5 5. Legendary Pokémon by fighter stats

There are clear differences between Pokémon types in their relation to legendary status. While more than 30% of flying and psychic Pokémon are legendary, there is no such thing as a legendary poison or fighting Pokémon!

Before fitting the model, we will consider the influence of a Pokémon's fighter stats (attack, defense, etc.) on its status. Rather than considering each stat in isolation, we will produce a boxplot for all of them simultaneously using the `facet_wrap()` function.

```
[90]: # Prepare the data
legend_by_stats <- pokedex %>%
  select(is_legendary, attack, sp_attack, defense, sp_defense, hp, speed) %>%
```

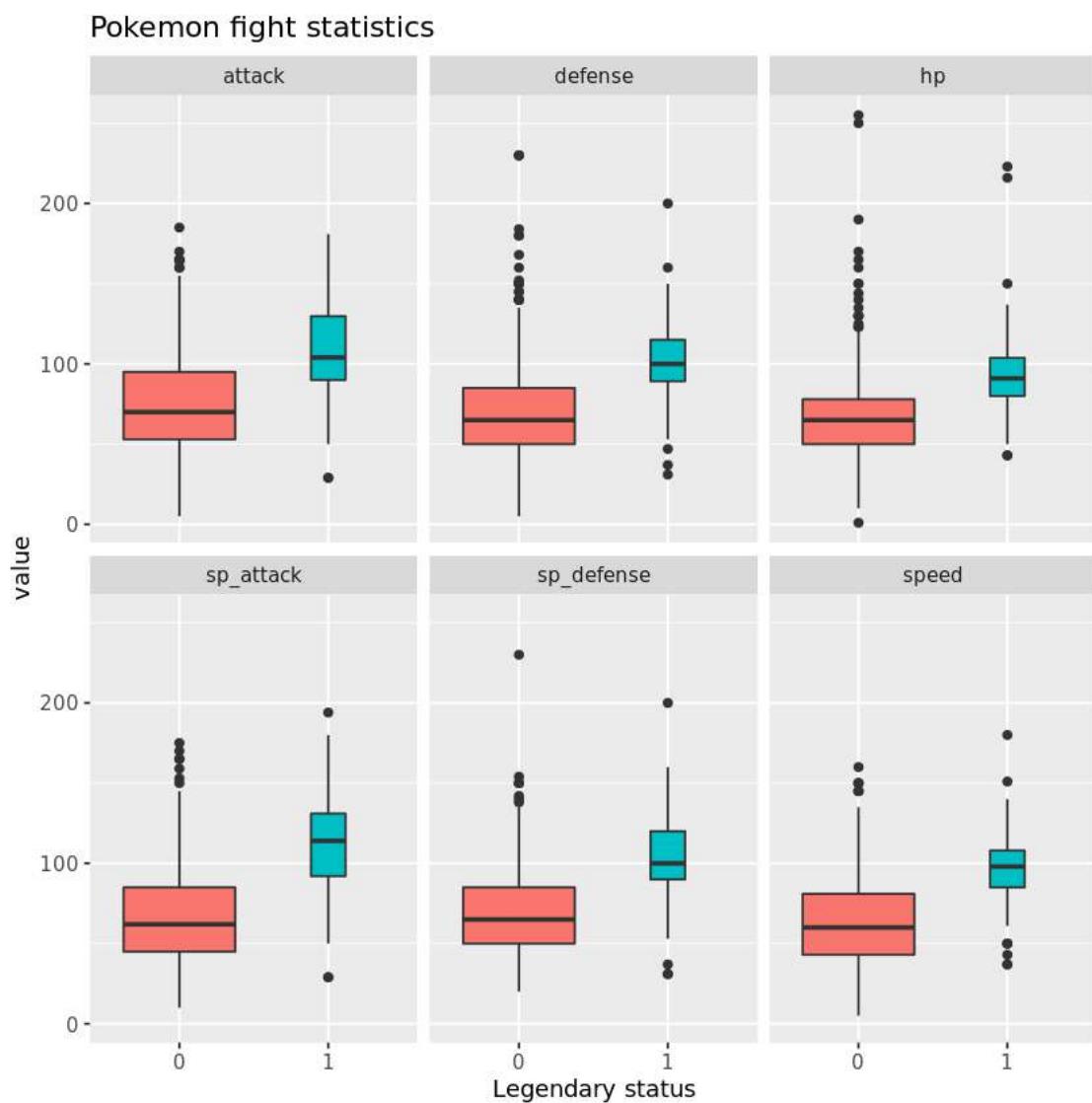
```

gather(key = fght_stats, value = value, -is_legendary)

# Prepare the plot
legend_by_stats_plot <- legend_by_stats %>%
  ggplot(aes(x = is_legendary, y = value, fill = is_legendary)) +
  geom_boxplot(varwidth = TRUE) +
  facet_wrap(~fght_stats) +
  labs(title = "Pokemon fight statistics",
       x = "Legendary status") +
  guides(fill = "none")

# Print the plot
legend_by_stats_plot

```



## 1.6 6. Create a training/test split

As we might expect, legendary Pokémons outshine their ordinary counterparts in all fighter stats. Although we haven't formally tested a difference in means, the boxplots suggest a significant difference with respect to all six variables. Nonetheless, there are a number of outliers in each case, meaning that some legendary Pokémons are anomalously weak.

We have now explored all of the predictor variables we will use to explain what makes a Pokémon legendary. Before fitting our model, we will split the pokedex into a training set (pokedex\_train) and a test set (pokedex\_test). This will allow us to test the model on unseen data.

```
[92]: # Set seed for reproducibility
set.seed(1234)

# Save number of rows in dataset
n <- nrow(pokedex)

# Generate 60% sample of rows
sample_rows <- sample(n, 0.6*n)

# Create training set
pokedex_train <- pokedex %>%
  filter(row_number() %in% sample_rows)

# Create test set
pokedex_test <- pokedex %>%
  filter(!row_number() %in% sample_rows)
```

## 1.7 7. Fit a decision tree

Now we have our training and test sets, we can go about building our classifier. But before we fit a random forest, we will fit a simple classification decision tree. This will give us a baseline fit against which to compare the results of the random forest, as well as an informative graphical representation of the model.

Here, and also in the random forest, we will omit incomplete observations by setting the na.action argument to na.omit. This will remove a few Pokémons with missing values for height\_m and weight\_kg from the training set. Remember the warning messages when we made our height/weight plot in Task 3? These are the Pokémons to blame!

```
[94]: # Load packages and set seed
library(rpart)
library(rpart.plot)
set.seed(1234)

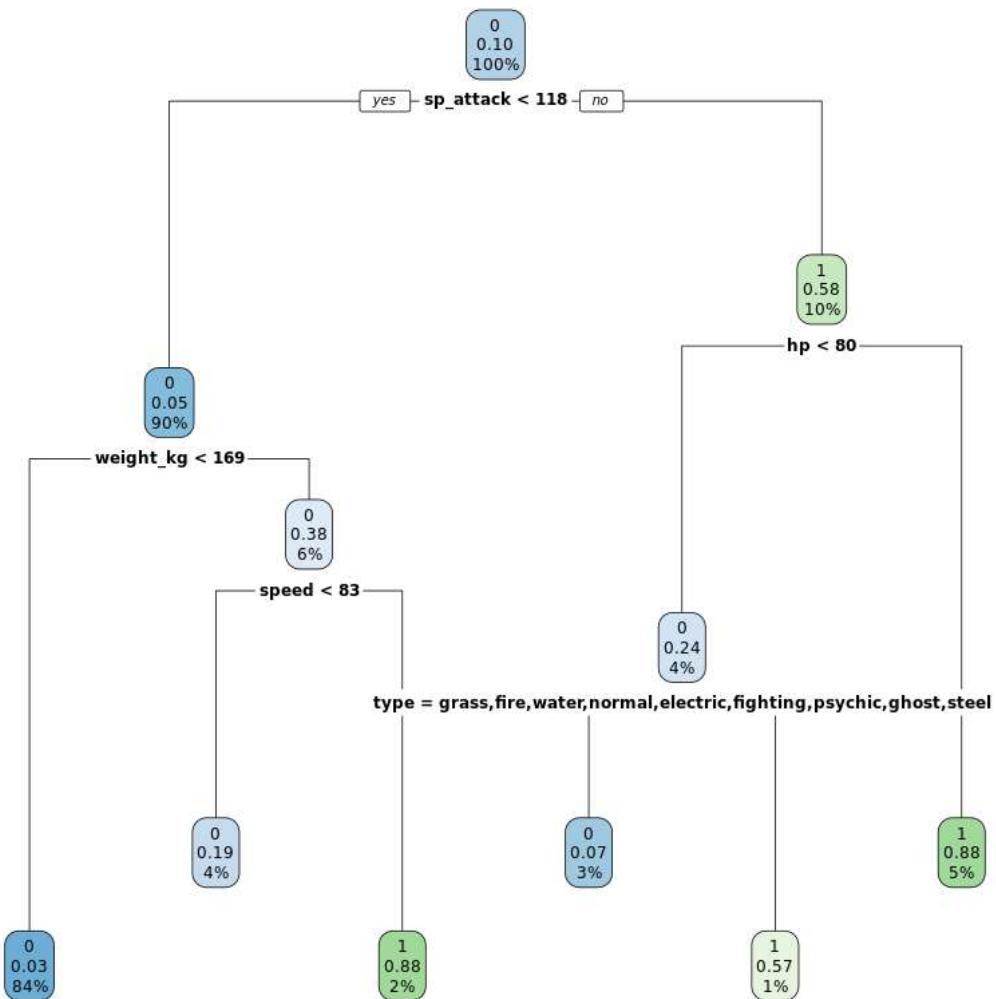
# Fit decision tree
model_tree <- rpart(is_legendary ~ attack + defense + height_m +
  hp + sp_attack + sp_defense + speed + type + weight_kg,
  data = pokedex_train,
```

```

method = "class",
na.action = na.omit)

# Plot decision tree
rpart.plot(model_tree)

```



## 1.8 8. Fit a random forest

Each node of the tree shows the predicted class, the probability of being legendary, and the percentage of Pokémon in that node. The bottom-left node, for example – for those with  $sp\_attack < 118$  and  $weight\_kg < 169$  – represents 84% of Pokémon in the training set, predicting that each only has a 3% chance of being legendary.

Decision trees place the most important variables at the top and exclude any they don't find to be

useful. In this case, sp\_attack occupies node 1 while attack, defense, sp\_defense and height\_m are all excluded.

However, decision trees are unstable and sensitive to small variations in the data. It therefore makes sense to fit a random forest – an ensemble method that averages over several decision trees all at once. This should give us a more robust model that classifies Pokémons with greater accuracy.

```
[96]: # Load package and set seed
library(randomForest)
set.seed(1234)

# Fit random forest
model_forest <- randomForest(is_legendary ~ attack + defense + height_m +
                                hp + sp_attack + sp_defense + speed + type + weight_kg,
                                data = pokedex_train,
                                importance = TRUE,
                                na.action = na.omit)

# Print model output
print(model_forest)
```

```
Call:
randomForest(formula = is_legendary ~ attack + defense + height_m +      hp +
sp_attack + sp_defense + speed + type + weight_kg, data = pokedex_train,
importance = TRUE, na.action = na.omit)
Type of random forest: classification
Number of trees: 500
No. of variables tried at each split: 3

OOB estimate of  error rate: 7.05%
Confusion matrix:
 0  1 class.error
0 411  9  0.02142857
1  24 24  0.50000000
```

## 1.9 9. Assess model fit

Looking at the model output, we can see that the random forest has an out-of-bag (OOB) error of 7.48%, which isn't bad by most accounts. However, since there are 24 true positives and 24 false negatives, the model only has a recall of 50%, which means that it struggles to successfully retrieve every legendary Pokémon in the dataset.

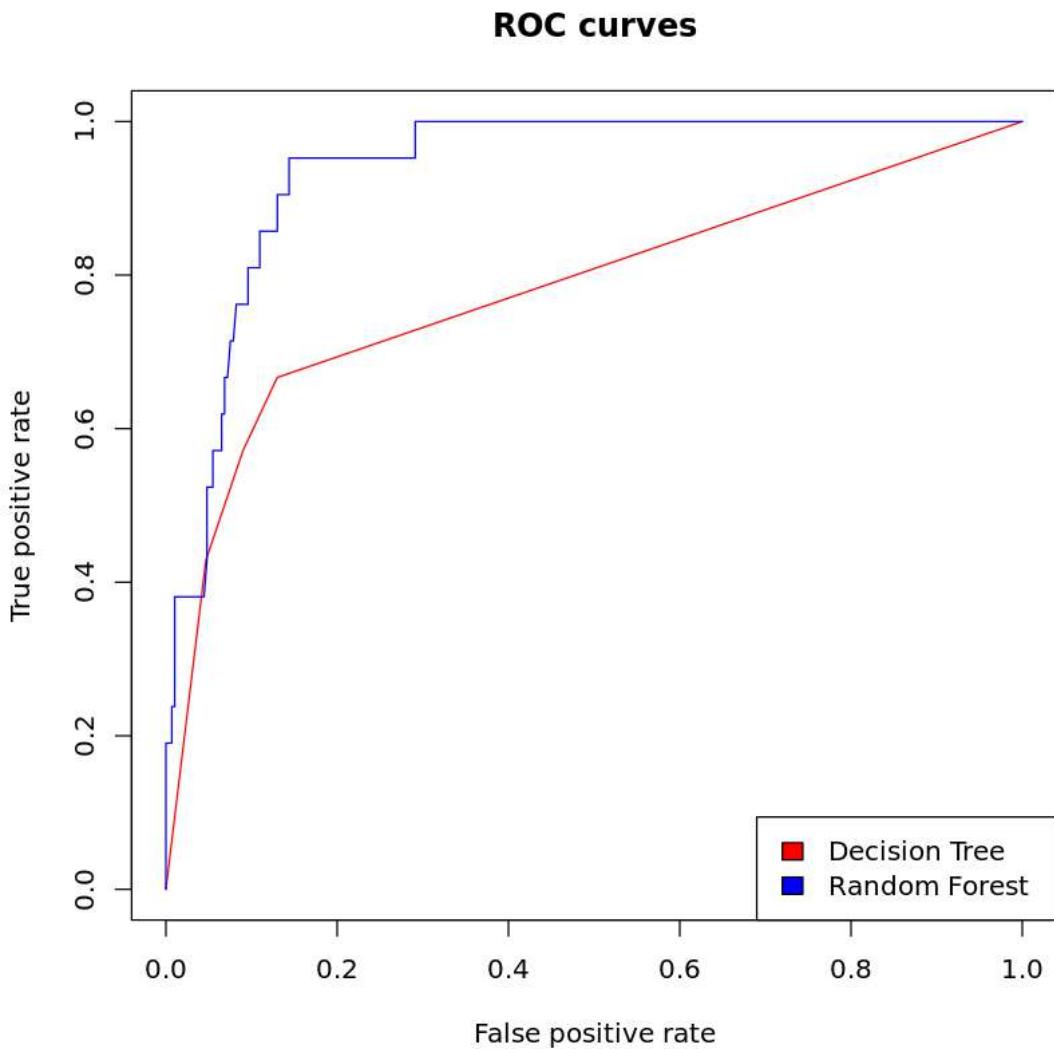
In order to allow direct comparison with the decision tree, we will plot the ROC curves for both models using the ROCR package, which will visualize their true positive rate (TPR) and false positive rate (FPR) respectively. The closer the curve is to the top left of the plot, the higher the area under the curve (AUC) and the better the model.

```
[98]: # Load the ROCR package
library(ROCR)

# Create prediction and performance objects for the decision tree
probs_tree <- predict(model_tree, pokedex_test, type = "prob")
pred_tree <- prediction(probs_tree[,2], pokedex_test$is_legendary)
perf_tree <- performance(pred_tree, "tpr", "fpr")

# Create prediction and performance objects for the random forest
probs_forest <- predict(model_forest, pokedex_test, type = "prob")
pred_forest <- prediction(probs_forest[,2], pokedex_test$is_legendary)
perf_forest <- performance(pred_forest, "tpr", "fpr")

# Plot the ROC curves: first for the decision tree, then for the random forest
plot(perf_tree, col = "red", main = "ROC curves")
plot(perf_forest, add = TRUE, col = "blue")
legend(x = "bottomright", legend = c("Decision Tree", "Random Forest"), fill = c("red", "blue"))
```



## 1.10 10. Analyze variable importance

It's clear from the ROC curves that the random forest is a substantially better model, boasting an AUC (not calculated above) of 91% versus the decision tree's 78%. When calculating variable importance, it makes sense to do so with the best model available, so we'll use the random forest for the final part of our analysis.

Note that a random forest returns two measures of variable importance:

MeanDecreaseAccuracy – how much the model accuracy suffers if you leave out a particular variable

MeanDecreaseGini – the degree to which a variable improves the probability of an observation being classified one way or another (i.e. ‘node purity’).

Together, these two measures will allow us to answer our original research question – what makes

a Pokémon legendary?

```
[100]: # Print variable importance measures
importance_forest <- importance(model_forest)
importance_forest

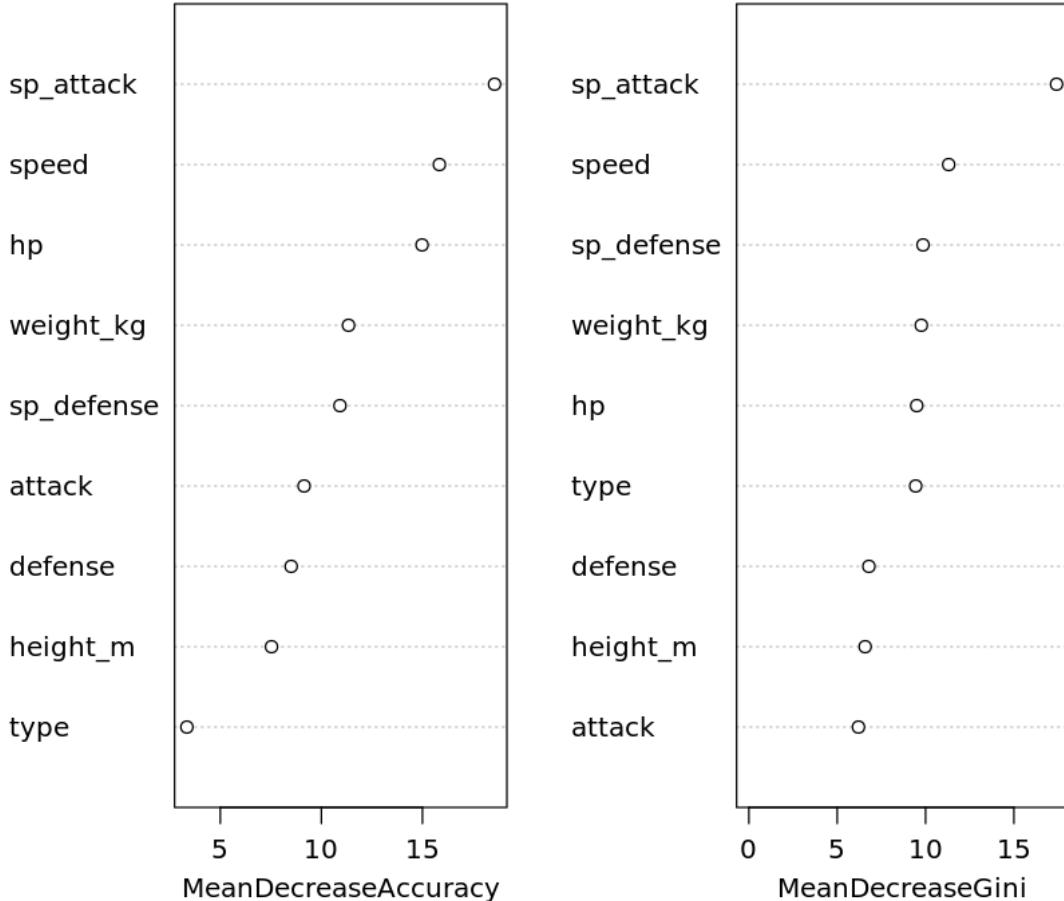
# Create a dotchart of variable importance
varImpPlot_forest <- varImpPlot(model_forest, main = "Variable Importance in
    ↵Random Forest")
varImpPlot_forest
```

	0	1	MeanDecreaseAccuracy	MeanDecreaseGini
attack	5.5880374	8.5275682	9.145018	6.203285
defense	0.6724663	12.2083778	8.506461	6.793236
height_m	4.1403952	6.7509476	7.533695	6.579863
hp	2.7180639	19.2406506	14.980599	9.499892
sp_attack	6.3947972	22.5417527	18.567857	17.415111
sp_defense	-0.8501697	15.1682650	10.919369	9.857089
speed	2.2799561	21.0596369	15.838444	11.308568
type	3.3473633	0.7726262	3.347259	9.440971
weight_kg	8.8857876	6.6917782	11.343187	9.760121

	MeanDecreaseAccuracy	MeanDecreaseGini
attack	9.145018	6.203285
defense	8.506461	6.793236
height_m	7.533695	6.579863
hp	14.980599	9.499892
sp_attack	18.567857	17.415111
sp_defense	10.919369	9.857089
speed	15.838444	11.308568
type	3.347259	9.440971
weight_kg	11.343187	9.760121

## Variable Importance in Random Forest



### 1.11 11. Conclusion

According to the variable importance plot, `sp_attack` is the most important factor in determining whether or not a Pokémon is legendary, followed by `speed`. The plot doesn't tell us whether the variables have a positive or a negative effect, but we know from our exploratory analysis that the relationship is generally positive. We therefore conclude that legendary Pokémon are characterized primarily by the power of their special attacks and secondarily by their speediness, while also exhibiting higher fighting abilities across the board.

Congratulations on completing your research into legendary Pokémon – Professor Oak is excited to share the findings! To finish, we'll answer a few of his questions about the variable importance results.

```
[102]: # According to the MeanDecreaseAccuracy plot:  
  
# Q1. Is the `attack` or `defense` variable more important?  
answer1 <- "attack"  
  
# Q2. Is the `weight_kg` or `height_m` variable more important?  
answer2 <- "weight_kg"  
  
# According to the MeanDecreaseGini plot:  
  
# Q3. Is the `attack` or `defense` variable more important?  
answer3 <- "defense"  
  
# Q4. Is the `weight_kg` or `height_m` variable more important?  
answer4 <- "weight_kg"
```



project

4

**Python Correlation Project**

# python-correlation-project

February 24, 2024

[1]: # Importing the libraries I will use for this project

```
import pandas as pd
import numpy as np
import seaborn as sns

import matplotlib.pyplot as plt
import matplotlib.mlab as mlab
import matplotlib
plt.style.use('ggplot')
from matplotlib.pyplot import figure

%matplotlib inline
matplotlib.rcParams['figure.figsize'] = (12,8)

pd.options.mode.chained_assignment = None
```

[2]: # Reading in the data

```
df = pd.read_csv(r'C:\Users\parri\Downloads\movies.csv')
```

[3]: # Looking at the data

```
df.head()
```

```
          name rating      genre    year   released    votes \
0     Friday the 13th      R    Horror  1980  1980-09-05  123000.0
1        Raging Bull      R  Biography  1980  1980-12-19  330000.0
2      The Long Riders      R  Biography  1980  1980-05-16   10000.0
3  Any Which Way You Can    PG    Action  1980  1980-12-17   18000.0
4  The Gods Must Be Crazy    PG Adventure  1980  1984-10-26   54000.0
```

```
          director        writer        star        country \
0  Sean S. Cunningham  Victor Miller  Betsy Palmer  United States
1    Martin Scorsese    Jake LaMotta  Robert De Niro  United States
2      Walter Hill     Bill Bryden  David Carradine  United States
3  Buddy Van Horn  Stanford Sherman  Clint Eastwood  United States
```

```

4           Jamie Uys           Jamie Uys          N!xau    South Africa
              budget      gross                  company   runtime
0     5500000.0  39754601.0        Paramount Pictures    95.0
1  18000000.0  23402427.0  Chartoff-Winkler Productions  129.0
2  10000000.0  15795189.0        United Artists   100.0
3  15000000.0  70687344.0        The Malpaso Company  116.0
4   5000000.0  30031783.0        C.A.T. Films    109.0

```

```

[4]: # Checking for missing data
# Let's loop through the data and see if there is anything missing

for col in df.columns:
    percent_missing = np.mean(df[col].isnull())
    print('{} - {}%'.format(col, round(percent_missing*100)))

```

```

name - 0%
rating - 1%
genre - 0%
year - 0%
released - 0%
votes - 0%
director - 0%
writer - 0%
star - 0%
country - 0%
budget - 29%
gross - 3%
company - 0%
runtime - 0%

```

```
[5]: # Cleaning the data
```

```
df = df.dropna()
```

```
[6]: # Dropping duplicates
```

```
df.drop_duplicates()
```

```

[6]:           name  rating  genre  year  released \
0     Friday the 13th      R  Horror  1980  1980-09-05
1       Raging Bull      R  Biography  1980  1980-12-19
2      The Long Riders      R  Biography  1980  1980-05-16
3  Any Which Way You Can    PG  Action  1980  1980-12-17
4  The Gods Must Be Crazy    PG Adventure  1980  1984-10-26
...
6007    Bad Boys for Life      R  Action  2020  2020-01-17

```

6008	Sonic the Hedgehog	PG	Action	2020	2020-02-14
6009	Dolittle	PG	Adventure	2020	2020-01-17
6010	The Call of the Wild	PG	Adventure	2020	2020-02-21
6011	The Eight Hundred	Not Rated	Action	2020	2020-08-28

	votes	director	writer	star	\
0	1230000.0	Sean S. Cunningham	Victor Miller	Betsy Palmer	
1	330000.0	Martin Scorsese	Jake LaMotta	Robert De Niro	
2	10000.0	Walter Hill	Bill Bryden	David Carradine	
3	18000.0	Buddy Van Horn	Stanford Sherman	Clint Eastwood	
4	54000.0	Jamie Uys	Jamie Uys	N!xau	
...	...	...	...	...	
6007	140000.0	Adil El Arbi	Peter Craig	Will Smith	
6008	102000.0	Jeff Fowler	Pat Casey	Ben Schwartz	
6009	53000.0	Stephen Gaghan	Stephen Gaghan	Robert Downey Jr.	
6010	42000.0	Chris Sanders	Michael Green	Harrison Ford	
6011	3700.0	Hu Guan	Hu Guan	Zhi-zhong Huang	

	country	budget	gross	\
0	United States	550000.0	39754601.0	
1	United States	18000000.0	23402427.0	
2	United States	10000000.0	15795189.0	
3	United States	15000000.0	70687344.0	
4	South Africa	5000000.0	30031783.0	
...	...	...	...	
6007	United States	90000000.0	426505244.0	
6008	United States	85000000.0	319715683.0	
6009	United States	175000000.0	245487753.0	
6010	Canada	135000000.0	111105497.0	
6011	China	80000000.0	461421559.0	

	company	runtime
0	Paramount Pictures	95.0
1	Chartoff-Winkler Productions	129.0
2	United Artists	100.0
3	The Malpaso Company	116.0
4	C.A.T. Films	109.0
...	...	...
6007	Columbia Pictures	124.0
6008	Paramount Pictures	99.0
6009	Universal Pictures	101.0
6010	20th Century Studios	100.0
6011	Beijing Diqi Yinxiang Entertainment	149.0

[4208 rows x 14 columns]

```
[7]: # Data Types for our columns

print(df.dtypes)
```

```
name      object
rating     object
genre      object
year       int64
released   object
votes      float64
director   object
writer     object
star       object
country    object
budget     float64
gross      float64
company    object
runtime    float64
dtype: object
```

```
[8]: # Changing the data type of columns
# Some of the year columns and released have different years, we will splicethe released date and use that year.
```

```
df['correctyear'] = df['released'].astype(str).str[:4]
```

```
df
```

	name	rating	genre	year	released	\
0	Friday the 13th	R	Horror	1980	1980-09-05	
1	Raging Bull	R	Biography	1980	1980-12-19	
2	The Long Riders	R	Biography	1980	1980-05-16	
3	Any Which Way You Can	PG	Action	1980	1980-12-17	
4	The Gods Must Be Crazy	PG	Adventure	1980	1984-10-26	
...	...	...	...	...	...	
6007	Bad Boys for Life	R	Action	2020	2020-01-17	
6008	Sonic the Hedgehog	PG	Action	2020	2020-02-14	
6009	Dolittle	PG	Adventure	2020	2020-01-17	
6010	The Call of the Wild	PG	Adventure	2020	2020-02-21	
6011	The Eight Hundred	Not Rated	Action	2020	2020-08-28	
	votes	director	writer	star	\	
0	123000.0	Sean S. Cunningham	Victor Miller	Betsy Palmer		
1	330000.0	Martin Scorsese	Jake LaMotta	Robert De Niro		
2	10000.0	Walter Hill	Bill Bryden	David Carradine		
3	18000.0	Buddy Van Horn	Stanford Sherman	Clint Eastwood		
4	54000.0	Jamie Uys	Jamie Uys	N!xau		

```

...
6007    ...          ...
       140000.0      Adil El Arbi      Peter Craig      Will Smith
6008    102000.0      Jeff Fowler      Pat Casey      Ben Schwartz
6009    53000.0       Stephen Gaghan  Stephen Gaghan  Robert Downey Jr.
6010    42000.0       Chris Sanders   Michael Green   Harrison Ford
6011    3700.0        Hu Guan        Hu Guan        Zhi-zhong Huang

           country      budget      gross  \
0     United States  550000.0  39754601.0
1     United States 18000000.0  23402427.0
2     United States 10000000.0  15795189.0
3     United States 15000000.0  70687344.0
4     South Africa  5000000.0  30031783.0
...
6007    ...          ...
       90000000.0  426505244.0
6008    United States 85000000.0  319715683.0
6009    United States 175000000.0 245487753.0
6010    Canada       135000000.0 111105497.0
6011    China        80000000.0  461421559.0

           company  runtime  correctyear
0            Paramount Pictures  95.0      1980
1  Chartoff-Winkler Productions 129.0      1980
2            United Artists  100.0      1980
3            The Malpaso Company 116.0      1980
4            C.A.T. Films  109.0      1984
...
6007            ...
       Columbia Pictures  124.0      ...
6008            Paramount Pictures  99.0      2020
6009            Universal Pictures 101.0      2020
6010            20th Century Studios 100.0      2020
6011  Beijing Diqi Yinxiang Entertainment 149.0      2020

```

[4208 rows x 15 columns]

[ ]:

[12]: # Ordering the Data

```
df.sort_values(by=['gross'], inplace=False, ascending=False)
```

[12]:

	name	rating	genre	year	\
4306	Avatar	PG-13	Action	2009	
5846	Avengers: Endgame	PG-13	Action	2019	
2404	Titanic	PG-13	Drama	1997	
5252	Star Wars: Episode VII - The Force Awakens	PG-13	Action	2015	
5691	Avengers: Infinity War	PG-13	Action	2018	

...														
567			Crimewave		PG-13	Comedy	1985							
4459			Tanner Hall		R	Drama	2009							
2912			Ginger Snaps	Not Rated		Drama	2000							
183			Parasite		R	Horror	1982							
2534			Trojan War	PG-13	Comedy	1997								
	released	votes		director				writer	\					
4306	2009-12-18	1100000.0		James Cameron				James Cameron						
5846	2019-04-26	903000.0		Anthony Russo				Christopher Markus						
2404	1997-12-19	1100000.0		James Cameron				James Cameron						
5252	2015-12-18	876000.0		J.J. Abrams				Lawrence Kasdan						
5691	2018-04-27	897000.0		Anthony Russo				Christopher Markus						
...	...	...	...	...	...	...	...	...						
567	1986-04-25	5300.0		Sam Raimi				Ethan Coen						
4459	2015-01-15	3500.0	Francesca Gregorini		Tatiana von Fürstenberg									
2912	2001-11-05	43000.0		John Fawcett				Karen Walton						
183	1982-12-03	2300.0		Charles Band				Alan J. Adler						
2534	1997-01-10	5800.0		George Huang				Andy Burg						
	star	country		budget				gross	\					
4306	Sam Worthington	United States		237000000.0				2.847246e+09						
5846	Robert Downey Jr.	United States		356000000.0				2.797501e+09						
2404	Leonardo DiCaprio	United States		200000000.0				2.201647e+09						
5252	Daisy Ridley	United States		245000000.0				2.069522e+09						
5691	Robert Downey Jr.	United States		321000000.0				2.048360e+09						
...	...	...	...	...	...	...	...	...						
567	Louise Lasser	United States		3000000.0				5.101000e+03						
4459	Rooney Mara	United States		3000000.0				5.073000e+03						
2912	Emily Perkins	Canada		5000000.0				2.554000e+03						
183	Robert Glaudini	United States		800000.0				2.270000e+03						
2534	Will Friedle	United States		15000000.0				3.090000e+02						
	company	runtime	correct	year										
4306	Twentieth Century Fox	162.0		2009										
5846	Marvel Studios	181.0		2019										
2404	Twentieth Century Fox	194.0		1997										
5252	Lucasfilm	138.0		2015										
5691	Marvel Studios	149.0		2018										
...	...	...	...	...										
567	Columbia Pictures	83.0		1986										
4459	Two Prong Lesson	96.0		2015										
2912	Copperheart Entertainment	108.0		2001										
183	Embassy Pictures	85.0		1982										
2534	Daybreak	85.0		1997										

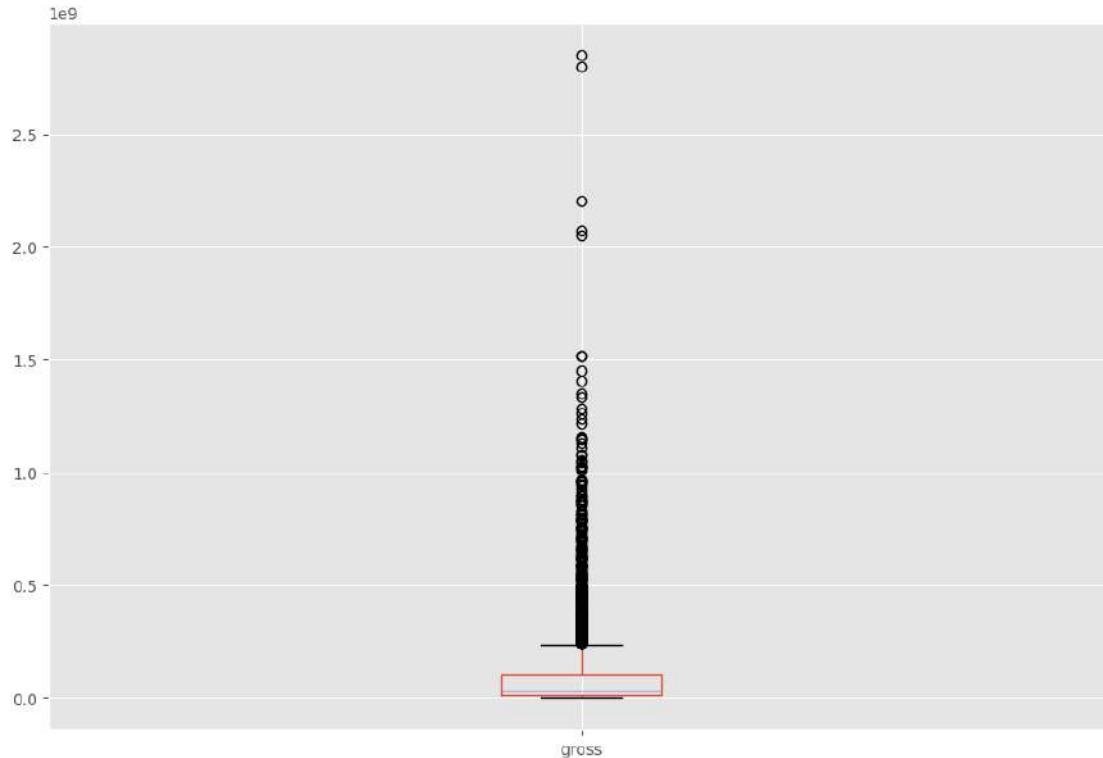
[4208 rows x 15 columns]

```
[ ]:
```

```
[13]: # Checking for Outliers
```

```
df.boxplot(column=['gross'])
```

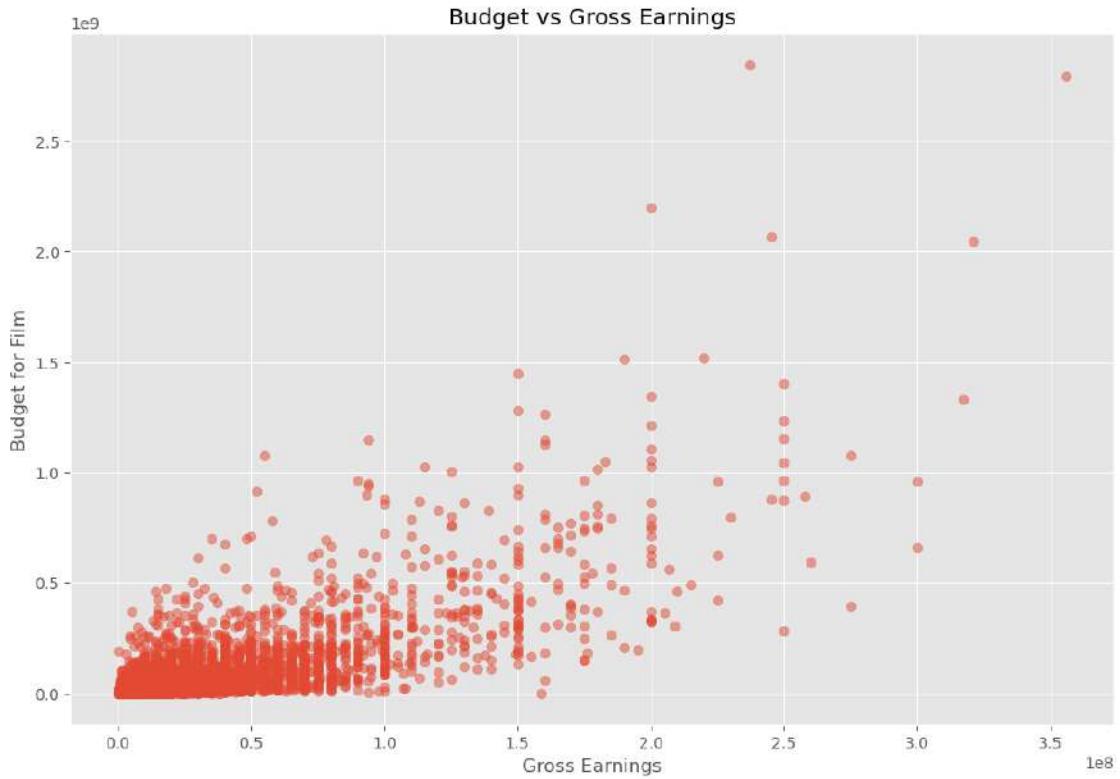
```
[13]: <AxesSubplot:>
```



```
[ ]:
```

```
[14]: # Looking at correlation
```

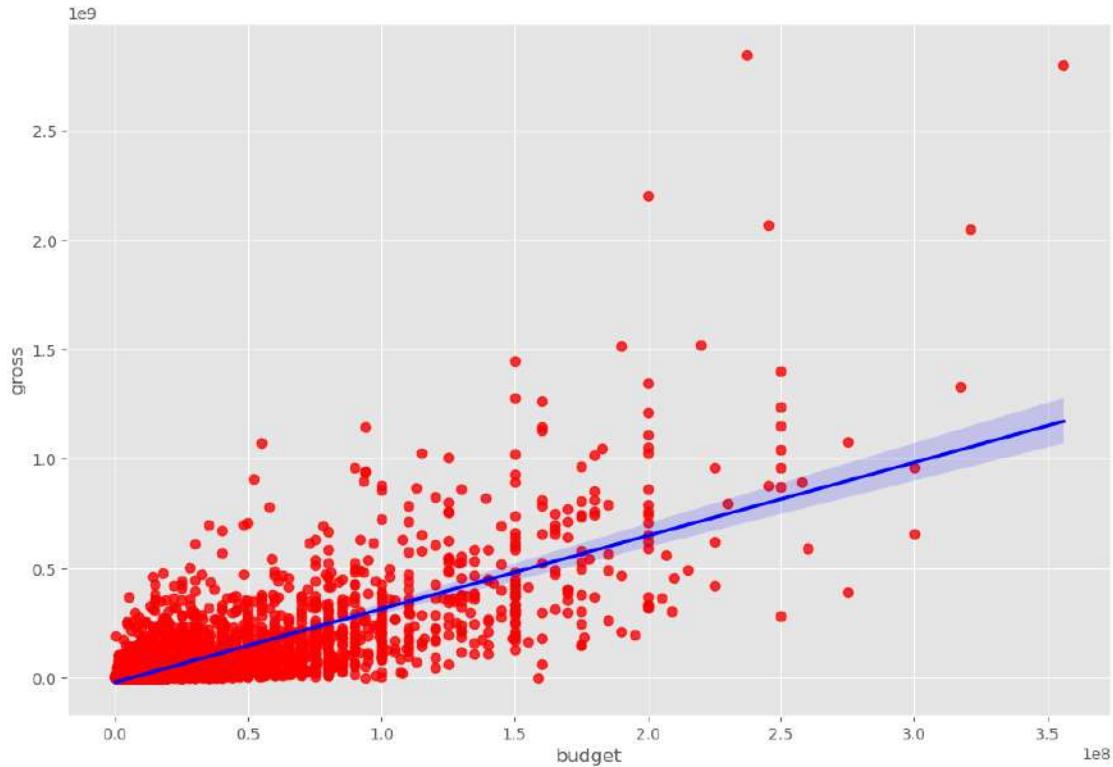
```
plt.scatter(x=df['budget'], y=df['gross'], alpha=0.5)
plt.title('Budget vs Gross Earnings')
plt.xlabel('Gross Earnings')
plt.ylabel('Budget for Film')
plt.show()
```



```
[15]: # Plot budget vs gross using seaborn
```

```
sns.regplot(x="budget", y="gross", data=df, scatter_kws = {"color": "red"}, line_kws = {"color": "blue"})
```

```
[15]: <AxesSubplot:xlabel='budget', ylabel='gross'>
```



```
[16]: # Correlation Matrix between all numeric columns
```

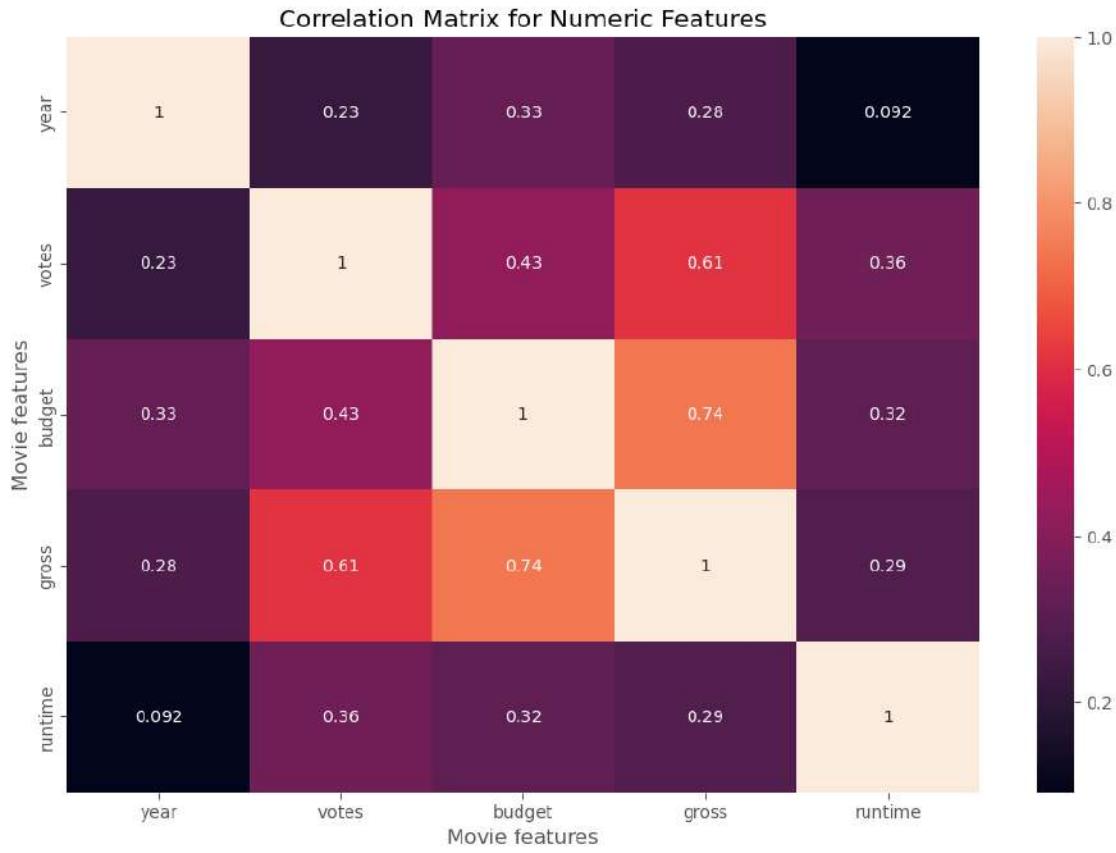
```
df.corr(method ='pearson')
```

```
[16]:
```

	year	votes	budget	gross	runtime
year	1.000000	0.227775	0.330178	0.278039	0.091519
votes	0.227775	1.000000	0.427663	0.612410	0.356359
budget	0.330178	0.427663	1.000000	0.741602	0.318132
gross	0.278039	0.612410	0.741602	1.000000	0.285549
runtime	0.091519	0.356359	0.318132	0.285549	1.000000

```
[17]:
```

```
correlation_matrix = df.corr()
sns.heatmap(correlation_matrix, annot = True)
plt.title("Correlation Matrix for Numeric Features")
plt.xlabel("Movie features")
plt.ylabel("Movie features")
plt.show()
```



```
[18]: # Updating all columns to numeric values
```

```
df_numerized = df

for col_name in df_numerized.columns:
    if(df_numerized[col_name].dtype == 'object'):
        df_numerized[col_name] = df_numerized[col_name].astype('category')
        df_numerized[col_name] = df_numerized[col_name].cat.codes

df_numerized
```

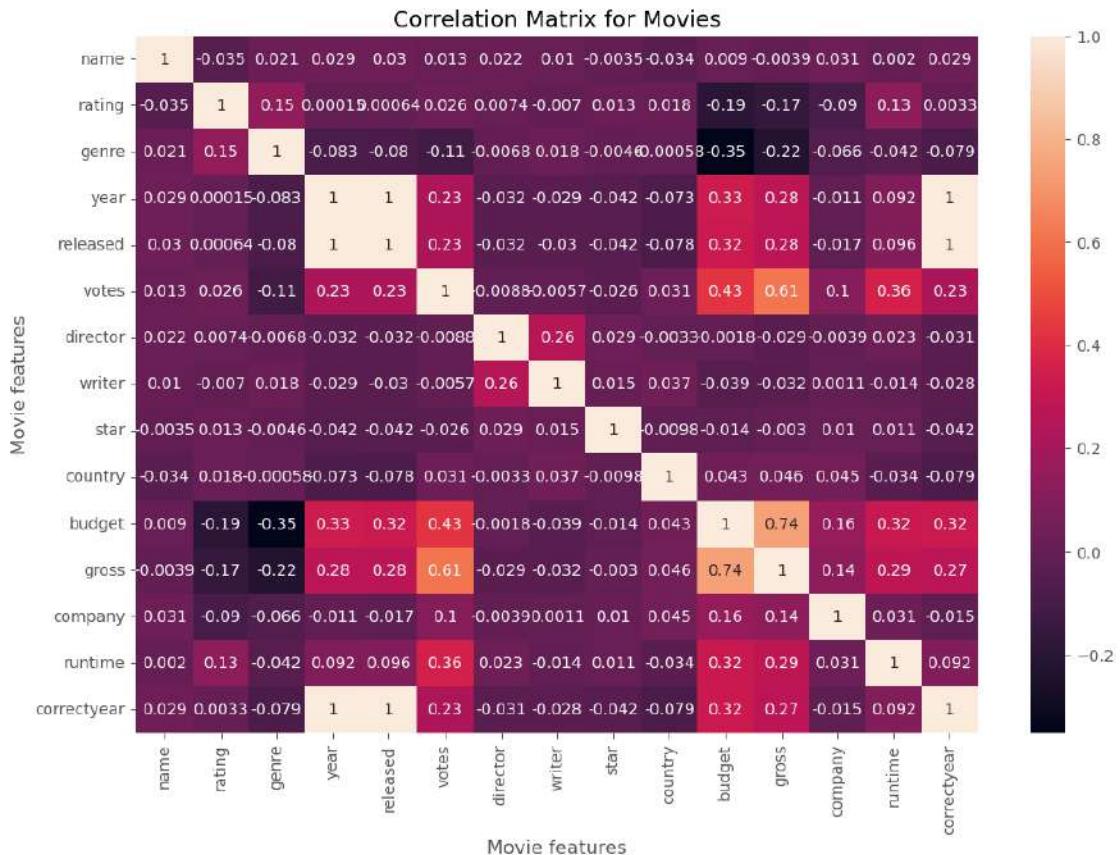
```
[18]:   name  rating  genre  year  released      votes  director  writer  star \
4306    310       4      0  2009     1278  1100000.0     678    1062  1320
5846    312       4      0  2019     1765  903000.0      90     421  1261
2404   3826       4      6  1997      694  1100000.0     678    1062   913
5252   2843       4      0  2015     1604  876000.0     665    1517   305
5691    313       4      0  2018     1714  897000.0      90     421  1261
...     ...
567     766       4      4  1985      188    5300.0     1507     792   945
```

4459	2953	5	6	2009	1559	3500.0	507	2455	1288
2912	1264	2	6	2000	878	43000.0	823	1414	448
183	2271	5	9	1982	61	2300.0	225	46	1264
2534	3868	4	4	1997	651	5800.0	564	138	1557
		country		budget	gross	company	runtime	correctyear	
4306	43	237000000.0		2.847246e+09		1183	162.0		29
5846	43	356000000.0		2.797501e+09		832	181.0		39
2404	43	200000000.0		2.201647e+09		1183	194.0		17
5252	43	245000000.0		2.069522e+09		797	138.0		35
5691	43	321000000.0		2.048360e+09		832	149.0		38
...	...	...		...	...	...	...		
567	43	3000000.0		5.101000e+03		387	83.0		6
4459	43	3000000.0		5.073000e+03		1186	96.0		35
2912	6	5000000.0		2.554000e+03		398	108.0		21
183	43	800000.0		2.270000e+03		494	85.0		2
2534	43	15000000.0		3.090000e+02		431	85.0		17

[4208 rows x 15 columns]

```
[24]: correlation_matrix = df_numerized.corr(method = 'pearson')

sns.heatmap(correlation_matrix, annot = True)
plt.title("Correlation Matrix for Movies")
plt.xlabel("Movie features")
plt.ylabel("Movie features")
plt.show()
```



[25]: # Looking at the highest correlation

```
corr_mat = df_numerized.corr()

corr_pairs = corr_mat.unstack()

print(corr_pairs)
```

name	name	1.000000
	rating	-0.035199
	genre	0.021224
	year	0.028812
	released	0.029970
		...
correctyear	budget	0.322562
	gross	0.272703
	company	-0.015498
	runtime	0.092199
	correctyear	1.000000

Length: 225, dtype: float64

```
[32]: sorted_pairs = corr_pairs.sort_values()  
  
sorted_pairs
```

```
[32]: genre          budget      -0.351819  
budget         genre      -0.351819  
genre          gross      -0.224968  
gross          genre      -0.224968  
rating         budget     -0.193708  
budget         rating     -0.193708  
gross          rating    -0.166113  
rating         gross     -0.166113  
votes           genre    -0.109998  
genre          votes    -0.109998  
company        rating   -0.089601  
rating         company  -0.089601  
year            genre   -0.083211  
genre          year    -0.083211  
released        genre   -0.079860  
genre          released -0.079860  
country        correctyear -0.079487  
correctyear    country  -0.079487  
                  genre   -0.079141  
genre          correctyear -0.079141  
country        released  -0.078488  
released        country  -0.078488  
country        year    -0.072766  
year            country -0.072766  
genre          company -0.065751  
company        genre   -0.065751  
genre          runtime -0.042353  
runtime        genre   -0.042353  
star            released -0.042277  
released        star    -0.042277  
correctyear    star    -0.042205  
star            correctyear -0.042205  
year            star    -0.041950  
star            year    -0.041950  
budget         writer  -0.038956  
writer         budget  -0.038956  
name            rating -0.035199  
rating         name   -0.035199  
country        runtime -0.034296  
runtime        country -0.034296  
country        name   -0.034025  
name            country -0.034025  
director       year   -0.032369
```

year	director	-0.032369
writer	gross	-0.032115
gross	writer	-0.032115
released	director	-0.032004
director	released	-0.032004
	correctyear	-0.030695
correctyear	director	-0.030695
released	writer	-0.030150
writer	released	-0.030150
year	writer	-0.028630
writer	year	-0.028630
director	gross	-0.028589
gross	director	-0.028589
writer	correctyear	-0.028070
correctyear	writer	-0.028070
star	votes	-0.026024
votes	star	-0.026024
released	company	-0.017069
company	released	-0.017069
correctyear	company	-0.015498
company	correctyear	-0.015498
star	budget	-0.014393
budget	star	-0.014393
runtime	writer	-0.013634
writer	runtime	-0.013634
year	company	-0.010689
company	year	-0.010689
country	star	-0.009845
star	country	-0.009845
director	votes	-0.008767
votes	director	-0.008767
writer	rating	-0.006954
rating	writer	-0.006954
director	genre	-0.006759
genre	director	-0.006759
votes	writer	-0.005709
writer	votes	-0.005709
genre	star	-0.004615
star	genre	-0.004615
gross	name	-0.003945
name	gross	-0.003945
company	director	-0.003923
director	company	-0.003923
star	name	-0.003451
name	star	-0.003451
director	country	-0.003290
country	director	-0.003290

star	gross	-0.002979
gross	star	-0.002979
budget	director	-0.001801
director	budget	-0.001801
country	genre	-0.000581
genre	country	-0.000581
rating	year	0.000151
year	rating	0.000151
rating	released	0.000636
released	rating	0.000636
writer	company	0.001097
company	writer	0.001097
name	runtime	0.002002
runtime	name	0.002002
rating	correctyear	0.003341
correctyear	rating	0.003341
director	rating	0.007382
rating	director	0.007382
name	budget	0.009019
budget	name	0.009019
star	company	0.010182
company	star	0.010182
writer	name	0.010225
name	writer	0.010225
star	runtime	0.011284
runtime	star	0.011284
star	rating	0.012830
rating	star	0.012830
votes	name	0.013228
name	votes	0.013228
star	writer	0.015319
writer	star	0.015319
rating	country	0.017780
country	rating	0.017780
genre	writer	0.018334
writer	genre	0.018334
genre	name	0.021224
name	genre	0.021224
director	name	0.022464
name	director	0.022464
runtime	director	0.023315
director	runtime	0.023315
rating	votes	0.026201
votes	rating	0.026201
correctyear	name	0.028597
name	correctyear	0.028597
year	name	0.028812

name	year	0.028812
director	star	0.029334
star	director	0.029334
name	released	0.029970
released	name	0.029970
name	company	0.030669
company	name	0.030669
country	votes	0.030868
votes	country	0.030868
company	runtime	0.030932
runtime	company	0.030932
writer	country	0.037413
country	writer	0.037413
	budget	0.042706
budget	country	0.042706
country	company	0.045121
company	country	0.045121
gross	country	0.046342
country	gross	0.046342
year	runtime	0.091519
runtime	year	0.091519
	correctyear	0.092199
correctyear	runtime	0.092199
released	runtime	0.096204
runtime	released	0.096204
votes	company	0.102788
company	votes	0.102788
rating	runtime	0.133165
runtime	rating	0.133165
gross	company	0.135490
company	gross	0.135490
rating	genre	0.148689
genre	rating	0.148689
company	budget	0.164546
budget	company	0.164546
votes	correctyear	0.225378
correctyear	votes	0.225378
votes	released	0.225693
released	votes	0.225693
votes	year	0.227775
year	votes	0.227775
writer	director	0.257260
director	writer	0.257260
correctyear	gross	0.272703
gross	correctyear	0.272703
	released	0.276956
released	gross	0.276956

```
year           gross          0.278039
gross          year          0.278039
runtime        gross          0.285549
gross          runtime        0.285549
runtime        budget         0.318132
budget         runtime        0.318132
                  correctyear  0.322562
correctyear    budget         0.322562
released       budget         0.323215
budget         released        0.323215
year           budget         0.330178
budget         year           0.330178
votes          runtime        0.356359
runtime        votes          0.356359
budget         votes          0.427663
votes          budget         0.427663
gross          votes          0.612410
votes          gross          0.612410
gross          budget         0.741602
budget         gross          0.741602
released       year           0.997525
year           released        0.997525
correctyear    released        0.998380
released       correctyear    0.998380
year           correctyear    0.998888
correctyear    year           0.998888
name           name           1.000000
writer         writer         1.000000
company        company        1.000000
gross          gross          1.000000
budget         budget         1.000000
country        country        1.000000
star           star           1.000000
director       director       1.000000
votes          votes          1.000000
released       released        1.000000
year           year           1.000000
genre          genre          1.000000
rating         rating         1.000000
runtime        runtime        1.000000
correctyear    correctyear    1.000000
dtype: float64
```

```
[33]: high_corr = sorted_pairs[(sorted_pairs) > 0.5]
```

```
high_corr
```

```
[33]: gross          votes        0.612410
      votes          gross        0.612410
      gross          budget       0.741602
      budget         gross        0.741602
      released       year         0.997525
      year           released     0.997525
      correctyear    released     0.998380
      released       correctyear  0.998380
      year           correctyear  0.998888
      correctyear    year         0.998888
      name           name         1.000000
      writer         writer       1.000000
      company        company     1.000000
      gross          gross        1.000000
      budget         budget       1.000000
      country        country     1.000000
      star            star         1.000000
      director       director     1.000000
      votes          votes        1.000000
      released       released     1.000000
      year           year         1.000000
      genre           genre        1.000000
      rating          rating       1.000000
      runtime         runtime      1.000000
      correctyear    correctyear  1.000000
      dtype: float64
```

```
[ ]: # Votes and budget have the highest correlation to gross earnings
```

```
# Company has no correlation
```



project

5

**Python - Movie Industry EDA Project**

# python-movie-industry-eda-project

February 24, 2024

## 1 Movie Industry Exploratory Data Analysis

1.1 Objective: Investigate the film industry to gain sufficient understanding of what attributes to success and in turn utilize this analysis to create *actionable* recommendations for companies to enter the industry.

1.1.1 Importing necessary libraries and the datasets.

```
[1]: import pandas as pd  
import seaborn as sns  
import matplotlib.pyplot as plt  
import datetime as dt  
import numpy as np
```

```
[57]: #Setting the default style for plots  
plt.style.use('ggplot')  
  
from matplotlib.pyplot import figure  
plt.rcParams['figure.figsize'] = (12,8)  
  
%matplotlib inline
```

```
[3]: movie_dates_df = pd.read_csv('movie_release_dates.csv', index_col=0)  
theaters_df = pd.read_csv('movie_theater_data.csv', index_col=0)  
awards_df = pd.read_csv('movie_awards.csv', index_col=0)  
actors_df = pd.read_csv('Actors_Table.csv')  
directors_df = pd.read_csv('Directors_Table.csv')  
imdb_base_df = pd.read_csv('IMDb_base.csv')  
imdb_budgets_df = pd.read_csv('IMDb_budgets.csv')  
studio_df = pd.read_csv('studiodf.csv')
```

```
[4]: #First remove any movies that had a $0 domestic gross.  
imdb_budgets_df = imdb_budgets_df[imdb_budgets_df['Domestic Gross'] != 0]
```

### 1.1.2 Previewing the head of each dataframe so we know what data we are working with.

```
[5]: imdb_budgets_df.head()
```

```
[5]:
```

	Movie	Year	IMDb	Rating	Runtime	\
0	Avengers: Endgame	2019	8.4	PG-13	181	
1	Avatar	2009	7.8	PG-13	162	
2	Black Panther	2018	7.3	PG-13	134	
3	Avengers: Infinity War	2018	8.4	PG-13	149	
4	Titanic	1997	7.8	PG-13	194	

	Genre	Release Date	Production Budget	\
0	Action, Adventure, Drama	Apr 23, 2019	400000000	
1	Action, Adventure, Fantasy	Dec 17, 2009	237000000	
2	Action, Adventure, Sci-Fi	Feb 13, 2018	200000000	
3	Action, Adventure, Sci-Fi	Apr 25, 2018	300000000	
4	Drama, Romance	Dec 18, 1997	200000000	

	Domestic Gross	Worldwide Gross
0	858373000	2797800564
1	760507625	2788701337
2	700059566	1346103376
3	678815482	2048359754
4	659363944	2208208395

```
[6]: movie_dates_df.head()
```

```
[6]:
```

	movie	release_date	release_month	release_day	\
0	Metropolis	1927-03-06	March	Sunday	
1	Dr. Mabuse, the Gambler	1927-08-08	August	Monday	
2	The Unknown	1927-06-03	June	Friday	
3	The Jazz Singer	1927-10-06	October	Thursday	
4	Chicago	1927-12-23	December	Friday	

	release_year
0	1927
1	1927
2	1927
3	1927
4	1927

```
[7]: theaters_df.head()
```

```
[7]:
```

	title	max_theaters	year	total_dom_gross(\$)	\
0	The Lion King	4802	2019	543638043	
1	Avengers: Endgame	4662	2019	858373000	
2	Spider-Man: Far from Home	4634	2019	390532085	

```
3           Toy Story 4        4575  2019        434038008
4           It Chapter Two      4570  2019        211593228
```

```
studio
0    Disney
1    Disney
2    Sony
3    Disney
4  Warner Bros.
```

```
[8]: actors_df.head()
```

```
[8]:          Movie  Year      value  Release Date \
0  Avengers: Endgame  2019  Robert Downey Jr.  Apr 23, 2019
1  Avengers: Endgame  2019       Chris Evans  Apr 23, 2019
2  Avengers: Endgame  2019       Mark Ruffalo  Apr 23, 2019
3  Avengers: Endgame  2019       Chris Hemsworth  Apr 23, 2019
4           Avatar    2009     Sam Worthington  Dec 17, 2009

   Production Budget  Domestic Gross  Worldwide Gross
0      4000000000      858373000      2797800564
1      4000000000      858373000      2797800564
2      4000000000      858373000      2797800564
3      4000000000      858373000      2797800564
4      2370000000      760507625      2788701337
```

```
[9]: directors_df.head()
```

```
[9]:          Movie  Year      value  Release Date \
0  Avengers: Endgame  2019       Joe Russo  Apr 23, 2019
1  Avengers: Endgame  2019  Anthony Russo  Apr 23, 2019
2           Avatar    2009  James Cameron  Dec 17, 2009
3      Black Panther  2018      Ryan Coogler  Feb 13, 2018
4  Avengers: Infinity War  2018       Joe Russo  Apr 25, 2018

   Production Budget  Domestic Gross  Worldwide Gross
0      4000000000      858373000      2797800564
1      4000000000      858373000      2797800564
2      2370000000      760507625      2788701337
3      2000000000      700059566      1346103376
4      3000000000      678815482      2048359754
```

```
[10]: imdb_base_df.head()
```

```
[10]:                               Movie  Year  IMDb Rating  Runtime \
0  Star Wars: Episode VII - The Force Awakens  2015      7.9    PG-13       138
1                  Avengers: Endgame  2019      8.4    PG-13       181
```

```
2                               Avatar   2009   7.8   PG-13      162
3                         Black Panther 2018   7.3   PG-13      134
4                   Avengers: Infinity War 2018   8.4   PG-13      149
```

```
Genre
0  Action, Adventure, Sci-Fi
1  Action, Adventure, Drama
2  Action, Adventure, Fantasy
3  Action, Adventure, Sci-Fi
4  Action, Adventure, Sci-Fi
```

```
[11]: studio_df.head()
```

```
[11]:          title        studio \
0           Toy Story 3    Buena Vista
1  Alice in Wonderland (2010)  Buena Vista
2  Harry Potter and the Deathly Hallows Part 1      WB
3                  Inception      WB
4       Shrek Forever After  Pixar/Dreamworks

  domestic_gross  foreign_gross  year
0     415000000.0     652000000  2010
1     334200000.0     691300000  2010
2     296000000.0     664300000  2010
3     292600000.0     535700000  2010
4     238700000.0     513900000  2010
```

## 2 Question 1: What are the most profitable movies and how much should you spend?

Let's calculate profit and profit margin for each of the movies in `imdb_budgets_df` dataframe and add those as new columns.

Here, we'll define profit as `Worldwide Gross - Production Budget`.

It will also be beneficial in our analysis to have uniformity when discussing movie budgets and profits so we will also create an adjusted budget and adjusted profit column to account for inflation.

We will use an average inflation rate of 3.22%.

```
[12]: imdb_budgets_df['Profit'] = imdb_budgets_df['Worldwide Gross'] - \
    ↪imdb_budgets_df['Production Budget']

imdb_budgets_df['Profit_Margin'] = (imdb_budgets_df['Worldwide Gross'] - \
    ↪imdb_budgets_df['Production Budget']) / \
    ↪imdb_budgets_df['Worldwide Gross']
```

```
[13]: imdb_budgets_df['Adjusted_Budget'] = (((2020-imdb_budgets_df['Year'])*
    ↪0322)+1)*
                           imdb_budgets_df['Production Budget'])

#Suppressing Scientific Notation
pd.options.display.float_format = '{:.2f}'.format

imdb_budgets_df['Adjusted_Profit'] = (((2020-imdb_budgets_df['Year'])*
    ↪0322)+1)*imdb_budgets_df['Profit']
imdb_budgets_df.head()
```

```
[13]:          Movie  Year  IMDb Rating Runtime \
0      Avengers: Endgame  2019   8.40  PG-13     181
1                  Avatar  2009   7.80  PG-13     162
2      Black Panther  2018   7.30  PG-13     134
3  Avengers: Infinity War  2018   8.40  PG-13     149
4            Titanic  1997   7.80  PG-13     194

          Genre Release Date Production Budget \
0  Action, Adventure, Drama  Apr 23, 2019        400000000
1  Action, Adventure, Fantasy  Dec 17, 2009        237000000
2  Action, Adventure, Sci-Fi  Feb 13, 2018        200000000
3  Action, Adventure, Sci-Fi  Apr 25, 2018        300000000
4      Drama, Romance  Dec 18, 1997        200000000

  Domestic Gross Worldwide Gross      Profit Profit_Margin \
0      858373000           2797800564  2397800564        0.86
1      760507625           2788701337  2551701337        0.92
2      700059566           1346103376  1146103376        0.85
3      678815482           2048359754  1748359754        0.85
4      659363944           2208208395  2008208395        0.91

  Adjusted_Budget  Adjusted_Profit
0      412880000.00    2475009742.16
1      320945400.00    3455513950.57
2      212880000.00    1219912433.41
3      319320000.00    1860954122.16
4      348120000.00    3495487532.34
```

For this question we are specifically looking at profitable movies. We'll create a separate dataframe called `profitable_movies_df` where the `Profit` column is greater than 0. We will then sort by `Adjusted_Profit` to rank movies in terms of profitability.

```
[14]: profitable_movies_df = imdb_budgets_df.loc[imdb_budgets_df['Profit'] > 0]
profitable_ranked_df = profitable_movies_df.sort_values(by=['Adjusted_Profit'], ↪
    ascending=False)
```

```
profitable_ranked_df.reset_index(inplace=True) #Modify the DataFrame in place
    ↴(do not create a new object).
profitable_ranked_df.head()
```

```
[14]:   index          Movie  Year  IMDb Rating Runtime \
0      4        Titanic  1997  7.80  PG-13     194
1      1         Avatar  2009  7.80  PG-13     162
2      0  Avengers: Endgame  2019  8.40  PG-13     181
3      3  Avengers: Infinity War  2018  8.40  PG-13     149
4     28       Jurassic Park  1993  8.10  PG-13     127

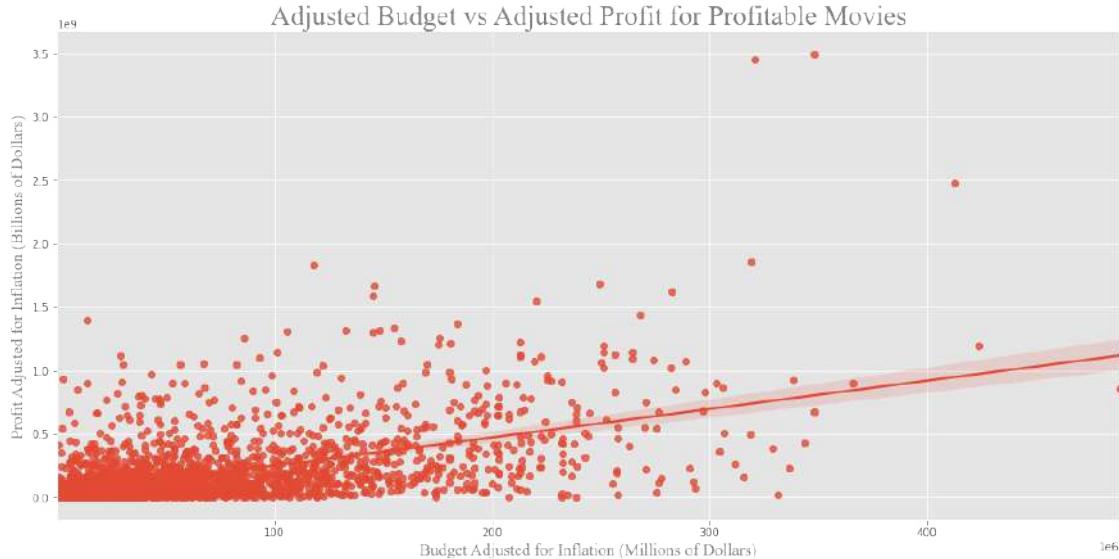
                           Genre Release Date Production Budget \
0  Drama, Romance  Dec 18, 1997        200000000
1  Action, Adventure, Fantasy  Dec 17, 2009        237000000
2  Action, Adventure, Drama  Apr 23, 2019        400000000
3  Action, Adventure, Sci-Fi  Apr 25, 2018        300000000
4  Action, Adventure, Sci-Fi  Jun 11, 1993        63000000

  Domestic Gross  Worldwide Gross      Profit  Profit_Margin \
0      659363944      2208208395  2008208395        0.91
1      760507625      2788701337  2551701337        0.92
2      858373000      2797800564  2397800564        0.86
3      678815482      2048359754  1748359754        0.85
4      402523348      1045627627  982627627        0.94

  Adjusted_Budget  Adjusted_Profit
0      348120000.00      3495487532.34
1      320945400.00      3455513950.57
2      412880000.00      2475009742.16
3      319320000.00      1860954122.16
4      117772200.00      1836924085.91
```

Now that we've got our profitable movie data, let's take a look at adjusted profit versus adjusted budget for each of the movies in the dataframe.

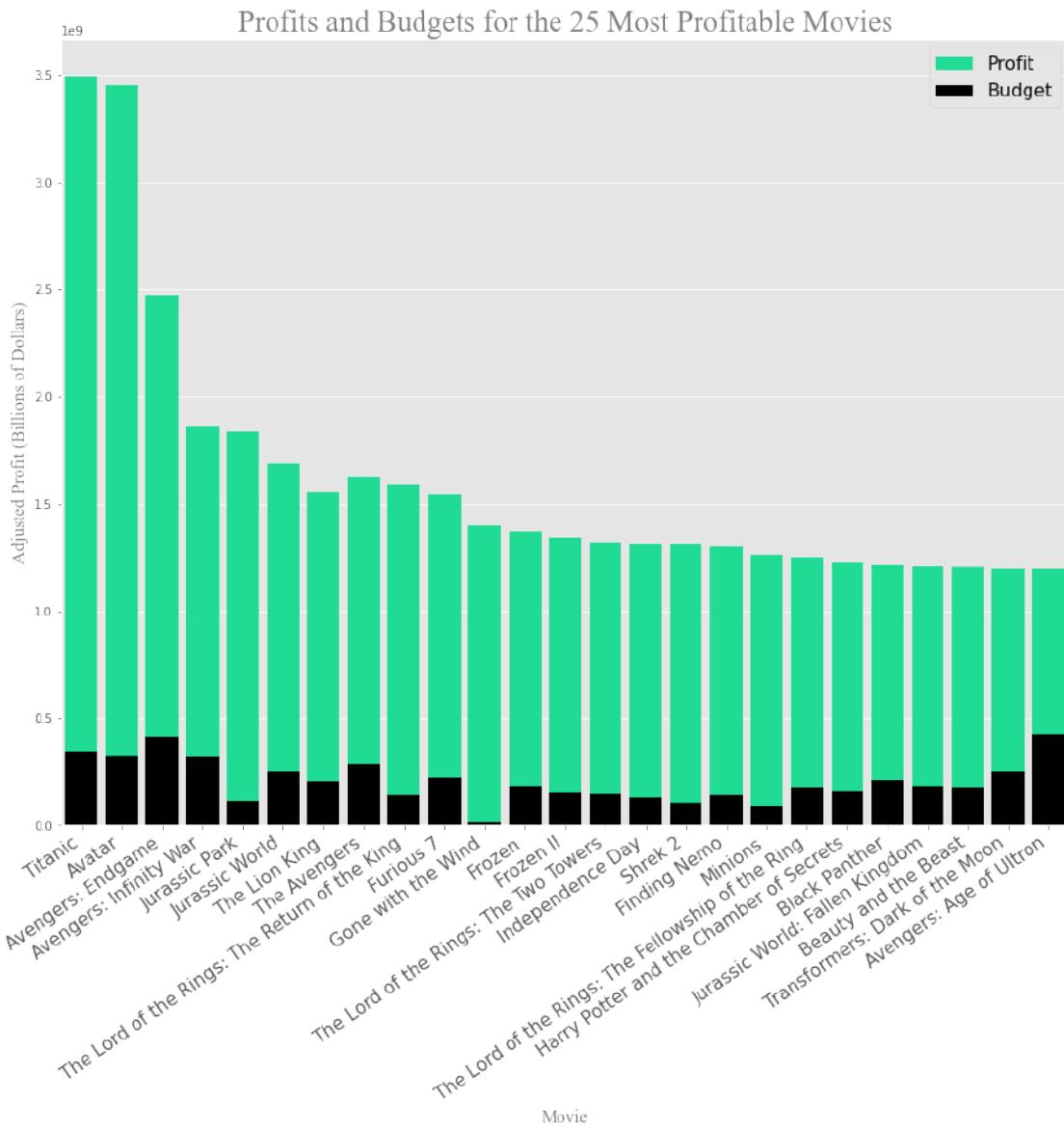
```
[15]: ax1 = sns.lmplot(x='Adjusted_Budget', y='Adjusted_Profit', 
    ↴data=profitable_ranked_df, height=7, aspect=2)
plt.xlabel('Budget Adjusted for Inflation (Millions of Dollars)', fontdict = 
    ↴{'fontname': 'Times New Roman', 'color': 'gray', 'fontsize' : '15'})
#setting x-axis label
plt.ticklabel_format(axis='x', style='sci', scilimits=(6,6))
plt.ylabel('Profit Adjusted for Inflation (Billions of Dollars)', fontdict = 
    ↴{'fontname': 'Times New Roman', 'color': 'gray', 'fontsize' : '15'})
plt.title('Adjusted Budget vs Adjusted Profit for Profitable Movies', fontdict = 
    ↴= {'fontname': 'Times New Roman', 'color': 'gray', 'fontsize' : '25'})
plt.savefig('BudgetVProfit', dpi = 300);
```



This scatter plot is helpful in beginning to understand how much money should be budgeted for a movie. The positive trend line indicates that an increase in the budget will result in an increase in profit.

Let's take a look at the most successful movies so that we can get a better idea of what the budget should be.

```
[16]: plt.figure(figsize=(15,12))
sns.barplot(x=profitable_ranked_df.loc[0:25, 'Movie'],y=profitable_ranked_df.
            loc[0:25, 'Adjusted_Profit'],
            color='mediumspringgreen', label='Profit', ci=None)
sns.barplot(x=profitable_ranked_df.loc[0:25, 'Movie'],y=profitable_ranked_df.
            loc[0:25, 'Adjusted_Budget'],
            color='black', label='Budget', ci=None)
plt.xlabel('Movie', fontdict = {'fontname': 'Times New Roman', 'color': 'gray',
                                'fontsize' : '15'})
plt.title("Profits and Budgets for the 25 Most Profitable Movies", fontdict =
           {'fontname': 'Times New Roman', 'color': 'gray', 'fontsize' : '25'})
plt.ylabel('Adjusted Profit (Billions of Dollars)', fontdict = {'fontname': 'Times New Roman', 'color': 'gray', 'fontsize' : '15'})
plt.xticks(rotation=35, horizontalalignment='right', fontsize=15)
plt.legend(loc='best', fontsize=15)
plt.savefig('ProfitBudgetTop25', dpi=300);
```



```
[17]: profitable_movies_df['Adjusted_Budget'].describe()
```

```
[17]: count      2836.00
mean      60689139.20
std       63199464.86
min       10606.40
25%      16608850.00
50%      38684100.00
75%      82247150.00
max      488834200.00
Name: Adjusted_Budget, dtype: float64
```

```
[18]: profitable_movies_df.loc[0:24, 'Adjusted_Budget'].describe()
```

```
[18]: count      25.00
mean     242777774.40
std      80698866.89
min     106064000.00
25%    180635000.00
50%    225760000.00
75%    282960000.00
max     423765000.00
Name: Adjusted_Budget, dtype: float64
```

```
[19]: profitable_movies_df['Profit_Margin'].describe()
```

```
[19]: count    2836.00
mean      0.62
std       0.24
min       0.00
25%      0.47
50%      0.67
75%      0.81
max      1.00
Name: Profit_Margin, dtype: float64
```

```
[20]: profitable_movies_df.loc[0:24, 'Profit_Margin'].describe()
```

```
[20]: count    25.00
mean      0.85
std       0.05
min       0.74
25%      0.81
50%      0.85
75%      0.87
max      0.93
Name: Profit_Margin, dtype: float64
```

```
[21]: len(profitable_ranked_df.loc[profitable_ranked_df['Profit_Margin'] > 0.5])
```

```
[21]: 2041
```

Clearly the most successful 25 movies have both incredible profits and profit margins. *Titanic* (1997), *Avatar*, and *Avengers: Endgame* are the most successful movies in terms of sheer profit.

So how do we know what to spend? We need to think about what sort of profit margin we want to see. 2043 out of 2841 total profitable movies have a profit margin over 50%. That's good news as it indicates that we can be more aggressive in choosing a threshold for the profit margin. The top 25 movies have a median profit margin of 84.9% with a median budget of \\$225,760,000. When looking at all of our profitable movies, the profit margin drops significantly to 67.1% and the budget

drops significantly to \\$38,676,000. We use the median to describe our data here as the mean will be skewed by outlier data.

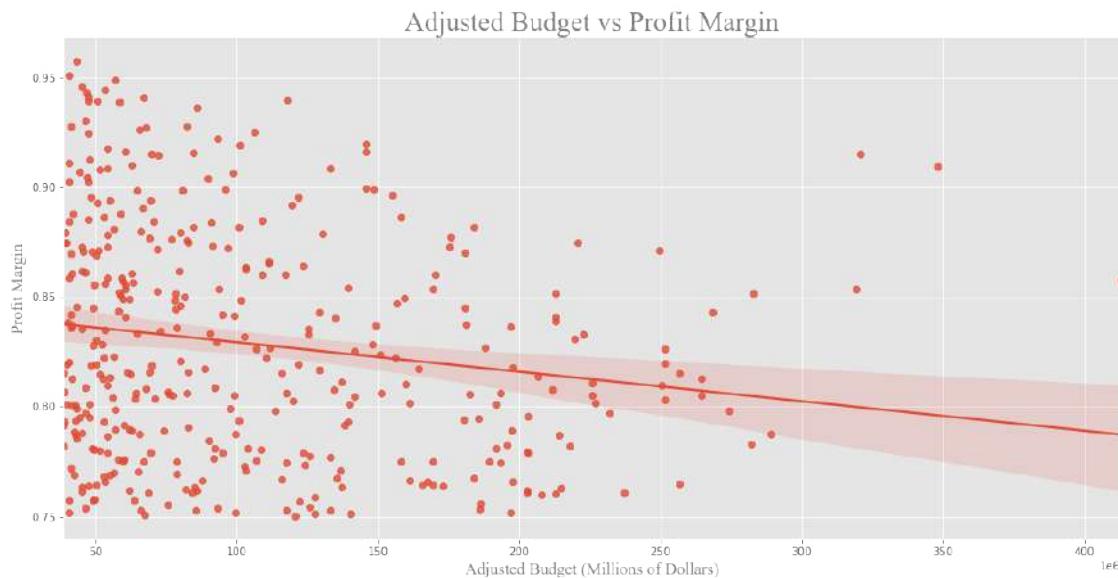
Let's filter the data with a profit margin of 75% or greater and a budget greater than \$38,676,000.

```
[23]: filtered_df = profitable_ranked_df.loc[(profitable_ranked_df['Profit_Margin'] >= 0.75) & (profitable_ranked_df['Adjusted_Budget'] > 38676000)]  
len(filtered_df)
```

```
[23]: 374
```

After filtering we still have 374 movies left upon which to draw conclusions.

```
[32]: ax2 = sns.lmplot(x='Adjusted_Budget', y='Profit_Margin', data=filtered_df, height=7, aspect=2)  
plt.xlabel('Adjusted Budget (Millions of Dollars)', fontdict = {'fontname': 'Times New Roman', 'color': 'gray', 'fontsize' : '15'})  
plt.ticklabel_format(axis='x', style='sci', scilimits=(6,6))  
plt.ylabel('Profit Margin', fontdict = {'fontname': 'Times New Roman', 'color': 'gray', 'fontsize' : '15'})  
plt.title('Adjusted Budget vs Profit Margin', fontdict = {'fontname': 'Times New Roman', 'color': 'gray', 'fontsize' : '25'})  
plt.savefig('BudgetVMargin', dpi=300);
```



```
[25]: filtered_df.describe()
```

```
[25]:    index      Year   IMDb  Runtime Production Budget Domestic_Gross \
count  374.00  374.00  374.00    374.00          374.00        374.00
```

```

mean    391.53 2004.97    7.01   118.60      77814178.13    193378841.67
std     378.20 10.81     0.90    24.02      57570152.51    127088965.57
min     0.00 1956.00     3.30    79.00     13500000.00    19019882.00
25%    111.25 1998.00    6.40   100.00     35000000.00    106948347.75
50%    279.50 2007.00    7.00   116.00     55000000.00    162801999.50
75%    550.50 2014.00    7.70   131.75    100000000.00   242081446.50
max    2424.00 2020.00   9.00   228.00    400000000.00   858373000.00

          Worldwide_Gross       Profit  Profit_Margin  Adjusted_Budget \
count            374.00      374.00      374.00        374.00
mean    484994903.63  407180725.50      0.83    105858522.51
std     377690264.14  329994078.69      0.05    66272237.80
min     69995385.00  54995385.00      0.75    38685000.00
25%    217288435.75 176354400.25      0.78    53471100.00
50%    350937609.00 299062980.00      0.82    82249300.00
75%    636084264.50 513979301.75      0.87   139654600.00
max    2797800564.00 2551701337.00      0.96   412880000.00

          Adjusted_Profit
count            374.00
mean    562879114.94
std     413114307.71
min     123209844.42
25%    274861614.08
50%    449229900.01
75%    719591073.46
max    3495487532.34

```

We examine the data in a scatter plot again to see if we can determine trends. Our data is much more spread out when comparing profit margin and budget. The trend line in this plot is negative which cautions against spending too much money as we may potentially hurt our profit margin. Looking at the filtered data, we have a median budget of \$82,249,300 and a median profit margin of 81.9%.

**Question 1 Conclusion:** We recommend that our Company should budget approximately \$82,250,000 to make a movie. This should correlate with a profit margin above 80%.

### 3 Question 2: Which movie genres are most commonly produced and does quantity equate to higher net profits?

```
[27]: #Create a genre table that separates each value in the genre column in their
      ↪own rows.

imdb_budgets_df['Genre'] = imdb_budgets_df['Genre'].str.split(', ')
imdb_budgets_df1 = imdb_budgets_df['Genre'].apply(pd.Series)

imdb_budgets_df2 = pd.merge(imdb_budgets_df, imdb_budgets_df1, right_index = True,
                           left_index = True)
```

```

imdb_budgets_df3 = imdb_budgets_df2.drop(['Genre'], axis = 1)

genre_budgets_df = imdb_budgets_df3.melt(id_vars=['Movie', 'Year'], ↴
    value_vars=[0, 1, 2] ,var_name = ['X'])
genre_budgets_df = pd.merge(genre_budgets_df, imdb_budgets_df)
genre_budgets_df = genre_budgets_df.drop(['Genre', 'X'], axis=1)
genre_budgets_df = genre_budgets_df.drop_duplicates()
genre_budgets_df = genre_budgets_df.rename(columns={'value': 'Genre'})
genre_budgets_df = genre_budgets_df.dropna()

```

[28]: #Do a count of all movies grouped by genre.

```

m_by_genre = genre_budgets_df.groupby('Genre', as_index=False)[['Movie']].count(). ↴
    sort_values(by='Movie', ascending=False)

```

[29]: m\_by\_genre

[29]:

	Genre	Movie
6	Drama	1876
4	Comedy	1444
0	Action	1045
1	Adventure	834
5	Crime	689
15	Romance	622
18	Thriller	615
11	Horror	410
14	Mystery	356
16	Sci-Fi	330
8	Fantasy	324
3	Biography	294
7	Family	265
2	Animation	205
10	History	123
12	Music	109
17	Sport	100
19	War	68
13	Musical	39
20	Western	32
9	Film-Noir	6

[31]: #Plot the above findings.

```

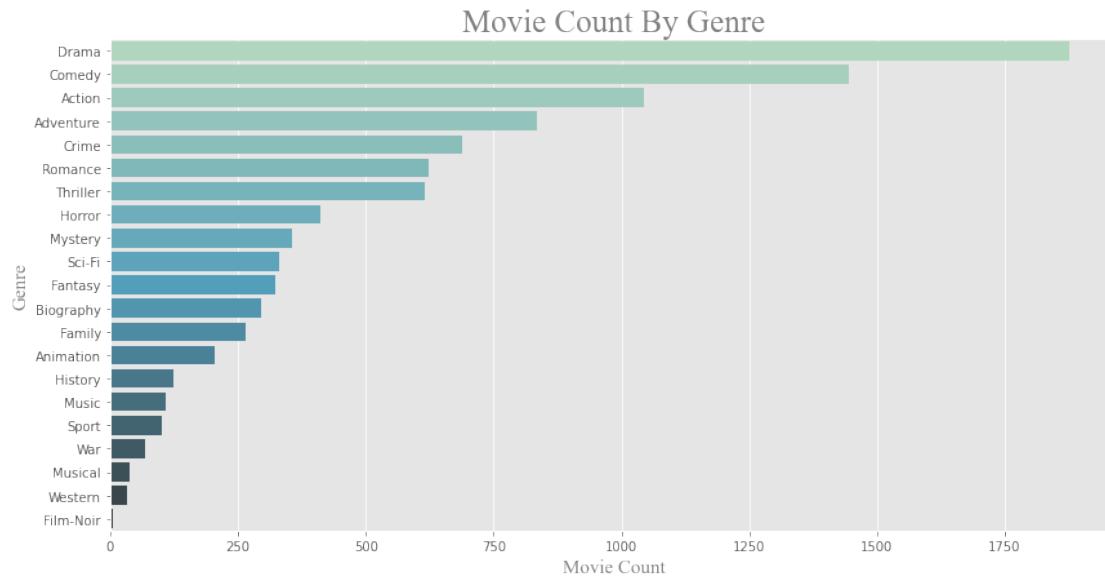
plt.figure(figsize=(14,7))
ax3 = sns.barplot(x=m_by_genre['Movie'], y=m_by_genre['Genre'], ↴
    palette='GnBu_d')
plt.xlabel('Movie Count', fontdict = {'fontname': 'Times New Roman', 'color': ↴
    'gray', 'fontsize' : '15'})

```

```

plt.ylabel('Genre', fontdict = {'fontname': 'Times New Roman', 'color': 'gray', 'fontsize' : '15'})
plt.title('Movie Count By Genre', fontdict = {'fontname': 'Times New Roman', 'color': 'gray', 'fontsize' : '25'})
plt.savefig('CountGenre', dpi=300);

```



We can see that drama, comedy, and action dominate the quantity of movie genres but does this necessarily mean these are the most profitable genres? In order to determine this we will once again group each genre but this time we are going to take a look at the average net profit for each.

[35]: #Once again group the movies by genre, showing the average net profit and profit margin for each.

```

p_by_genre = genre_budgets_df.groupby('Genre', as_index=False)[['Adjusted_Profit', 'Profit_Margin']].median().sort_values(by='Adjusted_Profit', ascending=False)

```

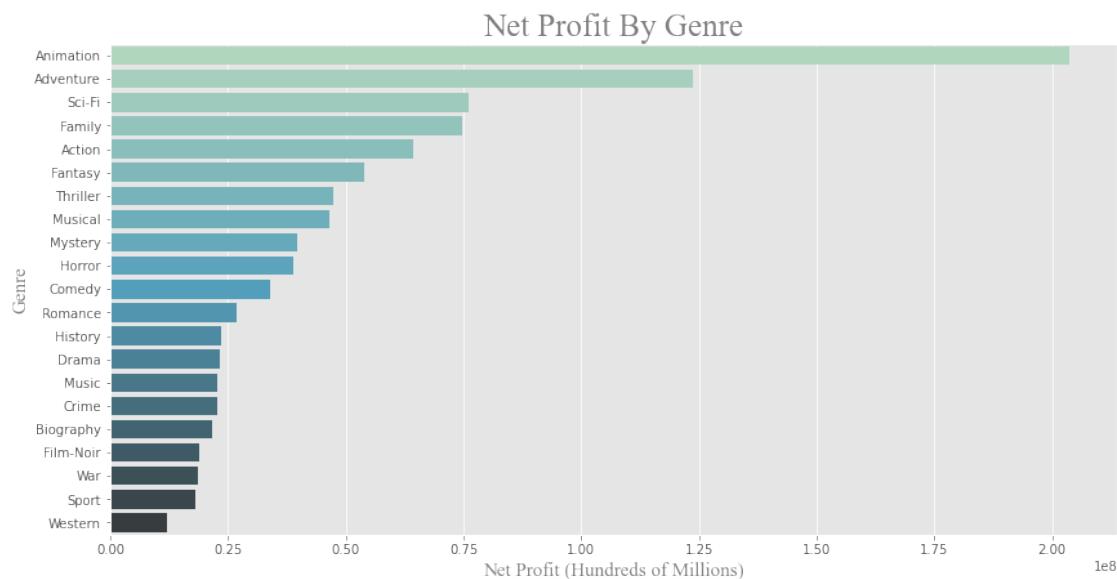
[36]: p\_by\_genre

	Genre	Adjusted_Profit	Profit_Margin
2	Animation	203606574.36	0.68
1	Adventure	123795016.96	0.61
16	Sci-Fi	76199115.79	0.60
7	Family	74621544.29	0.58
0	Action	64332532.19	0.52
8	Fantasy	54057582.24	0.54
18	Thriller	47338952.53	0.60
13	Musical	46631897.60	0.65
14	Mystery	39634323.82	0.61

11	Horror	38963349.12	0.67
4	Comedy	33917454.39	0.55
15	Romance	26739545.09	0.57
10	History	23435554.73	0.40
6	Drama	23258412.08	0.50
12	Music	22774962.29	0.55
5	Crime	22752334.82	0.40
3	Biography	21750633.96	0.43
9	Film-Noir	18766783.04	0.81
19	War	18653512.63	0.37
17	Sport	17950554.99	0.35
20	Western	12037135.33	0.39

[37]: *#Plot the above findings.*

```
plt.figure(figsize=(14,7))
ax4 = sns.barplot(x=p_by_genre['Adjusted_Profit'], y=p_by_genre['Genre'],
                   palette='GnBu_d')
plt.xlabel('Net Profit (Hundreds of Millions)', fontdict = {'fontname': 'Times New Roman', 'color': 'gray', 'fontsize' : '15'})
plt.ylabel('Genre',fontdict = {'fontname': 'Times New Roman', 'color': 'gray', 'fontsize' : '15'})
plt.title('Net Profit By Genre', fontdict = {'fontname': 'Times New Roman', 'color': 'gray', 'fontsize' : '25'})
plt.savefig('NetProfitGenre', dpi=300);
```



[38]: plt.figure(figsize=(14,7))

```

ax5 = sns.barplot(x=p_by_genre['Profit Margin'], y=p_by_genre['Genre'],  

                   palette='GnBu_d')  

plt.xlabel('Genre', fontdict = {'fontname': 'Times New Roman', 'color': 'gray',  

                                'fontsize' : '15'})  

plt.ylabel('Profit Margin', fontdict = {'fontname': 'Times New Roman', 'color':  

                                         'gray', 'fontsize' : '15'})  

plt.title('Profit Margin By Genre', fontdict = {'fontname': 'Times New Roman',  

                                                'color': 'gray', 'fontsize' : '25'})  

plt.xlim(0.3, 0.85)  

plt.savefig('ProfitMarginGenre', dpi=300);

```



Interesting, although they are not the most commonly released genres; animation, adventure, and sci-fi typically have the most success in terms of median net profit. We can also see that Animation has a desirable profit margin along with horror and musicals. Note: although Film Noir leads with a .8+ profit margin this is based on 6 movies and has to be disregarded due to the small sample size.

Lastly, of what percentage of the total net profit from all genres does each genre account?

```
[39]: #Grouped by genre, find the percent total of the net profit for each.  

per_by_genre = genre_budgets_df.groupby(['Genre'],  

                                         as_index=False)[['Adjusted_Profit']].sum().sort_values(by='Adjusted_Profit',  

                                         ascending=False)  

per_by_genre['Percent Total of Net Profit'] = (per_by_genre['Adjusted_Profit']/  

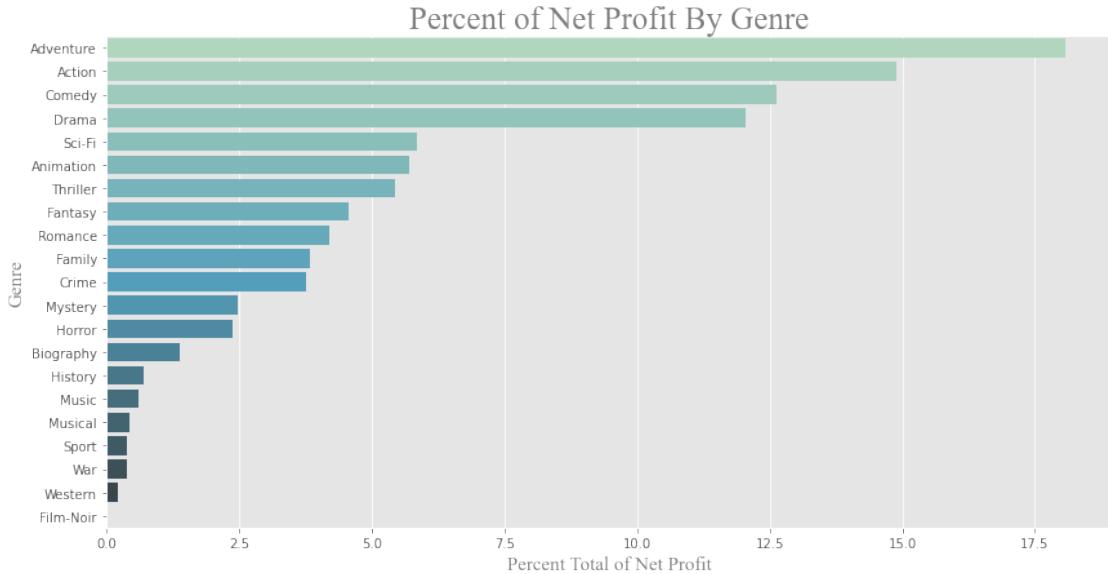
                                              per_by_genre['Adjusted_Profit'].sum()*100).round(2)  

per_by_genre
```

```
[39]:      Genre  Adjusted_Profit  Percent Total of Net Profit
1    Adventure   217335741708.40           18.07
0     Action    178930045524.32           14.88
4    Comedy    151922895671.69           12.63
6     Drama    144990041873.71           12.05
16   Sci-Fi     70465612908.78            5.86
2  Animation   68720987812.40            5.71
18  Thriller   65442236225.98            5.44
8   Fantasy    54797139085.80            4.56
15  Romance    50510744180.92            4.20
7   Family     46040638020.14            3.83
5   Crime      45194406614.69            3.76
14  Mystery    29903244700.35            2.49
11  Horror     28800384751.85            2.39
3  Biography   16776660619.24            1.39
10  History    8429562660.69            0.70
12   Music     7439929226.68            0.62
13  Musical    5228065825.20            0.43
17   Sport      4620549486.84            0.38
19    War       4619522490.02            0.38
20  Western    2551516786.77            0.21
9  Film-Noir   153313504.88             0.01
```

```
[40]: #Plot the above findings.
plt.figure(figsize=(14,7))
ax6 = sns.barplot(x=per_by_genre['Percent Total of Net Profit'],  

                   y=per_by_genre['Genre'], palette='GnBu_d')
plt.xlabel('Percent Total of Net Profit', fontdict = {'fontname': 'Times New Roman', 'color': 'gray', 'fontsize' : '15'})
plt.ylabel('Genre', fontdict = {'fontname': 'Times New Roman', 'color': 'gray', 'fontsize' : '15'})
plt.title('Percent of Net Profit By Genre', fontdict = {'fontname': 'Times New Roman', 'color': 'gray', 'fontsize' : '25'})
plt.savefig('PercentProfitGenre');
```



Now we can see that adventure, action, comedy and drama make up the lionshare of the overall net profits from all movies. However, from our recent observations we know there are also major opportunities in the animation and sci-fi markets due to lower saturation but high average net profits. We will soon determine which genres are most successful during which months.

**Question 2 Conclusion:** We recommend that our Company should focus their efforts on the top 6 most profitable movie genres: Adventure, Action, Comedy, Drama, Sci-Fi and Animation. A further recommendation to focus on Sci-Fi and Animation due to less competition and a higher opportunity to profit.

## 4 Question 3: What is the best time of the year to release a movie?

- ```
[41]: #Convert the Release Date field to type datetime.
imdb_budgets_df['Release Date'] = pd.to_datetime(imdb_budgets_df['Release Date'])

[42]: #Add a new column called month, displaying only the month from the release date.
dateData = [x.strftime('%B') for x in imdb_budgets_df['Release Date']]
imdb_budgets_df['Month'] = dateData

[43]: #Count the total number of movies and group by month.
m_by_month = imdb_budgets_df.groupby(['Month'], as_index=False)['Movie'].count().sort_values(by='Movie', ascending=False)
m_by_month
```

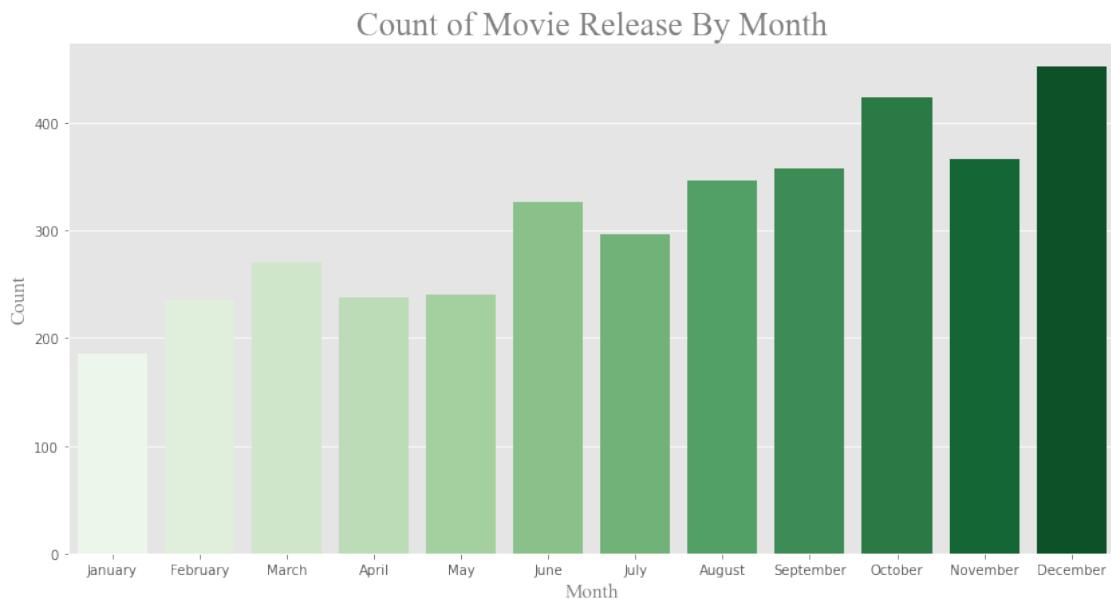
Let's first start by determining which months see the most movie releases.

[43]:

|    | Month     | Movie |
|----|-----------|-------|
| 2  | December  | 452   |
| 10 | October   | 424   |
| 9  | November  | 366   |
| 11 | September | 358   |
| 1  | August    | 346   |
| 6  | June      | 327   |
| 5  | July      | 296   |
| 7  | March     | 270   |
| 8  | May       | 241   |
| 0  | April     | 238   |
| 3  | February  | 236   |
| 4  | January   | 186   |

[44]: #Plot the above findings in order by month.

```
plt.figure(figsize=(14,7))
ax7 = sns.countplot(x=imdb_budgets_df['Month'], palette='Greens',
                     order=['January', 'February', 'March', 'April', 'May',
                            'June', 'July', 'August', 'September', 'October', 'November', 'December'])
plt.xlabel('Month', fontdict = {'fontname': 'Times New Roman', 'color': 'gray',
                                'fontsize' : '15'})
plt.ylabel('Count', fontdict = {'fontname': 'Times New Roman', 'color': 'gray',
                                'fontsize' : '15'})
plt.title('Count of Movie Release By Month', fontdict = {'fontname': 'Times New
Roman', 'color': 'gray', 'fontsize' : '25'})
plt.savefig('CountbyMonth', dpi=300);
```



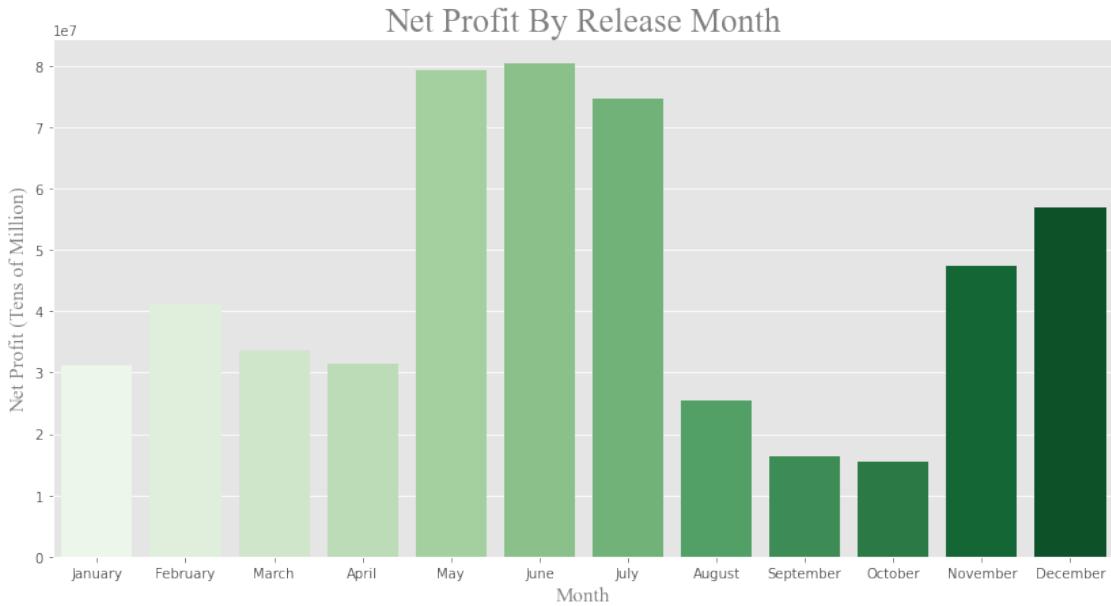
As you can see Decemeber and October lead the way in terms of sheer quantity of movies but does this suggest a higher level of profitability? Next we will look into the average net income by movie for each month.

```
[45]: #Once again group the movies by month, showing the average net profit for each.  
p_by_month = imdb_budgets_df.groupby('Month',  
    ↪as_index=False)[['Adjusted_Profit', 'Profit_Margin']].median().  
    ↪sort_values(by='Adjusted_Profit', ascending=False)  
p_by_month
```

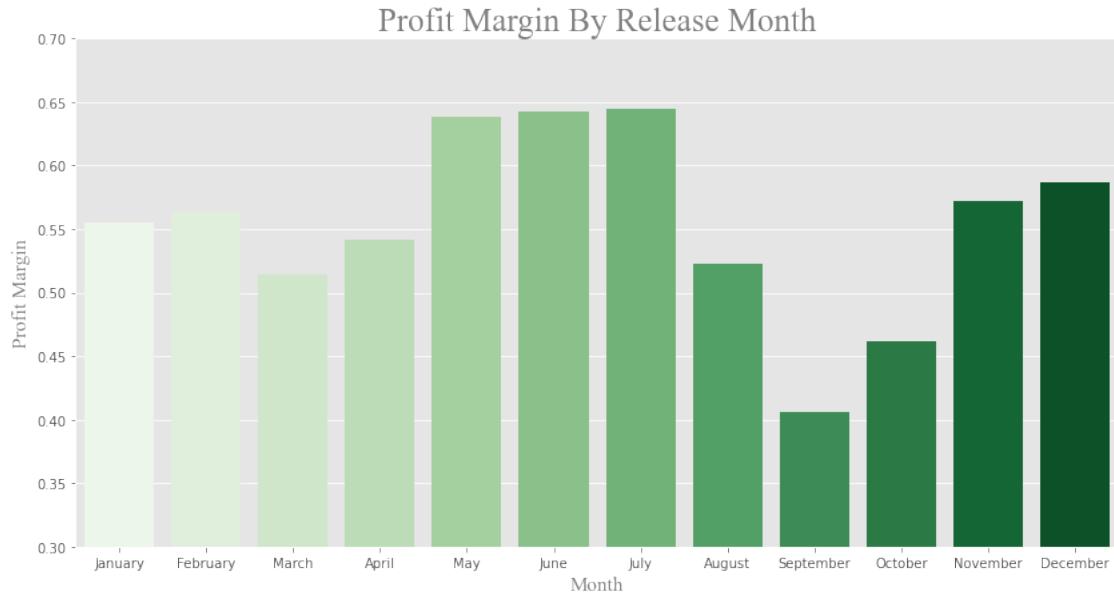
```
[45]:
```

|    | Month     | Adjusted_Profit | Profit_Margin |
|----|-----------|-----------------|---------------|
| 6  | June      | 80327640.00     | 0.64          |
| 8  | May       | 79372161.65     | 0.64          |
| 5  | July      | 74716618.14     | 0.64          |
| 2  | December  | 56823086.46     | 0.59          |
| 9  | November  | 47476647.51     | 0.57          |
| 3  | February  | 41089454.38     | 0.56          |
| 7  | March     | 33645813.78     | 0.51          |
| 0  | April     | 31435638.57     | 0.54          |
| 4  | January   | 31132342.98     | 0.56          |
| 1  | August    | 25383311.33     | 0.52          |
| 11 | September | 16430952.78     | 0.41          |
| 10 | October   | 15579534.04     | 0.46          |

```
[46]: #Plot your above findings in order by month.  
plt.figure(figsize=(14,7))  
ax8 = sns.barplot(x=p_by_month['Month'], y=p_by_month['Adjusted_Profit'],  
    ↪palette='Greens',  
    ↪order=['January', 'February', 'March', 'April', 'May',  
    ↪'June', 'July', 'August', 'September', 'October', 'November', 'December'])  
plt.xlabel('Month', fontdict = {'fontname': 'Times New Roman', 'color': 'gray',  
    ↪'fontsize' : '15'})  
plt.ylabel('Net Profit (Tens of Million)', fontdict = {'fontname': 'Times New  
    ↪Roman', 'color': 'gray', 'fontsize' : '15'})  
plt.title('Net Profit By Release Month', fontdict = {'fontname': 'Times New  
    ↪Roman', 'color': 'gray', 'fontsize' : '25'})  
plt.savefig('ProfitbyMonth', dpi=300);
```



```
[47]: plt.figure(figsize=(14,7))
ax9 = sns.barplot(x=p_by_month['Month'], y=p_by_month['Profit_Margin'], palette='Greens',
                    order=['January', 'February', 'March', 'April', 'May',
                           'June', 'July', 'August', 'September', 'October', 'November', 'December'])
plt.xlabel('Month', fontdict = {'fontname': 'Times New Roman', 'color': 'gray',
                                'fontsize' : '15'})
plt.ylabel('Profit Margin', fontdict = {'fontname': 'Times New Roman', 'color': 'gray',
   'fontsize' : '15'})
plt.title('Profit Margin By Release Month', fontdict = {'fontname': 'Times New Roman', 'color': 'gray', 'fontsize' : '25'})
plt.ylim(0.3, 0.7)
plt.savefig('MarginByMonth', dpi=300);
```



Interestingly, May, June and July shoot to the top in terms of both median net profit and profit margin. It appears that the summer months tend to result in greater success, perhaps as a result of an influx of children and their parents during summer break. Now as previously mentioned, let's dig a little further and see which genre tends to do the best in which month.

```
[49]: #Convert the Release Date field to type datetime
#Add a new column called month, displaying only the month from the release date.
genre_budgets_df['Release Date'] = pd.to_datetime(genre_budgets_df['Release Date'])
genreDate = [x.strftime('%B') for x in genre_budgets_df['Release Date']]
genre_budgets_df['Month'] = genreDate
```

```
[50]: #Create a new table called month_genre consisting of Genre, Month, Net Profit, and Release Date
month_genre = genre_budgets_df[['Genre', 'Month', 'Adjusted_Profit', 'Release Date']]
#Group by Genre and Month, displaying the average Net Profit for each combination.
month_genre = month_genre.groupby(['Genre', 'Month'], as_index=False)[['Adjusted_Profit']].mean().sort_values(by='Adjusted_Profit', ascending=False)
```

```
[51]: #Slice the top six most profitable genres from above.
Adventure_df = month_genre.loc[month_genre['Genre'].str.contains('Adventure')]
Action_df = month_genre.loc[month_genre['Genre'].str.contains('Action')]
Comedy_df = month_genre.loc[month_genre['Genre'].str.contains('Comedy')]
Drama_df = month_genre.loc[month_genre['Genre'].str.contains('Drama')]
```

```
Scifi_df = month_genre.loc[month_genre['Genre'].str.contains('Sci-Fi')]
Animation_df = month_genre.loc[month_genre['Genre'].str.contains('Animation')]
```

[52]: #Concatenate the six new tables into one new table.

```
genre_concat = [Adventure_df, Action_df, Comedy_df, Drama_df, Scifi_df, Animation_df]
month_genre_df = pd.concat(genre_concat)
```

[53]: #Create a table of the months in order.

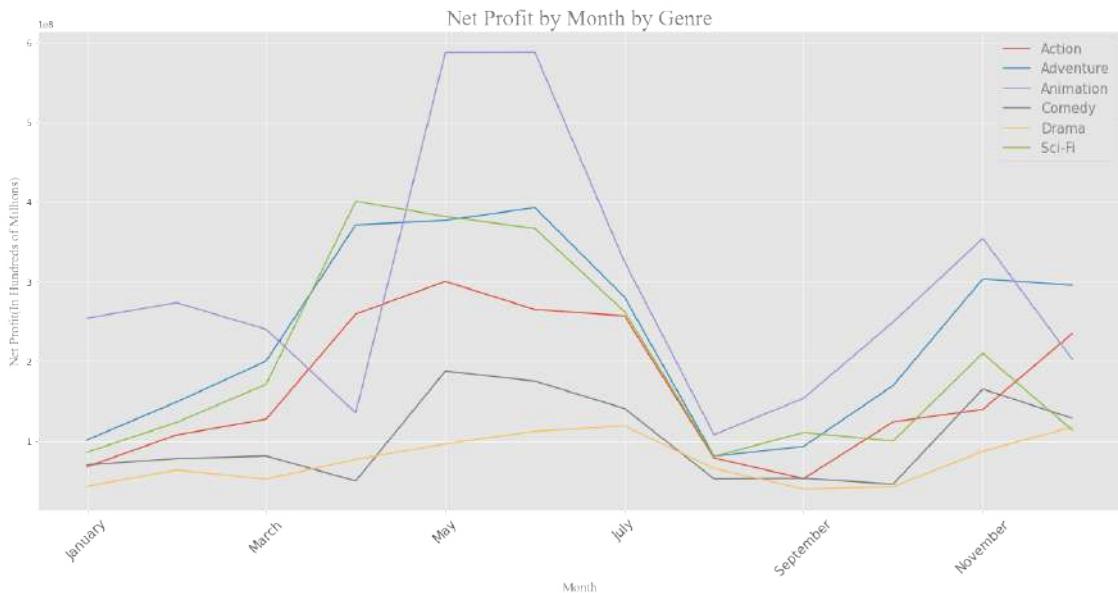
```
months_in_order = ['January', 'February', 'March', 'April', 'May', 'June', 'July', 'August', 'September', 'October', 'November', 'December']
#Create a pivot table of month_genre_df, use the month_in_order table to reindex the pivot table.
month_genre_pivoted = month_genre_df.pivot(index='Month', columns='Genre', values='Adjusted_Profit').reindex(months_in_order)
```

[54]: month\_genre\_pivoted

|           | Genre        | Action       | Adventure    | Animation    | Comedy       | Drama | \ |
|-----------|--------------|--------------|--------------|--------------|--------------|-------|---|
| Month     |              |              |              |              |              |       |   |
| January   | 67911226.86  | 101480251.68 | 254304586.21 | 70321717.64  | 43539017.01  |       |   |
| February  | 107741220.58 | 149172991.22 | 273699863.40 | 78129901.96  | 63807537.49  |       |   |
| March     | 127548996.11 | 200474749.59 | 240295152.35 | 81411129.63  | 52348133.09  |       |   |
| April     | 259392394.58 | 371426341.09 | 135514583.52 | 50050513.61  | 77199294.63  |       |   |
| May       | 300431780.23 | 376946029.72 | 587476204.76 | 187839907.64 | 96590740.22  |       |   |
| June      | 265101499.32 | 392963586.66 | 587763663.68 | 175416615.42 | 112382070.55 |       |   |
| July      | 257293527.76 | 280812330.30 | 325184250.83 | 140927144.14 | 119198995.62 |       |   |
| August    | 78993517.46  | 81128041.19  | 108115881.94 | 52702618.10  | 65637106.34  |       |   |
| September | 52980175.19  | 93388465.69  | 153847514.52 | 53288686.20  | 40194497.00  |       |   |
| October   | 124257794.43 | 169896169.96 | 249582645.96 | 46177500.88  | 42992650.53  |       |   |
| November  | 139749410.88 | 303503861.24 | 354381890.29 | 165340406.04 | 87265604.53  |       |   |
| December  | 235113158.91 | 295732977.48 | 202553251.30 | 128699177.32 | 117758948.19 |       |   |

|           | Genre        | Sci-Fi |
|-----------|--------------|--------|
| Month     |              |        |
| January   | 86131136.28  |        |
| February  | 123463145.04 |        |
| March     | 171335731.24 |        |
| April     | 400992743.36 |        |
| May       | 381838680.03 |        |
| June      | 366873462.47 |        |
| July      | 262513716.23 |        |
| August    | 80812011.13  |        |
| September | 110804792.63 |        |
| October   | 100120506.83 |        |
| November  | 210336333.85 |        |
| December  | 113695722.89 |        |

```
[63]: #Visualize the top 6 most profitable genre's by month
ax10 = month_genre_pivoted.plot(kind='line', figsize=(22, 10), rot=0)
plt.legend(labelcolor='grey', loc='best', prop={'size': 15})
plt.xlabel('Month', fontdict = {'fontname': 'Times New Roman', 'color': 'gray', 'fontsize' : '15'})
plt.ylabel('Net Profit(In Hundreds of Millions)', fontdict = {'fontname': 'Times New Roman', 'color': 'gray', 'fontsize' : '15'})
plt.title('Net Profit by Month by Genre', fontdict = {'fontname': 'Times New Roman', 'color': 'gray', 'fontsize' : '25'})
plt.xticks(fontsize=15, rotation=45)
plt.savefig('ProfitbyMonthbyGenre', dpi=300);
```



We can see that each genre follows the same basic pattern, with the summer months proving to be the most profitable time to release a movie. Some further analysis shows that releasing an animation movie in particular during the summer months will have the greatest potential for high net profits. On the other hand drama, although fluctuates slightly with the months, tends to have no impact based on release date. When considering what aspects go into creating a successful movie, it's clear that one must take into account the impact of a well timed release date.

**Question 3 Conclusion:** We recommend that our Company release the bulk of their movies, especially Animation, during the summer months. Adventure, Drama and Comedy movies would see similar success if released in November, but the recommendation remains to focus on summer.

## 5 Question 4: Now that we've got a better understanding of what attributes to a successful movie, which actors and directors tend to add the most value?

In this section we are going to take a look at the average net profit across all movies. From there we want to determine which actors and directors consistently appear in movies where the net profit substantially exceeds the average. We will represent this in a field called Value Above Replacement(VAR). To further simplify this concept; if across all movies the average net profit is 100 dollars and the average net profit of movies from 'Actor: X' is 200 dollars he/she would have a VAR of 2. This number represents X times over the average. To eliminate outliers we will look at actors who appear in 10 or more movies and directors who work in 5 or more.

```
[64]: #Similar to the imdb_budget_df table let's start by adjusting for inflation.  
actors_df['Production Budget'] = (((2020-actors_df['Year'])*.  
    ↪0322)+1)*actors_df['Production Budget']  
actors_df['Worldwide Gross'] = (((2020-actors_df['Year'])*.  
    ↪0322)+1)*actors_df['Worldwide Gross']  
actors_df['Domestic Gross'] = (((2020-actors_df['Year'])*.  
    ↪0322)+1)*actors_df['Domestic Gross']
```

```
[65]: #Calculate Net Profit and Profit Margin  
actors_df['Net Profit'] = actors_df['Worldwide Gross'] - actors_df['Production  
    ↪Budget']  
actors_df['Profit Margin'] = actors_df['Net Profit'] / actors_df['Worldwide  
    ↪Gross']
```

```
[66]: #Let's filter the actors_df table to only include actors that appeared in 10 or  
    ↪more movies  
actor_counts = actors_df['value'].value_counts()  
actor_list = actor_counts[actor_counts >= 10].index.tolist()  
actors_df = actors_df[actors_df['value'].isin(actor_list)]
```

```
[67]: #Calculate VAR, which is the average Net Profit by actor divided by average Net  
    ↪Profit for all movies.  
actor_total = actors_df.groupby(['value'], as_index=False)['Net Profit'].  
    ↪mean().sort_values(by='Net Profit', ascending=False)  
actor_total['VAR'] = (actor_total['Net Profit']/actor_total['Net Profit']).  
    ↪mean()
```

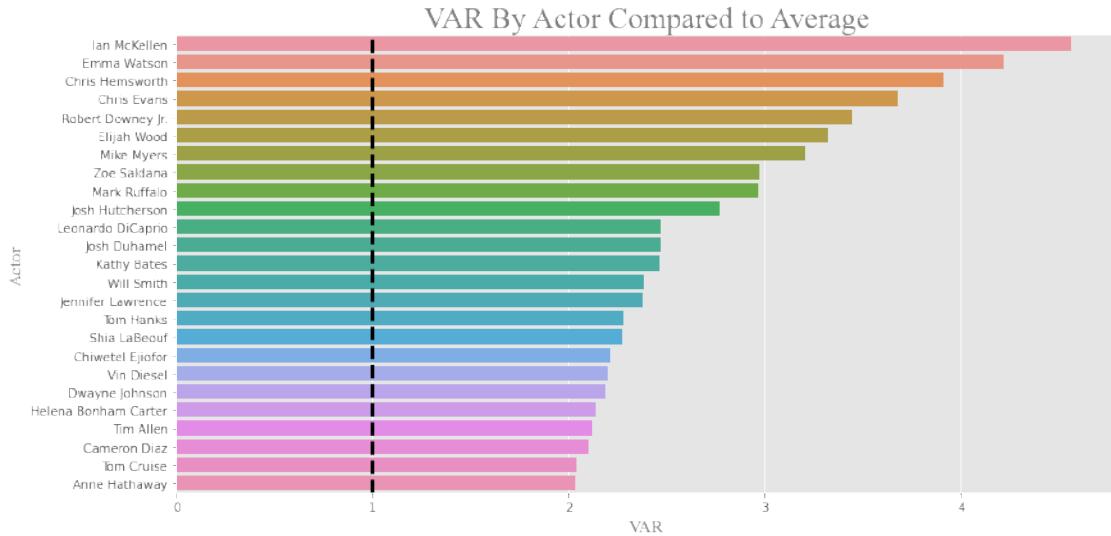
```
[68]: #Create new table consisting of top 25 actors by VAR.  
top_actors = actor_total.head(25)  
top_actors
```

```
[68]:
```

|     | value        | Net Profit   | VAR  |
|-----|--------------|--------------|------|
| 113 | Ian McKellen | 642641141.05 | 4.56 |
| 88  | Emma Watson  | 594070330.59 | 4.22 |

|     |                      |              |      |
|-----|----------------------|--------------|------|
| 48  | Chris Hemsworth      | 550993070.74 | 3.91 |
| 47  | Chris Evans          | 518397913.83 | 3.68 |
| 262 | Robert Downey Jr.    | 484884995.15 | 3.44 |
| 82  | Elijah Wood          | 468414890.65 | 3.33 |
| 227 | Mike Myers           | 451615981.41 | 3.21 |
| 324 | Zoe Saldana          | 418413981.69 | 2.97 |
| 205 | Mark Ruffalo         | 418051684.80 | 2.97 |
| 166 | Josh Hutcherson      | 389946768.85 | 2.77 |
| 197 | Leonardo DiCaprio    | 347929775.33 | 2.47 |
| 164 | Josh Duhamel         | 347668686.44 | 2.47 |
| 178 | Kathy Bates          | 347201332.37 | 2.47 |
| 316 | Will Smith           | 336002549.53 | 2.39 |
| 138 | Jennifer Lawrence    | 334744177.49 | 2.38 |
| 299 | Tom Hanks            | 320791739.32 | 2.28 |
| 285 | Shia LaBeouf         | 320522135.54 | 2.28 |
| 45  | Chiwetel Ejiofor     | 311862722.21 | 2.21 |
| 308 | Vin Diesel           | 309819051.08 | 2.20 |
| 78  | Dwayne Johnson       | 308538514.10 | 2.19 |
| 108 | Helena Bonham Carter | 301712229.56 | 2.14 |
| 296 | Tim Allen            | 298679367.49 | 2.12 |
| 36  | Cameron Diaz         | 295720384.55 | 2.10 |
| 298 | Tom Cruise           | 287290600.79 | 2.04 |
| 13  | Anne Hathaway        | 286762937.70 | 2.04 |

```
[70]: #Plot above finding and label the average of 1 with a black line.
plt.figure(figsize=(14,7))
ax11 = sns.barplot(x=top_actors['VAR'], y=top_actors['value'])
plt.axvline(1, ls='--', color='black', linewidth=3)
plt.xlabel('VAR', fontdict = {'fontname': 'Times New Roman', 'color': 'gray', 'fontsize' : '15'})
plt.ylabel('Actor', fontdict = {'fontname': 'Times New Roman', 'color': 'gray', 'fontsize' : '15'})
plt.title('VAR By Actor Compared to Average', fontdict = {'fontname': 'Times New Roman', 'color': 'gray', 'fontsize' : '25'})
plt.savefig('VARActor', dpi=300);
```



Wow, from this list we can see that all of these actors consistently appear in very profitable movies; anywhere from two times the norm to four and a half times the norm. When casting a movie this is a good short-list from where to start making calls.

```
[71]: #Adjust directors table for inflation.
directors_df['Production Budget'] = (((2020-directors_df['Year'])*.
    ↪0322)+1)*directors_df['Production Budget']
directors_df['Worldwide Gross'] = (((2020-directors_df['Year'])*.
    ↪0322)+1)*directors_df['Worldwide Gross']
directors_df['Domestic Gross'] = (((2020-directors_df['Year'])*.
    ↪0322)+1)*directors_df['Domestic Gross']
```

```
[72]: #Calucalte Net Profit and Profit Margin.
directors_df['Net Profit'] = directors_df['Worldwide Gross'] - ↪
    ↪directors_df['Production Budget']
directors_df['Profit Margin'] = directors_df['Net Profit'] / ↪
    ↪directors_df['Worldwide Gross']
```

```
[73]: #Let's filter the actors_df table to only include actors that appeared in 5 or ↪
    ↪more movies.
director_counts = directors_df['value'].value_counts()
director_list = director_counts[director_counts >= 5].index.tolist()
directors_df = directors_df[directors_df['value'].isin(director_list)]
```

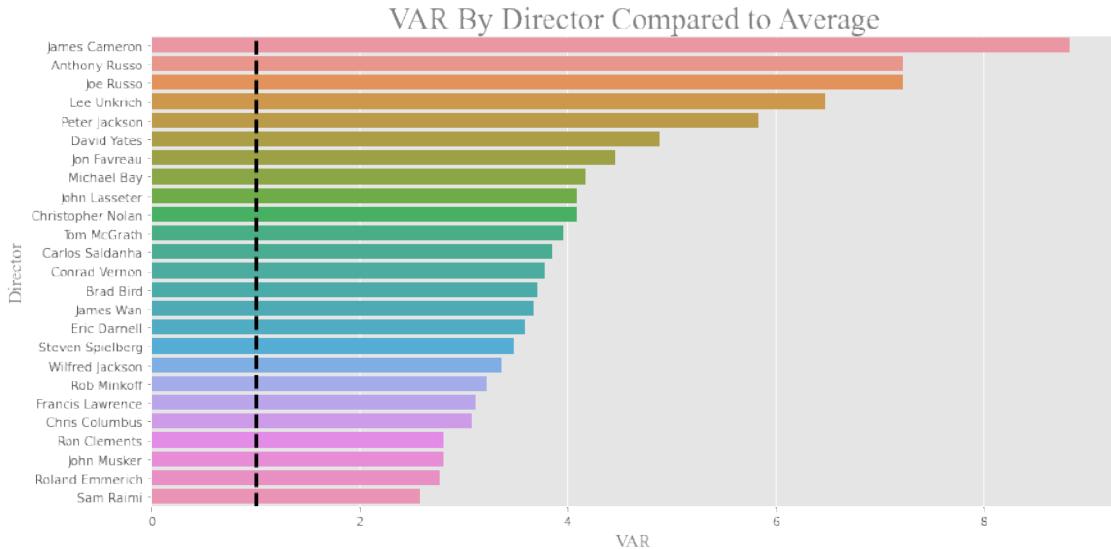
```
[74]: #Calculate VAR, which is the average Net Profit by director divided by average ↪
    ↪Net Profit for all movies.
director_total = directors_df.groupby(['value'], as_index=False)['Net Profit'].
    ↪mean().sort_values(by='Net Profit', ascending=False)
```

```
director_total['VAR'] = (director_total['Net Profit']/actor_total['Net Profit']).  
                         ↪mean()
```

```
[75]: #Create new table consisting of top 25 directors by VAR.  
top_directors = director_total.head(25)  
top_directors
```

```
[75]:          value  Net Profit   VAR  
78      James Cameron 1244750157.55 8.84  
11      Anthony Russo 1017389415.62 7.22  
89      Joe Russo 1017389415.62 7.22  
115     Lee Unkrich 912067911.25 6.48  
148     Peter Jackson 821878024.53 5.84  
50      David Yates 688135205.04 4.89  
104     Jon Favreau 628704113.52 4.46  
129     Michael Bay 588804626.49 4.18  
96      John Lasseter 577254528.66 4.10  
31     Christopher Nolan 576508914.30 4.09  
194     Tom McGrath 558026757.25 3.96  
27     Carlos Saldanha 542327603.19 3.85  
34     Conrad Vernon 533554799.18 3.79  
19      Brad Bird 522918604.82 3.71  
82      James Wan 517843475.89 3.68  
58      Eric Darnell 506570978.60 3.60  
188     Steven Spielberg 490403244.69 3.48  
200     Wilfred Jackson 473675805.64 3.36  
160     Rob Minkoff 453631830.01 3.22  
62     Francis Lawrence 439117499.61 3.12  
29     Chris Columbus 434315443.48 3.08  
171     Ron Clements 396185896.16 2.81  
101     John Musker 396185896.16 2.81  
169     Roland Emmerich 391218701.44 2.78  
175     Sam Raimi 364101893.22 2.59
```

```
[76]: #Plot above finding and label the average of 1 with a black line.  
plt.figure(figsize=(14,7))  
ax12 = sns.barplot(x=top_directors['VAR'], y=top_directors['value'])  
plt.axvline(1, ls='--', color='black', linewidth=3)  
plt.xlabel('VAR', fontdict = {'fontname': 'Times New Roman', 'color': 'gray',  
                           ↪'fontsize' : '15'})  
plt.ylabel('Director', fontdict = {'fontname': 'Times New Roman', 'color':  
                           ↪'gray', 'fontsize' : '15'})  
plt.title('VAR By Director Compared to Average', fontdict = {'fontname': 'Times  
                           ↪New Roman', 'color': 'gray', 'fontsize' : '25'})  
plt.savefig('VARDirector', dpi=300);
```



It appears that the most significant value added comes from the directors chair. James Cameron movies on average make almost nine times the amount of the average movie, this emphasizes what great leadership represents on a set. If we wanted to further investigate which actors and directors make the most impact it would be important to determine which genre of movies they appear in most.

**Question 4 Conclusion:** We recommend that our Comapany focus their cast and crew search to individuals who consistently score at least 1.0 on the VAR score. We can, with a high level of confidence, conclude that these individuals will elevate the overall production.

## 6 Question 5: How much should you spend on a movie to win an Oscar?

In order to answer this question we'll first need to join the `imdb_budgets_df` dataframe and the `awards_df` dataframe. As there may be movies with duplicate titles, we set the indices of both dataframes to the movie name and year so that matching data is correctly joined.

```
[77]: imdb_budgets_df.set_index(['Movie', 'Year'], inplace=True)
awards_df.set_index(['film_name', 'film_year'], inplace=True)
```

```
[78]: budgets_and_awards = imdb_budgets_df.join(awards_df, how='inner', on=['Movie', ↴ 'Year'])
budgets_and_awards.head()
```

| Movie         | Year | IMDb Rating | Runtime | Genre                            |
|---------------|------|-------------|---------|----------------------------------|
| Avatar        | 2009 | 7.80        | PG-13   | 162 [Action, Adventure, Fantasy] |
| Black Panther | 2018 | 7.30        | PG-13   | 134 [Action, Adventure, Sci-Fi]  |
| Titanic       | 1997 | 7.80        | PG-13   | 194 [Drama, Romance]             |

|                 |      |      |       |     |                                |
|-----------------|------|------|-------|-----|--------------------------------|
| The Dark Knight | 2008 | 9.00 | PG-13 | 152 | [Action, Crime, Drama]         |
| Toy Story 4     | 2019 | 7.80 | G     | 100 | [Animation, Adventure, Comedy] |

| Movie           | Year | Release Date | Production Budget | Domestic Gross | \ |
|-----------------|------|--------------|-------------------|----------------|---|
| Avatar          | 2009 | 2009-12-17   | 237000000         | 760507625      |   |
| Black Panther   | 2018 | 2018-02-13   | 200000000         | 700059566      |   |
| Titanic         | 1997 | 1997-12-18   | 200000000         | 659363944      |   |
| The Dark Knight | 2008 | 2008-07-11   | 185000000         | 533720947      |   |
| Toy Story 4     | 2019 | 2019-06-20   | 200000000         | 434038008      |   |

| Movie           | Year | Worldwide Gross | Profit     | Profit_Margin | \ |
|-----------------|------|-----------------|------------|---------------|---|
| Avatar          | 2009 | 2788701337      | 2551701337 | 0.92          |   |
| Black Panther   | 2018 | 1346103376      | 1146103376 | 0.85          |   |
| Titanic         | 1997 | 2208208395      | 2008208395 | 0.91          |   |
| The Dark Knight | 2008 | 1000742751      | 815742751  | 0.82          |   |
| Toy Story 4     | 2019 | 1073394813      | 873394813  | 0.81          |   |

| Movie           | Year | Adjusted_Budget | Adjusted_Profit | Month    | awards_won | \ |
|-----------------|------|-----------------|-----------------|----------|------------|---|
| Avatar          | 2009 | 320945400.00    | 3455513950.57   | December | 3          |   |
| Black Panther   | 2018 | 212880000.00    | 1219912433.41   | February | 3          |   |
| Titanic         | 1997 | 348120000.00    | 3495487532.34   | December | 11         |   |
| The Dark Knight | 2008 | 256484000.00    | 1130945749.99   | July     | 2          |   |
| Toy Story 4     | 2019 | 206440000.00    | 901518125.98    | June     | 1          |   |

| Movie           | Year | awards_nominated | win_rate |
|-----------------|------|------------------|----------|
| Avatar          | 2009 | 9                | 0.33     |
| Black Panther   | 2018 | 7                | 0.43     |
| Titanic         | 1997 | 14               | 0.79     |
| The Dark Knight | 2008 | 8                | 0.25     |
| Toy Story 4     | 2019 | 2                | 0.50     |

We've successfully joined the two dataframes. Let's filter the dataframe to include movies where the profit is greater than 0.

```
[79]: nominated_movies_df = budgets_and_awards.loc[budgets_and_awards['Profit'] > 0]
```

```
[82]: plt.figure(figsize=(16,6))
sns.boxplot(x='Adjusted_Budget', data=nominated_movies_df, showfliers=False, color='powderblue')
sns.stripplot(x='Adjusted_Budget', data=nominated_movies_df)
plt.ticklabel_format(axis='x', style='sci', scilimits=(6,6))
plt.xticks(fontsize=12)
```

```

plt.xlabel('Movie Budgets Adjusted for Inflation (Millions of Dollars)',  

    fontdict = {'fontname': 'Times New Roman', 'color': 'gray', 'fontsize' :  

    '15'});  

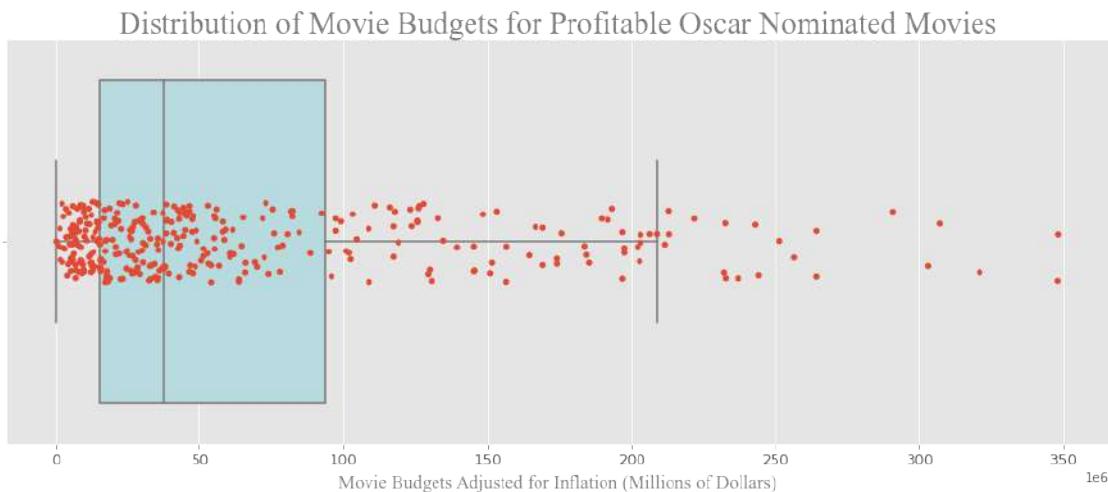
plt.title('Distribution of Movie Budgets for Profitable Oscar Nominated  

    Movies', fontdict = {'fontname': 'Times New Roman', 'color': 'gray',  

    'fontsize' : '25'})  

plt.savefig('Oscar_Nominated', dpi=300);

```



```
[83]: nominated_movies_df['Adjusted_Budget'].describe()
```

```

[83]: count      331.00
      mean     66479336.13
      std      72497186.73
      min      212790.00
      25%     15425660.00
      50%     37816500.00
      75%     93598000.00
      max     348300000.00
      Name: Adjusted_Budget, dtype: float64

```

By looking at the distribution of movie budgets we see that the majority of data is clustered in an area below \$100 million dollars.

We need to take this a step further as the above distribution includes movies that were nominated and won awards as well as movies that did not win awards. In order to properly answer our question we must win an Oscar.

We could filter by win rate and exclude those movies that did not win anything, however our data would still include movies that were nominated in a single category and won. This would skew the win rate as there would be several movies with a win rate of 100%. Let's take a look at the mean and median win rate to establish a threshold for award nominations.

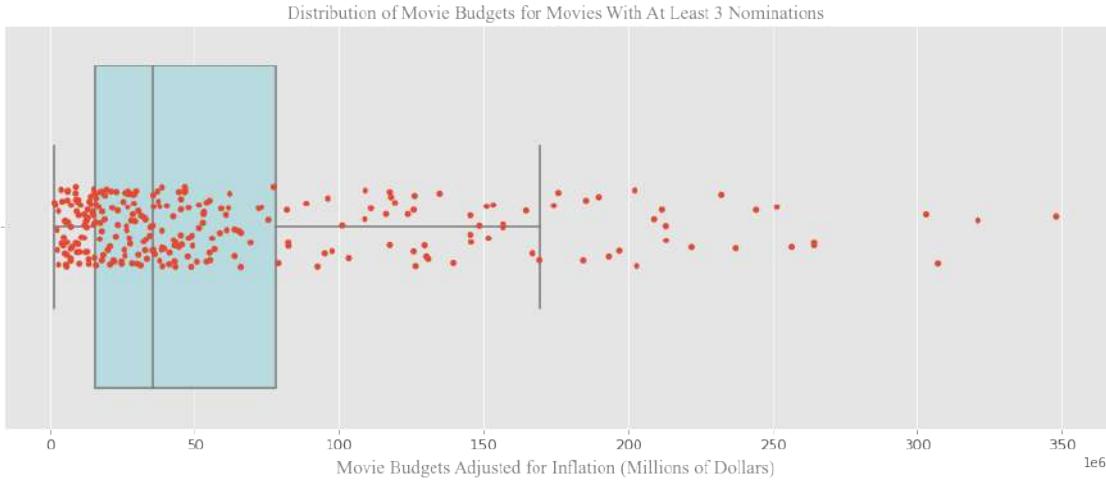
```
[84]: nominated_movies_df['win_rate'].describe()
#Let's be conservative for win rate and use the median win rate
#That means we would need to be nominated for at least 3 awards in order to win
→ 1 award.
```

```
[84]: count    330.00
      mean     0.45
      std      0.28
      min      0.00
      25%     0.25
      50%     0.39
      75%     0.60
      max      1.00
      Name: win_rate, dtype: float64
```

The mean win rate is 44.8% but as we mentioned is skewed by those movies with only 1 nomination. The median win rate is 39.2% which should be less skewed by the data and is a more conservative number. Using the median win rate of 39.2%, our movie would need to be nominated for at least 3 awards in order to get at least one win. 3 nominations will be the cutoff.

```
[85]: nominated_over_three = nominated_movies_df.
       ↪loc[nominated_movies_df['awards_nominated'] >= 3]
print(len(nominated_over_three))
plt.figure(figsize=(16,6))
sns.boxplot(x=nominated_over_three['Adjusted_Budget'], showfliers=False,
            ↪color='powderblue')
sns.stripplot(x='Adjusted_Budget', data=nominated_over_three)
plt.ticklabel_format(axis='x', style='sci', scilimits=(6,6))
plt.xticks(fontsize=12)
plt.xlabel('Movie Budgets Adjusted for Inflation (Millions of Dollars)',
           ↪fontdict = {'fontname': 'Times New Roman', 'color': 'gray', 'fontsize' :
           ↪'15'})
plt.title('Distribution of Movie Budgets for Movies With At Least 3
           ↪Nominations', fontdict = {'fontname': 'Times New Roman', 'color': 'gray',
           ↪'fontsize' : '15'})
plt.savefig('3_Nominations', dpi=300);
```

263



```
[86]: nominated_over_three['Adjusted_Budget'].describe()
```

```
[86]: count      263.00
mean      62404651.14
std       69126844.12
min      1224990.00
25%     15482900.00
50%     35465000.00
75%     78132000.00
max     348120000.00
Name: Adjusted_Budget, dtype: float64
```

It's important to note that the box plot of the `nominated_over_three` dataframe has shrunk! This means that our filter has decreased our interquartile range for the movie budget. Since this range is smaller there should be less variability in the middle of the data set. Since we have adjusted budgets that are extreme outliers, it is best to use the median as the primary measure of central tendency. The median adjusted budget for this data is \\$35,465,000.

**Question 5 Conclusion:** Our Company should spend at least \$35,465,000 in order to make an Oscar-winning movie.

*It is also worth noting that the 75th percentile of the adjusted budget for movies with at least three nominations is \$78,132,000. This is close to our recommendation of a \$82 million budget for a profitable movie with a profit margin of approximately 80%.*

## 7 Question 6: What impact, if any, does runtime and movie rating have on Net Profit, Profit Margin and IMDb rating?

Let's first start by analyzing the ratings. We want to include only movies rated G, PG, PG-13 or R.

```
[87]: rating_counts = imdb_budgets_df['Rating'].value_counts()
rating_list = rating_counts[rating_counts >= 50].index.tolist()
rating_df = imdb_budgets_df[imdb_budgets_df['Rating'].isin(rating_list)]
```

```
[88]: rating_df = rating_df.reset_index()
rating_df.head()
```

```
[88]:
```

|   | Movie                  | Year | IMDb | Rating | Runtime | \ |
|---|------------------------|------|------|--------|---------|---|
| 0 | Avengers: Endgame      | 2019 | 8.40 | PG-13  | 181     |   |
| 1 | Avatar                 | 2009 | 7.80 | PG-13  | 162     |   |
| 2 | Black Panther          | 2018 | 7.30 | PG-13  | 134     |   |
| 3 | Avengers: Infinity War | 2018 | 8.40 | PG-13  | 149     |   |
| 4 | Titanic                | 1997 | 7.80 | PG-13  | 194     |   |

|   | Genre                        | Release Date | Production Budget | \ |
|---|------------------------------|--------------|-------------------|---|
| 0 | [Action, Adventure, Drama]   | 2019-04-23   | 400000000         |   |
| 1 | [Action, Adventure, Fantasy] | 2009-12-17   | 237000000         |   |
| 2 | [Action, Adventure, Sci-Fi]  | 2018-02-13   | 200000000         |   |
| 3 | [Action, Adventure, Sci-Fi]  | 2018-04-25   | 300000000         |   |
| 4 | [Drama, Romance]             | 1997-12-18   | 200000000         |   |

|   | Domestic Gross | Worldwide Gross | Profit     | Profit Margin | \ |
|---|----------------|-----------------|------------|---------------|---|
| 0 | 858373000      | 2797800564      | 2397800564 | 0.86          |   |
| 1 | 760507625      | 2788701337      | 2551701337 | 0.92          |   |
| 2 | 700059566      | 1346103376      | 1146103376 | 0.85          |   |
| 3 | 678815482      | 2048359754      | 1748359754 | 0.85          |   |
| 4 | 659363944      | 2208208395      | 2008208395 | 0.91          |   |

|   | Adjusted_Budget | Adjusted_Profit | Month    |
|---|-----------------|-----------------|----------|
| 0 | 412880000.00    | 2475009742.16   | April    |
| 1 | 320945400.00    | 3455513950.57   | December |
| 2 | 212880000.00    | 1219912433.41   | February |
| 3 | 319320000.00    | 1860954122.16   | April    |
| 4 | 348120000.00    | 3495487532.34   | December |

```
[89]: #Count the total number of movies and group by month.
rating_count = rating_df.groupby(['Rating'], as_index=False)['Movie'].count().
    ↪sort_values(by='Movie', ascending=False)
rating_count
```

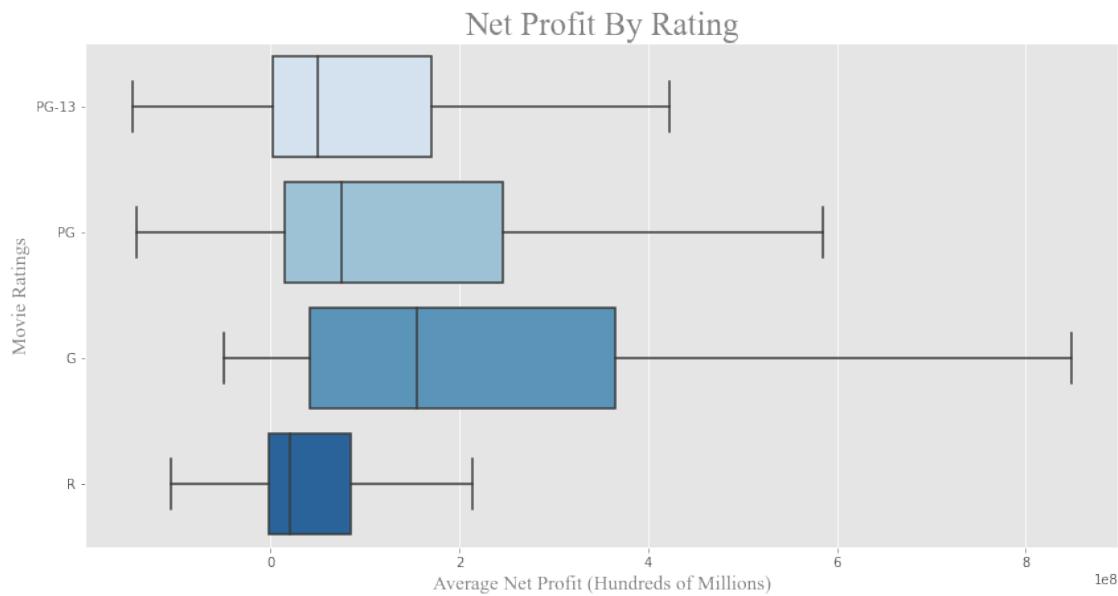
```
[89]:
```

|   | Rating | Movie |
|---|--------|-------|
| 3 | R      | 1631  |
| 2 | PG-13  | 1339  |
| 1 | PG     | 590   |
| 0 | G      | 93    |

```
[90]: #Group by Rating let's determine which has the highest median net profit and
      ↪profit margin.
rating_df2 = rating_df.groupby(['Rating'], as_index=False)[['Adjusted_Profit', ↪
      ↪'Profit_Margin', 'IMDb']].median().sort_values(by='Adjusted_Profit', ↪
      ↪ascending=False)
rating_df2
```

|   | Rating | Adjusted_Profit | Profit_Margin | IMDb |
|---|--------|-----------------|---------------|------|
| 0 | G      | 154376810.04    | 0.76          | 7.10 |
| 1 | PG     | 75404192.25     | 0.62          | 6.50 |
| 2 | PG-13  | 49565772.61     | 0.55          | 6.30 |
| 3 | R      | 20402474.98     | 0.51          | 6.60 |

```
[91]: # Plot your above findings
plt.figure(figsize=(14,7))
ax13 = sns.boxplot( y=rating_df["Rating"], x=rating_df["Adjusted_Profit"], ↪
      ↪showfliers=False, palette='Blues')
plt.xlabel('Average Net Profit (Hundreds of Millions)', fontdict = {'fontname': ↪
      ↪'Times New Roman', 'color': 'gray', 'fontsize' : '15'})
plt.ylabel('Movie Ratings', fontdict = {'fontname': 'Times New Roman', 'color': ↪
      ↪'gray', 'fontsize' : '15'})
plt.title('Net Profit By Rating', fontdict = {'fontname': 'Times New Roman', ↪
      ↪'color': 'gray', 'fontsize' : '25'})
plt.savefig('ProfitbyRating', dpi=300);
```



As you can see, G and PG rated movies tend to perform best and account for the smallest market share. This, like the animation genre, is another opportunity to enter the market in a highly

profitable arena with fewer competitors. It would be interesting to see a breakdown of total net profit by genre by rating to get a better idea of which rating and genres go best together.

```
[92]: # First drop the rating column from genre_budgets_df and genre from rating_df
genre_rating_df = genre_budgets_df.drop(['Rating'], axis=1)
rating_df = rating_df.drop(['Genre'], axis=1)
```

```
[93]: # Merge the genre_rating_df table and rating_df table
genre_rating_df = pd.merge(genre_rating_df, rating_df)
```

```
[94]: #Slice the top six most profitable genres.
Adv_df = genre_rating_df.loc[genre_rating_df['Genre'].str.contains('Adventure')]
Act_df = genre_rating_df.loc[genre_rating_df['Genre'].str.contains('Action')]
Com_df = genre_rating_df.loc[genre_rating_df['Genre'].str.contains('Comedy')]
Dra_df = genre_rating_df.loc[genre_rating_df['Genre'].str.contains('Drama')]
Sci_df = genre_rating_df.loc[genre_rating_df['Genre'].str.contains('Sci-Fi')]
Ani_df = genre_rating_df.loc[genre_rating_df['Genre'].str.contains('Animation')]

genre_concat = [Adv_df, Act_df, Com_df, Dra_df, Sci_df, Ani_df]
genre_rating = pd.concat(genre_concat)
```

```
[95]: # Create a pivot table from genre_rating
gr_df = genre_rating.groupby(['Genre', 'Rating'],
                           as_index=False)[['Adjusted_Profit']].sum().sort_values(by='Adjusted_Profit',
                           ascending=False)
gr_pivoted = gr_df.pivot(index='Genre', columns='Rating',
                           values='Adjusted_Profit')
```

```
[96]: # Preview the table.
gr_pivoted
```

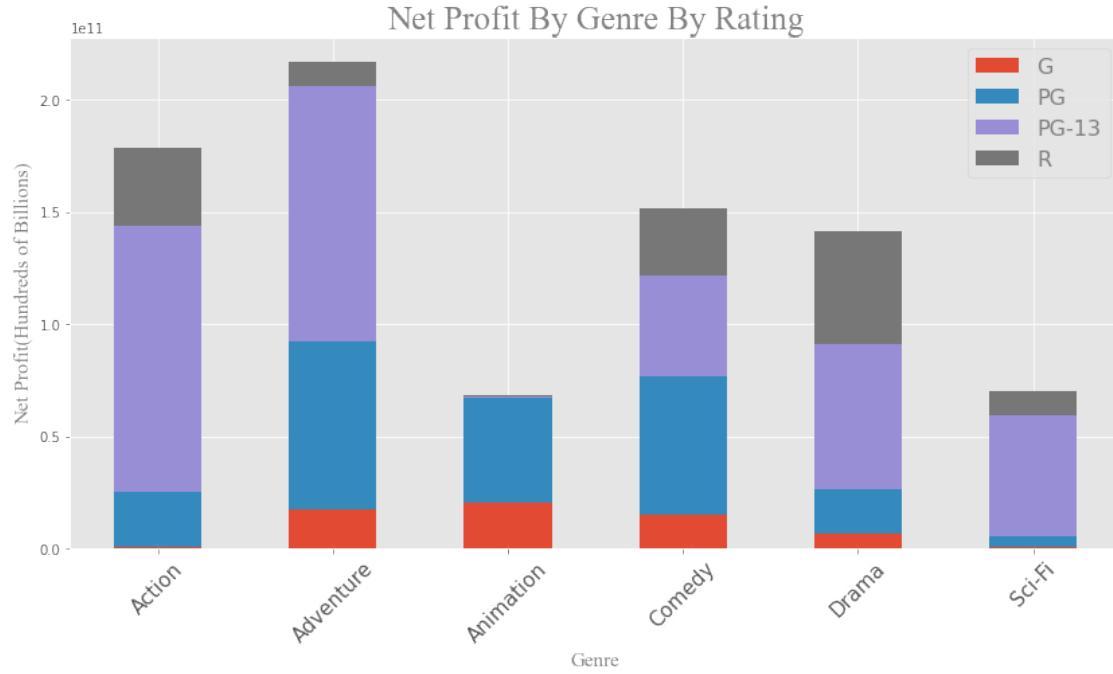
| Rating    | G              | PG             | PG-13           | R              |
|-----------|----------------|----------------|-----------------|----------------|
| Genre     |                |                |                 |                |
| Action    | 476713962.52   | 24806502581.61 | 118476527154.35 | 34527820240.94 |
| Adventure | 17497561206.41 | 74656830471.14 | 114180501731.83 | 10663312187.82 |
| Animation | 20451774875.23 | 46792514260.78 | 682637577.33    | 120368587.97   |
| Comedy    | 14989898831.46 | 61733858474.80 | 44722618139.99  | 30095649966.62 |
| Drama     | 6452247472.37  | 19785801203.02 | 64695667306.22  | 50557666303.54 |
| Sci-Fi    | 575199818.94   | 4693467863.02  | 54045363674.82  | 11072810424.26 |

```
[99]: # Plot the above findings.
ax14 = gr_pivoted.plot(kind='bar', stacked=True, figsize=(14,7))
plt.legend(labelcolor='grey', prop={'size': 16})
plt.xlabel('Genre', fontdict = {'fontname': 'Times New Roman', 'color': 'gray',
                               'fontsize' : '15'})
plt.ylabel('Net Profit(Hundreds of Billions)', fontdict = {'fontname': 'Times New Roman', 'color': 'gray', 'fontsize' : '15'})
```

```

plt.title('Net Profit By Genre By Rating', fontdict = {'fontname': 'Times New Roman', 'color': 'gray', 'fontsize' : '25'})
plt.xticks(fontsize=15, rotation=45)
plt.savefig('ProfitbyGenrebyRating');

```



As one could have probably guessed, animation is almost entirely made up of G and PG rated movies. We can see that for most other genres, the bulk of their total net profits come from PG-13 rated movies. From this we can focus on which rating to aim for in each genre to evoke the most success.

Now let's shift our focus to the film's runtime. Does movie length have an impact in terms of success?

```
[100]: # Create a new table with runtime, net profit and profit margin.
runtime_df = imdb_budgets_df[['Runtime', 'Adjusted_Profit', 'Profit_Margin']]
runtime_df
```

| Movie                  | Year | Runtime | Adjusted_Profit | Profit_Margin |
|------------------------|------|---------|-----------------|---------------|
| Avengers: Endgame      | 2019 | 181     | 2475009742.16   | 0.86          |
| Avatar                 | 2009 | 162     | 3455513950.57   | 0.92          |
| Black Panther          | 2018 | 134     | 1219912433.41   | 0.85          |
| Avengers: Infinity War | 2018 | 149     | 1860954122.16   | 0.85          |
| Titanic                | 1997 | 194     | 3495487532.34   | 0.91          |
| ...                    |      | ...     | ...             | ...           |
| The Misfits            | 1961 | 125     | 12179160.00     | 0.51          |

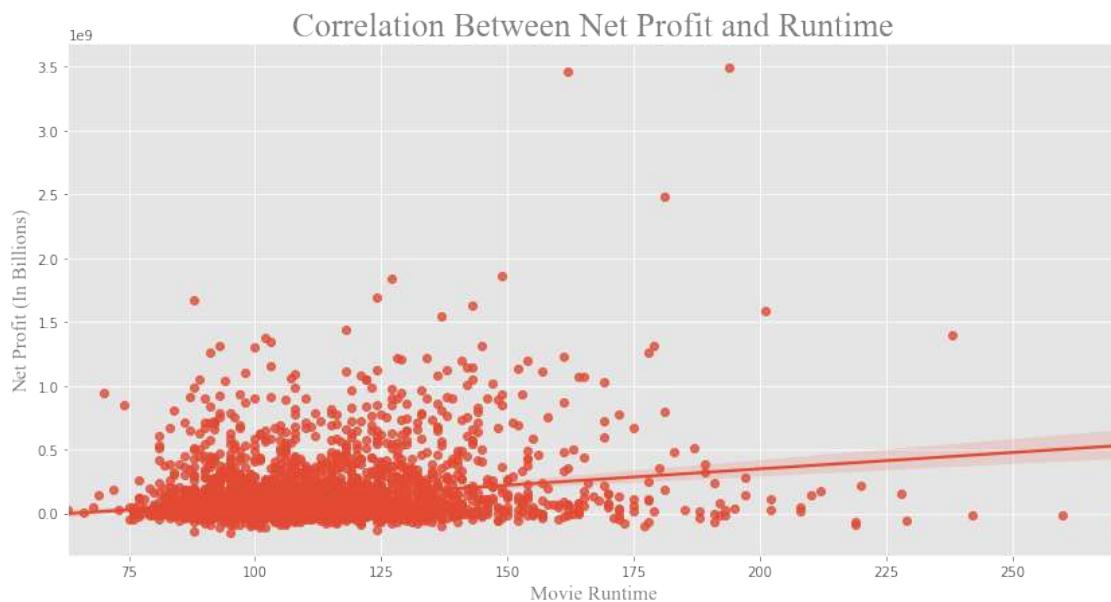
|                        |      |     |             |      |
|------------------------|------|-----|-------------|------|
| Judgment at Nuremberg  | 1961 | 179 | 20298600.00 | 0.70 |
| The Wrong Man          | 1956 | 105 | 2448640.00  | 0.40 |
| The Trouble with Harry | 1955 | 99  | 17939400.00 | 0.83 |
| Niagara                | 1953 | 92  | 3946750.00  | 0.50 |

[3740 rows x 3 columns]

```
[101]: # Let's start by taking a look at the correlation between runtime and net profit/profit margin.
pearsoncorr = runtime_df.corr(method='pearson')
pearsoncorr
```

|                 | Runtime | Adjusted_Profit | Profit_Margin |
|-----------------|---------|-----------------|---------------|
| Runtime         | 1.00    | 0.22            | 0.05          |
| Adjusted_Profit | 0.22    | 1.00            | 0.05          |
| Profit_Margin   | 0.05    | 0.05            | 1.00          |

```
[102]: # Plot the correlation.
plt.figure(figsize=(14,7))
ax15 = sns.regplot(x='Runtime', y='Adjusted_Profit', data=imdb_budgets_df)
plt.xlabel('Movie Runtime', fontdict = {'fontname': 'Times New Roman', 'color':'gray', 'fontsize' : '15'})
plt.ylabel('Net Profit (In Billions)', fontdict = {'fontname': 'Times New Roman', 'color': 'gray', 'fontsize' : '15'})
plt.title('Correlation Between Net Profit and Runtime', fontdict = {'fontname': 'Times New Roman', 'color': 'gray', 'fontsize' : '25'})
plt.savefig('CorrProfitRuntime', dpi=300);
```



Although there is a small positive correlation of .223 showing that the long the runtime the higher the net profit, it's incredibly minute. With that in mind, we can take from this that, typically, it is not important to keep a movie above or below a certain time threshold.

**Question 6 Conclusion:** We recommend that our Company take into consideration the rating of the movie based on the genre and target audience. If making animation movies, it is wise to stick to a G or PG rating, otherwise PG-13 is the sweet spot. In terms of runtime, there is little correlation in terms of overall profitability.

## 8 Question 7: Sticking to our analysis of Net Profit and Profit Margin, what should our Company determine to be the baseline for sustainable success?

We have an understanding of what goes into a successful movie but let's determine what our Company should expect in terms of profitability if they expect to compete with the other top movie studios.

```
[103]: # Merge studio_df and imdb_budgets_df
studiobudgets_df = pd.merge(studio_df, imdb_budgets_df, left_on = 'title', right_on='Movie')
studiobudgets_df.head()
```

|   | title                      | studio                | domestic_gross | foreign_gross | year | IMDb Rating | Runtime | Genre                          |
|---|----------------------------|-----------------------|----------------|---------------|------|-------------|---------|--------------------------------|
| 0 | Toy Story 3                | Buena Vista           | 415000000.00   | 652000000     | 2010 | 8.30        | G       | [Animation, Adventure, Comedy] |
| 1 | Inception                  | WB                    | 292600000.00   | 535700000     | 2010 | 8.80        | PG-13   | [Action, Adventure, Sci-Fi]    |
| 2 | Shrek Forever After        | Pixar/Dreamworks      | 238700000.00   | 513900000     | 2010 | 6.30        | PG      | [Animation, Adventure, Comedy] |
| 3 | The Twilight Saga: Eclipse | Sumbadhat Productions | 300500000.00   | 398000000     | 2010 | 5.00        | PG-13   | [Adventure, Drama, Fantasy]    |
| 4 | Iron Man 2                 | Paramount             | 312400000.00   | 311500000     | 2010 | 7.00        | PG-13   | [Action, Adventure, Sci-Fi]    |

|   | Release Date | Production Budget | Domestic Gross | Worldwide Gross | Profit    |
|---|--------------|-------------------|----------------|-----------------|-----------|
| 0 | 2010-06-18   | 200000000         | 415004880      | 1068879522      | 868879522 |
| 1 | 2010-07-16   | 160000000         | 292576195      | 832551961       | 672551961 |
| 2 | 2010-05-21   | 165000000         | 238736787      | 756244673       | 591244673 |
| 3 | 2010-06-30   | 680000000         | 300531751      | 706102828       | 638102828 |
| 4 | 2010-05-07   | 170000000         | 312433331      | 621156389       | 451156389 |

|   | Profit_Margin | Adjusted_Budget | Adjusted_Profit | Month |
|---|---------------|-----------------|-----------------|-------|
| 0 | 0.81          | 264400000.00    | 1148658728.08   | June  |
| 1 | 0.81          | 211520000.00    | 889113692.44    | July  |
| 2 | 0.78          | 218130000.00    | 781625457.71    | May   |

```

3          0.90    89896000.00    843571938.62   June
4          0.73    224740000.00    596428746.26   May

```

[237]: *# Let's remove some unnecessary fields.*

```

studiobudgets_df.drop(columns = {'title', 'domestic_gross', 'Domestic Gross', 'foreign_gross', 'year', 'Production Budget', 'Worldwide Gross', 'Profit'}, inplace = True)
studiobudgets_df.rename(columns = {'studio':'Studio', 'Worldwide Gross' : 'Worldwide Gross' }, inplace = True)
studiobudgets_df.head()

```

[237]:

|   | Studio                | IMDb | Rating | Runtime | \ |
|---|-----------------------|------|--------|---------|---|
| 0 | Buena Vista           | 8.3  | G      | 103     |   |
| 1 | WB                    | 8.8  | PG-13  | 148     |   |
| 2 | Pixar/Dreamworks      | 6.3  | PG     | 93      |   |
| 3 | Sumbadhat Productions | 5.0  | PG-13  | 124     |   |
| 4 | Paramount             | 7.0  | PG-13  | 124     |   |

|   | Genre                          | Release Date | Profit Margin | \ |
|---|--------------------------------|--------------|---------------|---|
| 0 | [Animation, Adventure, Comedy] | 2010-06-18   | 0.812888      |   |
| 1 | [Action, Adventure, Sci-Fi]    | 2010-07-16   | 0.807820      |   |
| 2 | [Animation, Adventure, Comedy] | 2010-05-21   | 0.781817      |   |
| 3 | [Adventure, Drama, Fantasy]    | 2010-06-30   | 0.903697      |   |
| 4 | [Action, Adventure, Sci-Fi]    | 2010-05-07   | 0.726317      |   |

|   | Adjusted_Budget | Adjusted_Profit | Month |
|---|-----------------|-----------------|-------|
| 0 | 264400000.0     | 1.148659e+09    | June  |
| 1 | 211520000.0     | 8.891137e+08    | July  |
| 2 | 218130000.0     | 7.816255e+08    | May   |
| 3 | 89896000.0      | 8.435719e+08    | June  |
| 4 | 224740000.0     | 5.964287e+08    | May   |

[254]: *# Group by studio, find median and filter to top 25 by Adjusted Profit*

```

profit_by_studiodf = studiobudgets_df.groupby('Studio').median()
profit_by_studiodf = profit_by_studiodf.reset_index()
profit_by_studiodf = profit_by_studiodf.nlargest(25, 'Adjusted_Profit')
profit_by_studiodf

```

[254]:

|    | Studio           | IMDb | Runtime | Profit Margin | \ |
|----|------------------|------|---------|---------------|---|
| 51 | UTV              | 8.45 | 141.5   | 0.958798      |   |
| 37 | Pixar/Dreamworks | 6.70 | 94.0    | 0.716170      |   |
| 9  | Buena Vista      | 7.10 | 117.0   | 0.667056      |   |
| 28 | MBox             | 7.80 | 158.0   | 0.624019      |   |
| 48 | Strand           | 6.50 | 112.0   | 0.741792      |   |
| 45 | Sony             | 6.30 | 105.0   | 0.658692      |   |
| 35 | Paramount        | 6.40 | 110.0   | 0.639187      |   |
| 20 | Fox              | 6.35 | 106.0   | 0.644465      |   |

|    |                                |                        |      |       |          |
|----|--------------------------------|------------------------|------|-------|----------|
| 52 |                                | Universal              | 6.20 | 108.0 | 0.686945 |
| 54 |                                | WB                     | 6.60 | 113.5 | 0.542261 |
| 15 |                                | Eros International     | 7.10 | 160.0 | 0.836702 |
| 55 |                                | Wein/Dimension         | 5.90 | 96.0  | 0.750298 |
| 44 |                                | Screen Gems            | 5.80 | 103.0 | 0.698444 |
| 27 | Lionsgate/Summit Entertainment |                        | 6.55 | 110.0 | 0.606561 |
| 32 |                                | Neon                   | 7.50 | 119.0 | 0.795529 |
| 46 |                                | Sony Pictures          | 6.70 | 112.0 | 0.664717 |
| 25 |                                | Lionsgate              | 6.15 | 103.5 | 0.601290 |
| 49 |                                | Sumbadhat Productions  | 6.60 | 100.0 | 0.446140 |
| 19 |                                | Focus Features         | 6.90 | 108.0 | 0.484553 |
| 56 |                                | Weinstein Company      | 7.20 | 106.5 | 0.694665 |
| 43 |                                | STX                    | 6.40 | 104.0 | 0.528697 |
| 40 |                                | Relativity Media       | 6.25 | 105.5 | 0.506080 |
| 10 |                                | CBS                    | 6.60 | 102.0 | 0.591352 |
| 50 |                                | The Orchard            | 7.90 | 101.0 | 0.891707 |
| 47 |                                | Sony Pictures Classics | 7.20 | 109.0 | 0.600112 |

|    | Adjusted_Budget | Adjusted_Profit |
|----|-----------------|-----------------|
| 51 | 33747300.0      | 6.921112e+08    |
| 37 | 182352000.0     | 4.921191e+08    |
| 9  | 176565000.0     | 1.928538e+08    |
| 28 | 116082000.0     | 1.926625e+08    |
| 48 | 50796000.0      | 1.459292e+08    |
| 45 | 65796000.0      | 1.296401e+08    |
| 35 | 53053600.0      | 1.270562e+08    |
| 20 | 65785200.0      | 1.171804e+08    |
| 52 | 47728000.0      | 1.081619e+08    |
| 54 | 66914000.0      | 8.010906e+07    |
| 15 | 10170820.0      | 5.211316e+07    |
| 55 | 27474000.0      | 5.093755e+07    |
| 44 | 30121600.0      | 5.004866e+07    |
| 27 | 44584000.0      | 4.695959e+07    |
| 32 | 12062600.0      | 4.693164e+07    |
| 46 | 26847000.0      | 4.222879e+07    |
| 25 | 32376500.0      | 3.662573e+07    |
| 49 | 41273600.0      | 3.615531e+07    |
| 19 | 20121600.0      | 3.370892e+07    |
| 56 | 18381000.0      | 3.362412e+07    |
| 43 | 32898000.0      | 3.331053e+07    |
| 40 | 33053600.0      | 2.923035e+07    |
| 10 | 21926600.0      | 2.688925e+07    |
| 50 | 2822000.0       | 2.323702e+07    |
| 47 | 8041120.0       | 1.587142e+07    |

```
[253]: # Let's take a look at the average of these median values.
profit_by_studiodf.describe()
```

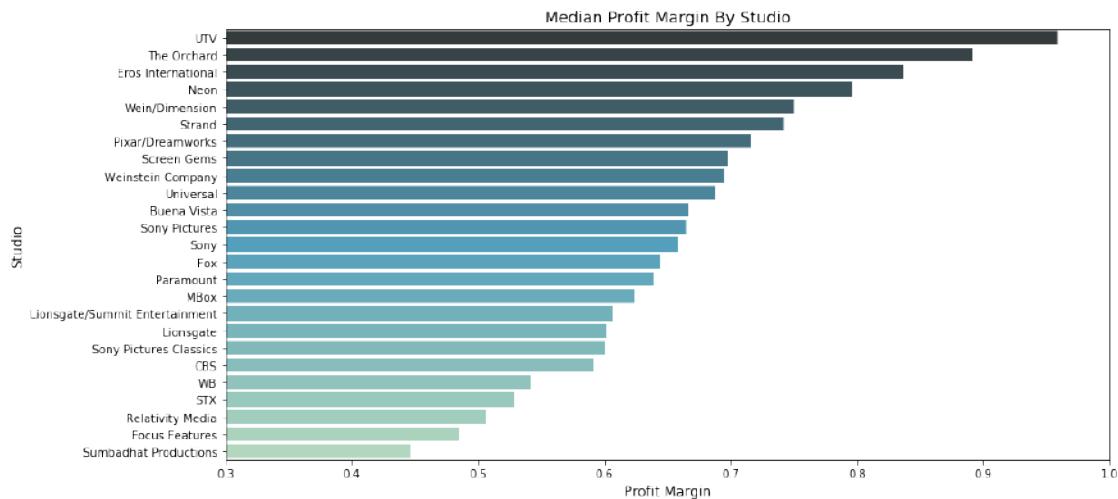
```
[253]:
```

|       | IMDb      | Runtime    | Profit_Margin | Adjusted_Budget | Adjusted_Profit |
|-------|-----------|------------|---------------|-----------------|-----------------|
| count | 25.000000 | 25.000000  | 25.000000     | 2.500000e+01    | 2.500000e+01    |
| mean  | 6.76600   | 112.180000 | 0.663049      | 4.883893e+07    | 1.134278e+08    |
| std   | 0.64108   | 16.751169  | 0.122761      | 4.612474e+07    | 1.557719e+08    |
| min   | 5.80000   | 94.000000  | 0.446140      | 2.822000e+06    | 1.587142e+07    |
| 25%   | 6.35000   | 103.500000 | 0.600112      | 2.192660e+07    | 3.370892e+07    |
| 50%   | 6.60000   | 108.000000 | 0.658692      | 3.305360e+07    | 5.004866e+07    |
| 75%   | 7.10000   | 112.000000 | 0.716170      | 5.305360e+07    | 1.270562e+08    |
| max   | 8.45000   | 160.000000 | 0.958798      | 1.823520e+08    | 6.921112e+08    |

We can see that if we want to strive to be in the top half of this elite list of movie studios we need to have a profit margin of 66% and a net profit of 50 million per movie.

```
[256]: #Plot the above findings.
```

```
plt.figure(figsize=(14,7))
ax16 = sns.barplot(x=profit_by_studiodf['Profit_Margin'], □
    ↵y=profit_by_studiodf['Studio'],
    ↵order=profit_by_studiodf.sort_values('Profit_Margin', □
    ↵ascending=False).Studio, palette='GnBu_d')
plt.xlabel('Profit Margin', fontsize=12)
plt.ylabel('Studio', fontsize=12)
plt.title('Median Profit Margin By Studio', fontsize=14)
plt.xlim(0.3, 1.0)
plt.savefig('ProfitMarginStudio')
```



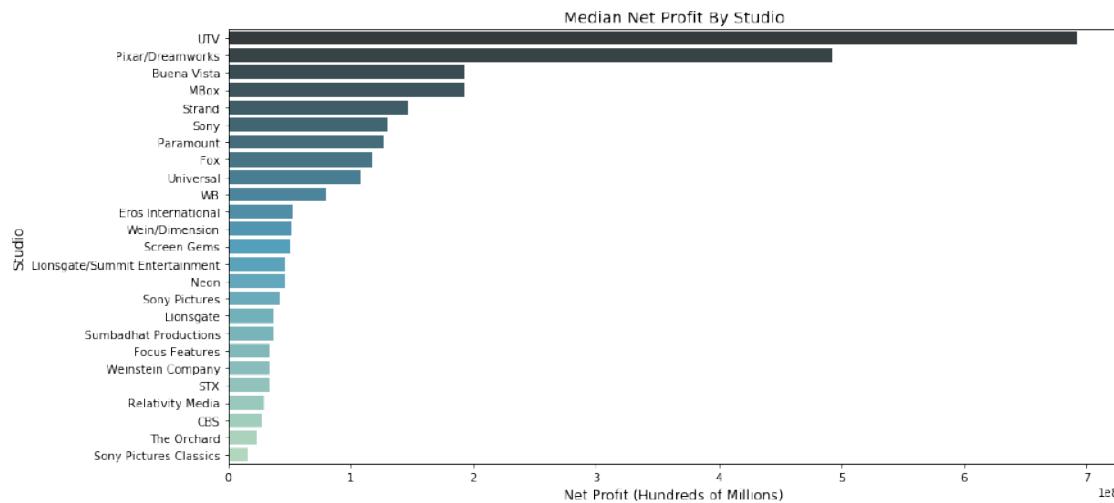
```
[255]: #Plot the above findings.
```

```
plt.figure(figsize=(14,7))
ax16 = sns.barplot(y=profit_by_studiodf['Studio'], □
    ↵x=profit_by_studiodf['Adjusted_Profit'], palette='GnBu_d')
plt.xlabel('Net Profit (Hundreds of Millions)', fontsize=12)
```

```

plt.ylabel('Studio', fontsize=12)
plt.title('Median Net Profit By Studio', fontsize=14)
plt.savefig('NetProfitStudio');

```



We can see from the graph above that the major players in the studio industry have profit margins ranging from 24% to 95%. That's quite a large range to define success. However, the top 25 studios shown are many of the studios that we often recognize when we go to the movies. As we've done previously, we use the median profit margin of the top 25 as a target for success among major studios. That profit margin is 66%. In the next analysis we'll take a closer look at some of these major studios to see what metrics we should try to mimic. Let's also keep this in mind as we go into our next analysis: UTV which has the greatest profit margin of all the studios is a subsidiary of Disney.

**Question 7 Conclusion:** Microsoft should aim for a profit margin of 66% and a net profit of slightly over 50 million per movie to compete with the top existing studios.

## 9 Question 8: Based on the success of current competitors, which should we look to for best practices?

We need to add a column to the `theaters_df` dataframe to calculate the money grossed per theater for a given movie. Then we can group by studio.

```

[106]: theaters_df['dollars_per_theater'] = theaters_df['total_dom_gross($)'] / theaters_df['max_theaters']
theaters_df.head()

```

```

[106]:          title  max_theaters  year  total_dom_gross($) \
0      The Lion King        4802  2019      543638043
1  Avengers: Endgame        4662  2019      858373000
2  Spider-Man: Far from Home        4634  2019      390532085

```

|   |                |      |      |           |
|---|----------------|------|------|-----------|
| 3 | Toy Story 4    | 4575 | 2019 | 434038008 |
| 4 | It Chapter Two | 4570 | 2019 | 211593228 |

|   | studio       | dollars_per_theater |
|---|--------------|---------------------|
| 0 | Disney       | 113210.75           |
| 1 | Disney       | 184121.19           |
| 2 | Sony         | 84275.37            |
| 3 | Disney       | 94871.70            |
| 4 | Warner Bros. | 46300.49            |

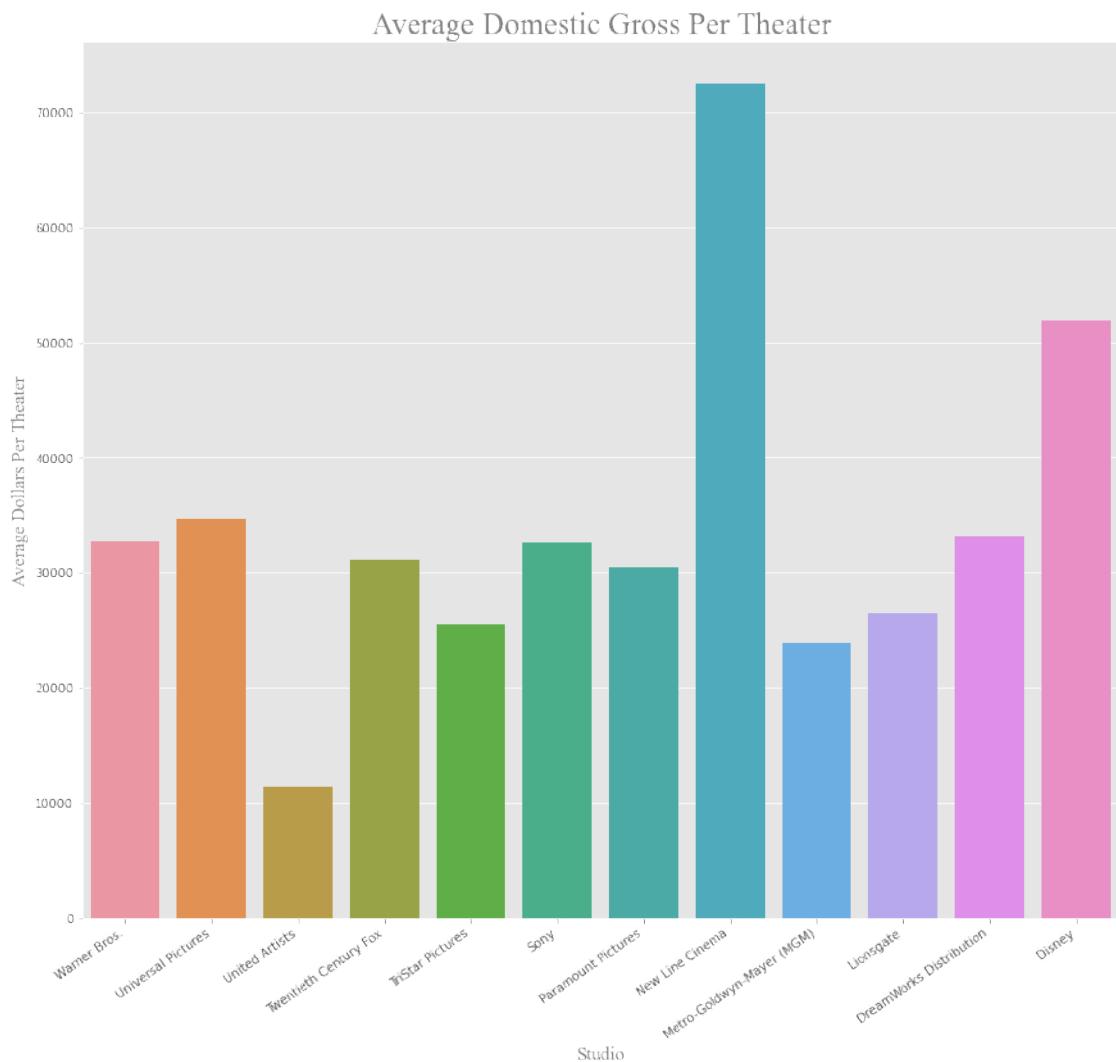
```
[107]: #Let's see what the average is for max number of theaters and for gross per ↵
       ↵theater for each studio
average_theaters = theaters_df.groupby('studio').mean()
average_theaters_ranked = average_theaters.
    ↵sort_values(by=['studio'], ascending=False)
average_theaters_ranked.reset_index(inplace=True)
average_theaters
```

|                           | max_theaters | year    | total_dom_gross(\$) | \ |
|---------------------------|--------------|---------|---------------------|---|
| studio                    |              |         |                     |   |
| Disney                    | 3682.32      | 2010.59 | 202617891.97        |   |
| DreamWorks Distribution   | 3408.26      | 2002.95 | 118198315.42        |   |
| Lionsgate                 | 3356.24      | 2014.47 | 95268293.14         |   |
| Metro-Goldwyn-Mayer (MGM) | 3259.14      | 2004.00 | 78437576.64         |   |
| New Line Cinema           | 3410.57      | 2001.86 | 249718149.29        |   |
| Paramount Pictures        | 3466.71      | 2010.71 | 108614912.30        |   |
| Sony                      | 3478.36      | 2010.56 | 116677932.63        |   |
| TriStar Pictures          | 3146.00      | 2014.00 | 80703217.29         |   |
| Twentieth Century Fox     | 3493.98      | 2011.21 | 111009777.12        |   |
| United Artists            | 3124.00      | 2003.00 | 35667218.00         |   |
| Universal Pictures        | 3488.41      | 2011.96 | 124914179.39        |   |
| Warner Bros.              | 3535.03      | 2011.59 | 120355240.25        |   |

|                           | dollars_per_theater |
|---------------------------|---------------------|
| studio                    |                     |
| Disney                    | 51856.14            |
| DreamWorks Distribution   | 33102.06            |
| Lionsgate                 | 26485.34            |
| Metro-Goldwyn-Mayer (MGM) | 23829.21            |
| New Line Cinema           | 72518.24            |
| Paramount Pictures        | 30508.47            |
| Sony                      | 32626.67            |
| TriStar Pictures          | 25546.75            |
| Twentieth Century Fox     | 31119.14            |
| United Artists            | 11417.16            |
| Universal Pictures        | 34679.48            |
| Warner Bros.              | 32678.01            |

```
[115]: plt.figure(figsize=(15,13))
ax16 = sns.barplot(x='studio', y='dollars_per_theater', data=average_theaters_ranked)
plt.xlabel('Studio', fontdict = {'fontname': 'Times New Roman', 'color':'gray', 'fontsize' : '15'})
plt.title("Average Domestic Gross Per Theater", fontdict = {'fontname': 'Times New Roman', 'color': 'gray', 'fontsize' : '25'});
plt.ylabel('Average Dollars Per Theater', fontdict = {'fontname': 'Times New Roman', 'color': 'gray', 'fontsize' : '15'});
plt.xticks(rotation=35, horizontalalignment='right')
plt.savefig('DomesticPerTheater', dpi=300);
```



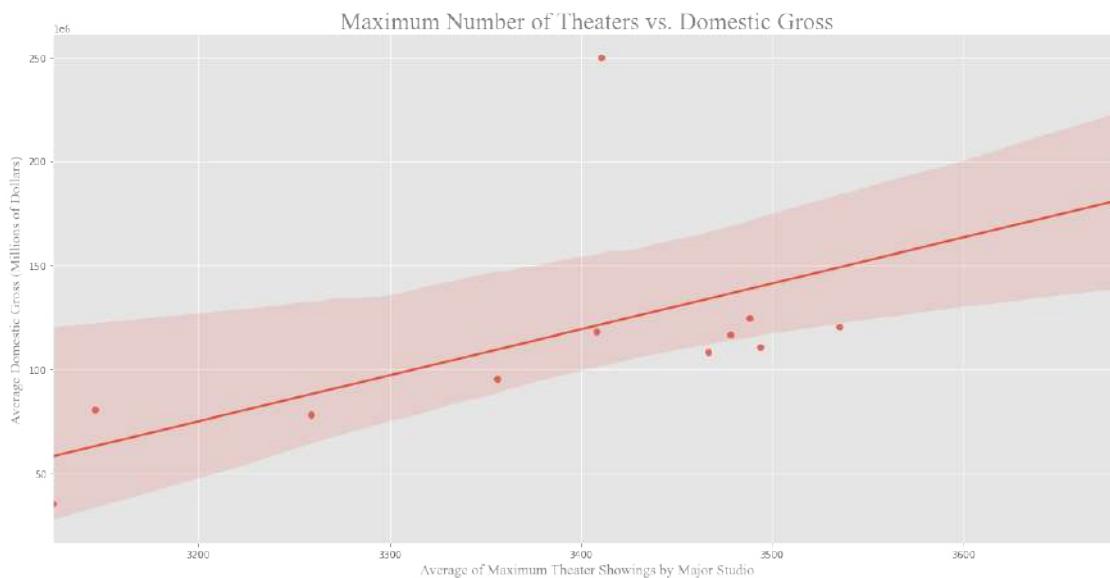
In the bar plot above, Disney and New Line Cinema stand out. We need to double check that there are an appropriate number of movies by each of these studios before jumping to conclusions.

```
[116]: theaters_df['studio'].value_counts()
```

```
[116]: Warner Bros.          208
        Twentieth Century Fox  165
        Disney                 147
        Universal Pictures     136
        Sony                  135
        Paramount Pictures     112
        Lionsgate              49
        DreamWorks Distribution 19
        Metro-Goldwyn-Mayer (MGM) 14
        New Line Cinema         7
        TriStar Pictures         7
        United Artists           1
Name: studio, dtype: int64
```

We can see that New Line Cinema only has 7 movies in this dataframe which means that their average domestic gross per theater is going to be skewed. Disney is certainly still a possibility and we should also consider Warner Bros. and Twentieth Century Fox.

```
[118]: ax17 = sns.lmplot(x='max_theaters', y='total_dom_gross($)', data=average_theaters, height=8, aspect=2)
plt.ticklabel_format(axis='y', style='sci', scilimits=(6,6))
plt.xlabel('Average of Maximum Theater Showings by Major Studio', fontdict = {'fontname': 'Times New Roman', 'color': 'gray', 'fontsize' : '15'})
plt.ylabel('Average Domestic Gross (Millions of Dollars)', fontdict = {'fontname': 'Times New Roman', 'color': 'gray', 'fontsize' : '15'})
plt.title('Maximum Number of Theaters vs. Domestic Gross', fontdict = {'fontname': 'Times New Roman', 'color': 'gray', 'fontsize' : '25'})
plt.savefig('TheatersVGross', dpi=300);
```



The scatter plot shows a positive trend between the average number of theaters and the average domestic gross. The sole outlier is New Line Cinemas due to how few movies they are associated with in our dataframe. Disney is farthest to the right and above the trend line further proving that they should be a strong consideration.

We'll join the theater and awards dataframes so that we can see which studios have the best win rate at the Oscars.

```
[119]: theaters_df.set_index(['title', 'year'], inplace=True)
```

```
[120]: theaters_and_awards = theaters_df.join(awards_df, how='inner', on=['title', 'year'])
```

```
[121]: theaters_and_awards.groupby('studio').count()
```

```
[121]: max_theaters  total_dom_gross($)  \
studio
Disney                22                  22
DreamWorks Distribution 4                   4
New Line Cinema          2                  2
Paramount Pictures       7                  7
Sony                   6                   6
Twentieth Century Fox   4                   4
Universal Pictures        6                  6
Warner Bros.             15                 15

dollars_per_theater  awards_won  awards_nominated  \
studio
Disney                22                  22                  22
DreamWorks Distribution 4                   4                   4
New Line Cinema          2                  2                   2
Paramount Pictures       7                  7                   7
Sony                   6                   6                   6
Twentieth Century Fox   4                   4                   4
Universal Pictures        6                  6                   6
Warner Bros.             15                 15                  15

win_rate
studio
Disney                22
DreamWorks Distribution 4
New Line Cinema          2
Paramount Pictures       7
Sony                   6
Twentieth Century Fox   4
Universal Pictures        6
```

Warner Bros.

15

```
[122]: theaters_and_awards.groupby('studio').mean()
```

```
[122]:          max_theaters  total_dom_gross($)  \
studio
Disney                  3818.73      305217242.45
DreamWorks Distribution  3444.25      153223630.75
New Line Cinema          3662.50      358408603.00
Paramount Pictures       3564.86      140835427.57
Sony                    3653.67      237842295.67
Twentieth Century Fox   3501.75      136874930.25
Universal Pictures        3338.83      149344665.00
Warner Bros.             3831.60      234055876.80

          dollars_per_theater  awards_won  awards_nominated  \
studio
Disney                  78797.61      1.36            3.00
DreamWorks Distribution  44447.63      2.00            4.25
New Line Cinema          97814.75      6.50            8.50
Paramount Pictures       38930.82      1.00            3.71
Sony                    64720.23      1.17            3.17
Twentieth Century Fox   38404.79      2.25            6.00
Universal Pictures        44970.82      1.33            3.33
Warner Bros.             60023.04      2.67            5.87

          win_rate
studio
Disney                0.60
DreamWorks Distribution 0.60
New Line Cinema         0.67
Paramount Pictures      0.45
Sony                  0.54
Twentieth Century Fox  0.43
Universal Pictures       0.51
Warner Bros.            0.56
```

Unfortunately, the joining of the dataframes only left us with 66 common movies. We would prefer to have more data to be more confident in establishing trends. We will consider the average number of theaters and average win rate to make a determination. Disney is associated with 22 movies in our joined dataframe while Warner Bros. is associated with 15. Warner Bros does have a higher average for the number of theaters, however Disney has a noticeable \$18,000 advantage in average domestic gross per theater. Disney also has the higher win rate for Oscars at nearly 60%.

**Question 8 Conclusion:** Our Company should research Disney's best practices and try to build off the success of this well established studio.

## 10 Conclusion

While there are many other factors that we could consider in a future analysis we feel that the following 8 conclusions will result in a successful business venture as our Comapany enters the movie industry.

1. I recommend that we should budget approximately \$82,250,000 to make a movie. This should correlate with a profit margin above 80%.
2. I recommend that we should focus their efforts on the top 6 most profitable movie genres: Adventure, Action, Comedy, Drama, Sci-Fi and Animation. A further recommendation to focus on Sci-Fi and Animation due to less competition and a higher opportunity to profit.
3. I recommend that we release the bulk of their movies, especially Animation, during the summer months. Adventure, Drama and Comedy movies would see similar success if released in November, but the recommendation remains to focus on summer.
4. Question 4 Conclusion: I recommend that we focus their cast and crew search to individuals who consistently score at least 1.0 on the VAR score. We can, with a high level of confidence, conclude that these individuals will elevate the overall production.
5. We should spend at least \$35,465,000 in order to make an Oscar-winning movie.
6. I recommend that we take into consideration the rating of the movie based on the genre and target audience. If making animation movies, it is wise to stick to a G or PG rating, otherwise PG-13 is the sweetspot. In terms of runtime, there is little correlation in terms of overall profitability.
7. We should aim for a profit margin of 66% and a net profit of slightly over 50 million per movie to compete with the top existing studios.
8. We should research Disney's best practices and try to build off the success of this well established studio.



project

6

**Python - Flipkart Review Sentiment Analysis**

# flipkart-review-sentiment-analysis

February 24, 2024

```
[5]: import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from wordcloud import WordCloud as wc, STOPWORDS as sw

[6]: df=pd.read_csv("Flipkart Reviews Sentiment analysis.csv")
df.head(1)

[6]: product_name product_price Rating \
0 Candes 12 L Room/Personal Air Cooler??????(Whi... 3999.00 5
      Review                               Summary Sentiment
0 super! great cooler excellent air flow and for this p... positive
```

## 0.1 EXPLORATORY ANALYSIS & CLEANING

```
[7]: df.shape

[7]: (205052, 6)

[8]: df.describe()

[8]: product_name product_price \
      count          205052          205052
      unique         958            525
      top    cello Pack of 18 Opalware Cello Dazzle Lush Fi... 1299.00
      freq           6005            9150

      Rating      Review Summary Sentiment
      count  205052  180388  205041  205052
      unique     8    1324   92923     3
      top        5 wonderful   good  positive
      freq  118765    9016   17430  166581
```

```
[9]: df.columns
```

```
[9]: Index(['product_name', 'product_price', 'Rating', 'Review', 'Summary',
       'Sentiment'],
       dtype='object')
```

```
[10]: df.info()
```

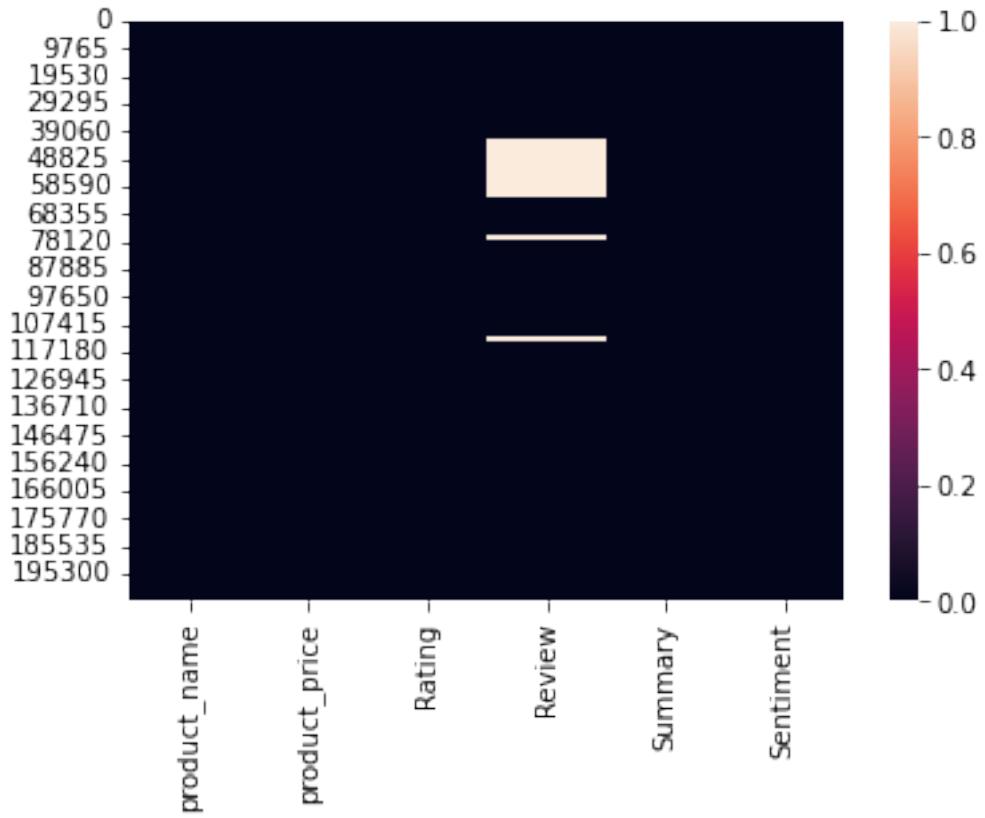
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205052 entries, 0 to 205051
Data columns (total 6 columns):
 #   Column           Non-Null Count  Dtype  
 ---  --  
 0   product_name    205052 non-null   object  
 1   product_price   205052 non-null   object  
 2   Rating          205052 non-null   object  
 3   Review          180388 non-null   object  
 4   Summary         205041 non-null   object  
 5   Sentiment        205052 non-null   object  
dtypes: object(6)
memory usage: 4.7+ MB
```

```
[11]: df.isnull().sum()
```

```
[11]: product_name      0
      product_price     0
      Rating           0
      Review          24664
      Summary          11
      Sentiment         0
      dtype: int64
```

```
[12]: sns.heatmap(df.isnull())
```

```
[12]: <AxesSubplot:>
```



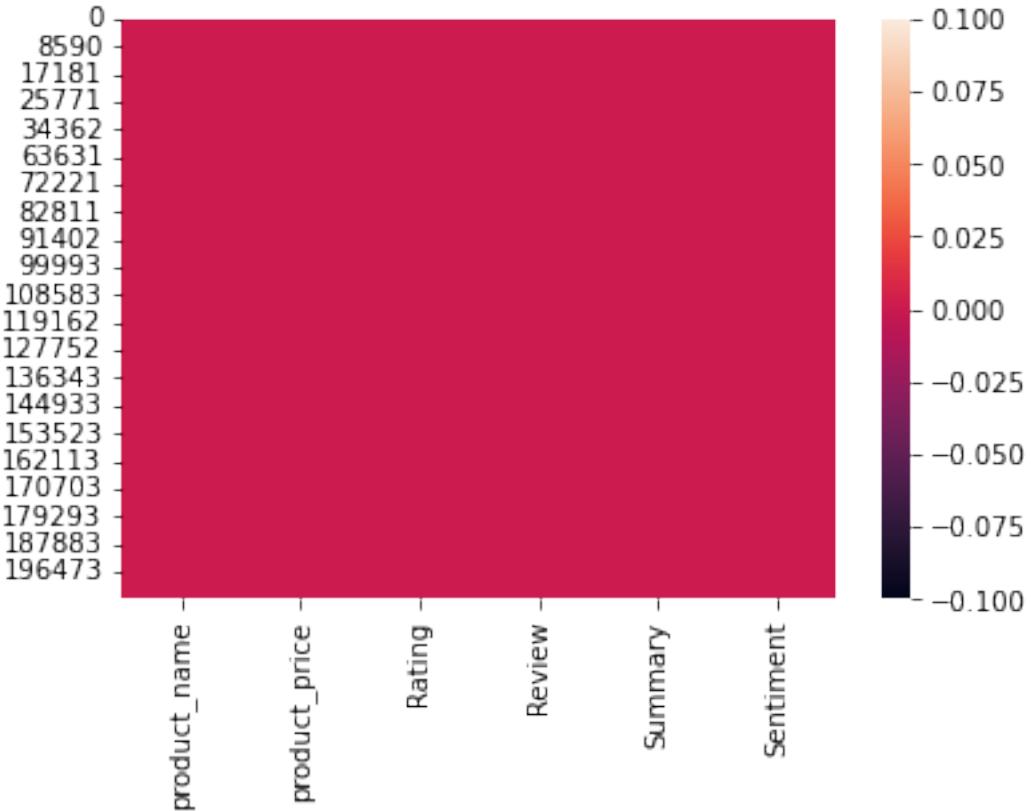
```
[13]: df=df.dropna()
```

```
[14]: df.isnull().sum()
```

```
[14]: product_name      0
product_price      0
Rating            0
Review            0
Summary           0
Sentiment          0
dtype: int64
```

```
[15]: sns.heatmap(df.isnull())
```

```
[15]: <AxesSubplot:>
```



```
[16]: df.shape
```

```
[16]: (180379, 6)
```

## 0.2 ANALYSIS

```
[17]: df.head(1)
```

```
[17]:          product_name product_price Rating \
0  Candes 12 L Room/Personal Air Cooler??????(Whi...      3999.00      5

                    Review             Summary Sentiment
0  super! great cooler excellent air flow and for this p...  positive
```

```
[25]: df.product_name.nunique()
```

```
[25]: 841
```

```
[19]: text = " ".join(i for i in df.Review)
word_cloud = wc(
    width=3000,
```

```

height=2000,
random_state=1,
background_color="black",
colormap="Pastel1",
collocations=False,
stopwords=sw,
).generate(text)

plt.imshow(word_cloud)
plt.axis("off")
plt.show()

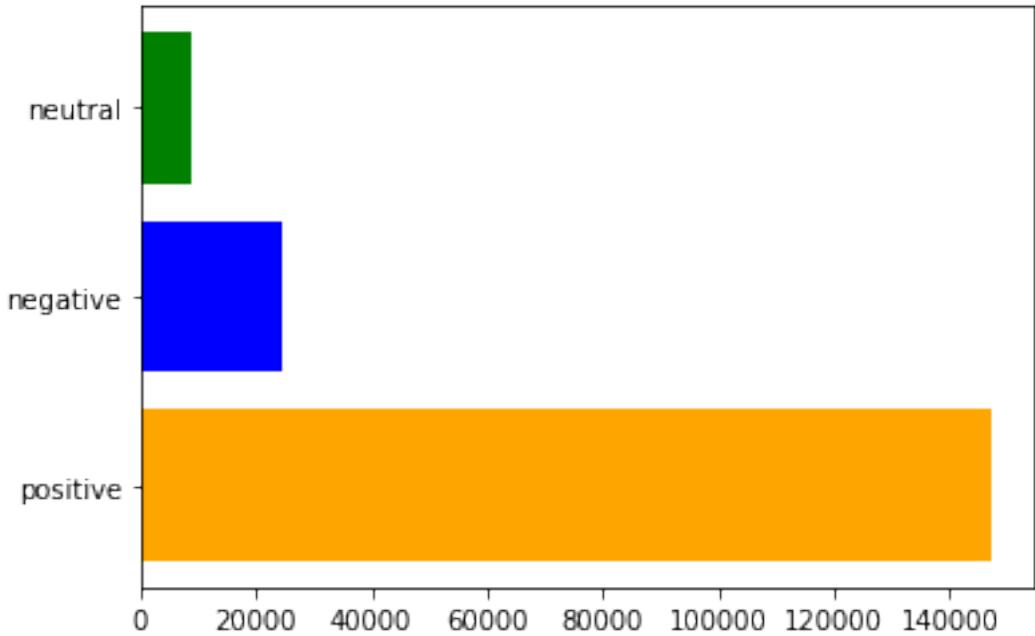
```



```
[20]: N=df['Sentiment'].value_counts()
s=df['Sentiment'].unique()
```

```
[21]: plt.barch(s,N,color={'green','orange','blue'})
```

```
[21]: <BarContainer object of 3 artists>
```

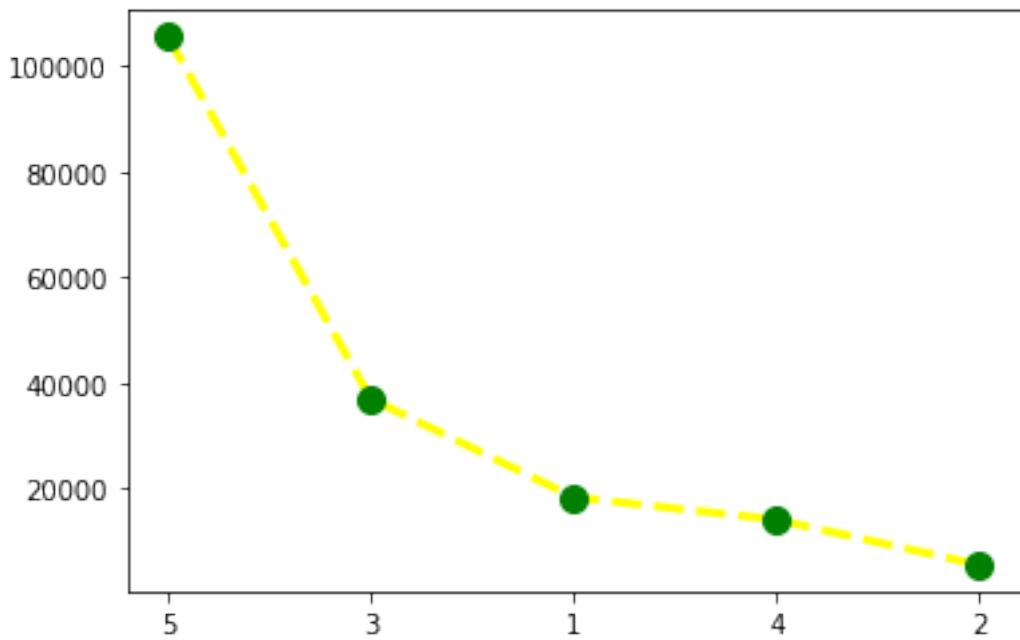


```
[22]: df=df.drop(index={17299,175895,175906})
```

```
[23]: rc=df.Rating.value_counts()  
r=df.Rating.unique()
```

```
[24]: plt.  
    plot(r,rc,linestyle='dashed',linewidth=3,color='yellow',marker="o",ms=10,mec='green',mfc='g
```

```
[24]: [matplotlib.lines.Line2D at 0x19beb80]
```



[ ]:



**project**

**7**

**Python - Trending Music on Instagram  
& Snapchat**

# ng-music-on-instagram-and-snapchat

February 24, 2024

```
[1]: import pandas as pd  
import matplotlib as plt
```

```
[2]: df=pd.read_csv('TrendingMusicOnInstagramAndSnapchat.csv')  
df.head(2)
```

```
[2]:          Platform      Month            Trend      Trend Type  \  
0  Instagram Reels  2022/06  DJ Wait A Minute  Montage, Vlog  
1  Instagram Reels  2022/06        Guud Girls  Montage, Vlog  
  
                           Music           Video Style     Theme  \  
0  DJ Wait A Minute - DJ Exe (2022)  Photo/Video Compilation Lifestyle  
1  Guud Girls - Breaking Beattz (2021)  Photo/Video Compilation Lifestyle  
  
    Part of Song  Video length in S  Music  Genre  
0      Prelude             29.0       House  
1      Prelude             29.0       House
```

## 0.1 show the no of rows and columns of the dataframe

```
[3]: df.shape
```

```
[3]: (49, 10)
```

## 0.2 Count the total null values in each column

```
[4]: df.isnull().sum()
```

```
[4]: Platform      0  
Month         1  
Trend         0  
Trend Type    0  
Music          8  
Video Style    0  
Theme          0  
Part of Song   7  
Video length in S  2
```

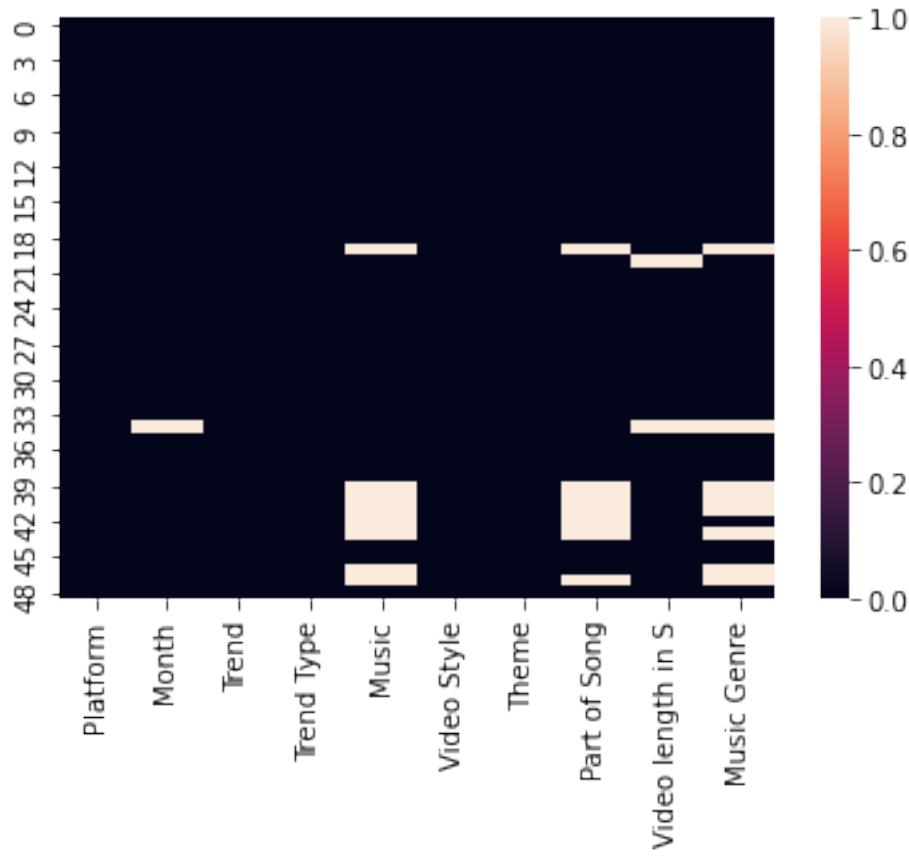
Music Genre  
dtype: int64

8

### 0.3 Represent all null values in Heat map

```
[5]: import seaborn as sns  
sns.heatmap(df.isnull())
```

[5]: <AxesSubplot:>



0.4 Replace all the null values with “No Value” for string and mean value for number column

```
[6]: df['Month'].fillna("NO VALUE", inplace=True)
      df['Music'].fillna("NO VALUE", inplace=True)
      df['Part of Song'].fillna("NO VALUE", inplace=True)
      df['Video length in S'].fillna(df['Video length in S'].mean(),inplace=True)
      df['Music Genre'].fillna("NO VALUE", inplace=True)
```

```
[7]: df.isnull().sum()
```

```
[7]: Platform      0  
Month         0  
Trend         0  
Trend Type    0  
Music          0  
Video Style   0  
Theme          0  
Part of Song  0  
Video length in S  0  
Music Genre   0  
dtype: int64
```

## 0.5 Count the unique values for all columns

```
[8]: df.nunique()
```

```
[8]: Platform      2  
Month         7  
Trend        49  
Trend Type   17  
Music         42  
Video Style  5  
Theme          9  
Part of Song 14  
Video length in S  17  
Music Genre  13  
dtype: int64
```

## 0.6 Find all the unique music genre

```
[9]: df['Music Genre'].unique()
```

```
[9]: array(['House', 'Pop', 'Latin Urbano', 'Dance/Electronic', 'R&B/Soul',  
       'Hip-Hop/Rap', 'Instrumental', 'Indie', 'NO VALUE', 'Classical',  
       'R&B,hip-hop', 'R&B', 'Rock'], dtype=object)
```

```
[10]: df.head(2)
```

```
[10]:      Platform     Month            Trend      Trend Type  \\\n0  Instagram Reels  2022/06  DJ Wait A Minute  Montage, Vlog  
1  Instagram Reels  2022/06           Guud Girls  Montage, Vlog  
  
                           Music            Video Style      Theme  \\\n0  DJ Wait A Minute - DJ Exe (2022)  Photo/Video Compilation  Lifestyle
```

```
1 Guud Girls - Breaking Beattz (2021) Photo/Video Compilation Lifestyle
```

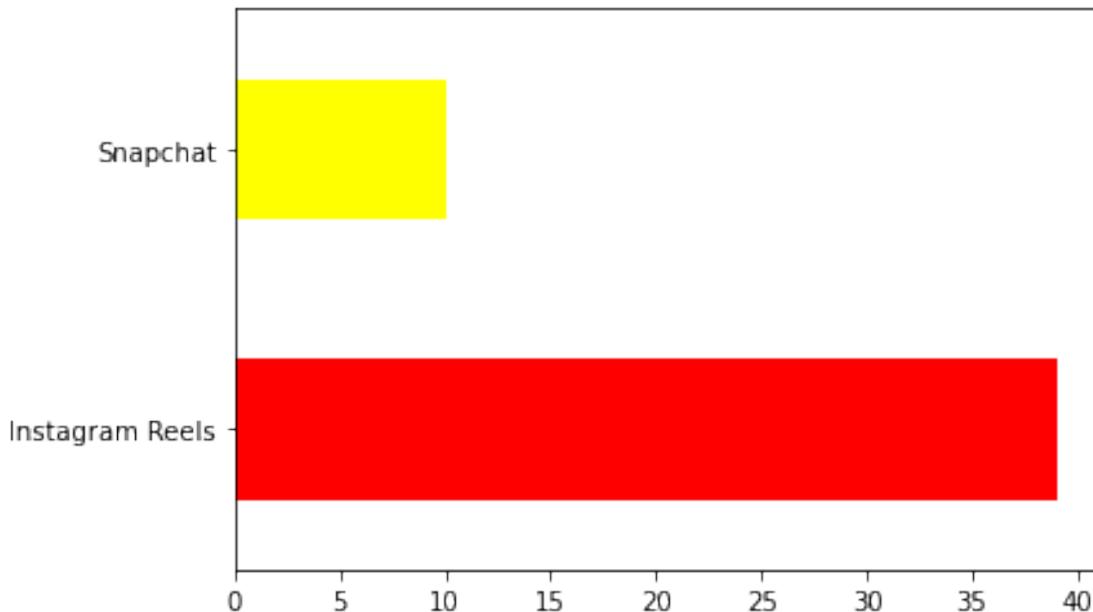
```
Part of Song Video length in S Music Genre
0 Prelude 29.0 House
1 Prelude 29.0 House
```

```
[11]: gp=df.Platform.value_counts()
gp
```

```
[11]: Instagram Reels    39
Snapchat        10
Name: Platform, dtype: int64
```

```
[12]: gp.plot(kind='barh',color=['red','yellow'])
```

```
[12]: <AxesSubplot:>
```

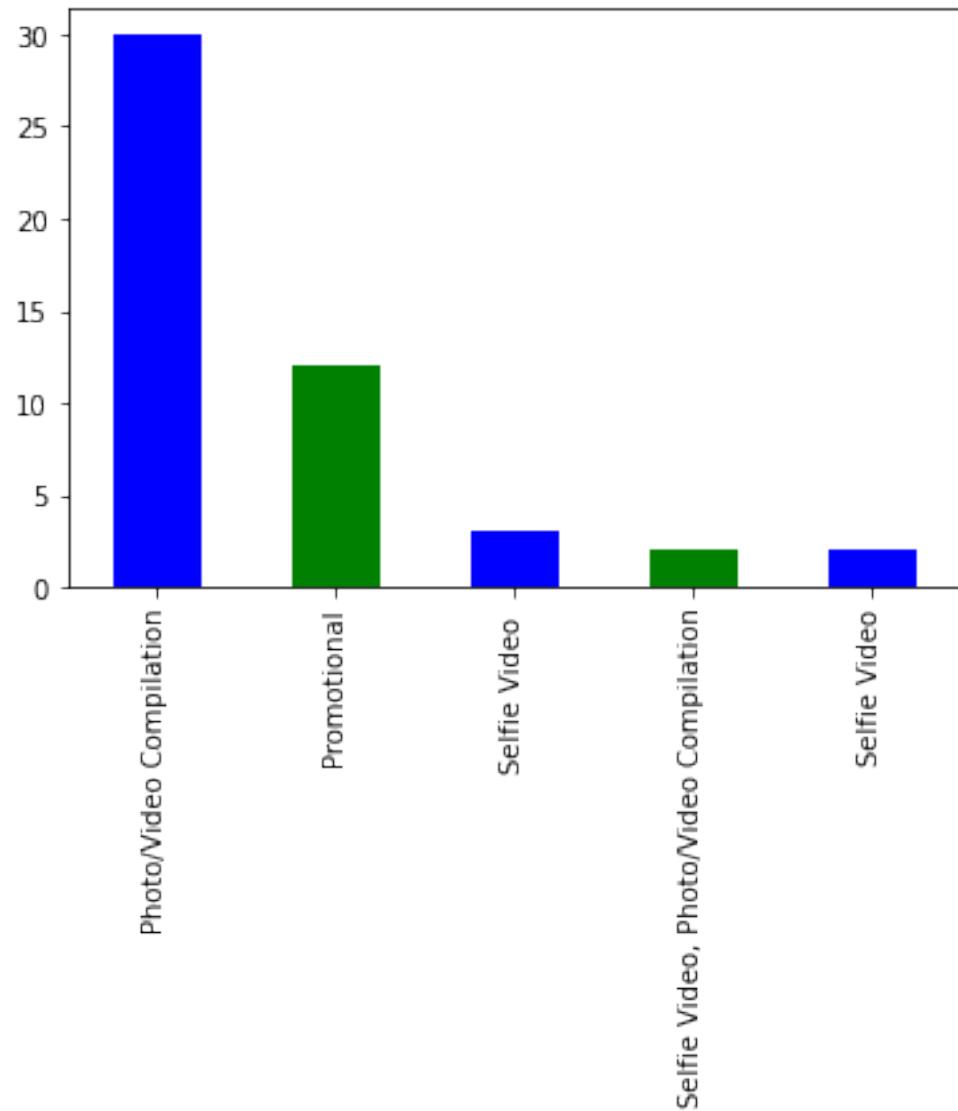


```
[13]: vs=df['Video Style'].value_counts()
vs
```

```
[13]: Photo/Video Compilation      30
Promotional                      12
Selfie Video                     3
Selfie Video, Photo/Video Compilation  2
Selfie Video                      2
Name: Video Style, dtype: int64
```

```
[14]: vs.plot(kind='bar',color=['blue','green'])
```

```
[14]: <AxesSubplot:>
```



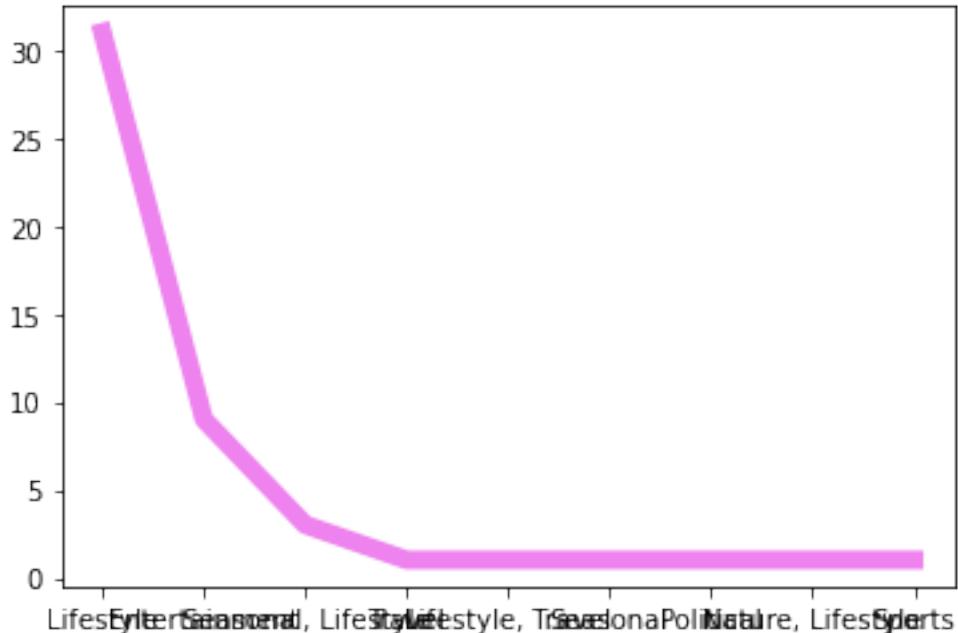
```
[15]: t=df.Theme.value_counts()  
t
```

```
[15]: Lifestyle          31  
Entertainment        9  
Seasonal, Lifestyle   3  
Travel                1  
Lifestyle, Travel     1  
Seasonal               1
```

```
Political          1  
Nature, Lifestyle    1  
Sports            1  
Name: Theme, dtype: int64
```

```
[16]: t.plot(kind="line", color="violet", linewidth="7")
```

```
[16]: <AxesSubplot:>
```



## 0.7 Find how many music trends in month 2022/08 and music genre is pop

```
[17]: df.head(2)
```

```
[17]: Platform      Month          Trend      Trend Type  \
0  Instagram Reels  2022/06  DJ Wait A Minute  Montage, Vlog
1  Instagram Reels  2022/06        Guud Girls  Montage, Vlog

  Music          Video Style      Theme  \
0  DJ Wait A Minute - DJ Exe (2022)  Photo/Video Compilation  Lifestyle
1  Guud Girls - Breaking Beattz (2021)  Photo/Video Compilation  Lifestyle

      Part of Song  Video length in S  Music  Genre
0      Prelude           29.0     House
1      Prelude           29.0     House
```

```
[18]: df[(df.Month == '2022/08') & (df['Music Genre'] == 'Pop')]['Trend Type'].  
      ↪value_counts()
```

```
[18]: Montage          3  
Act Out + Text on Screen + Montage    1  
Montage (Template)        1  
Dance + Challenge        1  
Transition             1  
Name: Trend Type, dtype: int64
```

## 0.8 Show all the records where instagram reels trend type is montage

```
[19]: df[(df['Platform'] == 'Instagram Reels') & (df['Trend Type'] == 'Montage')]
```

|    | Platform        | Month   | Trend                            | Trend Type | \ |
|----|-----------------|---------|----------------------------------|------------|---|
| 5  | Instagram Reels | 2022/06 | Still Don't Know My Name (Remix) | Montage    |   |
| 8  | Instagram Reels | 2022/06 | BREAK MY SOUL                    | Montage    |   |
| 13 | Instagram Reels | 2022/07 | Angel Eyes                       | Montage    |   |
| 16 | Instagram Reels | 2022/07 | Lucid Dreams Instrumental        | Montage    |   |
| 17 | Instagram Reels | 2022/07 | A Beautiful View                 | Montage    |   |
| 24 | Instagram Reels | 2022/08 | How Deep Is Your Love Mashup     | Montage    |   |
| 25 | Instagram Reels | 2022/08 | i jus flipped a switch           | Montage    |   |
| 26 | Instagram Reels | 2022/08 | SLIPPING THROUGH MY FINGERS      | Montage    |   |
| 29 | Instagram Reels | 2022/08 | Hold Me Closer                   | Montage    |   |
| 30 | Instagram Reels | 2022/08 | Overcome                         | Montage    |   |

|    | Music \                                           |  |  |  |
|----|---------------------------------------------------|--|--|--|
| 5  | Still Don't Know My Name (Remix) - Kimotion (2... |  |  |  |
| 8  | BREAK MY SOUL - Beyoncé (2022)                    |  |  |  |
| 13 | Angel Eyes - ABBA (1979)                          |  |  |  |
| 16 | Lucid Dreams Instrumental - Amarildo Notolli (... |  |  |  |
| 17 | Bittersweet Symphony - The Verve (1997)           |  |  |  |
| 24 | How deep is your love? - DJ Replica (2022)        |  |  |  |
| 25 | Remix by journalbymoon (Nonstop - Drake ft. Qu... |  |  |  |
| 26 | Slipping Through my Fingers - ABBA (1981) Cove... |  |  |  |
| 29 | Hold Me Closer - Elton John ft. Britney Spears... |  |  |  |
| 30 | Overcome - Skott (2022)                           |  |  |  |

|    | Video Style             | Theme     | Part of Song       | Video length in S | \ |
|----|-------------------------|-----------|--------------------|-------------------|---|
| 5  | Photo/Video Compilation | Lifestyle | Mashup             | 29.0              |   |
| 8  | Photo/Video Compilation | Lifestyle | Pre-Chorus, Chorus | 29.0              |   |
| 13 | Photo/Video Compilation | Lifestyle | Beginning          | 29.0              |   |
| 16 | Photo/Video Compilation | Lifestyle | Instrumental       | 10.0              |   |
| 17 | Photo/Video Compilation | Travel    | Prelude            | 7.0               |   |
| 24 | Photo/Video Compilation | Lifestyle | Mashup             | 25.0              |   |
| 25 | Photo/Video Compilation | Lifestyle | Mashup             | 7.0               |   |
| 26 | Photo/Video Compilation | Lifestyle | Bridge             | 29.0              |   |

```

29 Photo/Video Compilation Lifestyle Chorus 29.0
30 Photo/Video Compilation Lifestyle Beginning 29.0

      Music Genre
5   Dance/Electronic
8       R&B/Soul
13      Pop
16   Hip-Hop/Rap
17      Indie
24   Dance/Electronic
25   Dance/Electronic
26      Pop
29      Pop
30      Pop

```

## 0.9 Show all trends of snapchat

```
[20]: df.head(2)

[20]:          Platform    Month        Trend    Trend Type \
0  Instagram Reels  2022/06  DJ Wait A Minute  Montage, Vlog
1  Instagram Reels  2022/06       Guud Girls  Montage, Vlog

                           Music           Video Style     Theme \
0  DJ Wait A Minute - DJ Exe (2022)  Photo/Video Compilation  Lifestyle
1  Guud Girls - Breaking Beattz (2021)  Photo/Video Compilation  Lifestyle

      Part of Song  Video length in S  Music  Genre
0      Prelude            29.0      House
1      Prelude            29.0      House
```

```
[21]: df[df.Platform == 'Snapchat']['Trend'].unique()

[21]: array(['Veterans Day', 'Election Day', '111122', 'Black Panther',
       'Blood Moon', 'Football', 'Gym', 'Friendsgiving', 'Winter Fashion',
       'Cinco de Mayo'], dtype=object)
```

## 0.10 Show all the records for video length which has maximum music

```
[22]: df['Video length in S'].value_counts()

[22]: 29.000000    20
      30.000000     4
      22.000000     4
      7.000000      3
     10.000000     2
```

```

23.978723      2
13.000000      2
15.000000      2
52.000000      2
3.000000       1
12.000000      1
8.000000       1
54.000000      1
14.000000      1
18.000000      1
4.000000       1
25.000000      1

```

Name: Video length in S, dtype: int64

```
[23]: df[df['Video length in S'] == 29.000000]
```

|    | Platform        | Month   | Trend                            | Type                     |
|----|-----------------|---------|----------------------------------|--------------------------|
| 0  | Instagram Reels | 2022/06 | DJ Wait A Minute                 | Montage, Vlog            |
| 1  | Instagram Reels | 2022/06 | Guud Girls                       | Montage, Vlog            |
| 2  | Instagram Reels | 2022/06 | Running Up That Hill             | Cosplay, Montage         |
| 3  | Instagram Reels | 2022/06 | Gogo Dance                       | Vlog, Act Out/Dance      |
| 5  | Instagram Reels | 2022/06 | Still Don't Know My Name (Remix) | Montage                  |
| 6  | Instagram Reels | 2022/06 | Manifest                         | Montage (Template), Vlog |
| 7  | Instagram Reels | 2022/06 | Fur                              | Montage, Vlog            |
| 8  | Instagram Reels | 2022/06 | BREAK MY SOUL                    | Montage                  |
| 9  | Instagram Reels | 2022/06 | BILLIE EILISH                    | Montage                  |
| 11 | Instagram Reels | 2022/06 | The Night We Met                 | Montage                  |
| 13 | Instagram Reels | 2022/07 | Angel Eyes                       | Montage                  |
| 14 | Instagram Reels | 2022/07 | MASSIVE                          | Montage                  |
| 15 | Instagram Reels | 2022/07 | Vegas                            | Montage                  |
| 18 | Instagram Reels | 2022/07 | The Shade                        | Montage                  |
| 21 | Instagram Reels | 2022/08 | Wind                             | Montage                  |
| 22 | Instagram Reels | 2022/08 | One Step at a Time               | Montage                  |
| 26 | Instagram Reels | 2022/08 | SLIPPING THROUGH MY FINGERS      | Montage                  |
| 28 | Instagram Reels | 2022/08 | Leave Before You Love Me         | Montage                  |
| 29 | Instagram Reels | 2022/08 | Hold Me Closer                   | Montage                  |
| 30 | Instagram Reels | 2022/08 | Overcome                         | Montage                  |

|    |                                    |
|----|------------------------------------|
| 9  | Transition, Montage                |
| 11 | Montage, Vlog                      |
| 13 | Montage                            |
| 14 | Montage (Template)                 |
| 15 | Montage, Vlog                      |
| 18 | Montage, Vlog                      |
| 21 | Montage (Template)                 |
| 22 | Act Out + Text on Screen + Montage |
| 26 | Montage                            |
| 28 | Montage (Template)                 |
| 29 | Montage                            |
| 30 | Montage                            |

| Music \ |                                                   |  |
|---------|---------------------------------------------------|--|
| 0       | DJ Wait A Minute - DJ Exe (2022)                  |  |
| 1       | Guud Girls - Breaking Beattz (2021)               |  |
| 2       | Running Up That Hill (A Deal with God) - Kate ... |  |
| 3       | Gogo Dance - El Alfa (2022)                       |  |
| 5       | Still Don't Know My Name (Remix) - Kimotion (2... |  |
| 6       | Manifest Remix - Tiffany Laibhen-Spencer (2022)   |  |
| 7       | Fur - Endor (Fur Elise Remix) (2020)              |  |
| 8       | BREAK MY SOUL - Beyoncé (2022)                    |  |
| 9       | BILLIE EILISH - Armani White (2022)               |  |
| 11      | The Night We Met - Lord Huron (2015)              |  |
| 13      | Angel Eyes - ABBA (1979)                          |  |
| 14      | MASSIVE - Drake (2022)                            |  |
| 15      | Vegas - Doja Cat (2022)                           |  |
| 18      | THE SHADE - Rex Orange County (2022)              |  |
| 21      | Wind - Prm. (2020)                                |  |
| 22      | One Step At A Time - Jordin Sparks (2007)         |  |
| 26      | Slipping Through my Fingers - ABBA (1981) Cove... |  |
| 28      | Leave Before You Love Me - Marshmello ft. Jona... |  |
| 29      | Hold Me Closer - Elton John ft. Britney Spears... |  |
| 30      | Overcome - Skott (2022)                           |  |

|    | Video Style                           | Theme \   |
|----|---------------------------------------|-----------|
| 0  | Photo/Video Compilation               | Lifestyle |
| 1  | Photo/Video Compilation               | Lifestyle |
| 2  | Selfie Video, Photo/Video Compilation | Lifestyle |
| 3  | Selfie Video                          | Lifestyle |
| 5  | Photo/Video Compilation               | Lifestyle |
| 6  | Photo/Video Compilation               | Lifestyle |
| 7  | Photo/Video Compilation               | Lifestyle |
| 8  | Photo/Video Compilation               | Lifestyle |
| 9  | Selfie Video, Photo/Video Compilation | Lifestyle |
| 11 | Photo/Video Compilation               | Lifestyle |
| 13 | Photo/Video Compilation               | Lifestyle |

|    |                         |                   |
|----|-------------------------|-------------------|
| 14 | Photo/Video Compilation | Lifestyle         |
| 15 | Photo/Video Compilation | Lifestyle         |
| 18 | Photo/Video Compilation | Lifestyle         |
| 21 | Photo/Video Compilation | Lifestyle         |
| 22 | Selfie Video            | Lifestyle         |
| 26 | Photo/Video Compilation | Lifestyle         |
| 28 | Photo/Video Compilation | Lifestyle, Travel |
| 29 | Photo/Video Compilation | Lifestyle         |
| 30 | Photo/Video Compilation | Lifestyle         |

|    | Part of Song       | Video length in S | Music Genre      |
|----|--------------------|-------------------|------------------|
| 0  | Prelude            | 29.0              | House            |
| 1  | Prelude            | 29.0              | House            |
| 2  | Prelude, Chorus    | 29.0              | Pop              |
| 3  | Beginning          | 29.0              | Latin Urbano     |
| 5  | Mashup             | 29.0              | Dance/Electronic |
| 6  | Mashup             | 29.0              | Dance/Electronic |
| 7  | Prelude            | 29.0              | Dance/Electronic |
| 8  | Pre-Chorus, Chorus | 29.0              | R&B/Soul         |
| 9  | Chorus             | 29.0              | Hip-Hop/Rap      |
| 11 | Prelude            | 29.0              | Indie            |
| 13 | Beginning          | 29.0              | Pop              |
| 14 | Interlude          | 29.0              | Hip-Hop/Rap      |
| 15 | Chorus, Verse 1    | 29.0              | Hip-Hop/Rap      |
| 18 | Chorus             | 29.0              | Indie            |
| 21 | Beginning          | 29.0              | Classical        |
| 22 | Beginning          | 29.0              | Pop              |
| 26 | Bridge             | 29.0              | Pop              |
| 28 | Pre-chorus         | 29.0              | Pop              |
| 29 | Chorus             | 29.0              | Pop              |
| 30 | Beginning          | 29.0              | Pop              |

## 0.11 Show the records for each mashup

[24] : df.head(2)

|   | Platform                            | Month                   | Trend            | Trend Type    | \ |
|---|-------------------------------------|-------------------------|------------------|---------------|---|
| 0 | Instagram Reels                     | 2022/06                 | DJ Wait A Minute | Montage, Vlog |   |
| 1 | Instagram Reels                     | 2022/06                 | Guud Girls       | Montage, Vlog |   |
|   | Music                               | Video Style             | Theme            | \             |   |
| 0 | DJ Wait A Minute - DJ Exe (2022)    | Photo/Video Compilation | Lifestyle        |               |   |
| 1 | Guud Girls - Breaking Beattz (2021) | Photo/Video Compilation | Lifestyle        |               |   |
|   | Part of Song                        | Video length in S       | Music Genre      |               |   |
| 0 | Prelude                             | 29.0                    | House            |               |   |
| 1 | Prelude                             | 29.0                    | House            |               |   |

```
[25]: df.groupby(['Part of Song']).get_group('Mashup')
```

```
[25]:          Platform    Month            Trend \
5   Instagram Reels  2022/06  Still Don't Know My Name (Remix)
6   Instagram Reels  2022/06                Manifest
20  Instagram Reels  2022/07                Ianasher
24  Instagram Reels  2022/08  How Deep Is Your Love Mashup
25  Instagram Reels  2022/08        i jus flipped a switch

          Trend Type \
5                  Montage
6  Montage (Template), Vlog
20             Montage, Vlog
24             Montage
25             Montage

          Music \
5  Still Don't Know My Name (Remix) - Kimotion (2...
6  Manifest Remix - Tiffany Laibhen-Spencer (2022)
20 Stereo Love - Edward Maya, Vika Jigulina (2009...
24      How deep is your love? - DJ Replica (2022)
25 Remix by journalbymoon (Nonstop - Drake ft. Qu...

          Video Style     Theme Part of Song  Video length in S \
5  Photo/Video Compilation  Lifestyle  Mashup        29.000000
6  Photo/Video Compilation  Lifestyle  Mashup        29.000000
20 Photo/Video Compilation  Lifestyle  Mashup        23.978723
24 Photo/Video Compilation  Lifestyle  Mashup        25.000000
25 Photo/Video Compilation  Lifestyle  Mashup         7.000000

          Music Genre
5  Dance/Electronic
6  Dance/Electronic
20 Dance/Electronic
24 Dance/Electronic
25 Dance/Electronic
```

## 0.12 Show all records for video length greater than 50 or less than 10

```
[26]: df[(df['Video length in S']>50) | (df['Video length in S']<10)]
```

```
[26]:          Platform    Month            Trend \
4   Instagram Reels  2022/06        Zedsly
10  Instagram Reels  2022/06        Outlands
12  Instagram Reels  2022/07       mha4bii
17  Instagram Reels  2022/07  A Beautiful View
19  Instagram Reels  2022/07  Before, After
```

|    |                 |         |                        |
|----|-----------------|---------|------------------------|
| 25 | Instagram Reels | 2022/08 | i jus flipped a switch |
| 27 | Instagram Reels | 2022/08 | Body Language Sickmix  |
| 40 | Snapchat        | 2022/11 | Election Day           |
| 44 | Snapchat        | 2022/11 | Football               |

|    | Trend    | Type             | \ |
|----|----------|------------------|---|
| 4  | Montage  | (Template)       |   |
| 10 | Montage  | (Template), Vlog |   |
| 12 | Lip Sync | (music)          |   |
| 17 |          | Montage          |   |
| 19 |          | Transition       |   |
| 25 |          | Montage          |   |
| 27 |          | Montage, Vlog    |   |
| 40 |          | Transition, Vlog |   |
| 44 |          | Montage, Vlog    |   |

|    | Music                                             | \        |
|----|---------------------------------------------------|----------|
| 4  | The Way I Are - Timbaland                         | (2007)   |
| 10 | Outlands - Spencer Welling                        | (2022)   |
| 12 | 1, 2, 3 - Sofia Reyes feat. Jason Derulo, & De... |          |
| 17 | Bittersweet Symphony - The Verve                  | (1997)   |
| 19 |                                                   | NO VALUE |
| 25 | Remix by journalbymoon (Nonstop - Drake ft. Qu... |          |
| 27 | Body Language Sickmix - Sickick                   | (2022)   |
| 40 |                                                   | NO VALUE |
| 44 | Hayya Hayya (Better Together) - AISHA, Davido,... |          |

|    | Video Style             | Theme       | Part of Song | Video length in S | \    |
|----|-------------------------|-------------|--------------|-------------------|------|
| 4  | Photo/Video Compilation | Lifestyle   | Instrumental | 8.0               |      |
| 10 | Photo/Video Compilation | Lifestyle   | Instrumental | 4.0               |      |
| 12 | Selfie Video            | Lifestyle   | Bridge       | 7.0               |      |
| 17 | Photo/Video Compilation | Travel      | Prelude      | 7.0               |      |
| 19 | Photo/Video Compilation | Lifestyle   | NO VALUE     | 3.0               |      |
| 25 | Photo/Video Compilation | Lifestyle   | Mashup       | 7.0               |      |
| 27 | Photo/Video Compilation | Lifestyle   | Instrumental | 54.0              |      |
| 40 |                         | Promotional | Political    | NO VALUE          | 52.0 |
| 44 |                         | Promotional | Sports       | Verse             | 52.0 |

|    | Music Genre      |
|----|------------------|
| 4  | Dance/Electronic |
| 10 | Instrumental     |
| 12 | Latin Urbano     |
| 17 | Indie            |
| 19 | NO VALUE         |
| 25 | Dance/Electronic |
| 27 | Dance/Electronic |
| 40 | NO VALUE         |

### 0.13 Find trend and trend type for snapchat

```
[27]: df[df.Platform== 'Snapchat']['Trend'].sum()
```

```
[27]: 'Veterans DayElection Day111122Black PantherBlood  
MoonFootballGymFriendsgivingWinter FashionCinco de Mayo'
```

```
[28]: df[df.Platform== 'Snapchat']['Trend Type'].sum()
```

```
[28]: 'Text on ScreenTransition, VlogText on ScreenHashtagTransition, MontageMontage,  
VlogMontage, VlogHashtag + ChallengeMontage, VlogDance + Vlog'
```

### 0.14 Show all the records of pop genre

```
[29]: df.groupby('Music Genre').get_group('Pop')
```

|    | Platform        | Month   | Trend                       |
|----|-----------------|---------|-----------------------------|
| 2  | Instagram Reels | 2022/06 | Running Up That Hill        |
| 13 | Instagram Reels | 2022/07 | Angel Eyes                  |
| 22 | Instagram Reels | 2022/08 | One Step at a Time          |
| 26 | Instagram Reels | 2022/08 | SLIPPING THROUGH MY FINGERS |
| 28 | Instagram Reels | 2022/08 | Leave Before You Love Me    |
| 29 | Instagram Reels | 2022/08 | Hold Me Closer              |
| 30 | Instagram Reels | 2022/08 | Overcome                    |
| 31 | Instagram Reels | 2022/06 | YetToCome                   |
| 32 | Instagram Reels | 2022/07 | #MyBTStory                  |
| 33 | Instagram Reels | 2022/06 | #nayeon                     |
| 35 | Instagram Reels | 2022/07 | #SparklingChallenge         |
| 36 | Instagram Reels | 2022/07 | #BEFIRST                    |
| 37 | Instagram Reels | 2022/08 | #PinkVenomChallenge         |
| 38 | Instagram Reels | 2022/08 | Outfit Transitions          |

|    | Trend Type                         |
|----|------------------------------------|
| 2  | Cosplay, Montage                   |
| 13 | Montage                            |
| 22 | Act Out + Text on Screen + Montage |
| 26 | Montage                            |
| 28 | Montage (Template)                 |
| 29 | Montage                            |
| 30 | Montage                            |
| 31 | Hashtag                            |
| 32 | Hashtag + Challenge                |
| 33 | Dance                              |
| 35 | Dance + Challenge                  |

Music \

- 2 Running Up That Hill (A Deal with God) - Kate ...  
13 Angel Eyes - ABBA (1979)  
22 One Step At A Time - Jordin Sparks (2007)  
26 Slipping Through my Fingers - ABBA (1981) Cove...  
28 Leave Before You Love Me - Marshmello ft. Jona...  
29 Hold Me Closer - Elton John ft. Britney Spears...  
30 Overcome - Skott (2022)  
31 Yet To Come (The Most Beautiful Moment) - BTS ...  
32 BTS  
33 POP! - Nayeon (2022)  
35 Sparkling - Chung Ha (2022)  
36 Scream - Be:First (2022)  
37 Pink Venom - BLACKPINK (2022)  
38 WYAT (Where You At) - SB19 (2022)

## Video Style

## Theme \

- |    |                                       |                   |
|----|---------------------------------------|-------------------|
| 2  | Selfie Video, Photo/Video Compilation | Lifestyle         |
| 13 | Photo/Video Compilation               | Lifestyle         |
| 22 | Selfie Video                          | Lifestyle         |
| 26 | Photo/Video Compilation               | Lifestyle         |
| 28 | Photo/Video Compilation               | Lifestyle, Travel |
| 29 | Photo/Video Compilation               | Lifestyle         |
| 30 | Photo/Video Compilation               | Lifestyle         |
| 31 | Promotional                           | Entertainment     |
| 32 | Promotional                           | Entertainment     |
| 33 | Promotional                           | Entertainment     |
| 35 | Promotional                           | Entertainment     |
| 36 | Promotional                           | Entertainment     |
| 37 | Promotional                           | Entertainment     |
| 38 | Promotional                           | Entertainment     |

Part of Song Video length in S Music Genre

- |    |                      |      |     |
|----|----------------------|------|-----|
| 2  | Prelude, Chorus      | 29.0 | Pop |
| 13 | Beginning            | 29.0 | Pop |
| 22 | Beginning            | 29.0 | Pop |
| 26 | Bridge               | 29.0 | Pop |
| 28 | Pre-chorus           | 29.0 | Pop |
| 29 | Chorus               | 29.0 | Pop |
| 30 | Beginning            | 29.0 | Pop |
| 31 | Verse (instrumental) | 13.0 | Pop |
| 32 | Verse                | 15.0 | Pop |
| 33 | Chorus               | 13.0 | Pop |

```

35             Chorus          14.0      Pop
36             Chorus          15.0      Pop
37 Verse (instrumental)      10.0      Pop
38             Chorus          18.0      Pop

```

## 0.15 show the music for each video style

```
[30]: df.head(2)
```

```

[30]:           Platform    Month        Trend   Trend Type  \
0  Instagram Reels  2022/06  DJ Wait A Minute  Montage, Vlog
1  Instagram Reels  2022/06       Guud Girls  Montage, Vlog

                           Music          Video Style     Theme  \
0  DJ Wait A Minute - DJ Exe (2022)  Photo/Video Compilation  Lifestyle
1  Guud Girls - Breaking Beattz (2021)  Photo/Video Compilation  Lifestyle

  Part of Song  Video length in S Music Genre
0      Prelude            29.0      House
1      Prelude            29.0      House

```

```
[31]: df.groupby('Video Style')['Music'].sum()
```

```

[31]: Video Style
Photo/Video Compilation          DJ Wait A Minute - DJ Exe (2022)
Guud Girls - B...
Promotional                         Yet To Come (The Most Beautiful Moment)
- BTS ...
Selfie Video                      Gogo Dance - El Alfa (2022)
Sofia Rey...
Selfie Video                      One Step At A Time - Jordin Sparks
(2007)NO VA...
Selfie Video, Photo/Video Compilation  Running Up That Hill (A Deal with God)
- Kate ...
Name: Music, dtype: object

```

## 0.16 Find the records for “hashtag”

```
[65]: df[df['Trend Type'] == 'Hashtag']
```

```

[65]:           Platform    Month        Trend Trend Type  \
31  Instagram Reels  2022/06  YetToCome      Hashtag
42        Snapchat  2022/11  Black Panther      Hashtag

                           Music  Video Style  \
31 Yet To Come (The Most Beautiful Moment) - BTS ... Promotional

```

42

NO VALUE Promotional

|    | Theme         | Part of Song         | Video length in S | Music Genre |
|----|---------------|----------------------|-------------------|-------------|
| 31 | Entertainment | Verse (instrumental) | 13.0              | Pop         |
| 42 | Entertainment | NO VALUE             | 22.0              | R&B,hip-hop |

[32] : df.head(2)

|   | Platform        | Month   | Trend            | Trend Type \  |
|---|-----------------|---------|------------------|---------------|
| 0 | Instagram Reels | 2022/06 | DJ Wait A Minute | Montage, Vlog |
| 1 | Instagram Reels | 2022/06 | Guud Girls       | Montage, Vlog |

|   | Music                               | Video Style             | Theme \   |
|---|-------------------------------------|-------------------------|-----------|
| 0 | DJ Wait A Minute - DJ Exe (2022)    | Photo/Video Compilation | Lifestyle |
| 1 | Guud Girls - Breaking Beattz (2021) | Photo/Video Compilation | Lifestyle |

|   | Part of Song | Video length in S | Music Genre |
|---|--------------|-------------------|-------------|
| 0 | Prelude      | 29.0              | House       |
| 1 | Prelude      | 29.0              | House       |

1 Show the records with selfie video and latin urbano or in 2022/08

[49] : df[(df['Video Style'] == 'Selfie Video') & (df['Music Genre']== 'Latin Urbano') ↵ | (df['Month']== '2022/08')]

|    | Platform        | Month   | Trend \                      |
|----|-----------------|---------|------------------------------|
| 3  | Instagram Reels | 2022/06 | Gogo Dance                   |
| 12 | Instagram Reels | 2022/07 | mha4bii                      |
| 21 | Instagram Reels | 2022/08 | Wind                         |
| 22 | Instagram Reels | 2022/08 | One Step at a Time           |
| 23 | Instagram Reels | 2022/08 | Move Your Feet               |
| 24 | Instagram Reels | 2022/08 | How Deep Is Your Love Mashup |
| 25 | Instagram Reels | 2022/08 | i jus flipped a switch       |
| 26 | Instagram Reels | 2022/08 | SLIPPING THROUGH MY FINGERS  |
| 27 | Instagram Reels | 2022/08 | Body Language Sickmix        |
| 28 | Instagram Reels | 2022/08 | Leave Before You Love Me     |
| 29 | Instagram Reels | 2022/08 | Hold Me Closer               |
| 30 | Instagram Reels | 2022/08 | Overcome                     |
| 37 | Instagram Reels | 2022/08 | #PinkVenomChallenge          |
| 38 | Instagram Reels | 2022/08 | Outfit Transitions           |

|    | Trend Type \        |
|----|---------------------|
| 3  | Vlog, Act Out/Dance |
| 12 | Lip Sync (music)    |
| 21 | Montage (Template)  |

22 Act Out + Text on Screen + Montage  
 23 Montage (Template)  
 24 Montage  
 25 Montage  
 26 Montage  
 27 Montage, Vlog  
 28 Montage (Template)  
 29 Montage  
 30 Montage  
 37 Dance + Challenge  
 38 Transition

Music \

3 Gogo Dance - El Alfa (2022)  
 12 1, 2, 3 - Sofia Reyes feat. Jason Derulo, & De...  
 21 Wind - Prm. (2020)  
 22 One Step At A Time - Jordin Sparks (2007)  
 23 Move Your Feet - Autsin Millz (2022)  
 24 How deep is your love? - DJ Replica (2022)  
 25 Remix by journalbymoon (Nonstop - Drake ft. Qu...  
 26 Slipping Through my Fingers - ABBA (1981) Cove...  
 27 Body Language Sickmix - Sickick (2022)  
 28 Leave Before You Love Me - Marshmello ft. Jona...  
 29 Hold Me Closer - Elton John ft. Britney Spears...  
 30 Overcome - Skott (2022)  
 37 Pink Venom - BLACKPINK (2022)  
 38 WYAT (Where You At) - SB19 (2022)

|    | Video Style             | Theme             | Part of Song \       |
|----|-------------------------|-------------------|----------------------|
| 3  | Selfie Video            | Lifestyle         | Beginning            |
| 12 | Selfie Video            | Lifestyle         | Bridge               |
| 21 | Photo/Video Compilation | Lifestyle         | Beginning            |
| 22 | Selfie Video            | Lifestyle         | Beginning            |
| 23 | Photo/Video Compilation | Lifestyle         | Instrumental         |
| 24 | Photo/Video Compilation | Lifestyle         | Mashup               |
| 25 | Photo/Video Compilation | Lifestyle         | Mashup               |
| 26 | Photo/Video Compilation | Lifestyle         | Bridge               |
| 27 | Photo/Video Compilation | Lifestyle         | Instrumental         |
| 28 | Photo/Video Compilation | Lifestyle, Travel | Pre-chorus           |
| 29 | Photo/Video Compilation | Lifestyle         | Chorus               |
| 30 | Photo/Video Compilation | Lifestyle         | Beginning            |
| 37 | Promotional             | Entertainment     | Verse (instrumental) |
| 38 | Promotional             | Entertainment     | Chorus               |

| Video length in S | Music Genre       |
|-------------------|-------------------|
| 3                 | 29.0 Latin Urbano |
| 12                | 7.0 Latin Urbano  |

```

21          29.0      Classical
22          29.0        Pop
23         12.0  Dance/Electronic
24         25.0  Dance/Electronic
25          7.0  Dance/Electronic
26          29.0        Pop
27         54.0  Dance/Electronic
28          29.0        Pop
29          29.0        Pop
30          29.0        Pop
37          10.0        Pop
38          18.0        Pop

```

## 1.1 Count all the unique values for each month

[50] : df.head(2)

```

[50]:          Platform    Month            Trend    Trend Type \
0  Instagram Reels  2022/06  DJ Wait A Minute  Montage, Vlog
1  Instagram Reels  2022/06           Guud Girls  Montage, Vlog

                           Music            Video Style      Theme \
0  DJ Wait A Minute - DJ Exe (2022)  Photo/Video Compilation  Lifestyle
1  Guud Girls - Breaking Beattz (2021)  Photo/Video Compilation  Lifestyle

      Part of Song  Video length in S  Music  Genre
0      Prelude           29.0       House
1      Prelude           29.0       House

```

[58] : df.groupby('Month').nunique()

```

[58]:          Platform  Trend  Trend Type  Music  Video Style  Theme \
Month
2022/05          1      1          1      1          1      1
2022/06          1     14          9     14          4      2
2022/07          1     12          8     12          3      3
2022/08          1     12          6     12          3      3
2022/10          1      2          1      2          2      2
2022/11          1      7          6      2          3      7
NO VALUE         1      1          1      1          1      1

      Part of Song  Video length in S  Music  Genre
Month
2022/05           1                  1      1
2022/06           8                  4      8
2022/07          10                 7      6
2022/08           7                  7      3

```

|          |   |   |   |
|----------|---|---|---|
| 2022/10  | 2 | 2 | 2 |
| 2022/11  | 3 | 3 | 3 |
| NO VALUE | 1 | 1 | 1 |

## 1.2 Show all the record for video length less than 5 or on platform snapchat and theme is lifestyle

```
[68]: df[(df['Video length in S']<5) | (df['Platform']=='Snapchat') &
      ~(df['Theme']=='Lifestyle')]
```

```
[68]:
```

|    | Platform                               | Month        | Trend                   | Trend Type               | \ |
|----|----------------------------------------|--------------|-------------------------|--------------------------|---|
| 10 | Instagram Reels                        | 2022/06      | Outlands                | Montage (Template), Vlog |   |
| 19 | Instagram Reels                        | 2022/07      | Before, After           | Transition               |   |
| 41 | Snapchat                               | 2022/11      | 111122                  | Text on Screen           |   |
| 45 | Snapchat                               | 2022/10      | Gym                     | Montage, Vlog            |   |
|    |                                        |              | Music                   | Video Style              | \ |
| 10 | Outlands - Spencer Welling (2022)      |              | Photo/Video Compilation |                          |   |
| 19 |                                        |              | NO VALUE                | Photo/Video Compilation  |   |
| 41 |                                        |              | NO VALUE                | Selfie Video             |   |
| 45 | Ballin - A Boogie wit da Hoodie (2022) |              | Photo/Video Compilation |                          |   |
|    | Theme                                  | Part of Song | Video length in S       | Music Genre              |   |
| 10 | Lifestyle                              | Instrumental | 4.0                     | Instrumental             |   |
| 19 | Lifestyle                              | NO VALUE     | 3.0                     | NO VALUE                 |   |
| 41 | Lifestyle                              | NO VALUE     | 30.0                    | NO VALUE                 |   |
| 45 | Lifestyle                              | Chorus       | 30.0                    | R&B                      |   |

```
[ ]:
```



**project**

**8**

**Python - Women Clothing Reviews  
Sentiment Analysis**

# clothing-reviews-sentiment-analysis

February 24, 2024

```
[46]: import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from wordcloud import WordCloud, STOPWORDS
```

```
[47]: df=pd.read_csv("Customer Segmentation (Womens Clothing.csv")
df.head(1)
```

```
[47]:   Unnamed: 0  Clothing ID  Age  Title  \
0          0        767    33    NaN
  Review Text  Rating  Recommended IND  \
0  Absolutely wonderful - silky and sexy and comf...       4                  1
  Positive Feedback Count Division Name Department Name Class Name
0                      0           Initmates           Intimate  Intimates
```

```
[48]: df.shape
```

```
[48]: (23486, 11)
```

```
[49]: df.describe()
```

```
[49]:   Unnamed: 0  Clothing ID  Age  Rating  \
count  23486.000000  23486.000000  23486.000000  23486.000000
mean   11742.500000    918.118709   43.198544    4.196032
std    6779.968547   203.298980   12.279544    1.110031
min     0.000000    0.000000   18.000000    1.000000
25%    5871.250000   861.000000   34.000000    4.000000
50%   11742.500000   936.000000   41.000000    5.000000
75%   17613.750000  1078.000000   52.000000    5.000000
max   23485.000000  1205.000000   99.000000    5.000000
  Recommended IND  Positive Feedback Count
count      23486.000000                23486.000000
mean        0.822362                  2.535936
std         0.382216                  5.702202
```

```
min           0.000000           0.000000
25%          1.000000           0.000000
50%          1.000000           1.000000
75%          1.000000           3.000000
max          1.000000          122.000000
```

```
[50]: df.isnull().sum()
```

```
[50]: Unnamed: 0           0
Clothing ID          0
Age                  0
Title                3810
Review Text          845
Rating               0
Recommended IND      0
Positive Feedback Count 0
Division Name        14
Department Name      14
Class Name            14
dtype: int64
```

```
[51]: df=df.dropna()
```

```
[52]: df.isnull().sum()
```

```
[52]: Unnamed: 0           0
Clothing ID          0
Age                  0
Title                0
Review Text          0
Rating               0
Recommended IND      0
Positive Feedback Count 0
Division Name        0
Department Name      0
Class Name            0
dtype: int64
```

```
[53]: df.shape
```

```
[53]: (19662, 11)
```

```
[54]: df.head(1)
```

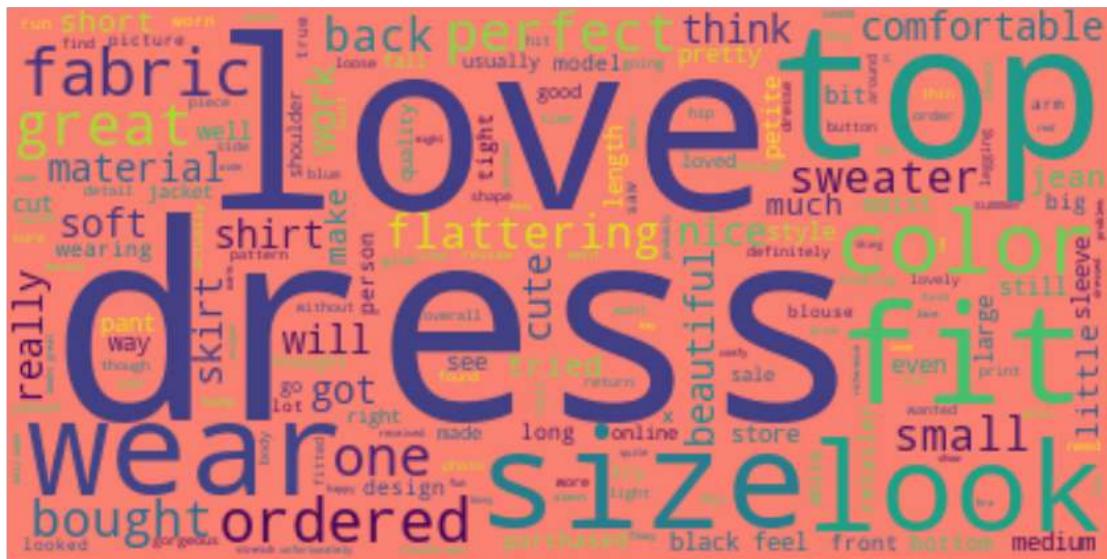
```
[54]:    Unnamed: 0  Clothing ID  Age           Title \
2              2          1077   60  Some major design flaws
```

Review Text Rating Recommended IND \ 2 I had such high hopes for this dress and reall... 3 0

Positive Feedback Count Division Name Department Name Class Name  
2 0 General Dresses Dresses

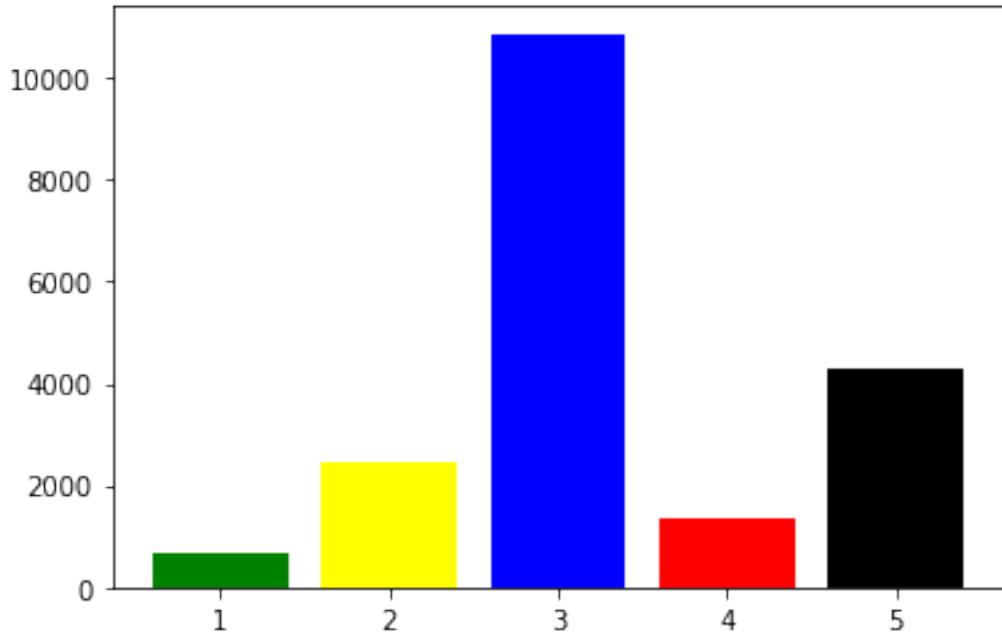
```
[115]: text = " ".join(i for i in df['Review Text'])

stopwords = set(STOPWORDS)
wordcloud = WordCloud(stopwords=stopwords, background_color="salmon").  
    generate(text)
plt.figure(figsize=(15,10))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```



```
[87]: ac=df.Rating.value_counts()
       a=df.Rating.unique()
       plt.bar(a,ac,color=['red','blue','yellow','green','black'])
```

[87]: <BarContainer object of 5 artists>



```
[73]: df.head(1)
```

```
[73]:   Unnamed: 0  Clothing ID  Age          Title \
2           2        1077    60  Some major design flaws

   Review Text  Rating  Recommended IND \
2  I had such high hopes for this dress and reall...            3             0

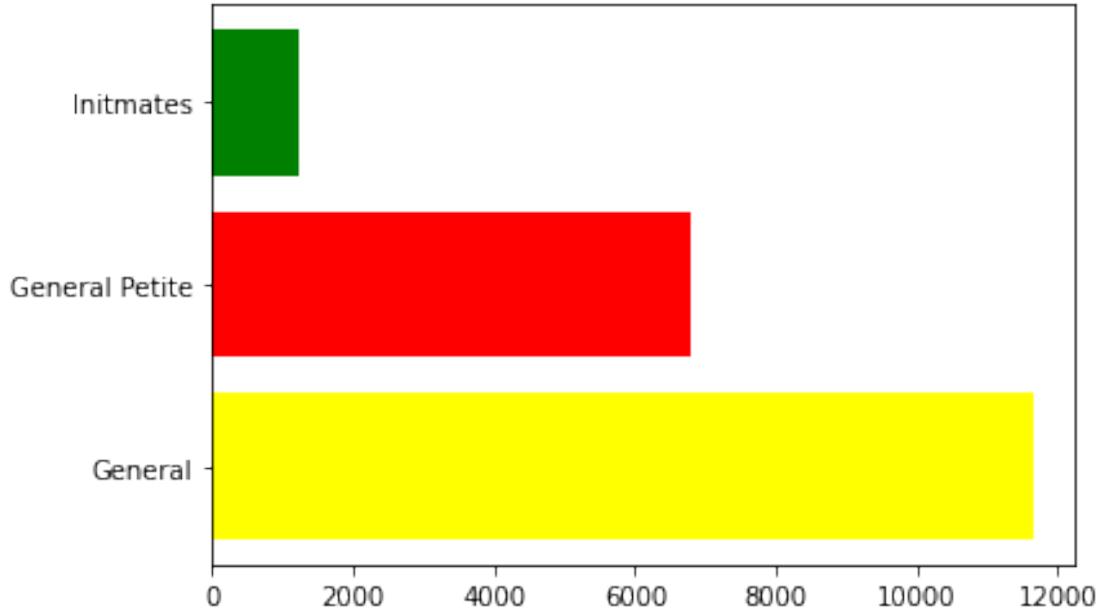
   Positive Feedback Count Division Name Department Name Class Name
2                           0                  General           Dresses       Dresses
```

```
[81]: dnn=df['Division Name'].value_counts()
dn=df['Division Name'].unique()
dnn
```

```
[81]: General          11664
General Petite      6778
Initmates          1220
Name: Division Name, dtype: int64
```

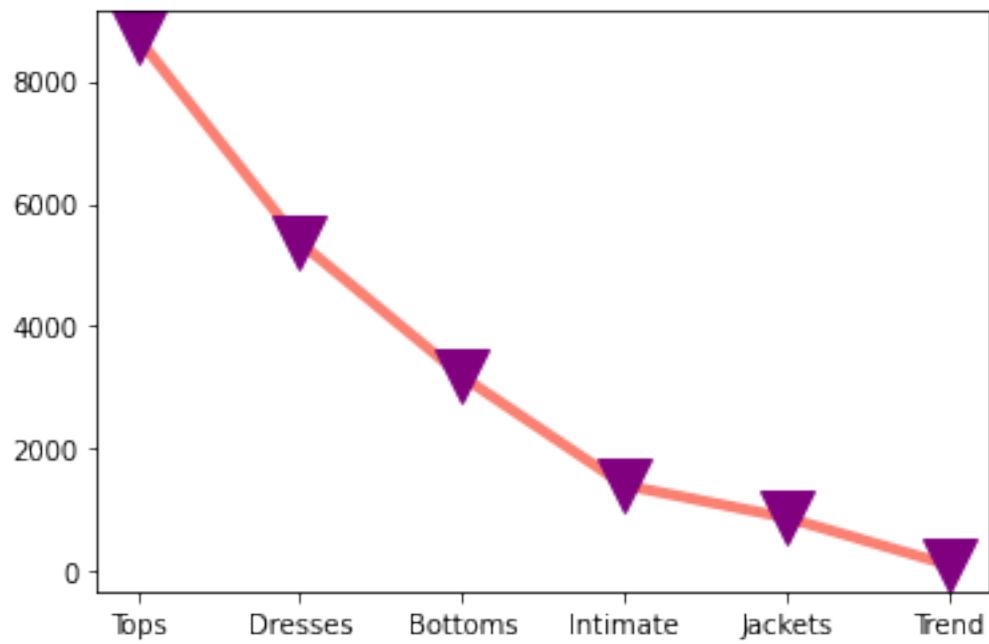
```
[90]: plt.bart(dn,dnn,color={'green','yellow','red'})
```

```
[90]: <BarContainer object of 3 artists>
```



```
[116]: plt.plot(df['Department Name'].  
             ↪value_counts(),marker="v",linewidth=4,color="salmon",ms=20,mfc="purple",mec="purple")
```

```
[116]: [matplotlib.lines.Line2D at 0x1339b520]
```



[ ]:

project

9

**Python - NBA Data Wrangling &  
Exploration**

# NBA Teams Offensive Data Exploration (1980-2021)

by **Vasanth Mohan**

## Summary of Project

For this project, I am mainly interested in conducting data exploration and analysis on the offensive stats and characteristics of different NBA teams based on Finals ranking which is a new column I will create that contains 4 values: Champion, Runner-Up, Knocked Out and Never Qualified. Knocked Out and Never Qualified implies that they have either been knocked out of or never qualified for the NBA playoffs. Some stats that you will see me analyze and visualize are Margin of Victory(MOV), 3P%, Age and shot attempts.

Part I - NBA Web Scraping.ipynb - To begin, I scraped data from the [Basketball Reference](#) website, which contains each team's performances throughout the years. I scraped a total of 4 different stats tables from the website and stored them in 4 different datasets. In this notebook, I used 3 different packages: Pandas, BeautifulSoup and Requests.

Part II - NBA Data Cleaning.ipynb - In this notebook, I conducted the cleaning process. Some steps I took here are dealing with null values, dropping unneeded columns, converting datatypes and cleaning up the values. After cleaning up the data, I merged 4 of the datasets into 2. I have also created a new column to indicate the NBA teams' Finals ranking in this notebook. One consists of the teams' total stats per year and one consists of the average stats per game for each year. In this notebook, I used 2 different packages: Pandas and Numpy.

Part III - NBA Data Exploration.ipynb - In this notebook, you'll find my analysis and visualizations of the stats. I started off by analyzing the total stats first to get a broad picture view by conducting and creating visuals for univariate and bivariate exploration. Afterward, I moved onto the average stats per game of each year where I conducted the same type of explorations along with multivariate exploration. You will find that I have also created a couple categorical variables for my analysis as well. In this notebook, I used 5 different packages: Pandas, Numpy, Seaborn, Matplotlib and Warnings. You will find the majority of the multivariate exploration near the end.

## Installation

- \* BeautifulSoup
- \* Requests
- \* Matplotlib
- \* Seaborn
- \* Numpy
- \* Pandas
- \* Warnings

## Datasets

`total_stats_df.csv` - Uncleaned dataset that contains the total stats from 1980-2021 scraped from the Basketball Reference website.

`avg_stats_df.csv` - Uncleaned dataset that contains the average stats from 1980-2021 scraped from the Basketball Reference website.

`advanced_stats_df.csv` - Uncleaned dataset that contains the advanced stats from 1980-2021 scraped from the Basketball Reference website. This dataset includes variables such as MOV, ORtg and DRtg.

`advanced_stats_df.csv` - Uncleaned dataset that contains the season summary from 1980-2021 scraped from the Basketball Reference website. This dataset includes variables such as Champions and Runner-up.

`cleaned_total_stats.csv` - Cleaned dataset that contains the total stats from 1980-2021 scraped from the Basketball Reference website. This dataset all needed variables such as Finals rankings(Finals\_Rk).

`cleaned_avg_stats.csv` - Cleaned dataset that contains the average stats from 1980-2021 scraped from the Basketball Reference website. This dataset all needed variables such as Finals rankings(Finals\_Rk), Age and Wins.

# part-i-nba-web-scraping

February 24, 2024

## 1 NBA Web Scraping

### 1.1 Introduction

In this project, I am interested in analyzing the numbers and statistics since the NBA began tracking 3-pointers, which was the year 1980. My goal for this project is to explore the performances of the NBA teams. Rather than each individual NBA player's performance, I am more interested in their team's offensive performance. So, for the first step of my personal data exploration project, I will be gathering all of the data that I deem necessary. In this short notebook, I will scrape data from the [Basketball Reference](#) website, which contains each team's performances throughout the years. I will only be focusing on the years 1980-2021; I will not include 2022 because the season is still in progress at the time of this writing.

```
[13]: # import packages
import requests
import bs4 as BeautifulSoup
import pandas as pd

pd.set_option('display.max_columns', None)

[14]: years = list(range(1980,2022))
url_start = "https://www.basketball-reference.com/leagues/NBA_{}.html"

[15]: # loop to get urls for each years
for year in years:
    url = url_start.format(year)
    data = requests.get(url)

    with open("team_stats/{}.html".format(year), "w+") as f:
        f.write(data.text)

[16]: # create lists to store dataframes
total_dfs = []
avg_dfs = []
advanced_dfs = []

for year in years:
```

```

with open("team_stats/{}.html".format(year)) as f:
    page = f.read()
soup = BeautifulSoup.BeautifulSoup(page, 'html.parser')
# decompose unnecessary parts of table
soup.find('div', class_= "table_container").decompose()
soup.find('tr', class_= "over_header").decompose()

total_stats_table = soup.find(id = "div_totals-team")
avg_stats_table = soup.find(id = "div_per_game-team")
advanced_table = soup.find(id = "advanced-team")

total_stats = pd.read_html(str(total_stats_table))[0]
avg_stats = pd.read_html(str(avg_stats_table))[0]
advanced_stats = pd.read_html(str(advanced_table))[0]

# Add a Year column
total_stats["Year"] = year
avg_stats["Year"] = year
advanced_stats["Year"] = year

total_dfs.append(total_stats)
avg_dfs.append(avg_stats)
advanced_dfs.append(advanced_stats)

```

I noticed that the tables do not include the NBA champion for that year which is something I would like to include in my analysis. Therefore, we will have to scrap the champions and runner-ups from this [Basketball Reference webpage](#). The steps are the same as before, but without the loop.

[17]:

```

url_2nd = 'https://www.basketball-reference.com/playoffs/'
data = requests.get(url_2nd)
with open("team_stats/champions.html", "w+") as f:
    f.write(data.text)

```

[18]:

```

with open("team_stats/champions.html") as f:
    page = f.read()
soup = BeautifulSoup.BeautifulSoup(page, 'html.parser')
# decompose unnecessary parts of table
soup.find('tr', class_= "over_header").decompose()
champions_table = soup.find(id = "div_champions_index")
champions_stat = pd.read_html(str(champions_table))[0]

```

## 1.2 Storing Data

```
[19]: total_stats_df = pd.concat(total_dfs)
total_stats_df.head()
```

```
[19]:    Rk              Team   G    MP   FG   FGA   FG%   3P   3PA   3P% \
0  1.0  San Antonio Spurs*  82  19755  3856  7738  0.498   52   206   0.252
1  2.0  Los Angeles Lakers*  82  19880  3898  7368  0.529   20   100   0.200
2  3.0  Cleveland Cavaliers  82  19930  3811  8041  0.474   36   187   0.193
3  4.0  New York Knicks    82  19780  3802  7672  0.496   42   191   0.220
4  5.0  Boston Celtics*    82  19880  3617  7387  0.490   162   422   0.384

      2P   2PA   2P%   FT   FTA   FT%   ORB   DRB   TRB   AST   STL   BLK \
0  3804  7532  0.505  2024  2528  0.801  1153  2515  3668  2326  771  333
1  3878  7268  0.534  1622  2092  0.775  1085  2653  3738  2413  774  546
2  3775  7854  0.481  1702  2205  0.772  1307  2381  3688  2108  764  342
3  3760  7481  0.503  1698  2274  0.747  1236  2303  3539  2265  881  457
4  3455  6965  0.496  1907  2449  0.779  1227  2457  3684  2198  809  308

      TOV   PF   PTS Year
0  1589  2103  9788  1980
1  1639  1784  9438  1980
2  1370  1934  9360  1980
3  1613  2168  9344  1980
4  1539  1974  9303  1980
```

```
[20]: avg_stats_df = pd.concat(avg_dfs)
avg_stats_df.head()
```

```
[20]:    Rk              Team   G    MP   FG   FGA   FG%   3P   3PA   3P% \
0  1.0  San Antonio Spurs*  82  240.9  47.0  94.4  0.498   0.6   2.5   0.252
1  2.0  Los Angeles Lakers*  82  242.4  47.5  89.9  0.529   0.2   1.2   0.200
2  3.0  Cleveland Cavaliers  82  243.0  46.5  98.1  0.474   0.4   2.3   0.193
3  4.0  New York Knicks    82  241.2  46.4  93.6  0.496   0.5   2.3   0.220
4  5.0  Boston Celtics*    82  242.4  44.1  90.1  0.490   2.0   5.1   0.384

      2P   2PA   2P%   FT   FTA   FT%   ORB   DRB   TRB   AST   STL   BLK \
0  46.4  91.9  0.505  24.7  30.8  0.801  14.1  30.7  44.7  28.4  9.4  4.1
1  47.3  88.6  0.534  19.8  25.5  0.775  13.2  32.4  45.6  29.4  9.4  6.7
2  46.0  95.8  0.481  20.8  26.9  0.772  15.9  29.0  45.0  25.7  9.3  4.2
3  45.9  91.2  0.503  20.7  27.7  0.747  15.1  28.1  43.2  27.6  10.7  5.6
4  42.1  84.9  0.496  23.3  29.9  0.779  15.0  30.0  44.9  26.8  9.9  3.8

      TOV   PF   PTS Year
0  19.4  25.6  119.4  1980
1  20.0  21.8  115.1  1980
2  16.7  23.6  114.1  1980
```

```
3 19.7 26.4 114.0 1980
4 18.8 24.1 113.5 1980
```

```
[21]: advanced_stats_df = pd.concat(advanced_dfs)
advanced_stats_df.head()
```

```
[21]:      Rk              Team    Age     W     L    PW    PL    MOV    SOS    SRS  \
0  1.0      Boston Celtics*  27.3  61.0  21.0   60   22  7.79 -0.42  7.37
1  2.0  Los Angeles Lakers*  26.2  60.0  22.0   55   27  5.90 -0.51  5.40
2  3.0  Seattle SuperSonics*  27.0  56.0  26.0   53   29  4.66 -0.42  4.24
3  4.0  Philadelphia 76ers*  27.0  59.0  23.0   52   30  4.22 -0.18  4.04
4  5.0  Milwaukee Bucks*   25.3  49.0  33.0   51   31  3.94 -0.37  3.57

      ORtg    DRtg    NRtg    Pace    FTr    3PAr    TS%  Unnamed: 17    eFG%    TOV%  \
0  109.4  101.9     7.5  102.6  0.332  0.057  0.550           NaN  0.501  15.4
1  109.5  103.9     5.6  104.1  0.284  0.014  0.569           NaN  0.530  16.5
2  105.8  101.2     4.6  101.8  0.298  0.025  0.520           NaN  0.474  14.9
3  105.0  101.0     4.0  103.0  0.340  0.017  0.544           NaN  0.494  17.2
4  106.8  102.9     3.9  102.4  0.278  0.021  0.532           NaN  0.491  15.0

      ORB%  FT/FGA  Unnamed: 22    eFG%.1    TOV%.1    DRB%  FT/FGA.1  Unnamed: 27  \
0  34.8    0.258           NaN  0.475     16.5   67.8    0.234           NaN
1  32.6    0.220           NaN  0.475     14.0   66.9    0.181           NaN
2  36.4    0.229           NaN  0.463     15.4   67.9    0.221           NaN
3  33.5    0.262           NaN  0.460     15.5   66.7    0.217           NaN
4  35.2    0.212           NaN  0.467     16.2   63.8    0.229           NaN

      Arena    Attend.  Attend./G    Year
0      Boston Garden  596349.0  14664.0  1980
1      The Forum    582882.0  17505.0  1980
2  King County Domed Stadium       NaN  28726.0  1980
3      The Spectrum       NaN       NaN  1980
4      MECCA Arena       NaN       NaN  1980
```

```
[22]: champions_df = pd.DataFrame(champions_stat)
champions_df
```

```
[22]:      Year    Lg          Champion          Runner-Up  \
0  2021.0  NBA  Milwaukee Bucks  Phoenix Suns
1  2020.0  NBA  Los Angeles Lakers  Miami Heat
2  2019.0  NBA  Toronto Raptors  Golden State Warriors
3  2018.0  NBA  Golden State Warriors  Cleveland Cavaliers
4  2017.0  NBA  Golden State Warriors  Cleveland Cavaliers
..    ... ...
83    NaN  NaN           NaN           NaN
84  1950.0  NBA  Minneapolis Lakers  Syracuse Nationals
85  1949.0  BAA  Minneapolis Lakers  Washington Capitols
```

|    |                        |                 |                       |                        |               |   |
|----|------------------------|-----------------|-----------------------|------------------------|---------------|---|
| 86 | 1948.0                 | BAA             | Baltimore Bullets     | Philadelphia Warriors  |               |   |
| 87 | 1947.0                 | BAA             | Philadelphia Warriors |                        | Chicago Stags |   |
|    |                        |                 | Finals MVP            | Unnamed: 5             | Points        | \ |
| 0  | G. Antetokounmpo       |                 | NaN                   | G. Antetokounmpo       | (634)         |   |
| 1  | L. James               |                 | NaN                   | A. Davis               | (582)         |   |
| 2  | K. Leonard             |                 | NaN                   | K. Leonard             | (732)         |   |
| 3  | K. Durant              |                 | NaN                   | L. James               | (748)         |   |
| 4  | K. Durant              |                 | NaN                   | L. James               | (591)         |   |
| .. | ...                    | ...             | ...                   | ...                    | ...           |   |
| 83 | NaN                    | NaN             | NaN                   |                        | NaN           |   |
| 84 | NaN                    | NaN             | NaN                   | G. Mikan               | (376)         |   |
| 85 | NaN                    | NaN             | NaN                   | G. Mikan               | (303)         |   |
| 86 | NaN                    | NaN             | NaN                   | J. Fulks               | (282)         |   |
| 87 | NaN                    | NaN             | NaN                   | J. Fulks               | (222)         |   |
|    |                        |                 | Rebounds              | Assists                | Win Shares    |   |
| 0  | G. Antetokounmpo (269) |                 | J. Holiday (199)      | G. Antetokounmpo (3.7) |               |   |
| 1  | L. James (226)         |                 | L. James (184)        | A. Davis (4.5)         |               |   |
| 2  | D. Green (223)         |                 | D. Green (187)        | K. Leonard (4.9)       |               |   |
| 3  | D. Green (222)         |                 | L. James (198)        | L. James (5.2)         |               |   |
| 4  | K. Love (191)          |                 | L. James (141)        | L. James (4.3)         |               |   |
| .. | ...                    | ...             | ...                   | ...                    | ...           |   |
| 83 | NaN                    |                 | NaN                   |                        | NaN           |   |
| 84 | NaN                    | J. Pollard (56) |                       | G. Mikan (3.7)         |               |   |
| 85 | NaN                    | J. Pollard (39) |                       | G. Mikan (4.2)         |               |   |
| 86 | NaN                    | H. Dallmar (37) |                       | C. Simmons (2.5)       |               |   |
| 87 | NaN                    | H. Dallmar (16) |                       | J. Fulks (2.3)         |               |   |

```
[88 rows x 10 columns]

total_stats_df.to_csv('data/total_stats_df.csv')
avg_stats_df.to_csv('data/avg_stats_df.csv')
advanced_stats_df.to_csv('data/advanced_stats_df.csv')
champions_df.to_csv('data/champions_df.csv')
```

Please head on over to Part II - NBA Data Cleaning.ipynb to continue.

# part-ii-nba-data-cleaning

February 24, 2024

## 1 NBA Data Cleaning Process

### 1.1 Introduction

In this notebook, I will be conducting the cleaning process to make sure that the datasets are tidy for further exploration. After inspecting the website further, I realize that I can merge 2 of the 4 datasets into 1; more specifically, I want to merge the `avg_df` with the `advanced_df` dataset. One dataset will consist of the total stats and the other will consist of average stats; the latter will be a merged dataset. Furthermore, I also find that it would be helpful to my analysis if I merged the `champion_df` dataset with the new average dataset and total dataset to keep track of the winner and runner-up for that year. Furthermore, I am also particularly interested in the offensive stats such as the 3-point stats; therefore, you will see that I have excluded almost all of the defensive ones.

```
[2]: import pandas as pd
      import numpy as np

      pd.set_option('display.max_columns', None)
      pd.set_option('display.max_rows', None)
```

```
[3]: advanced_df = pd.read_csv('data/advanced_stats_df.csv')
      avg_df = pd.read_csv('data/avg_stats_df.csv')
      total_df = pd.read_csv('data/total_stats_df.csv')
      champions_df = pd.read_csv('data/champions_df.csv')
```

### 1.2 Examining Data

#### 1.2.1 `champions_df`

```
[4]: champions_df.head()
```

```
[4]:   Unnamed: 0    Year    Lg          Champion          Runner-Up \
0            0  2021.0  NBA    Milwaukee Bucks    Phoenix Suns
1            1  2020.0  NBA    Los Angeles Lakers    Miami Heat
2            2  2019.0  NBA    Toronto Raptors  Golden State Warriors
3            3  2018.0  NBA  Golden State Warriors  Cleveland Cavaliers
4            4  2017.0  NBA  Golden State Warriors  Cleveland Cavaliers
```

|   | Finals MVP       | Unnamed: 5 | Points           | \     |
|---|------------------|------------|------------------|-------|
| 0 | G. Antetokounmpo | NaN        | G. Antetokounmpo | (634) |
| 1 | L. James         | NaN        | A. Davis         | (582) |
| 2 | K. Leonard       | NaN        | K. Leonard       | (732) |
| 3 | K. Durant        | NaN        | L. James         | (748) |
| 4 | K. Durant        | NaN        | L. James         | (591) |

|   | Rebounds               | Assists          | Win Shares             |
|---|------------------------|------------------|------------------------|
| 0 | G. Antetokounmpo (269) | J. Holiday (199) | G. Antetokounmpo (3.7) |
| 1 | L. James (226)         | L. James (184)   | A. Davis (4.5)         |
| 2 | D. Green (223)         | D. Green (187)   | K. Leonard (4.9)       |
| 3 | D. Green (222)         | L. James (198)   | L. James (5.2)         |
| 4 | K. Love (191)          | L. James (141)   | L. James (4.3)         |

[5]: champions\_df.Year.min() # we only need the rows starting from 1980 in the year  
↳ column

[5]: 1947.0

[6]: champions\_df.shape

[6]: (88, 11)

[7]: champions\_df.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 88 entries, 0 to 87
Data columns (total 11 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   Unnamed: 0    88 non-null    int64  
 1   Year         84 non-null    float64 
 2   Lg            84 non-null    object  
 3   Champion     84 non-null    object  
 4   Runner-Up    84 non-null    object  
 5   Finals MVP   53 non-null    object  
 6   Unnamed: 5    0 non-null    float64 
 7   Points        84 non-null    object  
 8   Rebounds      80 non-null    object  
 9   Assists       84 non-null    object  
 10  Win Shares    84 non-null    object  
dtypes: float64(2), int64(1), object(8)
memory usage: 7.7+ KB
```

[8]: champions\_df[champions\_df.Year >= 1980].isnull().sum()
champions\_df[champions\_df.Year >= 1980].Champion.value\_counts()

```
[8]: Los Angeles Lakers      11
      Chicago Bulls          6
      San Antonio Spurs      5
      Boston Celtics          4
      Golden State Warriors   3
      Miami Heat              3
      Detroit Pistons         3
      Houston Rockets         2
      Milwaukee Bucks          1
      Toronto Raptors         1
      Cleveland Cavaliers     1
      Dallas Mavericks        1
      Philadelphia 76ers       1
      Name: Champion, dtype: int64
```

**champions\_df Observation** > In this dataset, we find that we only need 3 columns: Year, Champion, Runner-Up. We can also change the year column into an integer. Other than that, everything else looks great! The names of the teams in our desired timeframe seems clean and ready to go.

```
### advanced_df
```

```
[9]: advanced_df.head()
```

|   | Unnamed: 0 | Rk  | Team                 | Age  | W    | L    | PW | PL | MOV  | \ |
|---|------------|-----|----------------------|------|------|------|----|----|------|---|
| 0 | 0          | 1.0 | Boston Celtics*      | 27.3 | 61.0 | 21.0 | 60 | 22 | 7.79 |   |
| 1 | 1          | 2.0 | Los Angeles Lakers*  | 26.2 | 60.0 | 22.0 | 55 | 27 | 5.90 |   |
| 2 | 2          | 3.0 | Seattle SuperSonics* | 27.0 | 56.0 | 26.0 | 53 | 29 | 4.66 |   |
| 3 | 3          | 4.0 | Philadelphia 76ers*  | 27.0 | 59.0 | 23.0 | 52 | 30 | 4.22 |   |
| 4 | 4          | 5.0 | Milwaukee Bucks*     | 25.3 | 49.0 | 33.0 | 51 | 31 | 3.94 |   |

|   | SOS   | SRS  | ORtg  | DRtg  | NRtg | Pace  | FTr   | 3PAr  | TS%   | Unnamed: 17 | \   |
|---|-------|------|-------|-------|------|-------|-------|-------|-------|-------------|-----|
| 0 | -0.42 | 7.37 | 109.4 | 101.9 | 7.5  | 102.6 | 0.332 | 0.057 | 0.550 |             | NaN |
| 1 | -0.51 | 5.40 | 109.5 | 103.9 | 5.6  | 104.1 | 0.284 | 0.014 | 0.569 |             | NaN |
| 2 | -0.42 | 4.24 | 105.8 | 101.2 | 4.6  | 101.8 | 0.298 | 0.025 | 0.520 |             | NaN |
| 3 | -0.18 | 4.04 | 105.0 | 101.0 | 4.0  | 103.0 | 0.340 | 0.017 | 0.544 |             | NaN |
| 4 | -0.37 | 3.57 | 106.8 | 102.9 | 3.9  | 102.4 | 0.278 | 0.021 | 0.532 |             | NaN |

|   | eFG%  | TOV% | ORB% | FT/FGA | Unnamed: 22 | eFG%.1 | TOV%.1 | DRB% | FT/FGA.1 | \     |  |
|---|-------|------|------|--------|-------------|--------|--------|------|----------|-------|--|
| 0 | 0.501 | 15.4 | 34.8 | 0.258  |             | NaN    | 0.475  | 16.5 | 67.8     | 0.234 |  |
| 1 | 0.530 | 16.5 | 32.6 | 0.220  |             | NaN    | 0.475  | 14.0 | 66.9     | 0.181 |  |
| 2 | 0.474 | 14.9 | 36.4 | 0.229  |             | NaN    | 0.463  | 15.4 | 67.9     | 0.221 |  |
| 3 | 0.494 | 17.2 | 33.5 | 0.262  |             | NaN    | 0.460  | 15.5 | 66.7     | 0.217 |  |
| 4 | 0.491 | 15.0 | 35.2 | 0.212  |             | NaN    | 0.467  | 16.2 | 63.8     | 0.229 |  |

|   | Unnamed: 27 | Arena                     | Attend.  | Attend./G | Year |
|---|-------------|---------------------------|----------|-----------|------|
| 0 | NaN         | Boston Garden             | 596349.0 | 14664.0   | 1980 |
| 1 | NaN         | The Forum                 | 582882.0 | 17505.0   | 1980 |
| 2 | NaN         | King County Domed Stadium | NaN      | 28726.0   | 1980 |

```

3          NaN           The Spectrum          NaN          NaN  1980
4          NaN            MECCA Arena          NaN          NaN  1980

```

[10]: advanced\_df.tail()

|      | Unnamed: 0  | Rk       | Team                  | Age                        | W             | L           | PW     | PL     | \     |       |   |
|------|-------------|----------|-----------------------|----------------------------|---------------|-------------|--------|--------|-------|-------|---|
| 1201 | 26          | 27.0     | Houston Rockets       | 26.5                       | 17.0          | 55.0        | 20     | 52     |       |       |   |
| 1202 | 27          | 28.0     | Cleveland Cavaliers   | 24.0                       | 22.0          | 50.0        | 18     | 54     |       |       |   |
| 1203 | 28          | 29.0     | Orlando Magic         | 25.6                       | 21.0          | 51.0        | 17     | 55     |       |       |   |
| 1204 | 29          | 30.0     | Oklahoma City Thunder | 22.8                       | 22.0          | 50.0        | 15     | 57     |       |       |   |
| 1205 | 30          | NaN      | League Average        | 26.3                       | NaN           | NaN         | 36     | 36     |       |       |   |
|      | MOV         | SOS      | SRS                   | ORtg                       | DRtg          | NRtg        | Pace   | FTr    | 3PAr  | TS%   | \ |
| 1201 | -7.90       | 0.40     | -7.50                 | 107.1                      | 114.9         | -7.8        | 101.4  | 0.252  | 0.459 | 0.553 |   |
| 1202 | -8.44       | 0.25     | -8.19                 | 105.8                      | 114.4         | -8.6        | 97.3   | 0.261  | 0.347 | 0.543 |   |
| 1203 | -9.31       | 0.29     | -9.02                 | 105.1                      | 114.5         | -9.4        | 98.7   | 0.240  | 0.356 | 0.527 |   |
| 1204 | -10.64      | 0.51     | -10.13                | 103.5                      | 114.0         | -10.5       | 101.0  | 0.242  | 0.399 | 0.539 |   |
| 1205 | 0.00        | 0.00     | 0.00                  | 112.3                      | 112.3         | NaN         | 99.2   | 0.247  | 0.392 | 0.572 |   |
|      | Unnamed: 17 | eFG%     | TOV%                  | ORB%                       | FT/FGA        | Unnamed: 22 | eFG%.1 | TOV%.1 | \     |       |   |
| 1201 | NaN         | 0.521    | 13.0                  | 19.8                       | 0.187         | NaN         | 0.555  | 12.9   |       |       |   |
| 1202 | NaN         | 0.508    | 13.9                  | 23.6                       | 0.194         | NaN         | 0.556  | 13.0   |       |       |   |
| 1203 | NaN         | 0.490    | 11.5                  | 21.6                       | 0.186         | NaN         | 0.547  | 11.5   |       |       |   |
| 1204 | NaN         | 0.509    | 14.2                  | 21.2                       | 0.176         | NaN         | 0.547  | 11.5   |       |       |   |
| 1205 | NaN         | 0.538    | 12.4                  | 22.2                       | 0.192         | NaN         | 0.538  | 12.4   |       |       |   |
|      | DRB%        | FT/FGA.1 | Unnamed: 27           |                            | Arena         | Attend.     | \      |        |       |       |   |
| 1201 | 77.1        | 0.201    | NaN                   |                            | Toyota Center | 117009.0    |        |        |       |       |   |
| 1202 | 76.6        | 0.183    | NaN                   | Rocket Mortgage Fieldhouse |               | 91476.0     |        |        |       |       |   |
| 1203 | 78.2        | 0.169    | NaN                   |                            | Amway Center  | 126463.0    |        |        |       |       |   |
| 1204 | 77.9        | 0.167    | NaN                   | Chesapeake Energy Arena    |               | NaN         |        |        |       |       |   |
| 1205 | 77.8        | 0.192    | NaN                   |                            | NaN           | 49476.0     |        |        |       |       |   |
|      | Attend./G   | Year     |                       |                            |               |             |        |        |       |       |   |
| 1201 | 3250.0      | 2021     |                       |                            |               |             |        |        |       |       |   |
| 1202 | 2541.0      | 2021     |                       |                            |               |             |        |        |       |       |   |
| 1203 | 3513.0      | 2021     |                       |                            |               |             |        |        |       |       |   |
| 1204 | NaN         | 2021     |                       |                            |               |             |        |        |       |       |   |
| 1205 | 1374.0      | 2021     |                       |                            |               |             |        |        |       |       |   |

[11]: advanced\_df['Team'].value\_counts() # teams with '\*' indicates they've made it to the playoff

|                         |    |
|-------------------------|----|
| League Average          | 42 |
| San Antonio Spurs*      | 36 |
| Los Angeles Lakers*     | 34 |
| Portland Trail Blazers* | 34 |

|                        |    |
|------------------------|----|
| Boston Celtics*        | 32 |
| Houston Rockets*       | 30 |
| Utah Jazz*             | 30 |
| Golden State Warriors  | 29 |
| Atlanta Hawks*         | 27 |
| Indiana Pacers*        | 27 |
| Sacramento Kings       | 26 |
| Milwaukee Bucks*       | 26 |
| Phoenix Suns*          | 26 |
| Chicago Bulls*         | 26 |
| Philadelphia 76ers*    | 25 |
| Los Angeles Clippers   | 24 |
| Denver Nuggets*        | 24 |
| Detroit Pistons*       | 23 |
| Minnesota Timberwolves | 23 |
| Cleveland Cavaliers    | 23 |
| Dallas Mavericks*      | 23 |
| New York Knicks*       | 22 |
| Miami Heat*            | 22 |
| New York Knicks        | 20 |
| Cleveland Cavaliers*   | 19 |
| Detroit Pistons        | 19 |
| New Jersey Nets        | 18 |
| Denver Nuggets         | 18 |
| Dallas Mavericks       | 18 |
| Seattle SuperSonics*   | 18 |
| Philadelphia 76ers     | 17 |
| Phoenix Suns           | 16 |
| Milwaukee Bucks        | 16 |
| Orlando Magic          | 16 |
| Chicago Bulls          | 16 |
| Orlando Magic*         | 16 |
| New Jersey Nets*       | 15 |
| Indiana Pacers         | 15 |
| Atlanta Hawks          | 15 |
| Washington Wizards     | 15 |
| Toronto Raptors        | 14 |
| Los Angeles Clippers*  | 13 |
| Charlotte Hornets      | 13 |
| Golden State Warriors* | 13 |
| Toronto Raptors*       | 12 |
| Houston Rockets        | 12 |
| Utah Jazz              | 12 |
| Miami Heat             | 11 |
| Memphis Grizzlies*     | 11 |
| Seattle SuperSonics    | 11 |
| Sacramento Kings*      | 10 |

```
Boston Celtics          10
Oklahoma City Thunder* 10
Washington Bullets      10
Minnesota Timberwolves* 9
Washington Wizards*     9
Memphis Grizzlies       9
Los Angeles Lakers      8
Washington Bullets*     8
Charlotte Bobcats       8
Portland Trail Blazers   8
Charlotte Hornets*      8
Brooklyn Nets*          6
New Orleans Pelicans    6
San Antonio Spurs       6
Vancouver Grizzlies     6
New Orleans Hornets*    5
San Diego Clippers      5
New Orleans Hornets      4
Oklahoma City Thunder    3
Kansas City Kings        3
Kansas City Kings*       3
Brooklyn Nets            3
New Orleans/Oklahoma City Hornets 2
Charlotte Bobcats*      2
New Orleans Pelicans*    2
Name: Team, dtype: int64
```

```
[12]: advanced_df.Age.min(), advanced_df.Age.max()
```

```
[12]: (22.7, 32.0)
```

```
[13]: advanced_df.shape
```

```
[13]: (1206, 33)
```

```
[14]: advanced_df.duplicated().sum()
```

```
[14]: 0
```

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1206 entries, 0 to 1205
Data columns (total 33 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   Unnamed: 0        1206 non-null   int64
```

```

1   Rk          1164 non-null  float64
2   Team        1206 non-null  object
3   Age          1206 non-null  float64
4   W            1164 non-null  float64
5   L            1164 non-null  float64
6   PW           1206 non-null  int64
7   PL           1206 non-null  int64
8   MOV          1206 non-null  float64
9   SOS          1206 non-null  float64
10  SRS          1206 non-null  float64
11  ORtg         1206 non-null  float64
12  DRtg         1206 non-null  float64
13  NRtg         1164 non-null  float64
14  Pace          1206 non-null  float64
15  FTr           1206 non-null  float64
16  3PAr          1206 non-null  float64
17  TS%           1206 non-null  float64
18  Unnamed: 17    0 non-null   float64
19  eFG%          1206 non-null  float64
20  TOV%          1206 non-null  float64
21  ORB%          1206 non-null  float64
22  FT/FGA         1206 non-null  float64
23  Unnamed: 22    0 non-null   float64
24  eFG%.1         1206 non-null  float64
25  TOV%.1         1206 non-null  float64
26  DRB%          1206 non-null  float64
27  FT/FGA.1       1206 non-null  float64
28  Unnamed: 27    0 non-null   float64
29  Arena          1164 non-null  object
30  Attend.        1182 non-null  float64
31  Attend./G      766 non-null   float64
32  Year           1206 non-null  int64
dtypes: float64(27), int64(4), object(2)
memory usage: 311.0+ KB

```

**advanced\_df Observation** > In this dataset, there are a total of 1206 rows and 33 columns. There are a number of columns that we can drop from this table; some of them are filled with null values. More specifically the columns with “named” in their names. There are also a couple of columns that are misnamed specifically the one with the “.1” in their names. Furthermore, we also see that there are quite a number of NBA teams that don’t exist anymore or have changed their names. This is very much unlike the last dataset where all the names are up-to-date. One of the NBA teams’ names is “League Average” which is something we will have to deal with because it seems to be affecting the ‘Rk’ column by producing null values. Therefore, the best way to deal with the it would be to drop it; furthermore, it’s easy to see how rows with this value will affect the future exploration. I also think its a good idea to turn the proportion into percentages as well. Lastly, we will also need to deal with the asterick next to every NBA team that made the playoff in a given year. We will create a new column for that. Keep in mind that we will only create this column for this dataset and not all of them.

## 1.2.2 avg\_df

[16]: avg\_df.head()

|   | Unnamed: 0 | Rk    | Team                | G    | MP    | FG   | FGA  | FG%   | 3P   | \    |      |      |   |
|---|------------|-------|---------------------|------|-------|------|------|-------|------|------|------|------|---|
| 0 | 0          | 1.0   | San Antonio Spurs*  | 82   | 240.9 | 47.0 | 94.4 | 0.498 | 0.6  |      |      |      |   |
| 1 | 1          | 2.0   | Los Angeles Lakers* | 82   | 242.4 | 47.5 | 89.9 | 0.529 | 0.2  |      |      |      |   |
| 2 | 2          | 3.0   | Cleveland Cavaliers | 82   | 243.0 | 46.5 | 98.1 | 0.474 | 0.4  |      |      |      |   |
| 3 | 3          | 4.0   | New York Knicks     | 82   | 241.2 | 46.4 | 93.6 | 0.496 | 0.5  |      |      |      |   |
| 4 | 4          | 5.0   | Boston Celtics*     | 82   | 242.4 | 44.1 | 90.1 | 0.490 | 2.0  |      |      |      |   |
|   | 3PA        | 3P%   | 2P                  | 2PA  | 2P%   | FT   | FTA  | FT%   | ORB  | DRB  | TRB  | AST  | \ |
| 0 | 2.5        | 0.252 | 46.4                | 91.9 | 0.505 | 24.7 | 30.8 | 0.801 | 14.1 | 30.7 | 44.7 | 28.4 |   |
| 1 | 1.2        | 0.200 | 47.3                | 88.6 | 0.534 | 19.8 | 25.5 | 0.775 | 13.2 | 32.4 | 45.6 | 29.4 |   |
| 2 | 2.3        | 0.193 | 46.0                | 95.8 | 0.481 | 20.8 | 26.9 | 0.772 | 15.9 | 29.0 | 45.0 | 25.7 |   |
| 3 | 2.3        | 0.220 | 45.9                | 91.2 | 0.503 | 20.7 | 27.7 | 0.747 | 15.1 | 28.1 | 43.2 | 27.6 |   |
| 4 | 5.1        | 0.384 | 42.1                | 84.9 | 0.496 | 23.3 | 29.9 | 0.779 | 15.0 | 30.0 | 44.9 | 26.8 |   |
|   | STL        | BLK   | TOV                 | PF   | PTS   | Year |      |       |      |      |      |      |   |
| 0 | 9.4        | 4.1   | 19.4                | 25.6 | 119.4 | 1980 |      |       |      |      |      |      |   |
| 1 | 9.4        | 6.7   | 20.0                | 21.8 | 115.1 | 1980 |      |       |      |      |      |      |   |
| 2 | 9.3        | 4.2   | 16.7                | 23.6 | 114.1 | 1980 |      |       |      |      |      |      |   |
| 3 | 10.7       | 5.6   | 19.7                | 26.4 | 114.0 | 1980 |      |       |      |      |      |      |   |
| 4 | 9.9        | 3.8   | 18.8                | 24.1 | 113.5 | 1980 |      |       |      |      |      |      |   |

[17]: avg\_df.tail()

|      | Unnamed: 0 | Rk   | Team                  | G    | MP    | FG    | FGA   | FG%   | \     |      |      |   |
|------|------------|------|-----------------------|------|-------|-------|-------|-------|-------|------|------|---|
| 1201 | 26         | 27.0 | Detroit Pistons       | 72   | 242.1 | 38.7  | 85.6  | 0.452 |       |      |      |   |
| 1202 | 27         | 28.0 | Oklahoma City Thunder | 72   | 241.0 | 38.8  | 88.0  | 0.441 |       |      |      |   |
| 1203 | 28         | 29.0 | Orlando Magic         | 72   | 240.7 | 38.3  | 89.2  | 0.429 |       |      |      |   |
| 1204 | 29         | 30.0 | Cleveland Cavaliers   | 72   | 242.1 | 38.6  | 85.8  | 0.450 |       |      |      |   |
| 1205 | 30         | NaN  | League Average        | 72   | 241.4 | 41.2  | 88.4  | 0.466 |       |      |      |   |
|      | 3P         | 3PA  | 3P%                   | 2P   | 2PA   | 2P%   | FT    | FTA   | FT%   | ORB  | DRB  | \ |
| 1201 | 11.6       | 32.9 | 0.351                 | 27.1 | 52.7  | 0.515 | 17.8  | 23.4  | 0.759 | 9.6  | 33.1 |   |
| 1202 | 11.9       | 35.1 | 0.339                 | 26.9 | 52.9  | 0.509 | 15.5  | 21.3  | 0.725 | 9.9  | 35.7 |   |
| 1203 | 10.9       | 31.8 | 0.343                 | 27.4 | 57.4  | 0.476 | 16.6  | 21.4  | 0.775 | 10.4 | 35.1 |   |
| 1204 | 10.0       | 29.7 | 0.336                 | 28.6 | 56.0  | 0.510 | 16.7  | 22.4  | 0.743 | 10.4 | 32.3 |   |
| 1205 | 12.7       | 34.6 | 0.367                 | 28.5 | 53.8  | 0.530 | 17.0  | 21.8  | 0.778 | 9.8  | 34.5 |   |
|      | TRB        | AST  | STL                   | BLK  | TOV   | PF    | PTS   | Year  |       |      |      |   |
| 1201 | 42.7       | 24.2 | 7.4                   | 5.2  | 14.9  | 20.5  | 106.6 | 2021  |       |      |      |   |
| 1202 | 45.6       | 22.1 | 7.0                   | 4.4  | 16.1  | 18.1  | 105.0 | 2021  |       |      |      |   |
| 1203 | 45.4       | 21.8 | 6.9                   | 4.4  | 12.8  | 17.2  | 104.0 | 2021  |       |      |      |   |
| 1204 | 42.8       | 23.8 | 7.8                   | 4.5  | 15.5  | 18.2  | 103.8 | 2021  |       |      |      |   |
| 1205 | 44.3       | 24.8 | 7.6                   | 4.9  | 13.8  | 19.3  | 112.1 | 2021  |       |      |      |   |

```
[18]: avg_df.shape
```

```
[18]: (1206, 27)
```

```
[19]: advanced_df['Team'].value_counts()
```

|                         |    |
|-------------------------|----|
| [19]: League Average    | 42 |
| San Antonio Spurs*      | 36 |
| Los Angeles Lakers*     | 34 |
| Portland Trail Blazers* | 34 |
| Boston Celtics*         | 32 |
| Houston Rockets*        | 30 |
| Utah Jazz*              | 30 |
| Golden State Warriors   | 29 |
| Atlanta Hawks*          | 27 |
| Indiana Pacers*         | 27 |
| Sacramento Kings        | 26 |
| Milwaukee Bucks*        | 26 |
| Phoenix Suns*           | 26 |
| Chicago Bulls*          | 26 |
| Philadelphia 76ers*     | 25 |
| Los Angeles Clippers    | 24 |
| Denver Nuggets*         | 24 |
| Detroit Pistons*        | 23 |
| Minnesota Timberwolves  | 23 |
| Cleveland Cavaliers     | 23 |
| Dallas Mavericks*       | 23 |
| New York Knicks*        | 22 |
| Miami Heat*             | 22 |
| New York Knicks         | 20 |
| Cleveland Cavaliers*    | 19 |
| Detroit Pistons         | 19 |
| New Jersey Nets         | 18 |
| Denver Nuggets          | 18 |
| Dallas Mavericks        | 18 |
| Seattle SuperSonics*    | 18 |
| Philadelphia 76ers      | 17 |
| Phoenix Suns            | 16 |
| Milwaukee Bucks         | 16 |
| Orlando Magic           | 16 |
| Chicago Bulls           | 16 |
| Orlando Magic*          | 16 |
| New Jersey Nets*        | 15 |
| Indiana Pacers          | 15 |
| Atlanta Hawks           | 15 |
| Washington Wizards      | 15 |
| Toronto Raptors         | 14 |

|                                   |    |
|-----------------------------------|----|
| Los Angeles Clippers*             | 13 |
| Charlotte Hornets                 | 13 |
| Golden State Warriors*            | 13 |
| Toronto Raptors*                  | 12 |
| Houston Rockets                   | 12 |
| Utah Jazz                         | 12 |
| Miami Heat                        | 11 |
| Memphis Grizzlies*                | 11 |
| Seattle SuperSonics               | 11 |
| Sacramento Kings*                 | 10 |
| Boston Celtics                    | 10 |
| Oklahoma City Thunder*            | 10 |
| Washington Bullets                | 10 |
| Minnesota Timberwolves*           | 9  |
| Washington Wizards*               | 9  |
| Memphis Grizzlies                 | 9  |
| Los Angeles Lakers                | 8  |
| Washington Bullets*               | 8  |
| Charlotte Bobcats                 | 8  |
| Portland Trail Blazers            | 8  |
| Charlotte Hornets*                | 8  |
| Brooklyn Nets*                    | 6  |
| New Orleans Pelicans              | 6  |
| San Antonio Spurs                 | 6  |
| Vancouver Grizzlies               | 6  |
| New Orleans Hornets*              | 5  |
| San Diego Clippers                | 5  |
| New Orleans Hornets               | 4  |
| Oklahoma City Thunder             | 3  |
| Kansas City Kings                 | 3  |
| Kansas City Kings*                | 3  |
| Brooklyn Nets                     | 3  |
| New Orleans/Oklahoma City Hornets | 2  |
| Charlotte Bobcats*                | 2  |
| New Orleans Pelicans*             | 2  |
| Name: Team, dtype: int64          |    |

[20]: avg\_df.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1206 entries, 0 to 1205
Data columns (total 27 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   Unnamed: 0    1206 non-null   int64  
 1   Rk           1164 non-null   float64 
 2   Team          1206 non-null   object 

```

```

3   G          1206 non-null  int64
4   MP         1206 non-null  float64
5   FG         1206 non-null  float64
6   FGA        1206 non-null  float64
7   FG%        1206 non-null  float64
8   3P         1206 non-null  float64
9   3PA        1206 non-null  float64
10  3P%        1206 non-null  float64
11  2P         1206 non-null  float64
12  2PA        1206 non-null  float64
13  2P%        1206 non-null  float64
14  FT         1206 non-null  float64
15  FTA        1206 non-null  float64
16  FT%        1206 non-null  float64
17  ORB        1206 non-null  float64
18  DRB        1206 non-null  float64
19  TRB        1206 non-null  float64
20  AST        1206 non-null  float64
21  STL        1206 non-null  float64
22  BLK        1206 non-null  float64
23  TOV        1206 non-null  float64
24  PF          1206 non-null  float64
25  PTS        1206 non-null  float64
26  Year       1206 non-null  int64
dtypes: float64(23), int64(3), object(1)
memory usage: 254.5+ KB

```

**avg\_df Observation** > The avg\_df dataset is a little bit cleaner. There are a total of 1206 rows and 27 columns. There are barely any null values compared to the last one that we examined. However, we also see some similar errors such as the names of the NBA teams. Luckily, the names of the NBA teams in this dataset are the same as the advanced\_df dataset, so we can create a function to deal with this particular problem. **### total\_df**

[21]: `total_df.head()`

|   | Unnamed: 0 | Rk    | Team                | G    | MP    | FG   | FGA  | FG%   | 3P   | \    |      |      |   |
|---|------------|-------|---------------------|------|-------|------|------|-------|------|------|------|------|---|
| 0 | 0          | 1.0   | San Antonio Spurs*  | 82   | 19755 | 3856 | 7738 | 0.498 | 52   |      |      |      |   |
| 1 | 1          | 2.0   | Los Angeles Lakers* | 82   | 19880 | 3898 | 7368 | 0.529 | 20   |      |      |      |   |
| 2 | 2          | 3.0   | Cleveland Cavaliers | 82   | 19930 | 3811 | 8041 | 0.474 | 36   |      |      |      |   |
| 3 | 3          | 4.0   | New York Knicks     | 82   | 19780 | 3802 | 7672 | 0.496 | 42   |      |      |      |   |
| 4 | 4          | 5.0   | Boston Celtics*     | 82   | 19880 | 3617 | 7387 | 0.490 | 162  |      |      |      |   |
|   | 3PA        | 3P%   | 2P                  | 2PA  | 2P%   | FT   | FTA  | FT%   | ORB  | DRB  | TRB  | AST  | \ |
| 0 | 206        | 0.252 | 3804                | 7532 | 0.505 | 2024 | 2528 | 0.801 | 1153 | 2515 | 3668 | 2326 |   |
| 1 | 100        | 0.200 | 3878                | 7268 | 0.534 | 1622 | 2092 | 0.775 | 1085 | 2653 | 3738 | 2413 |   |
| 2 | 187        | 0.193 | 3775                | 7854 | 0.481 | 1702 | 2205 | 0.772 | 1307 | 2381 | 3688 | 2108 |   |
| 3 | 191        | 0.220 | 3760                | 7481 | 0.503 | 1698 | 2274 | 0.747 | 1236 | 2303 | 3539 | 2265 |   |
| 4 | 422        | 0.384 | 3455                | 6965 | 0.496 | 1907 | 2449 | 0.779 | 1227 | 2457 | 3684 | 2198 |   |

|   | STL | BLK | TOV  | PF   | PTS  | Year |
|---|-----|-----|------|------|------|------|
| 0 | 771 | 333 | 1589 | 2103 | 9788 | 1980 |
| 1 | 774 | 546 | 1639 | 1784 | 9438 | 1980 |
| 2 | 764 | 342 | 1370 | 1934 | 9360 | 1980 |
| 3 | 881 | 457 | 1613 | 2168 | 9344 | 1980 |
| 4 | 809 | 308 | 1539 | 1974 | 9303 | 1980 |

[22]: total\_df.shape

[22]: (1206, 27)

[23]: total\_df.Team.value\_counts()

|                         |    |
|-------------------------|----|
| [23]: League Average    | 42 |
| San Antonio Spurs*      | 36 |
| Portland Trail Blazers* | 34 |
| Los Angeles Lakers*     | 34 |
| Boston Celtics*         | 32 |
| Utah Jazz*              | 30 |
| Houston Rockets*        | 30 |
| Golden State Warriors   | 29 |
| Indiana Pacers*         | 27 |
| Atlanta Hawks*          | 27 |
| Milwaukee Bucks*        | 26 |
| Phoenix Suns*           | 26 |
| Chicago Bulls*          | 26 |
| Sacramento Kings        | 26 |
| Philadelphia 76ers*     | 25 |
| Denver Nuggets*         | 24 |
| Los Angeles Clippers    | 24 |
| Detroit Pistons*        | 23 |
| Minnesota Timberwolves  | 23 |
| Dallas Mavericks*       | 23 |
| Cleveland Cavaliers     | 23 |
| Miami Heat*             | 22 |
| New York Knicks*        | 22 |
| New York Knicks         | 20 |
| Cleveland Cavaliers*    | 19 |
| Detroit Pistons         | 19 |
| New Jersey Nets         | 18 |
| Dallas Mavericks        | 18 |
| Denver Nuggets          | 18 |
| Seattle SuperSonics*    | 18 |
| Philadelphia 76ers      | 17 |
| Orlando Magic*          | 16 |
| Milwaukee Bucks         | 16 |

|                                   |    |
|-----------------------------------|----|
| Phoenix Suns                      | 16 |
| Orlando Magic                     | 16 |
| Chicago Bulls                     | 16 |
| New Jersey Nets*                  | 15 |
| Washington Wizards                | 15 |
| Indiana Pacers                    | 15 |
| Atlanta Hawks                     | 15 |
| Toronto Raptors                   | 14 |
| Golden State Warriors*            | 13 |
| Los Angeles Clippers*             | 13 |
| Charlotte Hornets                 | 13 |
| Houston Rockets                   | 12 |
| Utah Jazz                         | 12 |
| Toronto Raptors*                  | 12 |
| Seattle SuperSonics               | 11 |
| Miami Heat                        | 11 |
| Memphis Grizzlies*                | 11 |
| Sacramento Kings*                 | 10 |
| Washington Bullets                | 10 |
| Oklahoma City Thunder*            | 10 |
| Boston Celtics                    | 10 |
| Washington Wizards*               | 9  |
| Minnesota Timberwolves*           | 9  |
| Memphis Grizzlies                 | 9  |
| Charlotte Bobcats                 | 8  |
| Washington Bullets*               | 8  |
| Charlotte Hornets*                | 8  |
| Portland Trail Blazers            | 8  |
| Los Angeles Lakers                | 8  |
| Vancouver Grizzlies               | 6  |
| San Antonio Spurs                 | 6  |
| New Orleans Pelicans              | 6  |
| Brooklyn Nets*                    | 6  |
| San Diego Clippers                | 5  |
| New Orleans Hornets*              | 5  |
| New Orleans Hornets               | 4  |
| Oklahoma City Thunder             | 3  |
| Kansas City Kings*                | 3  |
| Kansas City Kings                 | 3  |
| Brooklyn Nets                     | 3  |
| New Orleans/Oklahoma City Hornets | 2  |
| Charlotte Bobcats*                | 2  |
| New Orleans Pelicans*             | 2  |
| Name: Team, dtype: int64          |    |

[24]: total\_df.info()

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1206 entries, 0 to 1205
Data columns (total 27 columns):
 #   Column      Non-Null Count Dtype  
--- 
 0   Unnamed: 0    1206 non-null   int64  
 1   Rk          1164 non-null   float64 
 2   Team         1206 non-null   object  
 3   G            1206 non-null   int64  
 4   MP           1206 non-null   int64  
 5   FG           1206 non-null   int64  
 6   FGA          1206 non-null   int64  
 7   FG%          1206 non-null   float64 
 8   3P           1206 non-null   int64  
 9   3PA          1206 non-null   int64  
 10  3P%          1206 non-null   float64 
 11  2P           1206 non-null   int64  
 12  2PA          1206 non-null   int64  
 13  2P%          1206 non-null   float64 
 14  FT           1206 non-null   int64  
 15  FTA          1206 non-null   int64  
 16  FT%          1206 non-null   float64 
 17  ORB          1206 non-null   int64  
 18  DRB          1206 non-null   int64  
 19  TRB          1206 non-null   int64  
 20  AST          1206 non-null   int64  
 21  STL          1206 non-null   int64  
 22  BLK          1206 non-null   int64  
 23  TOV          1206 non-null   int64  
 24  PF            1206 non-null   int64  
 25  PTS          1206 non-null   int64  
 26  Year         1206 non-null   int64  
dtypes: float64(5), int64(21), object(1)
memory usage: 254.5+ KB

```

**total\_df Observation** > In the last dataset, we can see that we have many of the same problems from the last two datasets. The most consistent problem that we have to deal with is the namings of the teams. We do not have to deal with that many null values, but there are some columns that we will need to drop because they are not pivotal to our exploration.

### 1.3 Creating Function

```
[25]: # function used to clean the Team column and create new column tracking playoff
      ↵contentions
def clean_team_col(df):
    """
    Input:
    df - (pandas dataframe) dataframe that we are interested in cleaning

```

```

Output:
df - (pandas dataframe) a dataframe where the Team column where the
following has been done:
1. Removed '*' from team names.
2. Deleted rows where Team is 'League Average'.
3. Replaced the team name with current/updated team names.
'''

# create new column for teams who made into the play off that year which is
indicated with a '*'
df['Team'] = df['Team'].apply(lambda x: x.replace('*', ''))

# drop rows with 'League Average' values
df.drop(df.index[df.Team == 'League Average'], inplace = True)

# updating teams' names to present day names; this step requires some extra
Google searches
df.replace({'Team': {'Seattle SuperSonics': 'Oklahoma City Thunder',
                     'Washington Bullets': 'Washington Wizards', 'Charlotte
                     Bobcats': 'Charlotte Hornets',
                     'New Orleans Hornets': 'New Orleans Pelicans', 'Kansas
                     City Kings': 'Sacramento Kings',
                     'Vancouver Grizzlies': 'Memphis Grizzlies', 'San Diego
                     Clippers': 'Los Angeles Clippers',
                     'New Orleans/Oklahoma City Hornets': 'New Orleans
                     Pelicans',
                     'New Jersey Nets': 'Brooklyn Nets'}},
            inplace = True)

return df

```

## 1.4 Cleaning Data

For the cleaning process, I will be using the define-clean-check framework. You might see the clean and check process in the same cell. This has been done to save space.

### advanced\_df 1. Create new 'Playoff' column.

```
[26]: # clean
advanced_df['Playoff'] = [True if '*' in x else False for x in
                         advanced_df['Team']]
#check
advanced_df.head(10)
```

|   | Unnamed: 0 | Rk  | Team                 | Age  | W    | L    | PW | PL | MOV  | \ |
|---|------------|-----|----------------------|------|------|------|----|----|------|---|
| 0 | 0          | 1.0 | Boston Celtics*      | 27.3 | 61.0 | 21.0 | 60 | 22 | 7.79 |   |
| 1 | 1          | 2.0 | Los Angeles Lakers*  | 26.2 | 60.0 | 22.0 | 55 | 27 | 5.90 |   |
| 2 | 2          | 3.0 | Seattle SuperSonics* | 27.0 | 56.0 | 26.0 | 53 | 29 | 4.66 |   |

|   |   |      |                     |      |      |      |    |    |      |
|---|---|------|---------------------|------|------|------|----|----|------|
| 3 | 3 | 4.0  | Philadelphia 76ers* | 27.0 | 59.0 | 23.0 | 52 | 30 | 4.22 |
| 4 | 4 | 5.0  | Milwaukee Bucks*    | 25.3 | 49.0 | 33.0 | 51 | 31 | 3.94 |
| 5 | 5 | 6.0  | Phoenix Suns*       | 26.5 | 55.0 | 27.0 | 50 | 32 | 3.60 |
| 6 | 6 | 7.0  | Kansas City Kings*  | 25.5 | 47.0 | 35.0 | 49 | 33 | 3.13 |
| 7 | 7 | 8.0  | Atlanta Hawks*      | 26.1 | 50.0 | 32.0 | 49 | 33 | 2.91 |
| 8 | 8 | 9.0  | Cleveland Cavaliers | 27.4 | 37.0 | 45.0 | 42 | 40 | 0.34 |
| 9 | 9 | 10.0 | Houston Rockets*    | 27.3 | 41.0 | 41.0 | 41 | 41 | 0.17 |

|   | SOS   | SRS  | ORtg  | DRtg  | NRtg | Pace  | FTr   | 3PAr  | TS%   | Unnamed: 17 | \ |
|---|-------|------|-------|-------|------|-------|-------|-------|-------|-------------|---|
| 0 | -0.42 | 7.37 | 109.4 | 101.9 | 7.5  | 102.6 | 0.332 | 0.057 | 0.550 | NaN         |   |
| 1 | -0.51 | 5.40 | 109.5 | 103.9 | 5.6  | 104.1 | 0.284 | 0.014 | 0.569 | NaN         |   |
| 2 | -0.42 | 4.24 | 105.8 | 101.2 | 4.6  | 101.8 | 0.298 | 0.025 | 0.520 | NaN         |   |
| 3 | -0.18 | 4.04 | 105.0 | 101.0 | 4.0  | 103.0 | 0.340 | 0.017 | 0.544 | NaN         |   |
| 4 | -0.37 | 3.57 | 106.8 | 102.9 | 3.9  | 102.4 | 0.278 | 0.021 | 0.532 | NaN         |   |
| 5 | -0.35 | 3.25 | 105.6 | 102.2 | 3.4  | 104.8 | 0.341 | 0.039 | 0.548 | NaN         |   |
| 6 | -0.32 | 2.82 | 104.0 | 101.0 | 3.0  | 103.2 | 0.300 | 0.015 | 0.522 | NaN         |   |
| 7 | -0.09 | 2.83 | 105.2 | 102.3 | 2.9  | 98.9  | 0.376 | 0.011 | 0.523 | NaN         |   |
| 8 | 0.09  | 0.43 | 106.7 | 106.4 | 0.3  | 105.6 | 0.274 | 0.023 | 0.519 | NaN         |   |
| 9 | 0.10  | 0.27 | 108.1 | 108.0 | 0.1  | 101.2 | 0.310 | 0.051 | 0.533 | NaN         |   |

|   | eFG%  | TOV% | ORB% | FT/FGA | Unnamed: 22 | eFG%.1 | TOV%.1 | DRB% | FT/FGA.1 | \ |
|---|-------|------|------|--------|-------------|--------|--------|------|----------|---|
| 0 | 0.501 | 15.4 | 34.8 | 0.258  | NaN         | 0.475  | 16.5   | 67.8 | 0.234    |   |
| 1 | 0.530 | 16.5 | 32.6 | 0.220  | NaN         | 0.475  | 14.0   | 66.9 | 0.181    |   |
| 2 | 0.474 | 14.9 | 36.4 | 0.229  | NaN         | 0.463  | 15.4   | 67.9 | 0.221    |   |
| 3 | 0.494 | 17.2 | 33.5 | 0.262  | NaN         | 0.460  | 15.5   | 66.7 | 0.217    |   |
| 4 | 0.491 | 15.0 | 35.2 | 0.212  | NaN         | 0.467  | 16.2   | 63.8 | 0.229    |   |
| 5 | 0.498 | 16.4 | 30.4 | 0.263  | NaN         | 0.483  | 16.5   | 66.9 | 0.213    |   |
| 6 | 0.480 | 14.5 | 31.0 | 0.223  | NaN         | 0.479  | 17.9   | 68.1 | 0.273    |   |
| 7 | 0.465 | 15.4 | 36.9 | 0.290  | NaN         | 0.461  | 17.1   | 65.6 | 0.291    |   |
| 8 | 0.476 | 13.2 | 33.1 | 0.212  | NaN         | 0.505  | 16.3   | 65.9 | 0.216    |   |
| 9 | 0.487 | 15.5 | 37.6 | 0.238  | NaN         | 0.499  | 16.1   | 63.2 | 0.230    |   |

|   | Unnamed: 27 | Arena                              | Attend.  | Attend./G | Year | \ |
|---|-------------|------------------------------------|----------|-----------|------|---|
| 0 | NaN         | Boston Garden                      | 596349.0 | 14664.0   | 1980 |   |
| 1 | NaN         | The Forum                          | 582882.0 | 17505.0   | 1980 |   |
| 2 | NaN         | King County Domed Stadium          | NaN      | 28726.0   | 1980 |   |
| 3 | NaN         | The Spectrum                       | NaN      | NaN       | 1980 |   |
| 4 | NaN         | MECCA Arena                        | NaN      | NaN       | 1980 |   |
| 5 | NaN         | Arizona Veterans Memorial Coliseum | NaN      | NaN       | 1980 |   |
| 6 | NaN         | Kemper Arena                       | NaN      | NaN       | 1980 |   |
| 7 | NaN         | Omni Coliseum                      | NaN      | NaN       | 1980 |   |
| 8 | NaN         | Coliseum at Richfield              | NaN      | 15443.0   | 1980 |   |
| 9 | NaN         | The Summit                         | NaN      | 13656.0   | 1980 |   |

|   | Playoff |
|---|---------|
| 0 | True    |
| 1 | True    |

```
2      True
3      True
4      True
5      True
6      True
7      True
8    False
9      True
```

## 2. Clean Team column with clean\_team\_col function.

```
[27]: clean_team_col(advanced_df)
advanced_df.head()
```

```
[27]:   Unnamed: 0   Rk          Team   Age     W     L   PW   PL   MOV \
0           0  1.0  Boston Celtics  27.3  61.0  21.0  60  22  7.79
1           1  2.0  Los Angeles Lakers  26.2  60.0  22.0  55  27  5.90
2           2  3.0 Oklahoma City Thunder  27.0  56.0  26.0  53  29  4.66
3           3  4.0 Philadelphia 76ers  27.0  59.0  23.0  52  30  4.22
4           4  5.0 Milwaukee Bucks  25.3  49.0  33.0  51  31  3.94

      SOS   SRS   ORtg   DRtg   NRtg   Pace    FTr    3PAr    TS%  Unnamed: 17 \
0 -0.42  7.37  109.4  101.9    7.5  102.6  0.332  0.057  0.550        NaN
1 -0.51  5.40  109.5  103.9    5.6  104.1  0.284  0.014  0.569        NaN
2 -0.42  4.24  105.8  101.2    4.6  101.8  0.298  0.025  0.520        NaN
3 -0.18  4.04  105.0  101.0    4.0  103.0  0.340  0.017  0.544        NaN
4 -0.37  3.57  106.8  102.9    3.9  102.4  0.278  0.021  0.532        NaN

      eFG%  TOV%  ORB%  FT/FGA  Unnamed: 22  eFG%.1  TOV%.1  DRB%  FT/FGA.1 \
0  0.501  15.4  34.8  0.258        NaN  0.475  16.5  67.8  0.234
1  0.530  16.5  32.6  0.220        NaN  0.475  14.0  66.9  0.181
2  0.474  14.9  36.4  0.229        NaN  0.463  15.4  67.9  0.221
3  0.494  17.2  33.5  0.262        NaN  0.460  15.5  66.7  0.217
4  0.491  15.0  35.2  0.212        NaN  0.467  16.2  63.8  0.229

      Unnamed: 27          Arena  Attend.  Attend./G  Year  Playoff
0           NaN  Boston Garden  596349.0  14664.0  1980    True
1           NaN  The Forum  582882.0  17505.0  1980    True
2           NaN King County Domed Stadium  NaN  28726.0  1980    True
3           NaN  The Spectrum  NaN        NaN  1980    True
4           NaN       MECCA Arena  NaN        NaN  1980    True
```

```
[28]: advanced_df[advanced_df.Team == 'League Average']
```

```
[28]: Empty DataFrame
Columns: [Unnamed: 0, Rk, Team, Age, W, L, PW, PL, MOV, SOS, SRS, ORtg, DRtg, NRtg, Pace, FTr, 3PAr, TS%, Unnamed: 17, eFG%, TOV%, ORB%, FT/FGA, Unnamed: 22,
```

```
eFG%.1, TOV%.1, DRB%, FT/FGA.1, Unnamed: 27, Arena, Attend., Attend./G, Year,  
Playoff]  
Index: []
```

```
[29]: len(advanced_df.Team.value_counts()) # there should be 30 teams
```

```
[29]: 30
```

### 3. Drop all unneeded columns.

```
[30]: advanced_df = advanced_df[['Team', 'Age', 'W', 'L', 'MOV', 'ORtg', 'DRtg',  
    ↵'Pace',  
    ↵'3PAr', 'eFG%', 'TS%', 'TOV%', 'ORB%', 'Year',  
    ↵'Playoff']]  
advanced_df.head()
```

```
[30]:
```

|   | Team                  | Age  | W    | L    | MOV  | ORtg  | DRtg  | Pace  | 3PAr  | \ |
|---|-----------------------|------|------|------|------|-------|-------|-------|-------|---|
| 0 | Boston Celtics        | 27.3 | 61.0 | 21.0 | 7.79 | 109.4 | 101.9 | 102.6 | 0.057 |   |
| 1 | Los Angeles Lakers    | 26.2 | 60.0 | 22.0 | 5.90 | 109.5 | 103.9 | 104.1 | 0.014 |   |
| 2 | Oklahoma City Thunder | 27.0 | 56.0 | 26.0 | 4.66 | 105.8 | 101.2 | 101.8 | 0.025 |   |
| 3 | Philadelphia 76ers    | 27.0 | 59.0 | 23.0 | 4.22 | 105.0 | 101.0 | 103.0 | 0.017 |   |
| 4 | Milwaukee Bucks       | 25.3 | 49.0 | 33.0 | 3.94 | 106.8 | 102.9 | 102.4 | 0.021 |   |

|   | eFG%  | TS%   | TOV% | ORB% | Year | Playoff |
|---|-------|-------|------|------|------|---------|
| 0 | 0.501 | 0.550 | 15.4 | 34.8 | 1980 | True    |
| 1 | 0.530 | 0.569 | 16.5 | 32.6 | 1980 | True    |
| 2 | 0.474 | 0.520 | 14.9 | 36.4 | 1980 | True    |
| 3 | 0.494 | 0.544 | 17.2 | 33.5 | 1980 | True    |
| 4 | 0.491 | 0.532 | 15.0 | 35.2 | 1980 | True    |

### 4. Multiply ‘eFG%’ and ‘3PAr’ by 100 for better readability.

```
[31]: advanced_df['eFG%'] = advanced_df['eFG%'].apply(lambda x: x * 100)  
advanced_df['3PAr'] = advanced_df['3PAr'].apply(lambda x: x * 100)  
advanced_df['TS%'] = advanced_df['TS%'].apply(lambda x: x * 100)
```

```
[32]: advanced_df.head()
```

```
[32]:
```

|   | Team                  | Age  | W    | L    | MOV  | ORtg  | DRtg  | Pace  | 3PAr | \ |
|---|-----------------------|------|------|------|------|-------|-------|-------|------|---|
| 0 | Boston Celtics        | 27.3 | 61.0 | 21.0 | 7.79 | 109.4 | 101.9 | 102.6 | 5.7  |   |
| 1 | Los Angeles Lakers    | 26.2 | 60.0 | 22.0 | 5.90 | 109.5 | 103.9 | 104.1 | 1.4  |   |
| 2 | Oklahoma City Thunder | 27.0 | 56.0 | 26.0 | 4.66 | 105.8 | 101.2 | 101.8 | 2.5  |   |
| 3 | Philadelphia 76ers    | 27.0 | 59.0 | 23.0 | 4.22 | 105.0 | 101.0 | 103.0 | 1.7  |   |
| 4 | Milwaukee Bucks       | 25.3 | 49.0 | 33.0 | 3.94 | 106.8 | 102.9 | 102.4 | 2.1  |   |

|   | eFG% | TS%  | TOV% | ORB% | Year | Playoff |
|---|------|------|------|------|------|---------|
| 0 | 50.1 | 55.0 | 15.4 | 34.8 | 1980 | True    |
| 1 | 53.0 | 56.9 | 16.5 | 32.6 | 1980 | True    |

```

2 47.4 52.0 14.9 36.4 1980      True
3 49.4 54.4 17.2 33.5 1980      True
4 49.1 53.2 15.0 35.2 1980      True

```

## 5. Convert 'W' and 'L' column to integer datatype.

```
[33]: advanced_df['W'] = advanced_df['W'].astype(int)
advanced_df['L'] = advanced_df['L'].astype(int)
np.dtype(advanced_df['W']), np.dtype(advanced_df['L'])
```

```
[33]: (dtype('int64'), dtype('int64'))
```

## 6. Rename 'W' column to 'Wins' for context.

```
[34]: advanced_df.rename(columns={'W': 'Wins', 'L': 'Losses'}, inplace=True)
advanced_df.head(1)
```

```
[34]:          Team    Age   Wins  Losses    MOV    ORtg    DRtg    Pace   3PAr   eFG% \
0  Boston Celtics  27.3     61      21  7.79  109.4  101.9  102.6  5.7  50.1
               TS%  TOV%  ORB%  Year Playoff
0   55.0  15.4  34.8  1980     True
```

### 1.4.1 total\_df

#### 1. Drop unneeded columns.

```
[35]: total_df.head()
```

```
[35]:          Unnamed: 0   Rk          Team    G    MP    FG    FGA    FG%    3P \
0            0  1.0  San Antonio Spurs*  82  19755  3856  7738  0.498  52
1            1  2.0  Los Angeles Lakers*  82  19880  3898  7368  0.529  20
2            2  3.0  Cleveland Cavaliers  82  19930  3811  8041  0.474  36
3            3  4.0  New York Knicks   82  19780  3802  7672  0.496  42
4            4  5.0  Boston Celtics*   82  19880  3617  7387  0.490  162
               3PA    3P%    2P    2PA    2P%    FT    FTA    FT%    ORB    DRB    TRB    AST \
0   206  0.252  3804  7532  0.505  2024  2528  0.801  1153  2515  3668  2326
1   100  0.200  3878  7268  0.534  1622  2092  0.775  1085  2653  3738  2413
2   187  0.193  3775  7854  0.481  1702  2205  0.772  1307  2381  3688  2108
3   191  0.220  3760  7481  0.503  1698  2274  0.747  1236  2303  3539  2265
4   422  0.384  3455  6965  0.496  1907  2449  0.779  1227  2457  3684  2198
               STL    BLK    TOV    PF    PTS  Year
0   771  333  1589  2103  9788  1980
1   774  546  1639  1784  9438  1980
2   764  342  1370  1934  9360  1980
3   881  457  1613  2168  9344  1980
```

```
4 809 308 1539 1974 9303 1980
```

```
[36]: total_df = total_df[['Team', 'FG', 'FGA', 'FG%', '2P', '2PA',  
                         '2P%', '3P', '3PA', '3P%', 'FT', 'FTA', 'ORB', 'TRB',  
                         'AST', 'TOV', 'PTS', 'Year']]
```

```
[37]: total_df.head()
```

```
[37]:
```

|   | Team                | FG   | FGA  | FG%   | 2P   | 2PA  | 2P%   | 3P  | 3PA | 3P%   | \ |
|---|---------------------|------|------|-------|------|------|-------|-----|-----|-------|---|
| 0 | San Antonio Spurs*  | 3856 | 7738 | 0.498 | 3804 | 7532 | 0.505 | 52  | 206 | 0.252 |   |
| 1 | Los Angeles Lakers* | 3898 | 7368 | 0.529 | 3878 | 7268 | 0.534 | 20  | 100 | 0.200 |   |
| 2 | Cleveland Cavaliers | 3811 | 8041 | 0.474 | 3775 | 7854 | 0.481 | 36  | 187 | 0.193 |   |
| 3 | New York Knicks     | 3802 | 7672 | 0.496 | 3760 | 7481 | 0.503 | 42  | 191 | 0.220 |   |
| 4 | Boston Celtics*     | 3617 | 7387 | 0.490 | 3455 | 6965 | 0.496 | 162 | 422 | 0.384 |   |

|   | FT   | FTA  | ORB  | TRB  | AST  | TOV  | PTS  | Year |
|---|------|------|------|------|------|------|------|------|
| 0 | 2024 | 2528 | 1153 | 3668 | 2326 | 1589 | 9788 | 1980 |
| 1 | 1622 | 2092 | 1085 | 3738 | 2413 | 1639 | 9438 | 1980 |
| 2 | 1702 | 2205 | 1307 | 3688 | 2108 | 1370 | 9360 | 1980 |
| 3 | 1698 | 2274 | 1236 | 3539 | 2265 | 1613 | 9344 | 1980 |
| 4 | 1907 | 2449 | 1227 | 3684 | 2198 | 1539 | 9303 | 1980 |

## 2. Multiply ‘FG%’, ‘2P%’, and ‘3P%’ by 100.

```
[38]: total_df['3P%'] = total_df['3P%'].apply(lambda x: x * 100)  
total_df['2P%'] = total_df['2P%'].apply(lambda x: x * 100)  
total_df['FG%'] = total_df['FG%'].apply(lambda x: x * 100)  
  
total_df.head()
```

```
[38]:
```

|   | Team                | FG   | FGA  | FG%  | 2P   | 2PA  | 2P%  | 3P  | 3PA | 3P%  | \ |
|---|---------------------|------|------|------|------|------|------|-----|-----|------|---|
| 0 | San Antonio Spurs*  | 3856 | 7738 | 49.8 | 3804 | 7532 | 50.5 | 52  | 206 | 25.2 |   |
| 1 | Los Angeles Lakers* | 3898 | 7368 | 52.9 | 3878 | 7268 | 53.4 | 20  | 100 | 20.0 |   |
| 2 | Cleveland Cavaliers | 3811 | 8041 | 47.4 | 3775 | 7854 | 48.1 | 36  | 187 | 19.3 |   |
| 3 | New York Knicks     | 3802 | 7672 | 49.6 | 3760 | 7481 | 50.3 | 42  | 191 | 22.0 |   |
| 4 | Boston Celtics*     | 3617 | 7387 | 49.0 | 3455 | 6965 | 49.6 | 162 | 422 | 38.4 |   |

|   | FT   | FTA  | ORB  | TRB  | AST  | TOV  | PTS  | Year |
|---|------|------|------|------|------|------|------|------|
| 0 | 2024 | 2528 | 1153 | 3668 | 2326 | 1589 | 9788 | 1980 |
| 1 | 1622 | 2092 | 1085 | 3738 | 2413 | 1639 | 9438 | 1980 |
| 2 | 1702 | 2205 | 1307 | 3688 | 2108 | 1370 | 9360 | 1980 |
| 3 | 1698 | 2274 | 1236 | 3539 | 2265 | 1613 | 9344 | 1980 |
| 4 | 1907 | 2449 | 1227 | 3684 | 2198 | 1539 | 9303 | 1980 |

## 3. Create new ‘Playoff’ column to keep track.

```
[39]: # clean
total_df['Playoff'] = [True if '*' in x else False for x in total_df['Team']]
#check
total_df.head(10)
```

|   | Team                | FG   | FGA  | FG%  | 2P   | 2PA  | 2P%  | 3P  | 3PA | 3P%  | \ |
|---|---------------------|------|------|------|------|------|------|-----|-----|------|---|
| 0 | San Antonio Spurs*  | 3856 | 7738 | 49.8 | 3804 | 7532 | 50.5 | 52  | 206 | 25.2 |   |
| 1 | Los Angeles Lakers* | 3898 | 7368 | 52.9 | 3878 | 7268 | 53.4 | 20  | 100 | 20.0 |   |
| 2 | Cleveland Cavaliers | 3811 | 8041 | 47.4 | 3775 | 7854 | 48.1 | 36  | 187 | 19.3 |   |
| 3 | New York Knicks     | 3802 | 7672 | 49.6 | 3760 | 7481 | 50.3 | 42  | 191 | 22.0 |   |
| 4 | Boston Celtics*     | 3617 | 7387 | 49.0 | 3455 | 6965 | 49.6 | 162 | 422 | 38.4 |   |
| 5 | Indiana Pacers      | 3639 | 7689 | 47.3 | 3551 | 7375 | 48.1 | 88  | 314 | 28.0 |   |
| 6 | Phoenix Suns*       | 3570 | 7235 | 49.3 | 3502 | 6955 | 50.4 | 68  | 280 | 24.3 |   |
| 7 | Houston Rockets*    | 3599 | 7496 | 48.0 | 3495 | 7117 | 49.1 | 104 | 379 | 27.4 |   |
| 8 | Milwaukee Bucks*    | 3685 | 7553 | 48.8 | 3635 | 7398 | 49.1 | 50  | 155 | 32.3 |   |
| 9 | Philadelphia 76ers* | 3523 | 7156 | 49.2 | 3496 | 7031 | 49.7 | 27  | 125 | 21.6 |   |

|   | FT   | FTA  | ORB  | TRB  | AST  | TOV  | PTS  | Year | Playoff |
|---|------|------|------|------|------|------|------|------|---------|
| 0 | 2024 | 2528 | 1153 | 3668 | 2326 | 1589 | 9788 | 1980 | True    |
| 1 | 1622 | 2092 | 1085 | 3738 | 2413 | 1639 | 9438 | 1980 | True    |
| 2 | 1702 | 2205 | 1307 | 3688 | 2108 | 1370 | 9360 | 1980 | False   |
| 3 | 1698 | 2274 | 1236 | 3539 | 2265 | 1613 | 9344 | 1980 | False   |
| 4 | 1907 | 2449 | 1227 | 3684 | 2198 | 1539 | 9303 | 1980 | True    |
| 5 | 1753 | 2333 | 1398 | 3724 | 2148 | 1517 | 9119 | 1980 | False   |
| 6 | 1906 | 2466 | 1071 | 3529 | 2283 | 1629 | 9114 | 1980 | True    |
| 7 | 1782 | 2326 | 1394 | 3611 | 2149 | 1565 | 9084 | 1980 | True    |
| 8 | 1605 | 2102 | 1245 | 3641 | 2277 | 1496 | 9025 | 1980 | True    |
| 9 | 1876 | 2431 | 1187 | 3822 | 2226 | 1708 | 8949 | 1980 | True    |

#### 4. Clean Team column with clean\_team\_col function.

```
[40]: total_df = clean_team_col(total_df)
total_df.head()
```

|   | Team                | FG   | FGA  | FG%  | 2P   | 2PA  | 2P%  | 3P  | 3PA | 3P%  | \ |
|---|---------------------|------|------|------|------|------|------|-----|-----|------|---|
| 0 | San Antonio Spurs   | 3856 | 7738 | 49.8 | 3804 | 7532 | 50.5 | 52  | 206 | 25.2 |   |
| 1 | Los Angeles Lakers  | 3898 | 7368 | 52.9 | 3878 | 7268 | 53.4 | 20  | 100 | 20.0 |   |
| 2 | Cleveland Cavaliers | 3811 | 8041 | 47.4 | 3775 | 7854 | 48.1 | 36  | 187 | 19.3 |   |
| 3 | New York Knicks     | 3802 | 7672 | 49.6 | 3760 | 7481 | 50.3 | 42  | 191 | 22.0 |   |
| 4 | Boston Celtics      | 3617 | 7387 | 49.0 | 3455 | 6965 | 49.6 | 162 | 422 | 38.4 |   |

|   | FT   | FTA  | ORB  | TRB  | AST  | TOV  | PTS  | Year | Playoff |
|---|------|------|------|------|------|------|------|------|---------|
| 0 | 2024 | 2528 | 1153 | 3668 | 2326 | 1589 | 9788 | 1980 | True    |
| 1 | 1622 | 2092 | 1085 | 3738 | 2413 | 1639 | 9438 | 1980 | True    |
| 2 | 1702 | 2205 | 1307 | 3688 | 2108 | 1370 | 9360 | 1980 | False   |
| 3 | 1698 | 2274 | 1236 | 3539 | 2265 | 1613 | 9344 | 1980 | False   |
| 4 | 1907 | 2449 | 1227 | 3684 | 2198 | 1539 | 9303 | 1980 | True    |

## 1.4.2 avg\_df

### 1. Drop unneeded rows by creating a subset.

```
[41]: avg_df = avg_df[['Team', 'FG', 'FGA', 'FG%', '2P', '2PA',  
                     '2P%', '3P', '3PA', '3P%', 'FT', 'FTA', 'FT%', 'AST', 'ORB',  
                     'TRB', 'PTS', 'Year']]
```

```
[42]: avg_df.head()
```

```
[42]:
```

|   | Team                | FG   | FGA  | FG%   | 2P   | 2PA  | 2P%   | 3P  | 3PA | 3P%   | \ |
|---|---------------------|------|------|-------|------|------|-------|-----|-----|-------|---|
| 0 | San Antonio Spurs*  | 47.0 | 94.4 | 0.498 | 46.4 | 91.9 | 0.505 | 0.6 | 2.5 | 0.252 |   |
| 1 | Los Angeles Lakers* | 47.5 | 89.9 | 0.529 | 47.3 | 88.6 | 0.534 | 0.2 | 1.2 | 0.200 |   |
| 2 | Cleveland Cavaliers | 46.5 | 98.1 | 0.474 | 46.0 | 95.8 | 0.481 | 0.4 | 2.3 | 0.193 |   |
| 3 | New York Knicks     | 46.4 | 93.6 | 0.496 | 45.9 | 91.2 | 0.503 | 0.5 | 2.3 | 0.220 |   |
| 4 | Boston Celtics*     | 44.1 | 90.1 | 0.490 | 42.1 | 84.9 | 0.496 | 2.0 | 5.1 | 0.384 |   |

|   | FT   | FTA  | FT%   | AST  | ORB  | TRB  | PTS   | Year |
|---|------|------|-------|------|------|------|-------|------|
| 0 | 24.7 | 30.8 | 0.801 | 28.4 | 14.1 | 44.7 | 119.4 | 1980 |
| 1 | 19.8 | 25.5 | 0.775 | 29.4 | 13.2 | 45.6 | 115.1 | 1980 |
| 2 | 20.8 | 26.9 | 0.772 | 25.7 | 15.9 | 45.0 | 114.1 | 1980 |
| 3 | 20.7 | 27.7 | 0.747 | 27.6 | 15.1 | 43.2 | 114.0 | 1980 |
| 4 | 23.3 | 29.9 | 0.779 | 26.8 | 15.0 | 44.9 | 113.5 | 1980 |

### 2. Clean Team column with clean\_team\_col function.

```
[43]: avg_df = clean_team_col(avg_df)  
avg_df.head()
```

```
[43]:
```

|   | Team                | FG   | FGA  | FG%   | 2P   | 2PA  | 2P%   | 3P  | 3PA | 3P%   | \ |
|---|---------------------|------|------|-------|------|------|-------|-----|-----|-------|---|
| 0 | San Antonio Spurs   | 47.0 | 94.4 | 0.498 | 46.4 | 91.9 | 0.505 | 0.6 | 2.5 | 0.252 |   |
| 1 | Los Angeles Lakers  | 47.5 | 89.9 | 0.529 | 47.3 | 88.6 | 0.534 | 0.2 | 1.2 | 0.200 |   |
| 2 | Cleveland Cavaliers | 46.5 | 98.1 | 0.474 | 46.0 | 95.8 | 0.481 | 0.4 | 2.3 | 0.193 |   |
| 3 | New York Knicks     | 46.4 | 93.6 | 0.496 | 45.9 | 91.2 | 0.503 | 0.5 | 2.3 | 0.220 |   |
| 4 | Boston Celtics      | 44.1 | 90.1 | 0.490 | 42.1 | 84.9 | 0.496 | 2.0 | 5.1 | 0.384 |   |

|   | FT   | FTA  | FT%   | AST  | ORB  | TRB  | PTS   | Year |
|---|------|------|-------|------|------|------|-------|------|
| 0 | 24.7 | 30.8 | 0.801 | 28.4 | 14.1 | 44.7 | 119.4 | 1980 |
| 1 | 19.8 | 25.5 | 0.775 | 29.4 | 13.2 | 45.6 | 115.1 | 1980 |
| 2 | 20.8 | 26.9 | 0.772 | 25.7 | 15.9 | 45.0 | 114.1 | 1980 |
| 3 | 20.7 | 27.7 | 0.747 | 27.6 | 15.1 | 43.2 | 114.0 | 1980 |
| 4 | 23.3 | 29.9 | 0.779 | 26.8 | 15.0 | 44.9 | 113.5 | 1980 |

### 3. Multiply ‘FG%’, ‘2P%’, and ‘3P%’ by 100.

```
[44]: for col in avg_df.columns:  
    if '%' in col:  
        avg_df[col] = avg_df[col].apply(lambda x: x * 100)  
avg_df.head()
```

[44]:

|   | Team                | FG   | FGA  | FG%  | 2P   | 2PA  | 2P%  | 3P  | 3PA | 3P%  | \ |
|---|---------------------|------|------|------|------|------|------|-----|-----|------|---|
| 0 | San Antonio Spurs   | 47.0 | 94.4 | 49.8 | 46.4 | 91.9 | 50.5 | 0.6 | 2.5 | 25.2 |   |
| 1 | Los Angeles Lakers  | 47.5 | 89.9 | 52.9 | 47.3 | 88.6 | 53.4 | 0.2 | 1.2 | 20.0 |   |
| 2 | Cleveland Cavaliers | 46.5 | 98.1 | 47.4 | 46.0 | 95.8 | 48.1 | 0.4 | 2.3 | 19.3 |   |
| 3 | New York Knicks     | 46.4 | 93.6 | 49.6 | 45.9 | 91.2 | 50.3 | 0.5 | 2.3 | 22.0 |   |
| 4 | Boston Celtics      | 44.1 | 90.1 | 49.0 | 42.1 | 84.9 | 49.6 | 2.0 | 5.1 | 38.4 |   |

|   | FT   | FTA  | FT%  | AST  | ORB  | TRB  | PTS   | Year |
|---|------|------|------|------|------|------|-------|------|
| 0 | 24.7 | 30.8 | 80.1 | 28.4 | 14.1 | 44.7 | 119.4 | 1980 |
| 1 | 19.8 | 25.5 | 77.5 | 29.4 | 13.2 | 45.6 | 115.1 | 1980 |
| 2 | 20.8 | 26.9 | 77.2 | 25.7 | 15.9 | 45.0 | 114.1 | 1980 |
| 3 | 20.7 | 27.7 | 74.7 | 27.6 | 15.1 | 43.2 | 114.0 | 1980 |
| 4 | 23.3 | 29.9 | 77.9 | 26.8 | 15.0 | 44.9 | 113.5 | 1980 |

#### 4. Merge advanced\_df with avg\_df.

[45]:

```
new_avg_df = advanced_df.merge(avg_df, how = "left", on = ['Team', 'Year'])
new_avg_df.head()
```

[45]:

|   | Team                  | Age  | Wins | Losses | MOV  | ORtg  | DRtg  | Pace  | 3PAr | \ |
|---|-----------------------|------|------|--------|------|-------|-------|-------|------|---|
| 0 | Boston Celtics        | 27.3 | 61   | 21     | 7.79 | 109.4 | 101.9 | 102.6 | 5.7  |   |
| 1 | Los Angeles Lakers    | 26.2 | 60   | 22     | 5.90 | 109.5 | 103.9 | 104.1 | 1.4  |   |
| 2 | Oklahoma City Thunder | 27.0 | 56   | 26     | 4.66 | 105.8 | 101.2 | 101.8 | 2.5  |   |
| 3 | Philadelphia 76ers    | 27.0 | 59   | 23     | 4.22 | 105.0 | 101.0 | 103.0 | 1.7  |   |
| 4 | Milwaukee Bucks       | 25.3 | 49   | 33     | 3.94 | 106.8 | 102.9 | 102.4 | 2.1  |   |

|   | eFG% | TS%  | TOV% | ORB% | Year | Playoff | FG   | FGA  | FG%  | 2P   | 2PA  | 2P%  | \ |
|---|------|------|------|------|------|---------|------|------|------|------|------|------|---|
| 0 | 50.1 | 55.0 | 15.4 | 34.8 | 1980 | True    | 44.1 | 90.1 | 49.0 | 42.1 | 84.9 | 49.6 |   |
| 1 | 53.0 | 56.9 | 16.5 | 32.6 | 1980 | True    | 47.5 | 89.9 | 52.9 | 47.3 | 88.6 | 53.4 |   |
| 2 | 47.4 | 52.0 | 14.9 | 36.4 | 1980 | True    | 43.3 | 92.3 | 47.0 | 42.6 | 90.0 | 47.4 |   |
| 3 | 49.4 | 54.4 | 17.2 | 33.5 | 1980 | True    | 43.0 | 87.3 | 49.2 | 42.6 | 85.7 | 49.7 |   |
| 4 | 49.1 | 53.2 | 15.0 | 35.2 | 1980 | True    | 44.9 | 92.1 | 48.8 | 44.3 | 90.2 | 49.1 |   |

|   | 3P  | 3PA | 3P%  | FT   | FTA  | FT%  | AST  | ORB  | TRB  | PTS   |
|---|-----|-----|------|------|------|------|------|------|------|-------|
| 0 | 2.0 | 5.1 | 38.4 | 23.3 | 29.9 | 77.9 | 26.8 | 15.0 | 44.9 | 113.5 |
| 1 | 0.2 | 1.2 | 20.0 | 19.8 | 25.5 | 77.5 | 29.4 | 13.2 | 45.6 | 115.1 |
| 2 | 0.7 | 2.3 | 31.2 | 21.1 | 27.5 | 76.8 | 24.9 | 16.8 | 47.9 | 108.5 |
| 3 | 0.3 | 1.5 | 21.6 | 22.9 | 29.6 | 77.2 | 27.1 | 14.5 | 46.6 | 109.1 |
| 4 | 0.6 | 1.9 | 32.3 | 19.6 | 25.6 | 76.4 | 27.8 | 15.2 | 44.4 | 110.1 |

#### 5. Rearrange columns for better viewing.

[46]:

```
new_avg_df = new_avg_df[['Team', 'Age', 'Wins', 'Losses', 'Year', 'Playoff', 'FG',
                        'FGA', 'FG%', '2P', '2PA', '2P%', '3P',
                        '3PA', '3P%', '3PAr', 'eFG%', 'FT', 'FTA', 'FT%', 'TS%',
                        'AST', 'ORB', 'TRB', 'PTS', 'MOV', 'ORtg', 'DRtg', 'Pace',
                        'TOV%', 'ORB%']]
```

new\_avg\_df.head()

```
[46]:
```

|   | Team                  | Age  | Wins | Losses | Year | Playoff | FG   | FGA  | FG%  | \ |
|---|-----------------------|------|------|--------|------|---------|------|------|------|---|
| 0 | Boston Celtics        | 27.3 | 61   | 21     | 1980 | True    | 44.1 | 90.1 | 49.0 |   |
| 1 | Los Angeles Lakers    | 26.2 | 60   | 22     | 1980 | True    | 47.5 | 89.9 | 52.9 |   |
| 2 | Oklahoma City Thunder | 27.0 | 56   | 26     | 1980 | True    | 43.3 | 92.3 | 47.0 |   |
| 3 | Philadelphia 76ers    | 27.0 | 59   | 23     | 1980 | True    | 43.0 | 87.3 | 49.2 |   |
| 4 | Milwaukee Bucks       | 25.3 | 49   | 33     | 1980 | True    | 44.9 | 92.1 | 48.8 |   |

|   | 2P   | 2PA  | 2P%  | 3P  | 3PA | 3P%  | 3PAr | eFG% | FT   | FTA  | FT%  | TS%  | AST  | \ |
|---|------|------|------|-----|-----|------|------|------|------|------|------|------|------|---|
| 0 | 42.1 | 84.9 | 49.6 | 2.0 | 5.1 | 38.4 | 5.7  | 50.1 | 23.3 | 29.9 | 77.9 | 55.0 | 26.8 |   |
| 1 | 47.3 | 88.6 | 53.4 | 0.2 | 1.2 | 20.0 | 1.4  | 53.0 | 19.8 | 25.5 | 77.5 | 56.9 | 29.4 |   |
| 2 | 42.6 | 90.0 | 47.4 | 0.7 | 2.3 | 31.2 | 2.5  | 47.4 | 21.1 | 27.5 | 76.8 | 52.0 | 24.9 |   |
| 3 | 42.6 | 85.7 | 49.7 | 0.3 | 1.5 | 21.6 | 1.7  | 49.4 | 22.9 | 29.6 | 77.2 | 54.4 | 27.1 |   |
| 4 | 44.3 | 90.2 | 49.1 | 0.6 | 1.9 | 32.3 | 2.1  | 49.1 | 19.6 | 25.6 | 76.4 | 53.2 | 27.8 |   |

|   | ORB  | TRB  | PTS   | MOV  | ORtg  | DRtg  | Pace  | TOV% | ORB% | \ |
|---|------|------|-------|------|-------|-------|-------|------|------|---|
| 0 | 15.0 | 44.9 | 113.5 | 7.79 | 109.4 | 101.9 | 102.6 | 15.4 | 34.8 |   |
| 1 | 13.2 | 45.6 | 115.1 | 5.90 | 109.5 | 103.9 | 104.1 | 16.5 | 32.6 |   |
| 2 | 16.8 | 47.9 | 108.5 | 4.66 | 105.8 | 101.2 | 101.8 | 14.9 | 36.4 |   |
| 3 | 14.5 | 46.6 | 109.1 | 4.22 | 105.0 | 101.0 | 103.0 | 17.2 | 33.5 |   |
| 4 | 15.2 | 44.4 | 110.1 | 3.94 | 106.8 | 102.9 | 102.4 | 15.0 | 35.2 |   |

```
[47]: len(new_avg_df.columns)
```

[47]: 31

### 1.4.3 champions\_df

#### 1. Drop unneeded rows by creating a subset.

```
[48]: champions_df = champions_df[['Year', 'Champion', 'Runner-Up']]
champions_df.head()
```

```
[48]:
```

|   | Year   | Champion              | Runner-Up             |
|---|--------|-----------------------|-----------------------|
| 0 | 2021.0 | Milwaukee Bucks       | Phoenix Suns          |
| 1 | 2020.0 | Los Angeles Lakers    | Miami Heat            |
| 2 | 2019.0 | Toronto Raptors       | Golden State Warriors |
| 3 | 2018.0 | Golden State Warriors | Cleveland Cavaliers   |
| 4 | 2017.0 | Golden State Warriors | Cleveland Cavaliers   |

#### 2. Remove unneeded rows.

```
[49]: # the rows we are interested are in the year column beginning with 1980
champions_df = champions_df[champions_df.Year >= 1980]
champions_df.tail()
```

```
[49]:
```

|    | Year   | Champion           | Runner-Up          |
|----|--------|--------------------|--------------------|
| 38 | 1984.0 | Boston Celtics     | Los Angeles Lakers |
| 39 | 1983.0 | Philadelphia 76ers | Los Angeles Lakers |

```

40 1982.0 Los Angeles Lakers Philadelphia 76ers
42 1981.0 Boston Celtics Houston Rockets
43 1980.0 Los Angeles Lakers Philadelphia 76ers

```

### 3. Convert Year column to a integer datatype.

```
[50]: champions_df['Year'] = champions_df['Year'].astype(int)
np.dtype(champions_df['Year'])
```

```
[50]: dtype('int64')
```

### 4. Merge **champions\_df** with **new\_avg\_df** and **total\_df**. > Here we will melt the **champions\_df** dataset before we merge it. I want to make sure that the 'Champions' and 'Runner-Up' column names become values before I merge it with the other 2 datasets.

```
[51]: melted_champions = pd.melt(champions_df, id_vars =['Year'], value_vars=
    ['Champion', 'Runner-Up'])
melted_champions.rename(columns = {'value':'Team', 'variable':'Finals_Rk'}, u
    s inplace = True)
melted_champions.columns
```

```
[51]: Index(['Year', 'Finals_Rk', 'Team'], dtype='object')
```

```
[52]: new_avg_df = new_avg_df.merge(melted_champions, how = "left", on = ['Year', u
    'Team'])
new_avg_df.head()
```

|   | Team                  | Age  | Wins  | Losses | Year  | Playoff | FG    | FGA  | FG%  |           |      |      |      |  |
|---|-----------------------|------|-------|--------|-------|---------|-------|------|------|-----------|------|------|------|--|
| 0 | Boston Celtics        | 27.3 | 61    | 21     | 1980  | True    | 44.1  | 90.1 | 49.0 |           |      |      |      |  |
| 1 | Los Angeles Lakers    | 26.2 | 60    | 22     | 1980  | True    | 47.5  | 89.9 | 52.9 |           |      |      |      |  |
| 2 | Oklahoma City Thunder | 27.0 | 56    | 26     | 1980  | True    | 43.3  | 92.3 | 47.0 |           |      |      |      |  |
| 3 | Philadelphia 76ers    | 27.0 | 59    | 23     | 1980  | True    | 43.0  | 87.3 | 49.2 |           |      |      |      |  |
| 4 | Milwaukee Bucks       | 25.3 | 49    | 33     | 1980  | True    | 44.9  | 92.1 | 48.8 |           |      |      |      |  |
|   | 2P                    | 2PA  | 2P%   | 3P     | 3PA   | 3P%     | 3PAr  | eFG% | FT   | FTA       | FT%  | TS%  | AST  |  |
| 0 | 42.1                  | 84.9 | 49.6  | 2.0    | 5.1   | 38.4    | 5.7   | 50.1 | 23.3 | 29.9      | 77.9 | 55.0 | 26.8 |  |
| 1 | 47.3                  | 88.6 | 53.4  | 0.2    | 1.2   | 20.0    | 1.4   | 53.0 | 19.8 | 25.5      | 77.5 | 56.9 | 29.4 |  |
| 2 | 42.6                  | 90.0 | 47.4  | 0.7    | 2.3   | 31.2    | 2.5   | 47.4 | 21.1 | 27.5      | 76.8 | 52.0 | 24.9 |  |
| 3 | 42.6                  | 85.7 | 49.7  | 0.3    | 1.5   | 21.6    | 1.7   | 49.4 | 22.9 | 29.6      | 77.2 | 54.4 | 27.1 |  |
| 4 | 44.3                  | 90.2 | 49.1  | 0.6    | 1.9   | 32.3    | 2.1   | 49.1 | 19.6 | 25.6      | 76.4 | 53.2 | 27.8 |  |
|   | ORB                   | TRB  | PTS   | MOV    | ORtg  | DRtg    | Pace  | TOV% | ORB% | Finals_Rk |      |      |      |  |
| 0 | 15.0                  | 44.9 | 113.5 | 7.79   | 109.4 | 101.9   | 102.6 | 15.4 | 34.8 |           | NaN  |      |      |  |
| 1 | 13.2                  | 45.6 | 115.1 | 5.90   | 109.5 | 103.9   | 104.1 | 16.5 | 32.6 | Champion  |      |      |      |  |
| 2 | 16.8                  | 47.9 | 108.5 | 4.66   | 105.8 | 101.2   | 101.8 | 14.9 | 36.4 |           | NaN  |      |      |  |
| 3 | 14.5                  | 46.6 | 109.1 | 4.22   | 105.0 | 101.0   | 103.0 | 17.2 | 33.5 | Runner-Up |      |      |      |  |
| 4 | 15.2                  | 44.4 | 110.1 | 3.94   | 106.8 | 102.9   | 102.4 | 15.0 | 35.2 |           | NaN  |      |      |  |

```
[53]: total_df = total_df.merge(melted_champions, how = "left", on = ['Year', 'Team'])
total_df.head()
```

|   | Team                | FG   | FGA  | FG%  | 2P   | 2PA  | 2P%  | 3P  | 3PA | 3P%  | \ |
|---|---------------------|------|------|------|------|------|------|-----|-----|------|---|
| 0 | San Antonio Spurs   | 3856 | 7738 | 49.8 | 3804 | 7532 | 50.5 | 52  | 206 | 25.2 |   |
| 1 | Los Angeles Lakers  | 3898 | 7368 | 52.9 | 3878 | 7268 | 53.4 | 20  | 100 | 20.0 |   |
| 2 | Cleveland Cavaliers | 3811 | 8041 | 47.4 | 3775 | 7854 | 48.1 | 36  | 187 | 19.3 |   |
| 3 | New York Knicks     | 3802 | 7672 | 49.6 | 3760 | 7481 | 50.3 | 42  | 191 | 22.0 |   |
| 4 | Boston Celtics      | 3617 | 7387 | 49.0 | 3455 | 6965 | 49.6 | 162 | 422 | 38.4 |   |

|   | FT   | FTA  | ORB  | TRB  | AST  | TOV  | PTS  | Year | Playoff | Finals_Rk |
|---|------|------|------|------|------|------|------|------|---------|-----------|
| 0 | 2024 | 2528 | 1153 | 3668 | 2326 | 1589 | 9788 | 1980 | True    | NaN       |
| 1 | 1622 | 2092 | 1085 | 3738 | 2413 | 1639 | 9438 | 1980 | True    | Champion  |
| 2 | 1702 | 2205 | 1307 | 3688 | 2108 | 1370 | 9360 | 1980 | False   | NaN       |
| 3 | 1698 | 2274 | 1236 | 3539 | 2265 | 1613 | 9344 | 1980 | False   | NaN       |
| 4 | 1907 | 2449 | 1227 | 3684 | 2198 | 1539 | 9303 | 1980 | True    | NaN       |

5. Fill in null values in new\_avg\_df and total\_df. > Here is an extra step that we will have to perform. We should try to avoid null values as much as possible so I am going to fill in the null values with ‘Knocked Out’ to indicate a team got knocked out of the playoff before they could reach the NBA Finals(Finals\_Rk column). We will also fill the Finals\_Rk column for those who never qualified for the playoff with ‘Never Qualified’.

```
[54]: # create new column 'Finals_Rk' based on conditions
for i, row in new_avg_df.iterrows():
    if ((row.Playoff == True) and (pd.isna(row.Finals_Rk))):
        new_avg_df.at[i, 'Finals_Rk'] = 'Knocked Out'
    elif ((row.Playoff == False) and (pd.isna(row.Finals_Rk))):
        new_avg_df.at[i, 'Finals_Rk'] = 'Never Qualified'
    else:
        pass

new_avg_df.head()
```

|   | Team                  | Age  | Wins | Losses | Year | Playoff | FG   | FGA  | FG%  | \ |
|---|-----------------------|------|------|--------|------|---------|------|------|------|---|
| 0 | Boston Celtics        | 27.3 | 61   | 21     | 1980 | True    | 44.1 | 90.1 | 49.0 |   |
| 1 | Los Angeles Lakers    | 26.2 | 60   | 22     | 1980 | True    | 47.5 | 89.9 | 52.9 |   |
| 2 | Oklahoma City Thunder | 27.0 | 56   | 26     | 1980 | True    | 43.3 | 92.3 | 47.0 |   |
| 3 | Philadelphia 76ers    | 27.0 | 59   | 23     | 1980 | True    | 43.0 | 87.3 | 49.2 |   |
| 4 | Milwaukee Bucks       | 25.3 | 49   | 33     | 1980 | True    | 44.9 | 92.1 | 48.8 |   |

|   | 2P   | 2PA  | 2P%  | 3P  | 3PA | 3P%  | 3PAr | eFG% | FT   | FTA  | FT%  | TS%  | AST  | \ |
|---|------|------|------|-----|-----|------|------|------|------|------|------|------|------|---|
| 0 | 42.1 | 84.9 | 49.6 | 2.0 | 5.1 | 38.4 | 5.7  | 50.1 | 23.3 | 29.9 | 77.9 | 55.0 | 26.8 |   |
| 1 | 47.3 | 88.6 | 53.4 | 0.2 | 1.2 | 20.0 | 1.4  | 53.0 | 19.8 | 25.5 | 77.5 | 56.9 | 29.4 |   |
| 2 | 42.6 | 90.0 | 47.4 | 0.7 | 2.3 | 31.2 | 2.5  | 47.4 | 21.1 | 27.5 | 76.8 | 52.0 | 24.9 |   |
| 3 | 42.6 | 85.7 | 49.7 | 0.3 | 1.5 | 21.6 | 1.7  | 49.4 | 22.9 | 29.6 | 77.2 | 54.4 | 27.1 |   |
| 4 | 44.3 | 90.2 | 49.1 | 0.6 | 1.9 | 32.3 | 2.1  | 49.1 | 19.6 | 25.6 | 76.4 | 53.2 | 27.8 |   |

|   | ORB  | TRB  | PTS   | MOV  | ORtg  | DRtg  | Pace  | TOV% | ORB% | Finals_Rk   |
|---|------|------|-------|------|-------|-------|-------|------|------|-------------|
| 0 | 15.0 | 44.9 | 113.5 | 7.79 | 109.4 | 101.9 | 102.6 | 15.4 | 34.8 | Knocked Out |
| 1 | 13.2 | 45.6 | 115.1 | 5.90 | 109.5 | 103.9 | 104.1 | 16.5 | 32.6 | Champion    |
| 2 | 16.8 | 47.9 | 108.5 | 4.66 | 105.8 | 101.2 | 101.8 | 14.9 | 36.4 | Knocked Out |
| 3 | 14.5 | 46.6 | 109.1 | 4.22 | 105.0 | 101.0 | 103.0 | 17.2 | 33.5 | Runner-Up   |
| 4 | 15.2 | 44.4 | 110.1 | 3.94 | 106.8 | 102.9 | 102.4 | 15.0 | 35.2 | Knocked Out |

```
[55]: for i, row in total_df.iterrows():
    if ((row.Playoff == True) and (pd.isna(row.Finals_Rk))):
        total_df.at[i, 'Finals_Rk'] = 'Knocked Out'
    elif ((row.Playoff == False) and (pd.isna(row.Finals_Rk))):
        total_df.at[i, 'Finals_Rk'] = 'Never Qualified'
    else:
        pass
total_df.head(10)
```

|   | Team                | FG   | FGA  | FG%  | 2P   | 2PA  | 2P%  | 3P  | 3PA | 3P%  | \ |
|---|---------------------|------|------|------|------|------|------|-----|-----|------|---|
| 0 | San Antonio Spurs   | 3856 | 7738 | 49.8 | 3804 | 7532 | 50.5 | 52  | 206 | 25.2 |   |
| 1 | Los Angeles Lakers  | 3898 | 7368 | 52.9 | 3878 | 7268 | 53.4 | 20  | 100 | 20.0 |   |
| 2 | Cleveland Cavaliers | 3811 | 8041 | 47.4 | 3775 | 7854 | 48.1 | 36  | 187 | 19.3 |   |
| 3 | New York Knicks     | 3802 | 7672 | 49.6 | 3760 | 7481 | 50.3 | 42  | 191 | 22.0 |   |
| 4 | Boston Celtics      | 3617 | 7387 | 49.0 | 3455 | 6965 | 49.6 | 162 | 422 | 38.4 |   |
| 5 | Indiana Pacers      | 3639 | 7689 | 47.3 | 3551 | 7375 | 48.1 | 88  | 314 | 28.0 |   |
| 6 | Phoenix Suns        | 3570 | 7235 | 49.3 | 3502 | 6955 | 50.4 | 68  | 280 | 24.3 |   |
| 7 | Houston Rockets     | 3599 | 7496 | 48.0 | 3495 | 7117 | 49.1 | 104 | 379 | 27.4 |   |
| 8 | Milwaukee Bucks     | 3685 | 7553 | 48.8 | 3635 | 7398 | 49.1 | 50  | 155 | 32.3 |   |
| 9 | Philadelphia 76ers  | 3523 | 7156 | 49.2 | 3496 | 7031 | 49.7 | 27  | 125 | 21.6 |   |

|   | FT   | FTA  | ORB  | TRB  | AST  | TOV  | PTS  | Year | Playoff | Finals_Rk       |
|---|------|------|------|------|------|------|------|------|---------|-----------------|
| 0 | 2024 | 2528 | 1153 | 3668 | 2326 | 1589 | 9788 | 1980 | True    | Knocked Out     |
| 1 | 1622 | 2092 | 1085 | 3738 | 2413 | 1639 | 9438 | 1980 | True    | Champion        |
| 2 | 1702 | 2205 | 1307 | 3688 | 2108 | 1370 | 9360 | 1980 | False   | Never Qualified |
| 3 | 1698 | 2274 | 1236 | 3539 | 2265 | 1613 | 9344 | 1980 | False   | Never Qualified |
| 4 | 1907 | 2449 | 1227 | 3684 | 2198 | 1539 | 9303 | 1980 | True    | Knocked Out     |
| 5 | 1753 | 2333 | 1398 | 3724 | 2148 | 1517 | 9119 | 1980 | False   | Never Qualified |
| 6 | 1906 | 2466 | 1071 | 3529 | 2283 | 1629 | 9114 | 1980 | True    | Knocked Out     |
| 7 | 1782 | 2326 | 1394 | 3611 | 2149 | 1565 | 9084 | 1980 | True    | Knocked Out     |
| 8 | 1605 | 2102 | 1245 | 3641 | 2277 | 1496 | 9025 | 1980 | True    | Knocked Out     |
| 9 | 1876 | 2431 | 1187 | 3822 | 2226 | 1708 | 8949 | 1980 | True    | Runner-Up       |

## 1.5 Storing Data

Before we store them, let's have one last look at the two datasets to make sure that we have cleaned it thoroughly.

```
[56]: new_avg_df.info()
```

```

<class 'pandas.core.frame.DataFrame'>
Int64Index: 1164 entries, 0 to 1163
Data columns (total 32 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   Team         1164 non-null   object  
 1   Age          1164 non-null   float64 
 2   Wins         1164 non-null   int64   
 3   Losses        1164 non-null   int64   
 4   Year          1164 non-null   int64   
 5   Playoff       1164 non-null   bool    
 6   FG            1164 non-null   float64 
 7   FGA           1164 non-null   float64 
 8   FG%           1164 non-null   float64 
 9   2P            1164 non-null   float64 
 10  2PA           1164 non-null   float64 
 11  2P%           1164 non-null   float64 
 12  3P            1164 non-null   float64 
 13  3PA           1164 non-null   float64 
 14  3P%           1164 non-null   float64 
 15  3PAr          1164 non-null   float64 
 16  eFG%          1164 non-null   float64 
 17  FT            1164 non-null   float64 
 18  FTA           1164 non-null   float64 
 19  FT%           1164 non-null   float64 
 20  TS%           1164 non-null   float64 
 21  AST           1164 non-null   float64 
 22  ORB           1164 non-null   float64 
 23  TRB           1164 non-null   float64 
 24  PTS           1164 non-null   float64 
 25  MOV           1164 non-null   float64 
 26  ORtg          1164 non-null   float64 
 27  DRtg          1164 non-null   float64 
 28  Pace          1164 non-null   float64 
 29  TOV%          1164 non-null   float64 
 30  ORB%          1164 non-null   float64 
 31  Finals_Rk    1164 non-null   object  
dtypes: bool(1), float64(26), int64(3), object(2)
memory usage: 324.4+ KB

```

[57]: total\_df.info()

```

<class 'pandas.core.frame.DataFrame'>
Int64Index: 1164 entries, 0 to 1163
Data columns (total 20 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   Team         1164 non-null   object  

```

```

1   FG          1164 non-null    int64
2   FGA         1164 non-null    int64
3   FG%        1164 non-null    float64
4   2P          1164 non-null    int64
5   2PA         1164 non-null    int64
6   2P%        1164 non-null    float64
7   3P          1164 non-null    int64
8   3PA         1164 non-null    int64
9   3P%        1164 non-null    float64
10  FT          1164 non-null    int64
11  FTA         1164 non-null    int64
12  ORB         1164 non-null    int64
13  TRB         1164 non-null    int64
14  AST          1164 non-null    int64
15  TOV          1164 non-null    int64
16  PTS          1164 non-null    int64
17  Year         1164 non-null    int64
18  Playoff      1164 non-null    bool
19  Finals_Rk   1164 non-null    object
dtypes: bool(1), float64(3), int64(14), object(2)
memory usage: 215.3+ KB

```

[58]: total\_df.head()

|   | Team                | FG   | FGA  | FG%  | 2P   | 2PA  | 2P%  | 3P   | 3PA     | 3P%             | \ |
|---|---------------------|------|------|------|------|------|------|------|---------|-----------------|---|
| 0 | San Antonio Spurs   | 3856 | 7738 | 49.8 | 3804 | 7532 | 50.5 | 52   | 206     | 25.2            |   |
| 1 | Los Angeles Lakers  | 3898 | 7368 | 52.9 | 3878 | 7268 | 53.4 | 20   | 100     | 20.0            |   |
| 2 | Cleveland Cavaliers | 3811 | 8041 | 47.4 | 3775 | 7854 | 48.1 | 36   | 187     | 19.3            |   |
| 3 | New York Knicks     | 3802 | 7672 | 49.6 | 3760 | 7481 | 50.3 | 42   | 191     | 22.0            |   |
| 4 | Boston Celtics      | 3617 | 7387 | 49.0 | 3455 | 6965 | 49.6 | 162  | 422     | 38.4            |   |
|   | FT                  | FTA  | ORB  | TRB  | AST  | TOV  | PTS  | Year | Playoff | Finals_Rk       |   |
| 0 | 2024                | 2528 | 1153 | 3668 | 2326 | 1589 | 9788 | 1980 | True    | Knocked Out     |   |
| 1 | 1622                | 2092 | 1085 | 3738 | 2413 | 1639 | 9438 | 1980 | True    | Champion        |   |
| 2 | 1702                | 2205 | 1307 | 3688 | 2108 | 1370 | 9360 | 1980 | False   | Never Qualified |   |
| 3 | 1698                | 2274 | 1236 | 3539 | 2265 | 1613 | 9344 | 1980 | False   | Never Qualified |   |
| 4 | 1907                | 2449 | 1227 | 3684 | 2198 | 1539 | 9303 | 1980 | True    | Knocked Out     |   |

[59]: new\_avg\_df.head(10)

|   | Team                  | Age  | Wins | Losses | Year | Playoff | FG   | FGA  | FG%  | \ |
|---|-----------------------|------|------|--------|------|---------|------|------|------|---|
| 0 | Boston Celtics        | 27.3 | 61   | 21     | 1980 | True    | 44.1 | 90.1 | 49.0 |   |
| 1 | Los Angeles Lakers    | 26.2 | 60   | 22     | 1980 | True    | 47.5 | 89.9 | 52.9 |   |
| 2 | Oklahoma City Thunder | 27.0 | 56   | 26     | 1980 | True    | 43.3 | 92.3 | 47.0 |   |
| 3 | Philadelphia 76ers    | 27.0 | 59   | 23     | 1980 | True    | 43.0 | 87.3 | 49.2 |   |
| 4 | Milwaukee Bucks       | 25.3 | 49   | 33     | 1980 | True    | 44.9 | 92.1 | 48.8 |   |
| 5 | Phoenix Suns          | 26.5 | 55   | 27     | 1980 | True    | 43.5 | 88.2 | 49.3 |   |

|   |                     |      |       |      |       |       |       |      |      |                 |      |      |      |             |
|---|---------------------|------|-------|------|-------|-------|-------|------|------|-----------------|------|------|------|-------------|
| 6 | Sacramento Kings    | 25.5 | 47    | 35   | 1980  | True  | 43.7  | 91.3 | 47.8 |                 |      |      |      |             |
| 7 | Atlanta Hawks       | 26.1 | 50    | 32   | 1980  | True  | 39.8  | 85.7 | 46.4 |                 |      |      |      |             |
| 8 | Cleveland Cavaliers | 27.4 | 37    | 45   | 1980  | False | 46.5  | 98.1 | 47.4 |                 |      |      |      |             |
| 9 | Houston Rockets     | 27.3 | 41    | 41   | 1980  | True  | 43.9  | 91.4 | 48.0 |                 |      |      |      |             |
|   | 2P                  | 2PA  | 2P%   | 3P   | 3PA   | 3P%   | 3PAr  | eFG% | FT   | FTA             | FT%  | TS%  | AST  | \           |
| 0 | 42.1                | 84.9 | 49.6  | 2.0  | 5.1   | 38.4  | 5.7   | 50.1 | 23.3 | 29.9            | 77.9 | 55.0 | 26.8 |             |
| 1 | 47.3                | 88.6 | 53.4  | 0.2  | 1.2   | 20.0  | 1.4   | 53.0 | 19.8 | 25.5            | 77.5 | 56.9 | 29.4 |             |
| 2 | 42.6                | 90.0 | 47.4  | 0.7  | 2.3   | 31.2  | 2.5   | 47.4 | 21.1 | 27.5            | 76.8 | 52.0 | 24.9 |             |
| 3 | 42.6                | 85.7 | 49.7  | 0.3  | 1.5   | 21.6  | 1.7   | 49.4 | 22.9 | 29.6            | 77.2 | 54.4 | 27.1 |             |
| 4 | 44.3                | 90.2 | 49.1  | 0.6  | 1.9   | 32.3  | 2.1   | 49.1 | 19.6 | 25.6            | 76.4 | 53.2 | 27.8 |             |
| 5 | 42.7                | 84.8 | 50.4  | 0.8  | 3.4   | 24.3  | 3.9   | 49.8 | 23.2 | 30.1            | 77.3 | 54.8 | 27.8 |             |
| 6 | 43.4                | 89.9 | 48.2  | 0.3  | 1.4   | 21.9  | 1.5   | 48.0 | 20.4 | 27.4            | 74.3 | 52.2 | 25.9 |             |
| 7 | 39.6                | 84.8 | 46.7  | 0.2  | 0.9   | 17.3  | 1.1   | 46.5 | 24.9 | 32.3            | 77.1 | 52.3 | 23.3 |             |
| 8 | 46.0                | 95.8 | 48.1  | 0.4  | 2.3   | 19.3  | 2.3   | 47.6 | 20.8 | 26.9            | 77.2 | 51.9 | 25.7 |             |
| 9 | 42.6                | 86.8 | 49.1  | 1.3  | 4.6   | 27.4  | 5.1   | 48.7 | 21.7 | 28.4            | 76.6 | 53.3 | 26.2 |             |
|   | ORB                 | TRB  | PTS   | MOV  | ORtg  | DRtg  | Pace  | TOV% | ORB% |                 |      |      |      | Finals_Rk   |
| 0 | 15.0                | 44.9 | 113.5 | 7.79 | 109.4 | 101.9 | 102.6 | 15.4 | 34.8 |                 |      |      |      | Knocked Out |
| 1 | 13.2                | 45.6 | 115.1 | 5.90 | 109.5 | 103.9 | 104.1 | 16.5 | 32.6 |                 |      |      |      | Champion    |
| 2 | 16.8                | 47.9 | 108.5 | 4.66 | 105.8 | 101.2 | 101.8 | 14.9 | 36.4 |                 |      |      |      | Knocked Out |
| 3 | 14.5                | 46.6 | 109.1 | 4.22 | 105.0 | 101.0 | 103.0 | 17.2 | 33.5 |                 |      |      |      | Runner-Up   |
| 4 | 15.2                | 44.4 | 110.1 | 3.94 | 106.8 | 102.9 | 102.4 | 15.0 | 35.2 |                 |      |      |      | Knocked Out |
| 5 | 13.1                | 43.0 | 111.1 | 3.60 | 105.6 | 102.2 | 104.8 | 16.4 | 30.4 |                 |      |      |      | Knocked Out |
| 6 | 14.5                | 44.1 | 108.0 | 3.13 | 104.0 | 101.0 | 103.2 | 14.5 | 31.0 |                 |      |      |      | Knocked Out |
| 7 | 16.7                | 46.0 | 104.5 | 2.91 | 105.2 | 102.3 | 98.9  | 15.4 | 36.9 |                 |      |      |      | Knocked Out |
| 8 | 15.9                | 45.0 | 114.1 | 0.34 | 106.7 | 106.4 | 105.6 | 13.2 | 33.1 | Never Qualified |      |      |      |             |
| 9 | 17.0                | 44.0 | 110.8 | 0.17 | 108.1 | 108.0 | 101.2 | 15.5 | 37.6 |                 |      |      |      | Knocked Out |

```
[60]: total_df.to_csv('data/cleaned_total_stats.csv', index=False)
new_avg_df.to_csv('data/cleaned_avg_stats.csv', index=False)
```

## 1.6 Conclusion

Now that I have cleaned and merged the datasets, I am now interested in exploring them, which I will be doing in a different notebook. In this notebook, some acts that I have performed during the cleaning process are cleaning up the Team column, creating new columns for later exploration, and dealing with null values. **Please head over to Part III - NBA Data Exploration.ipynb to continue.**

[ ]:

# part-iii-nba-data-exploration

February 24, 2024

## 1 NBA Data Exploration

### 1.1 Introduction

In this notebook, I mainly be analyzing the offensive stats since 1980 which most of it will be done in the first half. Most of it will be visualizing the total and averages of the entire league, but as you go further along, you will find that I will be interested in seperating these stats by teams and Finals rankings, as well as a new column called era. The goal of this notebook is to analyze the regular season offensive characteristics of champions, runner-ups, and those team who were knocked out or never made the playoff. First, I will examine the datasets and use the nba\_total\_df dataset to get a broad picture of what I am working with. Please keep in mind the year represents the season that ends in that year. For example, 1980 means the 1979-1980 season.

```
[53]: import pandas as pd
import numpy as np
import warnings
import seaborn as sb
import matplotlib.pyplot as plt

pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
warnings.filterwarnings('ignore')
```

### 1.2 Examining Dataset

```
[54]: nba_avg_df = pd.read_csv('data/cleaned_avg_stats.csv')
nba_total_df = pd.read_csv('data/cleaned_total_stats.csv')
```

```
[55]: # understanding the structure of the datasets
nba_avg_df.shape, nba_total_df.shape
```

```
[55]: ((1164, 32), (1164, 20))
```

```
[56]: nba_avg_df.isnull().sum()
```

```
[56]: Team      0
Age       0
```

```
Wins          0
Losses        0
Year          0
Playoff       0
FG            0
FGA           0
FG%           0
2P            0
2PA           0
2P%           0
3P            0
3PA           0
3P%           0
3PAr          0
eFG%          0
FT            0
FTA           0
FT%           0
TS%           0
AST           0
ORB           0
TRB           0
PTS           0
MOV           0
ORtg          0
DRtg          0
Pace          0
TOV%          0
ORB%          0
Finals_Rk    0
dtype: int64
```

```
[57]: nba_total_df.isnull().sum()
```

```
[57]: Team          0
FG            0
FGA           0
FG%           0
2P            0
2PA           0
2P%           0
3P            0
3PA           0
3P%           0
FT            0
FTA           0
ORB           0
```

```

TRB      0
AST      0
TOV      0
PTS      0
Year     0
Playoff   0
Finals_Rk 0
dtype: int64

```

[58]: nba\_avg\_df.head()

|   | Team                  | Age  | Wins  | Losses | Year  | Playoff | FG    | FGA  | FG%  | \    |             |      |      |   |
|---|-----------------------|------|-------|--------|-------|---------|-------|------|------|------|-------------|------|------|---|
| 0 | Boston Celtics        | 27.3 | 61    | 21     | 1980  | True    | 44.1  | 90.1 | 49.0 |      |             |      |      |   |
| 1 | Los Angeles Lakers    | 26.2 | 60    | 22     | 1980  | True    | 47.5  | 89.9 | 52.9 |      |             |      |      |   |
| 2 | Oklahoma City Thunder | 27.0 | 56    | 26     | 1980  | True    | 43.3  | 92.3 | 47.0 |      |             |      |      |   |
| 3 | Philadelphia 76ers    | 27.0 | 59    | 23     | 1980  | True    | 43.0  | 87.3 | 49.2 |      |             |      |      |   |
| 4 | Milwaukee Bucks       | 25.3 | 49    | 33     | 1980  | True    | 44.9  | 92.1 | 48.8 |      |             |      |      |   |
|   | 2P                    | 2PA  | 2P%   | 3P     | 3PA   | 3P%     | 3PAr  | eFG% | FT   | FTA  | FT%         | TS%  | AST  | \ |
| 0 | 42.1                  | 84.9 | 49.6  | 2.0    | 5.1   | 38.4    | 5.7   | 50.1 | 23.3 | 29.9 | 77.9        | 55.0 | 26.8 |   |
| 1 | 47.3                  | 88.6 | 53.4  | 0.2    | 1.2   | 20.0    | 1.4   | 53.0 | 19.8 | 25.5 | 77.5        | 56.9 | 29.4 |   |
| 2 | 42.6                  | 90.0 | 47.4  | 0.7    | 2.3   | 31.2    | 2.5   | 47.4 | 21.1 | 27.5 | 76.8        | 52.0 | 24.9 |   |
| 3 | 42.6                  | 85.7 | 49.7  | 0.3    | 1.5   | 21.6    | 1.7   | 49.4 | 22.9 | 29.6 | 77.2        | 54.4 | 27.1 |   |
| 4 | 44.3                  | 90.2 | 49.1  | 0.6    | 1.9   | 32.3    | 2.1   | 49.1 | 19.6 | 25.6 | 76.4        | 53.2 | 27.8 |   |
|   | ORB                   | TRB  | PTS   | MOV    | ORtg  | DRtg    | Pace  | TOV% | ORB% |      | Finals_Rk   |      |      |   |
| 0 | 15.0                  | 44.9 | 113.5 | 7.79   | 109.4 | 101.9   | 102.6 | 15.4 | 34.8 |      | Knocked Out |      |      |   |
| 1 | 13.2                  | 45.6 | 115.1 | 5.90   | 109.5 | 103.9   | 104.1 | 16.5 | 32.6 |      | Champion    |      |      |   |
| 2 | 16.8                  | 47.9 | 108.5 | 4.66   | 105.8 | 101.2   | 101.8 | 14.9 | 36.4 |      | Knocked Out |      |      |   |
| 3 | 14.5                  | 46.6 | 109.1 | 4.22   | 105.0 | 101.0   | 103.0 | 17.2 | 33.5 |      | Runner-Up   |      |      |   |
| 4 | 15.2                  | 44.4 | 110.1 | 3.94   | 106.8 | 102.9   | 102.4 | 15.0 | 35.2 |      | Knocked Out |      |      |   |

[59]: nba\_total\_df.head()

|   | Team                | FG   | FGA  | FG%  | 2P   | 2PA  | 2P%  | 3P   | 3PA     | 3P%  | \               |  |  |  |
|---|---------------------|------|------|------|------|------|------|------|---------|------|-----------------|--|--|--|
| 0 | San Antonio Spurs   | 3856 | 7738 | 49.8 | 3804 | 7532 | 50.5 | 52   | 206     | 25.2 |                 |  |  |  |
| 1 | Los Angeles Lakers  | 3898 | 7368 | 52.9 | 3878 | 7268 | 53.4 | 20   | 100     | 20.0 |                 |  |  |  |
| 2 | Cleveland Cavaliers | 3811 | 8041 | 47.4 | 3775 | 7854 | 48.1 | 36   | 187     | 19.3 |                 |  |  |  |
| 3 | New York Knicks     | 3802 | 7672 | 49.6 | 3760 | 7481 | 50.3 | 42   | 191     | 22.0 |                 |  |  |  |
| 4 | Boston Celtics      | 3617 | 7387 | 49.0 | 3455 | 6965 | 49.6 | 162  | 422     | 38.4 |                 |  |  |  |
|   | FT                  | FTA  | ORB  | TRB  | AST  | TOV  | PTS  | Year | Playoff |      | Finals_Rk       |  |  |  |
| 0 | 2024                | 2528 | 1153 | 3668 | 2326 | 1589 | 9788 | 1980 | True    |      | Knocked Out     |  |  |  |
| 1 | 1622                | 2092 | 1085 | 3738 | 2413 | 1639 | 9438 | 1980 | True    |      | Champion        |  |  |  |
| 2 | 1702                | 2205 | 1307 | 3688 | 2108 | 1370 | 9360 | 1980 | False   |      | Never Qualified |  |  |  |
| 3 | 1698                | 2274 | 1236 | 3539 | 2265 | 1613 | 9344 | 1980 | False   |      | Never Qualified |  |  |  |
| 4 | 1907                | 2449 | 1227 | 3684 | 2198 | 1539 | 9303 | 1980 | True    |      | Knocked Out     |  |  |  |

```
[60]: nba_total_df.describe()
```

|       | FG           | FGA         | FG%         | 2P          | 2PA         | \ |
|-------|--------------|-------------|-------------|-------------|-------------|---|
| count | 1164.000000  | 1164.000000 | 1164.000000 | 1164.000000 | 1164.000000 |   |
| mean  | 3122.521478  | 6747.084192 | 46.204811   | 2698.753436 | 5543.422680 |   |
| std   | 368.029085   | 640.322398  | 2.110644    | 508.265503  | 1025.623781 |   |
| min   | 1539.000000  | 3561.000000 | 40.100000   | 1327.000000 | 2761.000000 |   |
| 25%   | 2956.750000  | 6518.000000 | 44.700000   | 2400.750000 | 4939.000000 |   |
| 50%   | 3123.500000  | 6807.000000 | 46.100000   | 2578.000000 | 5384.500000 |   |
| 75%   | 3357.000000  | 7128.250000 | 47.500000   | 3117.000000 | 6419.500000 |   |
| max   | 3980.000000  | 8868.000000 | 54.500000   | 3954.000000 | 7873.000000 |   |
|       | 2P%          | 3P          | 3PA         | 3P%         | FT          | \ |
| count | 1164.000000  | 1164.000000 | 1164.000000 | 1164.000000 | 1164.000000 |   |
| mean  | 48.725859    | 423.768041  | 1203.661512 | 33.380241   | 1556.841065 |   |
| std   | 2.388647     | 269.368159  | 721.838568  | 4.696755    | 249.801021  |   |
| min   | 42.100000    | 10.000000   | 75.000000   | 10.400000   | 745.000000  |   |
| 25%   | 47.100000    | 203.500000  | 617.250000  | 32.200000   | 1409.750000 |   |
| 50%   | 48.600000    | 415.000000  | 1184.000000 | 34.600000   | 1555.000000 |   |
| 75%   | 50.200000    | 593.250000  | 1631.000000 | 36.300000   | 1721.250000 |   |
| max   | 56.700000    | 1323.000000 | 3721.000000 | 42.800000   | 2388.000000 |   |
|       | FTA          | ORB         | TRB         | AST         | TOV         | \ |
| count | 1164.000000  | 1164.000000 | 1164.000000 | 1164.000000 | 1164.000000 |   |
| mean  | 2061.621134  | 988.613402  | 3432.908076 | 1859.371993 | 1242.43299  |   |
| std   | 323.153204   | 184.373140  | 305.780930  | 259.814763  | 181.17617   |   |
| min   | 1077.000000  | 472.000000  | 1788.000000 | 782.000000  | 641.00000   |   |
| 25%   | 1867.750000  | 858.000000  | 3336.750000 | 1712.750000 | 1136.75000  |   |
| 50%   | 2075.000000  | 985.000000  | 3473.500000 | 1851.500000 | 1230.00000  |   |
| 75%   | 2288.000000  | 1115.250000 | 3611.000000 | 2028.250000 | 1351.00000  |   |
| max   | 3051.000000  | 1520.000000 | 4078.000000 | 2575.000000 | 1873.00000  |   |
|       | PTS          | Year        |             |             |             |   |
| count | 1164.000000  | 1164.000000 |             |             |             |   |
| mean  | 8225.652062  | 2001.587629 |             |             |             |   |
| std   | 865.297277   | 11.868369   |             |             |             |   |
| min   | 4095.000000  | 1980.000000 |             |             |             |   |
| 25%   | 7913.000000  | 1992.000000 |             |             |             |   |
| 50%   | 8291.500000  | 2002.000000 |             |             |             |   |
| 75%   | 8740.000000  | 2012.000000 |             |             |             |   |
| max   | 10371.000000 | 2021.000000 |             |             |             |   |

```
[61]: nba_avg_df.describe()
```

|       | Age         | Wins        | Losses      | Year        | FG          | \ |
|-------|-------------|-------------|-------------|-------------|-------------|---|
| count | 1164.000000 | 1164.000000 | 1164.000000 | 1164.000000 | 1164.000000 |   |
| mean  | 26.840206   | 40.118557   | 40.118557   | 2001.587629 | 38.870103   |   |

|     |           |           |           |             |           |
|-----|-----------|-----------|-----------|-------------|-----------|
| std | 1.646497  | 12.779687 | 12.731692 | 11.868369   | 3.272410  |
| min | 22.700000 | 7.000000  | 9.000000  | 1980.000000 | 30.800000 |
| 25% | 25.700000 | 30.000000 | 30.000000 | 1992.000000 | 36.400000 |
| 50% | 26.700000 | 41.000000 | 40.000000 | 2002.000000 | 38.500000 |
| 75% | 27.900000 | 50.000000 | 49.000000 | 2012.000000 | 41.300000 |
| max | 32.000000 | 73.000000 | 72.000000 | 2021.000000 | 48.500000 |

|       | FGA         | FG%         | 2P          | 2PA         | 2P%         | \ |
|-------|-------------|-------------|-------------|-------------|-------------|---|
| count | 1164.000000 | 1164.000000 | 1164.000000 | 1164.000000 | 1164.000000 |   |
| mean  | 84.049141   | 46.204811   | 33.544588   | 68.918299   | 48.725859   |   |
| std   | 4.668796    | 2.110644    | 5.362144    | 10.807625   | 2.388647    |   |
| min   | 71.200000   | 40.100000   | 23.100000   | 41.900000   | 42.100000   |   |
| 25%   | 80.600000   | 44.700000   | 29.700000   | 61.300000   | 47.100000   |   |
| 50%   | 83.800000   | 46.100000   | 31.500000   | 66.000000   | 48.600000   |   |
| 75%   | 87.400000   | 47.500000   | 38.000000   | 78.300000   | 50.200000   |   |
| max   | 108.100000  | 54.500000   | 48.200000   | 96.000000   | 56.700000   |   |

|       | 3P          | 3PA         | 3P%         | 3PAr        | eFG%        | \ |
|-------|-------------|-------------|-------------|-------------|-------------|---|
| count | 1164.000000 | 1164.000000 | 1164.000000 | 1164.000000 | 1164.000000 |   |
| mean  | 5.328351    | 15.130928   | 33.380241   | 18.121564   | 49.396220   |   |
| std   | 3.434205    | 9.220323    | 4.696755    | 10.729955   | 2.324549    |   |
| min   | 0.100000    | 0.900000    | 10.400000   | 1.100000    | 42.400000   |   |
| 25%   | 2.600000    | 7.900000    | 32.200000   | 9.000000    | 47.800000   |   |
| 50%   | 5.200000    | 14.700000   | 34.600000   | 18.250000   | 49.200000   |   |
| 75%   | 7.300000    | 20.200000   | 36.300000   | 24.800000   | 50.800000   |   |
| max   | 16.700000   | 45.400000   | 42.800000   | 51.900000   | 57.500000   |   |

|       | FT          | FTA         | FT%         | TS%         | AST         | \ |
|-------|-------------|-------------|-------------|-------------|-------------|---|
| count | 1164.000000 | 1164.000000 | 1164.000000 | 1164.000000 | 1164.000000 |   |
| mean  | 19.370103   | 25.657646   | 75.533247   | 53.709364   | 23.147509   |   |
| std   | 2.557043    | 3.294632    | 2.939163    | 2.144069    | 2.604412    |   |
| min   | 12.200000   | 16.600000   | 66.000000   | 46.800000   | 15.600000   |   |
| 25%   | 17.500000   | 23.300000   | 73.800000   | 52.300000   | 21.200000   |   |
| 50%   | 19.100000   | 25.500000   | 75.600000   | 53.600000   | 22.900000   |   |
| 75%   | 21.000000   | 28.000000   | 77.500000   | 55.100000   | 25.000000   |   |
| max   | 29.100000   | 37.200000   | 83.900000   | 61.000000   | 31.400000   |   |

|       | ORB         | TRB         | PTS         | MOV         | ORtg        | \ |
|-------|-------------|-------------|-------------|-------------|-------------|---|
| count | 1164.000000 | 1164.000000 | 1164.000000 | 1164.000000 | 1164.000000 |   |
| mean  | 12.304210   | 42.781959   | 102.439347  | -0.004072   | 106.776546  |   |
| std   | 2.002756    | 2.139628    | 7.235404    | 4.638996    | 3.939481    |   |
| min   | 7.600000    | 35.600000   | 81.900000   | -15.200000  | 92.200000   |   |
| 25%   | 10.800000   | 41.300000   | 97.000000   | -3.182500   | 104.175000  |   |
| 50%   | 12.150000   | 42.700000   | 102.000000  | 0.245000    | 106.600000  |   |
| 75%   | 13.600000   | 44.300000   | 107.800000  | 3.425000    | 109.500000  |   |
| max   | 18.500000   | 51.700000   | 126.500000  | 12.240000   | 118.300000  |   |

|       | DRtg        | Pace        | TOV%        | ORB%        |
|-------|-------------|-------------|-------------|-------------|
| count | 1164.000000 | 1164.000000 | 1164.000000 | 1164.000000 |
| mean  | 106.769674  | 95.209021   | 13.945876   | 28.744244   |
| std   | 3.661148    | 4.936386    | 1.268837    | 4.257576    |
| min   | 94.100000   | 82.300000   | 9.900000    | 17.900000   |
| 25%   | 104.300000  | 91.375000   | 13.100000   | 25.600000   |
| 50%   | 107.000000  | 94.600000   | 13.900000   | 29.000000   |
| 75%   | 109.400000  | 98.900000   | 14.800000   | 31.800000   |
| max   | 117.600000  | 113.700000  | 18.700000   | 39.100000   |

### 1.3 Data Exploration - Visualization and Analysis

#### 1.4 Questions - nba\_total\_df

**nba\_total\_df** 1. How many of the total points scored were from 3-pointers? 2. What is the trend like total for 3-point and 2-point shot attempts over the years? What about Free-Throw attempts? 3. Is there similar trend for the other offensive stats(Offensive Rebound & Assists)? Analyze them too. 4. How many times did each team make the playoff since 1980? 5. How many teams won the NBA championships since 1980? How many times did each team win? 6. Examine the Finals\_Rk column by Team. How many times did each team get knocked out, never qualified, become runner-up or champion?

##### 1.4.1 nba\_total\_df

First, I want to mainly focus on big numbers which is why I decided to start with the **nba\_total\_df** dataset. I do this to get a big picture view of the data that I am working with before I move onto the smaller numbers(percentages). I believe that after seeing numbers that big I might find some surprising results later on. However, I will also include some statistics when answering some of the questions during my analysis process of this dataset.

###### 1. How many of the total points scored were from 3-pointers?

```
[62]: # dataframe for total PTS scored throughout all NBA seasons
total_team_pts= pd.DataFrame(nba_total_df.groupby('Team')['PTS'].sum().
                                sort_values(ascending = False).reset_index())

# dataframe for total 3-PTS scored throughout all NBA seasons
# 3P only counts the number of shots that went in, so we must multiple that by 3
total_3pts_score = pd.DataFrame((nba_total_df.groupby('Team')['3P'].sum()*3).
                                reset_index())

# Initialize the matplotlib figure
f, ax = plt.subplots(figsize=(25, 15))

#plot total PTS on bar graph
base_color = sb.color_palette("Set2")[5]
sb.barplot(data = total_team_pts, x = 'PTS', y = 'Team', color = base_color,
```

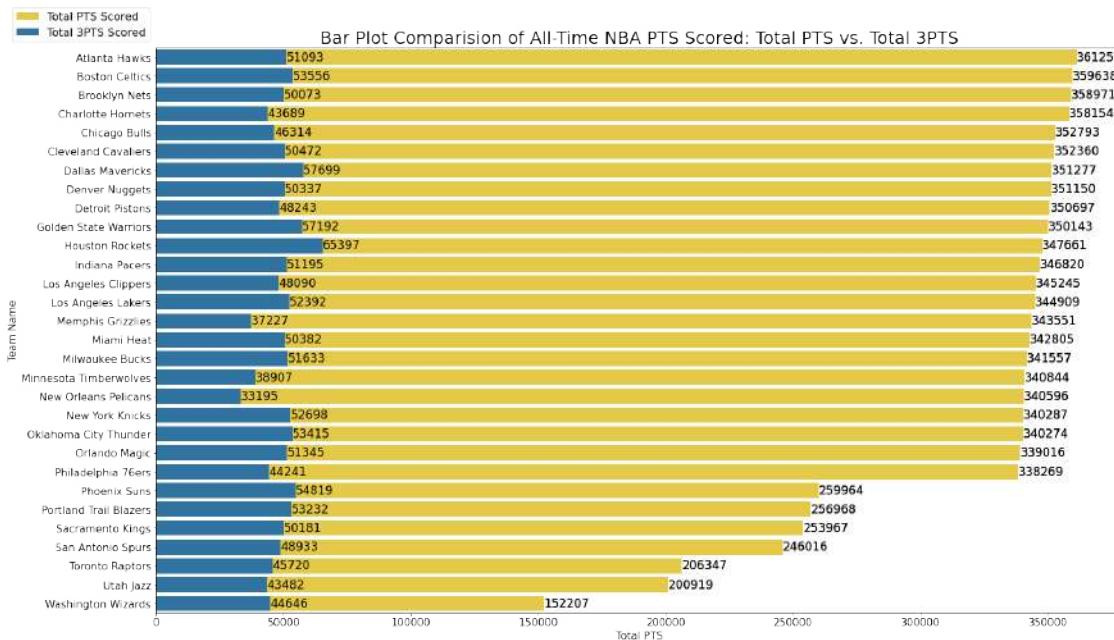
```

        label = 'Total PTS Scored')
for i in ax.containers:
    ax.bar_label(i, fontsize = 17);

# plot 3PT on bar graph
base_color_2 = sb.color_palette()[0]
sb.barplot(data = total_3pts_score, x = '3P', y = 'Team', color = base_color_2,
            label = 'Total 3PTS Scored')
for i in ax.containers:
    ax.bar_label(i, fontsize = 17);

ax.legend(frameon=True, prop={'size': 16}, bbox_to_anchor=(0, 1.0),
          loc="lower right")
plt.title('Bar Plot Comparision of All-Time NBA PTS Scored: Total PTS vs. Total 3PTS', fontsize = 25)
plt.ylabel('Team Name', fontsize = 15)
plt.xlabel('Total PTS', fontsize = 15)
plt.xticks(fontsize = 15)
plt.yticks(fontsize = 15);

```



[63]: prop\_3pts = ((nba\_total\_df.groupby('Team')['3P'].sum()\*3)/nba\_total\_df.  
 groupby('Team')['PTS'].sum()).sort\_values(ascending = False)  
prop\_3pts.head()

[63]: Team  
Toronto Raptors 0.221569

```
New Orleans Pelicans    0.218091
Orlando Magic          0.199811
Miami Heat              0.193804
Houston Rockets         0.186772
dtype: float64
```

**Question 1 Observation** > Before I jump into the explanation, please note that all teams are updated to their current names. For example, the Seattle Supersonics moved to Oklahoma City and became the Thunders. We did not count the SuperSonics' stats but combined it with Oklahoma City because of their shared franchise history. In this bar graph, we see that the Atlanta Hawks have scored the most regular-season points in NBA history with 361,254 points as of the end of the year 2021 (2020-2021 season). The Boston Celtics and Brooklyn Nets came in second and third, respectively. However, the Hawks do not take the top spot for the total of 3 points made; that title belongs to the Houston Rockets with 65,397 points that were scored from the 3-point line or beyond. Trailing the Rockets are the Dallas Mavericks and Golden State Warriors; the latter is very close to overtaking the former. Through further statistical analysis, I also found that 22.16% of the Raptors' total points were 3-pointers. However, upon even further research online, we found that the team was founded in 1995. They have more than Charlotte Hornets, who have scored far more total points and a long history! This leads me to my next question.

**2. What is the trend like total for 3-point and 2-point shot attempts over the years? What about Free-Throw attempts?**

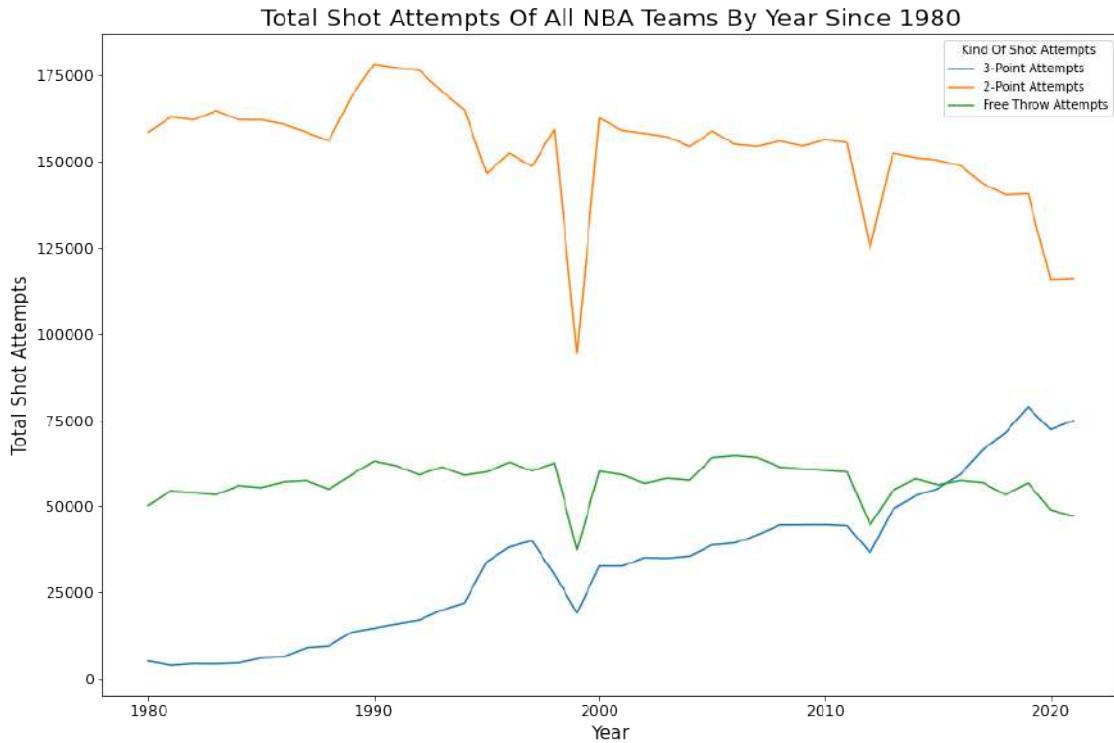
```
[64]: nba_total_df.groupby('Year')['3PA'].sum()
```

```
[64]: Year
1980      5003
1981      3815
1982      4308
1983      4248
1984      4484
1985      5917
1986      6293
1987      8913
1988      9421
1989     13431
1990     14608
1991     15812
1992     16898
1993     19824
1994     21907
1995     33889
1996     38161
1997     39943
1998     30231
1999     19080
2000     32614
2001     32597
```

```
2002    35074
2003    34912
2004    35492
2005    38748
2006    39313
2007    41672
2008    44544
2009    44583
2010    44622
2011    44313
2012    36395
2013    49067
2014    52974
2015    55137
2016    59241
2017    66422
2018    71340
2019    78742
2020    72252
2021    74822
Name: 3PA, dtype: int64
```

```
[65]: plt.figure(figsize = [15,10])
nba_total_df.groupby('Year')['3PA'].sum().plot(label = '3-Point Attempts')
nba_total_df.groupby('Year')['2PA'].sum().plot(label = '2-Point Attempts')
nba_total_df.groupby('Year')['FTA'].sum().plot(label = 'Free Throw Attempts');

plt.title('Total Shot Attempts Of All NBA Teams By Year Since 1980', fontsize = 20)
plt.legend(title = 'Kind Of Shot Attempts')
plt.xlabel('Year', fontsize = 15)
plt.ylabel('Total Shot Attempts', fontsize = 15)
plt.xticks(fontsize = 12.5)
plt.yticks(fontsize = 12.5);
```



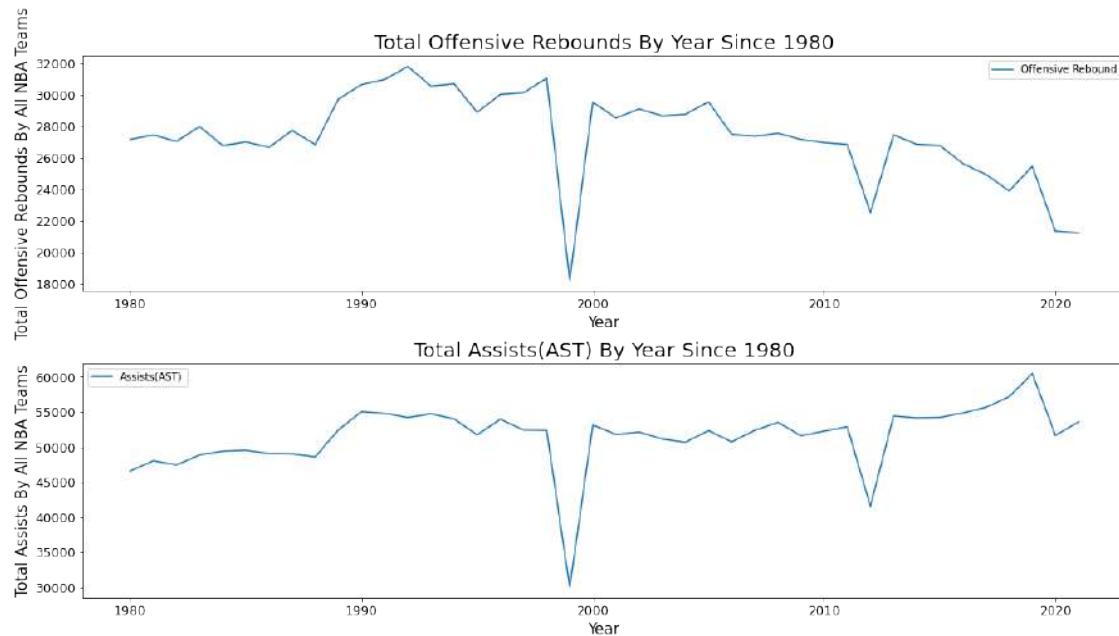
**Question 2 Observation >** In this graph, there are two lines; one represents the total 2-point shot attempts and one represents the total 3-point shot attempts during the regular season. Starting with the 2-point line, we can see that the total number of attempts has remained somewhat steady since 1980, but shows a very slight declining trend. It started off a little above 150,000 shots at the beginning of 1980 before shooting up at the end of the 80s. Going into the 90s it began to drop off and stayed close to 150,000 shot attempts up until 2010. There is a sharp decrease in 1999 before recovering to around 150,000. This sharp decrease and sudden increase happened again at the start of the 2010s era. This sharp decrease in total shot attempts is also telling us that the average total shot attempts must have decreased as well. After the recovery to 150,000, the line shows an obvious downtrend which ended at around 125,000 by 2021. This means that players have been taking fewer shots since around 2015. Moving on to the 3-point line, we can see the dramatic increase in this kind of shot attempt over the years. It started off at 5000 shot attempts at the start of the 1980s, then it increased drastically as time went on. By the end of 2021, the NBA 3-point attempts in a season totaled to about 75,000, which is 15x more than in the '80s when 3-point shots were first recorded! However, similar to the 2-point line we also see sharp declines in 1999 and the beginning of 2010. I am beginning to suspect something happened those years, but let's see how some other stats fare. The free-throw line has remained relatively steady since 1980, staying near the 50k attempts range. There is a slight decrease at the end, but this seems to be a pattern with all 3 lines. Furthermore, we also saw those massive dips in 1999 and at the start of 2010. We should check out some of the other offensive stats to see if this trend carries over.

**3. Is there similar trend for the other offensive stats(Offensive Rebound & Assists)? Analyze them too. >** Note: This question wasn't part of my original analysis, but came up after examining the total shot attempts and the drastic dips from the last graph.

```
[66]: #plt.figure(figsize = [15,8])
fig, ax = plt.subplots(2, 2, figsize = [15,8])
fig.tight_layout(h_pad=5)

plt.subplot(2,1,1)
nba_total_df.groupby('Year')['ORB'].sum().plot(label = 'Offensive Rebound')
plt.title('Total Offensive Rebounds By Year Since 1980', fontsize = 20)
plt.legend()
plt.xlabel('Year', fontsize = 15)
plt.ylabel('Total Offensive Rebounds By All NBA Teams', fontsize = 15)
plt.xticks(fontsize = 12.5)
plt.yticks(fontsize = 12.5);

plt.subplot(2,1,2)
nba_total_df.groupby('Year')['AST'].sum().plot(label = 'Assists(AST)')
plt.title('Total Assists(AST) By Year Since 1980', fontsize = 20)
plt.legend()
plt.xlabel('Year', fontsize = 15)
plt.ylabel('Total Assists By All NBA Teams', fontsize = 15)
plt.xticks(fontsize = 12.5)
plt.yticks(fontsize = 12.5);
```

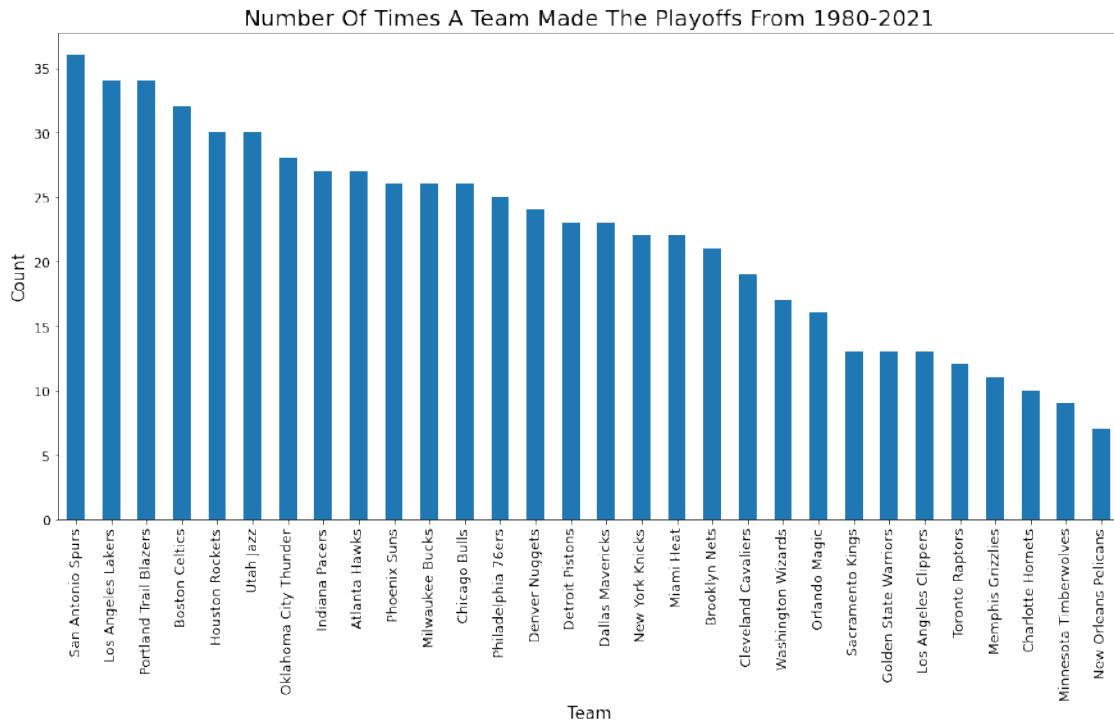


**Question 3 Observation** >After examining the other offensive stats, it seems like the pattern persists. There are sharp and sudden drops in the year axis at the exact same places, which is the same as the previous graphs we've examined. Upon further independent research online, we found that the NBA those drastic drops were due to the [NBA lockouts](#) when players didn't play due to

failed negotiations. Therefore, the season was shortened. However, this doesn't apply to the sudden dip in 2020, which was due to the COVID-19 pandemic. Starting with the offensive rebound (ORB) line plot, we can see that the total number of offensive rebounds in the NBA remained steady at around 27,000 rebounds a year. Going toward the end of the decade and halfway into the '90s decade, the total number of offensive rebounds shot up to over 30-32,000 years, not including the lockout years. Afterward, we see a steady decline over the years, ending with a total of about 21,000 rebounds in 2021. It seems like fewer offensive rebounds are being grabbed. Moving on to the assists (AST) line, we see that the total number of assists in the league remained under 50,000 assists in the '80s. This is the case until near the end of the decade when the total shot up to 55,000 going into the '90s. For the next two decades (except the lockout years), the total number of assists in the league stayed between 50-55k until around 2012, when the total began to increase. It 60,000 total assists in 2019 before dropping again due to the pandemic. It seems that there have been a lot of successful assists made over the last couple of years.

#### 4. How many times did each team make the playoff since 1980?

```
[67]: playoff_true_count = nba_total_df[nba_total_df['Playoff']==True]['Team'].  
       ↵value_counts()  
  
f = plt.figure(figsize = [17,8])  
playoff_true_count.plot(kind = 'bar')  
plt.title('Number Of Times A Team Made The Playoffs From 1980-2021', fontsize =  
          ↵20)  
  
plt.ylabel('Count', fontsize = 15)  
plt.xlabel('Team', fontsize = 15)  
plt.yticks(fontsize = 12.5)  
plt.xticks(fontsize = 12.5);
```



[68]: playoff\_true\_count

|                         |    |
|-------------------------|----|
| [68]: San Antonio Spurs | 36 |
| Los Angeles Lakers      | 34 |
| Portland Trail Blazers  | 34 |
| Boston Celtics          | 32 |
| Houston Rockets         | 30 |
| Utah Jazz               | 30 |
| Oklahoma City Thunder   | 28 |
| Indiana Pacers          | 27 |
| Atlanta Hawks           | 27 |
| Phoenix Suns            | 26 |
| Milwaukee Bucks         | 26 |
| Chicago Bulls           | 26 |
| Philadelphia 76ers      | 25 |
| Denver Nuggets          | 24 |
| Detroit Pistons         | 23 |
| Dallas Mavericks        | 23 |
| New York Knicks         | 22 |
| Miami Heat              | 22 |
| Brooklyn Nets           | 21 |
| Cleveland Cavaliers     | 19 |
| Washington Wizards      | 17 |
| Orlando Magic           | 16 |

```

Sacramento Kings      13
Golden State Warriors 13
Los Angeles Clippers   13
Toronto Raptors       12
Memphis Grizzlies     11
Charlotte Hornets     10
Minnesota Timberwolves 9
New Orleans Pelicans   7
Name: Team, dtype: int64

```

**Question 4 Observation** > From the bar graph above, we see that the San Antonio Spurs have made it to the playoffs the most over the years with a total of 36 appearances. Following the Spurs are the Lakers and Blazers, who are both tied for 2nd place with a total of 34 appearances. Going to the other side of the bar graph, we see that the New Orleans Pelicans have made the least appearances in the playoffs since 1980. They are followed by Timberwolves and Charlotte Hornets with 9 and 10 appearances respectively. Now that we've seen how many times each team made the playoffs, let's see who won the championships over the years!

5. How many teams won the NBA championships since 1980? How many times did each team win?

```
[69]: champion_count = nba_total_df[nba_total_df.Finals_Rk == 'Champion']['Team'].
    ↪value_counts(ascending = False)
labels = list(champion_count.index)
# making a list of team colors
colors = ['#552583', '#CE1141', '#000000', '#007A33', '#1D42BA',
          '#98002E', '#1D428A', '#CE1141', '#006BB6', '#7F9695',
          '#860038', '#B4975A', '#00471B']

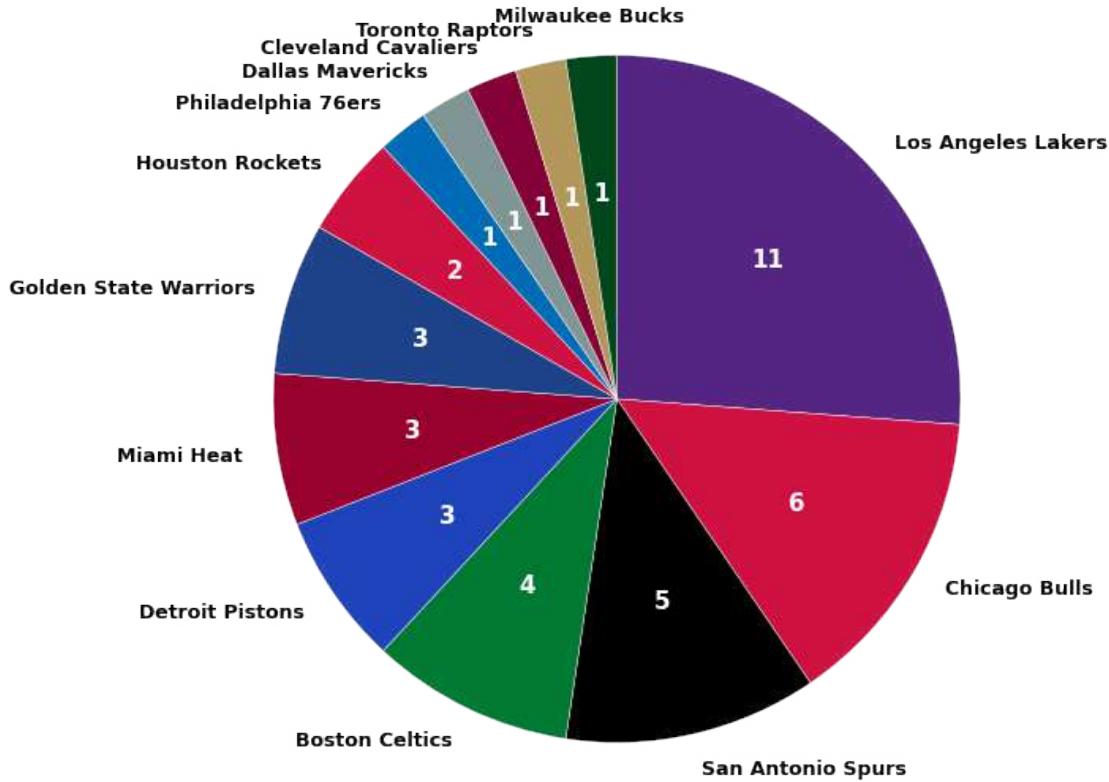
# Plot Pie Chart
fig1, ax1 = plt.subplots(figsize = [10,10])
patches, texts, autotexts = ax1.pie(champion_count, startangle = 90, ↪
    ↪counterclock = False, labels = labels,
   wedgeprops = dict(linewidth= 0.4, edgecolor = 'white'),
   labeldistance=1.1, textprops= dict(fontsize = 12.5, ↪
    ↪weight = 'bold'),
   autopct=lambda p: '{:.0f}'.format((p *  champion_count.
    ↪sum())/ 100), colors = colors)
plt.title('Pie Chart: Distribution Of Teams That Won The NBA Championship since ↪
    ↪1980', fontsize = 16)

# readjust 'Bucks' label because it overlaps with Rapotors
for t in texts:
    if 'Bucks' in t.get_text():
        t.set_horizontalalignment('center')
        t.set_verticalalignment('baseline')

# alter color, wright nad fontsize of auto texts to make more clear
```

```
# Help Cited --
# https://stackoverflow.com/questions/57062322/
↳python-how-to-change-autopct-text-to-white-and-bold-in-a-pie-chart
plt.setp(autotexts, **{'color':'white', 'weight':'extra bold', 'fontsize':15});
```

Pie Chart: Distribution Of Teams That Won The NBA Championship since 1980



**Question 5 Observation** > Here is a pie chart of all the NBA champions since 1980. In this chart, we can see that the Lakers have won a little over a quarter of all NBA championships over the last 40 years with 11. Yay! That's my team! Following them are the Chicago Bulls and San Antonio Spurs with 6 and 5 championships, respectively. Looking at the least number of championships over the last 4 decades, we can see that 5 teams are tied for last; those teams are the 76ers, Mavericks, Cavaliers, Raptors, and Bucks. Lets examine the distribution for the rest of the Finals rankings before we move on.

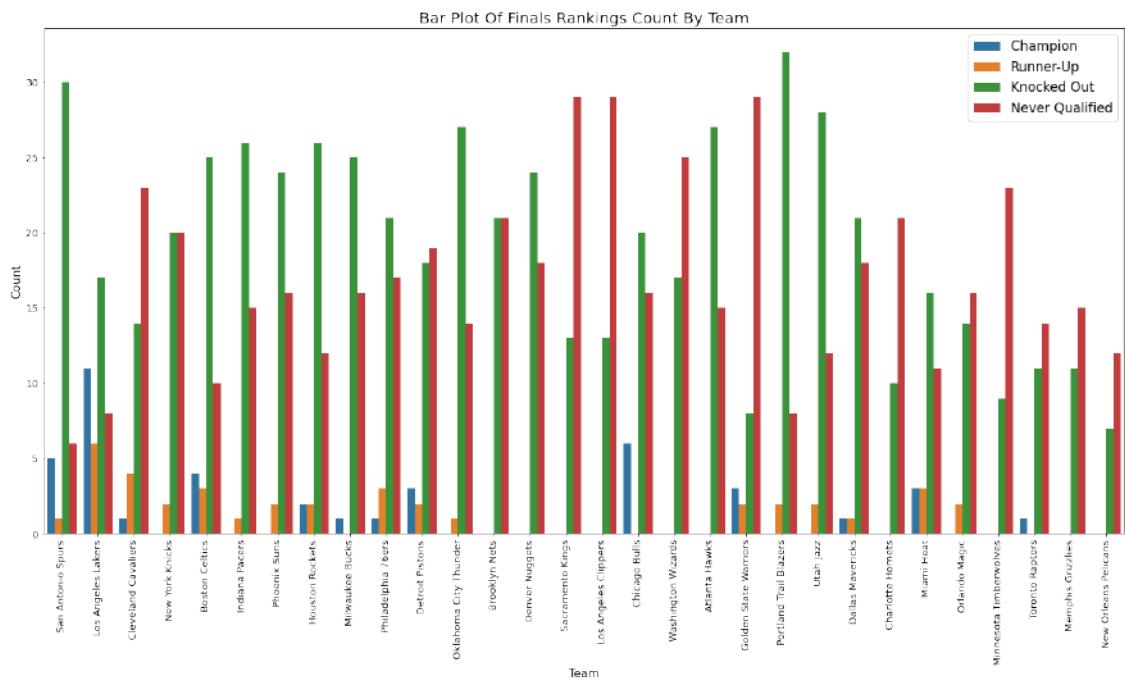
6. Examine the Finals\_Rk column by Team. How many times did each team get knocked out, never qualified, become runner-up or champion?

```
[70]: nba_avg_df['Finals_Rk'].unique()
```

```
[70]: array(['Knocked Out', 'Champion', 'Runner-Up', 'Never Qualified'],
      dtype=object)

[71]: # convert Finals_Rk to category data type
Finals_Rk_ordered = ['Champion', 'Runner-Up', 'Knocked Out', 'Never Qualified']
Rk_ordered_var = pd.api.types.CategoricalDtype(ordered = True, categories = [
    ↪Finals_Rk_ordered])
nba_total_df['Finals_Rk'] = nba_total_df['Finals_Rk'].astype(Rk_ordered_var)

[72]: fig, ax = plt.subplots(figsize = [25,12])
sb.countplot(data = nba_total_df, x = 'Team', hue = 'Finals_Rk')
plt.xticks(rotation = 90, fontsize = 13)
plt.yticks(fontsize = 13)
plt.xlabel('Team', fontsize = 15)
plt.ylabel('Count', fontsize = 15)
plt.title('Bar Plot Of Finals Rankings Count By Team', fontsize = 20)
plt.legend(prop={'size': 17}, loc = 'upper right');
```



**Question 6 Observation >** In a previous pie chart observation, we found that the Lakers have won the most NBA championships since 1940 with a total of 11. Following them are the Bulls and Spurs. Next, we will examine the runner-ups. Once again, the Lakers take first place. If we add up the total championship and runner-up years, we find that the Lakers have been to the Finals a total of 17 times in the past 4 decades. Following the Lakers for 2nd in runner-ups are the Cleveland Cavaliers. Moving on to the next rankings, we find that the Portland Trailblazers have been knocked out of the playoffs the most times in the past 40 years. They are followed by

Spurs. Finally, the Warriors, Kings, and Clippers fail to qualify for the playoff the most. On the right side, we can see that there is a far fewer count for some of the teams if we add all the bar charts up. That's because these are newer teams. Out of the newer teams (last 4), we can see that only one championship has been won between all of them, which belongs to the Toronto Raptors.

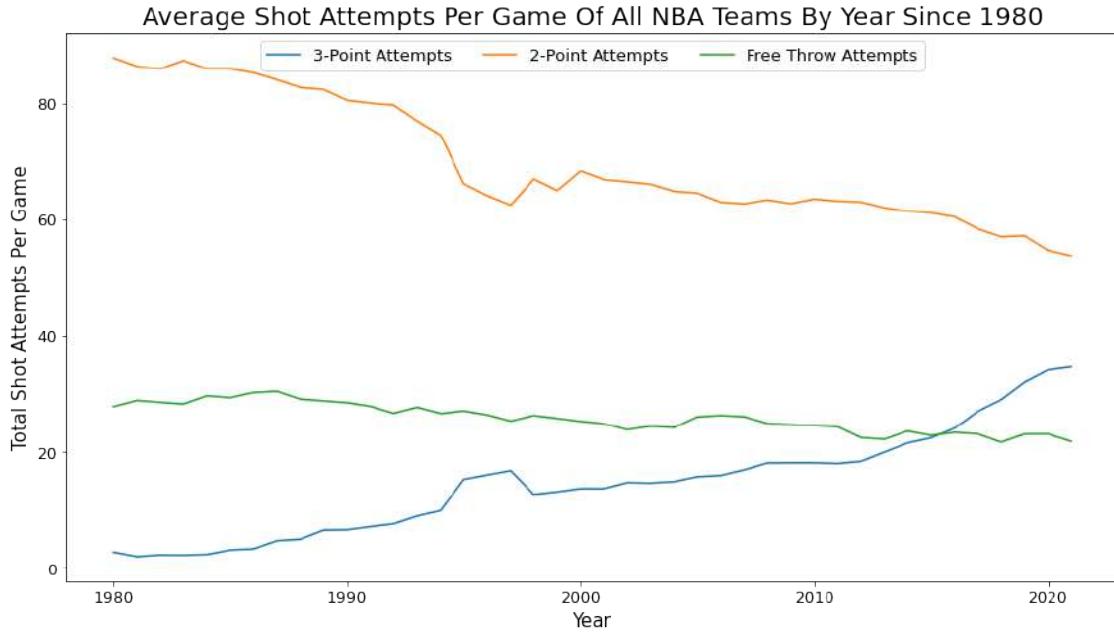
[ ]:

#### 1.4.2 nba\_avg\_df

Now that we are done examining the nba\_total\_df, I want to examine the averages of each team in every season since 1980. Luckily, we do not have to do the math since we web-scraped the information from Basketball-Reference earlier. While the total stats were enjoyable to analyze, examining the averages could give us a better look into each team's performance. Not to mention, we also have some newer columns to work with! Furthermore, the nba\_avg\_df has also been adjusted and accounts for the lockouts and COVID period by only calculating the averages of only the games played that season. Therefore, we shouldn't be seeing any sudden, drastic dips now. Let's check it out before we move on!

```
[73]: plt.figure(figsize = [15,8])
nba_avg_df.groupby('Year')['3PA'].mean().plot(label = '3-Point Attempts')
nba_avg_df.groupby('Year')['2PA'].mean().plot(label = '2-Point Attempts')
nba_avg_df.groupby('Year')['FTA'].mean().plot(label = 'Free Throw Attempts');

plt.title('Average Shot Attempts Per Game Of All NBA Teams By Year Since 1980', fontweight = 'bold', fontsize = 20)
plt.legend(title = 'Kind Of Shot Attempts')
plt.xlabel('Year', fontsize = 15)
plt.ylabel('Total Shot Attempts Per Game', fontsize = 15)
plt.xticks(fontsize = 12.5)
plt.yticks(fontsize = 12.5)
plt.legend(loc='upper center', ncol = 3, prop = {'size': 13});
```



Now, we don't see the sudden drastic drops in the years 1999, 2011, and 2019. Let's analyze it. Starting with the 2-point field goal attempts(2PA), we can see that the average attempts per game in the regular season have been declining since 1980. It took a big dip near the start of the '90s and never recovered, hovering around 60 shot attempts to this day. Next, we will move onto the free-throw attempts(FTA) line where we see a similar downtrend, but no drastic dips. It started off around 30 FTA in the '80s but slowly declined to around 20 over the years. Lastly, we will examine the 3-point attempts (3PA) line plot. In the '80s, we can see that NBA teams are barely taking any 3-pint attempts, but it has grown dramatically since then. As of 2021, we can see that NBA teams are taking an average of 30 3-point attempts a game!

## 1.5 Questions - nba\_avg\_df

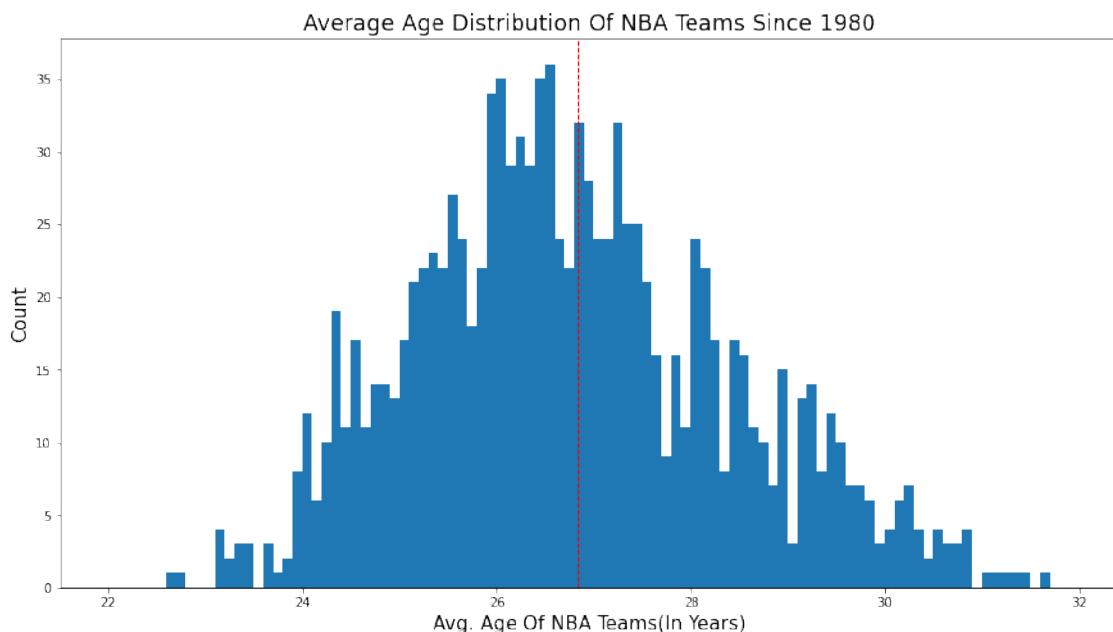
**nba\_avg\_df** 1. What is the distribution like for age for all NBA Teams for all years like? 2. What are the winning percantages for each NBA team throughout their history since 1980? 3. What is the distribution for the Margin of Victory(MOV) like? 4. Choose some numeric variables you are interested in and analyze the pairwise correlation between them. 5. Plot the two strongest negative and positive correlations from the heatmap and identify them by their Finals rankings and era played. What are some of the characteristics of champions, runner-ups, and teams that got knocked out or never qualified for the playoffs? 6. What is the True Shooting %(TS%) like for each era and Finals ranking? 7. What is the average 3PAr based on Finals ranking and era? 8.What is the age distribution like for each Finals rankings? Do the champions and runner-ups tend to be older or younger? 9. Historically, did the champions and runner-ups average more total assists per game than the teams that were knocked out or never qualified for the playoff? Use the latters' mean as a benchmark. 10. Seperating the data by Conference and Finals Ranking, what is the average regular season MOV

for each era?

1. What is the distribution like for age for all NBA Teams for all years like?

```
[74]: fig, ax = plt.subplots(figsize = [15,8])
bins = np.arange(22,32, 0.1)

# Plot Age histogram
nba_avg_df.Age.hist(bins = bins)
plt.grid(False)
plt.xlabel('Avg. Age Of NBA Teams(In Years)', fontsize = 15)
plt.ylabel('Count', fontsize = 15)
plt.title('Average Age Distribution Of NBA Teams Since 1980', fontsize = 17.5)
plt.axvline(nba_avg_df.Age.mean(), color='red', linestyle='dashed', linewidth=1);
```



**Question 1 Observation** > Although the graph is skewed to the right a little, the distribution seems pretty normal. We can see that the highest counts for average team age since 1980 are between the age of 26-27. The graph is also telling us the mean is roughly around 27 years of age.

2. What are the winning percentages for each NBA team throughout their history since 1980?

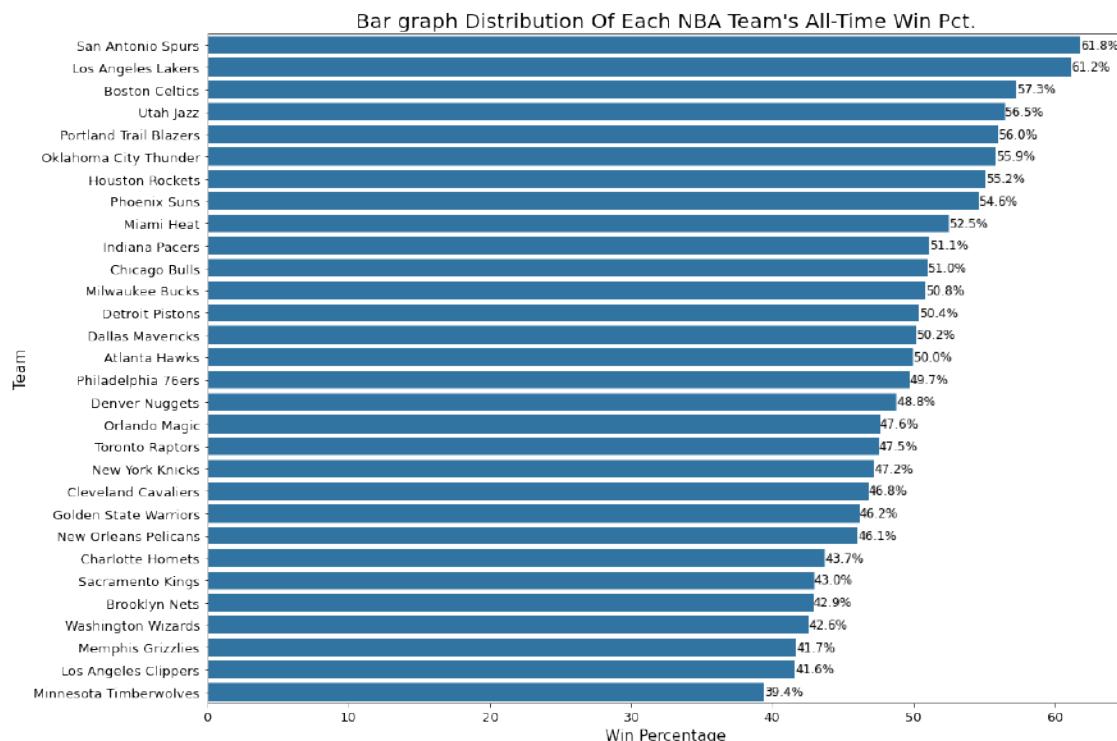
```
[75]: nba_avg_df['Win%'] = (nba_avg_df['Wins']/(nba_avg_df['Losses'] + nba_avg_df['Wins'])) * 100
alltime_win_order = pd.DataFrame(nba_avg_df.groupby('Team')['Win%'].mean().sort_values(ascending = False)).reset_index()
```

```

fig = plt.subplots(figsize = [16,12])
base_color = sb.color_palette()[0]
ax = sb.barplot(data = alltime_win_order, y = 'Team', x = 'Win%', color = base_color)
plt.title('Bar graph Distribution Of Each NBA Team\'s All-Time Win Pct.', fontsize = 20)
plt.ylabel('Team', fontsize = 15)
plt.xlabel('Win Percentage', fontsize = 15)
plt.yticks(fontsize = 13)
plt.xticks(fontsize = 13)

# print percentages next to bars
for i in ax.containers:
    ax.bar_label(i, fontsize = 12, fmt='%.1f%%');

```



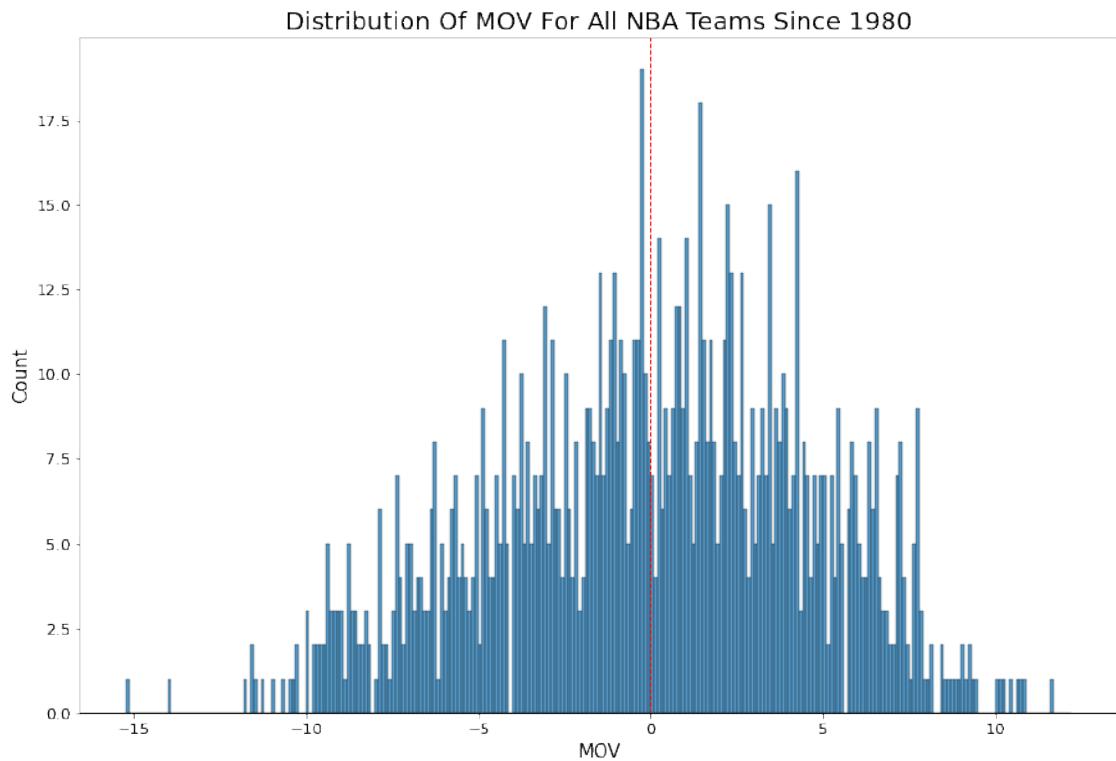
**Question 2 Observation** > In this chart, we can see the all-time winning percentage for every single team. The most winningest team in the NBA since 1980 is the San Antonio Spurs who won 61.8% of their regular-season games. Coming in 2nd are the Lakers with 61.2%. However, I would have never guessed that the Minnesota Timberwolves would have the worst winning percentage with 39.4%.

### 3. What is the distribution for the Margin of Victory(MOV) like?

```
[76]: fig, ax = plt.subplots(figsize=(15, 10))
bins = np.arange(nba_avg_df.MOV.min(), nba_avg_df.MOV.max(), 0.1)

#plot MOV graph
sb.histplot(nba_avg_df.MOV, kde = False, bins = bins, ax = ax)
plt.grid(False)
plt.axvline(nba_avg_df.MOV.mean(), color='red', linestyle='dashed', linewidth=1)

# texts-related
plt.title('Distribution Of MOV For All NBA Teams Since 1980', fontsize = 20)
plt.xlabel('MOV', fontsize = 15)
plt.ylabel('Count', fontsize = 15)
plt.xticks(fontsize = 12.5)
plt.yticks(fontsize = 12.5);
```



**Question 3 Observation** > In the graph above, we see the distribution of MOV. The graph is somewhat normally distributed, but seems to be skewing to the left a little.

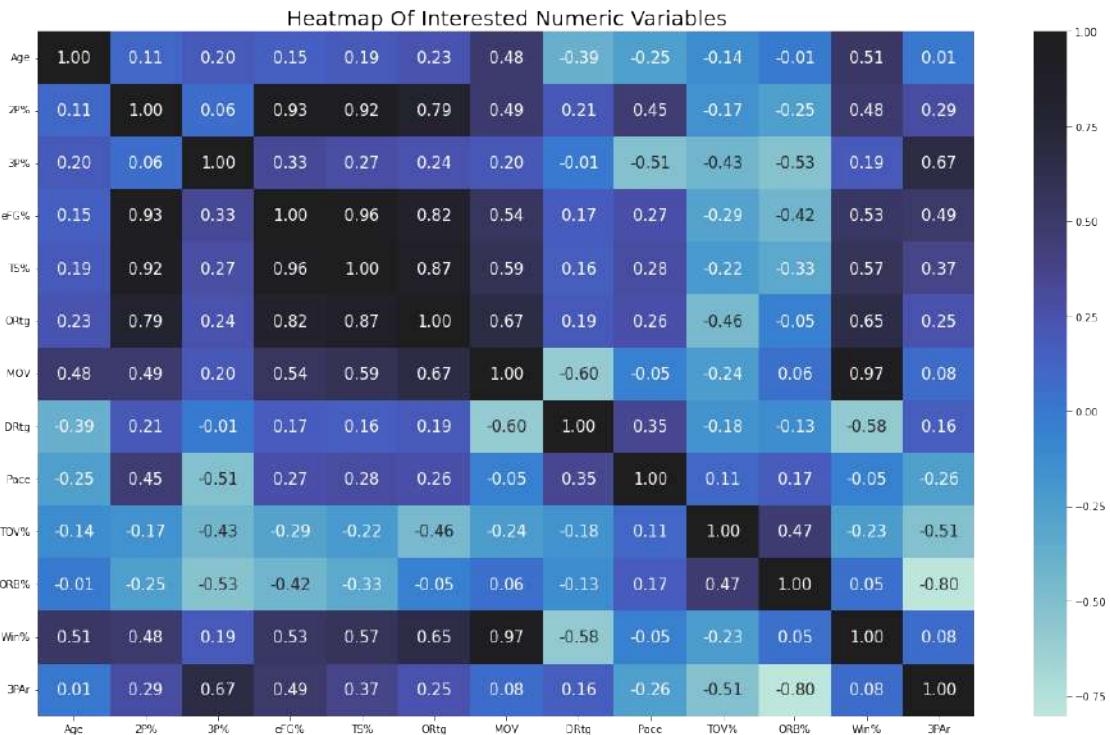
**4. Choose some numeric variables you are interested in and analyze the pairwise correlation between them.** > For this observation, I have chosen the stats that I am most interested in analyzing. As you know, I will be focused primarily on offensive stats. However, I have also included some additional non-offensive stats. My goal from this point on is to regular season offensive stats for every final rankings. Down below, I will be explaining the meaning of the stats, starting with 'eFG%'. You can also go to the [Basketball-Reference Glossary](#) to look up their meanings if

you prefer. 1.eFG%: Effective Field Goal Percentage; the formula is  $(\text{FG} + 0.5 * \text{3P}) / \text{FGA}$ . This statistic adjusts for the fact that a 3-point field goal is worth one more point than a 2-point field goal. 2.TS%: True Shooting Percentage; the formula is  $\text{PTS} / (2 * \text{TSA})$ . True shooting percentage is a measure of shooting efficiency that takes into account field goals, 3-point field goals, and free throws. 3.MOV: Margin of Victory 4.ORtg:Offensive Rating; for players, it is points produced per 100 possessions, while for teams it is points scored per 100 possessions. 5.DRtg: Defensive Rating; for players and teams, it is points allowed per 100 possessions. 6.Pace: The pace factor is an estimate of the number of possessions per 48 minutes by a team. 7.TOV%: Turnover percentage is an estimate of turnovers per 100 plays. 8.ORB%: Offensive rebound percentage is an estimate of the percentage of available offensive rebounds a player grabbed while he was on the floor. 9.3PAr: Percentage of FG Attempts from 3-Point Range.

Please remember for this question that correlation does not imply causation.

```
[77]: num_vars = ['Age', '2P%', '3P%', 'eFG%', 'TS%', 'ORtg',
               'MOV', 'DRtg', 'Pace', 'TOV%', 'ORB%', 'Win%', '3PAr']
```

```
[78]: # plot heatmap of numeric variables
fig, ax = plt.subplots(figsize = [20,12])
sb.heatmap(nba_avg_df[num_vars].corr(), annot = True, fmt = '0.2f',
            center = 1, annot_kws = {'size': 15})
plt.yticks(rotation = 360)
plt.title('Heatmap Of Interested Numeric Variables', fontsize = 20);
```



**Question 4 Observation** > There are many correlations here to go over, so I will mainly be

pointing out the ones that I am interested in. First, let's start off with the surprises; the first surprise that I've found in this heat map is the relationship between DRtg & MOV, which happens to have a strong negative correlation of -0.6. If DRtg goes down, MOV goes up. As a result because of the strong correlation between Win% and MOV, we find that the relationship between DRtg and Win% is also moderately strong in correlation. One very strong correlation displayed is the relationship between 3PAr and ORB%. Another negative correlation that I found interesting was the 3P% and its relationship with 2 other variables: Pace and ORB%. It seems like when the 3P% of a team goes up, they tend to grab fewer offensive rebounds and possess the ball less on the court. In fact, if we examine all of the shooting-related percentages (3P%, eFG%, and TS%) and their relationships with ORB%, we find that they all have moderately negative correlations. One negative correlation that I thought would be stronger is the one between TOV% and MOV, but it turns out to be weak. Next, we will examine positive correlations. Starting with the Win% and MOV relationship, we find that they have the strongest positive correlation with 0.97! This makes sense since an average negative MOV would mean that a team is losing most of their games. In fact, if we just analyze the MOV row, we'll find that MOV has a moderately strong relationship with all the shooting-related stats except for 3P%. Next, we find that all the shooting percentages -mainly, 2P%, eFG%, and TS%- are heavily correlated to one another, which comes as no surprise. However, one surprise that I found was the relationship between Win% and Age, which displays a moderately, strong positive correlation. One more positive relationship I found interesting was the one between DRtg and Pace, which displays a moderate correlation. This is telling me that if the pace increases, the DRtg also increases somewhat in the same direction.

**5. Plot the two strong negative and positive correlations from the heatmap and identify them by their Finals rankings and era played. What are some of the characteristics of champions, runner-ups, and teams that got knocked out or never qualified for the playoffs?**

```
[79]: year_list = list(nba_avg_df.Year.unique())

#creating the era column
for i, row in nba_avg_df.iterrows():
    if row.Year in year_list[:10]:
        nba_avg_df.at[i, 'Era'] = "1980s"
    if row.Year in year_list[10:20]:
        nba_avg_df.at[i, 'Era'] = "1990s"
    if row.Year in year_list[20:30]:
        nba_avg_df.at[i, 'Era'] = "2000s"
    if row.Year in year_list[30:]:
        nba_avg_df.at[i, 'Era'] = "2010 & Beyond"

nba_avg_df.head()
```

|   | Team                  | Age  | Wins | Losses | Year | Playoff | FG   | FGA  | FG%  | \ |
|---|-----------------------|------|------|--------|------|---------|------|------|------|---|
| 0 | Boston Celtics        | 27.3 | 61   | 21     | 1980 | True    | 44.1 | 90.1 | 49.0 |   |
| 1 | Los Angeles Lakers    | 26.2 | 60   | 22     | 1980 | True    | 47.5 | 89.9 | 52.9 |   |
| 2 | Oklahoma City Thunder | 27.0 | 56   | 26     | 1980 | True    | 43.3 | 92.3 | 47.0 |   |
| 3 | Philadelphia 76ers    | 27.0 | 59   | 23     | 1980 | True    | 43.0 | 87.3 | 49.2 |   |
| 4 | Milwaukee Bucks       | 25.3 | 49   | 33     | 1980 | True    | 44.9 | 92.1 | 48.8 |   |

|   | 2P   | 2PA  | 2P%  | 3P  | 3PA | 3P%  | 3PAr | eFG% | FT   | FTA  | FT%  | TS%  | AST  | \ |
|---|------|------|------|-----|-----|------|------|------|------|------|------|------|------|---|
| 0 | 42.1 | 84.9 | 49.6 | 2.0 | 5.1 | 38.4 | 5.7  | 50.1 | 23.3 | 29.9 | 77.9 | 55.0 | 26.8 |   |
| 1 | 47.3 | 88.6 | 53.4 | 0.2 | 1.2 | 20.0 | 1.4  | 53.0 | 19.8 | 25.5 | 77.5 | 56.9 | 29.4 |   |
| 2 | 42.6 | 90.0 | 47.4 | 0.7 | 2.3 | 31.2 | 2.5  | 47.4 | 21.1 | 27.5 | 76.8 | 52.0 | 24.9 |   |
| 3 | 42.6 | 85.7 | 49.7 | 0.3 | 1.5 | 21.6 | 1.7  | 49.4 | 22.9 | 29.6 | 77.2 | 54.4 | 27.1 |   |
| 4 | 44.3 | 90.2 | 49.1 | 0.6 | 1.9 | 32.3 | 2.1  | 49.1 | 19.6 | 25.6 | 76.4 | 53.2 | 27.8 |   |

|   | ORB  | TRB  | PTS   | MOV  | ORtg  | DRtg  | Pace  | TOV% | ORB% | Finals_Rk   | \ |
|---|------|------|-------|------|-------|-------|-------|------|------|-------------|---|
| 0 | 15.0 | 44.9 | 113.5 | 7.79 | 109.4 | 101.9 | 102.6 | 15.4 | 34.8 | Knocked Out |   |
| 1 | 13.2 | 45.6 | 115.1 | 5.90 | 109.5 | 103.9 | 104.1 | 16.5 | 32.6 | Champion    |   |
| 2 | 16.8 | 47.9 | 108.5 | 4.66 | 105.8 | 101.2 | 101.8 | 14.9 | 36.4 | Knocked Out |   |
| 3 | 14.5 | 46.6 | 109.1 | 4.22 | 105.0 | 101.0 | 103.0 | 17.2 | 33.5 | Runner-Up   |   |
| 4 | 15.2 | 44.4 | 110.1 | 3.94 | 106.8 | 102.9 | 102.4 | 15.0 | 35.2 | Knocked Out |   |

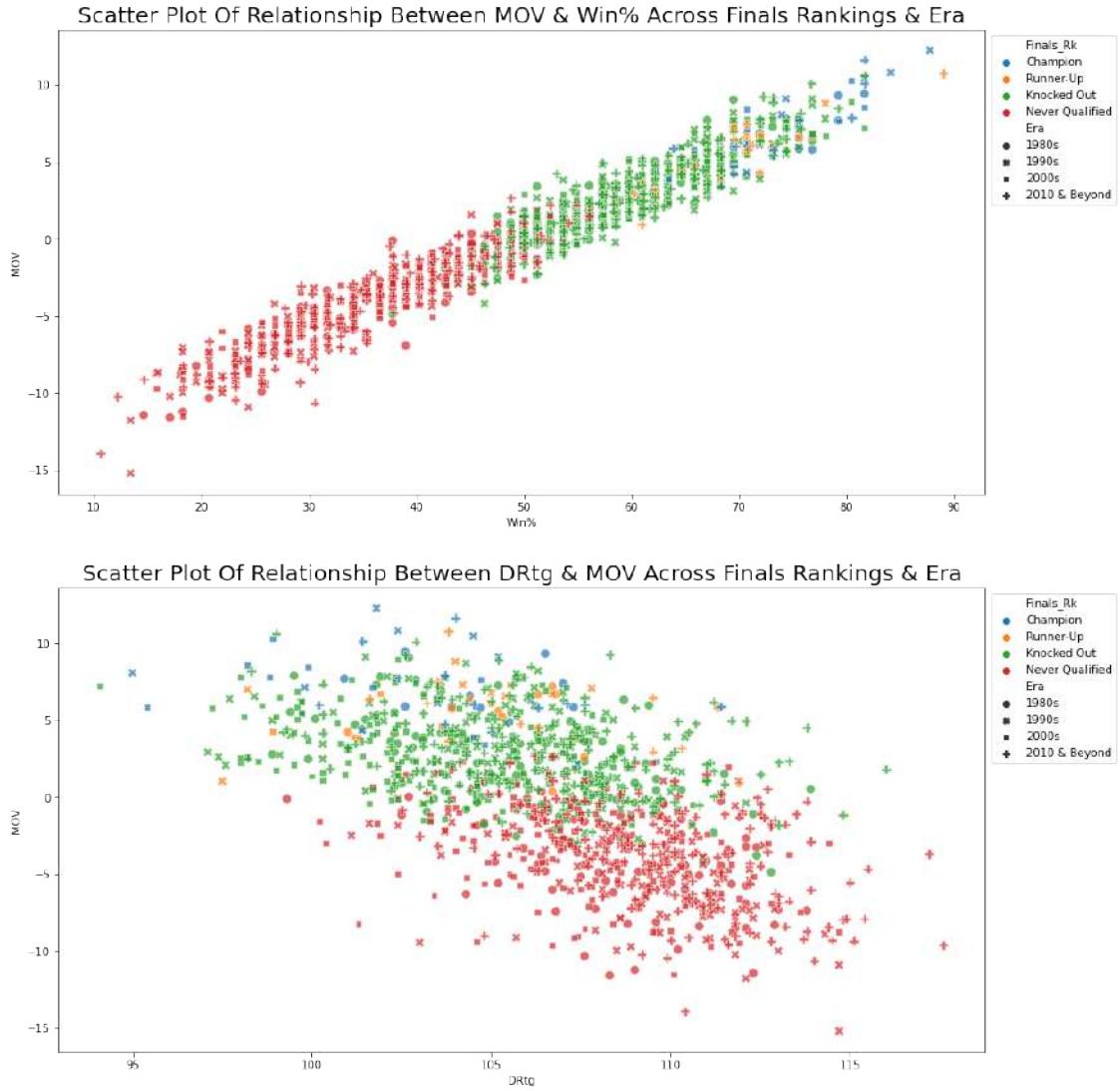
|   | Win%      | Era   |
|---|-----------|-------|
| 0 | 74.390244 | 1980s |
| 1 | 73.170732 | 1980s |
| 2 | 68.292683 | 1980s |
| 3 | 71.951220 | 1980s |
| 4 | 59.756098 | 1980s |

```
[80]: # convert Finals_Rk column to category
nba_avg_df['Finals_Rk'] = nba_avg_df['Finals_Rk'].astype(Rk_ordered_var)

fig, ax = plt.subplots(figsize = [15,17])

plt.subplot(2,1,1)
sb.scatterplot(data = nba_avg_df, x = 'Win%', y='MOV',
                hue = 'Finals_Rk', style ='Era', s =70, alpha = 0.7)
plt.title('Scatter Plot Of Relationship Between MOV & Win% Across Finals\u2192Rankings & Era',
          fontsize = 20)
plt.legend(bbox_to_anchor=(1.0, 1.0), loc='upper left');

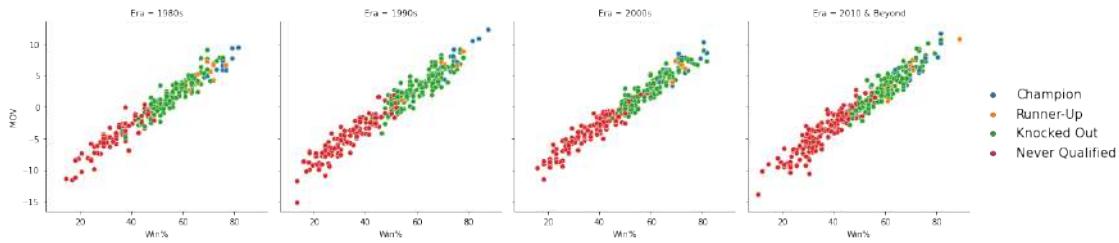
plt.subplot(2,1,2)
sb.scatterplot(data = nba_avg_df, x = 'DRtg', y='MOV',
                hue = 'Finals_Rk', style ='Era', s =70, alpha = 0.7)
plt.title('Scatter Plot Of Relationship Between DRtg & MOV Across Finals\u2192Rankings & Era',
          fontsize = 20)
plt.legend(bbox_to_anchor=(1.0, 1.0), loc='upper left');
```



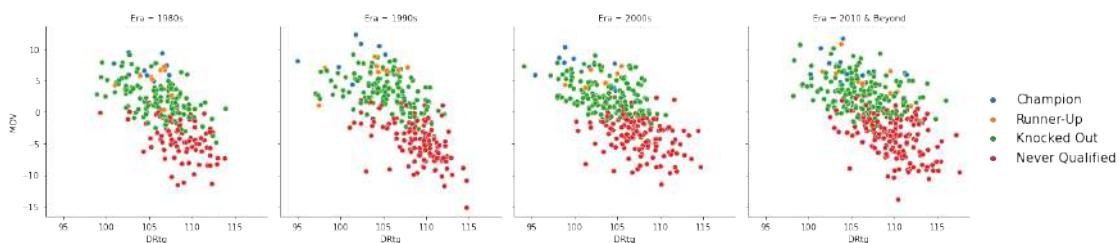
**Question 5 Observation** > Starting with the first scatterplot(MOV & Win%), we see that all the champions have over 60% Win% during the regular season. This statement is also true for the runner-ups, but the champions have the higher minimum, which seems to be about 65%. Furthermore, it seems like champions also have the higher minimum for MOV as well with around 2, while runner-ups' minimum MOV is 0. Historically, it seems like teams need to at least win at 60% of their games with a 0-point margin of victory in order to reach the finals. In the upper-right corner, we see two teams who achieved almost a 90% winning percentage with at least a 10-point MOV. One is from the 1990s era and the other from the 2010s & Beyond, but only the '90s was a champion while the other was a runner-up. In the bottom-right corner, we see two teams that have only about 10% of their games. What's staggering about it is their MOV, which is close to -15 which means they are losing on average 15 points per game! The one with the lowest MOV, but higher Win% out of the two comes from the 1990s era. The other one is from the 2010s era. Moving onto the second scatterplot, we can see that the majority of the champions' defensive ratings (DRtg) are packed between 100 and 110. Here we can clearly see that those with the higher defensive ratings

are more likely to be unqualified for the playoffs. The only champion that has a defensive rating higher than 110 comes from the 2010s & Beyond era. If we look below 0 on the MOV axis, we find that almost all of the teams were either knocked out or never qualified for the playoffs. Here we can see that they all have one thing in common a high defensive rating. Some of these may be a little hard to see. Therefore, I created a FacetGrid for a better viewing experience.

```
[81]: # replot first graph
g = sb.FacetGrid(nba_avg_df, col="Era", height = 4)
g.map_dataframe(sb.scatterplot, x="Win%", y="MOV", hue = "Finals_Rk")
g.add_legend(fontsize = 15);
```

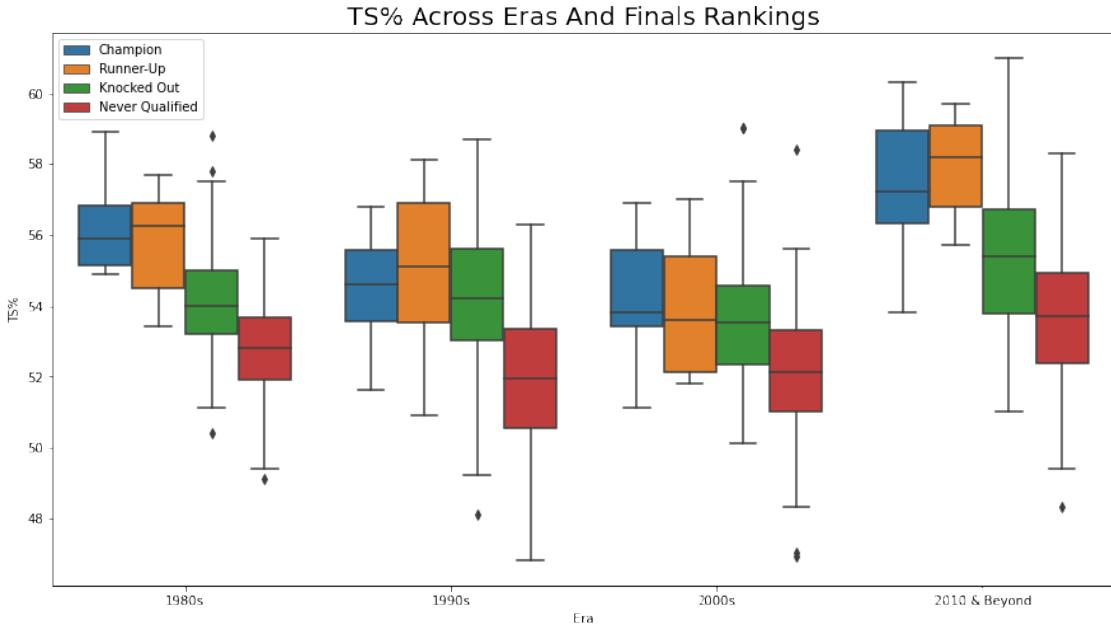


```
[82]: # replot first graph
g = sb.FacetGrid(nba_avg_df, col="Era", height = 4)
g.map_dataframe(sb.scatterplot, x="DRtg", y="MOV", hue = "Finals_Rk")
g.add_legend(fontsize = 15);
```



## 6. What is the True Shooting %(TS%) like for each era and Finals ranking?

```
[83]: fig = plt.figure(figsize = [15, 8])
sb.boxplot(data = nba_avg_df, y = 'TS%', x='Era', hue = 'Finals_Rk')
plt.title('TS% Across Eras And Finals Rankings', fontsize = 20)
plt.legend(loc="upper left");
```



**Question 6 Observation** > Starting with the 1980s era on the right, we see that the runner-ups have the highest TS% median during the regular season with a little over 56% while the Champions have the second highest. However, we also notice that the Champions have the shortest range from their minimum to the 25% quartile and their minimum is higher than the runner-ups' 25% quartile. Next, we find that those who have been knocked out in the playoffs have the 3rd highest TS% median. Furthermore, we also see that the median of the 'Knocked Out' group is higher than the 75% quartile of those who didn't qualify. This pattern between the 'Knocked Out' and 'Never Qualified' groups seems to persist in every era. Keep in mind that there is a lot more data in the last two. For the next era (the 1990s), the patterns for each ranking are very similar to the last era, except the champion's range between minimum and the 25% quartile is much longer than the previous. One thing to note is that the '90s era seems to have a lower TS% than the '80s. For the 2000s era, we finally see that the champions come out on top for the highest TS% median with about 54%. As we move to the right, more on the rankings, we see the median get lower each time. Lastly, for the 2010s & Beyond era, we see that the patterns revert back with the runner-ups having the higher TS% median. What's notable about this one is how much higher all of the medians are than all the previous ones. The runner-ups for this era have a 58% TS% median, while the champions have about 57%.

**7. What is the average 3PAr based on Finals ranking and era?** > A quick reminder that 3-Point Attempt Rate(3PAr) is the percentage of FG attempts from 3-point range.

```
[84]: # plotting heat map
fig, ax = plt.subplots(figsize = [20,12])

c_means = nba_avg_df.groupby(['Finals_Rk', 'Era'])['3PAr'].mean()
    .reset_index(name = 'Avg. 3PAr')
c_means = c_means.pivot(index = 'Finals_Rk', columns = 'Era',
```

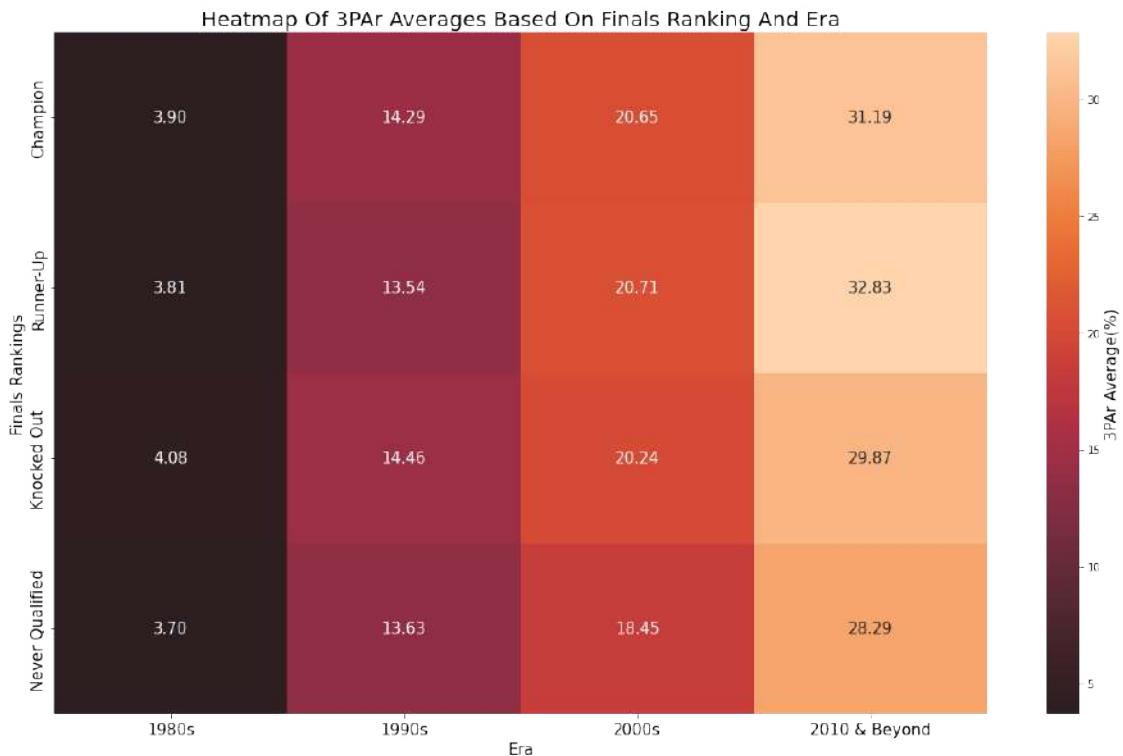
```

values = 'Avg. 3Par')

sb.heatmap(c_means, annot = True, fmt = '.2f', center = 1,
           annot_kws = {'size': 15}, cbar_kws={'label': '3Par Average(%)'})
plt.title('Heatmap Of 3Par Averages Based On Finals Ranking And Era', fontsize=20)
plt.xticks(fontsize = 15)
plt.yticks(fontsize = 15)
plt.xlabel('Era', fontsize = 15)
plt.ylabel('Finals Rankings', fontsize = 15)

# set color bar font size
# help: https://stackoverflow.com/questions/48586738/
#       seaborn-heatmap-colorbar-label-font-size
ax.figure.axes[-1].yaxis.label.set_size(15);

```

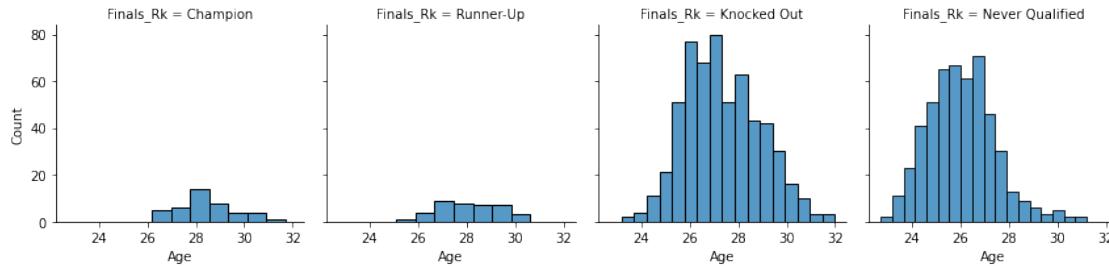


**Question 7 Observation** >From this heatmap that we've just plotted, we can tell that the 3-Point attempt rate increases after each era, no matter the Finals ranking. This means that players are taking more and more 3-point shots on attempts in the regular season. Earlier, we examined the increases in total 3-point attempts in a year and average 3-point attempts in a game over the years on separate line plots. This heatmap gives us a sense of how drastic an increase is by turning them into percentages. We can really get a sense of how the offensive game in the NBA has changed.

Using the ‘Champion’ row as an example, we see that only 3.9% of all shot attempts were from the 3-point line in the 1980s era. In the next era, the percentage nearly quadrupled to 13.54%; by the 2010s & Beyond era, we see that 31.19% of all shot attempts were taken from the 3-point line. This is the general trend for all of the Finals rankings.

**8. What is the age distribution like for each Finals rankings? Do the champions and runner-ups tend to be older or younger?**

```
[85]: g = sb.FacetGrid(data = nba_avg_df, col="Finals_Rk")
g.map(sb.histplot, 'Age');
```



```
[86]: nba_avg_df.Age.mean()
```

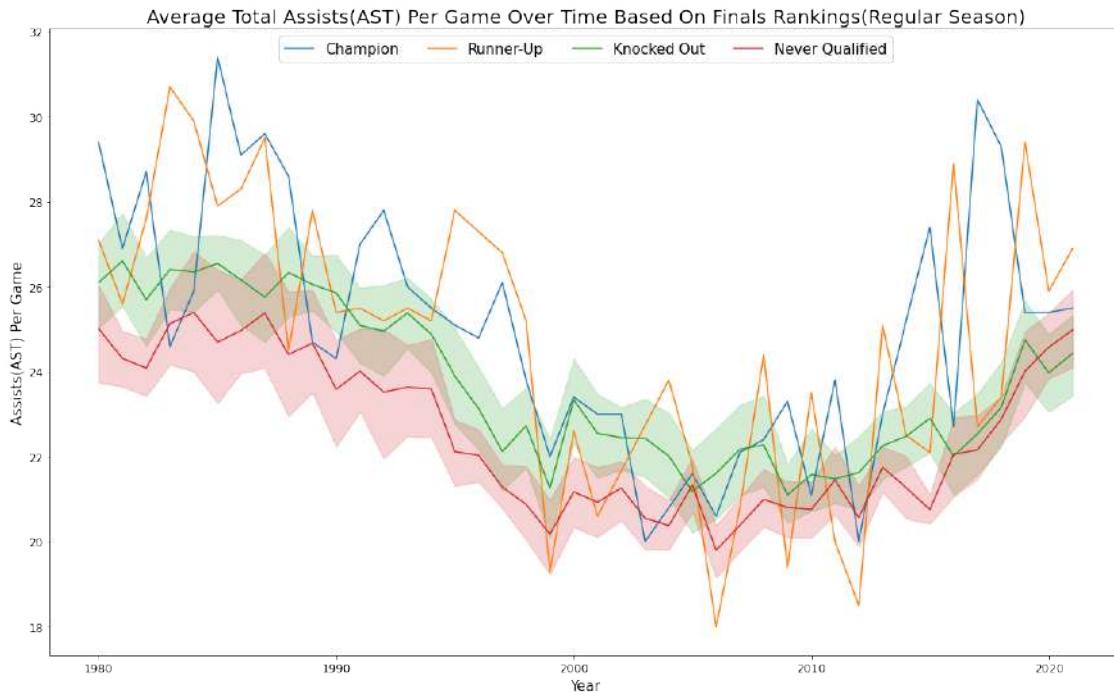
```
[86]: 26.84020618556701
```

**Question 8 Observation** > Although we have a smaller sample size that fits the criteria for Champion and Runner-up, we find that the teams who won NBA championships are on average older, while the runner-ups tend to be a little younger. In the champion histogram on the far-left, most teams are around 28 years of age. In the runner-up histogram, we see a majority of them are between 27-30. Furthermore, we don’t see any teams where the average age is 25 or below in either the champion or runner-up. However, when we move onto teams that have been knocked out or never qualified, we find all of the teams where the average age is 25 or younger.

**9. Historically, did the champions and runner-ups average more total assists per game than the teams that were knocked out or never qualified for the playoff? Use the latters’ mean as a benchmark.**

```
[87]: fig, ax = plt.subplots(figsize = [20,12])

sb.lineplot(data=nba_avg_df, x="Year", y="AST", hue="Finals_Rk")
plt.title('Average Total Assists(AST) Per Game Over Time Based On FinalsRankings(Regular Season)',
          fontsize = 20)
plt.xticks(fontsize = 12)
plt.yticks(fontsize = 12)
plt.xlabel('Year', fontsize = 15)
plt.ylabel('Assists(AST) Per Game', fontsize = 15)
plt.legend(loc = 'upper center', ncol = 4, prop = {'size': 15});
```



**Question 9 Observation** > From the line plot above, we can see that champions and runner-ups have averaged more assists per game during the regular season. Sometimes they both averaged the league's top 2 total assists per game, as seen in the mid-1980s. However, there are some cases when either one would average the lowest assists the league has seen in the mid-2000s and early 2010s. This seems to be telling me that those who average more assists during the regular season will have a higher likelihood of reaching the NBA Finals.

**10. Separating the data by Conference and Finals Ranking, what is the average regular season MOV for each era?** > For this final question, I will be creating one more column, Conference(Conf.). This column will have two values - “West” and “East”- which are based on their home arena location. You can google “NBA Western Conference Teams” to view the list of NBA teams and their respective locations.

```
[88]: # create list of western conference team
west_team = ['Los Angeles Lakers', 'Oklahoma City Thunder', 'Phoenix Suns',
             'Sacramento Kings', 'Houston Rockets', 'San Antonio Spurs',
             'Portland Trail Blazers', 'Brooklyn Nets', 'New York Knicks',
             'Los Angeles Clippers', 'Denver Nuggets', 'Golden State Warriors',
             'Utah Jazz', 'Dallas Mavericks', 'Minnesota Timberwolves',
             'Memphis Grizzlies', 'New Orleans Pelicans']

# assign the team's respective conference
for i, row in nba_avg_df.iterrows():
    if row.Team in west_team:
        nba_avg_df.at[i, 'Conf.'] = 'West'
```

```

    else:
        nba_avg_df.at[i, 'Conf.'] = 'East'

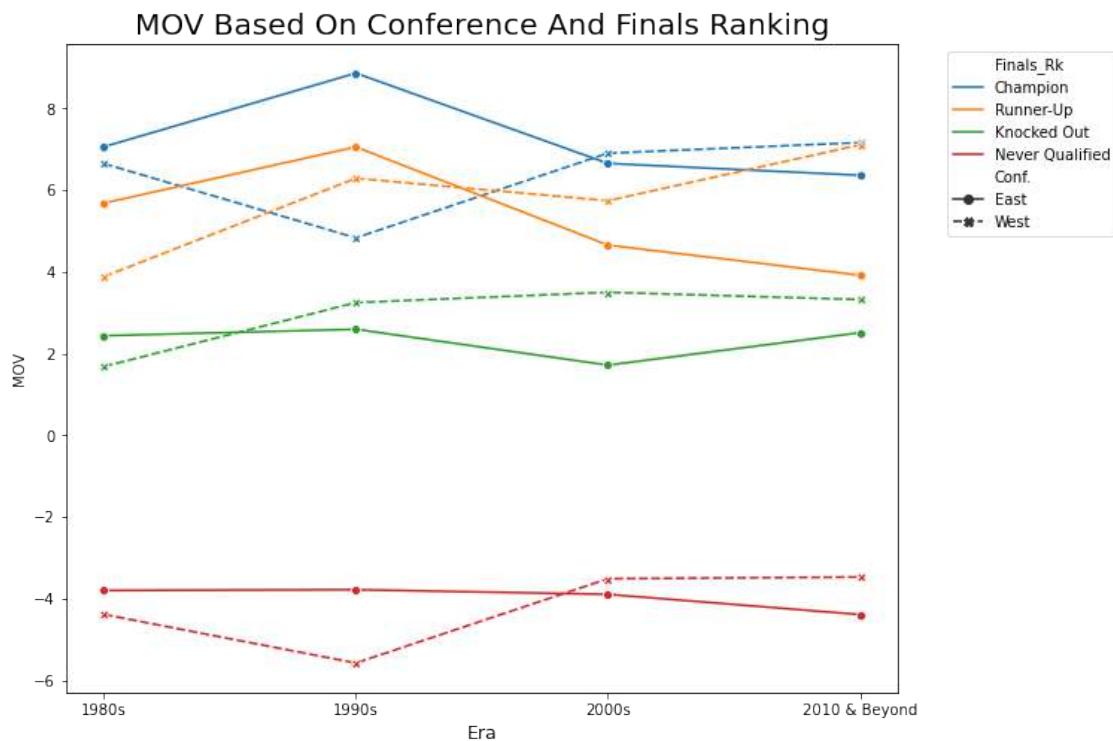
```

```
[89]: nba_avg_df[nba_avg_df['Finals_Rk'] == 'Champion']['Conf.'].value_counts()
```

```
[89]: West      22
      East     20
      Name: Conf., dtype: int64
```

```
[90]: plt.figure(figsize = [10,8])

sb.lineplot(data=nba_avg_df, x="Era", y="MOV", hue="Finals_Rk",
            style="Conf.", err_style= None, markers=True)
plt.xlabel('Era', fontsize = 12.5)
plt.ylabel('MOV')
plt.title('MOV Based On Conference And Finals Ranking',
          fontsize = 20)
plt.legend(bbox_to_anchor=(1.05, 1.0), loc='upper left');
```



```
[91]: nba_avg_df.groupby(['Era','Finals_Rk', 'Conf.'])['MOV'].mean()
```

```
[91]: Era           Finals_Rk       Conf.
      1980s        Champion        East      7.064000
```

|               |                 |      |           |
|---------------|-----------------|------|-----------|
|               |                 | West | 6.652000  |
|               | Runner-Up       | East | 5.684000  |
|               |                 | West | 3.884000  |
|               | Knocked Out     | East | 2.436863  |
|               |                 | West | 1.682329  |
|               | Never Qualified | East | -3.801290 |
|               |                 | West | -4.388929 |
| 1990s         | Champion        | East | 8.867143  |
|               |                 | West | 4.833333  |
|               | Runner-Up       | East | 7.060000  |
|               |                 | West | 6.291250  |
|               | Knocked Out     | East | 2.594483  |
|               |                 | West | 3.249277  |
|               | Never Qualified | East | -3.782241 |
|               |                 | West | -5.579167 |
| 2000s         | Champion        | East | 6.656667  |
|               |                 | West | 6.907143  |
|               | Runner-Up       | East | 4.656000  |
|               |                 | West | 5.743333  |
|               | Knocked Out     | East | 1.716230  |
|               |                 | West | 3.500370  |
|               | Never Qualified | East | -3.897288 |
|               |                 | West | -3.514605 |
| 2010 & Beyond | Champion        | East | 6.366000  |
|               |                 | West | 7.167143  |
|               | Runner-Up       | East | 3.918571  |
|               |                 | West | 7.112000  |
|               | Knocked Out     | East | 2.510541  |
|               |                 | West | 3.322766  |
|               | Never Qualified | East | -4.392000 |
|               |                 | West | -3.472551 |

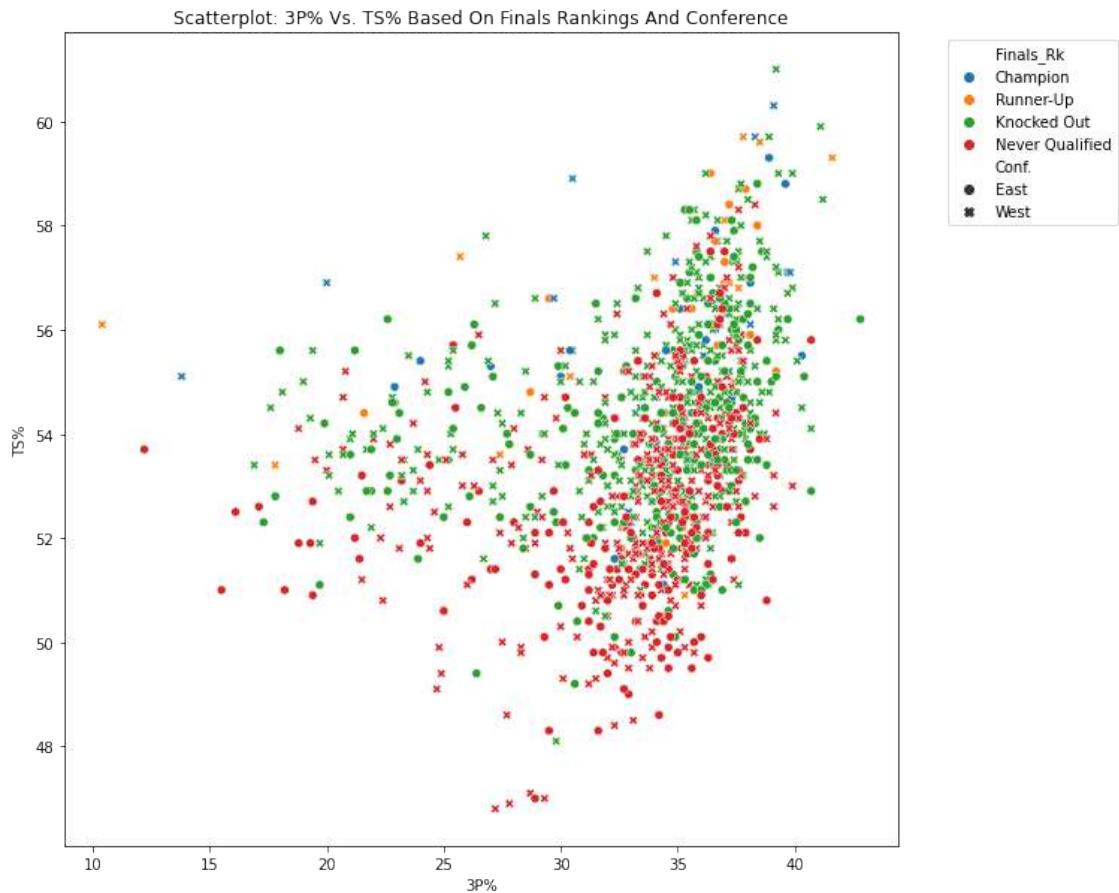
Name: MOV, dtype: float64

**Question 10 Observation** > Now that we have plotted everything, lets examine it , but, first, I wanted to check how many times each conference has won the championship over the past 4 decades. It seems like the Western Conference has won slightly more with a total of 22, while the East has a total of 20. Now we will jump to the lineplot where I will primarily focus on explaining the champions' and runner-ups' MOV. Starting off with the 1980s era, we see that the Eastern Conference champions have achieved an average margin of victory (MOV) of about 7 points during the regular season, while the Western Conference champions have just a little under that with 6.7 points. Those who played in the Eastern Conference and won second place (runner-up) have an average regular-season MOV of 5.7 points, while the Western Conference's runner-ups of the same era had an average MOV of 3.9. Moving onto the 1990s era, we find that those who have become the NBA champions and come from the East won by a slightly higher average MOV during the regular season than the previous era with 8.9 points. However, the Western champions' regular-season MOV decreased to 4.8 points. The runner-ups for both conferences in the 1990s seem to be doing better during the regular season during this time, too. It's better than the last era. The

MOV for the Eastern Conference runner-ups averaged out to about 7 points, while the Western Conference averaged 6.3 points- slight increases since the '80s. Moving onto the 2000s era, we see that the Western champions have averaged a higher MOV than the Eastern ones for the first time. The Western champions have an average regular-season MOV of 6.9 points while the East comes out at 6.7. This pattern also continues with the runner-ups as well! The Western runner-ups averaged 5.7 MOV while the Eastern ones averaged 4.7 points. It seems like the West has been getting more skilled since the '90s. Lastly, we move onto the current era(2010 & Beyond) where we see that the Western champion MOV increased slightly from the last era to 7.2 points. The Eastern champions'have an average of 6.4 points, which is a slight decrease from the previous era. Examining the runner-ups for this era, we see the East has an average MOV of 3.9, but the West has 7.1! That's almost doubled! One thing I do want to mention about those who were knocked out is that their MOV never went over 4, but we can see that over time it got larger.

#### Extra

```
[92]: fig = plt.figure(figsize = [10,10])
ax = sb.scatterplot(data=nba_avg_df, x="3P%", y="TS%", hue="Finals_Rk", style = "Conf.")
sb.move_legend(ax, "upper left", bbox_to_anchor=(1.05, 1.0))
plt.title('Scatterplot: 3P% Vs. TS% Based On Finals Rankings And Conference');
```



**Extra Observation >** This is quite interesting to see. Taking a look at the top right corner where True-Shooting Percentage(TS%) is greater than 58% and high 3-point percentage (3P%) is greater than 35%; we can see that most of the teams are from the Western Conference. Furthermore, we can see that there are a number of champions and runner-ups in this area with 2 and 7 respectively. However, when examining the champions and runner-ups in the other area of this graph, we find that they are pretty spreadout. One champion even had below a 15% 3-point average during the regular season. Another runner-up team even had a 10% 3-point average during the regular season!

## 1.6 Conclusion

That was a lot of visualizations! Now let's summarize what we found from our entire exploratory process. First, I will begin with the total stats dataset; then, we will move onto the average dataset where we really analyzed how teams played during the regular reason based on their Finals ranking. Beginning with the total stats dataset, we found that the Atlanta Hawks have scored the most points in NBA history with 361,254 points as of the end of the year 2021 (2020-2021 season). The Boston Celtics and Brooklyn Nets came in second and third, respectively. During the beginning of my investigation, I guessed that at least two of these would have won an NBA championship. However, that quickly changed after I plotted my bar graph on the number of playoff appearances by team. In the plot, I found that the San Antonio Spurs have made it to the playoffs the most over the years with a total of 36 appearances. Following the Spurs are the Lakers and Blazers, who are both tied for 2nd place with a total of 34 appearances. It seems more likely that those who have made the playoffs more would have a greater chance of winning the championship. In a way, my suspicions have been proven correct when we plotted a pie chart of the teams that have won at least one NBA championship. We found that the Lakers have won a little over a quarter of all NBA championships over the last 40 years with 11. Following them are the Chicago Bulls and San Antonio Spurs with 6 and 5 championships, respectively. Looking at the least number of championships over the last 4 decades, we can see that 5 teams are tied for last; those teams are the 76ers, Mavericks, Cavaliers, Raptors, and Bucks. During out investigation, we have also found that the number of 2-point attempts during the regular season has been decreasing over time, but the number of 3-pointers has been skyrocketing! Next, we moved onto the average stats dataset, where we found that the average age of an NBA team over the years is 27 years of age. Additionally, we also found that the winningest team in the NBA since 1980 is the San Antonio Spurs, who won 61.8% of their regular-season games. Coming in 2nd are the Lakers with 61.2%. However, I would have never guessed that the Minnesota Timberwolves would have the worst winning percentage with 39.4%. After that, we created a heatmap where we mapped out the correlation of all the numeric variables. There were certainly some surprises such as the negative relationship between TOV% and MOV which I expected to be strong, but it turns out to be weak. One positive relationship I found surprising was the one between age and regular-season winning percentage, which led me to believe that teams who won championships were on the older side. After plotting out the strongest positive relationship on a scatterplot (MOV & Win%), we found that all the champions have over 60% Win% during the regular season, and a minimum MOV of 2. Next, when we plotted out the true-shooting percentage based on Finals rankings and eras, we found that the median TS% has been higher than it has ever been regardless of the

rankings. This means that players are taking more efficient shots. In the current era, the runner-ups for this era have a 58% TS% median, while the champions have about 57%. Afterward, we plotted a heatmap of the 3-Point Attempt Rate (3PAr) and found that it has been increasing era after era. Using the ‘Champion’ row as an example, we see that only 3.9% of all shot attempts were from the 3-point line in the 1980s era. In the next era, the percentage nearly quadrupled to 13.54%; by the 2010s & Beyond era, we see that 31.19% of all shot attempts were taken from the 3-point line. This is the general trend for all of the Finals rankings. Next, we plotted the age distribution on a bar plot. Although we have a smaller sample size that fits the criteria for Champion and Runner-up, we find that the teams who won NBA championships are on average older, while the runner-ups tend to be a little younger. Furthermore, the champions and runner-ups have averaged more assists per game during the regular season. Sometimes they both averaged the league’s top 2 total assists per game, as seen in the mid-1980s. Finally, we separated the teams up into 2 conferences (West & East) for our last question. Examining only the champions and runner-ups, we found that the West’s regular-season MOV average was lower in the 1980s and 90s, but it recovered and outpaced the East in the last 2 eras. For example, the runner-ups for this era (2010s & Beyond), we see the East has an average MOV of 3.9, but the West has 7.1! That’s almost doubled!

[ ]:

[ ]:



**project**

**10**

**Financial Youtube Channel - Data  
Analysis and Visualization**

# Summary of Project

For this project, I've decided to analyze various financial Youtube channels that I've been following; most of channels talk about the stock market. If you've been watching Youtube, you oftentimes hear content creators telling you to like, comment, and subscribe to help promote their videos. While this might not necessarily be the case, it certainly is worth exploring to look for certain factors that may support this claim. That is exactly what I am going to be doing in this notebook, along with identifying trends among certain Youtube channels with a particular niche or topic. What I hope to gain are some insights from these content to help me better understand the Youtube investing landscape, such as the most-talked about topic/stock. Some numeric variables that I am going to analyze are the number of likes, views and comments. I will then use natural language processing to create a wordcloud of the most frequently used words in each video's comment section and title. Please note that the scope of this project is small; you will see that I've analyzed 9 of my favorite investment channels. To gather these data and create my own dataset, I used the [Google Youtube Data API v3](#).

Youtube\_API.ipynb - In this notebook, I gathered all of the channels' videos, statistics and comments using the Youtube API. I then cleaned/pre-processed the data for analysis and visualization.

## Objectives

In this project, I am looking to answering the following questions:

- What is the total view count by channel?
- What is the video distribution like based on the day uploaded? How about when based on the month?
- What is the relationship like between the view count and comment count? View and tags? View and like?
- What is the video distribution like based on duration?
- For each channel, what is the comment and view distribution like?
- What is the average view count for each channel based on upload day?
- (Natural Language Processing Question) What are some of the most used words in the video titles for all channel?
- (Natural Language Processing Question) What are some of the most used words in the video comments for all channels?
- Does title length affect view count?

## Installation

Packages used:

```
* Matplotlib
* Seaborn
* Numpy
* Pandas
* googleapiclient.discovery
* IPython.display
* itertools
* imageio
* wordcloud
* nltk.corpus
* nltk.tokenize
```

# Financial Youtube Channel - Data Analysis and Visualization

## Background Information

### Introduction

Since its inception, Youtube has grown to become one of the biggest platforms for content creators in the world, processing billions of searches per month. However, there are myths as to how the Youtube algorithm works, what makes certain videos get more views than others, etc. If you've been watching Youtube, you oftentimes hear content creators telling you to like, comment, and subscribe to help promote their videos. While this might not necessarily be the case, it certainly is worth exploring to look for certain factors that may support this claim. That is exactly what I am going to be doing in this notebook, along with identifying trends among certain Youtube channels with a particular niche or topic. As a retail investor, I've decided to analyze various financial Youtube channels that I've been following; most of channels talk about the stock market. What I hope to gain are some insights from these content to help me better understand the Youtube investing landscape, such as the most-talked about topic/stock. Please note that the scope of this project is small; you will see that I've analyzed 9 of my favorite investment channels. To gather these data and create my own dataset, I used the [Google Youtube Data API v3](#).

### Objectives

In this project, I am looking to answering the following questions:

- What is the total view count by channel?
- What is the video distribution like based on the day uploaded? How about when based on the month?
- What is the relationship like between the view count and comment count? View and tags? View and like?
- What is the video distribution like based on duration?
- For each channel, what is the comment and view distribution like?
- What is the average view count for each channel based on upload day?
- (Natural Language Processing Question) What are some of the most used words in the video titles for all channel?
- (Natural Language Processing Question) What are some of the most used words in the video comments for all channels?
- Does title length affect view count?

### Steps

1. Gathering Data With Youtube API
2. Pre-Processing/Cleaning

3. Exploratory Data Analysis
4. Conclusions

## Data Limitation

The dataset that I've gathered is real-world dataset gathered from Youtube itself. However, please keep in mind that the channels that I've gathered are purely my decisions and they do not represent the "popular" channels based on number of subscribers. The channels collected are based on what I follow and deem as 'good' investment channels. Also, I wanted to include a dislike count, but, as of today(3/20/22), the youtube dislike feature has been disabled.

```
import pandas as pd
import googleapiclient.discovery
from IPython.display import JSON
import itertools

import matplotlib.pyplot as plt
from matplotlib.ticker import FuncFormatter
%matplotlib inline
import seaborn as sb
import imageio
import isodate

#NLP
from wordcloud import WordCloud, STOPWORDS
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
```

## Gathering Data With Youtube API

For my first step, I created a project on Google Developers Console and requested an API key which then allowed me to enable Youtube API for my application. Afterwards, I gathered all the channel IDs that I'm interested in analyzing. Lastly, I created functions for gathering all the data and statistics that I need.

```
api_key = ['HIDDEN'] # API key is hidden for security purposes

# list of channels that I am interested
channel_ids = ['UCbta0n8i6Rljh0ob07HzG9A', 'UCfCT7SSFEWYG4th9ZmaGYqQ',
                'UCJwKCyEIFHwUOPQQ-4kC1Zw', 'UCxwoFvCK8RCq9J4RbIW2qrQ',
                'UCL-F-8nxFtFZ4M3RWkGn6Qw',
                'UC03tlaeZ6Z0ZN5frMZI3-uQ', 'UCKcnYVAVZQ0B-nXHechtXDg',
                'UCZ5V5HyAvkFrjsWkANqXscQ',
                'UC7r4-nZ4icT8SIcnisXFSHQ']

api_service_name = "youtube"
api_version = "v3"
# Get credentials and create an API client
```

```

youtube = googleapiclient.discovery.build(
    api_service_name, api_version, developerKey=api_key)

def get_channel_stats(youtube, channel_ids):
    """
        Function: Gather interested channel stats from youtube creator's
        channel page

    INPUT:
        youtube - build object from googleapiclient.discovery
        channel_ids - (list) list of channel ids to be analyzed

    OUTPUT:
        all_data - (pandas dataframe) dataframe that consists of the
        following columns: channelName, publishDate, subscribers, views,
        totalVideos, playlistId
    """
    all_data = []

    request = youtube.channels().list(
        part="snippet,contentDetails,statistics",
        id=', '.join(channel_ids)
    )
    response = request.execute()

    #loop through items
    for item in response['items']:
        data = {'channelName': item['snippet']['title'],
                'publishDate': item['snippet']['publishedAt'],
                'subscribers': item['statistics']['subscriberCount'],
                'views': item['statistics']['viewCount'],
                'totalVideos': item['statistics']['videoCount'],
                'playlistId': item['contentDetails']
                ['relatedPlaylists']['uploads']}
        all_data.append(data)
    all_data = pd.DataFrame(all_data)

    return(all_data)

def get_videos_ids(youtube, playlist_id):
    """
        Function: Gather videoIds from channel.

    INPUT:
        youtube - Get credentials and create an API client/Initialise a
        Youtube API service object.
        playlist_ids - (list) list of playlist ids to be analyzed.

    OUTPUT:
    """

```

```

    video_ids - (list) list of dictionary that contains all videoId
for channel.
"""
video_ids = []

request = youtube.playlistItems().list(
    part="snippet, contentDetails",
    playlistId= playlist_id,
    maxResults = 50
)

response = request.execute()

for item in response['items']:
    data = {
        'videoId': item['contentDetails']['videoId']
    }
    video_ids.append(data)

next_page_token = response.get('nextPageToken')
while next_page_token is not None:
    request = youtube.playlistItems().list(
        part="snippet, contentDetails",
        playlistId= playlist_id,
        maxResults = 50,
        pageToken = next_page_token
    )
    response = request.execute()

    for item in response['items']:
        data = {
            'videoId': item['contentDetails']['videoId']
        }
        video_ids.append(data)
    next_page_token = response.get('nextPageToken')

return video_ids

```

**def get\_video\_details(youtube, video\_ids):**

*Function: Gather interested information from videos and store in dataframe.*

**INPUT:**

*youtube - Get credentials and create an API client/Initialise a Youtube API service object.*

*video\_ids - (list) list of video ids.*

**OUTPUT:**

*video\_df - (pandas dataframe) dataframe of video statistics.*

```

Includes columns:
    'channelTitle', 'title', 'description', 'tags',
'publishedAt',      'viewCount', 'likeCount', 'favouriteCount',
'commentCount',     'duration', 'definition', 'caption'%
"""

all_video_info = []
for i in range(0, len(video_ids), 50):
    request = youtube.videos().list(
        part="snippet,contentDetails,statistics",
        id=','.join(video_ids[i:i+50])
    )
    response = request.execute()

    for video in response['items']:
        # create dictionary of stats I want to keep
        stats_keep = {'snippet': ['channelTitle', 'title',
'description', 'tags', 'publishedAt'],
                      'statistics': ['viewCount', 'likeCount',
'favouriteCount', 'commentCount'],
                      'contentDetails': ['duration', 'definition',
'caption']}
        }

# empty dictionary to keep track of keys and values
video_info = {}
video_info['video_id'] = video['id']

# extract values and append them into empty dictionary
for k in stats_keep.keys():
    for v in stats_keep[k]:
        try:
            video_info[v] = video[k][v]
        except:
            video_info[v] = None

    all_video_info.append(video_info)
video_df = pd.DataFrame(all_video_info)
return video_df

def get_videos_comments(youtube, video_ids):
"""
Function: Gather comments from videos and store in dataframe.

INPUT:
youtube - Get credentials and create an API client/Initialise a
Youtube API service object.
video_ids - (list) list of video ids.

```

```

OUTPUT:
all_comments_df - (pandas dataframe) dataframe of comments. Each
video has a max of
10 comments which are compiled in a list.
"""
all_comments = []

for video_id in video_ids:
    try:
        request = youtube.commentThreads().list(
            part="snippet,replies",
            videoId=video_id
        )
        response = request.execute()
        # help
        https://developers.google.com/youtube/v3/docs/commentThreads?
hl=en_US#snippet topLevelComment
        video_comments = [comment['snippet']['topLevelComment']
['snippet']['textOriginal']
                        for comment in response['items'][0:10]]
        video_comments_info = {'video_id': video_id, 'comments':
video_comments}
        all_comments.append(video_comments_info)
    except:
        # dealing with errors
        pass

all_comments_df = pd.DataFrame(all_comments)
return all_comments_df

```

## Gathering Channel Statistics

|   | channelName                | publishDate                 | subscribers |
|---|----------------------------|-----------------------------|-------------|
| 0 | Benjamin                   | 2021-02-02T15:23:39.467329Z | 283000      |
| 1 | Cents Invest               | 2019-03-22T21:32:26Z        | 92600       |
| 2 | Dividend Data              | 2020-04-19T20:17:08.795625Z | 96400       |
| 3 | Joseph Carlson After Hours | 2017-02-14T04:41:26Z        | 53900       |
| 4 | Joseph Carlson             | 2013-09-19T17:32:49Z        | 240000      |
| 5 | Nate O'Brien               | 2016-12-31T06:50:03Z        | 1230000     |

|   |                      |                      |        |
|---|----------------------|----------------------|--------|
| 6 | Daniel Pronk         | 2015-12-17T20:49:02Z | 197000 |
| 7 | The Investor Channel | 2016-02-16T23:11:03Z | 113000 |
| 8 | Tom Nash             | 2017-08-15T13:06:55Z | 282000 |

|   | views    | totalVideos | playlistId               |
|---|----------|-------------|--------------------------|
| 0 | 15877118 | 32          | UUkcnYVAVZQ0B-nXHechtXDg |
| 1 | 2992449  | 287         | UUxwoFvCK8RCq9J4RbIW2qrQ |
| 2 | 4886681  | 83          | UUZ5V5HyAvkFrjsWkANqXscQ |
| 3 | 2810611  | 85          | UUfCT7SSFEWyG4th9ZmaGYqQ |
| 4 | 16335202 | 235         | UUbta0n8i6Rljh0ob07HzG9A |
| 5 | 63768501 | 183         | UU03tlaeZ6Z0ZN5frMZI3-uQ |
| 6 | 9107258  | 193         | UUL-F-8nxFtFZ4M3RWkGn6Qw |
| 7 | 6947619  | 568         | UU7r4-nZ4icT8SIcnisXFHQ  |
| 8 | 35674132 | 535         | UUJwKCyEIFHwUOPQQ-4kC1Zw |

## Gathering Video Statistics For All Channels

In this step, I will be gathering the video statistics for all the channels. In total, there are 2201 rows of videos and 13 columns in the dataset.

```
playlist_ids = list(channel_stats.playlistId.unique()) # convert all
unique values in playlistId into a list
video_ids_list = []

# loop to get video ids from all interested channels
for playlist_id in playlist_ids:
    video_ids = get_videos_ids(youtube, playlist_id)
    video_ids_list.append(video_ids)
video_ids_list

[[{'videoId': 'jm55pm_ZIdI'},
 {'videoId': '9ZVvgxLhs2M'},
 {'videoId': 'v3NUSNQafYM'},
 {'videoId': 'YUcdYDQJkBU'},
 {'videoId': 'CSz67F0ZS0w'},
 {'videoId': 'KgH4rDtxqHg'},
 {'videoId': 'iFB0GMjxI0'},
 {'videoId': 'ZN6P9ErUc0g'},
 {'videoId': '1UY0uS6Z6oE'},
 {'videoId': 'Mvixia9iUWs'},
 {'videoId': '2Zus6SyQhW4'},
 {'videoId': 'CWXvIwv3pg4'},
 {'videoId': '8pYgz4YlQnE'},
 {'videoId': 'DEA2sCE2CJs'},
 {'videoId': '8QaP43sF05A'},
 {'videoId': 'TrTeLB8F98k'},
```

```
[{'videoId': 'TM042qkPio'},  
 {'videoId': 'dgisRHEQ2FM'},  
 {'videoId': '80BXR4zX2EE'},  
 {'videoId': 'DJZRUrYrZnc'},  
 {'videoId': '8fqGgIYggig'},  
 {'videoId': '7qrQbsJTiTA'},  
 {'videoId': 'DMw10sVuQn4'},  
 {'videoId': 'E7-e7NeoQzA'},  
 {'videoId': 'p1nBs05XkHo'},  
 {'videoId': 'SqPJ-hTRhsM'},  
 {'videoId': 'xr2DfWmQ4_Y'},  
 {'videoId': 'WCD58idv_U0'},  
 {'videoId': 'NW5927_LPUs'},  
 {'videoId': 'XYZqrkHTGo8'},  
 {'videoId': 'FEMFVM1PskM'},  
 {'videoId': 'XKPmtbCmkE4'}],  
 [{"videoId": "-dd2dTkg0-g"},  
 {"videoId": "XgYlbT8LjGE"},  
 {"videoId": "6IdPox_jJ7k"},  
 {"videoId": "vF4AUVFV3Ms"},  
 {"videoId": "QJ_iZwReJAo"},  
 {"videoId": "qqQhqoloOs"},  
 {"videoId": "Ug0vWIfrM"},  
 {"videoId": "q_iKG9WKI8A"},  
 {"videoId": "0jEWUDfMnDo"},  
 {"videoId": "z0ms0SS09ZE"},  
 {"videoId": "kXTH63T27xM"},  
 {"videoId": "AmnVvaYQ3Dw"},  
 {"videoId": "m49QJQDPbf0"},  
 {"videoId": "1cAG8Z-3dEc"},  
 {"videoId": "w5wte0s21KM"},  
 {"videoId": "a5eAuiwn5cc"},  
 {"videoId": "knEQHEOXlNU"},  
 {"videoId": "40L0-lJWldQ"},  
 {"videoId": "HVX9JyFGKpk"},  
 {"videoId": "zytdDMBmDlI"},  
 {"videoId": "7LKMJV3-dwY"},  
 {"videoId": "pwRWujL-i2Q"},  
 {"videoId": "9ulsPhQeM6w"},  
 {"videoId": "Um5xdlVi3PA"},  
 {"videoId": "ipPbj6AZWUs"},  
 {"videoId": "c0mNbMKpo2E"},  
 {"videoId": "xjH27La5n8U"},  
 {"videoId": "GFQvfCGwlZI"},  
 {"videoId": "AIip0gpmSlg"},  
 {"videoId": "26_Fr5ijIxI"},  
 {"videoId": "2gc-mUKzy-I"},  
 {"videoId": "6ouvD6i0s0Q"},  
 {"videoId": "b3mqhh5C9Aw"}]
```

```
{'videoId': 'YuvCvowmFqw'},  
{'videoId': 'oEsQnv0GrDc'},  
{'videoId': 'W0M63lLzmho'},  
{'videoId': 'zCKpjEAwNUE'},  
{'videoId': '0UD3XTJHoz8'},  
{'videoId': 'k1F8a2TKgig'},  
{'videoId': 'k-nkwBc0mRk'},  
{'videoId': 'pY5i8suffFyU'},  
{'videoId': '4Sm8Mha0wko'},  
{'videoId': 'T0Xd26D6igg'},  
{'videoId': '71xHBMuD4oc'},  
{'videoId': 'owfTagzHKMU'},  
{'videoId': 'deuh7M07DH8'},  
{'videoId': 'arUh1p5jGbU'},  
{'videoId': '3NmIJ5wN5Mw'},  
{'videoId': 'B0yx-vVsYrM'},  
{'videoId': 'ozRqoVp-quA'},  
{'videoId': 'ga0nlxxEo7E'},  
{'videoId': 'o-t80wxZECK'},  
{'videoId': 'g4n0IqGIE_k'},  
{'videoId': 'R7fQfg7PENs'},  
{'videoId': 'oMibeJGh8U8'},  
{'videoId': '8PTd2NuhEEY'},  
{'videoId': 'D6tMuFurIHc'},  
{'videoId': 'usglWgXK_TY'},  
{'videoId': 'wCS2Kftf09A'},  
{'videoId': 'dBcIu1Bm7xA'},  
{'videoId': 'hhaggBP-OFs'},  
{'videoId': 'RoAGiiDaLRQ'},  
{'videoId': 'Yg2ahncYXY'},  
{'videoId': 'fx7VIPzvYIA'},  
{'videoId': 'DsKZGPYCMQg'},  
{'videoId': '16Q8T8lw9eE'},  
{'videoId': 'DVGZ533SdUU'},  
{'videoId': 'eE0Q6HZrXY'},  
{'videoId': 'eJGxtF9RkAw'},  
{'videoId': 'dWsF2gAmesI'},  
{'videoId': 'tYwgDBNRp-M'},  
{'videoId': 'x8-5PUzhBf8'},  
{'videoId': 'UbONpc4ufPw'},  
{'videoId': '8HQeqstoKBY'},  
{'videoId': 'aZuwNH0cGtY'},  
{'videoId': '8V8CPzW9DeU'},  
{'videoId': 'dwU7_Tuqjo0'},  
{'videoId': 'mCnAackca0k'},  
{'videoId': 'EU-C0-3c_Ew'},  
{'videoId': 'nnxQhNTtb5M'},  
{'videoId': 'gDWeY2QcCAA'},  
{'videoId': 'gGVeCIG0H8k'},
```

```
{'videoId': '37btabkHeSo'},  
{'videoId': 'zrFTqg0NTPU'},  
{'videoId': 'Vy5kzlswXbo'},  
{'videoId': 'fDLnPFK3mI8'},  
{'videoId': 'puVDzWpvb1w'},  
{'videoId': 'iNe8fPp-tXs'},  
{'videoId': 'd7DjpEbo09Y'},  
{'videoId': 'kf40RfVerxM'},  
{'videoId': 'BiOLSP2bBPU'},  
{'videoId': 'qrip5sLFFPg'},  
{'videoId': 'HUI5ns2nXcQ'},  
{'videoId': 'B73Lm6ExZD8'},  
{'videoId': 'f4Gt9Uc1X5M'},  
{'videoId': 'DQzZIbAmB1E'},  
{'videoId': 'LnkKqnDNe04'},  
{'videoId': '0XaFuUZ5Ahc'},  
{'videoId': '-31BC7bfaqU'},  
{'videoId': 'M68oLxAbJKU'},  
{'videoId': 'xvJWuGVBl8E'},  
{'videoId': 'vuDHdKee9Ns'},  
{'videoId': 'BBNMJRVaAs0'},  
{'videoId': 'tlhhmnUB2_M'},  
{'videoId': 'FRMF0iqK-4'},  
{'videoId': 'BiXlXN9JeGg'},  
{'videoId': 'g5CjwoSVS-U'},  
{'videoId': 'BBwWJ9KDvHY'},  
{'videoId': 'PGBX9V5dJ0s'},  
{'videoId': 'hRNfJUPT15s'},  
{'videoId': 'j7aak-w_swY'},  
{'videoId': 'lR20oin3Tjc'},  
{'videoId': 'Sr4kJzBB3w0'},  
{'videoId': 'iXJzyKC74wg'},  
{'videoId': 'Y7M2A2Hw5DQ'},  
{'videoId': 'YftbFXApXL1'},  
{'videoId': '1Reyn7sCnNU'},  
{'videoId': 'lnlWutF6Njk'},  
{'videoId': '20n7Ub1RzD4'},  
{'videoId': '4CMWZeEkDew'},  
{'videoId': 'G9gjUKE_cjA'},  
{'videoId': 'JTcQqa0Ec7o'},  
{'videoId': '0RfLPaf07Ao'},  
{'videoId': 'xoiG2TVJfAI'},  
{'videoId': '4cYpt1SHbS8'},  
{'videoId': 'fg_33ujTqmo'},  
{'videoId': '6povLphdbTk'},  
{'videoId': 'npeWjArxAh4'},  
{'videoId': 'aDXDYhDNg2A'},  
{'videoId': 'doBofg0FD58'},  
{'videoId': 'uKl3Nw3aXfI'},
```

```
[{'videoId': 'xQ-IZLWy3r8'},  
 {'videoId': 'AT3AJeI2uTE'},  
 {'videoId': 'g7_GE-3RGKE'},  
 {'videoId': 'ih7hGKy_DUI'},  
 {'videoId': 'U3wiwoxZLlw'},  
 {'videoId': 'nJzIs73p4pM'},  
 {'videoId': 'Tp1G0qskNmw'},  
 {'videoId': '9mzCSx0e7eE'},  
 {'videoId': 'Q6WEH-qrKjk'},  
 {'videoId': 'YQl4Almbw2E'},  
 {'videoId': 'NnAYLXzW1Pw'},  
 {'videoId': '8tKLxuPn8_8'},  
 {'videoId': 'dt0uJ1r2G14'},  
 {'videoId': 'IJIHdCNDPZw'},  
 {'videoId': '07dPB9mySao'},  
 {'videoId': 'doQ-1RnghUI'},  
 {'videoId': 'nnt9yJQsyGg'},  
 {'videoId': 'fCSAClw0faE'},  
 {'videoId': 'kalgjzbNYD0'},  
 {'videoId': 'MH65uRpjKl0'},  
 {'videoId': '2F5IcN7DsEA'},  
 {'videoId': 'c0jgIjNNpUE'},  
 {'videoId': 'kezG5kh2THw'},  
 {'videoId': 'VB_IHtIQ_R0'},  
 {'videoId': 'fdmvxBGW3rY'},  
 {'videoId': 'zEujC2DHsHk'},  
 {'videoId': 'm5DrMuIhnAI'},  
 {'videoId': 'DBmWxd_TN1c'},  
 {'videoId': 's22UXbt298o'},  
 {'videoId': 'foSrFzGqkjc'}],  
 [{"videoId": "Vqq5m8e9Xk0"},  
 {"videoId": "nFUwYUK7jZc"},  
 {"videoId": "mXH7KXmuhxc"},  
 {"videoId": "f9QsGhrQPXg"},  
 {"videoId": "kpFz16tKVNC"},  
 {"videoId": "fKtjd26aQhE"},  
 {"videoId": "uVoVQBUDaUA"},  
 {"videoId": "ah4fwM0eo80"},  
 {"videoId": "t6o4j-5wy54"},  
 {"videoId": "63CBqmwIGl8"},  
 {"videoId": "x2Pu2qsR0Mw"},  
 {"videoId": "A8SsCulR-fM"},  
 {"videoId": "Yo8kQ5hcoUM"},  
 {"videoId": "GkamatmXsaw"},  
 {"videoId": "Ff9YUkY3-ek"},  
 {"videoId": "e0sgzE-CUkA"},  
 {"videoId": "UuuAB2Sg2f0"},  
 {"videoId": "VL51gxdvldI"},  
 {"videoId": "gVsJvLneNS0"},  
 {"videoId": "qwPj2LqkRhs"}]
```

```
{'videoId': 'voxZa-dLL_Y'},  
{'videoId': 'Eeb6PG7qDdI'},  
{'videoId': 'Dmy8JXoX548'},  
{'videoId': '8DgKeiWlFYY'},  
{'videoId': 'UkmHS80jxzE'},  
{'videoId': 'hume07Y528E'},  
{'videoId': 'fq-raK-WnN0'},  
{'videoId': 'YuMH0HssAkQ'},  
{'videoId': 'yk3Q_sqy4M8'},  
{'videoId': '3aPyB3dbNkg'},  
{'videoId': 'ITj9C4FvquQ'},  
{'videoId': 'y9BDeXoqEyo'},  
{'videoId': 'Kat0SDrb_5I'},  
{'videoId': '7e-Vxjs_QCk'},  
{'videoId': 'B6NeBKAoDSc'},  
{'videoId': '1nyrPYpKZkw'},  
{'videoId': '32kwoyJZS78'},  
{'videoId': 'fzU1m7eRyDk'},  
{'videoId': 'pcZljt0yCL0'},  
{'videoId': '03QK82jDSQQ'},  
{'videoId': 'hkMYKHbLi2Q'},  
{'videoId': 'aPn7ee1p3pk'},  
{'videoId': 'dAszqIYfKtE'},  
{'videoId': '6iSU1m3otd8'},  
{'videoId': 'uYdtZV2AQJc'},  
{'videoId': '1jWoU0Qw_FI'},  
{'videoId': 'Bg0opt0exUo'},  
{'videoId': 'SVpzpglbY08'},  
{'videoId': '5nYMKQpb4HI'},  
{'videoId': 'sUYE5vfp330'},  
{'videoId': 'XnJVKhxUkRU'},  
{'videoId': 'GuuZljH0ud0'},  
{'videoId': 'FpkAwHkKrd0'},  
{'videoId': '-cc4HNi_agS'},  
{'videoId': 'umBbS6WZS0E'},  
{'videoId': 'ZVxAnBL7VUU'},  
{'videoId': '2hTU-oD4duE'},  
{'videoId': '7cR2YCskKBo'},  
{'videoId': 'mmYh_rjLaHM'},  
{'videoId': 'a0zsZunCAvU'},  
{'videoId': 'wqHq3cMJpZ0'},  
{'videoId': 'QKHBTLZ07d4'},  
{'videoId': 'bvgou00S20'},  
{'videoId': 'wwBfzgPmBVY'},  
{'videoId': 'UYhJdbRRGfk'},  
{'videoId': 'kZ-JJsJOYXg'},  
{'videoId': '3H5cg2_F0Dg'},  
{'videoId': '0xi2wmRzwxQ'},  
{'videoId': 'p7whnWBCh8'},
```

```
{'videoId': 'Yvpa7q1vmEE'},  
{'videoId': 'MGSG7B_U9Sk'},  
{'videoId': 'kk1cVyje3eo'},  
{'videoId': 'rLnHx50jq3c'},  
{'videoId': 'S7dqldSzZX8'},  
{'videoId': 'ape2bCuRuxY'},  
{'videoId': 'iFTyRTlvi3k'},  
{'videoId': 'LxAniFgeC0g'},  
{'videoId': 'YRw_bpCcAOM'},  
{'videoId': '7K43sEjKWIE'},  
{'videoId': 'mAC9B5vreJ4'},  
{'videoId': 'l63bmKW0eJM'},  
{'videoId': 'NkhT5Uzacow'},  
{'videoId': 'kSSnIZHfteY'},  
{'videoId': 'E49pDL_r50U'},  
{'videoId': 'FmMaWQJTxdS'},  
{'videoId': 'L8W1tbEVy0M'},  
{'videoId': 'YH7JmqFz9QM'},  
{'videoId': 'Hj6nlIiL53U'},  
{'videoId': 'SYQwN7khKXo'},  
{'videoId': 'LTstvFf-Wuw'},  
{'videoId': 'kfM0v0S97X0'},  
{'videoId': '0iiprci_0yA'},  
{'videoId': 'Hs6QfsN9sq0'},  
{'videoId': 'QjuGisZpc98'},  
{'videoId': 'QeojCx4LoEg'},  
{'videoId': 'gXAbL-bxgyk'},  
{'videoId': 'M4-vQaWFRS0'},  
{'videoId': 'IHC4PwPzM8'},  
{'videoId': 'TasqYrb81Y8'},  
{'videoId': 'CXYzu0xHHKU'},  
{'videoId': 'aFn5KPC1itU'},  
{'videoId': 'NcKXxJ0XdAI'},  
{'videoId': 'e6CjErl00cE'},  
{'videoId': 'gIeT2Px6DvE'},  
{'videoId': 'aGspfZwehCw'},  
{'videoId': '8poZ0WqSsMI'},  
{'videoId': 'iqFWu3zn_ag'},  
{'videoId': 'sciBB_ym4s8'},  
{'videoId': 'AKCyf6Ti-bs'},  
{'videoId': '0kHOLCZTFtc'},  
{'videoId': 'fQL5uwWinB4'},  
{'videoId': 'dbjmiNFr7GY'},  
{'videoId': 'ypeePFkggCs'},  
{'videoId': 'xL68ELzbqfM'},  
{'videoId': '1IEZex0Jnac'},  
{'videoId': '0V6xU_AnWaU'},  
{'videoId': 'Dv3tkLmt060'},  
{'videoId': 'd33FUnIz21A'},
```

```
{'videoId': 'K6zk5RMcObM'},  
{'videoId': 'Yln8BS6iZkI'},  
{'videoId': '0Xn-4gIrnzI'},  
{'videoId': 'oV3D8BXsBbc'},  
{'videoId': 'c5J3l5AVGJw'},  
{'videoId': 'q0MlocAFKcg'},  
{'videoId': 'dtiLe3JBelo'},  
{'videoId': '2Qq8GZjmHqw'},  
{'videoId': 'wkibj0IcFJY'},  
{'videoId': '2MStl4dWUos'},  
{'videoId': '_C2I19BVFI'},  
{'videoId': '8hbXTtSsHok'},  
{'videoId': 'vyz7oEEo6-E'},  
{'videoId': 'ixaVyb9IzDY'},  
{'videoId': 'mW058N6IuQA'},  
{'videoId': 'i3fBzDoiw0A'},  
{'videoId': 'thJktgy4E0g'},  
{'videoId': '104d70xxnW0'},  
{'videoId': '6ezhuVvF99E'},  
{'videoId': 'S1ukBsrglQk'},  
{'videoId': 'oy-Cz9M7NpQ'},  
{'videoId': 'c0lrRJQ8bBk'},  
{'videoId': 'c6VwszujYKA'},  
{'videoId': 'ef-WFlZJVBg'},  
{'videoId': 'aF12PX5Rplo'},  
{'videoId': 'Co4b_eqnStQ'},  
{'videoId': 'INCYRkr6k2o'},  
{'videoId': 'dTHNuzGtRgM'},  
{'videoId': '2VP3hYHCLTs'},  
{'videoId': '0pU9IlkuJAU'},  
{'videoId': 'sqyD45EvA4c'},  
{'videoId': 'tTkeoN1yUXI'},  
{'videoId': '8zw5lp6ht0A'},  
{'videoId': 'VLVbQBQfwzE'},  
{'videoId': 'wLI_jV30lTM'},  
{'videoId': 'uVHGgSXtQmE'},  
{'videoId': '7THNE8xEcHk'},  
{'videoId': 'VhP1h7oDDDc'},  
{'videoId': 'u_rQp9ni5nw'},  
{'videoId': 'cXYm6jdRJt0'},  
{'videoId': 'rfuJAyLuMOU'},  
{'videoId': 'aSu44KoXA_M'},  
{'videoId': '0H6hzPmUqWE'},  
{'videoId': 'KonFsibt7BA'},  
{'videoId': '0ShLneAHQoA'},  
{'videoId': 'd4PYF0kLY1A'},  
{'videoId': 'JQZloDtur7w'},  
{'videoId': 'Jf4Nou2NJxo'},  
{'videoId': 'izlm5SiOG2k'},
```

```
{'videoId': 'ClapsqbDxXk'},  
{'videoId': 'sA0xd6hgUWk'},  
{'videoId': 'dk7zn1Yp7Lk'},  
{'videoId': 'iXDv5fAQ060'},  
{'videoId': 'qRjdRZKDDso'},  
{'videoId': 'YlTd8kOPTio'},  
{'videoId': 'IpI99EV2s0M'},  
{'videoId': '5FBYF5as14E'},  
{'videoId': 'fZpdE62SNN0'},  
{'videoId': 'YsEbk1Xrp68'},  
{'videoId': '9_vSbmlJBmI'},  
{'videoId': 'AduVNz20FFs'},  
{'videoId': '19zL1uJ6U4Y'},  
{'videoId': 'TVYolhtBHss'},  
{'videoId': '4eXIJSiuwUo'},  
{'videoId': 'uC2UZ4PxRWk'},  
{'videoId': 'TktsZ8SLYiQ'},  
{'videoId': 'hylSuJGJRDo'},  
{'videoId': 'YenP01_oKnU'},  
{'videoId': 'VA4lEyJXgKo'},  
{'videoId': 'BZFrLGQ3GyU'},  
{'videoId': 's0g-dnxaU-w'},  
{'videoId': 'TiaNHfH8UUk'},  
{'videoId': 'r6DURuedMNC'},  
{'videoId': 'yxdYYhAEEd-w'},  
{'videoId': 'iL_VErBeEeM'},  
{'videoId': '8zKeXNaxcNU'},  
{'videoId': 'g6EMa9Z0cRI'},  
{'videoId': 'c02prAgTICU'},  
{'videoId': 'vBVMVU0zUrU'},  
{'videoId': 'Po0S-h20GFk'},  
{'videoId': '1fVSom7dSM8'},  
{'videoId': 'LkfUkjanv3Y'},  
{'videoId': 'egq5638PeRI'},  
{'videoId': 'sowjfHoeT-s'},  
{'videoId': 'LBEufbIf7sE'},  
{'videoId': 'ntEtKidzr2U'},  
{'videoId': 'ALN2pvULcr4'},  
{'videoId': 'h9f9afQvE2E'},  
{'videoId': '4_UrugttfyE'},  
{'videoId': 'nyB6asQPsPY'},  
{'videoId': 'vRKRjDvpwxo'},  
{'videoId': 'u21jQ_ZXZ9s'},  
{'videoId': 'MK4012od6yk'},  
{'videoId': 'V6yNa4FfJyI'},  
{'videoId': 'tr_jKufWe58'},  
{'videoId': 'EAf9XvT9v5c'},  
{'videoId': 's7QKRvUd0_8'},  
{'videoId': 'pSGEHMMyorIA'},
```

```
{'videoId': 'eCGqb5mw-Ak'},  
'videoId': 'MBKRTUZg1vk'},  
'videoId': 'mphuyNvbP4U'},  
'videoId': '03UqH5XJtKM'},  
'videoId': 'fNV16sQ9a6E'},  
'videoId': 'klgu62fIZWw'},  
'videoId': 'kmGDGvNUFuc'},  
'videoId': '1THr83Y1f3Y'},  
'videoId': 'Kd5h4or152o'},  
'videoId': 'nxDQlTmypybQ'},  
'videoId': 'cE0rzroU760'},  
'videoId': 'Mat7d2Gy4io'},  
'videoId': 'IzXkhzpsv4k'},  
'videoId': '0lJucEgJ-g4'},  
'videoId': 'AfMBV-2XoxU'},  
'videoId': 'oMa2i618qc4'},  
'videoId': 'Yox37AITjTc'},  
'videoId': 'R4d6vJUInXM'},  
'videoId': 'ELlnn5BdafQ'},  
'videoId': 'PLUDgyW_34E'},  
'videoId': 'ujWBD34nEhM'},  
'videoId': '60rV84td6zw'},  
'videoId': 'uEWetByf-jg'},  
'videoId': 'Gh0Ggwg7_Jk'},  
'videoId': 'Zm_Weofl-Yc'},  
'videoId': 'GfNpf_dP8yk'},  
'videoId': 'o2nzf_H_if0'},  
'videoId': 'm3ktyS8qva8'},  
'videoId': '7RgxwFojcHQ'},  
'videoId': 'aHb_qZZJ6AY'},  
'videoId': 'RHU3izT4MW0'},  
'videoId': 'TH1P_yaXc4Q'},  
'videoId': 'eHFpfP4wm5c'},  
'videoId': 'mJ3GBMDXMvk'},  
'videoId': 'xgMfJxe9KU'},  
'videoId': 'GgjKbV4nMWY'},  
'videoId': 'M3W-z8ZdeXo'},  
'videoId': 'ZcvLdUrT53Q'},  
'videoId': 'jG8Go6ayKvQ'},  
'videoId': 'o44UXIVbb7I'},  
'videoId': '3mhQ5dNYTFU'},  
'videoId': '0giBtH2vQNU'},  
'videoId': 'lwbu9XTxxBI'},  
'videoId': 'DRSZh4AIBLU'},  
'videoId': 'gYqyL_efyhs'},  
'videoId': '0x0o4It4ohM'},  
'videoId': 'wSiKWoQQK3g'},  
'videoId': 'nR-KDrBFsCI'},  
'videoId': 'dGtGZMUPpIU'},  
'videoId': '51tw5CkcXxk'}
```

```

['videoId': 'EapFvLM90uo'],
['videoId': 'u9U_EVgYIog'],
['videoId': 'Z7J3IPp5rPo'],
['videoId': 'FFWthsqTm8Q'],
['videoId': 'DtsiS5fCPJc'],
['videoId': '0TQ-QxFGJkw'],
['videoId': '4qHiw6qlLDU'],
['videoId': '_gMoHoffmBs'],
['videoId': 'zHpkzl69kzo'],
['videoId': 'HZZeloZcKQM'],
['videoId': 'e2e1TrVC2SI'],
['videoId': 'SVMD4iB-Zb0'],
['videoId': 'tKWHr4EDpU8'],
['videoId': '-aeejtEdH-Q'],
['videoId': 'A1r_EXkhx1Y'],
['videoId': 'Ejk-9Z8di18'],
['videoId': '09bCNqd4ffU'],
['videoId': 'HPEBTWK7MZE'],
['videoId': 'Gj1LpymqJFA'],
['videoId': 'bWnq-bnfrqc'],
['videoId': 'lnr5ak4_fgg'],
['videoId': 'W0ZUSb8FVL8'],
['videoId': 'Xa-EtGhD3F8'],
['videoId': 'XBKVM07pzIk'],
['videoId': 'eD0m91AjwRg']]]

# chain all lists into one giant list
video_ids_list_clean= list(itertools.chain(*video_ids_list))
# only get video id value(str) and put into list
video_ids_list_clean = [d['videoId'] for d in video_ids_list_clean]

video_ids = video_ids_list_clean
video_df = get_video_details(youtube, video_ids)
video_df

      video_id channelTitle \
0      jm55pm_ZIdI    Benjamin
1      9ZVvgxLhs2M    Benjamin
2      v3NUSNQafYM    Benjamin
3      YUcdYDQJkBU    Benjamin
4      CSz67F0ZS0w    Benjamin
...     ...
2196   lnr5ak4_fgg    Tom Nash
2197   W0ZUSb8FVL8    Tom Nash
2198   Xa-EtGhD3F8    Tom Nash
2199   XBKVM07pzIk    Tom Nash
2200   eD0m91AjwRg    Tom Nash

                           title \
0           Investing Backtests That Make Me Sad

```

1 More Awful TikTok Trading Strategies?  
2 Beat the Market with Scary Youtube Finance Thu...  
3 The Stock Market Geniuses of Fiverr  
4 Testing the Biggest TikTok Daytrader's Course

...  
2196 The Contrepreneur Formula Exposed - Mike Winne...  
2197 Why Spotify bought Joe Rogan's podcast (The RE...  
2198 FED Desperately Trying to Prevent a Total Coll...  
2199 Wow! The FED just did something that may DESTR...  
2200 Who is Grant Cardone?

description \  
0 Subscribe to The Daily Upside using the link b...  
1 Make sure to get a 30 day free trial for Predi...  
2 Subscribe to The Daily Upside! \nhttps://bit.l...  
3 Subscribe to The Daily Upside! \nhttps://bit.l...  
4 https://www.patreon.com/givebenyourmoney\n\nHa...  
...  
2196 The CONtrepreneur Formula: Exposed is a video ...  
2197 Joe Rogan just signed a huge multiyear deal wi...  
2198 In economics, a depression is a sustained, lon...  
2199 The FED (Federal Reserve) recently announced t...  
2200 Is Grant Cardone Legit? - Grant Cardone is Mr....

tags  
publishedAt \  
0 None 2022-03-  
15T15:00:25Z  
1 None 2022-02-  
18T16:00:03Z  
2 None 2022-01-  
30T16:00:26Z  
3 None 2022-01-  
12T16:00:16Z  
4 None 2021-12-  
23T16:00:05Z  
...  
...  
2196 [mike winnet, mike winnet coffeezilla, mike wi... 2020-06-  
17T11:41:13Z  
2197 [joe rogan spotify podcast, joe rogan spotify,... 2020-05-  
22T13:03:38Z  
2198 [FED tries low interest rates to prevent a Dep... 2020-05-  
13T19:10:21Z  
2199 [the federal reserve, the federal reserve cras... 2020-05-  
10T12:04:10Z  
2200 [is grant cardone legit, is grant cardone a re... 2020-02-  
05T16:40:00Z

viewCount likeCount favouriteCount commentCount duration

```

definition \
0      149302      9150      None      1130    PT9M3S
hd
1      199238      10708      None      1139    PT14M10S
hd
2      193013      12214      None      974     PT10M42S
hd
3      258173      12673      None      920     PT10M8S
hd
4      548906      29512      None      2039    PT11M59S
hd
...
...
2196    8731       385       None      63      PT11M10S
hd
2197    7261       281       None      109     PT10M2S
hd
2198    1598       84        None      32      PT7M
hd
2199    4134       212       None      76      PT11M13S
hd
2200    144376      3384      None      995    PT14M54S

```

```

caption
0      false
1      false
2      false
3      false
4      false
...
2196    false
2197    false
2198    false
2199    false
2200    true

```

[2201 rows x 13 columns]

## Gathering Video comments For All Channels

```

all_comments_df = get_videos_comments(youtube, video_ids)
all_comments_df

```

|   | video_id    | comments                                          |
|---|-------------|---------------------------------------------------|
| 0 | jm55pm_ZIdI | [Subscribe to The Daily Upside!\nhttps://bit.l... |
| 1 | 9ZVvgxLhs2M | [I consistently earn massively on my investmen... |
| 2 | v3NUSNQafYM | [Subscribe to The Daily Upside! \nhttps://bit.... |
| 3 | YUcdYDQJkBU | [Subscribe to The Daily Upside!\nhttps://bit.l... |
| 4 | CSz67F0ZS0w | [I was trading options during the cold war LMA... |

```
...  ...
2195 lnr5ak4_fgg [Mike is a national treasure lolz, garden veri...
2196 W0ZUSb8FVL8 [Hit the subscribe button to join the Squad ○ ...
2197 Xa-EtGhD3F8 [Hit the subscribe button to join the Squad ○ ...
2198 XBKVM07pzIk [Hit the subscribe button to join the Squad ○ ...
2199 eD0m91AjwRg [Hit the subscribe button to join the Squad ○ ...

[2200 rows x 2 columns]
```

## Data Pre-Processing/Cleaning

To make sure that the entire analysis process run smoothly, I will now be conducting the cleaning phase of the data analysis process. Please continue reading to see how and what I changed to make the dataset clean and tidy.

```
video_df.isnull().any()

video_id      False
channelTitle   False
title         False
description   False
tags          True
publishedAt   False
viewCount     False
likeCount     True
favouriteCount True
commentCount  False
duration      False
definition    False
caption       False
dtype: bool

video_df.dtypes

video_id      object
channelTitle  object
title         object
description   object
tags          object
publishedAt  object
viewCount     object
likeCount     object
favouriteCount object
commentCount  object
duration      object
definition    object
caption       object
dtype: object
```

```

video_df.definition.unique()

array(['hd', 'sd'], dtype=object)

video_df.describe()

      video_id      channelTitle
title \
count    2201
2201
unique   2201
2200
top     jm55pm_ZIdI  The Investor Channel  How Warren Buffett Values
Stocks
freq       1
2
568

      description          tags
\
count    2201
1386
unique   2109
1175
top           [The Joseph Carlson Show, investing, stocks, s...
freq        24
85

      publishedAt  viewCount  likeCount  favouriteCount
commentCount \
count        2201      2201      2200
2201
unique      2201      2176      1700
764
top        2022-03-15T15:00:25Z      4339      175
27
freq         1
2
7
NaN
21

      duration  definition  caption
count    2201      2201      2201
unique   1315
2
2
top      PT8M1S
hd
false
freq       9
2199
2167

all_comments_df.isnull().sum()

video_id    0
comments    0
dtype: int64

```

1. Convert 'viewCount','likeCount','favouriteCount', 'commentCount' columns to numeric data types.

```
num_cols = ['viewCount', 'likeCount', 'favouriteCount', 'commentCount']
video_df[num_cols] = video_df[num_cols].apply(pd.to_numeric, errors = 'coerce', axis =1)

#Check
video_df.dtypes

video_id          object
channelTitle      object
title             object
description       object
tags              object
publishedAt       object
viewCount         float64
likeCount         float64
favouriteCount    float64
commentCount      float64
duration          object
definition        object
caption           object
dtype: object

video_df

      video_id channelTitle \
0      jm55pm_ZIdI    Benjamin
1      9ZVvgxLhs2M    Benjamin
2      v3NUSNQafYM    Benjamin
3      YUcdYDQJkBU    Benjamin
4      CSz67F0ZS0w    Benjamin
...      ...
2196   lnr5ak4_fgg    Tom Nash
2197   W0ZUSb8FVL8    Tom Nash
2198   Xa-EtGhD3F8    Tom Nash
2199   XBKVM07pzIk    Tom Nash
2200   eD0m91AjwRg    Tom Nash

                           title \
0      Investing Backtests That Make Me Sad
1      More Awful TikTok Trading Strategies?
2      Beat the Market with Scary Youtube Finance Thu...
3      The Stock Market Geniuses of Fiverr
4      Testing the Biggest TikTok Daytrader's Course
...
2196  The Entrepreneur Formula Exposed - Mike Winne...
2197  Why Spotify bought Joe Rogan's podcast (The RE...
2198  FED Desperately Trying to Prevent a Total Coll...
```

```
2199 Wow! The FED just did something that may DESTR...
2200 Who is Grant Cardone?
```

|      | description                                       | \ |
|------|---------------------------------------------------|---|
| 0    | Subscribe to The Daily Upside using the link b... |   |
| 1    | Make sure to get a 30 day free trial for Predi... |   |
| 2    | Subscribe to The Daily Upside! \nhttps://bit.l... |   |
| 3    | Subscribe to The Daily Upside! \nhttps://bit.l... |   |
| 4    | https://www.patreon.com/givebenyourmoney\n\nHa... |   |
| ...  | ...                                               |   |
| 2196 | The CONtrepreneur Formula: Exposed is a video ... |   |
| 2197 | Joe Rogan just signed a huge multiyear deal wi... |   |
| 2198 | In economics, a depression is a sustained, lon... |   |
| 2199 | The FED (Federal Reserve) recently announced t... |   |
| 2200 | Is Grant Cardone Legit? - Grant Cardone is Mr.... |   |

|                                                        | tags          |
|--------------------------------------------------------|---------------|
| publishedAt \                                          |               |
| 0                                                      | None 2022-03- |
| 15T15:00:25Z                                           |               |
| 1                                                      | None 2022-02- |
| 18T16:00:03Z                                           |               |
| 2                                                      | None 2022-01- |
| 30T16:00:26Z                                           |               |
| 3                                                      | None 2022-01- |
| 12T16:00:16Z                                           |               |
| 4                                                      | None 2021-12- |
| 23T16:00:05Z                                           |               |
| ...                                                    | ...           |
| ...                                                    |               |
| 2196 [mike winnet, mike winnet coffeezilla, mike wi... | 2020-06-      |
| 17T11:41:13Z                                           |               |
| 2197 [joe rogan spotify podcast, joe rogan spotify,... | 2020-05-      |
| 22T13:03:38Z                                           |               |
| 2198 [FED tries low interest rates to prevent a Dep... | 2020-05-      |
| 13T19:10:21Z                                           |               |
| 2199 [the federal reserve, the federal reserve cras... | 2020-05-      |
| 10T12:04:10Z                                           |               |
| 2200 [is grant cardone legit, is grant cardone a re... | 2020-02-      |
| 05T16:40:00Z                                           |               |

|              | viewCount | likeCount | favouriteCount | commentCount | duration |
|--------------|-----------|-----------|----------------|--------------|----------|
| definition \ |           |           |                |              |          |
| 0            | 149302.0  | 9150.0    | NaN            | 1130.0       | PT9M3S   |
| hd           |           |           |                |              |          |
| 1            | 199238.0  | 10708.0   | NaN            | 1139.0       | PT14M10S |
| hd           |           |           |                |              |          |
| 2            | 193013.0  | 12214.0   | NaN            | 974.0        | PT10M42S |
| hd           |           |           |                |              |          |
| 3            | 258173.0  | 12673.0   | NaN            | 920.0        | PT10M8S  |

```

hd
4      548906.0    29512.0          NaN      2039.0  PT11M59S
hd
...
...
2196     8731.0     385.0          NaN       63.0  PT11M10S
hd
2197     7261.0     281.0          NaN      109.0  PT10M2S
hd
2198     1598.0     84.0          NaN       32.0   PT7M
hd
2199     4134.0     212.0          NaN      76.0  PT11M13S
hd
2200    144376.0    3384.0          NaN      995.0 PT14M54S
hd

      caption
0      false
1      false
2      false
3      false
4      false
...
2196    false
2197    false
2198    false
2199    false
2200    true

[2201 rows x 13 columns]

```

## 2. Convert 'publishedAt' to datetime.

```

video_df['publishedAt'] = pd.to_datetime(video_df['publishedAt'],
format="%Y-%m-%dT%H:%M:%SZ")
#Check
video_df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2201 entries, 0 to 2200
Data columns (total 13 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   video_id        2201 non-null   object 
 1   channelTitle    2201 non-null   object 
 2   title           2201 non-null   object 
 3   description     2201 non-null   object 
 4   tags            1386 non-null   object 
 5   publishedAt     2201 non-null   datetime64[ns]

```

```

6  viewCount      2201 non-null   float64
7  likeCount      2200 non-null   float64
8  favouriteCount 0 non-null    float64
9  commentCount    2201 non-null   float64
10 duration        2201 non-null   object
11 definition      2201 non-null   object
12 caption         2201 non-null   object
dtypes: datetime64[ns](1), float64(4), object(8)
memory usage: 223.7+ KB

```

### 3. Create new column from 'publishedAt', containing name of day.

```

# Help: https://stackoverflow.com/questions/29096381/num-day-to-name-day-with-pandas
video_df['publishedDayName'] = video_df['publishedAt'].dt.day_name()
video_df['publishedMonthName'] =
video_df['publishedAt'].dt.month_name()

#Check
video_df.head(1)

      video_id channelTitle           title \
0  jm55pm_ZIdI      Benjamin  Investing Backtests That Make Me Sad

                               description  tags \
0  Subscribe to The Daily Upside using the link b...  None

      publishedAt  viewCount  likeCount  favouriteCount
commentCount \
0  2022-03-15 15:00:25     149302.0      9150.0          NaN
1130.0

      duration  definition  caption  publishedDayName  publishedMonthName
0    PT9M3S          hd    false        Tuesday            March

```

### 4. Convert duration column to seconds(float).

```

# convert duration column to seconds with isodate
# help: https://stackoverflow.com/questions/16742381/how-to-convert-youtube-api-duration-to-seconds
# time delta help:
https://pandas.pydata.org/docs/user\_guide/timedeltas.html

video_df['durationSec'] = video_df['duration'].apply(lambda
x:isodate.parse_duration(x))
video_df['durationSec'] =
video_df['durationSec'].astype('timedelta64[s]')
#check
video_df.head(1)

```

```

        video_id channelTitle           title \
0 jm55pm_ZIdI      Benjamin Investing Backtests That Make Me Sad

   description  tags \
0 Subscribe to The Daily Upside using the link b... None

            publishedAt  viewCount  likeCount favouriteCount
commentCount \
0 2022-03-15 15:00:25    149302.0      9150.0          NaN
1130.0

duration definition caption publishedDayName publishedMonthName
durationSec
0 PT9M3S         hd   false       Tuesday       March
543.0

```

## 5. Create column to count number of tags.

```

# len(video_df['tags'][2195]) - produced a number
# len(video_df['tags'][0]) - produced a Nonetype error; must address
video_df['tagsCount'] = video_df['tags'].apply(lambda x: 0 if x is
None else len(x))

#check
video_df.tail()

        video_id channelTitle \
2196 lnr5ak4_fgg      Tom Nash
2197 W0ZUSb8FVL8      Tom Nash
2198 Xa-EtGhD3F8      Tom Nash
2199 XBKVM07pzIk      Tom Nash
2200 eD0m91AjwRg      Tom Nash

   title \
2196 The Contrepreneur Formula Exposed - Mike Winne...
2197 Why Spotify bought Joe Rogan's podcast (The RE...
2198 FED Desperately Trying to Prevent a Total Coll...
2199 Wow! The FED just did something that may DESTR...
2200 Who is Grant Cardone?

   description \
2196 The CONtrepreneur Formula: Exposed is a video ...
2197 Joe Rogan just signed a huge multiyear deal wi...
2198 In economics, a depression is a sustained, lon...
2199 The FED (Federal Reserve) recently announced t...
2200 Is Grant Cardone Legit? - Grant Cardone is Mr...

   tags
publishedAt \
2196 [mike winnet, mike winnet coffeezilla, mike wi... 2020-06-17

```

```

11:41:13
2197 [joe rogan spotify podcast, joe rogan spotify,... 2020-05-22
13:03:38
2198 [FED tries low interest rates to prevent a Dep... 2020-05-13
19:10:21
2199 [the federal reserve, the federal reserve cras... 2020-05-10
12:04:10
2200 [is grant cardone legit, is grant cardone a re... 2020-02-05
16:40:00

      viewCount likeCount favouriteCount commentCount duration
definition \
2196    8731.0     385.0           NaN        63.0 PT11M10S
hd
2197    7261.0     281.0           NaN       109.0 PT10M2S
hd
2198    1598.0     84.0            NaN        32.0   PT7M
hd
2199    4134.0     212.0           NaN        76.0 PT11M13S
hd
2200   144376.0    3384.0          NaN       995.0 PT14M54S
hd

      caption publishedDayName publishedMonthName durationSec
tagsCount
2196  false       Wednesday        June       670.0
33
2197  false       Friday          May        602.0
25
2198  false       Wednesday        May        420.0
16
2199  false       Sunday          May        673.0
17
2200  true        Wednesday        February  894.0
21

```

#### 6. Convert publishedDayName column to categorical variable.

```

days_ordered = ['Monday', 'Tuesday', 'Wednesday', 'Thursday',
'Friday', 'Saturday', 'Sunday']
days_ordered_var = pd.api.types.CategoricalDtype(ordered = True,
categories = days_ordered)
video_df.publishedDayName =
video_df.publishedDayName.astype(days_ordered_var)

```

#### 7. Convert publishedMonthName column to categorical variable.

```

months_ordered = ['January', 'February', 'March', 'April', 'May',
'June', 'July'],

```

```

'August', 'September', 'October', 'November', 'December']
months_ordered_var = pd.api.types.CategoricalDtype(ordered = True,
categories = months_ordered)
video_df.publishedMonthName =
video_df.publishedMonthName.astype(months_ordered_var)

```

#### 8. Drop favouriteCount column which is full of nulls.

```

video_df.drop(columns = ['favouriteCount'], inplace = True)
# check
video_df.head()

      video_id channelTitle \
0  jm55pm_ZIdI    Benjamin
1  9ZVvgxLhs2M    Benjamin
2  v3NUSNQafYM    Benjamin
3  YUcdYDQJkBU    Benjamin
4  CSz67F0ZS0w    Benjamin

                           title \
0  Investing Backtests That Make Me Sad
1  More Awful TikTok Trading Strategies?
2  Beat the Market with Scary Youtube Finance Thu...
3  The Stock Market Geniuses of Fiverr
4  Testing the Biggest TikTok Daytrader's Course

                           description  tags \
0  Subscribe to The Daily Upside using the link b...  None
1  Make sure to get a 30 day free trial for Predi...  None
2  Subscribe to The Daily Upside! \nhttps://bit.l...  None
3  Subscribe to The Daily Upside! \nhttps://bit.l...  None
4  https://www.patreon.com/givebenyourmoney\n\nHa...  None

      publishedAt  viewCount  likeCount  commentCount  duration \
0  2022-03-15 15:00:25  149302.0   9150.0     1130.0  PT9M3S
1  2022-02-18 16:00:03  199238.0   10708.0     1139.0  PT14M10S
2  2022-01-30 16:00:26  193013.0   12214.0      974.0  PT10M42S
3  2022-01-12 16:00:16  258173.0   12673.0      920.0  PT10M8S
4  2021-12-23 16:00:05  548906.0   29512.0     2039.0  PT11M59S

      definition  caption  publishedDayName  publishedMonthName  durationSec \
0        hd      false        Tuesday            March          543.0
1        hd      false        Friday           February          850.0
2        hd      false       Sunday           January          642.0
3        hd      false      Wednesday          January          608.0

```

|           |    |       |          |          |       |
|-----------|----|-------|----------|----------|-------|
| 4         | hd | false | Thursday | December | 719.0 |
| tagsCount |    |       |          |          |       |
| 0         |    | 0     |          |          |       |
| 1         |    | 0     |          |          |       |
| 2         |    | 0     |          |          |       |
| 3         |    | 0     |          |          |       |
| 4         |    | 0     |          |          |       |

## 9. Create new column for title length.

```
video_df['title_length'] = video_df['title'].apply(lambda x: len(x))
video_df.head()
```

|                                                                    | video_id    | channelTitle | title                                             | description                                       | tags  | duration |
|--------------------------------------------------------------------|-------------|--------------|---------------------------------------------------|---------------------------------------------------|-------|----------|
| 0                                                                  | jm55pm_ZIdI | Benjamin     | Investing Backtests That Make Me Sad              | Subscribe to The Daily Upside using the link b... | None  | PT9M3S   |
| 1                                                                  | 9ZVvgxLhs2M | Benjamin     | More Awful TikTok Trading Strategies?             | Make sure to get a 30 day free trial for Predi... | None  | PT14M10S |
| 2                                                                  | v3NUSNQafYM | Benjamin     | Beat the Market with Scary Youtube Finance Thu... | Subscribe to The Daily Upside! \nhttps://bit.l... | None  | PT10M42S |
| 3                                                                  | YUcdYDQJkBU | Benjamin     | The Stock Market Geniuses of Fiverr               | Subscribe to The Daily Upside! \nhttps://bit.l... | None  | PT10M8S  |
| 4                                                                  | CSz67F0ZS0w | Benjamin     | Testing the Biggest TikTok Daytrader's Course     | https://www.patreon.com/givebenyourmoney\n\nHa... | None  | PT11M59S |
| definition caption publishedDayName publishedMonthName durationSec |             |              |                                                   |                                                   |       |          |
| 0                                                                  | hd          | false        | Tuesday                                           | March                                             | 543.0 |          |
| 1                                                                  | hd          | false        | Friday                                            | February                                          | 850.0 |          |
| 2                                                                  | hd          | false        | Sunday                                            | January                                           | 642.0 |          |

|                        |    |       |           |          |       |
|------------------------|----|-------|-----------|----------|-------|
| 3                      | hd | false | Wednesday | January  | 608.0 |
| 4                      | hd | false | Thursday  | December | 719.0 |
| tagsCount title_length |    |       |           |          |       |
| 0                      | 0  | 36    |           |          |       |
| 1                      | 0  | 37    |           |          |       |
| 2                      | 0  | 53    |           |          |       |
| 3                      | 0  | 35    |           |          |       |
| 4                      | 0  | 45    |           |          |       |

## Exploratory Data Analysis

### Total View Count By Channel

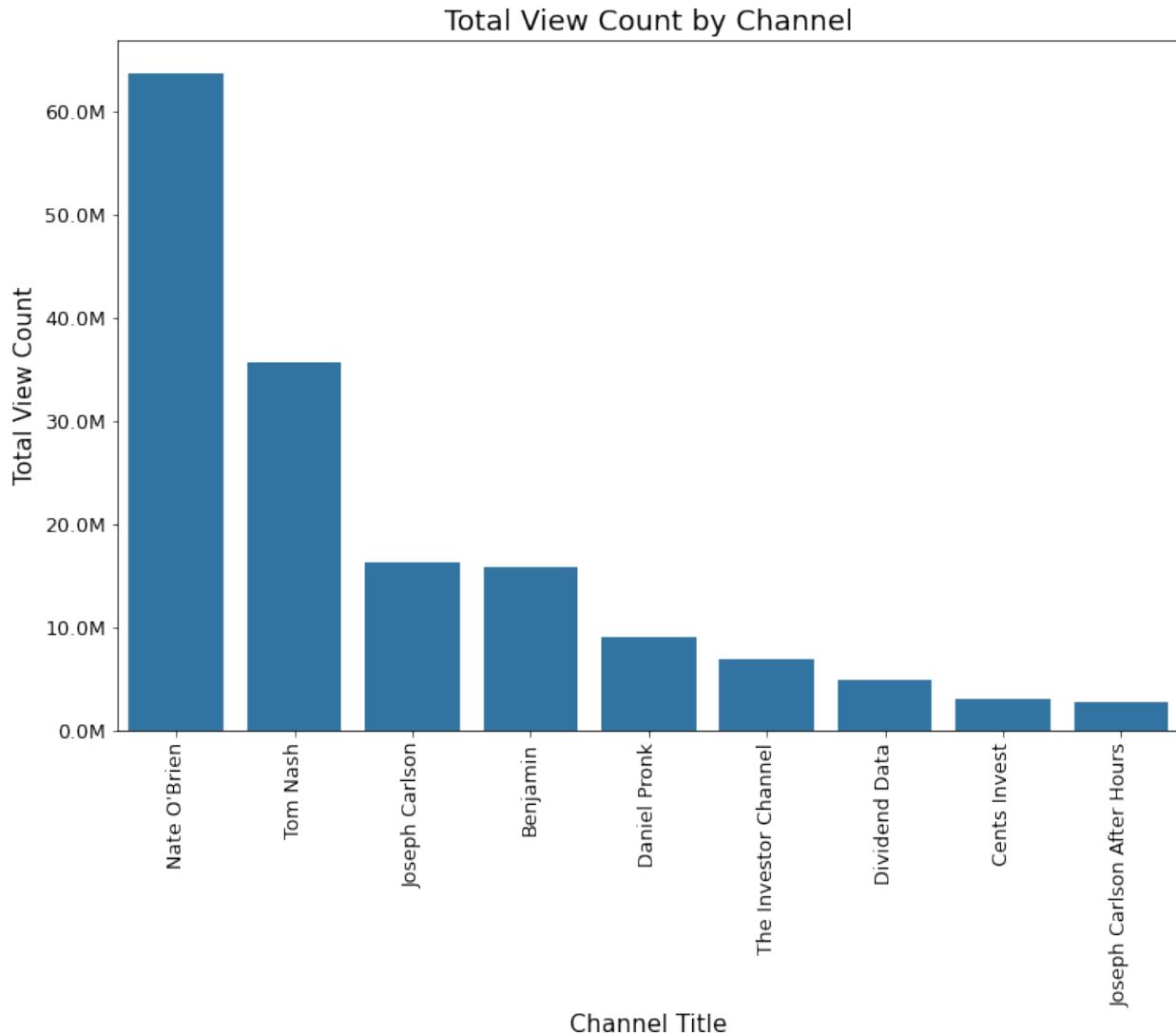
```
# Convert big numbers into string for the y-axis
# Help: https://stackoverflow.com/questions/40511476/how-to-properly-
use-funcformattefunc
def millions(x, pos):
    """
    INPUT:
    x: numerical value
    pos: tick position

    OUTPUT: formatted string of % (x*1e-6) with 'M' to represent
millions
    """
    return '%1.1fM' % (x*1e-6)

formatter = FuncFormatter(millions)

base_color = sb.color_palette()[0]
plt.figure(figsize = (12, 8))
ax = sb.barplot(x = 'channelTitle', y = 'viewCount',
                 data = video_df.groupby('channelTitle')
['viewCount'].sum().sort_values(ascending = False).reset_index(),
                 color = base_color)
ax.yaxis.set_major_formatter(formatter)

plt.title('Total View Count by Channel', fontsize = 18)
plt.xticks(rotation = 90, fontsize = 12.5)
plt.yticks(fontsize = 12.5)
plt.xlabel('Channel Title', fontsize = 15)
plt.ylabel('Total View Count', fontsize = 15);
```



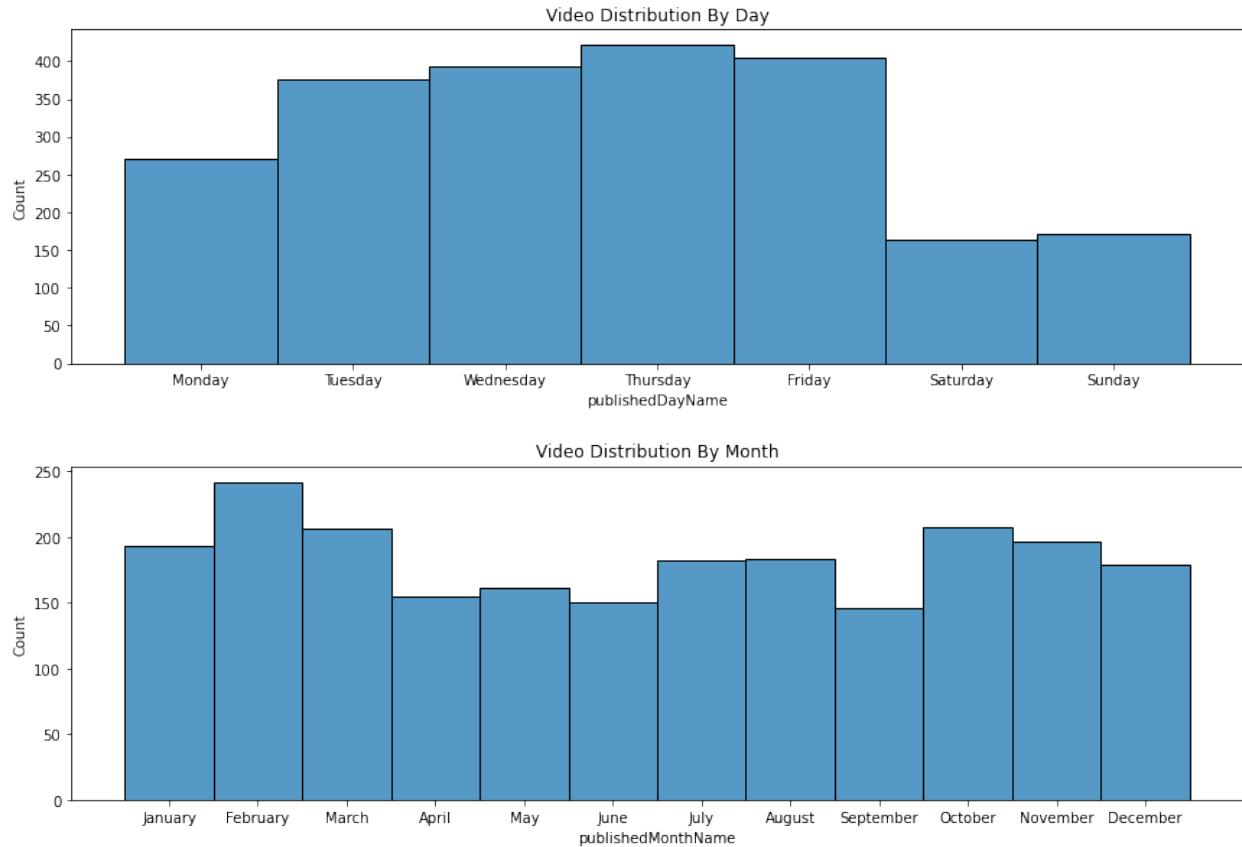
**My first visualization has to do with viewing the sum of all view counts based on the channel. From the barplot above, we can see that Nate O'Brien has the most of them all with over 60 million views. That channel alone is almost double that of the 2nd place, which is Tom Nash.**

#### Video Upload Distribution By Day And Month

```
fig, ax = plt.subplots(2, 2, figsize = [12, 8])
fig.tight_layout(h_pad=5)

plt.subplot(2, 1, 1)
sb.histplot(data = video_df, x = "publishedDayName")
plt.title('Video Distribution By Day')

plt.subplot(2, 1, 2)
sb.histplot(data = video_df, x = "publishedMonthName")
plt.title('Video Distribution By Month');
```



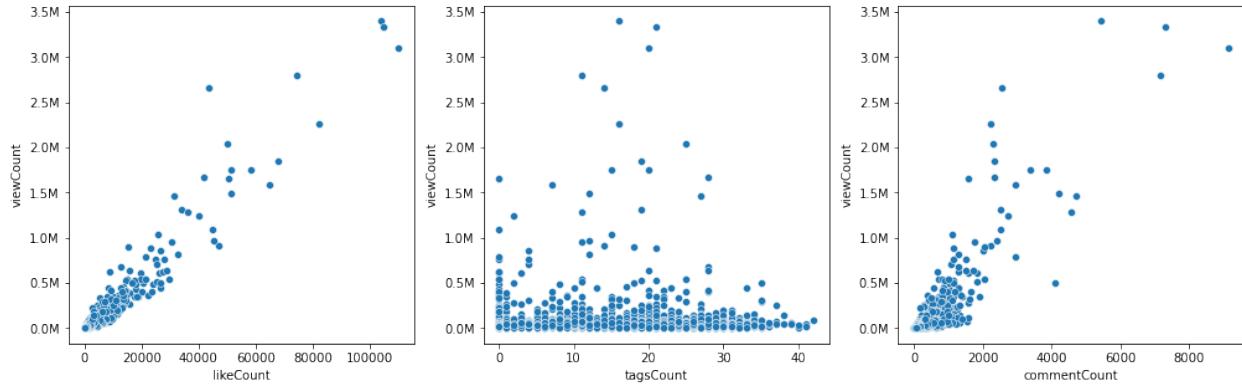
Next, I analyzed the distribution all the videos based on day and month. In the first plot, we can see that Thursday is the day with the most video uploads, but it doesn't seem like the rest of the weekdays are that far behind. However, if we take a look at the weekends, we find that the output is cut in half! This makes sense since the stock market is closed on the weekends. In the next plot, we see the video upload distribution by month. Although it seems like the upload rate is pretty consistent, the first and last 3 months experience a jolt of video uploads with all of them well above 150.

#### Views vs. likes, comments & tags

```
fig, ax = plt.subplots(1,3, figsize = [17, 5])

sb.scatterplot(data = video_df, x = 'likeCount', y = 'viewCount', ax = ax[0])
sb.scatterplot(data = video_df, x = 'tagsCount', y = 'viewCount', ax = ax[1])
sb.scatterplot(data = video_df, x = 'commentCount', y = 'viewCount',
ax = ax[2])

# convert scientific notations on y-axis to millions
for i in range(3):
    ax[i].yaxis.set_major_formatter(formatter);
```

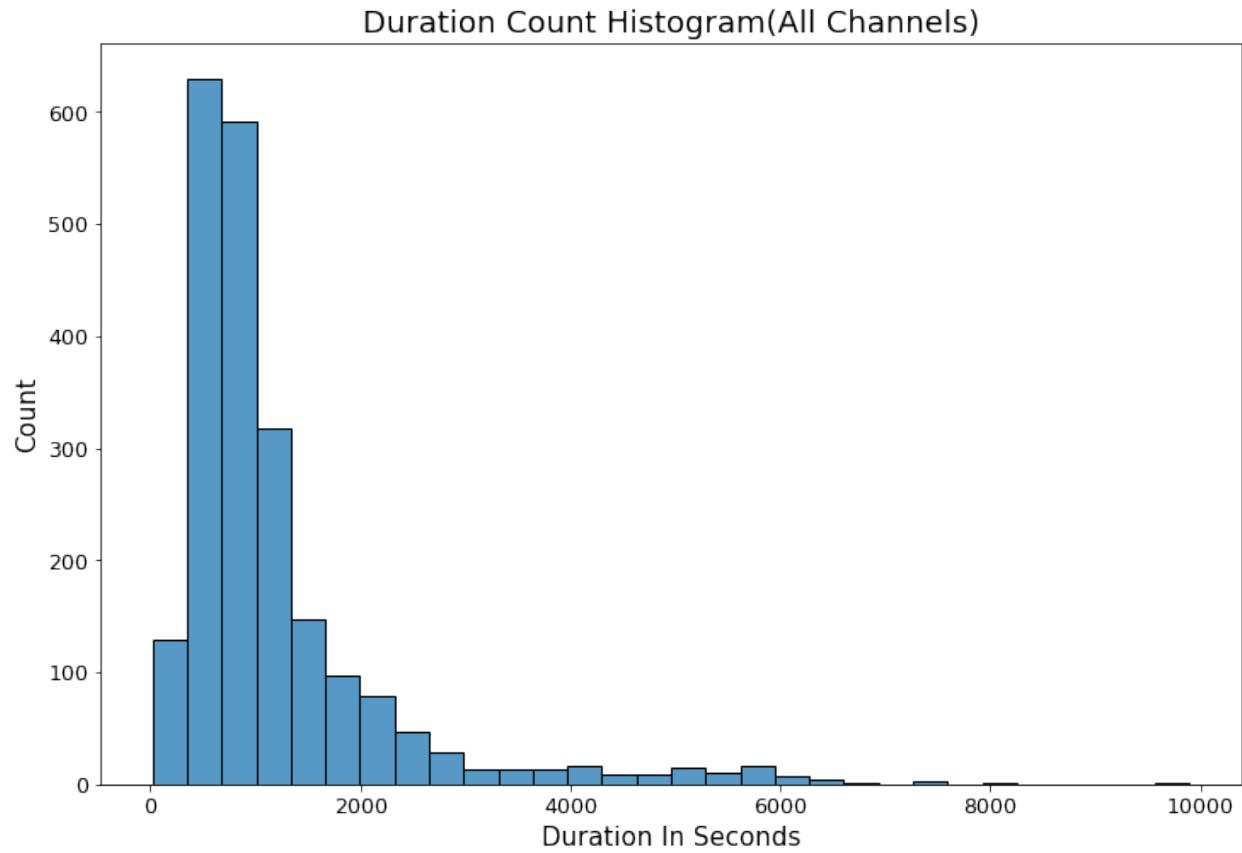


In the first scatterplot, where we compare the relationship between likeCount and viewCount, we find a positive and strong relationship between the 2 variables. The more views one has, the more likes the video will most likely have. This makes sense, but there is a limitation to this. One can have a lot of views-say 500k views- and a lot of dislikes. Sadly, Youtube has removed that feature, so we won't be able to find out. In the next plot, we can see a scatterplot that examines the relationship between viewCount and tagsCount. It literally forms a straight line across the bottom, which indicates that there isn't much of a relationship. In other words, it doesn't seem like tagsCount affects viewCount at all. What's interesting is that we see that most of the outliers are in-between the 10-30 tagsCount. In the final scatterplot, we see a positive relationship between viewCount and commentCount. This makes sense for the same reason the first scatter plot makes sense. However, I would characterize the strength as moderate because it seems like the data points are more spread out than the first plot.

#### Distribution Of Video Duration

```
plt.figure(figsize = (12, 8))
sb.histplot(data=video_df, x="durationSec", bins = 30)

plt.title('Duration Count Histogram(All Channels)', fontsize = 18)
plt.xticks(fontsize = 12.5)
plt.yticks(fontsize = 12.5)
plt.xlabel('Duration In Seconds', fontsize = 15)
plt.ylabel('Count', fontsize = 15);
```

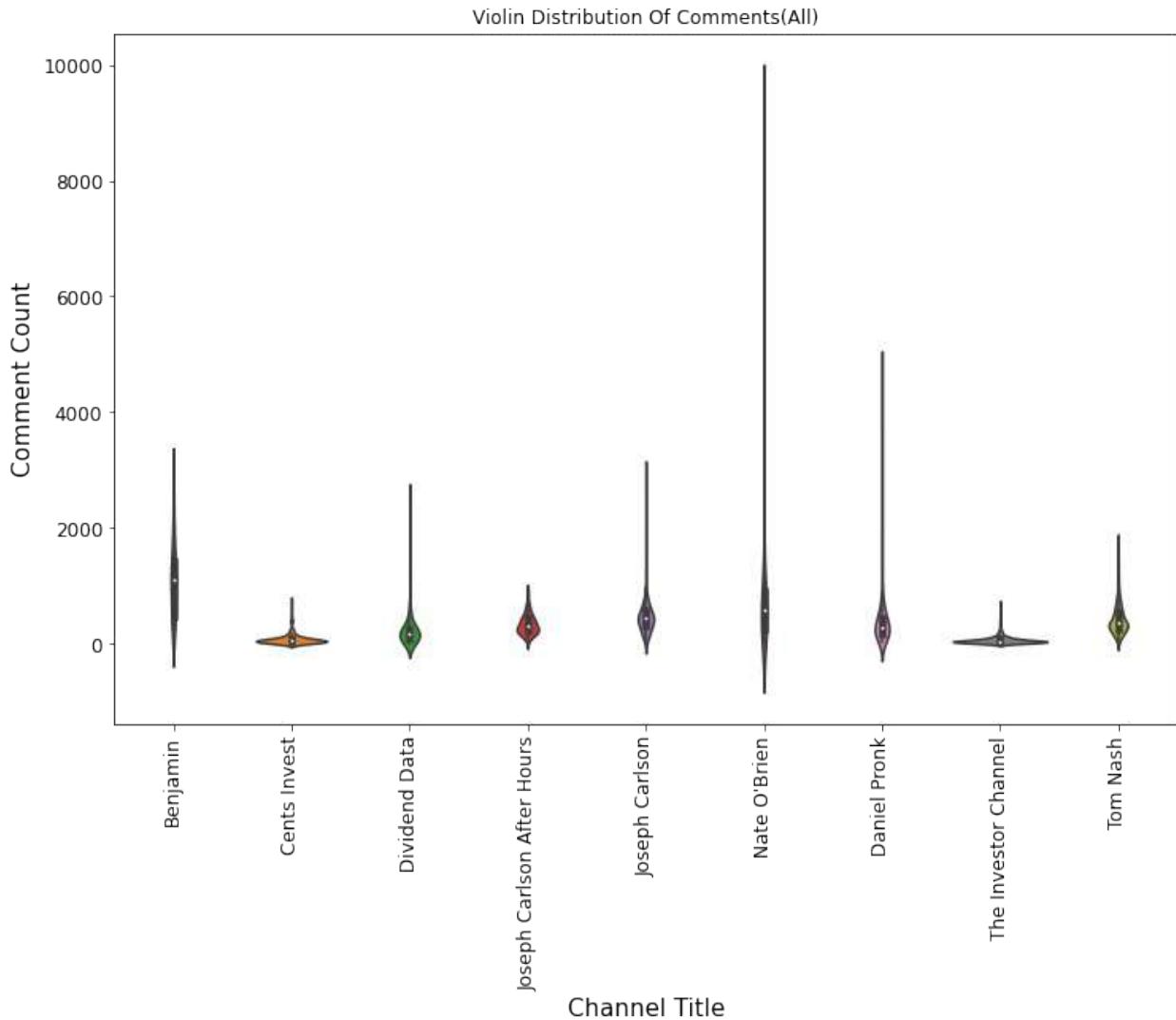


In this histogram plot, we see that the majority of videos uploaded from all the channels are less than 2000 seconds which means that it skews to the left. The longest video is almost 10,000 seconds long!

#### Comment Distribution Per Video

```
fig = plt.figure(figsize = (12, 8))
ax = sb.violinplot(data = video_df, x = 'channelTitle', y =
'commentCount')

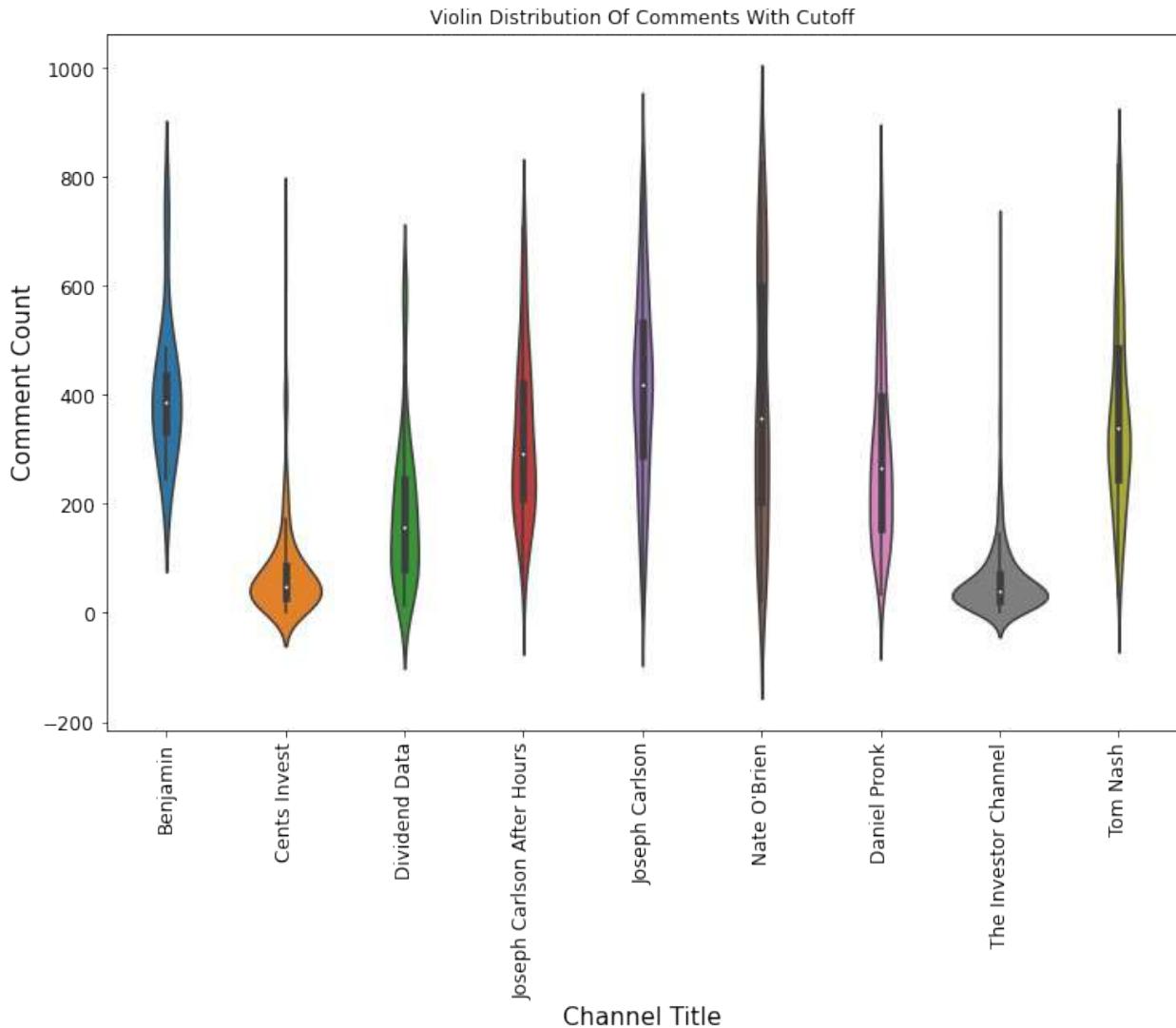
plt.xticks(rotation = 90, fontsize = 12)
plt.yticks(fontsize = 12)
plt.xlabel("Channel Title", fontsize = 15)
plt.ylabel("Comment Count", fontsize = 15)
plt.title("Violin Distribution Of Comments(All)");
```



\*\*There are tons of outliers for Nate O'Brien which make it a little hard to analyze, so we will re-plot it. The only difference is that we will cut it off at Q3\*2.\*\*

```
# establish cutoff point, get rid of outliers for better viewing
double_Q3 = (video_df['commentCount'].quantile(0.75))*2

#plot
fig = plt.figure(figsize = (12, 8))
ax = sb.violinplot(data = video_df[video_df.commentCount < double_Q3],
x = 'channelTitle', y = 'commentCount')
plt.xticks(rotation = 90, fontsize = 12)
plt.yticks(fontsize = 12)
plt.xlabel("Channel Title", fontsize = 15)
plt.ylabel("Comment Count", fontsize = 15)
plt.title("Violin Distribution Of Comments With Cutoff");
```



**Now that we have a much better look, we can properly analyze this violin plot of commentCount based on the channel. Please keep in mind that some channels and videos might have a lot more views than others. Look at how thin the violin plot is for Nate O'Brien. This is an indication of a large variance in their comments, which means some of the channel's videos get a lot more comments than others. We can say the same for both Joesph Carlson's channels and Tom Nash, but they seem to be more consistent. Other than those, all the other channels are more consistent. In other words, the wider the violin plot, the more consistent.**

#### View Distribution Per Video

```
# function similair to the million() function, but for thousands
def thousands(x, pos):
    """
    INPUT:
    x: numerical value
    pos: tick position
```

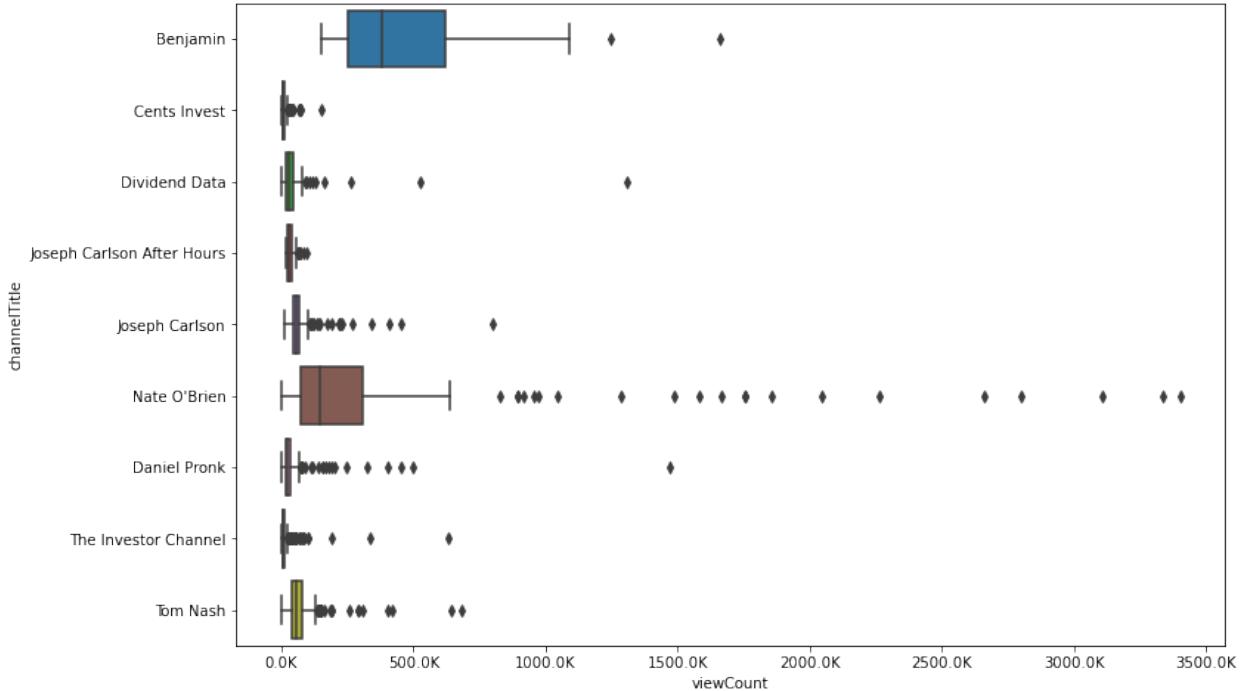
```

OUTPUT: formatted string of % ( $x \cdot 10^{-3}$ ) with K to represent
thousands
"""
    return '%1.1fK' % (x*1e-3)

formatter = FuncFormatter(thousands)

#plot
fig = plt.figure(figsize = (12, 8))
ax = sb.boxplot(data = video_df, y = 'channelTitle', x = 'viewCount')
ax.xaxis.set_major_formatter(formatter)

```



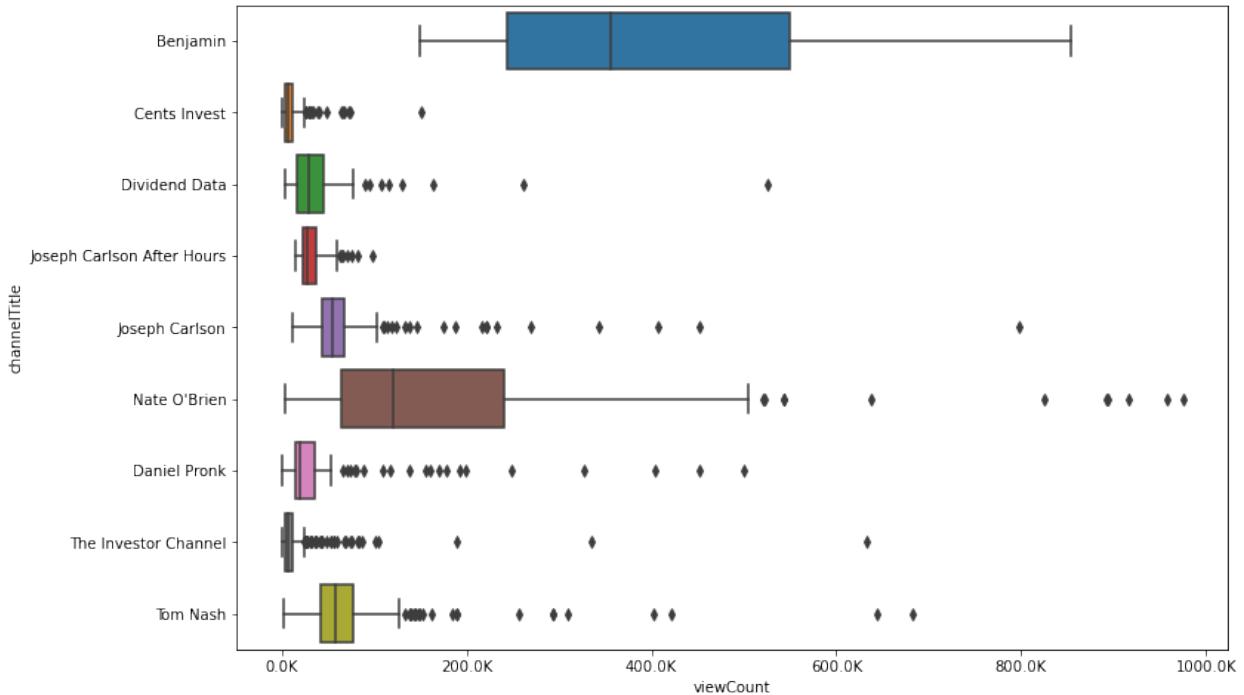
Much like the last one, it's good to get a overview, but we should make a plot with a cutoff to get rid of the outliers.

```

#plot
cutoff = 10000000

fig = plt.figure(figsize = (12, 8))
ax = sb.boxplot(data = video_df[video_df.viewCount < cutoff], y =
'channelTitle', x = 'viewCount')
ax.xaxis.set_major_formatter(formatter)

```



In the second bar plot, we can clearly see that Benjamin has the highest median out of all the channels with a figure close to 400k. Nate O'Brien comes in second. Other than those two, we can see that the IQR for the other channels is much lower. Most of these channels' Q4s do not even cross 100k; as a result, we see a lot of outliers. For example, take a look at the Investor Channel, where there are tons of outliers.

#### Average View Count Per Channel Based On Upload Day

```
channel_view_means = video_df.groupby(['channelTitle',
'publishedDayName'])['viewCount'].mean().reset_index(name = 'viewAvg')
channel_view_means = channel_view_means.pivot(columns =
'channelTitle', index = 'publishedDayName', values = 'viewAvg')

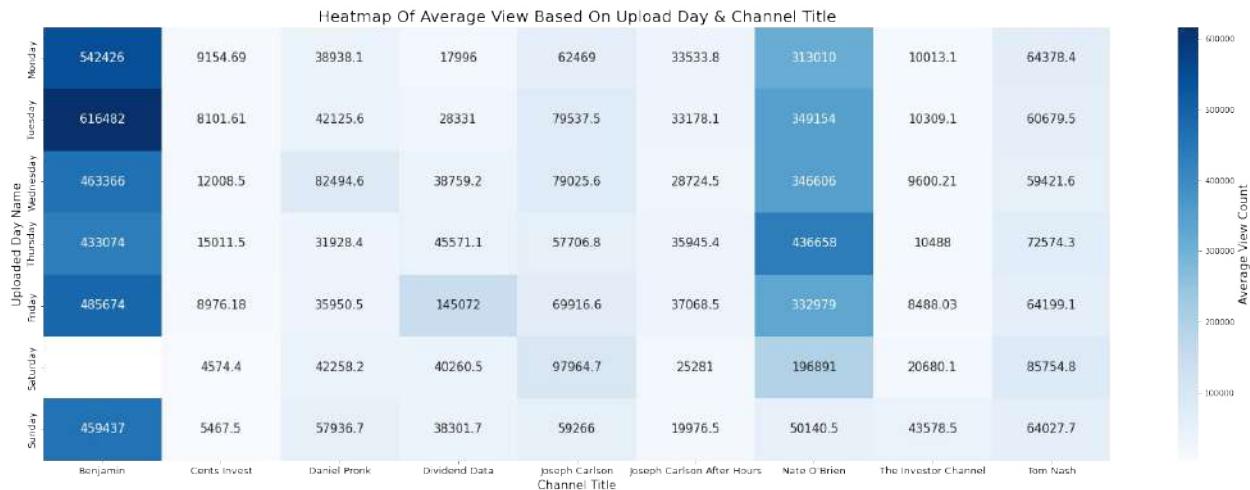
fig = plt.figure(figsize = (30,10))
ax = sb.heatmap(channel_view_means, annot = True, cmap='Blues',
                 fmt='g', annot_kws = {'size': 15}, cbar_kws =
{'label': 'Average View Count'})

plt.title('Heatmap Of Average View Based On Upload Day & Channel
Title',
          fontsize = 20)
plt.xticks(rotation = 360, fontsize = 12.5)
plt.yticks(fontsize = 12.5)
plt.xlabel("Channel Title", fontsize = 15)
plt.ylabel("Uploaded Day Name", fontsize = 15)

# setting color bar fontsize
# help: https://stackoverflow.com/questions/48586738/seaborn-heatmap-
```

```
colorbar-label-font-size
```

```
ax.figure.axes[-1].yaxis.label.set_size(15);
```



This heatmap displays each channel's average view based on the upload day. Starting from the highest mean, we see that Benjamin has some of the darkest shades of blue, which indicates the videos uploaded on those given days have the highest views out of all the channels. However, we also see a blank box also; this indicates that Benjamin has never uploaded a video on a Saturday. Benjamin has never averaged below 400k views for any given upload day. What's amazing is that the highest mean is 614k and the video was uploaded on a Tuesday! Keep in mind that Benjamin's channel started in 2021. Not enough time has passed to analyze the performance over time, but we can say that he went viral very quickly! Next, we see that the Nate O'Brien channel has the second darkest shades of blue. Most are not darker than any of Benjamin's. The highest mean for the Nate O'Brien channel is 437k for videos uploaded on Thursday. Cents Invest seems to average the lowest viewCount regardless of the day. Their highest average is only 15k and most of their averages are only 4 digits.

## Video Title WordCloud

```
# function for plotting wordcloud
# word cloud help: https://www.youtube.com/watch?v=f1TJXu5H8ZM
def plot_youtube_cloud(data):
    """
    INPUT:
    data: Input data structure.
    title: (str) Title of Word Cloud

    OUTPUT:
    No Output
    """

    # import youtube outline
    image = imageio.imread('youtube_image.png')
    wordcloud = WordCloud(background_color = 'white',
                          mask = image,
```

```
stopwords = stop_words,
max_words = 200,
max_font_size = 70,
scale = 3,
random_state = 1,
# coloring help:
https://github.com/amueller/word\_cloud/issues/185
color_func=lambda *args, **kwargs:
"red").generate(str(data))
fig = plt.figure(1, figsize = (20,20), dpi=80)
plt.axis('off')

plt.imshow(wordcloud)
plt.show()

# setting up stop words
stop_words = set(stopwords.words('english'))
# word tokenization help: https://stackabuse.com/removing-stop-words-from-strings-in-python/
video_df['title_no_stopwords'] = video_df['title'].apply(lambda x: [w
for w in word_tokenize(x) if w not in stop_words])

# append all words 'title_no_stopwords' into single list
all_words = list([a for b in video_df['title_no_stopwords'].tolist()
for a in b])
# join all words together in list into single string
all_words_str = ' '.join(all_words)
plot youtube cloud(all words str)
```



In the final analysis above, we see a word cloud in the shape of the Youtube logo. Most of the popular words in the video title are stock-related (no surprise). We see the following words/phrases have been used the most: 'Stock', 'Stock Analysis', and 'Buy'. Furthermore, as we examine more of the words, we find that the most-used words after those 3 are basically name-dropping, companies' names, investing terminologies, and stock ticker symbols.

## Comments WordCloud

```
# join list of comments into a single string
for i in range(len(all_comments_df['comments'])):
    all_comments_df['comments'][i] = ""
    ".join(all_comments_df['comments'][i])
all_comments_df['comments'].head()

0    Subscribe to The Daily Upside!\nhttps://bit.ly...
1    I consistently earn massively on my investment...
2    Subscribe to The Daily Upside! \nhttps://bit.l...
3    Subscribe to The Daily Upside!\nhttps://bit.ly...
4    I was trading options during the cold war LMAO...
Name: comments, dtype: object

# new column for words not in stopwords
all_comments_df['comments_no_stopwords'] =
all_comments_df['comments'].apply(lambda x: [w for w in
word_tokenize(x)

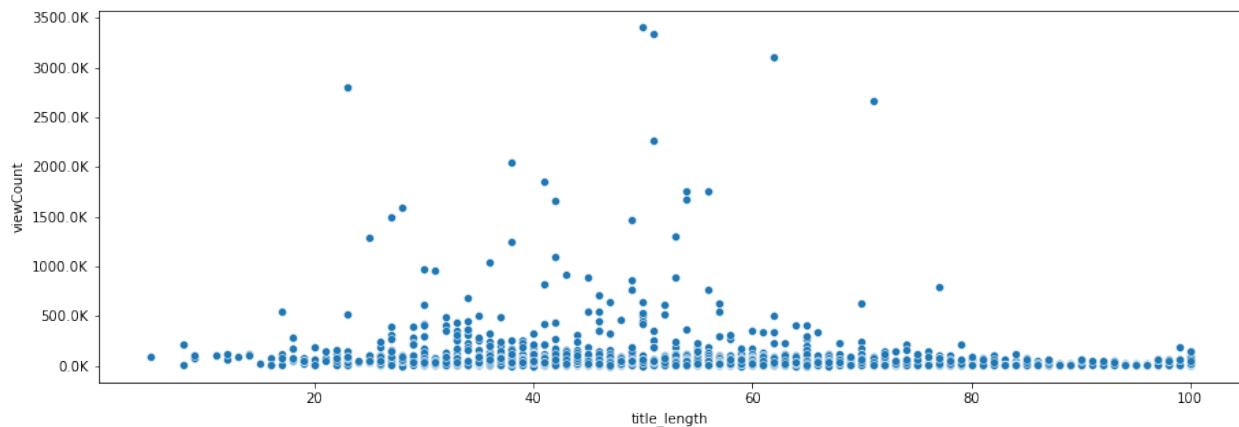
if w not in stop_words])
# append all words 'title_no_stopwords' into single list
all_words = list([a for b in
all_comments_df['comments_no_stopwords'].tolist() for a in b])
# join all words together in list into single string
all_words_str = ' '.join(all_words)
plot_youtube_cloud(all_words_str)
```



It seems like there are quite a number of similarities between the most-frequently-used words in the title and comments, but also a lot of differences. Some of the most frequently used words come as no surprise given the topic. These words include "stock", "company", "Thank" and various stock/company names. From the word cloud above, we can see that Tesla is the most talked about company in the comment section. Some of the differences that we spot between the title and comment word clouds are the verbs. In this word cloud, we can see that the most popular verbs are 'would' and 'think'. In the title word cloud, we saw that the most used verb is 'buy'.

## Relationship between viewCount & title length

```
fig, ax = plt.subplots(figsize = [15, 5])
sb.scatterplot(data = video_df, x = 'title_length', y = 'viewCount')
ax.yaxis.set_major_formatter(formatter);
```



**From the scatterplot above, we can see that there is no relationship between the title length and word count. However, we can also see that the most viewed videos have between 20-80 characters.**

## Conclusion

In this project, we explored the video data collected from 9 different financial Youtube channels and made a number of interesting discoveries about the videos and the topics covered on these channels. To do a quick recap, we found the following:

- The more likes and comments a video has, the more views it'll most likely have. However, it seems the number of 'like' has a stronger positive relationship with the view count, which means that the more views a video has, the more people will like it.
- Most financial youtube videos are uploaded during the weekdays. Also, most were uploaded in the first 3 and last 3 months of the year.
- Most of these financial youtube videos have 0-30 tags.
- A majority of title length falls between 20-80, but anywhere between 20-60 tags seems to increase your chance at increasing your viewCount.
- The most frequently used word in the title and comment section is "stock". Furthermore, the comment section seems to be positive. Two words in the comment nordcloud that stood out to me were 'Thank' and 'think'. The second one is interesting because the commentator might be asking for the content creator's opinion on something which creates room for more content to be made.

Please do not take these findings seriously. Here are some limitations that this project has presented:

- There are only a total of 2201 videos in this dataset.
- These videos are collected from 9 YouTubers. Therefore, this isn't a good representation of all financial YouTube channels where there could be thousands.
- I only added the first 10 comments into the comments dataset.
- There are many factors that I haven't included in my analysis. Perhaps, the biggest one is the quality of the content itself.



**project**

**11**

**LLondon Crime Hypothesis Testing  
(2008-2016)**

# Summary of Project

In this project, I am interested in working to understand the crime rate in London, England. My goal is to work through this notebook to understand violent crime rates and when they are likely to occur throughout the year. This dataset contains all crimes (non-violent and violent) committed between 2008 and 2016. However, the nature of the crime-violent or non-violent- is not specified in this dataset, so we will need to deal with that during the preprocessing phase. We will also need to specify which months are the ones when daylight saving is in effect.

There are a total of 2 notebooks dedicated to this project. This first notebook(London Crime Hypothesis Testing.ipynb) is dedicated to testing the individual factors(borough/location & daylight saving) that I think may affect the violent crime rate while the second notebook(London Crime Hypothesis Testing Part II.ipynb) to testing the interactions between location/borough and daylight saving for any significant effect on the rate.

My initial assumption is that violent crime rates increase when daylight saving is not in effect, which means that the night is longer. Therefore, our hypotheses are:

Null Hypothesis: The difference between the violent crime rates when daylight saving is and isn't in effect is less than or equal to 0.

Alternative Hypothesis: The difference between the violent crime rates when daylight saving is and isn't in effect is greater than 0.

We will also assume that a Type 1 Error rate of 0.05.

## Installation

Packages used:

- \* Matplotlib
- \* Numpy
- \* Pandas
- \* statsmodels.api
- \* random
- \* statsmodels.stats.proportion

## Dataset

If you are interested in downloading the dataset, please click on [this Kaggle link](#). There are a total of 13490604 rows in the dataset and 7 columns. Columns include variables such as month, LSOA borough, and major/minor category from Jan 2008-Dec 2016.

# London Crimes - Hypothesis Testing

In this project, I am interested in working to understand the crime rate in London, England. My goal is to work through this notebook to understand violent crime rates and when they are likely to occur throughout the year. This dataset contains all crimes (non-violent and violent) committed between 2008 and 2016. However, the nature of the crime-violent or non-violent- is not specified in this dataset, so we will need to deal with that during the preprocessing phase. We will also need to specify which months are the ones when daylight saving is in effect. My initial assumption is that violent crime rates increase when daylight saving is not in effect, which means that the night is longer. If you are interested in downloading the dataset, please click on [this Kaggle link](#).

```
import pandas as pd
import numpy as np
import random
import matplotlib.pyplot as plt
from statsmodels.stats.proportion import proportions_ztest
import statsmodels.api as sm

%matplotlib inline

# read csv and import data
df = pd.read_csv('london_crime.csv')

# make a copy
crime_df = df.copy()
```

## Examining Dataset

### 1. Understanding The Dataset

```
crime_df.head()
```

```
   lsoa_code      borough          major_category \
0  E01001116     Croydon            Burglary
1  E01001646    Greenwich  Violence Against the Person
2  E01000677     Bromley  Violence Against the Person
3  E01003774   Redbridge            Burglary
4  E01004563   Wandsworth           Robbery

          minor_category  value  year  month
0  Burglary in Other Buildings    0  2016    11
1                  Other violence    0  2016    11
2                  Other violence    0  2015     5
3  Burglary in Other Buildings    0  2016     3
4          Personal Property    0  2008     6
```

```
print(f"There are a total of {crime_df.shape[0]} rows in the dataset.")
```

```
There are a total of 13490604 rows in the dataset.
```

```
crime_df.borough.value_counts()
```

|                        |        |
|------------------------|--------|
| Croydon                | 602100 |
| Barnet                 | 572832 |
| Ealing                 | 549396 |
| Bromley                | 523908 |
| Lambeth                | 519048 |
| Enfield                | 511164 |
| Wandsworth             | 498636 |
| Brent                  | 490644 |
| Lewisham               | 485136 |
| Southwark              | 483300 |
| Newham                 | 471420 |
| Redbridge              | 445716 |
| Hillingdon             | 442584 |
| Greenwich              | 421200 |
| Hackney                | 417744 |
| Haringey               | 413856 |
| Tower Hamlets          | 412128 |
| Waltham Forest         | 406296 |
| Havering               | 399600 |
| Hounslow               | 395928 |
| Bexley                 | 385668 |
| Camden                 | 378432 |
| Westminster            | 366660 |
| Harrow                 | 365688 |
| Islington              | 359208 |
| Merton                 | 339876 |
| Hammersmith and Fulham | 328752 |
| Sutton                 | 322488 |
| Barking and Dagenham   | 311040 |
| Richmond upon Thames   | 304128 |
| Kensington and Chelsea | 296784 |
| Kingston upon Thames   | 259524 |
| City of London         | 9720   |

```
Name: borough, dtype: int64
```

It seems like most of the crimes took place in the Croydon borough which is then followed by Barnet and Ealing. However, the least amount of crimes took place in the City of London which is pretty surprising since I thought that the city center would have the higher crime rates. Not only that, the crime rate for the City of London is almost 1/3 that of the second lowest - Kingston upon Thames.

```
crime_df.month.value_counts().sort_index()
```

```
1    1124217
2    1124217
3    1124217
4    1124217
5    1124217
6    1124217
7    1124217
8    1124217
9    1124217
10   1124217
11   1124217
12   1124217
Name: month, dtype: int64

crime_df.value.min(), crime_df.value.max()
(0, 309)

crime_df.isnull().sum()

lsoa_code      0
borough       0
major_category 0
minor_category 0
value          0
year           0
month          0
dtype: int64

crime_df.duplicated().sum()

0
```

Luckily, there are not duplicates or null in the dataset. However, even though if there were some duplicates, it wouldn't make any sense to remove them since the same crime can happen in the same month and in the same location.

## 2. Understanding The Different Types Of Crimes

```
crime_df.major_category.unique()

array(['Burglary', 'Violence Against the Person', 'Robbery',
       'Theft and Handling', 'Criminal Damage', 'Drugs',
       'Fraud or Forgery', 'Other Notifiable Offences', 'Sexual
Offences'],
      dtype=object)

crime_df.minor_category.unique()

array(['Burglary in Other Buildings', 'Other violence',
       'Personal Property', 'Other Theft', 'Offensive Weapon',
```

```

'Criminal Damage To Other Building', 'Theft/Taking of Pedal
Cycle',
'Motor Vehicle Interference & Tampering',
'Theft/Taking Of Motor Vehicle', 'Wounding/GBH',
'Other Theft Person', 'Common Assault', 'Theft From Shops',
'Possession Of Drugs', 'Harassment', 'Handling Stolen Goods',
'Criminal Damage To Dwelling', 'Burglary in a Dwelling',
'Criminal Damage To Motor Vehicle', 'Other Criminal Damage',
'Counted per Victim', 'Going Equipped', 'Other Fraud &
Forgery',
'Assault with Injury', 'Drug Trafficking', 'Other Drugs',
'Business Property', 'Other Notifiable', 'Other Sexual',
'Theft From Motor Vehicle', 'Rape', 'Murder'], dtype=object)

# checking the different types of violent crimes
crime_df[crime_df.major_category == 'Violence Against the
Person'].minor_category.value_counts()

Common Assault      522180
Harassment          522072
Assault with Injury 521856
Wounding/GBH        519372
Other violence      512028
Offensive Weapon    481896
Murder              92340
Name: minor_category, dtype: int64

crime_df[crime_df.major_category == 'Other Notifiable
Offences'].minor_category.value_counts()

Other Notifiable    519696
Going Equipped       256608
Name: minor_category, dtype: int64

crime_df[crime_df.major_category ==
'Reббbery'].minor_category.value_counts()

Personal Property   520668
Business Property   418716
Name: minor_category, dtype: int64

crime_df[crime_df.major_category ==
'Burglary'].minor_category.value_counts()

Burglary in Other Buildings   522072
Burglary in a Dwelling       521532
Name: minor_category, dtype: int64

crime_df[crime_df.major_category == 'Sexual
Offences'].minor_category.value_counts()

```

```

Other Sexual      81108
Rape             27000
Name: minor_category, dtype: int64

crime_df[crime_df.major_category == 'Theft and
Handling'].minor_category.value_counts()

Other Theft          522180
Theft From Motor Vehicle 522180
Theft/Taking Of Motor Vehicle 522072
Motor Vehicle Interference & Tampering 520452
Other Theft Person   519480
Theft/Taking of Pedal Cycle 516996
Handling Stolen Goods 426168
Theft From Shops     416772
Name: minor_category, dtype: int64

crime_df[crime_df.major_category == 'Criminal
Damage'].minor_category.value_counts()

Criminal Damage To Motor Vehicle 521964
Other Criminal Damage 521856
Criminal Damage To Dwelling 521424
Criminal Damage To Other Building 503928
Name: minor_category, dtype: int64

```

## Preprocessing Data

```

def create_columns(df):
    """
        Description: this function creates two columns: violent and
        is_daylight_saving
                    with 0 and 1 binary encodings.

        INPUT:
        df - (pandas dataframe) a dataframe that we are interested in
        preprocessing.

        OUTPUT:
        df - (pandas dataframe) a dataframe where the following columns
        have been created:
            1. violent - 0 for non-violent crime and 1 for violent crime.
            2. is_daylight_saving - 0 for not daylight saving and 1 for is
        daylight saving.
    """

    # identifying the violent crimes and daylight saving months and
    # put them all in a list
    violent_crimes = ['Burglary', 'Violence Against the Person',
    'Robbery', 'Sexual Offences']
    is_daylight_saving = list(range(3,12))

```

```

# encode the each respective columns with 0 and 1(binary)
df['violent'] = [1 if x in violent_crimes else 0 for x in
df['major_category']]
df['daylight_saving'] = [1 if x in is_daylight_saving else 0 for x
in df['month']]

return df

```

create\_columns(crime\_df)

|          | lsoa_code | borough    | major_category \            |
|----------|-----------|------------|-----------------------------|
| 0        | E01001116 | Croydon    | Burglary                    |
| 1        | E01001646 | Greenwich  | Violence Against the Person |
| 2        | E01000677 | Bromley    | Violence Against the Person |
| 3        | E01003774 | Redbridge  | Burglary                    |
| 4        | E01004563 | Wandsworth | Robbery                     |
| ...      | ...       | ...        | ...                         |
| 13490599 | E01000504 | Brent      | Criminal Damage             |
| 13490600 | E01002504 | Hillingdon | Robbery                     |
| 13490601 | E01004165 | Sutton     | Burglary                    |
| 13490602 | E01001134 | Croydon    | Robbery                     |
| 13490603 | E01003413 | Merton     | Violence Against the Person |

|          | minor_category              | value | year | month | violent \ |
|----------|-----------------------------|-------|------|-------|-----------|
| 0        | Burglary in Other Buildings | 0     | 2016 | 11    | 1         |
| 1        | Other violence              | 0     | 2016 | 11    | 1         |
| 2        | Other violence              | 0     | 2015 | 5     | 1         |
| 3        | Burglary in Other Buildings | 0     | 2016 | 3     | 1         |
| 4        | Personal Property           | 0     | 2008 | 6     | 1         |
| ...      | ...                         | ...   | ...  | ...   | ...       |
| 13490599 | Criminal Damage To Dwelling | 0     | 2015 | 2     | 0         |
| 13490600 | Personal Property           | 1     | 2015 | 6     | 1         |
| 13490601 | Burglary in a Dwelling      | 0     | 2011 | 2     | 1         |
| 13490602 | Business Property           | 0     | 2011 | 5     | 1         |
| 13490603 | Wounding/GBH                | 0     | 2015 | 6     | 1         |

|          | daylight_saving |
|----------|-----------------|
| 0        | 1               |
| 1        | 1               |
| 2        | 1               |
| 3        | 1               |
| 4        | 1               |
| ...      | ...             |
| 13490599 | 0               |
| 13490600 | 1               |
| 13490601 | 0               |
| 13490602 | 1               |
| 13490603 | 1               |

```
[13490604 rows x 9 columns]
```

## Probability

1. What is the probability of a violent crime occurring regardless of daylight saving status?

```
violent_prob_mean = crime_df['violent'].mean()  
print(f"Probability of violent crime regardless of daylight saving  
status: {violent_prob_mean}.")
```

```
Probability of violent crime regardless of daylight saving status:  
0.3901115176162609.
```

2. Given that daylight saving is happening, what is the probability of a violent crime occurring?

```
violent_isdaylight = crime_df[crime_df.daylight_saving == 1]  
['violent'].mean()  
print(f"Probability of violent crime during daylight saving:  
{violent_isdaylight}.")
```

```
Probability of violent crime during daylight saving:  
0.3901115176162609.
```

3. Given that daylight saving is not happening, what is the probability of a violent crime occurring?

```
violent_notdaylight = crime_df[crime_df.daylight_saving == 0]  
['violent'].mean()  
print(f"Probability of violent crime not during daylight saving:  
{violent_notdaylight}.")
```

```
Probability of violent crime not during daylight saving:  
0.3901115176162609.
```

4. What is the probability of a crime occurring when daylight saving is not in effect?

```
print(f"Probability of a crime occurring when daylight saving is not in  
effect: \\\n{crime_df.daylight_saving.value_counts()[0]/len(crime_df)}")
```

```
Probability of a crime occurring when daylight saving is not in effect:  
0.25.
```

From the probabilities that we've calculated above, there doesn't seem to be sufficient evidence to suggest that the percentage of violent crimes increases when daylight saving is not in effect. In fact, we can see that the violent crime rate is equal to one another, regardless of whether or not daylight saving is in effect.

# Hypothesis Testing

At the beginning, we assumed that violent crime rates is higher when daylight saving is not in effect. Therefore, our hypotheses are as shown down below. We will also assume that a Type I Error rate of 0.05. Null Hypothesis: The difference between the violent crime rates when daylight saving is and isn't in effect is less than or equal to 0.

$H_0: p_{notdaylight} - p_{isdaylight} \leq 0$  Alternative Hypothesis: The difference between the violent crime rates when daylight saving is and isn't in effect is greater than 0.

$$H_1: p_{notdaylight} - p_{isdaylight} > 0$$

First, let us reiterate what we found above. We will assume that the  $p_{notdaylight}$  and  $p_{isdaylight}$  have the same rate equal to the violent crime rate regardless of whether or not daylight saving is in effect. 1. Find the violent crime rate under the null. Also, find out what  $n_{notdaylight}$  and  $n_{isdaylight}$ .

```
# p_notdaylight = p_isdaylight = violent_prob_mean
violent_prob_mean = crime_df['violent'].mean()
# number of crimes committed not during daylight saving
n_notdaylight = crime_df.query("daylight_saving == 0").shape[0]
# number of crimes committed during daylight saving
n_isdaylight = crime_df.query("daylight_saving == 1").shape[0]
```

2. Simulate  $n_{notdaylight}$  and  $n_{isdaylight}$  with their respective violent crime rate. In this case,  $p_{notdaylight}$  and  $p_{isdaylight}$  are both represented by `violent_prob_mean`.

```
# Since there are only 2 mutually exclusive outcomes for our trial, we
# will use random.binomial(n, p, size)
notdaylight_violent = np.random.binomial(1, violent_prob_mean,
n_notdaylight)
notdaylight_violent

array([0, 0, 1, ..., 0, 1, 0])

isdaylight_violent = np.random.binomial(1, violent_prob_mean,
n_isdaylight)
isdaylight_violent

array([0, 0, 0, ..., 0, 1, 1])
```

3. Find  $p_{notdaylight} - p_{isdaylight}$  for our simulated values.

```
notdaylight_violent.mean() - isdaylight_violent.mean()
1.779015972891429e-05
```

4. Run Simulation

```

p_diffs = []

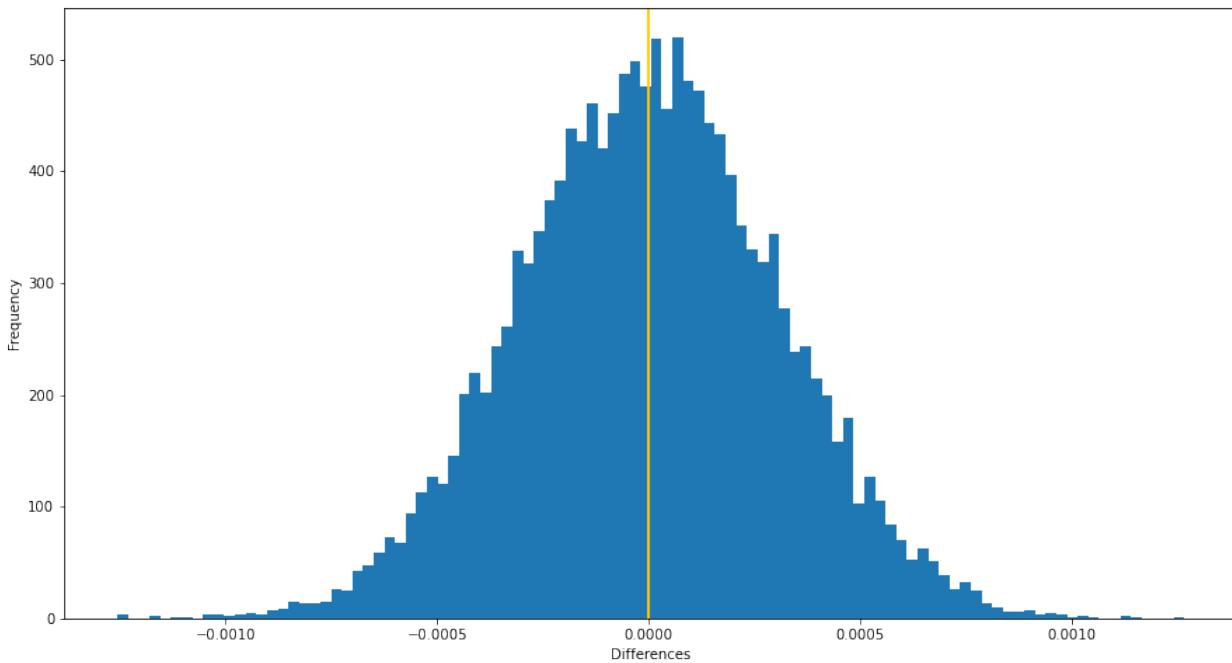
# running binomial simulations
notdaylight_violent_simulate = np.random.binomial(n_notdaylight,
violent_prob_mean, 15000)/n_notdaylight
isdaylight_violent_simulate = np.random.binomial(n_isdaylight,
violent_prob_mean, 15000)/n_isdaylight

# calculate and store simulated differences
p_diffs = notdaylight_violent_simulate - isdaylight_violent_simulate

# calculating observed difference
obs_diff = violent_notdaylight - violent_isdaylight

# plot the results
plt.figure(figsize = (15,8))
plt.hist(p_diffs, bins = 100)
plt.axvline(p_diffs.mean(), color = 'red', label = 'Simulated Difference')
plt.axvline(obs_diff, color = 'yellow', label = 'Observed Difference')
plt.xlabel("Differences")
plt.ylabel("Frequency");

```



```

print(f"Our observed difference is {obs_diff}.")
print(f"The proportion of p_diffs greater than observed difference
is(p-value): \
{(p_diffs > obs_diff).mean()}")

```

```
Our observed difference is 0.0.  
The proportion of p_diffs greater than observed difference is(p-value): 0.5014666666666666
```

Since our p-value is greater than the Type 1 Error of 0.05, we have failed to reject the null hypothesis, which means that there is strong evidence supporting it. Since the p-value is greater than 0.05, we can conclude that it isn't statistically significant. This also means that the violent crime rate in London isn't higher when daylight saving is not in effect. Using the `proportions_ztest` method, we can see that we have also achieved similar results. Please refer to [this link for how to import and use it](#).

```
isdaylight_violent_count = crime_df[crime_df.daylight_saving == 1]  
['violent'].sum()  
notdaylight_violent_count = crime_df[crime_df.daylight_saving == 0]  
['violent'].sum()  
  
n_notdaylight = crime_df.query("daylight_saving == 0").shape[0]  
n_isdaylight = crime_df.query("daylight_saving == 1").shape[0]  
  
z_score, pval = proportions_ztest([isdaylight_violent_count,  
notdaylight_violent_count],  
                                    [n_isdaylight,  
n_notdaylight], alternative = 'larger')  
z_score, pval  
(0.0, 0.5)
```

From above, we have calculated the z-score along with the p-value; the z-score refers to the difference between our test statistics. In our case, it would be the difference in violent crime rate when daylight saving is and isn't in effect. After running the test, we find that our null hypothesis is 0 standard deviation above our mean, while the p-value remains relatively the same which is above our alpha threshold/Type I Error rate of 0.05. As a result, we can conclude that the violent crime rates are not statistically different from one another; thus, giving us no reason to reject the null hypothesis. However, I would still like to try a different approach: the regression approach.

## Regression Approach

However, please note that our null and alternative hypothesis have changed to the following:  
Null Hypothesis: The difference between the violent crime rates when daylight saving is and isn't in effect is equal to 0. This means that there is no difference in violent crime rate when daylight saving is and isn't in effect.  $H_0: p_{notdaylight} - p_{isdaylight} = 0$   
Alternative Hypothesis: The difference between the violent crime rates when daylight saving is and isn't in effect is not equal to 0.  $H_1: p_{notdaylight} - p_{isdaylight} \neq 0$  This is because this is a two-sided t-test compared to the one-sided test we saw before.

```
crime_df2 = crime_df.copy()
```

```

# create intercept
crime_df2['intercept'] = 1

# Instantiate and fit the model
logit_mod = sm.Logit(crime_df2['violent'], crime_df2[['intercept',
'daylight_saving']])
results = logit_mod.fit()

# provide summary of the model
results.summary()

Optimization terminated successfully.
    Current function value: 0.668798
    Iterations 4

<class 'statsmodels.iolib.summary.Summary'>
"""
                    Logit Regression Results

=====
=====
Dep. Variable:          violent    No. Observations:      13490604
Model:                 Logit     Df Residuals:           13490602
Method:                MLE      Df Model:              1
Date: Sun, 10 Apr 2022   Pseudo R-squ.:      1.708e-10
Time:                  21:24:54   Log-Likelihood:  -9.0225e+06
converged:             True    LL-Null:            9.0225e+06
Covariance Type:       nonrobust   LLR p-value:  0.9557
=====
=====

                   coef      std err          z      P>|z|
[0.025      0.975]
-----
-----
intercept      -0.4468      0.001   -400.277      0.000      -
0.449      -0.445
daylight_saving -1.312e-14      0.001   -1.02e-11      1.000      -
0.003      0.003
=====
=====
"""

```

As we can see, the p-value increased to 1 which is even greater than what we have calculated before. This model attempts to predict the likeliness of a violent crime occurring depending on whether or not daylight saving is in effect. The higher p-value can be attributed to the intercept which is meant to account for bias and make the p-value more accurate. We are closer to the true p-value, but we still do not have enough evidence to reject the null hypothesis. Additionally, our model has failed. Quite miserably, if I may add. Now that we have gotten the results of the model, it's time to consider other variables that might influence the violent crime rate. Next, we will take into account the location.

```
all_borough = list(crime_df2.borough.unique())
crime_df2[all_borough] = pd.get_dummies(crime_df2.borough)

crime_df2.head()

      lsoa_code      borough          major_category \
0    E01001116     Croydon            Burglary
1    E01001646   Greenwich  Violence Against the Person
2    E01000677    Bromley  Violence Against the Person
3    E01003774  Redbridge            Burglary
4    E01004563  Wandsworth            Robbery

           minor_category  value  year  month  violent
daylight_saving \
0  Burglary in Other Buildings      0  2016     11       1
1
1
1
2  Other violence                  0  2016     11       1
1
2  Other violence                  0  2015      5       1
1
3  Burglary in Other Buildings      0  2016      3       1
1
4  Personal Property                0  2008      6       1
1

      intercept  ...  Barnet  Waltham Forest  Camden  Bexley \
0        1  ...      0            0        0        0
1        1  ...      0            0        0        0
2        1  ...      0            0        0        0
3        1  ...      0            0        1        0
4        1  ...      0            0        0        0

      Kensington and Chelsea  Islington  Tower Hamlets  Hammersmith and
Fulham \
0
0
1
0
2
0
3
```

```

0          0          0          0
4          0          0          0
0

Merton  City of London
0          0          0
1          0          0
2          0          0
3          0          0
4          1          0

[5 rows x 43 columns]

interest_vars = all_borough[:-1]
interest_vars.append('intercept')

# we will use the City of London as our baseline
# this might take a while since we have a quite a lot of variables
logit_mod = sm.Logit(crime_df2['violent'], crime_df2[interest_vars])
results = logit_mod.fit()
results.summary()

Optimization terminated successfully.
    Current function value: 0.668760
    Iterations 4

<class 'statsmodels.iolib.summary.Summary'>
"""
        Logit Regression Results

=====
=====

Dep. Variable:           violent   No. Observations:      13490604
Model:                 Logit    Df Residuals:           13490571
Method:                MLE     Df Model:              32
Date:      Sun, 10 Apr 2022   Pseudo R-squ.:  5.675e-05
Time:      21:31:59   Log-Likelihood:  -9.0220e+06
converged:                      True   LL-Null:  -9.0225e+06
Covariance Type:            nonrobust   LLR p-value: 1.442e-194
=====

                   coef      std err          z      P>|z|
[0.025      0.975]

```

| Croydon              |        | 0.0323  | 0.005 | 6.463  | 0.000 |
|----------------------|--------|---------|-------|--------|-------|
| 0.023                | 0.042  |         |       |        |       |
| Greenwich            |        | 0.0330  | 0.004 | 7.612  | 0.000 |
| 0.025                | 0.042  |         |       |        |       |
| Bromley              |        | 0.0462  | 0.005 | 9.780  | 0.000 |
| 0.037                | 0.056  |         |       |        |       |
| Redbridge            |        | 0.0198  | 0.004 | 4.426  | 0.000 |
| 0.011                | 0.029  |         |       |        |       |
| Wandsworth           |        | 0.0182  | 0.004 | 4.108  | 0.000 |
| 0.009                | 0.027  |         |       |        |       |
| Ealing               |        | -0.0072 | 0.005 | -1.502 | 0.133 |
| -0.016               | 0.002  |         |       |        |       |
| Hounslow             |        | 0.0177  | 0.021 | 0.841  | 0.400 |
| -0.024               | 0.059  |         |       |        |       |
| Newham               |        | 0.0568  | 0.004 | 13.216 | 0.000 |
| 0.048                | 0.065  |         |       |        |       |
| Sutton               |        | 0.0225  | 0.004 | 5.130  | 0.000 |
| 0.014                | 0.031  |         |       |        |       |
| Haringey             |        | 0.0252  | 0.004 | 5.668  | 0.000 |
| 0.016                | 0.034  |         |       |        |       |
| Lambeth              |        | 0.0471  | 0.005 | 10.176 | 0.000 |
| 0.038                | 0.056  |         |       |        |       |
| Richmond upon Thames |        | 0.0110  | 0.005 | 2.357  | 0.018 |
| 0.002                | 0.020  |         |       |        |       |
| Hillingdon           |        | -0.0194 | 0.005 | -3.933 | 0.000 |
| -0.029               | -0.010 |         |       |        |       |
| Havering             |        | 0.0230  | 0.005 | 4.934  | 0.000 |
| 0.014                | 0.032  |         |       |        |       |
| Barking and Dagenham |        | 0.0341  | 0.005 | 7.119  | 0.000 |
| 0.025                | 0.044  |         |       |        |       |
| Kingston upon Thames |        | 0.0394  | 0.005 | 8.403  | 0.000 |
| 0.030                | 0.049  |         |       |        |       |
| Westminster          |        | 0.0222  | 0.005 | 4.837  | 0.000 |
| 0.013                | 0.031  |         |       |        |       |
| Hackney              |        | 0.0020  | 0.005 | 0.428  | 0.668 |
| -0.007               | 0.011  |         |       |        |       |
| Enfield              |        | 0.0222  | 0.005 | 4.612  | 0.000 |
| 0.013                | 0.032  |         |       |        |       |
| Harrow               |        | -0.0264 | 0.005 | -5.190 | 0.000 |
| -0.036               | -0.016 |         |       |        |       |
| Lewisham             |        | -0.0042 | 0.005 | -0.798 | 0.425 |
| -0.015               | 0.006  |         |       |        |       |
| Brent                |        | 0.0221  | 0.004 | 4.991  | 0.000 |
| 0.013                | 0.031  |         |       |        |       |
| Southwark            |        | 0.0291  | 0.004 | 6.470  | 0.000 |
| 0.020                | 0.038  |         |       |        |       |
| Barnet               |        | 0.0086  | 0.005 | 1.755  | 0.079 |

|                        |        |         |       |          |       |
|------------------------|--------|---------|-------|----------|-------|
| -0.001                 | 0.018  |         |       |          |       |
| Waltham Forest         |        | 0.0326  | 0.005 | 7.225    | 0.000 |
| 0.024                  | 0.042  |         |       |          |       |
| Camden                 |        | 0.0361  | 0.005 | 7.892    | 0.000 |
| 0.027                  | 0.045  |         |       |          |       |
| Bexley                 |        | 0.0116  | 0.005 | 2.300    | 0.021 |
| 0.002                  | 0.021  |         |       |          |       |
| Kensington and Chelsea |        | 0.0090  | 0.004 | 2.000    | 0.046 |
| 0.000                  | 0.018  |         |       |          |       |
| Islington              |        | 0.0357  | 0.005 | 7.210    | 0.000 |
| 0.026                  | 0.045  |         |       |          |       |
| Tower Hamlets          |        | 0.0166  | 0.005 | 3.567    | 0.000 |
| 0.007                  | 0.026  |         |       |          |       |
| Hammersmith and Fulham |        | 0.0445  | 0.005 | 9.526    | 0.000 |
| 0.035                  | 0.054  |         |       |          |       |
| Merton                 |        | 0.0309  | 0.004 | 6.927    | 0.000 |
| 0.022                  | 0.040  |         |       |          |       |
| intercept              |        | -0.4697 | 0.003 | -138.378 | 0.000 |
| -0.476                 | -0.463 |         |       |          |       |
| <hr/>                  |        |         |       |          |       |
| <hr/>                  |        |         |       |          |       |
| ***                    |        |         |       |          |       |

Wow! There are tons of p-values that are below our threshold of 0.05. We are better off finding the variables that are above our threshold. Here is a list of those location where the p-value is greater than our threshold: Ealing, Hounslow, Hackney, Lewisham, and Barnet. This means that the locations listed have no impact on the violent crime rate. With a quick Google search, we find that some of these areas are very in-demand and rather upscale compared to the others. Now that I've looked at the individual factors of location/borough and daylight savings, I want to take a look at the interaction between the two variables to see if there is any significant effect on the violent crime rate. I will create the necessary columns in this notebook, but the modeling and fitting will be done in a different notebook because the kernel keeps crashing due to the size of the new dataset.

```
# create the new columns' name
all_borough_daylight = '_daylight'.join(all_borough)
all_borough_daylight = list(all_borough_daylight.split(',')[:-1])

all_borough_daylight

['Croydon_daylight',
 'Greenwich_daylight',
 'Bromley_daylight',
 'Redbridge_daylight',
 'Wandsworth_daylight',
 'Ealing_daylight',
 'Hounslow_daylight',
 'Newham_daylight',
 'Sutton_daylight',
```

```

'Haringey_daylight',
'Lambeth_daylight',
'Richmond upon Thames_daylight',
'Hillingdon_daylight',
'Havering_daylight',
'Barking and Dagenham_daylight',
'Kingston upon Thames_daylight',
'Westminster_daylight',
'Hackney_daylight',
'Enfield_daylight',
'Harrow_daylight',
'Lewisham_daylight',
'Brent_daylight',
'Southwark_daylight',
'Barnet_daylight',
'Waltham Forest_daylight',
'Camden_daylight',
'Bexley_daylight',
'Kensington and Chelsea_daylight',
'Islington_daylight',
'Tower Hamlets_daylight',
'Hammersmith and Fulham_daylight',
'Merton_daylight']

```

```

crime_df3 = crime_df2.copy()
crime_df3

```

|          | lsoa_code                   | borough        | major_category              | \    |
|----------|-----------------------------|----------------|-----------------------------|------|
| 0        | E01001116                   | Croydon        | Burglary                    |      |
| 1        | E01001646                   | Greenwich      | Violence Against the Person |      |
| 2        | E01000677                   | Bromley        | Violence Against the Person |      |
| 3        | E01003774                   | Redbridge      | Burglary                    |      |
| 4        | E01004563                   | Wandsworth     | Robbery                     |      |
| ..       | ..                          | ..             | ..                          | ..   |
| 13490599 | E01000504                   | Brent          | Criminal Damage             |      |
| 13490600 | E01002504                   | Hillingdon     | Robbery                     |      |
| 13490601 | E01004165                   | Sutton         | Burglary                    |      |
| 13490602 | E01001134                   | Croydon        | Robbery                     |      |
| 13490603 | E01003413                   | Merton         | Violence Against the Person |      |
| ..       | ..                          | ..             | ..                          | ..   |
|          |                             | minor_category | value                       | year |
| 0        | Burglary in Other Buildings | 0              | 2016                        | 11   |
| 1        | Other violence              | 0              | 2016                        | 11   |
| 2        | Other violence              | 0              | 2015                        | 5    |
| 3        | Burglary in Other Buildings | 0              | 2016                        | 3    |
| 4        | Personal Property           | 0              | 2008                        | 6    |
| ..       | ..                          | ..             | ..                          | ..   |
| 13490599 | Criminal Damage To Dwelling | 0              | 2015                        | 2    |
| 13490600 | Personal Property           | 1              | 2015                        | 6    |
| 13490601 | Burglary in a Dwelling      | 0              | 2011                        | 2    |

|                        |                        |                 |               |     |        |                |     |
|------------------------|------------------------|-----------------|---------------|-----|--------|----------------|-----|
| 13490602               |                        | Business        | Property      | 0   | 2011   | 5              | 1   |
| 13490603               |                        | Wounding/GBH    |               | 0   | 2015   | 6              | 1   |
| Camden                 | \                      | daylight_saving | intercept     | ... | Barnet | Waltham Forest |     |
| 0                      |                        | 1               | 1             | ... | 0      |                | 0   |
| 0                      |                        | 1               | 1             | ... | 0      |                | 0   |
| 1                      |                        | 1               | 1             | ... | 0      |                | 0   |
| 0                      |                        | 1               | 1             | ... | 0      |                | 0   |
| 2                      |                        | 1               | 1             | ... | 0      |                | 0   |
| 0                      |                        | 1               | 1             | ... | 0      |                | 0   |
| 3                      |                        | 1               | 1             | ... | 0      |                | 0   |
| 1                      |                        | 1               | 1             | ... | 0      |                | 0   |
| 4                      |                        | 1               | 1             | ... | 0      |                | 0   |
| 0                      |                        | 1               | 1             | ... | 0      |                | 0   |
| ...                    |                        | ...             | ...           | ... | ...    | ...            | ... |
| ...                    |                        | ...             | ...           | ... | ...    | ...            | ... |
| 13490599               |                        | 0               | 1             | ... | 0      |                | 0   |
| 0                      |                        | 1               | 1             | ... | 0      |                | 0   |
| 13490600               |                        | 1               | 1             | ... | 0      |                | 0   |
| 0                      |                        | 0               | 1             | ... | 0      |                | 0   |
| 13490601               |                        | 0               | 1             | ... | 0      |                | 0   |
| 0                      |                        | 1               | 1             | ... | 0      |                | 0   |
| 13490602               |                        | 1               | 1             | ... | 0      |                | 0   |
| 0                      |                        | 1               | 1             | ... | 1      |                | 0   |
| 13490603               |                        | 1               | 1             | ... | 1      |                | 0   |
| 0                      |                        | 0               | 0             |     | 0      |                | 0   |
| Bexley                 | Kensington and Chelsea | Islington       | Tower Hamlets | \   |        |                |     |
| 0                      | 0                      | 0               | 0             |     |        |                | 0   |
| 1                      | 0                      | 0               | 0             |     |        |                | 0   |
| 2                      | 0                      | 0               | 0             |     |        |                | 0   |
| 3                      | 0                      | 0               | 0             |     |        |                | 0   |
| 4                      | 0                      | 0               | 0             |     |        |                | 0   |
| ...                    | ...                    | ...             | ...           |     |        |                | ... |
| 13490599               | 0                      | 0               | 0             |     |        |                | 0   |
| 13490600               | 0                      | 0               | 0             |     |        |                | 0   |
| 13490601               | 0                      | 0               | 1             |     |        |                | 0   |
| 13490602               | 0                      | 0               | 0             |     |        |                | 0   |
| 13490603               | 0                      | 0               | 0             |     |        |                | 0   |
| Hammersmith and Fulham | Merton                 | City of London  |               |     |        |                |     |
| 0                      | 0                      | 0               |               |     |        |                | 0   |
| 1                      | 0                      | 0               |               |     |        |                | 0   |
| 2                      | 0                      | 0               |               |     |        |                | 0   |
| 3                      | 0                      | 0               |               |     |        |                | 0   |
| 4                      | 0                      | 1               |               |     |        |                | 0   |
| ...                    | ...                    | ...             |               |     |        |                | ... |
| 13490599               | 0                      | 0               |               |     |        |                | 0   |
| 13490600               | 0                      | 0               |               |     |        |                | 0   |

```

13490601          0      0      0
13490602          0      0      0
13490603          0      0      0

[13490604 rows x 43 columns]

# multiply 'daylight_saving' and each borough columns to create interactions between the two variables
new_crime = crime_df3[crime_df3.columns[10:-1]].multiply(crime_df3["daylight_saving"], axis="index")

# rename columns
col_rename_dict = {i:j for i,j in zip(crime_df3.columns[10:-1], all_borough_daylight)}
new_crime.rename(columns=col_rename_dict, inplace=True)
new_crime.head()

    Croydon_daylight  Greenwich_daylight  Bromley_daylight
Redbridge_daylight \
0                  0                  0                  0
0
1                  0                  0                  0
0
2                  0                  0                  0
0
3                  0                  0                  0
0
4                  0                  0                  0
0

    Wandsworth_daylight  Ealing_daylight  Hounslow_daylight
Newham_daylight \
0                  0                  0                  0
1
1                  0                  0                  0
0
2                  1                  0                  0
0
3                  0                  0                  0
0
4                  0                  0                  0
0

    Sutton_daylight  Haringey_daylight  ...  Southwark_daylight \
0                  0                  0      ...                  0
1                  0                  0      ...                  0
2                  0                  0      ...                  0
3                  0                  0      ...                  0
4                  0                  0      ...                  0

```

```

Barnet_daylight  Waltham Forest_daylight  Camden_daylight
Bexley_daylight \
0               0                   0                   0
0
1               0                   0                   0
0
2               0                   0                   0
0
3               0                   0                   1
0
4               0                   0                   0
0

Kensington and Chelsea_daylight  Islington_daylight \
0                   0                   0
1                   0                   0
2                   0                   0
3                   0                   0
4                   0                   0

Tower Hamlets_daylight  Hammersmith and Fulham_daylight
Merton_daylight
0               0                   0
0
1               0                   0
0
2               0                   0
0
3               0                   0
0
4               0                   0
1

```

[5 rows x 32 columns]

```
# concat the two dataframes
combined_df = pd.concat([crime_df3, new_crime], axis=1)
combined_df.head()
```

|   | lsoa_code | borough    | major_category              | minor_category              | value | year | month | violent |
|---|-----------|------------|-----------------------------|-----------------------------|-------|------|-------|---------|
| 0 | E01001116 | Croydon    | Burglary                    | Burglary in Other Buildings | 0     | 2016 | 11    | 1       |
| 1 | E01001646 | Greenwich  | Violence Against the Person |                             |       |      |       |         |
| 2 | E01000677 | Bromley    | Violence Against the Person |                             |       |      |       |         |
| 3 | E01003774 | Redbridge  | Burglary                    |                             |       |      |       |         |
| 4 | E01004563 | Wandsworth | Robbery                     |                             |       |      |       |         |

```

1          Other violence      0  2016      11      1
1
2          Other violence      0  2015       5      1
1
3 Burglary in Other Buildings      0  2016       3      1
1
4          Personal Property    0  2008       6      1
1

   intercept ... Southwark_daylight Barnet_daylight \
0           1 ...                      0                      0
1           1 ...                      0                      0
2           1 ...                      0                      0
3           1 ...                      0                      0
4           1 ...                      0                      0

   Waltham Forest_daylight Camden_daylight Bexley_daylight \
0                   0                      0                      0
1                   0                      0                      0
2                   0                      0                      0
3                   0                      1                      0
4                   0                      0                      0

   Kensington and Chelsea_daylight Islington_daylight \
0                           0                      0
1                           0                      0
2                           0                      0
3                           0                      0
4                           0                      0

   Tower Hamlets_daylight Hammersmith and Fulham_daylight
Merton_daylight
0                         0                      0
0
1                         0                      0
0
2                         0                      0
0
3                         0                      0
0
4                         0                      0
1

[5 rows x 75 columns]

# save new dataframe to csv file for fitting and modeling
combined_df.to_csv('london_crime_final.csv', index = False)

```

# London Crimes - Hypothesis Testing Part II

This notebook is a continuation of Part I where I looked at how the individual factors of daylight savings and locations. In the previous notebook, I also created the interaction between location/borough and daylight savings by multiplying them together. Now that I've created all of the necessary columns needed to fit the new model.

|           |                |                        |             |                |               |               |           |
|-----------|----------------|------------------------|-------------|----------------|---------------|---------------|-----------|
| 0         |                |                        |             |                |               |               |           |
| 4         | 1              | 0                      | 0           | 0              | 0             | 0             | 0         |
| 0         |                |                        |             |                |               |               |           |
|           | Hounslow       | Newham                 | Sutton      | Haringey       | Lambeth       | Richmond upon |           |
| Thames    | \              |                        |             |                |               |               |           |
| 0         | 0              | 1                      | 0           | 0              | 0             | 0             | 0         |
| 1         | 0              | 0                      | 0           | 0              | 1             |               | 0         |
| 2         | 0              | 0                      | 0           | 0              | 0             |               | 0         |
| 3         | 0              | 0                      | 0           | 0              | 0             |               | 0         |
| 4         | 0              | 0                      | 0           | 0              | 0             |               | 0         |
|           |                |                        |             |                |               |               |           |
|           | Hillingdon     | Havering               | Barking and | Dagenham       | Kingston upon | Thames        | \         |
| 0         | 0              | 0                      |             | 0              |               | 0             |           |
| 1         | 0              | 0                      |             | 0              |               | 0             |           |
| 2         | 0              | 0                      |             | 0              |               | 0             |           |
| 3         | 0              | 0                      |             | 0              |               | 0             |           |
| 4         | 0              | 0                      |             | 0              |               | 0             |           |
|           |                |                        |             |                |               |               |           |
|           | Westminster    | Hackney                | Enfield     | Harrow         | Lewisham      | Brent         | Southwark |
| Barnet    | \              |                        |             |                |               |               |           |
| 0         | 0              | 0                      | 0           | 0              | 0             | 0             | 0         |
| 0         |                |                        |             |                |               |               |           |
| 1         | 0              | 0                      | 0           | 0              | 0             | 0             | 0         |
| 0         |                |                        |             |                |               |               |           |
| 2         | 0              | 0                      | 0           | 0              | 0             | 0             | 0         |
| 0         |                |                        |             |                |               |               |           |
| 3         | 0              | 0                      | 0           | 0              | 0             | 0             | 0         |
| 0         |                |                        |             |                |               |               |           |
| 4         | 0              | 0                      | 0           | 0              | 0             | 0             | 0         |
| 0         |                |                        |             |                |               |               |           |
|           |                |                        |             |                |               |               |           |
|           | Waltham Forest | Camden                 | Bexley      | Kensington and | Chelsea       |               |           |
| Islington | \              |                        |             |                |               |               |           |
| 0         | 0              | 0                      | 0           |                | 0             | 0             | 0         |
| 1         | 0              | 0                      | 0           |                | 0             | 0             | 0         |
| 2         | 0              | 0                      | 0           |                | 0             | 0             | 0         |
| 3         | 0              | 1                      | 0           |                | 0             | 0             | 0         |
| 4         | 0              | 0                      | 0           |                | 0             | 0             | 0         |
|           |                |                        |             |                |               |               |           |
|           | Tower Hamlets  | Hammersmith and Fulham | Merton      | City of London | \             |               |           |

|   |   |   |   |   |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 |
| 4 | 0 | 0 | 1 | 0 |

|                      | Croydon_daylight | Greenwich_daylight | Bromley_daylight |
|----------------------|------------------|--------------------|------------------|
| Redbridge_daylight \ | 0                | 0                  | 0                |
| 0                    | 0                | 0                  | 0                |
| 1                    | 0                | 0                  | 0                |
| 0                    | 0                | 0                  | 0                |
| 2                    | 0                | 0                  | 0                |
| 0                    | 0                | 0                  | 0                |
| 3                    | 0                | 0                  | 0                |
| 0                    | 0                | 0                  | 0                |
| 4                    | 0                | 0                  | 0                |
| 0                    | 0                | 0                  | 0                |

|                   | Wandsworth_daylight | Ealing_daylight | Hounslow_daylight |
|-------------------|---------------------|-----------------|-------------------|
| Newham_daylight \ | 0                   | 0               | 0                 |
| 0                 | 0                   | 0               | 0                 |
| 1                 | 0                   | 0               | 0                 |
| 0                 | 0                   | 0               | 0                 |
| 2                 | 1                   | 0               | 0                 |
| 0                 | 0                   | 0               | 0                 |
| 3                 | 0                   | 0               | 0                 |
| 0                 | 0                   | 0               | 0                 |
| 4                 | 0                   | 0               | 0                 |
| 0                 | 0                   | 0               | 0                 |

|   | Sutton_daylight | Haringey_daylight | Lambeth_daylight | \ |
|---|-----------------|-------------------|------------------|---|
| 0 | 0               | 0                 | 0                | 0 |
| 1 | 0               | 0                 | 0                | 1 |
| 2 | 0               | 0                 | 0                | 0 |
| 3 | 0               | 0                 | 0                | 0 |
| 4 | 0               | 0                 | 0                | 0 |

|                     | Richmond upon Thames_daylight | Hillingdon_daylight |
|---------------------|-------------------------------|---------------------|
| Havering_daylight \ | 0                             | 0                   |
| 0                   | 0                             | 0                   |
| 1                   | 0                             | 0                   |
| 0                   | 0                             | 0                   |
| 2                   | 0                             | 0                   |
| 0                   | 0                             | 0                   |
| 3                   | 0                             | 0                   |
| 0                   | 0                             | 0                   |
| 4                   | 0                             | 0                   |

0

|   | Barking and Dagenham_daylight | Kingston upon Thames_daylight | \ |
|---|-------------------------------|-------------------------------|---|
| 0 | 0                             | 0                             | 0 |
| 1 | 0                             | 0                             | 0 |
| 2 | 0                             | 0                             | 0 |
| 3 | 0                             | 0                             | 0 |
| 4 | 0                             | 0                             | 0 |

|   | Westminster_daylight | Hackney_daylight | Enfield_daylight |
|---|----------------------|------------------|------------------|
| 0 | 0                    | 0                | 0                |
| 1 | 0                    | 0                | 0                |
| 2 | 0                    | 0                | 0                |
| 3 | 0                    | 0                | 0                |
| 4 | 0                    | 0                | 0                |
| 0 | 0                    | 0                | 0                |

|   | Lewisham_daylight | Brent_daylight | Southwark_daylight |
|---|-------------------|----------------|--------------------|
| 0 | 0                 | 0              | 0                  |
| 1 | 0                 | 0              | 0                  |
| 2 | 0                 | 0              | 0                  |
| 3 | 0                 | 0              | 0                  |
| 4 | 0                 | 0              | 0                  |
| 0 | 0                 | 0              | 0                  |

|   | Waltham Forest_daylight | Camden_daylight | Bexley_daylight | \ |
|---|-------------------------|-----------------|-----------------|---|
| 0 | 0                       | 0               | 0               | 0 |
| 1 | 0                       | 0               | 0               | 0 |
| 2 | 0                       | 0               | 0               | 0 |
| 3 | 0                       | 1               | 0               | 0 |
| 4 | 0                       | 0               | 0               | 0 |

|   | Kensington and Chelsea_daylight | Islington_daylight | \ |
|---|---------------------------------|--------------------|---|
| 0 | 0                               | 0                  | 0 |
| 1 | 0                               | 0                  | 0 |
| 2 | 0                               | 0                  | 0 |
| 3 | 0                               | 0                  | 0 |
| 4 | 0                               | 0                  | 0 |

Tower Hamlets\_daylight Hammersmith and Fulham\_daylight

```
Merton_daylight
0 0 0
0 0 0
1 0 0
0 0 0
2 0 0
0 0 0
3 0 0
0 0 0
4 0 0
1

interested_var = list(crime_final_df.columns[8:])
# remove 'City of London' from list since its our baseline
interested_var.remove('City of London')
interested_var

['daylight_saving',
 'intercept',
 'Croydon',
 'Greenwich',
 'Bromley',
 'Redbridge',
 'Wandsworth',
 'Ealing',
 'Hounslow',
 'Newham',
 'Sutton',
 'Haringey',
 'Lambeth',
 'Richmond upon Thames',
 'Hillingdon',
 'Havering',
 'Barking and Dagenham',
 'Kingston upon Thames',
 'Westminster',
 'Hackney',
 'Enfield',
 'Harrow',
 'Lewisham',
 'Brent',
 'Southwark',
 'Barnet',
 'Waltham Forest',
 'Camden',
 'Bexley',
 'Kensington and Chelsea',
 'Islington',
 'Tower Hamlets',
 'Hammersmith and Fulham',
```

```
'Merton',
'Croydon_daylight',
'Greenwich_daylight',
'Bromley_daylight',
'Redbridge_daylight',
'Wandsworth_daylight',
'Ealing_daylight',
'Hounslow_daylight',
>Newham_daylight',
>Sutton_daylight',
>Haringey_daylight',
>Lambeth_daylight',
'Richmond upon Thames_daylight',
'Hillingdon_daylight',
>Havering_daylight',
>Barking and Dagenham_daylight',
'Kingston upon Thames_daylight',
'Westminster_daylight',
>Hackney_daylight',
>Enfield_daylight',
>Harrow_daylight',
>Lewisham_daylight',
>Brent_daylight',
>Southwark_daylight',
>Barnet_daylight',
>Waltham Forest_daylight',
>Camden_daylight',
>Bexley_daylight',
>Kensington and Chelsea_daylight',
>Islington_daylight',
>Tower Hamlets_daylight',
>Hammersmith and Fulham_daylight',
>Merton_daylight']

logit_mod = sm.Logit(crime_final_df['violent'],
crime_final_df[interested_var])
results = logit_mod.fit()

results.summary()

Optimization terminated successfully.
    Current function value: 0.668760
    Iterations 4

<class 'statsmodels.iolib.summary.Summary'>
"""
        Logit Regression Results
=====
=====
```

|                      |                  |                   |           |
|----------------------|------------------|-------------------|-----------|
| Dep. Variable:       | violent          | No. Observations: |           |
| 13490604             |                  |                   |           |
| Model:               | Logit            | Df Residuals:     |           |
| 13490538             |                  |                   |           |
| Method:              | MLE              | Df Model:         |           |
| 65                   |                  |                   |           |
| Date:                | Mon, 11 Apr 2022 | Pseudo R-squ.:    |           |
| 5.675e-05            |                  |                   |           |
| Time:                | 12:08:55         | Log-Likelihood:   | -         |
| 9.0220e+06           |                  |                   |           |
| converged:           | True             | LL-Null:          | -         |
| 9.0225e+06           |                  |                   |           |
| Covariance Type:     | nonrobust        | LLR p-value:      |           |
| 2.128e-172           |                  |                   |           |
| <hr/>                |                  |                   |           |
| <hr/>                |                  |                   |           |
| P> z                 | [0.025           | 0.975]            |           |
|                      | -----            | -----             | -----     |
| daylight_saving      |                  | -7.071e-14        | 0.008     |
| 1.000                | -0.015           | 0.015             | -9.02e-12 |
| intercept            |                  | -0.4697           | 0.007     |
| 0.000                | -0.483           | -0.456            | -69.189   |
| Croydon              |                  | 0.0323            | 0.010     |
| 0.001                | 0.013            | 0.052             | 3.231     |
| Greenwich            |                  | 0.0330            | 0.009     |
| 0.000                | 0.016            | 0.050             | 3.806     |
| Bromley              |                  | 0.0462            | 0.009     |
| 0.000                | 0.028            | 0.065             | 4.890     |
| Redbridge            |                  | 0.0198            | 0.009     |
| 0.027                | 0.002            | 0.037             | 2.213     |
| Wandsworth           |                  | 0.0182            | 0.009     |
| 0.040                | 0.001            | 0.035             | 2.054     |
| Ealing               |                  | -0.0072           | 0.010     |
| 0.453                | -0.026           | 0.012             | -0.751    |
| Hounslow             |                  | 0.0177            | 0.042     |
| 0.674                | -0.065           | 0.100             | 0.421     |
| Newham               |                  | 0.0568            | 0.009     |
| 0.000                | 0.040            | 0.074             | 6.608     |
| Sutton               |                  | 0.0225            | 0.009     |
| 0.010                | 0.005            | 0.040             | 2.565     |
| Haringey             |                  | 0.0252            | 0.009     |
| 0.005                | 0.008            | 0.043             | 2.834     |
| Lambeth              |                  | 0.0471            | 0.009     |
| 0.000                | 0.029            | 0.065             | 5.088     |
| Richmond upon Thames |                  | 0.0110            | 0.009     |
| 0.239                | -0.007           | 0.029             | 1.179     |
| Hillingdon           |                  | -0.0194           | 0.010     |
|                      |                  |                   | -1.967    |

|                        |        |           |           |       |          |
|------------------------|--------|-----------|-----------|-------|----------|
| 0.049                  | -0.039 | -6.57e-05 |           |       |          |
| Havering               |        |           | 0.0230    | 0.009 | 2.467    |
| 0.014                  | 0.005  | 0.041     |           |       |          |
| Barking and Dagenham   |        |           | 0.0341    | 0.010 | 3.560    |
| 0.000                  | 0.015  | 0.053     |           |       |          |
| Kingston upon Thames   |        |           | 0.0394    | 0.009 | 4.201    |
| 0.000                  | 0.021  | 0.058     |           |       |          |
| Westminster            |        |           | 0.0222    | 0.009 | 2.419    |
| 0.016                  | 0.004  | 0.040     |           |       |          |
| Hackney                |        |           | 0.0020    | 0.009 | 0.214    |
| 0.830                  | -0.016 | 0.020     |           |       |          |
| Enfield                |        |           | 0.0222    | 0.010 | 2.306    |
| 0.021                  | 0.003  | 0.041     |           |       |          |
| Harrow                 |        |           | -0.0264   | 0.010 | -2.595   |
| 0.009                  | -0.046 | -0.006    |           |       |          |
| Lewisham               |        |           | -0.0042   | 0.011 | -0.399   |
| 0.690                  | -0.025 | 0.016     |           |       |          |
| Brent                  |        |           | 0.0221    | 0.009 | 2.495    |
| 0.013                  | 0.005  | 0.039     |           |       |          |
| Southwark              |        |           | 0.0291    | 0.009 | 3.235    |
| 0.001                  | 0.011  | 0.047     |           |       |          |
| Barnet                 |        |           | 0.0086    | 0.010 | 0.878    |
| 0.380                  | -0.011 | 0.028     |           |       |          |
| Waltham Forest         |        |           | 0.0326    | 0.009 | 3.613    |
| 0.000                  | 0.015  | 0.050     |           |       |          |
| Camden                 |        |           | 0.0361    | 0.009 | 3.946    |
| 0.000                  | 0.018  | 0.054     |           |       |          |
| Bexley                 |        |           | 0.0116    | 0.010 | 1.150    |
| 0.250                  | -0.008 | 0.031     |           |       |          |
| Kensington and Chelsea |        |           | 0.0090    | 0.009 | 1.000    |
| 0.317                  | -0.009 | 0.027     |           |       |          |
| Islington              |        |           | 0.0357    | 0.010 | 3.605    |
| 0.000                  | 0.016  | 0.055     |           |       |          |
| Tower Hamlets          |        |           | 0.0166    | 0.009 | 1.783    |
| 0.075                  | -0.002 | 0.035     |           |       |          |
| Hammersmith and Fulham |        |           | 0.0445    | 0.009 | 4.763    |
| 0.000                  | 0.026  | 0.063     |           |       |          |
| Merton                 |        |           | 0.0309    | 0.009 | 3.463    |
| 0.001                  | 0.013  | 0.048     |           |       |          |
| Croydon_daylight       |        |           | 6.534e-14 | 0.012 | 5.66e-12 |
| 1.000                  | -0.023 | 0.023     |           |       |          |
| Greenwich_daylight     |        |           | 6.88e-14  | 0.010 | 6.86e-12 |
| 1.000                  | -0.020 | 0.020     |           |       |          |
| Bromley_daylight       |        |           | 6.202e-14 | 0.011 | 5.68e-12 |
| 1.000                  | -0.021 | 0.021     |           |       |          |
| Redbridge_daylight     |        |           | 4.065e-14 | 0.010 | 3.93e-12 |
| 1.000                  | -0.020 | 0.020     |           |       |          |
| Wandsworth_daylight    |        |           | 6.64e-14  | 0.010 | 6.5e-12  |
| 1.000                  | -0.020 | 0.020     |           |       |          |

|                                 |       |           |       |          |
|---------------------------------|-------|-----------|-------|----------|
| Ealing_daylight                 |       | 7.463e-14 | 0.011 | 6.78e-12 |
| 1.000 -0.022                    | 0.022 | 7e-14     | 0.049 | 1.44e-12 |
| Hounslow_daylight               |       | 0.095     |       |          |
| 1.000 -0.095                    | 0.019 | 7.357e-14 | 0.010 | 7.42e-12 |
| Newham_daylight                 |       | 0.020     |       |          |
| 1.000 -0.019                    | 0.020 | 7.773e-14 | 0.010 | 7.69e-12 |
| Sutton_daylight                 |       | 0.020     |       |          |
| 1.000 -0.020                    | 0.020 | 6.082e-14 | 0.010 | 5.93e-12 |
| Haringey_daylight               |       | 0.020     |       |          |
| 1.000 -0.020                    | 0.021 | 6.582e-14 | 0.011 | 6.15e-12 |
| Lambeth_daylight                |       | 0.021     |       |          |
| 1.000 -0.021                    | 0.021 | 6.361e-14 | 0.011 | 5.92e-12 |
| Richmond upon Thames_daylight   |       | 0.021     |       |          |
| 1.000 -0.021                    | 0.022 | 7.303e-14 | 0.011 | 6.4e-12  |
| Hillingdon_daylight             |       | 0.022     |       |          |
| 1.000 -0.022                    | 0.021 | 6.132e-14 | 0.011 | 5.7e-12  |
| Havering_daylight               |       | 0.021     |       |          |
| 1.000 -0.021                    | 0.022 | 8.417e-14 | 0.011 | 7.6e-12  |
| Barking and Dagenham_daylight   |       | 0.021     |       |          |
| 1.000 -0.022                    | 0.021 | 7.379e-14 | 0.011 | 6.81e-12 |
| Kingston upon Thames_daylight   |       | 0.021     |       |          |
| 1.000 -0.021                    | 0.021 | 9.139e-14 | 0.011 | 8.63e-12 |
| Westminster_daylight            |       | 0.021     |       |          |
| 1.000 -0.021                    | 0.021 | 7.602e-14 | 0.011 | 6.99e-12 |
| Hackney_daylight                |       | 0.021     |       |          |
| 1.000 -0.021                    | 0.022 | 7.676e-14 | 0.011 | 6.9e-12  |
| Enfield_daylight                |       | 0.022     |       |          |
| 1.000 -0.022                    | 0.023 | 6.291e-14 | 0.012 | 5.36e-12 |
| Harrow_daylight                 |       | 0.023     |       |          |
| 1.000 -0.023                    | 0.024 | 7.268e-14 | 0.012 | 5.97e-12 |
| Lewisham_daylight               |       | 0.024     |       |          |
| 1.000 -0.024                    | 0.020 | 5.868e-14 | 0.010 | 5.74e-12 |
| Brent_daylight                  |       | 0.020     |       |          |
| 1.000 -0.020                    | 0.020 | 8.318e-14 | 0.010 | 8.02e-12 |
| Southwark_daylight              |       | 0.020     |       |          |
| 1.000 -0.020                    | 0.020 | 6.55e-14  | 0.011 | 5.8e-12  |
| Barnet_daylight                 |       | 0.022     |       |          |
| 1.000 -0.022                    | 0.020 | 8.143e-14 | 0.010 | 7.8e-12  |
| Waltham Forest_daylight         |       | 0.021     |       |          |
| 1.000 -0.020                    | 0.023 | 7.446e-14 | 0.011 | 7.05e-12 |
| Camden_daylight                 |       | 0.021     |       |          |
| 1.000 -0.021                    | 0.023 | 6.328e-14 | 0.012 | 5.44e-12 |
| Bexley_daylight                 |       | 0.020     |       |          |
| 1.000 -0.023                    | 0.020 | 6.479e-14 | 0.010 | 6.23e-12 |
| Kensington and Chelsea_daylight |       | 0.020     |       |          |
| 1.000 -0.020                    | 0.020 | 7.298e-14 | 0.011 | 6.38e-12 |
| Islington_daylight              |       | 0.022     |       |          |
| 1.000 -0.022                    | 0.022 | 6.71e-14  | 0.011 | 6.23e-12 |
| Tower Hamlets_daylight          |       |           |       |          |

|                        |          |       |           |       |          |  |
|------------------------|----------|-------|-----------|-------|----------|--|
| 1.000                  | -0.021   | 0.021 |           |       |          |  |
| Hammersmith and Fulham | daylight |       | 7.32e-14  | 0.011 | 6.79e-12 |  |
| 1.000                  | -0.021   | 0.021 |           |       |          |  |
| Merton                 | daylight |       | 6.271e-14 | 0.010 | 6.08e-12 |  |
| 1.000                  | -0.020   | 0.020 |           |       |          |  |
| <hr/>                  |          |       |           |       |          |  |
| <hr/>                  |          |       |           |       |          |  |
| ***                    |          |       |           |       |          |  |

## Conclusions

It doesn't seem like much has changed at all, but there is one small difference. We see the p-values of Richmond upon Thames, Kensington & Chelsea, and Tower Hamlets have increased; thus, making them above our 0.05 threshold. It seems like adding these extra variables (interactions) only amplified the p-value. At the beginning (Notebook I), we found that the violent crime rate result isn't dependent on the daylight saving effect, but the location seems to affect the rate. Thus, we do not have enough evidence to accept our alternative hypothesis that the violent crime rate is higher when daylight saving is not in effect. Instead, we should be focusing on the locations since there are a number of statistically significant results where the p-value is less than our alpha threshold. When we add more variables in Part II, we find that it amplifies some of the locations, making their p-value higher than the threshold. Some of the locations that accepted the null hypothesis at the start rejected it once we added these variables. In this second test, we found that the interactions between daylight savings and borough are not statistically significant, which means that the result isn't dependent on it. In other words, the p-value is big enough for us to not reject the null hypothesis. Therefore, I would recommend that the focus should be placed on the locations/boroughs that are statistically significant ( $p\text{-value} \leq 0.05$ ) regardless of the daylight saving status, if the focus is only on violent crimes.

**project**

**12**

**Python - Airbnb Modelling Project**

# Motivation

For this project, I am interested in using the 2016-17 Seattle and Boston Airbnb datasets to answer the following questions:

- What are the important amenities of these listings? Compare the two cities.
- Is it possible to predict the price with 5 features? If yes, compare the 2 cities.
- How does the price in each city change each month? Be sure to compare the 2 cities.
- How does the total number of listings change each month? Be sure to compare the 2 cities.

I chose not to include the .csv file because I would like the extra practice of using and writing in the .gitignore file.

## File Descriptions

You will find 4 Jupyter Notebook files in this repository. Part I mainly deals with examining and understanding the datasets. I have also answered some basic questions about the datasets in this part.

The next 3 parts deal with answering the questions that I have posed. Most of them require a bit of legwork before I could answer them.

The catboost\_info folder is related to the model I tested in Part III.

## Packages

List of packages used:

- Matplotlib
- Numpy
- Pandas
- Calendar
- Seaborn
- Scikit-learn

## Medium Article

Medium Article Link: <https://medium.com/@mr.dcn/a-study-of-airbnb-listings-seattle-boston-ff3a69646edf>

# airbnb-part-i

February 24, 2024

## 1 Boston vs. Seattle Airbnb I - Business and Data Understanding

In this project, I am interested in applying the CRISP-DM process to come up with business questions for AirBnB. I will answer these questions with the datasets provided by the company via Udacity. The two cities I will be examining are Seattle and Boston. At this point, I know that I am interested in studying the prices of both cities, but I will be able to ask more specific questions after looking at the datasets.

```
[1]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb
%matplotlib inline

pd.set_option('display.max_columns', None)
```

### 1.1 Step 1: Business Understanding

#### 1.1.1 Set 1: Seattle Dataset

##### Seattle Listings

```
[2]: seattle_listings = pd.read_csv('seattle_airbnb/listings.csv')
seattle_listings.head(5)
```

```
[2]:      id          listing_url      scrape_id last_scraped \
0    241032  https://www.airbnb.com/rooms/241032  20160104002432  2016-01-04
1    953595  https://www.airbnb.com/rooms/953595  20160104002432  2016-01-04
2   3308979  https://www.airbnb.com/rooms/3308979  20160104002432  2016-01-04
3   7421966  https://www.airbnb.com/rooms/7421966  20160104002432  2016-01-04
4   278830  https://www.airbnb.com/rooms/278830  20160104002432  2016-01-04

                           name \
0        Stylish Queen Anne Apartment
1  Bright & Airy Queen Anne Apartment
2  New Modern House-Amazing water view
3                  Queen Anne Chateau
4  Charming craftsman 3 bdm house
```

```
summary \  
0 NaN  
1 Chemically sensitive? We've removed the irrita...  
2 New modern house built in 2013. Spectacular s...  
3 A charming apartment that sits atop Queen Anne...  
4 Cozy family craftman house in beautiful neighb...  
  
space \  
0 Make your self at home in this charming one-be...  
1 Beautiful, hypoallergenic apartment in an extr...  
2 Our house is modern, light and fresh with a wa...  
3 NaN  
4 Cozy family craftman house in beautiful neighb...  
  
description experiences_offered \  
0 Make your self at home in this charming one-be... none  
1 Chemically sensitive? We've removed the irrita... none  
2 New modern house built in 2013. Spectacular s... none  
3 A charming apartment that sits atop Queen Anne... none  
4 Cozy family craftman house in beautiful neighb... none  
  
neighborhood_overview \  
0 NaN  
1 Queen Anne is a wonderful, truly functional vi...  
2 Upper Queen Anne is a charming neighborhood fu...  
3 NaN  
4 We are in the beautiful neighborhood of Queen ...  
  
notes \  
0 NaN  
1 What's up with the free pillows? Our home was...  
2 Our house is located just 5 short blocks to To...  
3 NaN  
4 Belltown  
  
transit \  
0 NaN  
1 Convenient bus stops are just down the block, ...  
2 A bus stop is just 2 blocks away. Easy bus a...  
3 NaN  
4 The nearest public transit bus (D Line) is 2 b...  
  
thumbnail_url \  
0 NaN  
1 https://a0.muscache.com/ac/pictures/14409893/f...  
2 NaN  
3 NaN
```

```

4                               NaN
   medium_url  \
0                               NaN
1 https://a0.muscache.com/im/pictures/14409893/f...
2                               NaN
3                               NaN
4                               NaN

   picture_url  \
0 https://a1.muscache.com/ac/pictures/67560560/c...
1 https://a0.muscache.com/ac/pictures/14409893/f...
2 https://a2.muscache.com/ac/pictures/b4324e0f-a...
3 https://a0.muscache.com/ac/pictures/94146944/6...
4 https://a1.muscache.com/ac/pictures/6120468/b0...

   xl_picture_url  host_id  \
0                               NaN      956883
1 https://a0.muscache.com/ac/pictures/14409893/f...  5177328
2                               NaN    16708587
3                               NaN    9851441
4                               NaN    1452570

   host_url host_name  host_since  \
0 https://www.airbnb.com/users/show/956883      Maija  2011-08-11
1 https://www.airbnb.com/users/show/5177328     Andrea  2013-02-21
2 https://www.airbnb.com/users/show/16708587     Jill   2014-06-12
3 https://www.airbnb.com/users/show/9851441     Emily  2013-11-06
4 https://www.airbnb.com/users/show/1452570     Emily  2011-11-29

   host_location  \
0 Seattle, Washington, United States
1 Seattle, Washington, United States
2 Seattle, Washington, United States
3 Seattle, Washington, United States
4 Seattle, Washington, United States

   host_about  host_response_time  \
0 I am an artist, interior designer, and run a s...  within a few hours
1 Living east coast/left coast/overseas. Time i...  within an hour
2 i love living in Seattle. i grew up in the mi...  within a few hours
3                               NaN                  NaN
4 Hi, I live in Seattle, Washington but I'm orig...  within an hour

   host_response_rate host_acceptance_rate host_is_superhost  \
0             96%                 100%                      f
1             98%                 100%                      t

```

|   |                                                                          |                                       |   |
|---|--------------------------------------------------------------------------|---------------------------------------|---|
| 2 | 67%                                                                      | 100%                                  | f |
| 3 | NaN                                                                      | NaN                                   | f |
| 4 | 100%                                                                     | NaN                                   | f |
|   |                                                                          | host_thumbnail_url \                  |   |
| 0 | https://a0.muscache.com/ac/users/956883/profil...                        |                                       |   |
| 1 | https://a0.muscache.com/ac/users/5177328/profi...                        |                                       |   |
| 2 | https://a1.muscache.com/ac/users/16708587/prof...                        |                                       |   |
| 3 | https://a2.muscache.com/ac/users/9851441/profi...                        |                                       |   |
| 4 | https://a0.muscache.com/ac/users/1452570/profi...                        |                                       |   |
|   |                                                                          | host_picture_url host_neighbourhood \ |   |
| 0 | https://a0.muscache.com/ac/users/956883/profil...                        | Queen Anne                            |   |
| 1 | https://a0.muscache.com/ac/users/5177328/profi...                        | Queen Anne                            |   |
| 2 | https://a1.muscache.com/ac/users/16708587/prof...                        | Queen Anne                            |   |
| 3 | https://a2.muscache.com/ac/users/9851441/profi...                        | Queen Anne                            |   |
| 4 | https://a0.muscache.com/ac/users/1452570/profi...                        | Queen Anne                            |   |
|   | host_listings_count host_total_listings_count \                          |                                       |   |
| 0 | 3.0                                                                      | 3.0                                   |   |
| 1 | 6.0                                                                      | 6.0                                   |   |
| 2 | 2.0                                                                      | 2.0                                   |   |
| 3 | 1.0                                                                      | 1.0                                   |   |
| 4 | 2.0                                                                      | 2.0                                   |   |
|   | host_verifications host_has_profile_pic \                                |                                       |   |
| 0 | ['email', 'phone', 'reviews', 'kba']                                     | t                                     |   |
| 1 | ['email', 'phone', 'facebook', 'linkedin', 're...                        | t                                     |   |
| 2 | ['email', 'phone', 'google', 'reviews', 'jumio']                         | t                                     |   |
| 3 | ['email', 'phone', 'facebook', 'reviews', 'jum...                        | t                                     |   |
| 4 | ['email', 'phone', 'facebook', 'reviews', 'kba']                         | t                                     |   |
|   | host_identity_verified                                                   | street \                              |   |
| 0 | t Gilman Dr W, Seattle, WA 98119, United States                          |                                       |   |
| 1 | t 7th Avenue West, Seattle, WA 98119, United States                      |                                       |   |
| 2 | t West Lee Street, Seattle, WA 98119, United States                      |                                       |   |
| 3 | t 8th Avenue West, Seattle, WA 98119, United States                      |                                       |   |
| 4 | t 14th Ave W, Seattle, WA 98119, United States                           |                                       |   |
|   | neighbourhood neighbourhood_cleansed neighbourhood_group_cleansed city \ |                                       |   |
| 0 | Queen Anne West Queen Anne Queen Anne Seattle                            |                                       |   |
| 1 | Queen Anne West Queen Anne Queen Anne Seattle                            |                                       |   |
| 2 | Queen Anne West Queen Anne Queen Anne Seattle                            |                                       |   |
| 3 | Queen Anne West Queen Anne Queen Anne Seattle                            |                                       |   |
| 4 | Queen Anne West Queen Anne Queen Anne Seattle                            |                                       |   |
|   | state zipcode market smart_location country_code country \               |                                       |   |

|   |                                                   |                 |                   |                  |                 |                 |
|---|---------------------------------------------------|-----------------|-------------------|------------------|-----------------|-----------------|
| 0 | WA                                                | 98119           | Seattle           | Seattle, WA      | US              | United States   |
| 1 | WA                                                | 98119           | Seattle           | Seattle, WA      | US              | United States   |
| 2 | WA                                                | 98119           | Seattle           | Seattle, WA      | US              | United States   |
| 3 | WA                                                | 98119           | Seattle           | Seattle, WA      | US              | United States   |
| 4 | WA                                                | 98119           | Seattle           | Seattle, WA      | US              | United States   |
|   |                                                   |                 |                   |                  |                 |                 |
| 0 | latitude                                          | longitude       | is_location_exact | property_type    | room_type       | \               |
| 0 | 47.636289                                         | -122.371025     |                   | t                | Apartment       | Entire home/apt |
| 1 | 47.639123                                         | -122.365666     |                   | t                | Apartment       | Entire home/apt |
| 2 | 47.629724                                         | -122.369483     |                   | t                | House           | Entire home/apt |
| 3 | 47.638473                                         | -122.369279     |                   | t                | Apartment       | Entire home/apt |
| 4 | 47.632918                                         | -122.372471     |                   | t                | House           | Entire home/apt |
|   |                                                   |                 |                   |                  |                 |                 |
| 0 | accommodates                                      | bathrooms       | bedrooms          | beds             | bed_type        | \               |
| 0 | 4                                                 | 1.0             | 1.0               | 1.0              | Real Bed        |                 |
| 1 | 4                                                 | 1.0             | 1.0               | 1.0              | Real Bed        |                 |
| 2 | 11                                                | 4.5             | 5.0               | 7.0              | Real Bed        |                 |
| 3 | 3                                                 | 1.0             | 0.0               | 2.0              | Real Bed        |                 |
| 4 | 6                                                 | 2.0             | 3.0               | 3.0              | Real Bed        |                 |
|   |                                                   |                 |                   |                  |                 |                 |
| 0 | amenities                                         | square_feet     | price             | \                |                 |                 |
| 0 | {TV,"Cable TV",Internet,"Wireless Internet","A... | NaN             | \$85.00           |                  |                 |                 |
| 1 | {TV,Internet,"Wireless Internet",Kitchen,"Free... | NaN             | \$150.00          |                  |                 |                 |
| 2 | {TV,"Cable TV",Internet,"Wireless Internet","A... | NaN             | \$975.00          |                  |                 |                 |
| 3 | {Internet,"Wireless Internet",Kitchen,"Indoor ... | NaN             | \$100.00          |                  |                 |                 |
| 4 | {TV,"Cable TV",Internet,"Wireless Internet",Ki... | NaN             | \$450.00          |                  |                 |                 |
|   |                                                   |                 |                   |                  |                 |                 |
| 0 | weekly_price                                      | monthly_price   | security_deposit  | cleaning_fee     | guests_included | \               |
| 0 | NaN                                               | NaN             | NaN               | NaN              | 2               |                 |
| 1 | \$1,000.00                                        | \$3,000.00      | \$100.00          | \$40.00          | 1               |                 |
| 2 | NaN                                               | NaN             | \$1,000.00        | \$300.00         | 10              |                 |
| 3 | \$650.00                                          | \$2,300.00      | NaN               | NaN              | 1               |                 |
| 4 | NaN                                               | NaN             | \$700.00          | \$125.00         | 6               |                 |
|   |                                                   |                 |                   |                  |                 |                 |
| 0 | extra_people                                      | minimum_nights  | maximum_nights    | calendar_updated |                 | \               |
| 0 | \$5.00                                            | 1               | 365               | 4 weeks ago      |                 |                 |
| 1 | \$0.00                                            | 2               | 90                | today            |                 |                 |
| 2 | \$25.00                                           | 4               | 30                | 5 weeks ago      |                 |                 |
| 3 | \$0.00                                            | 1               | 1125              | 6 months ago     |                 |                 |
| 4 | \$15.00                                           | 1               | 1125              | 7 weeks ago      |                 |                 |
|   |                                                   |                 |                   |                  |                 |                 |
| 0 | has_availability                                  | availability_30 | availability_60   | availability_90  |                 | \               |
| 0 | t                                                 | 14              | 41                | 71               |                 |                 |
| 1 | t                                                 | 13              | 13                | 16               |                 |                 |
| 2 | t                                                 | 1               | 6                 | 17               |                 |                 |
| 3 | t                                                 | 0               | 0                 | 0                |                 |                 |
| 4 | t                                                 | 30              | 60                | 90               |                 |                 |

```

availability_365 calendar_last_scraped number_of_reviews first_review \
0           346      2016-01-04          207  2011-11-01
1           291      2016-01-04          43   2013-08-19
2           220      2016-01-04          20   2014-07-30
3           143      2016-01-04           0    NaN
4           365      2016-01-04          38  2012-07-10

last_review review_scores_rating review_scores_accuracy \
0  2016-01-02            95.0          10.0
1  2015-12-29            96.0          10.0
2  2015-09-03            97.0          10.0
3        NaN              NaN          NaN
4  2015-10-24            92.0          9.0

review_scores_cleanliness review_scores_checkin \
0             10.0          10.0
1             10.0          10.0
2             10.0          10.0
3             NaN            NaN
4             9.0          10.0

review_scores_communication review_scores_location review_scores_value \
0             10.0          9.0          10.0
1             10.0         10.0          10.0
2             10.0         10.0          10.0
3             NaN            NaN          NaN
4             10.0          9.0          9.0

requires_license license jurisdiction_names instant_bookable \
0             f       NaN      WASHINGTON          f
1             f       NaN      WASHINGTON          f
2             f       NaN      WASHINGTON          f
3             f       NaN      WASHINGTON          f
4             f       NaN      WASHINGTON          f

cancellation_policy require_guest_profile_picture \
0      moderate            f
1       strict             t
2       strict             f
3     flexible             f
4       strict             f

require_guest_phone_verification calculated_host_listings_count \
0                 f            2
1                 t            6
2                 f            2

```

```
3                      f          1
4                      f          1

  reviews_per_month
0              4.07
1              1.48
2              1.15
3                NaN
4              0.89
```

```
[3]: print(f"There are a total of {seattle_listings.shape[0]} rows.")
print(f"There are a total of {seattle_listings.shape[1]} columns.")
```

There are a total of 3818 rows.  
There are a total of 92 columns.

## Seattle Calendar

```
[4]: seattle_calendar = pd.read_csv('seattle_airbnb/calendar.csv')
seattle_calendar.head(5)
```

```
[4]:   listing_id      date available    price
0     241032  2016-01-04        t $85.00
1     241032  2016-01-05        t $85.00
2     241032  2016-01-06        f      NaN
3     241032  2016-01-07        f      NaN
4     241032  2016-01-08        f      NaN
```

```
[5]: print(f"There are a total of {seattle_calendar.shape[0]} rows.")
print(f"There are a total of {seattle_calendar.shape[1]} columns.")
```

There are a total of 1393570 rows.  
There are a total of 4 columns.

```
[6]: seattle_min_date = seattle_calendar.date.min()
seattle_max_date = seattle_calendar.date.max()

print(f"The date range for Seattle's listings is {seattle_min_date} to "
      f"{seattle_max_date}.")
```

The date range for Seattle's listings is 2016-01-04 to 2017-01-02.

## Seattle Reviews

```
[7]: seattle_reviews = pd.read_csv('seattle_airbnb/reviews.csv')
seattle_reviews.head()
```

```
[7]:   listing_id      id      date reviewer_id reviewer_name \
0     7202016  38917982  2015-07-19  28943674      Bianca
```

```

1    7202016  39087409  2015-07-20      32440555      Frank
2    7202016  39820030  2015-07-26      37722850      Ian
3    7202016  40813543  2015-08-02      33671805     George
4    7202016  41986501  2015-08-10      34959538      Ming

```

|   |                                                   |  | comments |
|---|---------------------------------------------------|--|----------|
| 0 | Cute and cozy place. Perfect location to every... |  |          |
| 1 | Kelly has a great room in a very central locat... |  |          |
| 2 | Very spacious apartment, and in a great neighb... |  |          |
| 3 | Close to Seattle Center and all it has to offe... |  |          |
| 4 | Kelly was a great host and very accommodating ... |  |          |

```
[8]: print(f"There are a total of {seattle_reviews.shape[0]} rows.")
print(f"There are a total of {seattle_reviews.shape[1]} columns.")
```

There are a total of 84849 rows.

There are a total of 6 columns.

### 1.1.2 Set 2: Boston Dataset

#### Boston Listings

```
[9]: boston_listings = pd.read_csv('boston_airbnb/listings.csv')
boston_listings.head(5)
```

|   | id       | listing_url                                                                               | scrape_id      | \ |
|---|----------|-------------------------------------------------------------------------------------------|----------------|---|
| 0 | 12147973 | <a href="https://www.airbnb.com/rooms/12147973">https://www.airbnb.com/rooms/12147973</a> | 20160906204935 |   |
| 1 | 3075044  | <a href="https://www.airbnb.com/rooms/3075044">https://www.airbnb.com/rooms/3075044</a>   | 20160906204935 |   |
| 2 | 6976     | <a href="https://www.airbnb.com/rooms/6976">https://www.airbnb.com/rooms/6976</a>         | 20160906204935 |   |
| 3 | 1436513  | <a href="https://www.airbnb.com/rooms/1436513">https://www.airbnb.com/rooms/1436513</a>   | 20160906204935 |   |
| 4 | 7651065  | <a href="https://www.airbnb.com/rooms/7651065">https://www.airbnb.com/rooms/7651065</a>   | 20160906204935 |   |

|   | last_scraped | name                                          | \ |
|---|--------------|-----------------------------------------------|---|
| 0 | 2016-09-07   | Sunny Bungalow in the City                    |   |
| 1 | 2016-09-07   | Charming room in pet friendly apt             |   |
| 2 | 2016-09-07   | Mexican Folk Art Haven in Boston              |   |
| 3 | 2016-09-07   | Spacious Sunny Bedroom Suite in Historic Home |   |
| 4 | 2016-09-07   | Come Home to Boston                           |   |

|   | summary                                           | \ |
|---|---------------------------------------------------|---|
| 0 | Cozy, sunny, family home. Master bedroom high...  |   |
| 1 | Charming and quiet room in a second floor 1910... |   |
| 2 | Come stay with a friendly, middle-aged guy in ... |   |
| 3 | Come experience the comforts of home away from... |   |
| 4 | My comfy, clean and relaxing home is one block... |   |

|   | space                                             | \ |
|---|---------------------------------------------------|---|
| 0 | The house has an open and cozy feel at the sam... |   |

- 1 Small but cozy and quite room with a full size...
- 2 Come stay with a friendly, middle-aged guy in ...
- 3 Most places you find in Boston are small howev...
- 4 Clean, attractive, private room, one block fro...

description experiences\_offered \

- 0 Cozy, sunny, family home. Master bedroom high... none
- 1 Charming and quiet room in a second floor 1910... none
- 2 Come stay with a friendly, middle-aged guy in ... none
- 3 Come experience the comforts of home away from... none
- 4 My comfy, clean and relaxing home is one block... none

neighborhood\_overview \

- 0 Roslindale is quiet, convenient and friendly. ...
- 1 The room is in Roslindale, a diverse and prima...
- 2 The LOCATION: Roslindale is a safe and diverse...
- 3 Roslindale is a lovely little neighborhood loc...
- 4 I love the proximity to downtown, the neighbor...

notes \

- 0 NaN
- 1 If you don't have a US cell phone, you can tex...
- 2 I am in a scenic part of Boston with a couple ...
- 3 Please be mindful of the property as it is old...
- 4 I have one roommate who lives on the lower lev...

transit \

- 0 The bus stop is 2 blocks away, and frequent. B...
- 1 Plenty of safe street parking. Bus stops a few...
- 2 PUBLIC TRANSPORTATION: From the house, quick p...
- 3 There are buses that stop right in front of th...
- 4 From Logan Airport and South Station you have...

access \

- 0 You will have access to 2 bedrooms, a living r...
- 1 Apt has one more bedroom (which I use) and lar...
- 2 I am living in the apartment during your stay,...
- 3 The basement has a washer dryer and gym area. ...
- 4 You will have access to the front and side por...

interaction \

- 0 NaN
- 1 If I am at home, I am likely working in my hom...
- 2 ABOUT ME: I'm a laid-back, friendly, unmarried...
- 3 We do live in the house therefore might be som...
- 4 I love my city and really enjoy sharing it wit...

```
                house_rules \  
0 Clean up and treat the home the way you'd like...  
1 Pet friendly but please confirm with me if the...  
2 I encourage you to use my kitchen, cooking and...  
3 - The bathroom and house are shared so please ...  
4 Please no smoking in the house, porch or on th...
```

```
                thumbnail_url \  
0 https://a2.muscache.com/im/pictures/c0842db1-e...  
1 https://a1.muscache.com/im/pictures/39327812/d...  
2 https://a2.muscache.com/im/pictures/6ae8335d-9...  
3 https://a2.muscache.com/im/pictures/39764190-1...  
4 https://a1.muscache.com/im/pictures/97154760/8...
```

```
                medium_url \  
0 https://a2.muscache.com/im/pictures/c0842db1-e...  
1 https://a1.muscache.com/im/pictures/39327812/d...  
2 https://a2.muscache.com/im/pictures/6ae8335d-9...  
3 https://a2.muscache.com/im/pictures/39764190-1...  
4 https://a1.muscache.com/im/pictures/97154760/8...
```

```
                picture_url \  
0 https://a2.muscache.com/im/pictures/c0842db1-e...  
1 https://a1.muscache.com/im/pictures/39327812/d...  
2 https://a2.muscache.com/im/pictures/6ae8335d-9...  
3 https://a2.muscache.com/im/pictures/39764190-1...  
4 https://a1.muscache.com/im/pictures/97154760/8...
```

```
                xl_picture_url host_id \  
0 https://a2.muscache.com/im/pictures/c0842db1-e... 31303940  
1 https://a1.muscache.com/im/pictures/39327812/d... 2572247  
2 https://a2.muscache.com/im/pictures/6ae8335d-9... 16701  
3 https://a2.muscache.com/im/pictures/39764190-1... 6031442  
4 https://a1.muscache.com/im/pictures/97154760/8... 15396970
```

```
                host_url host_name host_since \  
0 https://www.airbnb.com/users/show/31303940 Virginia 2015-04-15  
1 https://www.airbnb.com/users/show/2572247 Andrea 2012-06-07  
2 https://www.airbnb.com/users/show/16701 Phil 2009-05-11  
3 https://www.airbnb.com/users/show/6031442 Meghna 2013-04-21  
4 https://www.airbnb.com/users/show/15396970 Linda 2014-05-11
```

```
                host_location \  
0 Boston, Massachusetts, United States  
1 Boston, Massachusetts, United States  
2 Boston, Massachusetts, United States  
3 Boston, Massachusetts, United States
```

4 Boston, Massachusetts, United States

```
host_about host_response_time \
0 We are country and city connecting in our deck...           NaN
1 I live in Boston and I like to travel and have...       within an hour
2 I am a middle-aged, single male with a wide ra...   within a few hours
3 My husband and I live on the property. He's a... within a few hours
4 I work full time for a public school district...    within an hour

host_response_rate host_acceptance_rate host_is_superhost \
0          NaN            NaN      f
1         100%           100%      f
2         100%            88%      t
3         100%            50%      f
4         100%           100%      t

host_thumbnail_url \
0 https://a2.muscache.com/im/pictures/5936fef0-b...
1 https://a2.muscache.com/im/users/2572247/profi...
2 https://a2.muscache.com/im/users/16701/profile...
3 https://a2.muscache.com/im/pictures/5d430cde-7...
4 https://a0.muscache.com/im/users/15396970/prof...

host_picture_url host_neighbourhood \
0 https://a2.muscache.com/im/pictures/5936fef0-b... Roslindale
1 https://a2.muscache.com/im/users/2572247/profi... Roslindale
2 https://a2.muscache.com/im/users/16701/profile... Roslindale
3 https://a2.muscache.com/im/pictures/5d430cde-7...     NaN
4 https://a0.muscache.com/im/users/15396970/prof... Roslindale

host_listings_count host_total_listings_count \
0                 1                  1
1                 1                  1
2                 1                  1
3                 1                  1
4                 1                  1

host_verifications host_has_profile_pic \
0      ['email', 'phone', 'facebook', 'reviews']          t
1      ['email', 'phone', 'facebook', 'linkedin', 'am...  t
2      ['email', 'phone', 'reviews', 'jumio']            t
3      ['email', 'phone', 'reviews']                      t
4      ['email', 'phone', 'reviews', 'kba']              t

host_identity_verified street \
0             f      Birch Street, Boston, MA 02131, United States
1             t      Pinehurst Street, Boston, MA 02131, United States
```

```

2           t      Ardale St., Boston, MA 02131, United States
3           f      Boston, MA, United States
4           t      Durnell Avenue, Boston, MA 02131, United States

neighbourhood neighbourhood_cleansed neighbourhood_group_cleansed city \
0   Roslindale          Roslindale             NaN  Boston
1   Roslindale          Roslindale             NaN  Boston
2   Roslindale          Roslindale             NaN  Boston
3     NaN                Roslindale             NaN  Boston
4   Roslindale          Roslindale             NaN  Boston

state zipcode market smart_location country_code      country  latitude \
0    MA    02131  Boston    Boston, MA        US  United States  42.282619
1    MA    02131  Boston    Boston, MA        US  United States  42.286241
2    MA    02131  Boston    Boston, MA        US  United States  42.292438
3    MA     NaN  Boston    Boston, MA        US  United States  42.281106
4    MA    02131  Boston    Boston, MA        US  United States  42.284512

longitude is_location_exact property_type      room_type accommodates \
0 -71.133068            t      House  Entire home/apt       4
1 -71.134374            t      Apartment  Private room       2
2 -71.135765            t      Apartment  Private room       2
3 -71.121021            f      House  Private room       4
4 -71.136258            t      House  Private room       2

bathrooms bedrooms beds bed_type \
0      1.5      2.0  3.0  Real Bed
1      1.0      1.0  1.0  Real Bed
2      1.0      1.0  1.0  Real Bed
3      1.0      1.0  2.0  Real Bed
4      1.5      1.0  2.0  Real Bed

amenities square_feet price \
0  {TV,"Wireless Internet",Kitchen,"Free Parking ...  NaN  $250.00
1  {TV,Internet,"Wireless Internet","Air Conditio...  NaN  $65.00
2  {TV,"Cable TV","Wireless Internet","Air Condit...  NaN  $65.00
3  {TV,Internet,"Wireless Internet","Air Conditio...  NaN  $75.00
4  {Internet,"Wireless Internet","Air Conditionin...  NaN  $79.00

weekly_price monthly_price security_deposit cleaning_fee guests_included \
0        NaN        NaN        NaN  $35.00           1
1    $400.00        NaN        $95.00  $10.00           0
2    $395.00    $1,350.00        NaN        NaN           1
3        NaN        NaN        $100.00  $50.00           2
4        NaN        NaN        NaN  $15.00           1

extra_people minimum_nights maximum_nights calendar_updated \

```

|   |                             |                        |                        |                  |
|---|-----------------------------|------------------------|------------------------|------------------|
| 0 | \$0.00                      | 2                      | 1125                   | 2 weeks ago      |
| 1 | \$0.00                      | 2                      | 15                     | a week ago       |
| 2 | \$20.00                     | 3                      | 45                     | 5 days ago       |
| 3 | \$25.00                     | 1                      | 1125                   | a week ago       |
| 4 | \$0.00                      | 2                      | 31                     | 2 weeks ago      |
|   |                             |                        |                        | \                |
| 0 | has_availability            | availability_30        | availability_60        | availability_90  |
| 1 | NaN                         | 0                      | 0                      | 0                |
| 2 | NaN                         | 26                     | 54                     | 84               |
| 3 | NaN                         | 19                     | 46                     | 61               |
| 4 | NaN                         | 6                      | 16                     | 26               |
|   |                             | 13                     | 34                     | 59               |
|   |                             |                        |                        | \                |
| 0 | availability_365            | calendar_last_scraped  | number_of_reviews      | first_review     |
| 1 | 0                           | 2016-09-06             | 0                      | NaN              |
| 2 | 359                         | 2016-09-06             | 36                     | 2014-06-01       |
| 3 | 319                         | 2016-09-06             | 41                     | 2009-07-19       |
| 4 | 98                          | 2016-09-06             | 1                      | 2016-08-28       |
|   |                             | 334                    | 2016-09-06             | 29               |
|   |                             |                        |                        | 2015-08-18       |
|   |                             |                        |                        | \                |
| 0 | last_review                 | review_scores_rating   | review_scores_accuracy |                  |
| 1 | NaN                         | NaN                    | NaN                    |                  |
| 2 | 2016-08-13                  | 94.0                   | 10.0                   |                  |
| 3 | 2016-08-05                  | 98.0                   | 10.0                   |                  |
| 4 | 2016-08-28                  | 100.0                  | 10.0                   |                  |
|   |                             | 99.0                   | 10.0                   |                  |
|   |                             |                        |                        | \                |
| 0 | review_scores_cleanliness   | review_scores_checkin  |                        |                  |
| 1 | NaN                         | NaN                    |                        |                  |
| 2 | 9.0                         | 10.0                   |                        |                  |
| 3 | 9.0                         | 10.0                   |                        |                  |
| 4 | 10.0                        | 10.0                   |                        |                  |
|   |                             | 10.0                   |                        |                  |
|   |                             |                        |                        | \                |
| 0 | review_scores_communication | review_scores_location | review_scores_value    |                  |
| 1 | NaN                         | NaN                    | NaN                    |                  |
| 2 | 10.0                        | 9.0                    | 9.0                    |                  |
| 3 | 10.0                        | 9.0                    | 10.0                   |                  |
| 4 | 10.0                        | 10.0                   | 10.0                   |                  |
|   |                             | 9.0                    | 10.0                   |                  |
|   |                             |                        |                        | \                |
| 0 | requires_license            | license                | jurisdiction_names     | instant_bookable |
| 1 | f                           | NaN                    | NaN                    | f                |
| 2 | f                           | NaN                    | NaN                    | t                |
| 3 | f                           | NaN                    | NaN                    | f                |
| 4 | f                           | NaN                    | NaN                    | f                |

```

cancellation_policy require_guest_profile_picture \
0      moderate                  f
1      moderate                  f
2      moderate                  t
3      moderate                  f
4      flexible                  f

require_guest_phone_verification calculated_host_listings_count \
0                      f                  1
1                      f                  1
2                      f                  1
3                      f                  1
4                      f                  1

reviews_per_month
0          NaN
1          1.30
2          0.47
3          1.00
4          2.25

```

[10]:

```
print(f"There are a total of {boston_listings.shape[0]} rows.")
print(f"There are a total of {boston_listings.shape[1]} columns.")
```

There are a total of 3585 rows.

There are a total of 95 columns.

## Boston Calendar

[11]:

```
boston_calendar = pd.read_csv('boston_airbnb/calendar.csv')
boston_calendar.head(5)
```

[11]:

|   | listing_id | date       | available | price |
|---|------------|------------|-----------|-------|
| 0 | 12147973   | 2017-09-05 | f         | NaN   |
| 1 | 12147973   | 2017-09-04 | f         | NaN   |
| 2 | 12147973   | 2017-09-03 | f         | NaN   |
| 3 | 12147973   | 2017-09-02 | f         | NaN   |
| 4 | 12147973   | 2017-09-01 | f         | NaN   |

[12]:

```
print(f"There are a total of {boston_calendar.shape[0]} rows.")
print(f"There are a total of {boston_calendar.shape[1]} columns.")
```

There are a total of 1308890 rows.

There are a total of 4 columns.

[13]:

```
boston_min_date = boston_calendar.date.min()
boston_max_date = boston_calendar.date.max()
```

```
print(f"The date range for Seattle's listings is {boston_min_date} to {boston_max_date}.")
```

The date range for Seattle's listings is 2016-09-06 to 2017-09-05.

## Boston Reviews

```
[14]: boston_reviews = pd.read_csv('boston_airbnb/reviews.csv')
boston_reviews.head()
```

```
[14]:   listing_id      id      date  reviewer_id reviewer_name \
0      1178162  4724140  2013-05-21      4298113        Olivier
1      1178162  4869189  2013-05-29      6452964       Charlotte
2      1178162  5003196  2013-06-06      6449554      Sebastian
3      1178162  5150351  2013-06-15      2215611        Marine
4      1178162  5171140  2013-06-16      6848427        Andrew

   comments
0  My stay at islam's place was really cool! Good...
1  Great location for both airport and city - gre...
2  We really enjoyed our stay at Islams house. Fr...
3  The room was nice and clean and so were the co...
4  Great location. Just 5 mins walk from the Airp...
```

```
[15]: print(f"There are a total of {boston_reviews.shape[0]} rows.")
print(f"There are a total of {boston_reviews.shape[1]} columns.")
```

There are a total of 68275 rows.

There are a total of 6 columns.

Notes > 1. Looking at each individual city's listings.csv, Seattle has slightly more unique listings than Boston - about 300 more. 2. For both cities, it seems that we are examining 1 year worth of data. Furthermore, this isn't the most recent dataset. As shown in the calendar.csv dataset of each city, we are examining data from the year 2016-17; the start dates are different. 3. There seems to be significantly more reviews for Seattle than Boston. This could be an indication that each property in Seattle has a shorter occupation time on average. 4. In the Boston dataset, the neighbourhood\_group\_cleansed column is full of nulls, so it would be better to use the neighbourhood\_cleansed column to compare by neighbourhood.

Questions Taking a rough look at all the datasets, here are some questions that I am interested in solving over the course of this project:

- > 1. What are the important amenities of these listings? Compare the two cities.
- 2. Is it possible to predict the price with 8 features? If yes, compare the 2 cities.
- 3. How does the price in each city change each month? Be sure to compare the 2 cities.
- 4. How does total number of listings change each month? Be sure to compare the 2 cities.

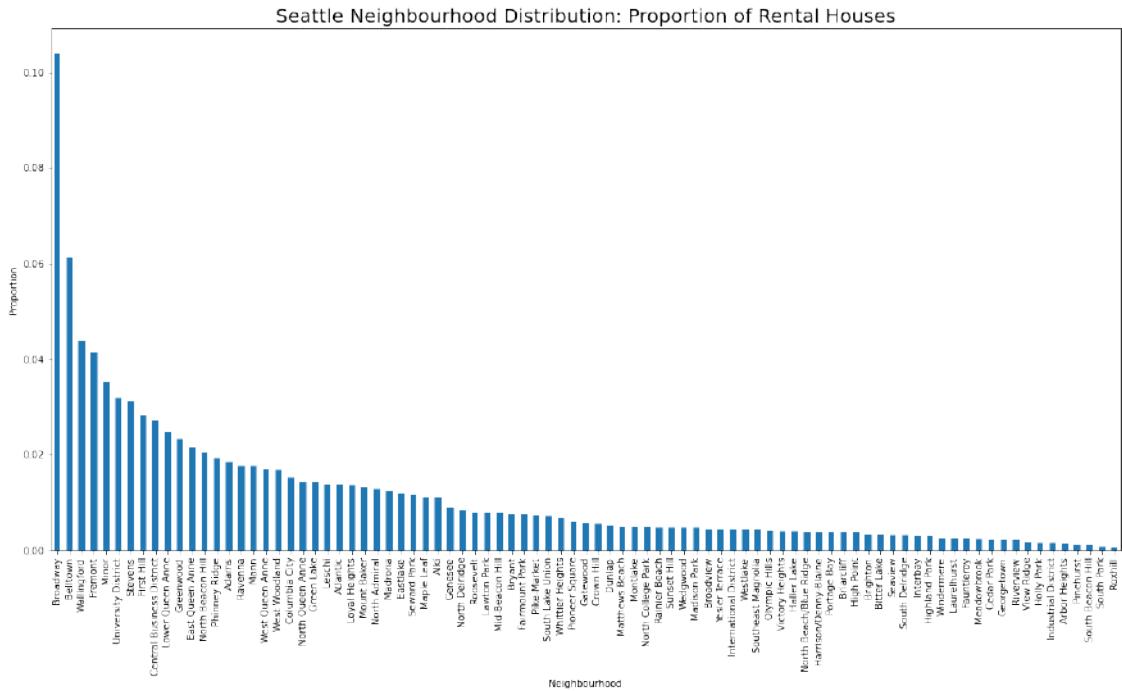
## 1.2 Step 2: Data Understanding

Before getting into the fun and dirty work, it's important that I derive some understanding of each dataset. In this process, I do not expect to get any questions answered during this step, but there is always an unexpected chance. I might even have more questions!

### 1.2.1 Neighbourhood Distribution in Seattle and Boston:

```
[16]: plt.rcParams['figure.figsize'] = [20, 10] #set figure size

seattle_neighbourhood = seattle_listings['neighbourhood_cleansed'].
    ↪value_counts()
(seattle_neighbourhood/seattle_listings.shape[0]).plot(kind = 'bar')
plt.title('Seattle Neighbourhood Distribution: Proportion of Rental Houses', ↪
    ↪fontsize = 20)
plt.savefig('Seattle Neighbourhood Distribution.png')
plt.xlabel('Neighbourhood')
plt.ylabel('Proportion');
```

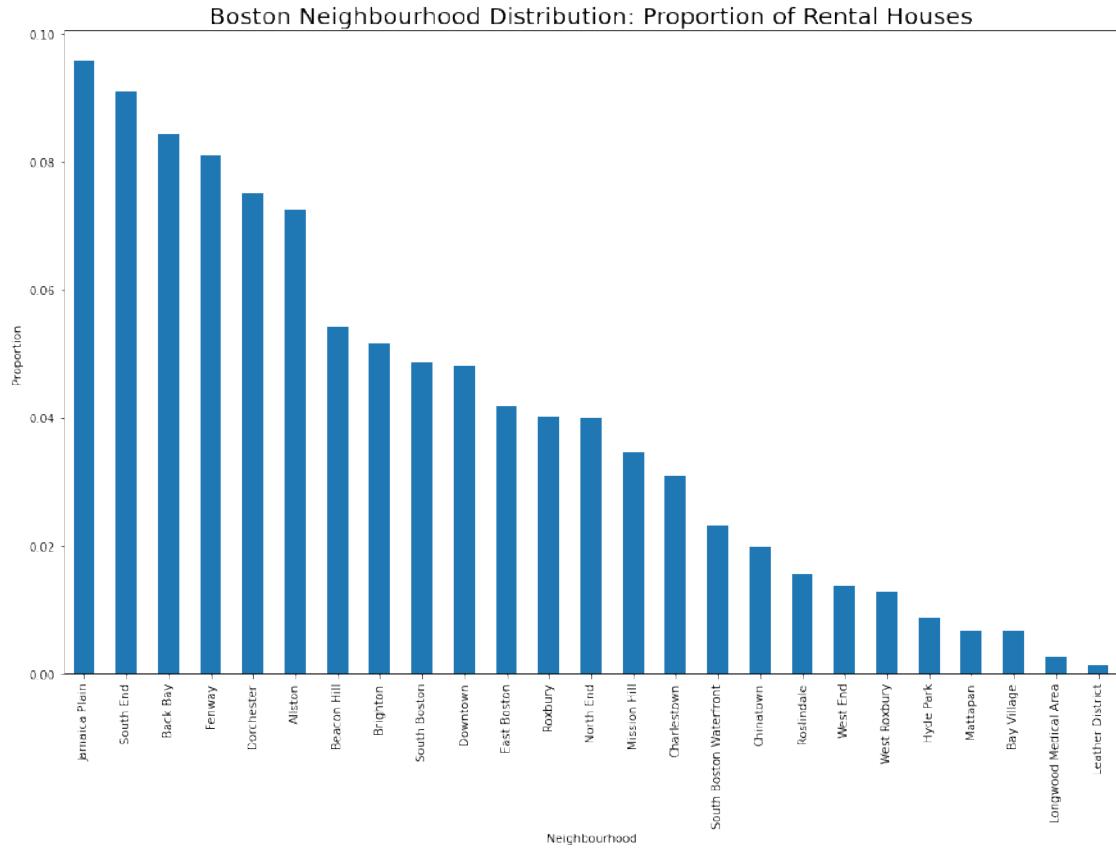


```
[17]: plt.rcParams['figure.figsize'] = [16, 10] #set figure size  
  
boston_neighbourhood = boston_listings['neighbourhood_cleansed'].value_counts()  
(boston_neighbourhood/boston_listings.shape[0]).plot(kind = 'bar')
```

```

plt.title('Boston Neighbourhood Distribution: Proportion of Rental Houses', fontweight='bold', fontsize = 20)
plt.savefig('Boston Neighbourhood Distribution.png')
plt.xlabel('Neighbourhood')
plt.ylabel('Proportion');

```



**Neighbourhood Notes:** > 1. There are a greater number of neighbourhoods in Seattle than Boston. This could simply be that there are far more neighbourhoods in Seattle than Boston. 2. For Seattle, the top neighbourhood(Broadway) is the only one that takes up a double-digit percentage. The next top neighbourhood is only half of Broadway's proportion.

### 1.2.2 Boxplot: Prices Range By Neighbourhood

[18]: boston\_listings['price'].sort\_values() #need to clean up the values before we can plot and analyze

[18]: 1764      \$1,000.00  
3242      \$1,000.00  
1896      \$1,235.00

```
3096    $1,250.00
1262    $1,250.00
...
805      $99.00
1560     $99.00
1112     $99.00
879      $99.00
2485     $999.00
Name: price, Length: 3585, dtype: object
```

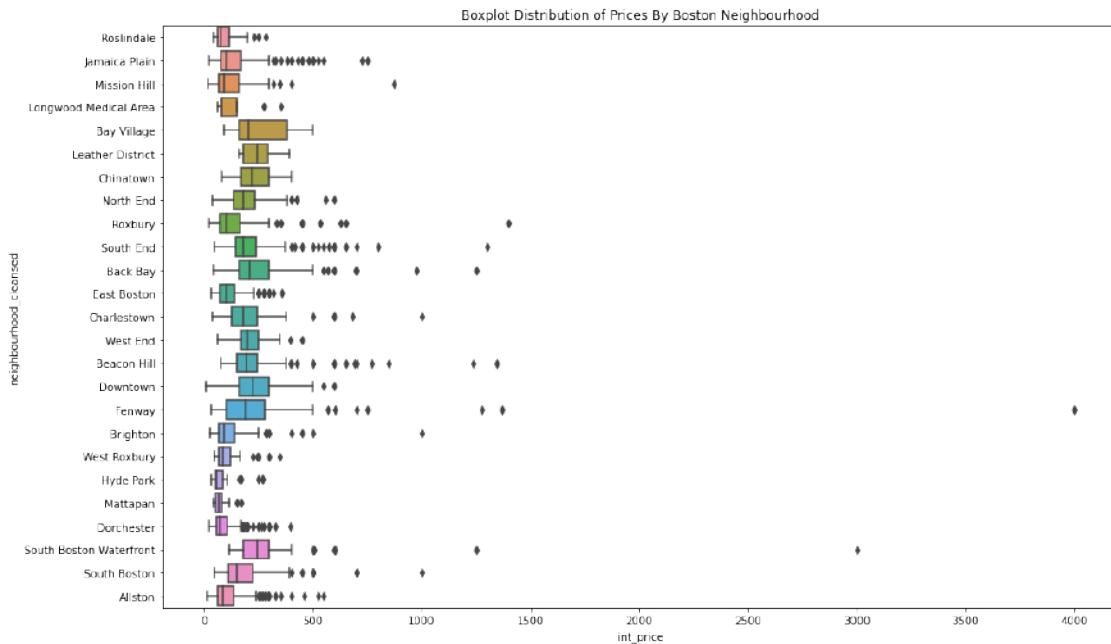
```
[23]: #Get rid of "," and convert to 'price' column to integer
#boston_listings['int_price'] = boston_listings['price'].map(lambda x: int(x[1:-3].replace(",","")))
```

```
[20]: def clean_price(df):
    """
    INPUT:
    df - pandas dataframe with categorical variables that I want to clean

    OUTPUT:
    df - a new dataframe with the following characteristics:
        1. new column with cleaned price as an integer
    """
    df['int_price'] = df['price'].map(lambda x: int(x[1:-3].replace(",","")))
    return df
```

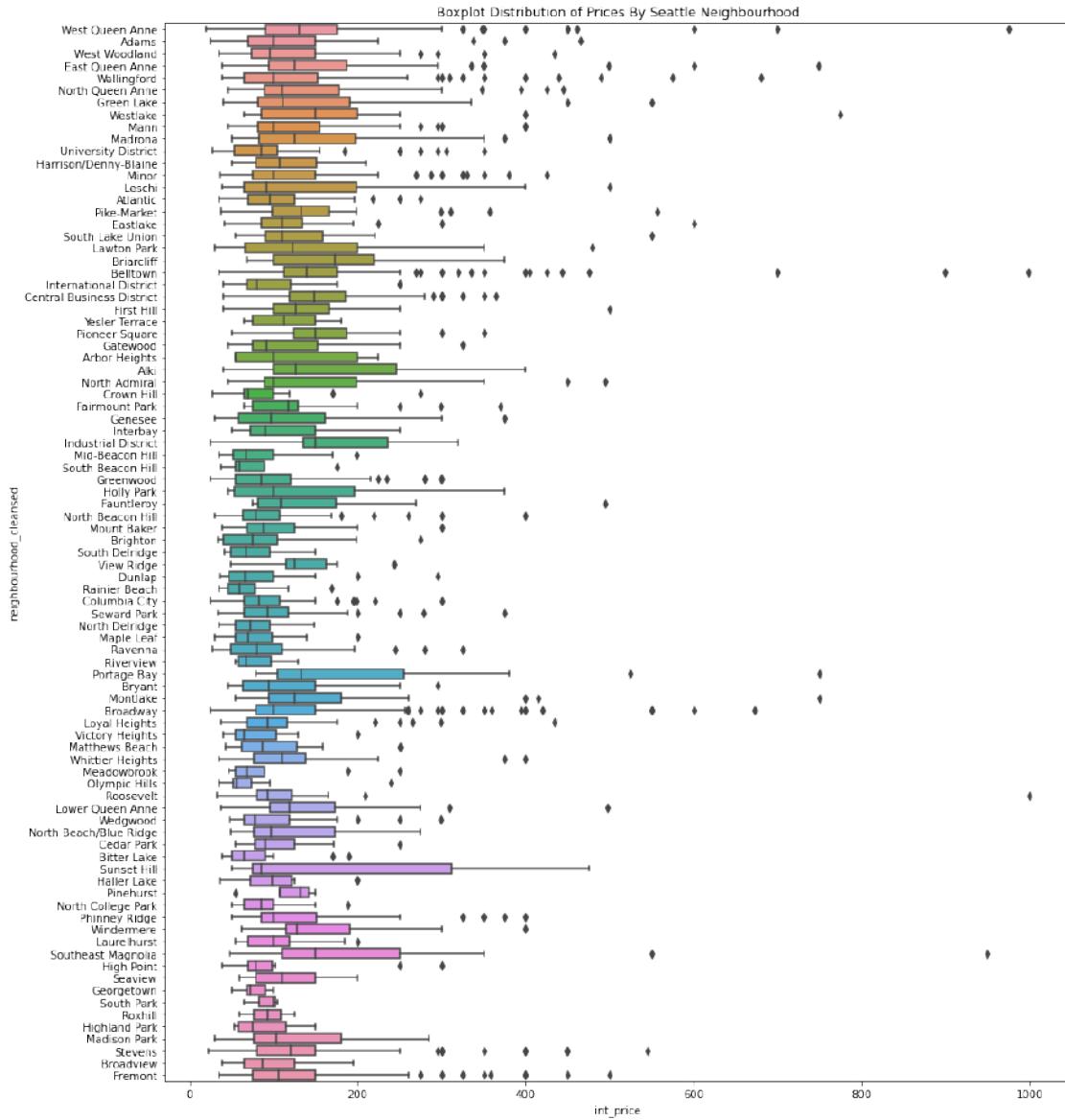
```
[21]: df = boston_listings
df = clean_price(df)

plt.rcParams['figure.figsize'] = [16, 10] #set figure size
sb.boxplot(x = 'int_price', y = 'neighbourhood_cleaned', data = df)
plt.title("Boxplot Distribution of Prices By Boston Neighbourhood");
```



```
[22]: df = seattle_listings
df = clean_price(df)

plt.rcParams['figure.figsize'] = [15, 18] #set figure size
sb.boxplot(x = 'int_price', y = 'neighbourhood_cleaned', data = df)
plt.title("Boxplot Distribution of Prices By Seattle Neighbourhood");
```



**Neighbourhood and Prices Notes:**

- There are quite a number of outliers in both datasets. However, when comparing the 2 cities visually, we can tell that Seattle neighbourhoods' rentals are more expensive on average, along with a much wider range(IQR).
- However, it's a little hard to examine because of the many outliers, so I will be trimming the outliers to get a better look. I will only be interested in those less than or equal to the largest overall max of the IQR of each city. Rather than using the `describe()` function, we will derive it visually.

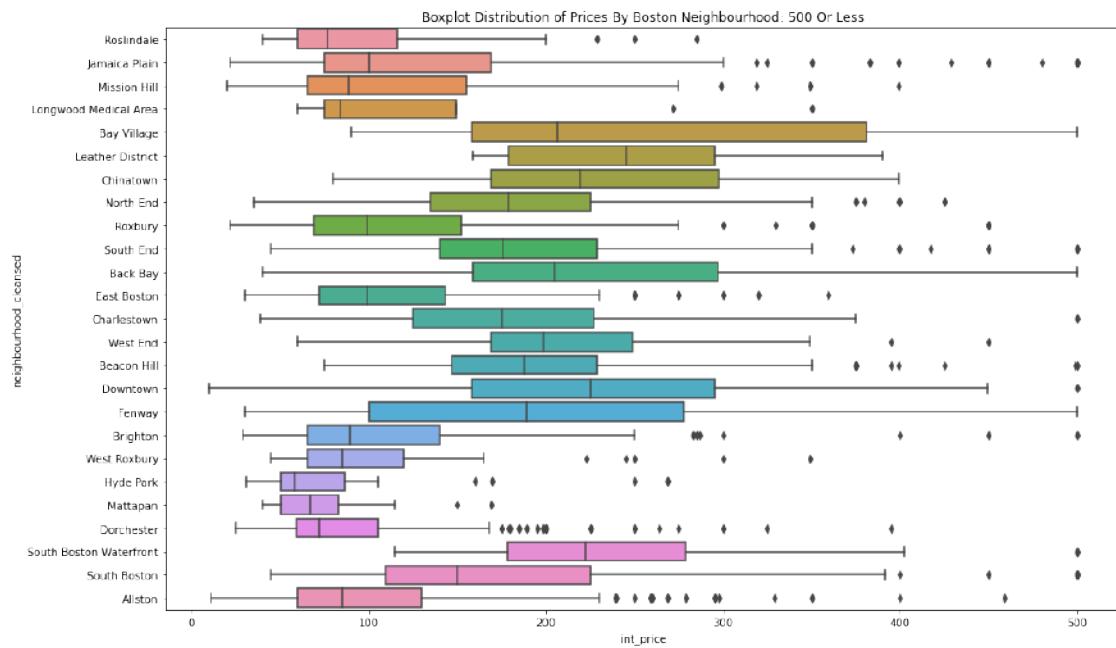
```
[24]: df = boston_listings
df = clean_price(df)

plt.rcParams['figure.figsize'] = [16, 10] #set figure size
```

```

#Get rid of "," and convert to 'price' column to integer
#boston_listings['int_price'] = boston_listings['price'].map(lambda x: int(x[1:-3].replace(",","")))
sb.boxplot(x = 'int_price', y = 'neighbourhood_cleaned',
           data = boston_listings[df['int_price'] <= 500])
plt.title("Boxplot Distribution of Prices By Boston Neighbourhood: 500 Or Less")
plt.savefig('Boxplot Distribution of Prices By Boston Neighbourhood.png');

```

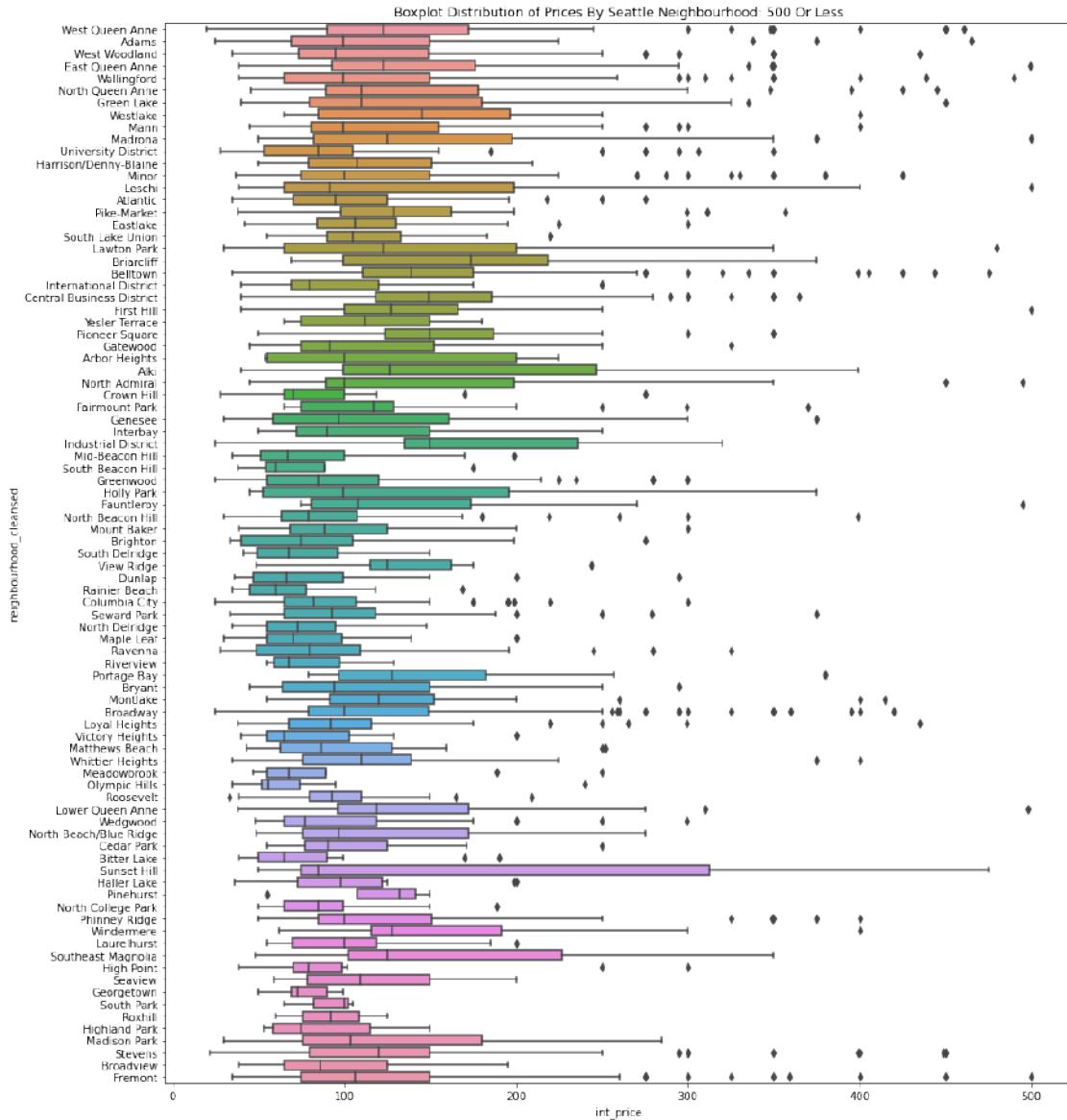


```

[25]: df = seattle_listings
df = clean_price(df)

plt.rcParams['figure.figsize'] = [15, 18] #set figure size
#Get rid of "," and convert to 'price' column to integer
#seattle_listings['int_price'] = seattle_listings['price'].map(lambda x:int(x[1:-3].replace(",","")))
sb.boxplot(x = 'int_price', y = 'neighbourhood_cleaned',
           data = seattle_listings[df['int_price'] <= 500])
plt.title("Boxplot Distribution of Prices By Seattle Neighbourhood: 500 Or Less")
plt.savefig('Boxplot Distribution of Prices By Seattle Neighbourhood.png');

```



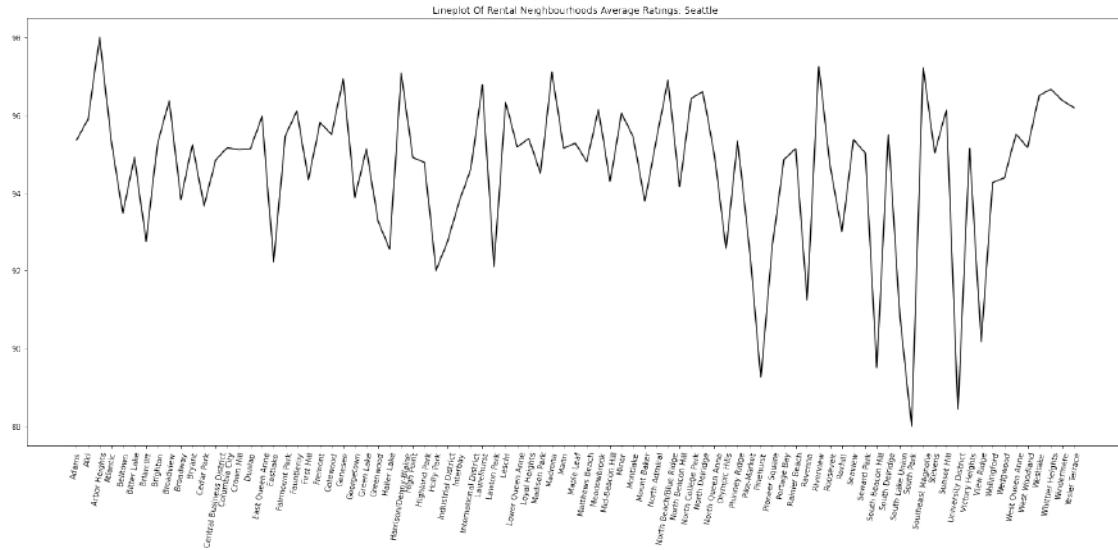
More Neighbourhood and Prices Notes: >1. Now that we have a better look, we can see that the Leather District neighbourhood has the highest mean price for the city of Boston. For the city of Seattle, we can see that the mean prices are generally cheaper and are much closer to one another. Braircliff takes the cake for highest mean price.

### 1.2.3 Lineplot Of Neighbourhoods Average Ratings

#### Seattle

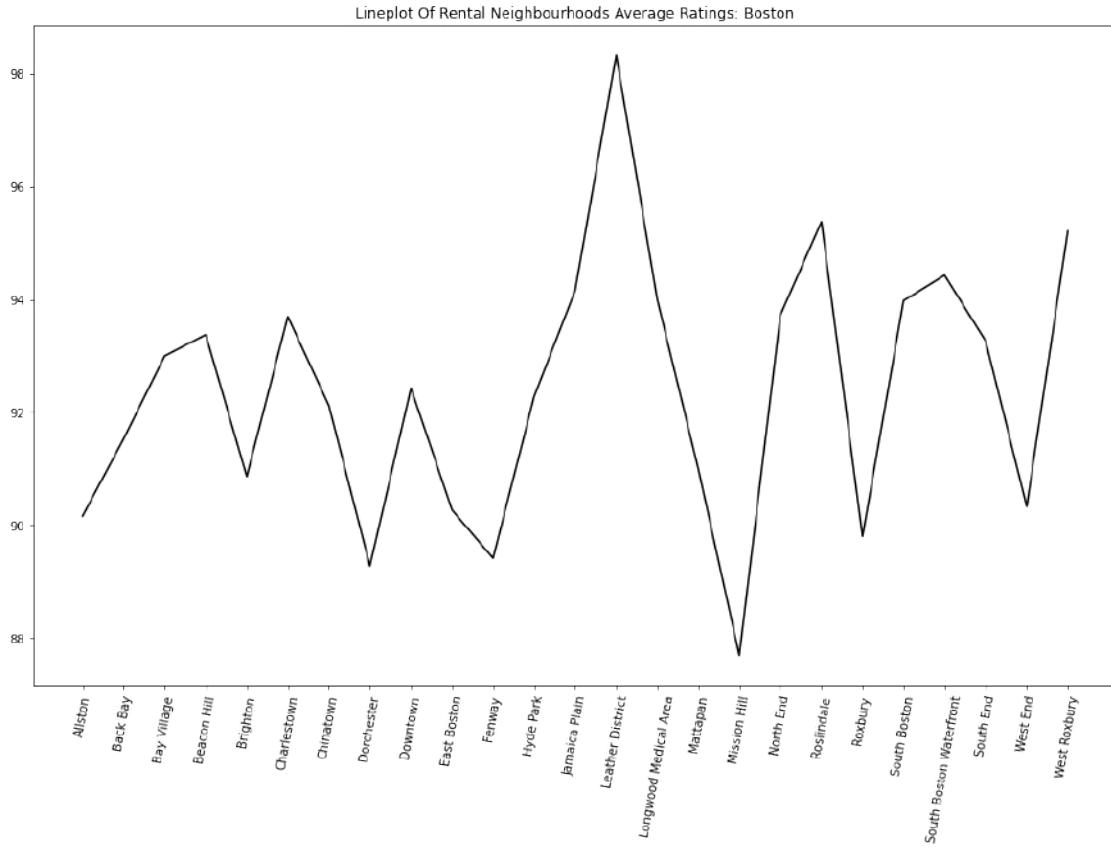
```
[26]: plt.rcParams['figure.figsize'] = [25, 10] #set figure size
neighbourhood_ratings = seattle_listings.
    ↪groupby('neighbourhood_cleansed')['review_scores_rating'].mean()
```

```
plt.plot(neighbourhood_ratings, color = "black")
plt.xticks(rotation = 80)
plt.title('Lineplot Of Rental Neighbourhoods Average Ratings: Seattle')
plt.savefig('Lineplot Of Rental Neighbourhoods Average Ratings: Seattle.png');
```



## Boston

```
[23]: plt.rcParams['figure.figsize'] = [16, 10] #set figure size
neighbourhood_ratings = boston_listings.
    ↪groupby('neighbourhood_cleaned')['review_scores_rating'].mean()
plt.plot(neighbourhood_ratings, color = 'black')
plt.xticks(rotation = 80)
plt.title('Lineplot Of Rental Neighbourhoods Average Ratings: Boston')
plt.savefig('Lineplot Of Rental Neighbourhoods Average Ratings: Boston.png');
```



Average Rating Notes: >1. For the plot we created of average rating of each Seattle neighbourhood, we can see that the average ratings are pretty high, mostly hovering around 93. The highest rating belongs to the Alki neighbourhood. 2. For the Boston lineplot, we can see that most of the averages are somewhat lower - hovering between 90 to 94. The highest rating belongs to the Leather District which also happens to have the highest average price. The lowest rating belongs to Mission Hill.

#### 1.2.4 Examining One Listing From Each Dataset

I will be examining a single listing from each city by using the listing id column.

##### Boston

```
[24]: boston_listings.head(5) # let's examine id = 6976
```

```
[24]:      id          listing_url      scrape_id \
0  12147973  https://www.airbnb.com/rooms/12147973  20160906204935
1  3075044   https://www.airbnb.com/rooms/3075044   20160906204935
2    6976     https://www.airbnb.com/rooms/6976     20160906204935
3  1436513   https://www.airbnb.com/rooms/1436513   20160906204935
4  7651065   https://www.airbnb.com/rooms/7651065   20160906204935
```

```

last_scraped                                name \
0 2016-09-07          Sunny Bungalow in the City
1 2016-09-07          Charming room in pet friendly apt
2 2016-09-07          Mexican Folk Art Haven in Boston
3 2016-09-07          Spacious Sunny Bedroom Suite in Historic Home
4 2016-09-07          Come Home to Boston

summary \
0 Cozy, sunny, family home. Master bedroom high...
1 Charming and quiet room in a second floor 1910...
2 Come stay with a friendly, middle-aged guy in ...
3 Come experience the comforts of home away from...
4 My comfy, clean and relaxing home is one block...

space \
0 The house has an open and cozy feel at the sam...
1 Small but cozy and quite room with a full size...
2 Come stay with a friendly, middle-aged guy in ...
3 Most places you find in Boston are small howev...
4 Clean, attractive, private room, one block fro...

description experiences_offered \
0 Cozy, sunny, family home. Master bedroom high...      none
1 Charming and quiet room in a second floor 1910...    none
2 Come stay with a friendly, middle-aged guy in ...   none
3 Come experience the comforts of home away from...  none
4 My comfy, clean and relaxing home is one block...  none

neighborhood_overview \
0 Roslindale is quiet, convenient and friendly. ...
1 The room is in Roslindale, a diverse and prima...
2 The LOCATION: Roslindale is a safe and diverse...
3 Roslindale is a lovely little neighborhood loc...
4 I love the proximity to downtown, the neighbor...

notes \
0           NaN
1 If you don't have a US cell phone, you can tex...
2 I am in a scenic part of Boston with a couple ...
3 Please be mindful of the property as it is old...
4 I have one roommate who lives on the lower lev...

transit \
0 The bus stop is 2 blocks away, and frequent. B...
1 Plenty of safe street parking. Bus stops a few...
2 PUBLIC TRANSPORTATION: From the house, quick p...
3 There are buses that stop right in front of th...

```

4 From Logan Airport and South Station you have...

```
access \
0 You will have access to 2 bedrooms, a living r...
1 Apt has one more bedroom (which I use) and lar...
2 I am living in the apartment during your stay, ...
3 The basement has a washer dryer and gym area. ...
4 You will have access to the front and side por...
```

```
interaction \
0 NaN
1 If I am at home, I am likely working in my hom...
2 ABOUT ME: I'm a laid-back, friendly, unmarried...
3 We do live in the house therefore might be som...
4 I love my city and really enjoy sharing it wit...
```

```
house_rules \
0 Clean up and treat the home the way you'd like...
1 Pet friendly but please confirm with me if the...
2 I encourage you to use my kitchen, cooking and...
3 - The bathroom and house are shared so please ...
4 Please no smoking in the house, porch or on th...
```

```
thumbnail_url \
0 https://a2.muscache.com/im/pictures/c0842db1-e...
1 https://a1.muscache.com/im/pictures/39327812/d...
2 https://a2.muscache.com/im/pictures/6ae8335d-9...
3 https://a2.muscache.com/im/pictures/39764190-1...
4 https://a1.muscache.com/im/pictures/97154760/8...
```

```
medium_url \
0 https://a2.muscache.com/im/pictures/c0842db1-e...
1 https://a1.muscache.com/im/pictures/39327812/d...
2 https://a2.muscache.com/im/pictures/6ae8335d-9...
3 https://a2.muscache.com/im/pictures/39764190-1...
4 https://a1.muscache.com/im/pictures/97154760/8...
```

```
picture_url \
0 https://a2.muscache.com/im/pictures/c0842db1-e...
1 https://a1.muscache.com/im/pictures/39327812/d...
2 https://a2.muscache.com/im/pictures/6ae8335d-9...
3 https://a2.muscache.com/im/pictures/39764190-1...
4 https://a1.muscache.com/im/pictures/97154760/8...
```

```
xl_picture_url host_id \
0 https://a2.muscache.com/im/pictures/c0842db1-e... 31303940
1 https://a1.muscache.com/im/pictures/39327812/d... 2572247
```

```

2 https://a2.muscache.com/im/pictures/6ae8335d-9...      16701
3 https://a2.muscache.com/im/pictures/39764190-1...      6031442
4 https://a1.muscache.com/im/pictures/97154760/8...      15396970

                           host_url host_name host_since \
0 https://www.airbnb.com/users/show/31303940  Virginia 2015-04-15
1 https://www.airbnb.com/users/show/2572247    Andrea 2012-06-07
2 https://www.airbnb.com/users/show/16701       Phil   2009-05-11
3 https://www.airbnb.com/users/show/6031442    Meghna 2013-04-21
4 https://www.airbnb.com/users/show/15396970    Linda  2014-05-11

                           host_location \
0 Boston, Massachusetts, United States
1 Boston, Massachusetts, United States
2 Boston, Massachusetts, United States
3 Boston, Massachusetts, United States
4 Boston, Massachusetts, United States

                           host_about host_response_time \
0 We are country and city connecting in our deck...          NaN
1 I live in Boston and I like to travel and have...        within an hour
2 I am a middle-aged, single male with a wide ra...        within a few hours
3 My husband and I live on the property. He's a...        within a few hours
4 I work full time for a public school district...       within an hour

host_response_rate host_acceptance_rate host_is_superhost \
0           NaN             NaN          f
1         100%            100%          f
2         100%            88%          t
3         100%            50%          f
4         100%            100%         t

                           host_thumbnail_url \
0 https://a2.muscache.com/im/pictures/5936fef0-b...
1 https://a2.muscache.com/im/users/2572247/profi...
2 https://a2.muscache.com/im/users/16701/profile...
3 https://a2.muscache.com/im/pictures/5d430cde-7...
4 https://a0.muscache.com/im/users/15396970/prof...

                           host_picture_url host_neighbourhood \
0 https://a2.muscache.com/im/pictures/5936fef0-b...      Roslindale
1 https://a2.muscache.com/im/users/2572247/profi...      Roslindale
2 https://a2.muscache.com/im/users/16701/profile...      Roslindale
3 https://a2.muscache.com/im/pictures/5d430cde-7...          NaN
4 https://a0.muscache.com/im/users/15396970/prof...      Roslindale

host_listings_count host_total_listings_count \

```

```

0          1          1
1          1          1
2          1          1
3          1          1
4          1          1

host_verifications host_has_profile_pic \
0      ['email', 'phone', 'facebook', 'reviews']          t
1  ['email', 'phone', 'facebook', 'linkedin', 'am...']      t
2      ['email', 'phone', 'reviews', 'jumio']            t
3      ['email', 'phone', 'reviews']            t
4      ['email', 'phone', 'reviews', 'kba']            t

host_identity_verified   street \
0          f      Birch Street, Boston, MA 02131, United States
1          t      Pinehurst Street, Boston, MA 02131, United States
2          t      Ardale St., Boston, MA 02131, United States
3          f      Boston, MA, United States
4          t      Durnell Avenue, Boston, MA 02131, United States

neighbourhood neighbourhood_cleansed neighbourhood_group_cleansed city \
0  Roslindale           Roslindale           NaN  Boston
1  Roslindale           Roslindale           NaN  Boston
2  Roslindale           Roslindale           NaN  Boston
3  NaN                 Roslindale           NaN  Boston
4  Roslindale           Roslindale           NaN  Boston

state zipcode market smart_location country_code       country   latitude \
0  MA    02131  Boston  Boston, MA        US  United States  42.282619
1  MA    02131  Boston  Boston, MA        US  United States  42.286241
2  MA    02131  Boston  Boston, MA        US  United States  42.292438
3  MA    NaN    Boston  Boston, MA        US  United States  42.281106
4  MA    02131  Boston  Boston, MA        US  United States  42.284512

longitude is_location_exact property_type      room_type accommodates \
0 -71.133068          t      House  Entire home/apt        4
1 -71.134374          t      Apartment  Private room        2
2 -71.135765          t      Apartment  Private room        2
3 -71.121021          f      House  Private room        4
4 -71.136258          t      House  Private room        2

bathrooms bedrooms beds bed_type \
0      1.5      2.0  3.0  Real Bed
1      1.0      1.0  1.0  Real Bed
2      1.0      1.0  1.0  Real Bed
3      1.0      1.0  2.0  Real Bed
4      1.5      1.0  2.0  Real Bed

```

```

amenities    square_feet    price  \
0  {TV,"Wireless Internet",Kitchen,"Free Parking ...      NaN  $250.00
1  {TV,Internet,"Wireless Internet","Air Conditio...      NaN  $65.00
2  {TV,"Cable TV","Wireless Internet","Air Condit...      NaN  $65.00
3  {TV,Internet,"Wireless Internet","Air Conditio...      NaN  $75.00
4  {Internet,"Wireless Internet","Air Conditionin...      NaN  $79.00

weekly_price monthly_price security_deposit cleaning_fee  guests_included  \
0          NaN            NaN            NaN        $35.00                  1
1      $400.00            NaN        $95.00        $10.00                  0
2      $395.00      $1,350.00            NaN            NaN                  1
3          NaN            NaN        $100.00        $50.00                  2
4          NaN            NaN            NaN        $15.00                  1

extra_people minimum_nights maximum_nights calendar_updated  \
0      $0.00                2           1125   2 weeks ago
1      $0.00                2             15   a week ago
2      $20.00                3             45   5 days ago
3      $25.00                1           1125   a week ago
4      $0.00                2             31   2 weeks ago

has_availability availability_30 availability_60 availability_90  \
0          NaN                0                 0                  0
1          NaN               26                 54                 84
2          NaN               19                 46                 61
3          NaN                6                 16                 26
4          NaN               13                 34                 59

availability_365 calendar_last_scraped number_of_reviews first_review  \
0              0       2016-09-06                  0            NaN
1            359       2016-09-06                 36  2014-06-01
2            319       2016-09-06                 41  2009-07-19
3              98       2016-09-06                  1  2016-08-28
4            334       2016-09-06                 29  2015-08-18

last_review review_scores_rating review_scores_accuracy  \
0          NaN            NaN            NaN
1  2016-08-13            94.0            10.0
2  2016-08-05            98.0            10.0
3  2016-08-28           100.0            10.0
4  2016-09-01            99.0            10.0

review_scores_cleanliness review_scores_checkin  \
0            NaN            NaN
1            9.0            10.0
2            9.0            10.0

```

```

3           10.0          10.0
4           10.0          10.0

    review_scores_communication  review_scores_location  review_scores_value \
0                  NaN            NaN             NaN
1                  10.0           9.0             9.0
2                  10.0           9.0            10.0
3                  10.0          10.0            10.0
4                  10.0           9.0            10.0

    requires_license  license  jurisdiction_names instant_bookable \
0                 f      NaN            NaN            f
1                 f      NaN            NaN            t
2                 f      NaN            NaN            f
3                 f      NaN            NaN            f
4                 f      NaN            NaN            f

    cancellation_policy require_guest_profile_picture \
0        moderate            f
1        moderate            f
2        moderate            t
3        moderate            f
4       flexible            f

    require_guest_phone_verification calculated_host_listings_count \
0                      f            1
1                      f            1
2                      f            1
3                      f            1
4                      f            1

    reviews_per_month  int_price
0            NaN        250
1           1.30         65
2           0.47         65
3           1.00         75
4           2.25         79

```

```
[25]: boston_calendar[boston_calendar['listing_id'] == 6976].head(25)
```

```
[25]:   listing_id      date available  price
 730      6976  2017-05-12      t $65.00
 731      6976  2017-05-11      t $65.00
 732      6976  2017-05-10      t $65.00
 733      6976  2017-05-09      t $65.00
 734      6976  2017-05-08      t $65.00
 735      6976  2017-05-07      t $65.00
```

```

736      6976 2017-05-06      t $65.00
737      6976 2017-05-05      t $65.00
738      6976 2017-05-04      t $65.00
739      6976 2017-05-03      t $65.00
740      6976 2017-05-02      t $65.00
741      6976 2017-05-01      t $65.00
742      6976 2017-04-30      t $65.00
743      6976 2017-04-29      t $65.00
744      6976 2017-04-28      t $65.00
745      6976 2017-04-27      t $65.00
746      6976 2017-04-26      t $65.00
747      6976 2017-04-25      t $65.00
748      6976 2017-04-24      t $65.00
749      6976 2017-04-23      t $65.00
750      6976 2017-04-22      t $65.00
751      6976 2017-04-21      t $65.00
752      6976 2017-04-20      t $65.00
753      6976 2017-04-19      t $65.00
754      6976 2017-04-18      t $65.00

```

```
[26]: boston_reviews[boston_reviews['listing_id'] == 6976].head(25)
```

|       | listing_id | id       | date       | reviewer_id | reviewer_name |
|-------|------------|----------|------------|-------------|---------------|
| 57563 | 6976       | 5808     | 2009-07-19 | 23549       | Gary          |
| 57564 | 6976       | 67912    | 2010-07-23 | 154097      | Dominik       |
| 57565 | 6976       | 277461   | 2011-05-23 | 232785      | Aviva         |
| 57566 | 6976       | 341306   | 2011-06-28 | 579421      | Daniela       |
| 57567 | 6976       | 393798   | 2011-07-24 | 824245      | Carlos        |
| 57568 | 6976       | 660258   | 2011-10-24 | 1246312     | Kei           |
| 57569 | 6976       | 685634   | 2011-11-01 | 812255      | Stephanie     |
| 57570 | 6976       | 1382998  | 2012-05-30 | 1634171     | Laura         |
| 57571 | 6976       | 1591064  | 2012-06-29 | 2643567     | Prajakt       |
| 57572 | 6976       | 2034401  | 2012-08-19 | 3143289     | Maciek        |
| 57573 | 6976       | 2523075  | 2012-10-06 | 1295674     | Aor           |
| 57574 | 6976       | 2574957  | 2012-10-10 | 3421694     | Lauri         |
| 57575 | 6976       | 3467426  | 2013-02-03 | 4699326     | Robert        |
| 57576 | 6976       | 3899188  | 2013-03-25 | 4699326     | Robert        |
| 57577 | 6976       | 6186719  | 2013-08-02 | 7764124     | Settimio      |
| 57578 | 6976       | 7702734  | 2013-09-30 | 7785147     | Chey          |
| 57579 | 6976       | 7952430  | 2013-10-09 | 9034365     | Mahavir       |
| 57580 | 6976       | 8095574  | 2013-10-15 | 1644716     | Angel         |
| 57581 | 6976       | 8273252  | 2013-10-22 | 8541589     | Anthony       |
| 57582 | 6976       | 12094709 | 2014-04-22 | 10932793    | Bill          |
| 57583 | 6976       | 12966551 | 2014-05-15 | 4699326     | Robert        |
| 57584 | 6976       | 13120367 | 2014-05-19 | 8364428     | Jennifer      |
| 57585 | 6976       | 17602469 | 2014-08-14 | 12803661    | Ramon         |
| 57586 | 6976       | 17809584 | 2014-08-17 | 19675677    | Vlad          |

57587

6976 18573887 2014-08-28

19195786

Troy

## comments

57563 A Wonderful, pleasant, and charming host. The...

57564 Firstly Phil is a great host! He's very helpfu...

57565 Phil was a great host and his home was very co...

57566 Phil is a very friendly host and I felt safe a...

57567 Firstly, the room itself was very comfortable ...

57568 The room was very comfortable, with clean and ...

57569 Phil was very accommodating to both my short n...

57570 Phil was a great host. He was accomodating an...

57571 Phil is a wonderful host. The house is located...

57572 My stay at Phil's was great. Phil is a great h...

57573 If you look for a place that the price is very...

57574 Phil is a great host who goes over and above t...

57575 I was in Boston on a business trip that was sc...

57576 This was my second stay at Phil's place while ...

57577 Phil was great, helpful and very kind during m...

57578 I could not be happier with my first Airbnb ex...

57579 Staying at Phil's was an excellent way to star...

57580 Phil is an excellent host, he went above and b...

57581 The reservation was canceled 3 days before arr...

57582 Phil was quick to respond and to reach out bef...

57583 Very quiet neighborhood and Phil opens up his ...

57584 Phil was a wonderful host and was very accommo...

57585 Phil was an extremely pleasant and kind host. ...

57586 Great place! Cozy room. Very safe area. Phil i...

57587 I couldn't have asked for a better host for my...

**Boston ID Notes - Data Understanding:** >1. For this particulair Boston property, we can see that the owner seem to have a difficult time renting out their property. However, after looking at their acceptance rate, this could be a reason as to why. 2. I do not have any comments about the reviews database.

## Seattle

```
[27]: seattle_listings.head(5) # let's examine id = 241032
```

```
[27]:      id          listing_url      scrape_id last_scraped \
0    241032  https://www.airbnb.com/rooms/241032  20160104002432  2016-01-04
1    953595  https://www.airbnb.com/rooms/953595  20160104002432  2016-01-04
2   3308979  https://www.airbnb.com/rooms/3308979  20160104002432  2016-01-04
3   7421966  https://www.airbnb.com/rooms/7421966  20160104002432  2016-01-04
4   278830  https://www.airbnb.com/rooms/278830  20160104002432  2016-01-04
```

|   | name \                             |
|---|------------------------------------|
| 0 | Stylish Queen Anne Apartment       |
| 1 | Bright & Airy Queen Anne Apartment |

```

2 New Modern House-Amazing water view
3             Queen Anne Chateau
4     Charming craftsman 3 bdm house

summary \
0           NaN
1 Chemically sensitive? We've removed the irrita...
2 New modern house built in 2013. Spectacular s...
3 A charming apartment that sits atop Queen Anne...
4 Cozy family craftsman house in beautiful neigh...

space \
0 Make your self at home in this charming one-be...
1 Beautiful, hypoallergenic apartment in an extr...
2 Our house is modern, light and fresh with a wa...
3           NaN
4 Cozy family craftsman house in beautiful neigh...

description experiences_offered \
0 Make your self at home in this charming one-be...      none
1 Chemically sensitive? We've removed the irrita...      none
2 New modern house built in 2013. Spectacular s...      none
3 A charming apartment that sits atop Queen Anne...      none
4 Cozy family craftsman house in beautiful neigh...      none

neighborhood_overview \
0           NaN
1 Queen Anne is a wonderful, truly functional vi...
2 Upper Queen Anne is a charming neighborhood fu...
3           NaN
4 We are in the beautiful neighborhood of Queen ...

notes \
0           NaN
1 What's up with the free pillows? Our home was...
2 Our house is located just 5 short blocks to To...
3           NaN
4           Belltown

transit \
0           NaN
1 Convenient bus stops are just down the block, ...
2 A bus stop is just 2 blocks away. Easy bus a...
3           NaN
4 The nearest public transit bus (D Line) is 2 b...

thumbnail_url \

```

```

0                               NaN
1 https://a0.muscache.com/ac/pictures/14409893/f...      \
2                               NaN
3                               NaN
4                               NaN

                           medium_url  \
0                               NaN
1 https://a0.muscache.com/im/pictures/14409893/f...      \
2                               NaN
3                               NaN
4                               NaN

                           picture_url  \
0 https://a1.muscache.com/ac/pictures/67560560/c...
1 https://a0.muscache.com/ac/pictures/14409893/f...
2 https://a2.muscache.com/ac/pictures/b4324e0f-a...
3 https://a0.muscache.com/ac/pictures/94146944/6...
4 https://a1.muscache.com/ac/pictures/6120468/b0...

                           xl_picture_url  host_id  \
0                               NaN  956883
1 https://a0.muscache.com/ac/pictures/14409893/f...  5177328
2                               NaN  16708587
3                               NaN  9851441
4                               NaN  1452570

                           host_url host_name  host_since  \
0 https://www.airbnb.com/users/show/956883      Maija  2011-08-11
1 https://www.airbnb.com/users/show/5177328      Andrea 2013-02-21
2 https://www.airbnb.com/users/show/16708587      Jill   2014-06-12
3 https://www.airbnb.com/users/show/9851441      Emily   2013-11-06
4 https://www.airbnb.com/users/show/1452570      Emily   2011-11-29

                           host_location  \
0 Seattle, Washington, United States
1 Seattle, Washington, United States
2 Seattle, Washington, United States
3 Seattle, Washington, United States
4 Seattle, Washington, United States

                           host_about  host_response_time  \
0 I am an artist, interior designer, and run a s...  within a few hours
1 Living east coast/left coast/overseas. Time i...  within an hour
2 i love living in Seattle. i grew up in the mi...  within a few hours
3                               NaN                  NaN
4 Hi, I live in Seattle, Washington but I'm orig...  within an hour

```

```

host_response_rate host_acceptance_rate host_is_superhost \
0 96% 100% f
1 98% 100% t
2 67% 100% f
3 NaN NaN f
4 100% NaN f

host_thumbnail_url \
0 https://a0.muscache.com/ac/users/956883/profil...
1 https://a0.muscache.com/ac/users/5177328/profi...
2 https://a1.muscache.com/ac/users/16708587/prof...
3 https://a2.muscache.com/ac/users/9851441/profi...
4 https://a0.muscache.com/ac/users/1452570/profi...

host_picture_url host_neighbourhood \
0 https://a0.muscache.com/ac/users/956883/profil... Queen Anne
1 https://a0.muscache.com/ac/users/5177328/profi... Queen Anne
2 https://a1.muscache.com/ac/users/16708587/prof... Queen Anne
3 https://a2.muscache.com/ac/users/9851441/profi... Queen Anne
4 https://a0.muscache.com/ac/users/1452570/profi... Queen Anne

host_listings_count host_total_listings_count \
0 3.0 3.0
1 6.0 6.0
2 2.0 2.0
3 1.0 1.0
4 2.0 2.0

host_verifications host_has_profile_pic \
0 ['email', 'phone', 'reviews', 'kba'] t
1 ['email', 'phone', 'facebook', 'linkedin', 're... t
2 ['email', 'phone', 'google', 'reviews', 'jumio'] t
3 ['email', 'phone', 'facebook', 'reviews', 'jum... t
4 ['email', 'phone', 'facebook', 'reviews', 'kba'] t

host_identity_verified street \
0 t Gilman Dr W, Seattle, WA 98119, United States
1 t 7th Avenue West, Seattle, WA 98119, United States
2 t West Lee Street, Seattle, WA 98119, United States
3 t 8th Avenue West, Seattle, WA 98119, United States
4 t 14th Ave W, Seattle, WA 98119, United States

neighbourhood neighbourhood_cleansed neighbourhood_group_cleansed city \
0 Queen Anne West Queen Anne Queen Anne Seattle
1 Queen Anne West Queen Anne Queen Anne Seattle
2 Queen Anne West Queen Anne Queen Anne Seattle

```

|   | Queen Anne                                        | West Queen Anne | Queen Anne        | Seattle          |                 |               |
|---|---------------------------------------------------|-----------------|-------------------|------------------|-----------------|---------------|
|   | Queen Anne                                        | West Queen Anne | Queen Anne        | Seattle          |                 |               |
| 0 | WA                                                | 98119           | Seattle           | Seattle, WA      | US              | United States |
| 1 | WA                                                | 98119           | Seattle           | Seattle, WA      | US              | United States |
| 2 | WA                                                | 98119           | Seattle           | Seattle, WA      | US              | United States |
| 3 | WA                                                | 98119           | Seattle           | Seattle, WA      | US              | United States |
| 4 | WA                                                | 98119           | Seattle           | Seattle, WA      | US              | United States |
|   | latitude                                          | longitude       | is_location_exact | property_type    | room_type       | \             |
| 0 | 47.636289                                         | -122.371025     | t                 | Apartment        | Entire home/apt |               |
| 1 | 47.639123                                         | -122.365666     | t                 | Apartment        | Entire home/apt |               |
| 2 | 47.629724                                         | -122.369483     | t                 | House            | Entire home/apt |               |
| 3 | 47.638473                                         | -122.369279     | t                 | Apartment        | Entire home/apt |               |
| 4 | 47.632918                                         | -122.372471     | t                 | House            | Entire home/apt |               |
|   | accommodates                                      | bathrooms       | bedrooms          | beds             | bed_type        | \             |
| 0 | 4                                                 | 1.0             | 1.0               | 1.0              | Real Bed        |               |
| 1 | 4                                                 | 1.0             | 1.0               | 1.0              | Real Bed        |               |
| 2 | 11                                                | 4.5             | 5.0               | 7.0              | Real Bed        |               |
| 3 | 3                                                 | 1.0             | 0.0               | 2.0              | Real Bed        |               |
| 4 | 6                                                 | 2.0             | 3.0               | 3.0              | Real Bed        |               |
|   | amenities                                         | square_feet     | price             |                  |                 | \             |
| 0 | {TV,"Cable TV",Internet,"Wireless Internet","A... | NaN             | \$85.00           |                  |                 |               |
| 1 | {TV,Internet,"Wireless Internet",Kitchen,"Free... | NaN             | \$150.00          |                  |                 |               |
| 2 | {TV,"Cable TV",Internet,"Wireless Internet","A... | NaN             | \$975.00          |                  |                 |               |
| 3 | {Internet,"Wireless Internet",Kitchen,"Indoor ... | NaN             | \$100.00          |                  |                 |               |
| 4 | {TV,"Cable TV",Internet,"Wireless Internet",Ki... | NaN             | \$450.00          |                  |                 |               |
|   | weekly_price                                      | monthly_price   | security_deposit  | cleaning_fee     | guests_included | \             |
| 0 | NaN                                               | NaN             | NaN               | NaN              | 2               |               |
| 1 | \$1,000.00                                        | \$3,000.00      | \$100.00          | \$40.00          | 1               |               |
| 2 | NaN                                               | NaN             | \$1,000.00        | \$300.00         | 10              |               |
| 3 | \$650.00                                          | \$2,300.00      | NaN               | NaN              | 1               |               |
| 4 | NaN                                               | NaN             | \$700.00          | \$125.00         | 6               |               |
|   | extra_people                                      | minimum_nights  | maximum_nights    | calendar_updated |                 | \             |
| 0 | \$5.00                                            | 1               | 365               | 4 weeks ago      |                 |               |
| 1 | \$0.00                                            | 2               | 90                | today            |                 |               |
| 2 | \$25.00                                           | 4               | 30                | 5 weeks ago      |                 |               |
| 3 | \$0.00                                            | 1               | 1125              | 6 months ago     |                 |               |
| 4 | \$15.00                                           | 1               | 1125              | 7 weeks ago      |                 |               |
|   | has_availability                                  | availability_30 | availability_60   | availability_90  |                 | \             |
| 0 | t                                                 | 14              | 41                | 71               |                 |               |

|   |                             |                               |                        |                    |
|---|-----------------------------|-------------------------------|------------------------|--------------------|
| 1 | t                           | 13                            | 13                     | 16                 |
| 2 | t                           | 1                             | 6                      | 17                 |
| 3 | t                           | 0                             | 0                      | 0                  |
| 4 | t                           | 30                            | 60                     | 90                 |
|   | availability_365            | calendar_last_scraped         | number_of_reviews      | first_review \     |
| 0 | 346                         | 2016-01-04                    | 207                    | 2011-11-01         |
| 1 | 291                         | 2016-01-04                    | 43                     | 2013-08-19         |
| 2 | 220                         | 2016-01-04                    | 20                     | 2014-07-30         |
| 3 | 143                         | 2016-01-04                    | 0                      | NaN                |
| 4 | 365                         | 2016-01-04                    | 38                     | 2012-07-10         |
|   | last_review                 | review_scores_rating          | review_scores_accuracy | \                  |
| 0 | 2016-01-02                  | 95.0                          | 10.0                   |                    |
| 1 | 2015-12-29                  | 96.0                          | 10.0                   |                    |
| 2 | 2015-09-03                  | 97.0                          | 10.0                   |                    |
| 3 | NaN                         | NaN                           | NaN                    |                    |
| 4 | 2015-10-24                  | 92.0                          | 9.0                    |                    |
|   | review_scores_cleanliness   | review_scores_checkin         | \                      |                    |
| 0 | 10.0                        | 10.0                          |                        |                    |
| 1 | 10.0                        | 10.0                          |                        |                    |
| 2 | 10.0                        | 10.0                          |                        |                    |
| 3 | NaN                         | NaN                           |                        |                    |
| 4 | 9.0                         | 10.0                          |                        |                    |
|   | review_scores_communication | review_scores_location        | review_scores_value    | \                  |
| 0 | 10.0                        | 9.0                           | 10.0                   |                    |
| 1 | 10.0                        | 10.0                          | 10.0                   |                    |
| 2 | 10.0                        | 10.0                          | 10.0                   |                    |
| 3 | NaN                         | NaN                           | NaN                    |                    |
| 4 | 10.0                        | 9.0                           | 9.0                    |                    |
|   | requires_license            | license                       | jurisdiction_names     | instant_bookable \ |
| 0 | f                           | NaN                           | WASHINGTON             | f                  |
| 1 | f                           | NaN                           | WASHINGTON             | f                  |
| 2 | f                           | NaN                           | WASHINGTON             | f                  |
| 3 | f                           | NaN                           | WASHINGTON             | f                  |
| 4 | f                           | NaN                           | WASHINGTON             | f                  |
|   | cancellation_policy         | require_guest_profile_picture | \                      |                    |
| 0 | moderate                    |                               | f                      |                    |
| 1 | strict                      |                               | t                      |                    |
| 2 | strict                      |                               | f                      |                    |
| 3 | flexible                    |                               | f                      |                    |
| 4 | strict                      |                               | f                      |                    |

```

require_guest_phone_verification calculated_host_listings_count \
0                               f                           2
1                               t                           6
2                               f                           2
3                               f                           1
4                               f                           1

reviews_per_month int_price
0             4.07        85
1             1.48       150
2             1.15      975
3             NaN         100
4             0.89       450

```

[28]: `seattle_calendar[seattle_calendar['listing_id'] == 241032].head(25)`

```

[28]:   listing_id      date available  price
0      241032 2016-01-04      t $85.00
1      241032 2016-01-05      t $85.00
2      241032 2016-01-06      f   NaN
3      241032 2016-01-07      f   NaN
4      241032 2016-01-08      f   NaN
5      241032 2016-01-09      f   NaN
6      241032 2016-01-10      f   NaN
7      241032 2016-01-11      f   NaN
8      241032 2016-01-12      f   NaN
9      241032 2016-01-13      t $85.00
10     241032 2016-01-14      t $85.00
11     241032 2016-01-15      f   NaN
12     241032 2016-01-16      f   NaN
13     241032 2016-01-17      f   NaN
14     241032 2016-01-18      t $85.00
15     241032 2016-01-19      t $85.00
16     241032 2016-01-20      t $85.00
17     241032 2016-01-21      f   NaN
18     241032 2016-01-22      f   NaN
19     241032 2016-01-23      f   NaN
20     241032 2016-01-24      t $85.00
21     241032 2016-01-25      t $85.00
22     241032 2016-01-26      t $85.00
23     241032 2016-01-27      t $85.00
24     241032 2016-01-28      t $85.00

```

[29]: `seattle_reviews[seattle_reviews['listing_id'] == 241032].head(25)`

```

[29]:   listing_id      id      date reviewer_id reviewer_name \
46381     241032 682061 2011-11-01      479824           Bro

```

|       |        |         |            |         |             |
|-------|--------|---------|------------|---------|-------------|
| 46382 | 241032 | 691712  | 2011-11-04 | 357699  | Megan       |
| 46383 | 241032 | 702999  | 2011-11-08 | 1285567 | Marylee     |
| 46384 | 241032 | 717262  | 2011-11-14 | 647857  | Graham      |
| 46385 | 241032 | 730226  | 2011-11-19 | 1389821 | Franka      |
| 46386 | 241032 | 739809  | 2011-11-23 | 620277  | Robin       |
| 46387 | 241032 | 746462  | 2011-11-27 | 1261162 | Mara        |
| 46388 | 241032 | 755647  | 2011-11-30 | 719078  | Kiran       |
| 46389 | 241032 | 764019  | 2011-12-05 | 693268  | Tatyana     |
| 46390 | 241032 | 773602  | 2011-12-09 | 1243196 | Sal         |
| 46391 | 241032 | 784989  | 2011-12-14 | 482299  | Christopher |
| 46392 | 241032 | 795862  | 2011-12-20 | 814238  | Alison      |
| 46393 | 241032 | 815052  | 2011-12-30 | 1471446 | Wilhemina   |
| 46394 | 241032 | 854486  | 2012-01-10 | 1382578 | Sara        |
| 46395 | 241032 | 860550  | 2012-01-13 | 1266005 | Anne        |
| 46396 | 241032 | 865460  | 2012-01-16 | 1509172 | Jonathan    |
| 46397 | 241032 | 890978  | 2012-01-30 | 335342  | Diane       |
| 46398 | 241032 | 911807  | 2012-02-10 | 1544991 | James       |
| 46399 | 241032 | 932490  | 2012-02-20 | 751416  | Erna        |
| 46400 | 241032 | 945726  | 2012-02-24 | 893874  | Melanie     |
| 46401 | 241032 | 1096041 | 2012-04-08 | 1487451 | Lynne       |
| 46402 | 241032 | 1313496 | 2012-05-21 | 1600460 | Ausrine     |
| 46403 | 241032 | 1380839 | 2012-05-30 | 209246  | Meg & Josh  |
| 46404 | 241032 | 1431692 | 2012-06-07 | 2115606 | Liz         |
| 46405 | 241032 | 1506643 | 2012-06-18 | 1555983 | Sheri       |

#### comments

46381 Excellent all the way around. \r\n\r\nMaija wa...

46382 Maija's apartment was a wonderful place to sta...

46383 one of the most pleasant stays i've had in my ...

46384 Maija's suite is beautiful, cozy and convenien...

46385 Our stay was short and pleasant. With its own ...

46386 I had a wonderful 4 night stay in Maija's apar...

46387 The apartment was really nice and homey. It wa...

46388 I stayed at Maija's place for 3 days and had n...

46389 I did booking for my sister, and she liked the...

46390 We were unintentionally poor guests; we broke ...

46391 the apartment is spacious, comfortable, and ve...

46392 Maija was very welcoming to us. She showed us ...

46393 We stayed in Maija's wonderful space for two d...

46394 My husband, mom and I thoroughly enjoyed our w...

46395 My husband and I stayed here for three nights ...

46396 I was very comfortable at Maija's apartment. ...

46397 This was a perfect spot for our stay! We neede...

46398 We absolutely adored Maija and her lovely apar...

46399 Great apartment in Seattle, very comfortable. ...

46400 Maija welcomed us warmly and provided lots of ...

46401 It was great! Maija is an incredible host. We ...

```

46402 Es ist ein sehr gemütliches und schönes Appart...
46403 My husband and I visited Seattle for a confere...
46404 My college age daughter and I spent a week in ...
46405 Maija's apartment was comfortable and spacious...

```

**Seattle ID Notes - Data Understanding:** >1. In this case, the listing price is consistent with the calendar dataset price, but the interesting thing to note is the null values. From the dataset, we know that the place is unavailable. However, the ordering of the null values in the column brings up questions about price changes. Another question that pops up is “Even if they don’t take it off the market, do other owners change their prices based on the day?” 2. I do not have any comments about the reviews database. 3. Before moving on, I would like to examine one more listing that I found to be interesting.

```
[30]: seattle_listings.head(5) # let's examine id = 278830
seattle_listings[seattle_listings['id'] == 278830]
```

```

[30]:          id           listing_url      scrape_id last_scraped \
4 278830  https://www.airbnb.com/rooms/278830  20160104002432  2016-01-04

                           name \
4 Charming craftsman 3 bdm house

                           summary \
4 Cozy family craftman house in beautiful neighb...

                           space \
4 Cozy family craftman house in beautiful neighb...

                           description experiences_offered \
4 Cozy family craftman house in beautiful neighb...           none

                           neighborhood_overview      notes \
4 We are in the beautiful neighborhood of Queen ... Belltown

                           transit thumbnail_url medium_url \
4 The nearest public transit bus (D Line) is 2 b...           NaN        NaN

                           picture_url xl_picture_url host_id \
4 https://a1.muscache.com/ac/pictures/6120468/b0...           NaN  1452570

                           host_url host_name host_since \
4 https://www.airbnb.com/users/show/1452570      Emily  2011-11-29

                           host_location \
4 Seattle, Washington, United States

```

```

host_about host_response_time \
4 Hi, I live in Seattle, Washington but I'm orig... within an hour

host_response_rate host_acceptance_rate host_is_superhost \
4 100%          NaN          f

host_thumbnail_url \
4 https://a0.muscache.com/ac/users/1452570/profi...

host_picture_url host_neighbourhood \
4 https://a0.muscache.com/ac/users/1452570/profi... Queen Anne

host_listings_count host_total_listings_count \
4 2.0            2.0

host_verifications host_has_profile_pic \
4 ['email', 'phone', 'facebook', 'reviews', 'kba'] t

host_identity_verified street \
4 t 14th Ave W, Seattle, WA 98119, United States

neighbourhood neighbourhood_cleansed neighbourhood_group_cleansed city \
4 Queen Anne        West Queen Anne           Queen Anne Seattle

state zipcode market smart_location country_code country \
4 WA    98119   Seattle    Seattle, WA      US    United States

latitude longitude is_location_exact property_type room_type \
4 47.632918 -122.372471 t House Entire home/apt

accommodates bathrooms bedrooms beds bed_type \
4 6       2.0     3.0   3.0 Real Bed

amenities square_feet price \
4 {TV,"Cable TV",Internet,"Wireless Internet",Ki... NaN $450.00

weekly_price monthly_price security_deposit cleaning_fee guests_included \
4 NaN          NaN          $700.00        $125.00          6

extra_people minimum_nights maximum_nights calendar_updated \
4 $15.00        1           1125      7 weeks ago

has_availability availability_30 availability_60 availability_90 \
4 t             30          60          90

availability_365 calendar_last_scraped number_of_reviews first_review \
4 365          2016-01-04      38      2012-07-10

```

```

last_review  review_scores_rating  review_scores_accuracy  \
4  2015-10-24                  92.0                      9.0

    review_scores_cleanliness  review_scores_checkin  \
4                           9.0                      10.0

    review_scores_communication  review_scores_location  review_scores_value  \
4                               10.0                      9.0                      9.0

    requires_license  license jurisdiction_names instant_bookable  \
4                   f          NaN           WASHINGTON                  f

    cancellation_policy require_guest_profile_picture  \
4                     strict                                f

    require_guest_phone_verification calculated_host_listings_count  \
4                               f                                  1

    reviews_per_month  int_price
4             0.89            450

```

[31]: `seattle_calendar[seattle_calendar['listing_id'] == 278830].head(30)`

|      | listing_id | date       | available | price    |
|------|------------|------------|-----------|----------|
| 1460 | 278830     | 2016-01-04 | t         | \$600.00 |
| 1461 | 278830     | 2016-01-05 | t         | \$600.00 |
| 1462 | 278830     | 2016-01-06 | t         | \$600.00 |
| 1463 | 278830     | 2016-01-07 | t         | \$600.00 |
| 1464 | 278830     | 2016-01-08 | t         | \$600.00 |
| 1465 | 278830     | 2016-01-09 | t         | \$600.00 |
| 1466 | 278830     | 2016-01-10 | t         | \$500.00 |
| 1467 | 278830     | 2016-01-11 | t         | \$500.00 |
| 1468 | 278830     | 2016-01-12 | t         | \$500.00 |
| 1469 | 278830     | 2016-01-13 | t         | \$500.00 |
| 1470 | 278830     | 2016-01-14 | t         | \$500.00 |
| 1471 | 278830     | 2016-01-15 | t         | \$500.00 |
| 1472 | 278830     | 2016-01-16 | t         | \$500.00 |
| 1473 | 278830     | 2016-01-17 | t         | \$500.00 |
| 1474 | 278830     | 2016-01-18 | t         | \$500.00 |
| 1475 | 278830     | 2016-01-19 | t         | \$500.00 |
| 1476 | 278830     | 2016-01-20 | t         | \$500.00 |
| 1477 | 278830     | 2016-01-21 | t         | \$500.00 |
| 1478 | 278830     | 2016-01-22 | t         | \$500.00 |
| 1479 | 278830     | 2016-01-23 | t         | \$500.00 |
| 1480 | 278830     | 2016-01-24 | t         | \$500.00 |
| 1481 | 278830     | 2016-01-25 | t         | \$500.00 |

|      |        |            |   |          |
|------|--------|------------|---|----------|
| 1482 | 278830 | 2016-01-26 | t | \$500.00 |
| 1483 | 278830 | 2016-01-27 | t | \$500.00 |
| 1484 | 278830 | 2016-01-28 | t | \$500.00 |
| 1485 | 278830 | 2016-01-29 | t | \$500.00 |
| 1486 | 278830 | 2016-01-30 | t | \$500.00 |
| 1487 | 278830 | 2016-01-31 | t | \$500.00 |
| 1488 | 278830 | 2016-02-01 | t | \$500.00 |
| 1489 | 278830 | 2016-02-02 | t | \$500.00 |

In this case, we can see that the price in the `listings` dataset is different from the `calendar` one. In the former, the price is set at 450 while the price in the `calendar` dataset starts off at 600 and begins decreasing. This could be an indicator that the owner wants to rent out his property as fast as possible to generate income.

[ ]:

[ ]:

# airbnb-question-1-part-ii

February 24, 2024

## 1 Boston vs. Seattle Airbnb II

In this notebook(Part 2), I will be answering the question: What are the important amenities of these listings? Compare the two cities.

```
[1]: import warnings
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb
%matplotlib inline
warnings.filterwarnings('ignore')

pd.set_option('display.max_columns', None)
```

### 1.1 Question 1: What are the important amenities of these listings? Compare the two cities.

In Part I, we found out the names of each neighbourhood that are in each cities along with the prices. Please refer back to that to study the distributions of the listings. For this part, we will mainly be working on finding out the amenities and using the results to compare the 2 cities side by side with the help of a bar plot..

In order to find out the importance, we will be using the ratings column.

```
[2]: #import necessary datasets
seattle_listings = pd.read_csv('seattle_airbnb/listings.csv')
boston_listings = pd.read_csv('boston_airbnb/listings.csv')
```

#### 1.1.1 Seattle Dataset Examination

```
[3]: seattle_listings.head(2)
```

```
[3]:      id          listing_url      scrape_id last_scraped \
0  241032  https://www.airbnb.com/rooms/241032  20160104002432  2016-01-04
1  953595  https://www.airbnb.com/rooms/953595  20160104002432  2016-01-04
   name  \
```

```
0      Stylish Queen Anne Apartment
1 Bright & Airy Queen Anne Apartment

summary \
0          NaN
1 Chemically sensitive? We've removed the irrita...

space \
0 Make your self at home in this charming one-be...
1 Beautiful, hypoallergenic apartment in an extr...

description experiences_offered \
0 Make your self at home in this charming one-be...      none
1 Chemically sensitive? We've removed the irrita...      none

neighborhood_overview \
0          NaN
1 Queen Anne is a wonderful, truly functional vi...

notes \
0          NaN
1 What's up with the free pillows? Our home was...

transit \
0          NaN
1 Convenient bus stops are just down the block, ...

thumbnail_url \
0          NaN
1 https://a0.muscache.com/ac/pictures/14409893/f...

medium_url \
0          NaN
1 https://a0.muscache.com/im/pictures/14409893/f...

picture_url \
0 https://a1.muscache.com/ac/pictures/67560560/c...
1 https://a0.muscache.com/ac/pictures/14409893/f...

xl_picture_url host_id \
0          NaN      956883
1 https://a0.muscache.com/ac/pictures/14409893/f...  5177328

host_url host_name host_since \
0 https://www.airbnb.com/users/show/956883      Maija  2011-08-11
1 https://www.airbnb.com/users/show/5177328      Andrea 2013-02-21
```

```

host_location \
0 Seattle, Washington, United States
1 Seattle, Washington, United States

host_about host_response_time \
0 I am an artist, interior designer, and run a s... within a few hours
1 Living east coast/left coast/overseas. Time i... within an hour

host_response_rate host_acceptance_rate host_is_superhost \
0 96% 100% f
1 98% 100% t

host_thumbnail_url \
0 https://a0.muscache.com/ac/users/956883/profil...
1 https://a0.muscache.com/ac/users/5177328/profi...

host_picture_url host_neighbourhood \
0 https://a0.muscache.com/ac/users/956883/profil... Queen Anne
1 https://a0.muscache.com/ac/users/5177328/profi... Queen Anne

host_listings_count host_total_listings_count \
0 3.0 3.0
1 6.0 6.0

host_verifications host_has_profile_pic \
0 ['email', 'phone', 'reviews', 'kba'] t
1 ['email', 'phone', 'facebook', 'linkedin', 're... t

host_identity_verified street \
0 t Gilman Dr W, Seattle, WA 98119, United States
1 t 7th Avenue West, Seattle, WA 98119, United States

neighbourhood neighbourhood_cleansed neighbourhood_group_cleansed city \
0 Queen Anne West Queen Anne Queen Anne Seattle
1 Queen Anne West Queen Anne Queen Anne Seattle

state zipcode market smart_location country_code country \
0 WA 98119 Seattle Seattle, WA US United States
1 WA 98119 Seattle Seattle, WA US United States

latitude longitude is_location_exact property_type room_type \
0 47.636289 -122.371025 t Apartment Entire home/apt
1 47.639123 -122.365666 t Apartment Entire home/apt

accommodates bathrooms bedrooms beds bed_type \
0 4 1.0 1.0 1.0 Real Bed
1 4 1.0 1.0 1.0 Real Bed

```

```

                    amenities  square_feet    price  \
0  {TV,"Cable TV",Internet,"Wireless Internet","A...           NaN   $85.00
1  {TV,Internet,"Wireless Internet",Kitchen,"Free...           NaN  $150.00

  weekly_price monthly_price security_deposit cleaning_fee  guests_included  \
0            NaN          NaN          NaN          NaN             2
1  $1,000.00     $3,000.00      $100.00      $40.00             1

  extra_people  minimum_nights  maximum_nights calendar_updated  \
0        $5.00            1            365   4 weeks ago
1        $0.00            2            90       today

  has_availability  availability_30  availability_60  availability_90  \
0              t            14            41            71
1              t            13            13            16

  availability_365 calendar_last_scraped  number_of_reviews first_review  \
0            346        2016-01-04         207  2011-11-01
1            291        2016-01-04         43   2013-08-19

  last_review  review_scores_rating  review_scores_accuracy  \
0  2016-01-02            95.0            10.0
1  2015-12-29            96.0            10.0

  review_scores_cleanliness  review_scores_checkin  \
0                  10.0            10.0
1                  10.0            10.0

  review_scores_communication  review_scores_location  review_scores_value  \
0                  10.0            9.0            10.0
1                  10.0           10.0            10.0

  requires_license  license jurisdiction_names instant_bookable  \
0                  f        NaN      WASHINGTON             f
1                  f        NaN      WASHINGTON             f

  cancellation_policy require_guest_profile_picture  \
0            moderate                 f
1            strict                   t

  require_guest_phone_verification  calculated_host_listings_count  \
0                      f                  2
1                      t                  6

  reviews_per_month
0            4.07

```

```
1           1.48
```

```
[4]: #make a copy with the interested columns
seattle_amenities = seattle_listings[['id','price','neighbourhood_group_cleansed', 'amenities','review_scores_rating',
   'number_of_reviews']] # I included the price and reviews as well
seattle_amenities.head(3)
```

```
[4]:      id    price neighbourhood_group_cleansed \
0   241032   $85.00                  Queen Anne
1   953595   $150.00                 Queen Anne
2   3308979   $975.00                Queen Anne

  amenities  review_scores_rating \
0  {TV,"Cable TV",Internet,"Wireless Internet","A...             95.0
1  {TV,Internet,"Wireless Internet",Kitchen,"Free...             96.0
2  {TV,"Cable TV",Internet,"Wireless Internet","A...             97.0

  number_of_reviews
0              207
1               43
2               20
```

```
[5]: seattle_amenities.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3818 entries, 0 to 3817
Data columns (total 6 columns):
 #   Column            Non-Null Count  Dtype  
--- 
 0   id                3818 non-null    int64  
 1   price              3818 non-null    object 
 2   neighbourhood_group_cleansed  3818 non-null    object 
 3   amenities          3818 non-null    object 
 4   review_scores_rating 3171 non-null    float64
 5   number_of_reviews   3818 non-null    int64  
dtypes: float64(1), int64(2), object(3)
memory usage: 179.1+ KB
```

Here we can see that the `review_scores_rating` column has null values.

### 1.1.2 Boston Dataset Examination

```
[6]: #make a copy with the interested columns
boston_amenities = boston_listings[['id', 'price', 'neighbourhood_cleansed', 'amenities', 'review_scores_rating', 'number_of_reviews']] # I included the price as well
boston_amenities.head(3)
```

```
[6]:      id      price neighbourhood_cleansed \
0  12147973  $250.00          Roslindale
1   3075044    $65.00          Roslindale
2     6976    $65.00          Roslindale

  amenities  review_scores_rating \
0  {TV,"Wireless Internet",Kitchen,"Free Parking ...           NaN
1  {TV,Internet,"Wireless Internet","Air Conditio...         94.0
2  {TV,"Cable TV","Wireless Internet","Air Condit...         98.0

  number_of_reviews
0                  0
1                 36
2                 41
```

```
[7]: boston_amenities.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3585 entries, 0 to 3584
Data columns (total 6 columns):
 #   Column            Non-Null Count  Dtype  
--- 
 0   id                3585 non-null   int64  
 1   price              3585 non-null   object 
 2   neighbourhood_cleansed  3585 non-null   object 
 3   amenities          3585 non-null   object 
 4   review_scores_rating 2772 non-null   float64
 5   number_of_reviews   3585 non-null   int64  
dtypes: float64(1), int64(2), object(3)
memory usage: 168.2+ KB
```

### 1.1.3 Step 1: Addressing And Handling Missing Data

We will first view a dataframe where the `review_scores_rating` has null values then decide what to do from there.

```
[8]: seattle_amenities[(seattle_amenities.review_scores_rating.isnull())]
```

```
[8]:      id      price neighbourhood_group_cleansed \
3      7421966   $100.00           Queen Anne
18     7735464   $200.00           Queen Anne
23     10106055   $75.00           Queen Anne
26     9025039   $150.00           Queen Anne
46     9550869   $349.00           Queen Anne
...
...      ...
3800    5482204   $185.00       Other neighborhoods
3802    8562314   $200.00       Other neighborhoods
3815    10267360   $93.00        Rainier Valley
3816    9604740   $99.00        Capitol Hill
3817    10208623   $87.00        Queen Anne

                                amenities  review_scores_rating \
3      {Internet,"Wireless Internet",Kitchen,"Indoor ...           NaN
18     {TV,"Cable TV",Internet,"Wireless Internet",Ki...           NaN
23     {Internet,"Wireless Internet","Air Conditionin...           NaN
26     {"Cable TV","Wireless Internet",Kitchen,"Free ...           NaN
46     {Internet,"Wireless Internet",Kitchen,"Free Pa...           NaN
...
...      ...
3800    {TV,"Cable TV",Internet,"Wireless Internet","A...           NaN
3802    {TV,"Wireless Internet",Kitchen,"Pets Allowed"...           NaN
3815    {"Cable TV","Wireless Internet",Kitchen,"Free ...           NaN
3816    {TV,"Wireless Internet",Kitchen,"Free Parking ...           NaN
3817    {TV,"Cable TV",Internet,"Wireless Internet",Ki...           NaN

      number_of_reviews
3                  0
18                 0
23                 0
26                 0
46                 0
...
...      ...
3800                0
3802                0
3815                0
3816                0
3817                0

[647 rows x 6 columns]
```

This is interesting. There are quite a number of rows with 0 reviews. Let's see if there are any with at least 1 review.

```
[9]: seattle_amenities[(seattle_amenities.review_scores_rating.isnull()) &
                     (seattle_amenities.number_of_reviews) > 0]
```

[9]:

|      |         | id       | price | neighbourhood_group_cleansed | amenities | review_scores_rating | number_of_reviews |
|------|---------|----------|-------|------------------------------|-----------|----------------------|-------------------|
| 204  | 9028447 | \$100.00 |       | Queen Anne                   |           | NaN                  | 1                 |
| 283  | 5336585 | \$75.00  |       | Other neighborhoods          |           | NaN                  | 1                 |
| 361  | 8207848 | \$259.00 |       | Other neighborhoods          |           | NaN                  |                   |
| 482  | 186328  | \$250.00 |       | Queen Anne                   |           | NaN                  |                   |
| 527  | 790860  | \$450.00 |       | Other neighborhoods          |           | NaN                  |                   |
| 774  | 7536074 | \$185.00 |       | University District          |           | NaN                  |                   |
| 804  | 7807658 | \$118.00 |       | University District          |           | NaN                  |                   |
| 1024 | 8103432 | \$140.00 |       | Central Area                 |           | NaN                  |                   |
| 1457 | 9012948 | \$130.00 |       | Downtown                     |           | NaN                  |                   |
| 1702 | 8338155 | \$65.00  |       | Downtown                     |           | NaN                  |                   |
| 1766 | 4887845 | \$235.00 |       | West Seattle                 |           | NaN                  |                   |
| 1863 | 692080  | \$370.00 |       | West Seattle                 |           | NaN                  |                   |
| 2493 | 3126153 | \$115.00 |       | Other neighborhoods          |           | NaN                  |                   |
| 2870 | 7614244 | \$105.00 |       | Capitol Hill                 |           | NaN                  |                   |
| 2946 | 7874053 | \$250.00 |       | Capitol Hill                 |           | NaN                  |                   |
| 3307 | 7969698 | \$97.00  |       | Northgate                    |           | NaN                  |                   |
| 3671 | 6392029 | \$136.00 |       | Other neighborhoods          |           | NaN                  |                   |
| 3709 | 8080939 | \$200.00 |       | Other neighborhoods          |           | NaN                  |                   |
| 3722 | 5902452 | \$150.00 |       | Other neighborhoods          |           | NaN                  |                   |
| 3766 | 6043482 | \$50.00  |       | Other neighborhoods          |           | NaN                  |                   |

```

361          1
482          1
527          1
774          1
804          1
1024         1
1457         1
1702         1
1766         1
1863         1
2493         1
2870         1
2946         1
3307         1
3671         1
3709         1
3722         1
3766         1

```

Solutions: > 1. For those where the number of reviews are greater than 0, it makes sense to drop these rows because we have no way of figuring out what the actual rating is. 2. For those where the number of reviews is 0, we will keep and impute the null values with 0. 3. We will apply this to both cities' datasets.

```
[10]: # get inverted version
seattle_amenities = seattle_amenities.loc[~(((seattle_amenities.
    ↪review_scores_rating.isnull()) &
    (seattle_amenities.number_of_reviews) > 0))]
# replace all remaining null value in review_scores_rating with 0
seattle_amenities.fillna(value = 0, inplace = True)
```

```
[11]: # Check by using info()
seattle_amenities.info()
```

```

<class 'pandas.core.frame.DataFrame'>
Int64Index: 3798 entries, 0 to 3817
Data columns (total 6 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   id               3798 non-null   int64  
 1   price            3798 non-null   object  
 2   neighbourhood_group_cleansed 3798 non-null   object  
 3   amenities        3798 non-null   object  
 4   review_scores_rating 3798 non-null   float64 
 5   number_of_reviews 3798 non-null   int64  
dtypes: float64(1), int64(2), object(3)
memory usage: 207.7+ KB

```

```
[12]: boston_amenities = boston_amenities.loc[~(((boston_amenities.
    ↪review_scores_rating.isnull()) &
        (boston_amenities.number_of_reviews) > 0))]
# replace all remaining null value in review_scores_rating with 0
boston_amenities.fillna(value = 0, inplace = True)
```

```
[13]: # Check by using info()
boston_amenities.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3534 entries, 0 to 3584
Data columns (total 6 columns):
 #   Column           Non-Null Count  Dtype  
---  --  
 0   id               3534 non-null   int64  
 1   price             3534 non-null   object  
 2   neighbourhood_cleansed  3534 non-null   object  
 3   amenities          3534 non-null   object  
 4   review_scores_rating 3534 non-null   float64 
 5   number_of_reviews   3534 non-null   int64  
dtypes: float64(1), int64(2), object(3)
memory usage: 193.3+ KB
```

#### 1.1.4 Step 2: Cleaning Data

```
[14]: # view the 1st value of the amenities column
seattle_amenities.amenities[0]
```

```
[14]: '{TV,"Cable TV",Internet,"Wireless Internet","Air
Conditioning",Kitchen,Heating,"Family/Kid Friendly",Washer,Dryer}'
```

```
[15]: # split() help: https://www.w3schools.com/python/ref_string_split.asp
clean_seattle_amenities = (seattle_amenities.amenities.str.replace('{','').str.
    ↪replace('}','')
    .str.replace('\"','').str.get_dummies(sep=','))
```

```
[16]: seattle_amenities = pd.concat([seattle_amenities, clean_seattle_amenities], ↴
    ↪axis=1) #combine the datasets into one
seattle_amenities.drop(columns = ['amenities'], inplace = True)
seattle_amenities.head(2)
```

```
[16]:      id      price neighbourhood_group_cleansed  review_scores_rating \
0  241032    $85.00                  Queen Anne            95.0
1  953595   $150.00                  Queen Anne            96.0

      number_of_reviews  24-Hour Check-in  Air Conditioning  Breakfast \
0                      207                 0                  1                  0
```

|   |                            |                     |                          |                      |                           |         |   |
|---|----------------------------|---------------------|--------------------------|----------------------|---------------------------|---------|---|
| 1 | 43                         | 0                   | 0                        | 0                    |                           |         |   |
| 0 | Buzzer/Wireless Intercom   | Cable TV            | Carbon Monoxide Detector | Cat(s)               | \                         |         |   |
| 1 | 0                          | 1                   | 0                        | 0                    | 0                         |         |   |
| 0 | Dog(s)                     | Doorman             | Dryer                    | Elevator in Building | Essentials                | \       |   |
| 1 | 0                          | 0                   | 1                        | 0                    | 0                         | 1       |   |
| 0 | Family/Kid Friendly        | Fire Extinguisher   | First Aid Kit            |                      | \                         |         |   |
| 1 | 1                          | 0                   | 1                        | 1                    |                           |         |   |
| 0 | Free Parking on Premises   | Gym                 | Hair Dryer               | Hangers              | Heating                   | Hot Tub | \ |
| 1 | 0                          | 0                   | 0                        | 0                    | 1                         | 0       |   |
| 0 | 1                          | 0                   | 0                        | 0                    | 1                         | 0       |   |
| 0 | Indoor Fireplace           | Internet            | Iron                     | Kitchen              | Laptop Friendly Workspace | \       |   |
| 1 | 0                          | 1                   | 0                        | 1                    |                           | 0       |   |
| 0 | 0                          | 1                   | 0                        | 1                    |                           | 0       |   |
| 0 | Lock on Bedroom Door       | Other pet(s)        | Pets Allowed             |                      | \                         |         |   |
| 1 | 0                          | 0                   | 0                        | 0                    |                           |         |   |
| 0 | Pets live on this property | Pool                | Safety Card              | Shampoo              | Smoke Detector            | \       |   |
| 1 | 0                          | 0                   | 0                        | 1                    | 0                         | 1       |   |
| 0 | 0                          | 0                   | 0                        | 0                    | 0                         | 0       |   |
| 0 | Smoking Allowed            | Suitable for Events | TV                       | Washer               | Washer / Dryer            | \       |   |
| 1 | 0                          | 0                   | 1                        | 1                    |                           | 0       |   |
| 0 | 0                          | 0                   | 1                        | 1                    |                           | 0       |   |
| 0 | Wheelchair Accessible      | Wireless Internet   |                          |                      |                           |         |   |
| 1 | 0                          | 1                   |                          |                      |                           |         |   |

```
[17]: # this time we will do the same to the Boston dataset
clean_boston_amenities = (boston_amenities.amenities.str.replace('{','').str.
                           replace('}',''))
                           .str.replace('"','').str.get_dummies(sep=','))
boston_amenities = pd.concat([boston_amenities, clean_boston_amenities], axis=1)
boston_amenities.drop(columns = ['amenities'], inplace = True)
boston_amenities.head(2)
```

```
[17]:      id      price neighbourhood_cleansed  review_scores_rating \
0  12147973  $250.00                  Roslindale          0.0
1  3075044   $65.00                  Roslindale         94.0

      number_of_reviews  24-Hour Check-in  Air Conditioning  Breakfast \
0                      0                 0                  0            0
1                     36                 0                  1            0

      Buzzer/Wireless Intercom  Cable TV  Carbon Monoxide Detector  Cat(s) \
0                      0                 0                  0            0
1                      0                 0                  1            0

      Dog(s)  Doorman  Dryer  Elevator in Building  Essentials \
0          1        0      1                  0            1
1          1        0      1                  0            1

      Family/Kid Friendly  Fire Extinguisher  First Aid Kit \
0                  1                  1            0
1                  1                  1            0

      Free Parking on Premises  Free Parking on Street  Gym  Hair Dryer  Hangers \
0                      1                  0            0            0            0
1                      0                  0            0            1            1

      Heating  Hot Tub  Indoor Fireplace  Internet  Iron  Kitchen \
0          1        0                  0            0            0            1
1          1        0                  0            1            1            1

      Laptop Friendly Workspace  Lock on Bedroom Door  Other pet(s) \
0                  1                  0            0
1                  0                  1            0

      Paid Parking Off Premises  Pets Allowed  Pets live on this property  Pool \
0                      0                  0            1            0
1                      0                  1            1            0

      Safety Card  Shampoo  Smoke Detector  Smoking Allowed  Suitable for Events \
0          0        1                  1            0            0
1          0        1                  1            0            0

      TV  Washer  Washer / Dryer  Wheelchair Accessible  Wireless Internet \
0          1        1                  0            0            1
1          1        1                  0            0            1

      translation missing: en.hosting_amenity_49 \
0                      0
1                      0
```

```

translation missing: en.hosting_amenity_50
0
1

```

```
[18]: # personal reminder - iloc = [m,n]
seattle_offerings = seattle_amenities.iloc[:,5:].sum().sort_values(ascending = False).reset_index() # get sum of each amenity and sort them
seattle_offerings.columns = ['Amenity','Total # Of Listings']
# find out the % of offerings in each city that offers said amenity
seattle_offerings['% Of Houses'] = (seattle_offerings['Total # Of Listings']/seattle_amenities.shape[0])*100

seattle_offerings.head()
```

|   | Amenity           | Total # Of Listings | % Of Houses |
|---|-------------------|---------------------|-------------|
| 0 | Wireless Internet | 3650                | 96.103212   |
| 1 | Heating           | 3612                | 95.102686   |
| 2 | Kitchen           | 3405                | 89.652449   |
| 3 | Smoke Detector    | 3267                | 86.018957   |
| 4 | Essentials        | 3224                | 84.886783   |

```
[19]: # do it again with Boston dataset
boston_offerings = boston_amenities.iloc[:,5:].sum().sort_values(ascending = False).reset_index() # get sum of each amenity and sort them
boston_offerings.columns = ['Amenity','Total # Of Listings']
boston_offerings['% Of Houses'] = (boston_offerings['Total # Of Listings']/boston_amenities.shape[0])*100

boston_offerings.head()
```

|   | Amenity           | Total # Of Listings | % Of Houses |
|---|-------------------|---------------------|-------------|
| 0 | Wireless Internet | 3376                | 95.529145   |
| 1 | Heating           | 3335                | 94.368987   |
| 2 | Kitchen           | 3236                | 91.567629   |
| 3 | Essentials        | 2952                | 83.531409   |
| 4 | Smoke Detector    | 2876                | 81.380872   |

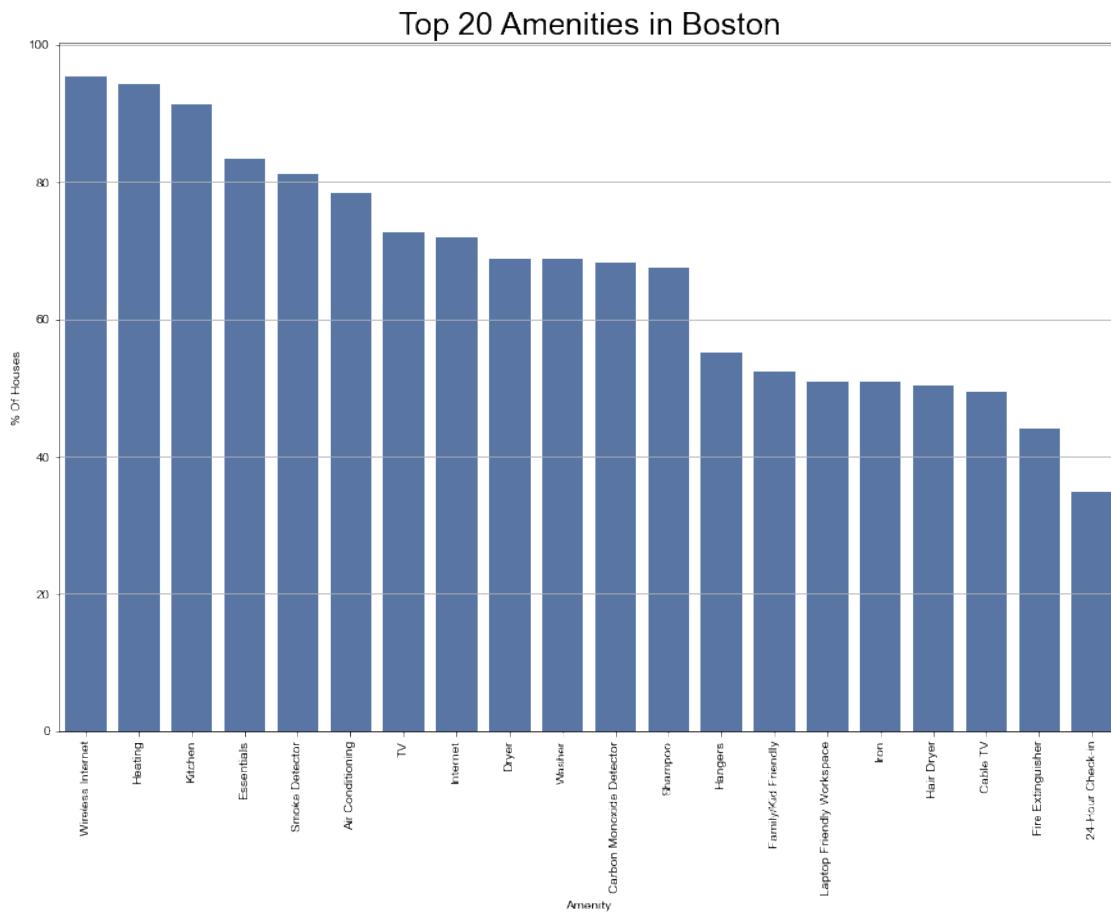
### 1.1.5 Step 3: Visualizing Data

Note: When I plotted all of the amenities, the resulting bar chart looked very cluttered and noisy. Therefore, I will only be focusing on the top 20 amenities. Furthermore, if I include all of them, I will be answering the question. The list must be narrowed down.

#### Bar Chart Visuals: Boston Top 20 Amenities

```
[20]: plt.rcParams['figure.figsize'] = [15, 10]
plt.grid(b=None)

sb.set_theme(style="whitegrid")
base_color = sb.color_palette()[0]
sb.barplot(x="Amenity", y="% Of Houses", data=boston_offerings[:20], color = base_color)
plt.xticks(rotation = 90)
plt.title("Top 20 Amenities in Boston", fontsize=25)
plt.savefig('Barplot:Top 20 Amenities in Boston.png');
```



### Bar Chart Visuals: Seattle Top 20 Amenities

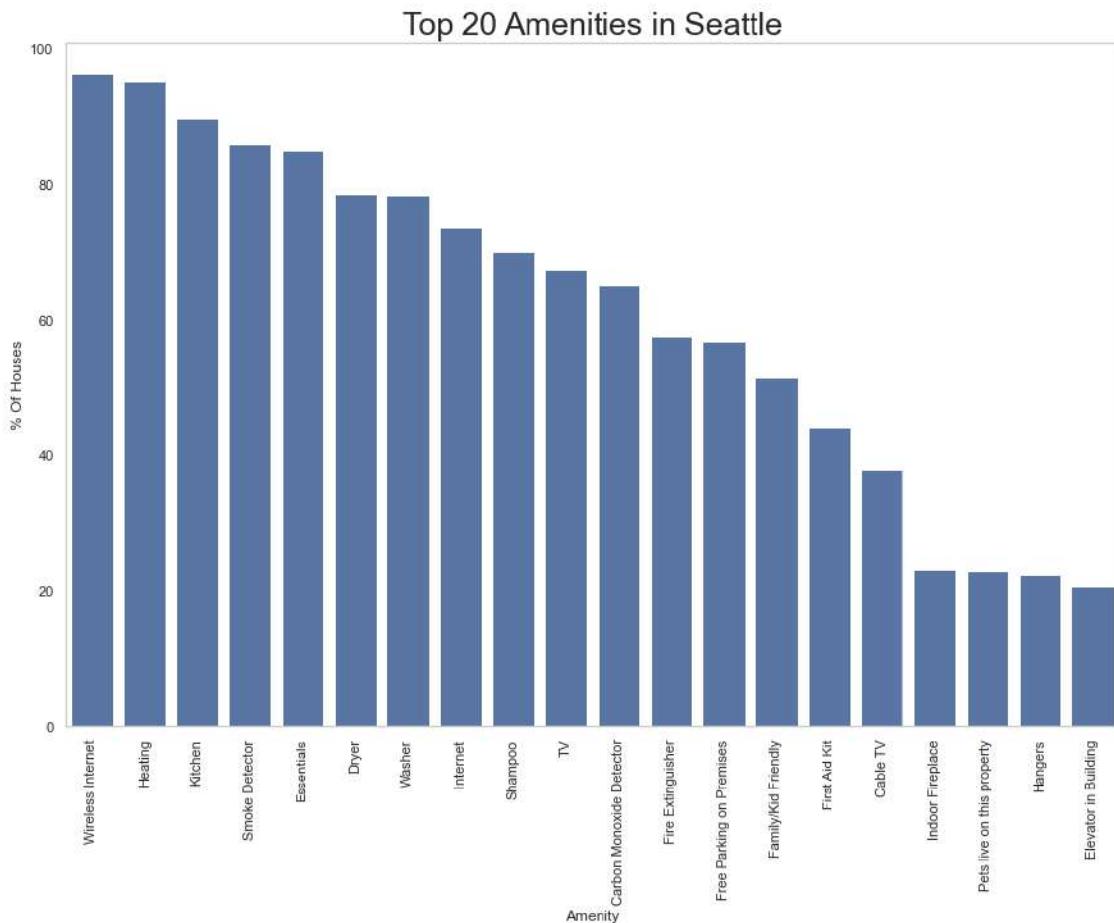
```
[21]: plt.rcParams['figure.figsize'] = [15, 10]
base_color = sb.color_palette()[0]
plt.grid(b=None)

sb.set_theme(style="whitegrid")
```

```

sb.barplot(x="Amenity", y="% Of Houses", data=seattle_offerings[:20], color = base_color)
plt.xticks(rotation = 90)
plt.title("Top 20 Amenities in Seattle", fontsize=25)
plt.savefig('Barplot:Top 20 Amenities in Seattle.png');

```



Here we can see that the 2 cities share many of the same amenities. One of the only difference I can find is that Boston has more properties with elevators. However, it since the top 20 are a little different I will need to create 2 bar charts to fully compare the top amenities in each cities. So, I will start off with Boston first.

### Bar Plot Visualization #1 (ORDERED BY % Of Houses\_Boston )

```

[22]: # merging help : https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.merge.html
all_amenities = boston_offerings.merge(seattle_offerings, on = 'Amenity', suffixes = ('_Boston', '_Seattle'))
all_amenities = all_amenities.sort_values(by = '% Of Houses_Boston', ascending=False)

```

```
all_amenities.head()

[22]:
```

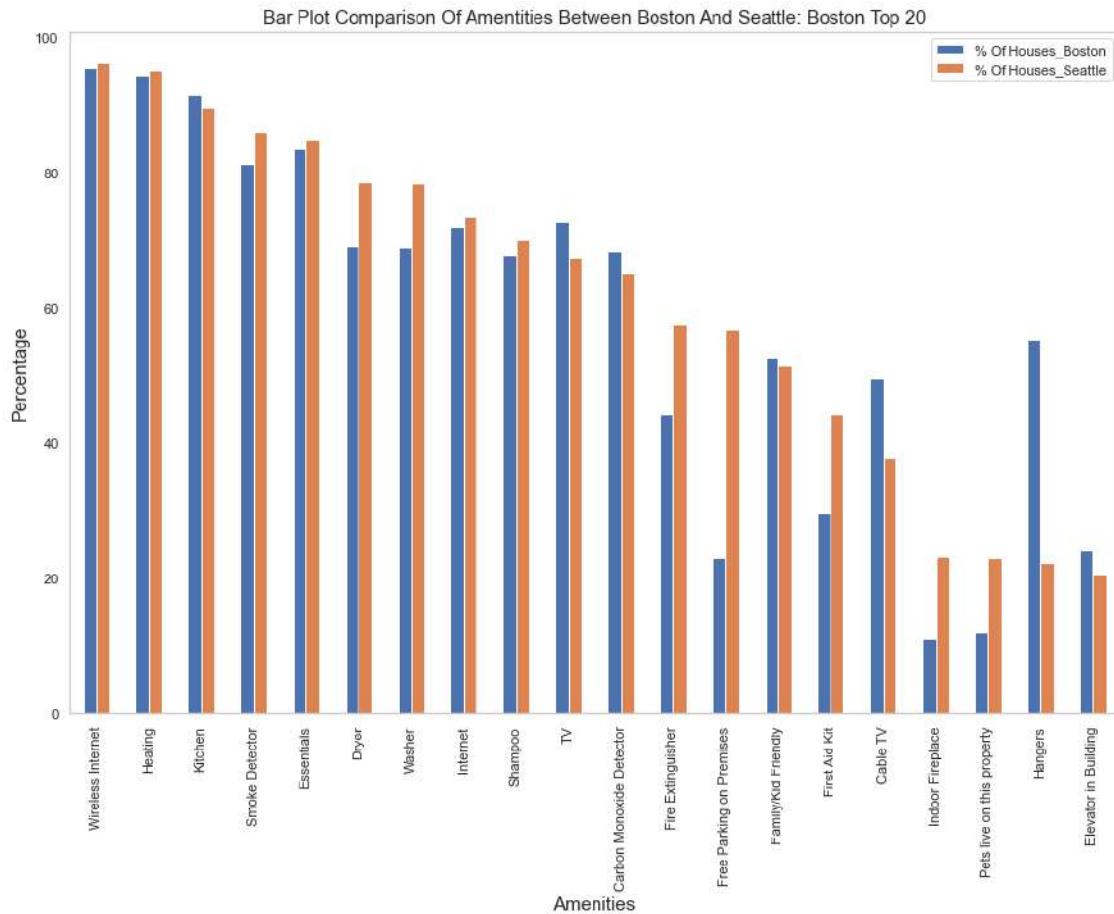
|   | Amenity           | Total # Of Listings_Boston | % Of Houses_Boston | \ |
|---|-------------------|----------------------------|--------------------|---|
| 0 | Wireless Internet | 3376                       | 95.529145          |   |
| 1 | Heating           | 3335                       | 94.368987          |   |
| 2 | Kitchen           | 3236                       | 91.567629          |   |
| 3 | Essentials        | 2952                       | 83.531409          |   |
| 4 | Smoke Detector    | 2876                       | 81.380872          |   |

|   | Total # Of Listings_Seattle | % Of Houses_Seattle |
|---|-----------------------------|---------------------|
| 0 | 3650                        | 96.103212           |
| 1 | 3612                        | 95.102686           |
| 2 | 3405                        | 89.652449           |
| 3 | 3224                        | 84.886783           |
| 4 | 3267                        | 86.018957           |

```
[27]: plt.rcParams['figure.figsize'] = [15, 10]

# plot top 20 only: first we will compare the list with Seattle's top 20
all_amenities.head(20).plot(x = 'Amenity', y = ['% Of Houses_Boston', '% Of Houses_Seattle'],
                             kind = 'bar', grid = False)
plt.title('Bar Plot Comparison Of Amentities Between Boston And Seattle: Boston Top 20', fontsize=15)
plt.ylabel('Percentage', fontsize=15)
plt.xlabel('Amenities', fontsize=15)
plt.savefig('Barplot:Comparison Of Amentities Between Boston And Seattle: Boston Top 20.png',
            bbox_inches='tight');
```



### Bar Plot Visualization #2 (ORDERED BY % Of Houses\_Seattle )

```
[24]: # we must repeat the above step because the first comparison was ordered by
      ↵Boston
all_amenities = seattle_offerings.merge(boston_offerings, on = 'Amenity',
      ↵suffixes = ('_Seattle', '_Boston'))
all_amenities = all_amenities.sort_values(by = '% Of Houses_Seattle', ascending
      ↵= False)
all_amenities.head()
```

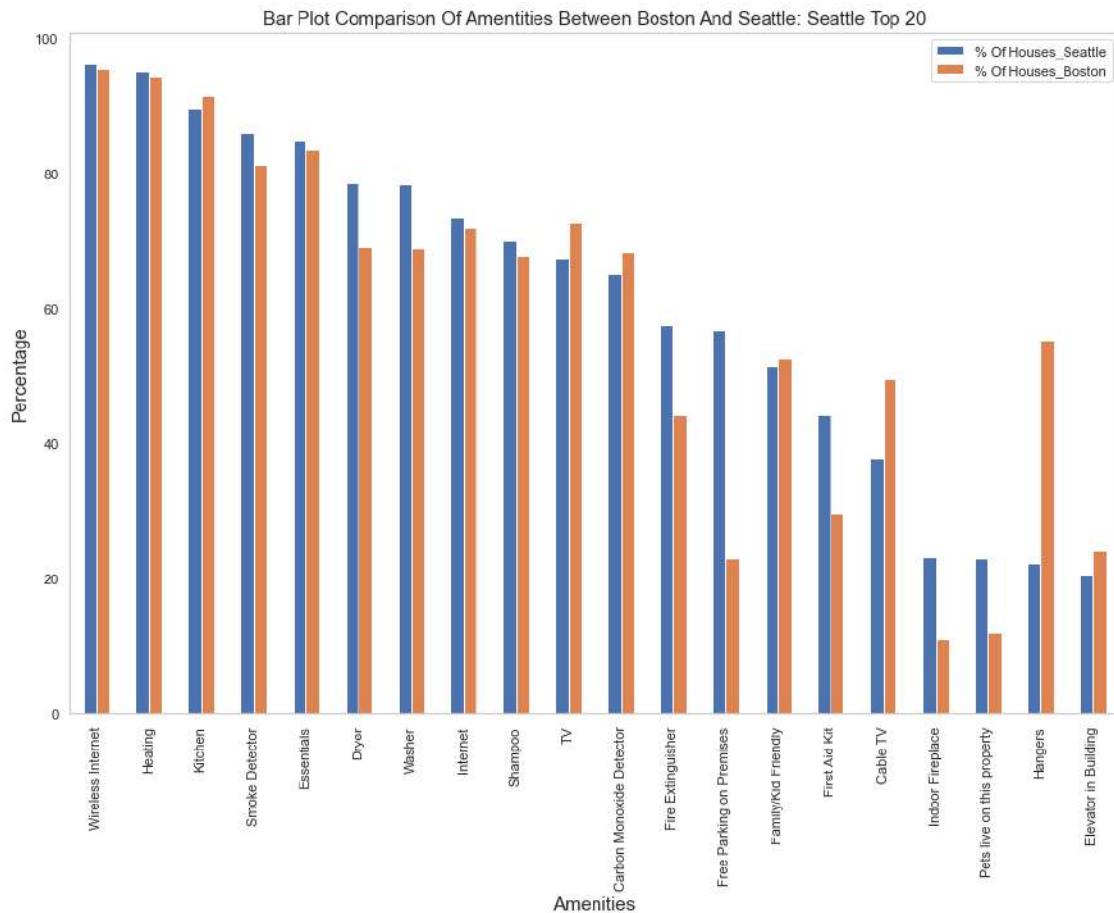
|   | Amenity           | Total # Of Listings_Seattle | % Of Houses_Seattle |
|---|-------------------|-----------------------------|---------------------|
| 0 | Wireless Internet | 3650                        | 96.103212           |
| 1 | Heating           | 3612                        | 95.102686           |
| 2 | Kitchen           | 3405                        | 89.652449           |
| 3 | Smoke Detector    | 3267                        | 86.018957           |
| 4 | Essentials        | 3224                        | 84.886783           |

Total # Of Listings\_Boston % Of Houses\_Boston

|   |      |           |
|---|------|-----------|
| 0 | 3376 | 95.529145 |
| 1 | 3335 | 94.368987 |
| 2 | 3236 | 91.567629 |
| 3 | 2876 | 81.380872 |
| 4 | 2952 | 83.531409 |

```
[28]: plt.rcParams['figure.figsize'] = [15, 10]
```

```
# plot top 20 only: first we will compare the list with Seattle's top 20
all_amenities.head(20).plot(x = 'Amenity', y = ['% Of Houses_Seattle', '% Of Houses_Boston'],
                           kind = 'bar', grid = False)
plt.title('Bar Plot Comparison Of Amenties Between Boston And Seattle: Seattle Top 20', fontsize=15)
plt.ylabel('Percentage', fontsize=15)
plt.xlabel('Amenities', fontsize=15)
plt.savefig('Barplot:Comparison Of Amenties Between Boston And Seattle: Seattle Top 20.png',
            bbox_inches='tight');
```



Takeaways: > 1. As started earlier, it seems that the majority of the top 20 amenities in both cities are very similar. In Boston, we can see that ‘elevator in building’ cracks the top 20 while in Seattle it doesn’t. 2. One interesting thing that I see is that Boston’s properties have quite a lot of ‘Laptop friendly Workplace’. 3. It seems like the top 3 amenities in both cities are the same: Wireless Internet, Heating and Kitchen. 4. Seattle has almost a 1/3 of Boston’s total for the ‘air conditioning column’ (Living in California, I can understand why). The city also has far more properties with first-aid kits & free parking on premise.

|      |  |
|------|--|
| [ ]: |  |
| [ ]: |  |
| [ ]: |  |
| [ ]: |  |
| [ ]: |  |
| [ ]: |  |
| [ ]: |  |
| [ ]: |  |

# airbnb-question-2-part-iii

February 24, 2024

## 1 Boston vs. Seattle Airbnb III

In this notebook(Part 3), I will be answering the question: Is it possible to predict the price with 5 features? If yes, compare the 2 cities.

```
[1]: import warnings
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.model_selection import train_test_split
%matplotlib inline
warnings.filterwarnings('ignore')

pd.set_option('display.max_columns', None)
```

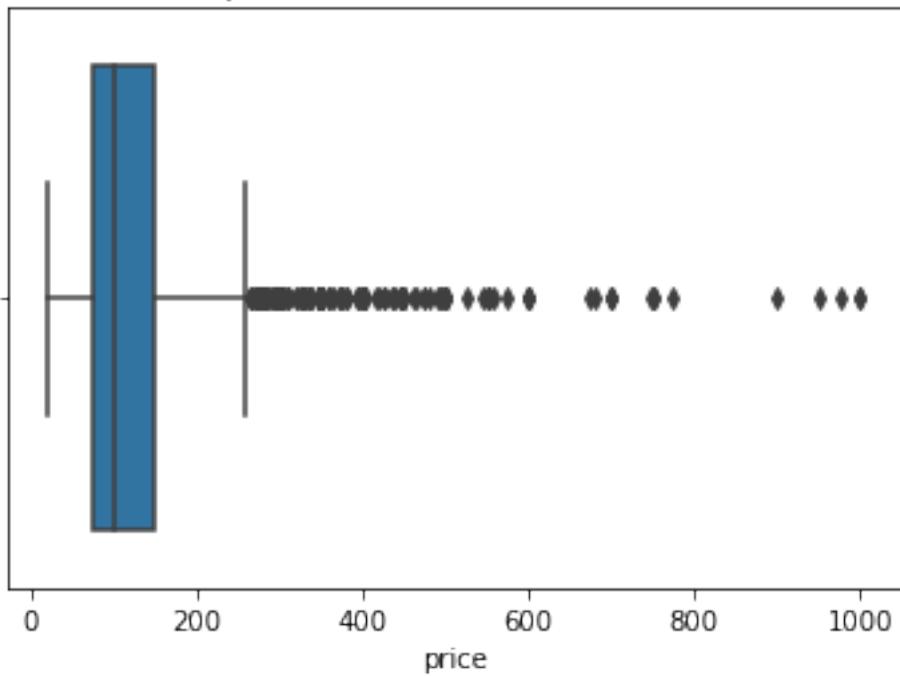
1.1 Question 2: Is it possible to predict the price with 8 features? If yes, compare the 2 cities.

### 1.1.1 Step 1: Examining and Cleaning Data

```
[2]: # import both datasets
seattle_listings = pd.read_csv('seattle_airbnb/listings.csv')
boston_listings = pd.read_csv('boston_airbnb/listings.csv')
```

```
[3]: #cleaning up the price
seattle_listings['price'] = seattle_listings['price'].map(lambda x: int(x[1:-3].replace(',', '')))
# similair to what we did with neighbourhood, this time we will examine the overall city
sb.boxplot(x = 'price', data = seattle_listings)
plt.title("Boxplot Distribution Of Seattle Price");
```

Boxplot Distribution Of Seattle Price



As we can see, there are quite alot of outliers. I will need to address these outliers which is what I'm going to do next. I will be using help from the link down below. > Link: <https://towardsdatascience.com/ways-to-detect-and-remove-the-outliers-404d16608dba>

```
[4]: # addressing and dropping the outliers using (1.5*IQR + Q3)
Q1 = seattle_listings['price'].quantile(0.25)
Q3 = seattle_listings['price'].quantile(0.75)
IQR = Q3 - Q1
upper = 1.5*(IQR) + Q3

upper
```

[4]: 262.5

```
[5]: seattle_listings = seattle_listings[seattle_listings['price'] <= upper]
seattle_listings.head(3)
```

```
[5]:      id          listing_url      scrape_id last_scraped \
0    241032  https://www.airbnb.com/rooms/241032  20160104002432  2016-01-04
1    953595  https://www.airbnb.com/rooms/953595  20160104002432  2016-01-04
3   7421966  https://www.airbnb.com/rooms/7421966  20160104002432  2016-01-04
```

name \

```
0      Stylish Queen Anne Apartment
1  Bright & Airy Queen Anne Apartment
3          Queen Anne Chateau

summary \
0           NaN
1 Chemically sensitive? We've removed the irrita...
3 A charming apartment that sits atop Queen Anne...

space \
0 Make your self at home in this charming one-be...
1 Beautiful, hypoallergenic apartment in an extr...
3           NaN

description experiences_offered \
0 Make your self at home in this charming one-be...      none
1 Chemically sensitive? We've removed the irrita...      none
3 A charming apartment that sits atop Queen Anne...      none

neighborhood_overview \
0           NaN
1 Queen Anne is a wonderful, truly functional vi...
3           NaN

notes \
0           NaN
1 What's up with the free pillows? Our home was...
3           NaN

transit \
0           NaN
1 Convenient bus stops are just down the block, ...
3           NaN

thumbnail_url \
0           NaN
1 https://a0.muscache.com/ac/pictures/14409893/f...
3           NaN

medium_url \
0           NaN
1 https://a0.muscache.com/im/pictures/14409893/f...
3           NaN

picture_url \
0 https://a1.muscache.com/ac/pictures/67560560/c...
1 https://a0.muscache.com/ac/pictures/14409893/f...
```

```

3 https://a0.muscache.com/ac/pictures/94146944/6...
          xl_picture_url host_id \
0                      NaN 956883
1 https://a0.muscache.com/ac/pictures/14409893/f... 5177328
3                                     NaN 9851441

          host_url host_name host_since \
0 https://www.airbnb.com/users/show/956883 Maija 2011-08-11
1 https://www.airbnb.com/users/show/5177328 Andrea 2013-02-21
3 https://www.airbnb.com/users/show/9851441 Emily 2013-11-06

          host_location \
0 Seattle, Washington, United States
1 Seattle, Washington, United States
3 Seattle, Washington, United States

          host_about host_response_time \
0 I am an artist, interior designer, and run a s... within a few hours
1 Living east coast/left coast/overseas. Time i... within an hour
3                                     NaN           NaN

host_response_rate host_acceptance_rate host_is_superhost \
0             96%           100%           f
1             98%           100%           t
3             NaN            NaN           f

          host_thumbnail_url \
0 https://a0.muscache.com/ac/users/956883/profil...
1 https://a0.muscache.com/ac/users/5177328/profi...
3 https://a2.muscache.com/ac/users/9851441/profi...

          host_picture_url host_neighbourhood \
0 https://a0.muscache.com/ac/users/956883/profil... Queen Anne
1 https://a0.muscache.com/ac/users/5177328/profi... Queen Anne
3 https://a2.muscache.com/ac/users/9851441/profi... Queen Anne

host_listings_count host_total_listings_count \
0                 3.0           3.0
1                 6.0           6.0
3                 1.0           1.0

          host_verifications host_has_profile_pic \
0 ['email', 'phone', 'reviews', 'kba']           t
1 ['email', 'phone', 'facebook', 'linkedin', 're...   t
3 ['email', 'phone', 'facebook', 'reviews', 'jum...   t

```

```

host_identity_verified   street \
0                      t      Gilman Dr W, Seattle, WA 98119, United States
1                      t    7th Avenue West, Seattle, WA 98119, United States
3                      t    8th Avenue West, Seattle, WA 98119, United States

neighbourhood neighbourhood_cleansed neighbourhood_group_cleansed     city \
0   Queen Anne          West Queen Anne                  Queen Anne  Seattle
1   Queen Anne          West Queen Anne                  Queen Anne  Seattle
3   Queen Anne          West Queen Anne                  Queen Anne  Seattle

state zipcode  market smart_location country_code      country \
0   WA    98119  Seattle    Seattle, WA           US  United States
1   WA    98119  Seattle    Seattle, WA           US  United States
3   WA    98119  Seattle    Seattle, WA           US  United States

latitude longitude is_location_exact property_type      room_type \
0  47.636289 -122.371025                     t    Apartment  Entire home/apt
1  47.639123 -122.365666                     t    Apartment  Entire home/apt
3  47.638473 -122.369279                     t    Apartment  Entire home/apt

accommodates  bathrooms  bedrooms  beds  bed_type \
0            4          1.0       1.0    1.0  Real Bed
1            4          1.0       1.0    1.0  Real Bed
3            3          1.0       0.0    2.0  Real Bed

amenities  square_feet  price \
0  {TV,"Cable TV",Internet,"Wireless Internet","A...      NaN    85
1  {TV,Internet,"Wireless Internet",Kitchen,"Free...      NaN   150
3  {Internet,"Wireless Internet",Kitchen,"Indoor ...      NaN   100

weekly_price monthly_price security_deposit cleaning_fee  guests_included \
0        NaN          NaN          NaN          NaN             2
1  $1,000.00    $3,000.00    $100.00    $40.00            1
3   $650.00    $2,300.00          NaN          NaN            1

extra_people  minimum_nights  maximum_nights calendar_updated \
0        $5.00            1            365   4 weeks ago
1        $0.00            2              90      today
3        $0.00            1            1125   6 months ago

has_availability  availability_30  availability_60  availability_90 \
0                  t            14            41            71
1                  t            13            13            16
3                  t              0              0              0

availability_365 calendar_last_scraped  number_of_reviews first_review \
0            346        2016-01-04         207    2011-11-01

```

```

1           291          2016-01-04          43   2013-08-19
3           143          2016-01-04          0      NaN

last_review  review_scores_rating  review_scores_accuracy \
0  2016-01-02             95.0            10.0
1  2015-12-29             96.0            10.0
3      NaN                  NaN            NaN

review_scores_cleanliness  review_scores_checkin \
0                   10.0            10.0
1                   10.0            10.0
3                   NaN            NaN

review_scores_communication  review_scores_location  review_scores_value \
0                   10.0            9.0            10.0
1                   10.0            10.0           10.0
3                   NaN            NaN            NaN

requires_license  license jurisdiction_names instant_bookable \
0                 f        NaN    WASHINGTON          f
1                 f        NaN    WASHINGTON          f
3                 f        NaN    WASHINGTON          f

cancellation_policy require_guest_profile_picture \
0       moderate                f
1        strict                 t
3     flexible                f

require_guest_phone_verification  calculated_host_listings_count \
0                     f                  2
1                     t                  6
3                     f                  1

reviews_per_month
0           4.07
1           1.48
3          NaN

```

Now, it's time to determine the features that I am interested in. I will then divide them into independent variables X and dependent variable y.

Interested Variables/Features For Seattle Dataset:

- > 1. host\_response\_rate
- host\_response\_time
- 3. bedrooms
- 4. review\_scores\_rating
- 5. number\_of\_reviews
- 6. neighbourhood\_group\_cleansed
- 7. property\_type
- 8. bathrooms

```
[6]: X = seattle_listings[['neighbourhood_group_cleansed', 'host_response_time', 'host_response_rate',
```

```
'bedrooms', 'bathrooms', 'property_type',  
↳'number_of_reviews','review_scores_rating']]  
y = seattle_listings['price']
```

```
[7]: # cleaning up the columns' names  
X = X.rename(columns = lambda x: str(x.replace('_', ' ')).strip().title())
```

## Splitting dataset

```
[8]: # random_state help: https://scikit-learn.org/stable/glossary.  
↳html#term-random_state  
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = .3,  
↳random_state= 42)
```

### 1.1.2 Step 2: Addressing And Handling Missing Data

```
[9]: # Checking for nulls  
X_train.isnull().any()
```

```
[9]: Neighbourhood Group Cleansed      False  
Host Response Time                  True  
Host Response Rate                 True  
Bedrooms                           True  
Bathrooms                          True  
Property Type                      True  
Number Of Reviews                  False  
Review Scores Rating               True  
dtype: bool
```

```
[10]: # after examination, it doesn't seem like there is any need to drop any of the  
↳columns  
X_train.isnull().sum()/X_train.shape[0]
```

```
[10]: Neighbourhood Group Cleansed      0.0000  
Host Response Time                  0.1272  
Host Response Rate                 0.1272  
Bedrooms                           0.0020  
Bathrooms                          0.0052  
Property Type                      0.0004  
Number Of Reviews                  0.0000  
Review Scores Rating               0.1688  
dtype: float64
```

```
[11]: X_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
Int64Index: 2500 entries, 922 to 3389
```

```
Data columns (total 8 columns):
 #  Column                Non-Null Count Dtype  
 --- 
 0  Neighbourhood Group Cleansed    2500 non-null   object  
 1  Host Response Time             2182 non-null   object  
 2  Host Response Rate            2182 non-null   object  
 3  Bedrooms                      2495 non-null   float64 
 4  Bathrooms                     2487 non-null   float64 
 5  Property Type                 2499 non-null   object  
 6  Number Of Reviews             2500 non-null   int64  
 7  Review Scores Rating         2078 non-null   float64 

dtypes: float64(3), int64(1), object(4)
memory usage: 175.8+ KB
```

```
[12]: # Viewing the head to make imputation decisions
# For numeric variables, I will be replacing them with the mean
# For categorical variables, it makes more sense to replace them with the mode
X_train.head()
```

```
[12]:      Neighbourhood Group Cleansed  Host Response Time Host Response Rate \
922          Central Area           within a day        100%
1671         Downtown            within an hour       100%
3691  Other neighborhoods     within a few hours      100%
1878        West Seattle          within an hour       100%
2088        Beacon Hill          within a day        100%\\

      Bedrooms  Bathrooms Property Type  Number Of Reviews \
922      1.0      1.0      House          10
1671      1.0      1.0     Apartment         3
3691      1.0      1.0      House          58
1878      1.0      1.0    Townhouse         27
2088      2.0      1.0     Apartment         99\\

      Review Scores Rating
922              96.0
1671             100.0
3691              94.0
1878              96.0
2088              97.0
```

```
[13]: # replace categorical nulls with mode in X_train and X_test
X_train['Host Response Time'] = X_train.loc[:, 'Host Response Time'].
    ↪fillna(X_train['Host Response Time'].mode()[0])
X_train['Host Response Rate'] = X_train.loc[:, 'Host Response Rate'].
    ↪fillna(X_train['Host Response Rate'].mode()[0])
X_train['Property Type'] = X_train.loc[:, 'Property Type'].
    ↪fillna(X_train['Property Type'].mode()[0])
```

```

X_test['Host Response Time'] = X_test.loc[:, 'Host Response Time'].
    ↪fillna(X_test['Host Response Time'].mode()[0])
X_test['Host Response Rate'] = X_test.loc[:, 'Host Response Rate'].
    ↪fillna(X_test['Host Response Rate'].mode()[0])
X_test['Property Type'] = X_test.loc[:, 'Property Type'].fillna(X_test['Property Type'].
    ↪mode()[0])

```

[14]: # replace numeric nulls with mean in X\_train and X\_test

```

X_train['Bedrooms'] = X_train.loc[:, 'Bedrooms'].fillna(X_train['Bedrooms'].
    ↪mean())
X_train['Bathrooms'] = X_train.loc[:, 'Bathrooms'].fillna(X_train['Bathrooms'].
    ↪mean())
X_train['Review Scores Rating'] = X_train.loc[:, 'Review Scores Rating'].
    ↪fillna(X_train['Review Scores Rating'].mean())

X_test['Bedrooms'] = X_test.loc[:, 'Bedrooms'].fillna(X_test['Bedrooms'].mean())
X_test['Bathrooms'] = X_test.loc[:, 'Bathrooms'].fillna(X_test['Bathrooms'].
    ↪mean())
X_test['Review Scores Rating'] = X_test.loc[:, 'Review Scores Rating'].
    ↪fillna(X_test['Review Scores Rating'].mean())

```

### 1.1.3 Step 3: Encoding and Predicting Model

[15]: #Encoding categorical values

```

# help - https://www.youtube.com/watch?v=irHhDmbw3xo&t=901s
# help - https://stackoverflow.com/questions/64656728/
    ↪using-make-column-transformer-for-categorical-variables-giving-error-during-fit
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import make_column_transformer
col_trans = make_column_transformer((OneHotEncoder(handle_unknown = 'ignore'),
    ↪['Neighbourhood Group Cleansed', 'Host
    ↪Response Time',
        'Host Response Rate', 'Property Type']),
    ↪remainder='passthrough')
X_train = col_trans.fit_transform(X_train).toarray()
X_test = col_trans.transform(X_test).toarray()

```

### Linear Regression Method

[16]: # Create linear regression object and train the model using the training sets

```

lm_model = LinearRegression(normalize = True).fit(X_train, y_train)

```

[17]: # Make predictions using the testing and train set

```

y_test_pred = lm_model.predict(X_test)
y_train_pred = lm_model.predict(X_train)

```

```
[18]: # score using model
test_score = r2_score(y_test, y_test_pred)
train_score = r2_score(y_train, y_train_pred)

print(f"R-Squared for test data: {test_score}")
print(f"R-Squared for train data: {train_score}")
```

R-Squared for test data: -1.5474230108318648e+23  
R-Squared for train data: 0.4353396555494863

As you can see from above, the model is terrible and isn't even close; this is indicated by the negative r-squared value for the test data. Therefore, I will need to find a different modeling method. To do that, I used the link below to search for different models to try out. I complemented this with many other outside research. > Helpful Link: <https://www.educative.io/blog/scikit-learn-cheat-sheet-classification-regression-methods>

If you take a look near the bottom, there are additional linear regression methods that are suggested. I will be trying out as many as I can.

### Support Vector Machine Method

```
[19]: # help -https://www.javatpoint.com/
    ↵machine-learning-support-vector-machine-algorithm
# help - https://towardsdatascience.com/
    ↵support-vector-machines-explained-with-python-examples-cb65e8172c85
from sklearn.svm import SVR
model = SVR(kernel = 'linear')
model.fit(X_train, y_train)

# Predicting train and test(same steps as above)
y_pred_test = model.predict(X_test)
y_pred_train = model.predict(X_train)

# r2_score
svr_model_train = r2_score(y_train, y_pred_train)
svr_model_test = r2_score(y_test, y_pred_test)

print(f"R-squared value for train data: {svr_model_train}")
print(f"R-squared value for test data: {svr_model_test}")

#mean_square_error help - https://www.statisticshowto.com/
    ↵probability-and-statistics/statistics-definitions/mean-squared-error/
# more help - https://scikit-learn.org/stable/modules/generated/sklearn.metrics.
    ↵mean_squared_error.html
print(f"Average Missed Distance From Price: {mean_squared_error(y_test, y_pred_test, squared = False)}")
```

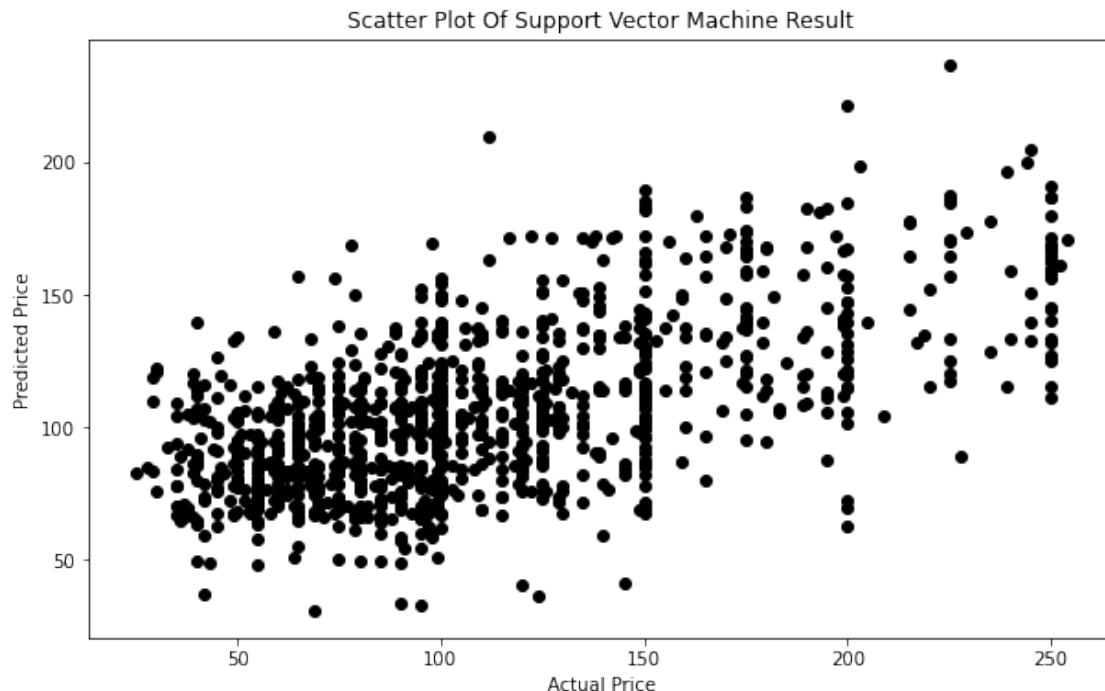
R-squared value for train data: 0.39828558893388966  
R-squared value for test data: 0.36973247235487805

```
Average Missed Distance From Price: 40.914781541448406
```

### Support Vector Machine Method Visuals

```
[20]: plt.rcParams['figure.figsize'] = [10, 6]

plt.scatter(y_test, y_pred_test, color = 'black')
plt.title('Scatter Plot Of Support Vector Machine Result')
plt.xlabel('Actual Price')
plt.ylabel('Predicted Price');
```



### XGB Regressor Method

```
[21]: # help - https://thinkingneuron.com/
       ↪how-to-create-an-xgboost-model-for-regression-in-python/
# help - https://stackoverflow.com/questions/58318685/
       ↪how-to-hide-warnings-from-xgboost-library-in-jupyter
# we can use the sqrt of mean_squared_error, to get the distance from line
from xgboost.sklearn import XGBRegressor
model = XGBRegressor('reg:linear',verbosity = 0)
model.fit(X_train, y_train)

# Predicting train and test(same steps as above)
y_pred_test = model.predict(X_test)
y_pred_train = model.predict(X_train)
```

```
# r2_score
xgb_model_train = r2_score(y_train, y_pred_train)
xgb_model_test = r2_score(y_test, y_pred_test)

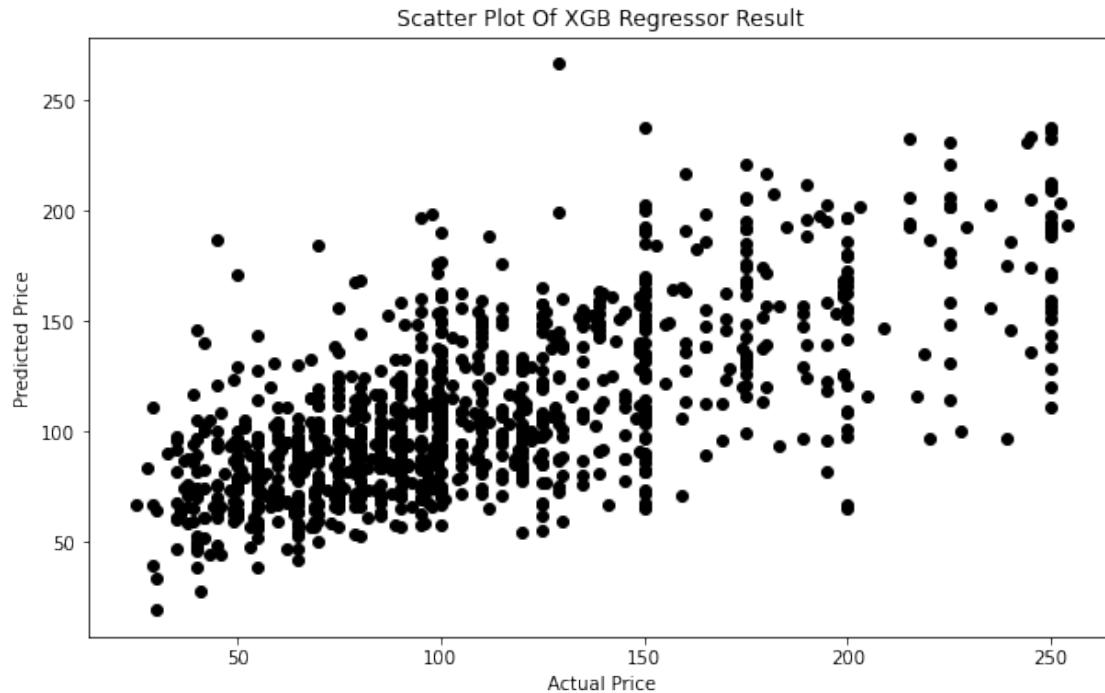
print(f"R-squared value for train data: {xgb_model_train}")
print(f"R-squared value for test data: {xgb_model_test}")
print(f"Average Missed Distance From Price: {mean_squared_error(y_test, y_pred_test, squared = False)}")
```

R-squared value for train data: 0.7822738838469437  
R-squared value for test data: 0.4570009405371742  
Average Missed Distance From Price: 37.97670746708108

### XGB Regressor Method Visuals

[22]: plt.rcParams['figure.figsize'] = [10, 6]

```
plt.scatter(y_test, y_pred_test, color = 'black')
plt.title('Scatter Plot Of XGB Regressor Result')
plt.xlabel('Actual Price')
plt.ylabel('Predicted Price');
```



### Bayesian Ridge Method

```
[23]: from sklearn.linear_model import BayesianRidge

model = BayesianRidge()
model.fit(X_train, y_train)

# Predicting train and test(same steps as above)
y_pred_test = model.predict(X_test)
y_pred_train = model.predict(X_train)

# r2_score
baye_model_train = r2_score(y_train, y_pred_train)
baye_model_test = r2_score(y_test, y_pred_test)

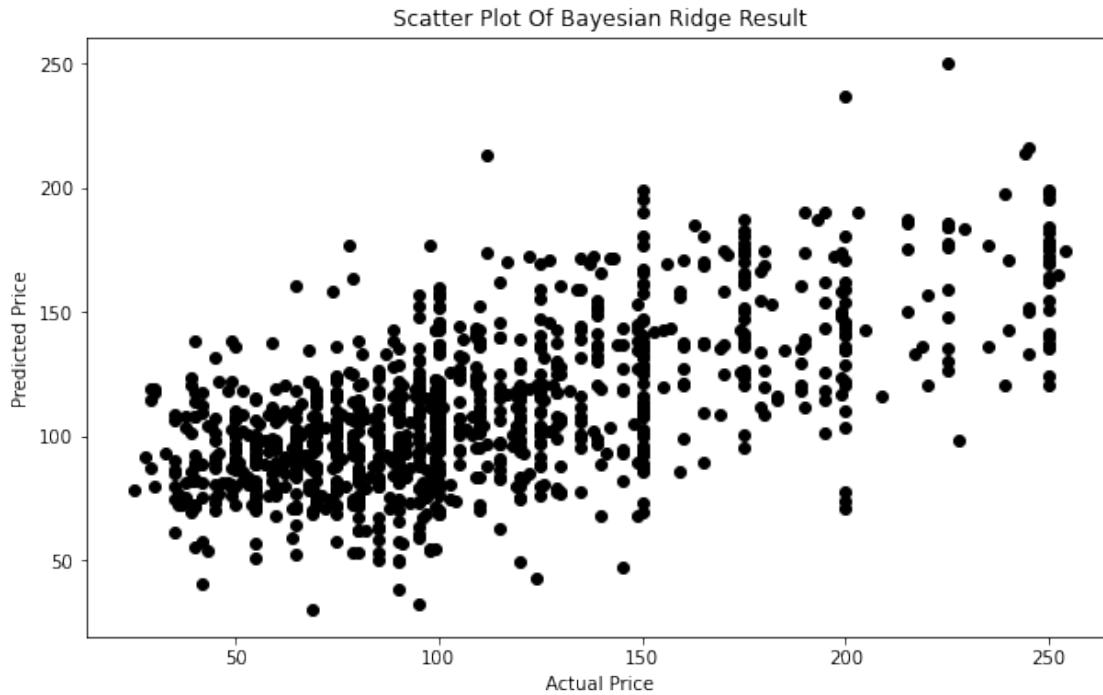
print(f"R-squared value for train data: {baye_model_train}")
print(f"R-squared value for test data: {baye_model_test}")
print(f"Average Missed Distance From Price: {mean_squared_error(y_test, y_pred_test, squared = False)}")
```

R-squared value for train data: 0.4265742873361258  
R-squared value for test data: 0.3967437735491848  
Average Missed Distance From Price: 40.02844112055365

### Bayesian Ridge Visuals

```
[24]: plt.rcParams['figure.figsize'] = [10, 6]

plt.scatter(y_test, y_pred_test, color = 'black')
plt.title('Scatter Plot Of Bayesian Ridge Result')
plt.xlabel('Actual Price')
plt.ylabel('Predicted Price');
```



### Random Forest Classifier Method

```
[25]: from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier()
model.fit(X_train, y_train)

# Predicting train and test(same steps as above)
y_pred_test = model.predict(X_test)
y_pred_train = model.predict(X_train)

# r2_score
rfc_model_train = r2_score(y_train, y_pred_train)
rfc_model_test = r2_score(y_test, y_pred_test)

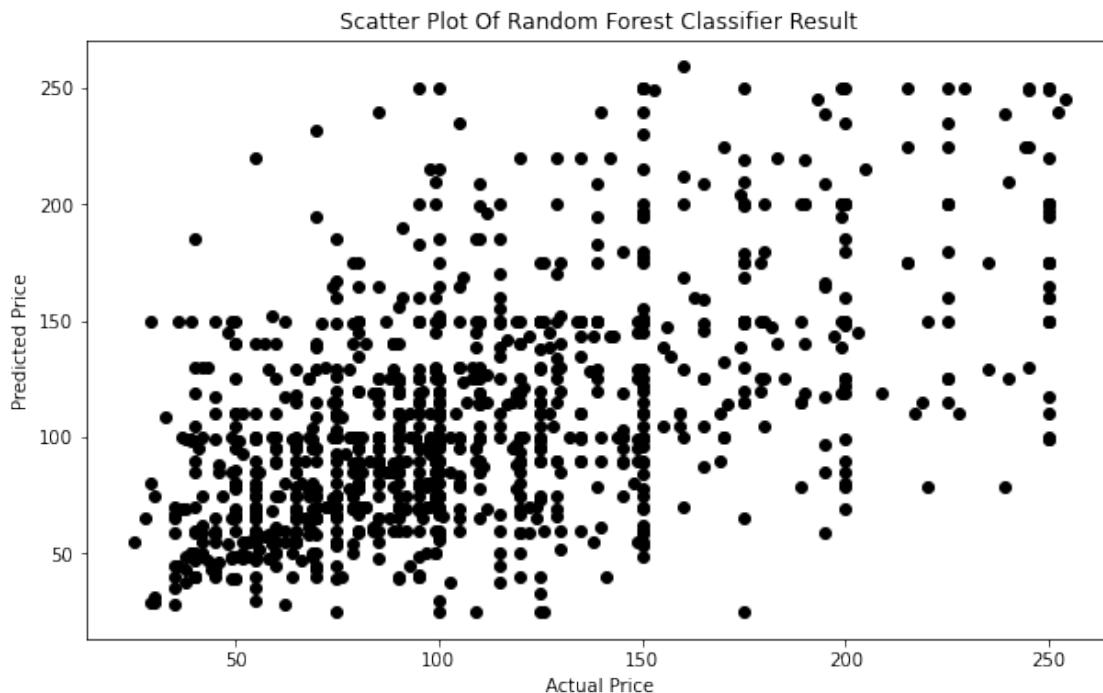
print(f"R-squared value for train data: {rfc_model_train}")
print(f"R-squared value for test data: {rfc_model_test}")
print(f"Average Missed Distance From Price: {mean_squared_error(y_test, y_pred_test, squared = False)}")
```

R-squared value for train data: 0.8798376527180126  
 R-squared value for test data: 0.16035121943307962  
 Average Missed Distance From Price: 47.224416591293746

### Random Forest Classifier Visuals

```
[26]: plt.rcParams['figure.figsize'] = [10, 6]

plt.scatter(y_test, y_pred_test, color = 'black')
plt.title('Scatter Plot Of Random Forest Classifier Result')
plt.xlabel('Actual Price')
plt.ylabel('Predicted Price');
```



### Gradient Boosting Regressor Method

```
[27]: # help - https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html

from sklearn.ensemble import GradientBoostingRegressor
# we will go with the deafult parameters

model = GradientBoostingRegressor()
model.fit(X_train, y_train)

# Predicting train and test(same steps as above)
y_pred_test = model.predict(X_test)
y_pred_train = model.predict(X_train)

# r2_score
gradient_model_train = r2_score(y_train, y_pred_train)
```

```

gradient_model_test = r2_score(y_test, y_pred_test)

print(f"R-squared value for train data: {gradient_model_train}")
print(f"R-squared value for test data: {gradient_model_test}")
print(f"Average Missed Distance From Price: {mean_squared_error(y_test, y_pred_test, squared = False)}")

```

R-squared value for train data: 0.5426674347024946  
 R-squared value for test data: 0.47566460944842215  
 Average Missed Distance From Price: 37.31834338125408

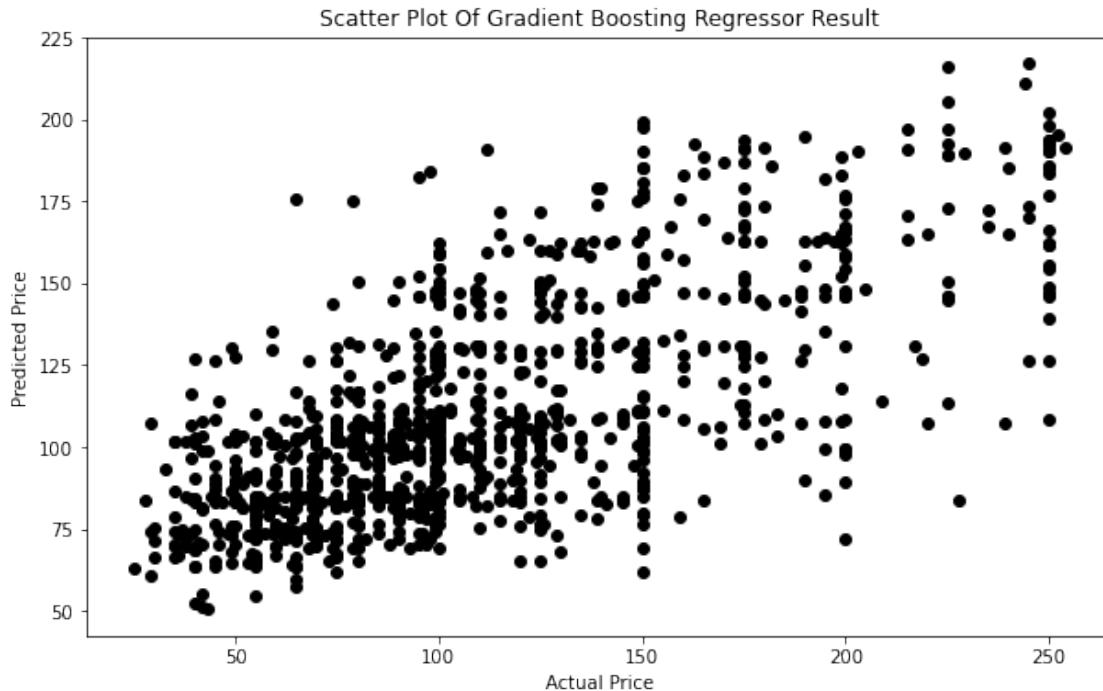
### Gradient Boosting Regressor Visuals

```

[28]: plt.rcParams['figure.figsize'] = [10, 6]

plt.scatter(y_test, y_pred_test, color = 'black')
plt.title('Scatter Plot Of Gradient Boosting Regressor Result')
plt.xlabel('Actual Price')
plt.ylabel('Predicted Price');

```



### Cat Boost Regressor Method

```

[29]: from catboost import CatBoostRegressor

model = CatBoostRegressor()

```

```

model.fit(X_train, y_train)

# Predicting train and test(same steps as above)
y_pred_test = model.predict(X_test)
y_pred_train = model.predict(X_train)

# r2_score
cat_model_train = r2_score(y_train, y_pred_train)
cat_model_test = r2_score(y_test, y_pred_test)

```

Learning rate set to 0.047321

|     |                   |               |                  |
|-----|-------------------|---------------|------------------|
| 0:  | learn: 50.5058642 | total: 54.3ms | remaining: 54.2s |
| 1:  | learn: 49.6381421 | total: 55.7ms | remaining: 27.8s |
| 2:  | learn: 48.8384827 | total: 56.8ms | remaining: 18.9s |
| 3:  | learn: 48.1009007 | total: 57.9ms | remaining: 14.4s |
| 4:  | learn: 47.3968857 | total: 58.9ms | remaining: 11.7s |
| 5:  | learn: 46.7251168 | total: 60ms   | remaining: 9.94s |
| 6:  | learn: 46.1580852 | total: 61ms   | remaining: 8.65s |
| 7:  | learn: 45.5826276 | total: 61.6ms | remaining: 7.63s |
| 8:  | learn: 45.0794893 | total: 62.6ms | remaining: 6.89s |
| 9:  | learn: 44.5680349 | total: 63.4ms | remaining: 6.28s |
| 10: | learn: 44.1039180 | total: 64.5ms | remaining: 5.8s  |
| 11: | learn: 43.6545201 | total: 65.4ms | remaining: 5.38s |
| 12: | learn: 43.2791719 | total: 66.4ms | remaining: 5.04s |
| 13: | learn: 42.9242935 | total: 67.3ms | remaining: 4.74s |
| 14: | learn: 42.5356967 | total: 68.2ms | remaining: 4.48s |
| 15: | learn: 42.1884845 | total: 69ms   | remaining: 4.24s |
| 16: | learn: 41.8623724 | total: 69.8ms | remaining: 4.04s |
| 17: | learn: 41.5787551 | total: 70.7ms | remaining: 3.85s |
| 18: | learn: 41.3125579 | total: 71.4ms | remaining: 3.69s |
| 19: | learn: 41.0502523 | total: 72.1ms | remaining: 3.53s |
| 20: | learn: 40.8505795 | total: 72.8ms | remaining: 3.4s  |
| 21: | learn: 40.6193699 | total: 73.6ms | remaining: 3.27s |
| 22: | learn: 40.4680229 | total: 74.2ms | remaining: 3.15s |
| 23: | learn: 40.2656696 | total: 74.9ms | remaining: 3.05s |
| 24: | learn: 40.1112679 | total: 75.6ms | remaining: 2.95s |
| 25: | learn: 39.9225955 | total: 76.4ms | remaining: 2.86s |
| 26: | learn: 39.7683097 | total: 77ms   | remaining: 2.78s |
| 27: | learn: 39.6233684 | total: 77.8ms | remaining: 2.7s  |
| 28: | learn: 39.4807716 | total: 78.6ms | remaining: 2.63s |
| 29: | learn: 39.3453141 | total: 79.7ms | remaining: 2.58s |
| 30: | learn: 39.2377733 | total: 80.4ms | remaining: 2.51s |
| 31: | learn: 39.1556827 | total: 81.2ms | remaining: 2.46s |
| 32: | learn: 39.0555028 | total: 82.1ms | remaining: 2.4s  |
| 33: | learn: 38.9760031 | total: 82.8ms | remaining: 2.35s |
| 34: | learn: 38.8841091 | total: 83.6ms | remaining: 2.3s  |
| 35: | learn: 38.7861895 | total: 84.4ms | remaining: 2.26s |

|     |                   |               |                  |
|-----|-------------------|---------------|------------------|
| 36: | learn: 38.6864415 | total: 85.2ms | remaining: 2.22s |
| 37: | learn: 38.5786879 | total: 86ms   | remaining: 2.17s |
| 38: | learn: 38.4921612 | total: 86.7ms | remaining: 2.13s |
| 39: | learn: 38.3973750 | total: 87.3ms | remaining: 2.1s  |
| 40: | learn: 38.3142943 | total: 88.3ms | remaining: 2.06s |
| 41: | learn: 38.2445355 | total: 89.1ms | remaining: 2.03s |
| 42: | learn: 38.1932544 | total: 89.9ms | remaining: 2s    |
| 43: | learn: 38.1348321 | total: 90.7ms | remaining: 1.97s |
| 44: | learn: 38.0559097 | total: 91.4ms | remaining: 1.94s |
| 45: | learn: 37.9794490 | total: 92.1ms | remaining: 1.91s |
| 46: | learn: 37.9084128 | total: 92.8ms | remaining: 1.88s |
| 47: | learn: 37.8528232 | total: 93.6ms | remaining: 1.86s |
| 48: | learn: 37.7984529 | total: 94.6ms | remaining: 1.84s |
| 49: | learn: 37.7501569 | total: 95.4ms | remaining: 1.81s |
| 50: | learn: 37.6923433 | total: 96.2ms | remaining: 1.79s |
| 51: | learn: 37.6445328 | total: 97ms   | remaining: 1.77s |
| 52: | learn: 37.5993915 | total: 97.7ms | remaining: 1.75s |
| 53: | learn: 37.5703594 | total: 98.5ms | remaining: 1.73s |
| 54: | learn: 37.5333536 | total: 99.3ms | remaining: 1.71s |
| 55: | learn: 37.4986357 | total: 100ms  | remaining: 1.69s |
| 56: | learn: 37.4646866 | total: 101ms  | remaining: 1.67s |
| 57: | learn: 37.4258003 | total: 102ms  | remaining: 1.65s |
| 58: | learn: 37.3820907 | total: 102ms  | remaining: 1.63s |
| 59: | learn: 37.3450200 | total: 103ms  | remaining: 1.61s |
| 60: | learn: 37.3197855 | total: 104ms  | remaining: 1.6s  |
| 61: | learn: 37.2888655 | total: 104ms  | remaining: 1.58s |
| 62: | learn: 37.2574988 | total: 105ms  | remaining: 1.56s |
| 63: | learn: 37.2314109 | total: 106ms  | remaining: 1.55s |
| 64: | learn: 37.1979850 | total: 107ms  | remaining: 1.54s |
| 65: | learn: 37.1708762 | total: 108ms  | remaining: 1.52s |
| 66: | learn: 37.1472760 | total: 108ms  | remaining: 1.51s |
| 67: | learn: 37.1166971 | total: 109ms  | remaining: 1.5s  |
| 68: | learn: 37.0784403 | total: 110ms  | remaining: 1.49s |
| 69: | learn: 37.0363970 | total: 111ms  | remaining: 1.48s |
| 70: | learn: 37.0047751 | total: 112ms  | remaining: 1.46s |
| 71: | learn: 36.9746919 | total: 113ms  | remaining: 1.45s |
| 72: | learn: 36.9320554 | total: 113ms  | remaining: 1.44s |
| 73: | learn: 36.9101541 | total: 114ms  | remaining: 1.43s |
| 74: | learn: 36.8911479 | total: 115ms  | remaining: 1.42s |
| 75: | learn: 36.8761510 | total: 116ms  | remaining: 1.41s |
| 76: | learn: 36.8462248 | total: 117ms  | remaining: 1.4s  |
| 77: | learn: 36.8222845 | total: 117ms  | remaining: 1.39s |
| 78: | learn: 36.7993066 | total: 118ms  | remaining: 1.38s |
| 79: | learn: 36.7792838 | total: 119ms  | remaining: 1.37s |
| 80: | learn: 36.7434545 | total: 120ms  | remaining: 1.36s |
| 81: | learn: 36.7033230 | total: 122ms  | remaining: 1.36s |
| 82: | learn: 36.6800372 | total: 122ms  | remaining: 1.35s |
| 83: | learn: 36.6365464 | total: 124ms  | remaining: 1.35s |

|      |                   |              |                  |
|------|-------------------|--------------|------------------|
| 84:  | learn: 36.6177630 | total: 125ms | remaining: 1.34s |
| 85:  | learn: 36.5893338 | total: 126ms | remaining: 1.33s |
| 86:  | learn: 36.5502359 | total: 126ms | remaining: 1.32s |
| 87:  | learn: 36.5350449 | total: 127ms | remaining: 1.32s |
| 88:  | learn: 36.5201083 | total: 128ms | remaining: 1.31s |
| 89:  | learn: 36.5049285 | total: 129ms | remaining: 1.3s  |
| 90:  | learn: 36.4786401 | total: 129ms | remaining: 1.29s |
| 91:  | learn: 36.4402695 | total: 130ms | remaining: 1.28s |
| 92:  | learn: 36.4248002 | total: 131ms | remaining: 1.28s |
| 93:  | learn: 36.4077597 | total: 132ms | remaining: 1.27s |
| 94:  | learn: 36.3897297 | total: 133ms | remaining: 1.27s |
| 95:  | learn: 36.3765014 | total: 134ms | remaining: 1.26s |
| 96:  | learn: 36.3617023 | total: 135ms | remaining: 1.25s |
| 97:  | learn: 36.3314946 | total: 135ms | remaining: 1.25s |
| 98:  | learn: 36.3067874 | total: 136ms | remaining: 1.24s |
| 99:  | learn: 36.2853874 | total: 137ms | remaining: 1.23s |
| 100: | learn: 36.2630136 | total: 138ms | remaining: 1.23s |
| 101: | learn: 36.2489263 | total: 139ms | remaining: 1.22s |
| 102: | learn: 36.2243755 | total: 140ms | remaining: 1.21s |
| 103: | learn: 36.2072593 | total: 140ms | remaining: 1.21s |
| 104: | learn: 36.1990296 | total: 141ms | remaining: 1.2s  |
| 105: | learn: 36.1807009 | total: 142ms | remaining: 1.2s  |
| 106: | learn: 36.1726827 | total: 143ms | remaining: 1.19s |
| 107: | learn: 36.1629834 | total: 144ms | remaining: 1.19s |
| 108: | learn: 36.1505683 | total: 144ms | remaining: 1.18s |
| 109: | learn: 36.1347927 | total: 146ms | remaining: 1.18s |
| 110: | learn: 36.1019401 | total: 147ms | remaining: 1.18s |
| 111: | learn: 36.0831629 | total: 148ms | remaining: 1.17s |
| 112: | learn: 36.0606900 | total: 149ms | remaining: 1.17s |
| 113: | learn: 36.0444748 | total: 150ms | remaining: 1.17s |
| 114: | learn: 36.0307169 | total: 151ms | remaining: 1.16s |
| 115: | learn: 36.0223201 | total: 152ms | remaining: 1.16s |
| 116: | learn: 35.9983423 | total: 153ms | remaining: 1.15s |
| 117: | learn: 35.9882039 | total: 154ms | remaining: 1.15s |
| 118: | learn: 35.9560898 | total: 155ms | remaining: 1.14s |
| 119: | learn: 35.9303829 | total: 155ms | remaining: 1.14s |
| 120: | learn: 35.9130141 | total: 156ms | remaining: 1.13s |
| 121: | learn: 35.9052112 | total: 157ms | remaining: 1.13s |
| 122: | learn: 35.9033698 | total: 157ms | remaining: 1.12s |
| 123: | learn: 35.8706005 | total: 158ms | remaining: 1.12s |
| 124: | learn: 35.8473774 | total: 159ms | remaining: 1.11s |
| 125: | learn: 35.8174511 | total: 160ms | remaining: 1.11s |
| 126: | learn: 35.7976614 | total: 161ms | remaining: 1.11s |
| 127: | learn: 35.7576831 | total: 162ms | remaining: 1.1s  |
| 128: | learn: 35.7501779 | total: 163ms | remaining: 1.1s  |
| 129: | learn: 35.7307956 | total: 163ms | remaining: 1.09s |
| 130: | learn: 35.7182903 | total: 164ms | remaining: 1.09s |
| 131: | learn: 35.6885480 | total: 165ms | remaining: 1.08s |

|      |                   |              |                  |
|------|-------------------|--------------|------------------|
| 132: | learn: 35.6786537 | total: 166ms | remaining: 1.08s |
| 133: | learn: 35.6601476 | total: 167ms | remaining: 1.08s |
| 134: | learn: 35.6387126 | total: 168ms | remaining: 1.07s |
| 135: | learn: 35.6341906 | total: 168ms | remaining: 1.07s |
| 136: | learn: 35.6272875 | total: 169ms | remaining: 1.06s |
| 137: | learn: 35.6134127 | total: 170ms | remaining: 1.06s |
| 138: | learn: 35.5904114 | total: 171ms | remaining: 1.06s |
| 139: | learn: 35.5659684 | total: 171ms | remaining: 1.05s |
| 140: | learn: 35.5418021 | total: 172ms | remaining: 1.05s |
| 141: | learn: 35.5168298 | total: 173ms | remaining: 1.04s |
| 142: | learn: 35.5083937 | total: 174ms | remaining: 1.04s |
| 143: | learn: 35.4934255 | total: 174ms | remaining: 1.03s |
| 144: | learn: 35.4686792 | total: 175ms | remaining: 1.03s |
| 145: | learn: 35.4484168 | total: 176ms | remaining: 1.03s |
| 146: | learn: 35.4291528 | total: 178ms | remaining: 1.03s |
| 147: | learn: 35.4124602 | total: 179ms | remaining: 1.03s |
| 148: | learn: 35.3944105 | total: 179ms | remaining: 1.02s |
| 149: | learn: 35.3877184 | total: 180ms | remaining: 1.02s |
| 150: | learn: 35.3738673 | total: 182ms | remaining: 1.02s |
| 151: | learn: 35.3659955 | total: 183ms | remaining: 1.02s |
| 152: | learn: 35.3541829 | total: 183ms | remaining: 1.01s |
| 153: | learn: 35.3481115 | total: 184ms | remaining: 1.01s |
| 154: | learn: 35.3221992 | total: 185ms | remaining: 1.01s |
| 155: | learn: 35.3028296 | total: 186ms | remaining: 1.01s |
| 156: | learn: 35.2703641 | total: 187ms | remaining: 1s    |
| 157: | learn: 35.2543936 | total: 188ms | remaining: 1s    |
| 158: | learn: 35.2203363 | total: 190ms | remaining: 1s    |
| 159: | learn: 35.2108313 | total: 191ms | remaining: 1s    |
| 160: | learn: 35.1903965 | total: 192ms | remaining: 999ms |
| 161: | learn: 35.1765147 | total: 193ms | remaining: 999ms |
| 162: | learn: 35.1600953 | total: 194ms | remaining: 996ms |
| 163: | learn: 35.1441688 | total: 195ms | remaining: 993ms |
| 164: | learn: 35.1108958 | total: 196ms | remaining: 990ms |
| 165: | learn: 35.0927043 | total: 196ms | remaining: 987ms |
| 166: | learn: 35.0792384 | total: 197ms | remaining: 983ms |
| 167: | learn: 35.0647864 | total: 198ms | remaining: 980ms |
| 168: | learn: 35.0566000 | total: 199ms | remaining: 977ms |
| 169: | learn: 35.0313308 | total: 199ms | remaining: 974ms |
| 170: | learn: 35.0254970 | total: 200ms | remaining: 971ms |
| 171: | learn: 35.0177013 | total: 201ms | remaining: 967ms |
| 172: | learn: 35.0075135 | total: 202ms | remaining: 964ms |
| 173: | learn: 34.9820788 | total: 203ms | remaining: 962ms |
| 174: | learn: 34.9629710 | total: 204ms | remaining: 963ms |
| 175: | learn: 34.9469933 | total: 206ms | remaining: 964ms |
| 176: | learn: 34.9305305 | total: 207ms | remaining: 961ms |
| 177: | learn: 34.9176855 | total: 207ms | remaining: 958ms |
| 178: | learn: 34.9079540 | total: 208ms | remaining: 955ms |
| 179: | learn: 34.8947844 | total: 209ms | remaining: 952ms |

|      |                   |              |                  |
|------|-------------------|--------------|------------------|
| 180: | learn: 34.8787236 | total: 210ms | remaining: 949ms |
| 181: | learn: 34.8682350 | total: 211ms | remaining: 949ms |
| 182: | learn: 34.8348719 | total: 212ms | remaining: 946ms |
| 183: | learn: 34.8230799 | total: 213ms | remaining: 944ms |
| 184: | learn: 34.8011305 | total: 214ms | remaining: 942ms |
| 185: | learn: 34.7925915 | total: 215ms | remaining: 939ms |
| 186: | learn: 34.7757320 | total: 215ms | remaining: 936ms |
| 187: | learn: 34.7502472 | total: 216ms | remaining: 933ms |
| 188: | learn: 34.7462541 | total: 218ms | remaining: 934ms |
| 189: | learn: 34.7259123 | total: 219ms | remaining: 932ms |
| 190: | learn: 34.6923262 | total: 219ms | remaining: 930ms |
| 191: | learn: 34.6765569 | total: 221ms | remaining: 928ms |
| 192: | learn: 34.6515209 | total: 221ms | remaining: 925ms |
| 193: | learn: 34.6413421 | total: 223ms | remaining: 925ms |
| 194: | learn: 34.6176942 | total: 227ms | remaining: 939ms |
| 195: | learn: 34.6092652 | total: 228ms | remaining: 937ms |
| 196: | learn: 34.5917660 | total: 229ms | remaining: 935ms |
| 197: | learn: 34.5799727 | total: 230ms | remaining: 933ms |
| 198: | learn: 34.5753484 | total: 232ms | remaining: 935ms |
| 199: | learn: 34.5566909 | total: 233ms | remaining: 932ms |
| 200: | learn: 34.5443363 | total: 234ms | remaining: 932ms |
| 201: | learn: 34.5242635 | total: 235ms | remaining: 930ms |
| 202: | learn: 34.5160126 | total: 237ms | remaining: 929ms |
| 203: | learn: 34.5032694 | total: 238ms | remaining: 929ms |
| 204: | learn: 34.4861734 | total: 239ms | remaining: 927ms |
| 205: | learn: 34.4467971 | total: 240ms | remaining: 924ms |
| 206: | learn: 34.4255652 | total: 241ms | remaining: 922ms |
| 207: | learn: 34.4228360 | total: 242ms | remaining: 920ms |
| 208: | learn: 34.4172918 | total: 242ms | remaining: 917ms |
| 209: | learn: 34.3999611 | total: 243ms | remaining: 915ms |
| 210: | learn: 34.3974327 | total: 244ms | remaining: 913ms |
| 211: | learn: 34.3719487 | total: 245ms | remaining: 911ms |
| 212: | learn: 34.3689139 | total: 246ms | remaining: 908ms |
| 213: | learn: 34.3410907 | total: 247ms | remaining: 906ms |
| 214: | learn: 34.3317013 | total: 247ms | remaining: 903ms |
| 215: | learn: 34.3133749 | total: 249ms | remaining: 903ms |
| 216: | learn: 34.3097679 | total: 249ms | remaining: 900ms |
| 217: | learn: 34.2911668 | total: 250ms | remaining: 898ms |
| 218: | learn: 34.2743820 | total: 251ms | remaining: 896ms |
| 219: | learn: 34.2567313 | total: 252ms | remaining: 895ms |
| 220: | learn: 34.2365493 | total: 253ms | remaining: 892ms |
| 221: | learn: 34.2066884 | total: 254ms | remaining: 890ms |
| 222: | learn: 34.2004076 | total: 255ms | remaining: 887ms |
| 223: | learn: 34.1656569 | total: 256ms | remaining: 885ms |
| 224: | learn: 34.1471204 | total: 256ms | remaining: 883ms |
| 225: | learn: 34.1182838 | total: 257ms | remaining: 881ms |
| 226: | learn: 34.1038647 | total: 258ms | remaining: 878ms |
| 227: | learn: 34.0849097 | total: 259ms | remaining: 876ms |

|      |                   |              |                  |
|------|-------------------|--------------|------------------|
| 228: | learn: 34.0818870 | total: 260ms | remaining: 875ms |
| 229: | learn: 34.0645030 | total: 261ms | remaining: 873ms |
| 230: | learn: 34.0318622 | total: 262ms | remaining: 871ms |
| 231: | learn: 34.0294815 | total: 262ms | remaining: 869ms |
| 232: | learn: 34.0263894 | total: 263ms | remaining: 866ms |
| 233: | learn: 34.0095216 | total: 264ms | remaining: 864ms |
| 234: | learn: 33.9982949 | total: 265ms | remaining: 862ms |
| 235: | learn: 33.9847787 | total: 265ms | remaining: 859ms |
| 236: | learn: 33.9701826 | total: 267ms | remaining: 859ms |
| 237: | learn: 33.9676452 | total: 268ms | remaining: 857ms |
| 238: | learn: 33.9652958 | total: 268ms | remaining: 854ms |
| 239: | learn: 33.9394918 | total: 269ms | remaining: 853ms |
| 240: | learn: 33.9149930 | total: 270ms | remaining: 851ms |
| 241: | learn: 33.8976269 | total: 271ms | remaining: 850ms |
| 242: | learn: 33.8907800 | total: 272ms | remaining: 849ms |
| 243: | learn: 33.8738580 | total: 273ms | remaining: 846ms |
| 244: | learn: 33.8710470 | total: 274ms | remaining: 844ms |
| 245: | learn: 33.8463147 | total: 275ms | remaining: 842ms |
| 246: | learn: 33.8201857 | total: 276ms | remaining: 842ms |
| 247: | learn: 33.8042189 | total: 277ms | remaining: 840ms |
| 248: | learn: 33.7893432 | total: 278ms | remaining: 837ms |
| 249: | learn: 33.7762960 | total: 278ms | remaining: 835ms |
| 250: | learn: 33.7580185 | total: 279ms | remaining: 833ms |
| 251: | learn: 33.7441363 | total: 280ms | remaining: 831ms |
| 252: | learn: 33.7410645 | total: 281ms | remaining: 829ms |
| 253: | learn: 33.7308093 | total: 281ms | remaining: 827ms |
| 254: | learn: 33.7198531 | total: 282ms | remaining: 824ms |
| 255: | learn: 33.6940819 | total: 283ms | remaining: 822ms |
| 256: | learn: 33.6736001 | total: 284ms | remaining: 820ms |
| 257: | learn: 33.6706488 | total: 284ms | remaining: 818ms |
| 258: | learn: 33.6638906 | total: 285ms | remaining: 816ms |
| 259: | learn: 33.6609721 | total: 286ms | remaining: 814ms |
| 260: | learn: 33.6381416 | total: 287ms | remaining: 812ms |
| 261: | learn: 33.6279447 | total: 287ms | remaining: 810ms |
| 262: | learn: 33.6256764 | total: 288ms | remaining: 808ms |
| 263: | learn: 33.6051730 | total: 290ms | remaining: 810ms |
| 264: | learn: 33.5965981 | total: 291ms | remaining: 808ms |
| 265: | learn: 33.5816861 | total: 292ms | remaining: 806ms |
| 266: | learn: 33.5677991 | total: 293ms | remaining: 804ms |
| 267: | learn: 33.5654693 | total: 294ms | remaining: 803ms |
| 268: | learn: 33.5632150 | total: 295ms | remaining: 801ms |
| 269: | learn: 33.5508040 | total: 295ms | remaining: 799ms |
| 270: | learn: 33.5487046 | total: 296ms | remaining: 797ms |
| 271: | learn: 33.5423887 | total: 297ms | remaining: 795ms |
| 272: | learn: 33.5276991 | total: 298ms | remaining: 793ms |
| 273: | learn: 33.5126484 | total: 299ms | remaining: 791ms |
| 274: | learn: 33.4824011 | total: 299ms | remaining: 789ms |
| 275: | learn: 33.4696118 | total: 300ms | remaining: 787ms |

|      |                   |              |                  |
|------|-------------------|--------------|------------------|
| 276: | learn: 33.4675044 | total: 301ms | remaining: 785ms |
| 277: | learn: 33.4443826 | total: 302ms | remaining: 784ms |
| 278: | learn: 33.4298712 | total: 303ms | remaining: 782ms |
| 279: | learn: 33.4130534 | total: 304ms | remaining: 781ms |
| 280: | learn: 33.3950976 | total: 305ms | remaining: 779ms |
| 281: | learn: 33.3914352 | total: 305ms | remaining: 777ms |
| 282: | learn: 33.3889557 | total: 306ms | remaining: 775ms |
| 283: | learn: 33.3708103 | total: 307ms | remaining: 773ms |
| 284: | learn: 33.3373087 | total: 307ms | remaining: 771ms |
| 285: | learn: 33.3314280 | total: 308ms | remaining: 770ms |
| 286: | learn: 33.3179094 | total: 309ms | remaining: 768ms |
| 287: | learn: 33.3033596 | total: 310ms | remaining: 766ms |
| 288: | learn: 33.2939300 | total: 311ms | remaining: 765ms |
| 289: | learn: 33.2919971 | total: 312ms | remaining: 764ms |
| 290: | learn: 33.2728021 | total: 313ms | remaining: 762ms |
| 291: | learn: 33.2565053 | total: 314ms | remaining: 760ms |
| 292: | learn: 33.2387908 | total: 315ms | remaining: 759ms |
| 293: | learn: 33.2318011 | total: 316ms | remaining: 758ms |
| 294: | learn: 33.2299472 | total: 317ms | remaining: 758ms |
| 295: | learn: 33.2281611 | total: 319ms | remaining: 758ms |
| 296: | learn: 33.2181532 | total: 320ms | remaining: 757ms |
| 297: | learn: 33.2074877 | total: 320ms | remaining: 755ms |
| 298: | learn: 33.1995751 | total: 321ms | remaining: 753ms |
| 299: | learn: 33.1981207 | total: 322ms | remaining: 751ms |
| 300: | learn: 33.1856338 | total: 323ms | remaining: 750ms |
| 301: | learn: 33.1789662 | total: 324ms | remaining: 749ms |
| 302: | learn: 33.1768906 | total: 325ms | remaining: 748ms |
| 303: | learn: 33.1571678 | total: 326ms | remaining: 746ms |
| 304: | learn: 33.1554527 | total: 327ms | remaining: 744ms |
| 305: | learn: 33.1220538 | total: 327ms | remaining: 742ms |
| 306: | learn: 33.1050867 | total: 328ms | remaining: 741ms |
| 307: | learn: 33.0904275 | total: 329ms | remaining: 740ms |
| 308: | learn: 33.0810709 | total: 330ms | remaining: 739ms |
| 309: | learn: 33.0732128 | total: 331ms | remaining: 737ms |
| 310: | learn: 33.0602476 | total: 332ms | remaining: 735ms |
| 311: | learn: 33.0585747 | total: 333ms | remaining: 733ms |
| 312: | learn: 33.0458459 | total: 333ms | remaining: 731ms |
| 313: | learn: 33.0184804 | total: 334ms | remaining: 730ms |
| 314: | learn: 33.0162162 | total: 335ms | remaining: 728ms |
| 315: | learn: 33.0147152 | total: 336ms | remaining: 727ms |
| 316: | learn: 33.0057598 | total: 336ms | remaining: 725ms |
| 317: | learn: 32.9946957 | total: 337ms | remaining: 723ms |
| 318: | learn: 32.9804148 | total: 338ms | remaining: 721ms |
| 319: | learn: 32.9784419 | total: 339ms | remaining: 719ms |
| 320: | learn: 32.9631664 | total: 340ms | remaining: 719ms |
| 321: | learn: 32.9569095 | total: 341ms | remaining: 718ms |
| 322: | learn: 32.9553808 | total: 342ms | remaining: 716ms |
| 323: | learn: 32.9471440 | total: 343ms | remaining: 716ms |

|      |                   |              |                  |
|------|-------------------|--------------|------------------|
| 324: | learn: 32.9456685 | total: 344ms | remaining: 715ms |
| 325: | learn: 32.9438871 | total: 345ms | remaining: 713ms |
| 326: | learn: 32.9423957 | total: 346ms | remaining: 711ms |
| 327: | learn: 32.9411221 | total: 346ms | remaining: 710ms |
| 328: | learn: 32.9355246 | total: 347ms | remaining: 708ms |
| 329: | learn: 32.9342937 | total: 348ms | remaining: 707ms |
| 330: | learn: 32.9290828 | total: 349ms | remaining: 705ms |
| 331: | learn: 32.9094656 | total: 349ms | remaining: 703ms |
| 332: | learn: 32.9081623 | total: 350ms | remaining: 702ms |
| 333: | learn: 32.9068914 | total: 351ms | remaining: 700ms |
| 334: | learn: 32.8999483 | total: 352ms | remaining: 698ms |
| 335: | learn: 32.8979329 | total: 353ms | remaining: 697ms |
| 336: | learn: 32.8740121 | total: 353ms | remaining: 695ms |
| 337: | learn: 32.8727817 | total: 354ms | remaining: 694ms |
| 338: | learn: 32.8605033 | total: 355ms | remaining: 693ms |
| 339: | learn: 32.8487909 | total: 356ms | remaining: 691ms |
| 340: | learn: 32.8463528 | total: 357ms | remaining: 690ms |
| 341: | learn: 32.8386371 | total: 358ms | remaining: 689ms |
| 342: | learn: 32.8229799 | total: 359ms | remaining: 687ms |
| 343: | learn: 32.8074322 | total: 360ms | remaining: 686ms |
| 344: | learn: 32.7905090 | total: 360ms | remaining: 684ms |
| 345: | learn: 32.7639627 | total: 361ms | remaining: 683ms |
| 346: | learn: 32.7580156 | total: 362ms | remaining: 681ms |
| 347: | learn: 32.7480250 | total: 363ms | remaining: 680ms |
| 348: | learn: 32.7317763 | total: 364ms | remaining: 678ms |
| 349: | learn: 32.7188437 | total: 365ms | remaining: 677ms |
| 350: | learn: 32.7177017 | total: 365ms | remaining: 676ms |
| 351: | learn: 32.7028934 | total: 366ms | remaining: 674ms |
| 352: | learn: 32.6965916 | total: 367ms | remaining: 672ms |
| 353: | learn: 32.6947029 | total: 368ms | remaining: 671ms |
| 354: | learn: 32.6800903 | total: 368ms | remaining: 669ms |
| 355: | learn: 32.6790585 | total: 369ms | remaining: 668ms |
| 356: | learn: 32.6642303 | total: 370ms | remaining: 666ms |
| 357: | learn: 32.6489201 | total: 371ms | remaining: 666ms |
| 358: | learn: 32.6296391 | total: 372ms | remaining: 664ms |
| 359: | learn: 32.6262264 | total: 373ms | remaining: 663ms |
| 360: | learn: 32.6207100 | total: 374ms | remaining: 661ms |
| 361: | learn: 32.6103200 | total: 374ms | remaining: 660ms |
| 362: | learn: 32.6092242 | total: 375ms | remaining: 658ms |
| 363: | learn: 32.5982145 | total: 376ms | remaining: 657ms |
| 364: | learn: 32.5972955 | total: 377ms | remaining: 655ms |
| 365: | learn: 32.5907226 | total: 378ms | remaining: 654ms |
| 366: | learn: 32.5889598 | total: 379ms | remaining: 653ms |
| 367: | learn: 32.5873343 | total: 379ms | remaining: 651ms |
| 368: | learn: 32.5629113 | total: 380ms | remaining: 650ms |
| 369: | learn: 32.5423689 | total: 381ms | remaining: 648ms |
| 370: | learn: 32.5353059 | total: 382ms | remaining: 647ms |
| 371: | learn: 32.5196201 | total: 383ms | remaining: 646ms |

|      |                   |              |                  |
|------|-------------------|--------------|------------------|
| 372: | learn: 32.5180495 | total: 383ms | remaining: 644ms |
| 373: | learn: 32.5053837 | total: 384ms | remaining: 643ms |
| 374: | learn: 32.4902944 | total: 385ms | remaining: 641ms |
| 375: | learn: 32.4789111 | total: 386ms | remaining: 641ms |
| 376: | learn: 32.4689661 | total: 388ms | remaining: 641ms |
| 377: | learn: 32.4658171 | total: 389ms | remaining: 640ms |
| 378: | learn: 32.4607253 | total: 390ms | remaining: 638ms |
| 379: | learn: 32.4468714 | total: 390ms | remaining: 637ms |
| 380: | learn: 32.4407985 | total: 391ms | remaining: 635ms |
| 381: | learn: 32.4272387 | total: 392ms | remaining: 634ms |
| 382: | learn: 32.4121648 | total: 393ms | remaining: 633ms |
| 383: | learn: 32.4060931 | total: 395ms | remaining: 633ms |
| 384: | learn: 32.3824936 | total: 396ms | remaining: 633ms |
| 385: | learn: 32.3664850 | total: 397ms | remaining: 631ms |
| 386: | learn: 32.3579591 | total: 398ms | remaining: 630ms |
| 387: | learn: 32.3519537 | total: 399ms | remaining: 629ms |
| 388: | learn: 32.3354060 | total: 401ms | remaining: 630ms |
| 389: | learn: 32.3345873 | total: 402ms | remaining: 628ms |
| 390: | learn: 32.3337830 | total: 402ms | remaining: 627ms |
| 391: | learn: 32.3221560 | total: 405ms | remaining: 628ms |
| 392: | learn: 32.3080152 | total: 407ms | remaining: 628ms |
| 393: | learn: 32.3071968 | total: 407ms | remaining: 626ms |
| 394: | learn: 32.2950335 | total: 408ms | remaining: 625ms |
| 395: | learn: 32.2936509 | total: 409ms | remaining: 624ms |
| 396: | learn: 32.2806654 | total: 410ms | remaining: 622ms |
| 397: | learn: 32.2796240 | total: 410ms | remaining: 621ms |
| 398: | learn: 32.2674304 | total: 411ms | remaining: 620ms |
| 399: | learn: 32.2666417 | total: 412ms | remaining: 618ms |
| 400: | learn: 32.2591206 | total: 413ms | remaining: 617ms |
| 401: | learn: 32.2438350 | total: 414ms | remaining: 615ms |
| 402: | learn: 32.2285428 | total: 414ms | remaining: 614ms |
| 403: | learn: 32.2277730 | total: 415ms | remaining: 612ms |
| 404: | learn: 32.2136355 | total: 416ms | remaining: 611ms |
| 405: | learn: 32.2128833 | total: 417ms | remaining: 609ms |
| 406: | learn: 32.2066391 | total: 417ms | remaining: 608ms |
| 407: | learn: 32.1956935 | total: 418ms | remaining: 606ms |
| 408: | learn: 32.1864594 | total: 419ms | remaining: 606ms |
| 409: | learn: 32.1752204 | total: 420ms | remaining: 604ms |
| 410: | learn: 32.1640222 | total: 421ms | remaining: 603ms |
| 411: | learn: 32.1526211 | total: 421ms | remaining: 601ms |
| 412: | learn: 32.1429186 | total: 422ms | remaining: 600ms |
| 413: | learn: 32.1276397 | total: 423ms | remaining: 599ms |
| 414: | learn: 32.1202150 | total: 424ms | remaining: 597ms |
| 415: | learn: 32.1112687 | total: 424ms | remaining: 596ms |
| 416: | learn: 32.1023564 | total: 425ms | remaining: 594ms |
| 417: | learn: 32.0935413 | total: 426ms | remaining: 593ms |
| 418: | learn: 32.0920399 | total: 426ms | remaining: 591ms |
| 419: | learn: 32.0855945 | total: 427ms | remaining: 590ms |

|      |                   |              |                  |
|------|-------------------|--------------|------------------|
| 420: | learn: 32.0747439 | total: 428ms | remaining: 588ms |
| 421: | learn: 32.0653571 | total: 429ms | remaining: 587ms |
| 422: | learn: 32.0522063 | total: 429ms | remaining: 586ms |
| 423: | learn: 32.0324636 | total: 430ms | remaining: 584ms |
| 424: | learn: 32.0197981 | total: 431ms | remaining: 582ms |
| 425: | learn: 32.0158021 | total: 431ms | remaining: 581ms |
| 426: | learn: 32.0012917 | total: 432ms | remaining: 579ms |
| 427: | learn: 31.9974198 | total: 432ms | remaining: 578ms |
| 428: | learn: 31.9961753 | total: 433ms | remaining: 576ms |
| 429: | learn: 31.9833499 | total: 434ms | remaining: 575ms |
| 430: | learn: 31.9825128 | total: 434ms | remaining: 573ms |
| 431: | learn: 31.9813331 | total: 435ms | remaining: 572ms |
| 432: | learn: 31.9763913 | total: 436ms | remaining: 570ms |
| 433: | learn: 31.9631526 | total: 437ms | remaining: 570ms |
| 434: | learn: 31.9578595 | total: 438ms | remaining: 568ms |
| 435: | learn: 31.9506011 | total: 438ms | remaining: 567ms |
| 436: | learn: 31.9341903 | total: 439ms | remaining: 565ms |
| 437: | learn: 31.9081072 | total: 440ms | remaining: 564ms |
| 438: | learn: 31.9074035 | total: 440ms | remaining: 562ms |
| 439: | learn: 31.8988638 | total: 441ms | remaining: 561ms |
| 440: | learn: 31.8851732 | total: 441ms | remaining: 559ms |
| 441: | learn: 31.8843029 | total: 442ms | remaining: 558ms |
| 442: | learn: 31.8731498 | total: 443ms | remaining: 557ms |
| 443: | learn: 31.8563532 | total: 444ms | remaining: 556ms |
| 444: | learn: 31.8466191 | total: 444ms | remaining: 554ms |
| 445: | learn: 31.8423707 | total: 445ms | remaining: 553ms |
| 446: | learn: 31.8409853 | total: 445ms | remaining: 551ms |
| 447: | learn: 31.8403215 | total: 446ms | remaining: 549ms |
| 448: | learn: 31.8268329 | total: 447ms | remaining: 548ms |
| 449: | learn: 31.8139353 | total: 447ms | remaining: 547ms |
| 450: | learn: 31.7981057 | total: 448ms | remaining: 545ms |
| 451: | learn: 31.7909738 | total: 449ms | remaining: 544ms |
| 452: | learn: 31.7846569 | total: 449ms | remaining: 542ms |
| 453: | learn: 31.7775664 | total: 450ms | remaining: 541ms |
| 454: | learn: 31.7703970 | total: 451ms | remaining: 540ms |
| 455: | learn: 31.7627384 | total: 452ms | remaining: 539ms |
| 456: | learn: 31.7540809 | total: 453ms | remaining: 538ms |
| 457: | learn: 31.7534357 | total: 453ms | remaining: 537ms |
| 458: | learn: 31.7507303 | total: 454ms | remaining: 535ms |
| 459: | learn: 31.7436085 | total: 455ms | remaining: 534ms |
| 460: | learn: 31.7427684 | total: 455ms | remaining: 532ms |
| 461: | learn: 31.7230470 | total: 456ms | remaining: 531ms |
| 462: | learn: 31.7152097 | total: 457ms | remaining: 530ms |
| 463: | learn: 31.7087306 | total: 457ms | remaining: 528ms |
| 464: | learn: 31.6855154 | total: 458ms | remaining: 527ms |
| 465: | learn: 31.6783678 | total: 459ms | remaining: 526ms |
| 466: | learn: 31.6716232 | total: 459ms | remaining: 524ms |
| 467: | learn: 31.6704890 | total: 460ms | remaining: 523ms |

|      |                   |              |                  |
|------|-------------------|--------------|------------------|
| 468: | learn: 31.6625662 | total: 461ms | remaining: 522ms |
| 469: | learn: 31.6527294 | total: 461ms | remaining: 520ms |
| 470: | learn: 31.6373283 | total: 462ms | remaining: 519ms |
| 471: | learn: 31.6362069 | total: 463ms | remaining: 518ms |
| 472: | learn: 31.6327067 | total: 463ms | remaining: 516ms |
| 473: | learn: 31.6203669 | total: 464ms | remaining: 515ms |
| 474: | learn: 31.6012231 | total: 465ms | remaining: 513ms |
| 475: | learn: 31.5925237 | total: 466ms | remaining: 513ms |
| 476: | learn: 31.5866190 | total: 467ms | remaining: 512ms |
| 477: | learn: 31.5822324 | total: 467ms | remaining: 510ms |
| 478: | learn: 31.5810284 | total: 468ms | remaining: 509ms |
| 479: | learn: 31.5740508 | total: 468ms | remaining: 507ms |
| 480: | learn: 31.5675731 | total: 469ms | remaining: 506ms |
| 481: | learn: 31.5597611 | total: 470ms | remaining: 505ms |
| 482: | learn: 31.5463900 | total: 470ms | remaining: 503ms |
| 483: | learn: 31.5433671 | total: 471ms | remaining: 502ms |
| 484: | learn: 31.5403712 | total: 472ms | remaining: 501ms |
| 485: | learn: 31.5256507 | total: 472ms | remaining: 500ms |
| 486: | learn: 31.5133706 | total: 473ms | remaining: 498ms |
| 487: | learn: 31.5126502 | total: 474ms | remaining: 497ms |
| 488: | learn: 31.4987036 | total: 474ms | remaining: 496ms |
| 489: | learn: 31.4975382 | total: 475ms | remaining: 494ms |
| 490: | learn: 31.4886033 | total: 476ms | remaining: 493ms |
| 491: | learn: 31.4717958 | total: 476ms | remaining: 492ms |
| 492: | learn: 31.4626571 | total: 477ms | remaining: 490ms |
| 493: | learn: 31.4487438 | total: 478ms | remaining: 489ms |
| 494: | learn: 31.4436064 | total: 478ms | remaining: 488ms |
| 495: | learn: 31.4354075 | total: 479ms | remaining: 486ms |
| 496: | learn: 31.4292830 | total: 479ms | remaining: 485ms |
| 497: | learn: 31.4191364 | total: 481ms | remaining: 485ms |
| 498: | learn: 31.4031941 | total: 482ms | remaining: 483ms |
| 499: | learn: 31.4001634 | total: 482ms | remaining: 482ms |
| 500: | learn: 31.3940868 | total: 483ms | remaining: 481ms |
| 501: | learn: 31.3883802 | total: 484ms | remaining: 480ms |
| 502: | learn: 31.3850385 | total: 484ms | remaining: 478ms |
| 503: | learn: 31.3676650 | total: 485ms | remaining: 477ms |
| 504: | learn: 31.3497640 | total: 485ms | remaining: 476ms |
| 505: | learn: 31.3490643 | total: 486ms | remaining: 474ms |
| 506: | learn: 31.3461373 | total: 487ms | remaining: 473ms |
| 507: | learn: 31.3379724 | total: 487ms | remaining: 472ms |
| 508: | learn: 31.3305558 | total: 488ms | remaining: 471ms |
| 509: | learn: 31.3296273 | total: 489ms | remaining: 470ms |
| 510: | learn: 31.3262027 | total: 489ms | remaining: 468ms |
| 511: | learn: 31.3188261 | total: 490ms | remaining: 467ms |
| 512: | learn: 31.3025516 | total: 491ms | remaining: 466ms |
| 513: | learn: 31.2960261 | total: 491ms | remaining: 465ms |
| 514: | learn: 31.2906592 | total: 492ms | remaining: 463ms |
| 515: | learn: 31.2751630 | total: 493ms | remaining: 462ms |

|      |                   |              |                  |
|------|-------------------|--------------|------------------|
| 516: | learn: 31.2694233 | total: 493ms | remaining: 461ms |
| 517: | learn: 31.2559623 | total: 494ms | remaining: 460ms |
| 518: | learn: 31.2504548 | total: 495ms | remaining: 459ms |
| 519: | learn: 31.2436264 | total: 496ms | remaining: 458ms |
| 520: | learn: 31.2360840 | total: 497ms | remaining: 457ms |
| 521: | learn: 31.2252480 | total: 497ms | remaining: 455ms |
| 522: | learn: 31.2211608 | total: 498ms | remaining: 454ms |
| 523: | learn: 31.2133323 | total: 499ms | remaining: 453ms |
| 524: | learn: 31.2125169 | total: 499ms | remaining: 452ms |
| 525: | learn: 31.2117517 | total: 500ms | remaining: 451ms |
| 526: | learn: 31.2108124 | total: 501ms | remaining: 449ms |
| 527: | learn: 31.1959284 | total: 501ms | remaining: 448ms |
| 528: | learn: 31.1796697 | total: 502ms | remaining: 447ms |
| 529: | learn: 31.1788437 | total: 503ms | remaining: 446ms |
| 530: | learn: 31.1749743 | total: 503ms | remaining: 445ms |
| 531: | learn: 31.1673313 | total: 504ms | remaining: 443ms |
| 532: | learn: 31.1648527 | total: 505ms | remaining: 442ms |
| 533: | learn: 31.1619015 | total: 505ms | remaining: 441ms |
| 534: | learn: 31.1405656 | total: 506ms | remaining: 440ms |
| 535: | learn: 31.1340233 | total: 506ms | remaining: 438ms |
| 536: | learn: 31.1273424 | total: 507ms | remaining: 437ms |
| 537: | learn: 31.1185528 | total: 508ms | remaining: 436ms |
| 538: | learn: 31.1159886 | total: 509ms | remaining: 435ms |
| 539: | learn: 31.1082120 | total: 509ms | remaining: 434ms |
| 540: | learn: 31.1021901 | total: 510ms | remaining: 433ms |
| 541: | learn: 31.0996299 | total: 511ms | remaining: 432ms |
| 542: | learn: 31.0990707 | total: 512ms | remaining: 431ms |
| 543: | learn: 31.0937406 | total: 512ms | remaining: 429ms |
| 544: | learn: 31.0833817 | total: 513ms | remaining: 428ms |
| 545: | learn: 31.0811164 | total: 514ms | remaining: 427ms |
| 546: | learn: 31.0624968 | total: 515ms | remaining: 426ms |
| 547: | learn: 31.0561949 | total: 515ms | remaining: 425ms |
| 548: | learn: 31.0494397 | total: 517ms | remaining: 425ms |
| 549: | learn: 31.0487215 | total: 518ms | remaining: 424ms |
| 550: | learn: 31.0389144 | total: 518ms | remaining: 422ms |
| 551: | learn: 31.0307675 | total: 519ms | remaining: 421ms |
| 552: | learn: 31.0252868 | total: 520ms | remaining: 420ms |
| 553: | learn: 31.0160126 | total: 520ms | remaining: 419ms |
| 554: | learn: 31.0102904 | total: 521ms | remaining: 418ms |
| 555: | learn: 30.9943429 | total: 522ms | remaining: 417ms |
| 556: | learn: 30.9846901 | total: 522ms | remaining: 416ms |
| 557: | learn: 30.9839846 | total: 523ms | remaining: 414ms |
| 558: | learn: 30.9642004 | total: 524ms | remaining: 413ms |
| 559: | learn: 30.9635143 | total: 525ms | remaining: 412ms |
| 560: | learn: 30.9594520 | total: 526ms | remaining: 411ms |
| 561: | learn: 30.9569165 | total: 527ms | remaining: 410ms |
| 562: | learn: 30.9437580 | total: 528ms | remaining: 409ms |
| 563: | learn: 30.9368959 | total: 528ms | remaining: 408ms |

|      |                   |              |                  |
|------|-------------------|--------------|------------------|
| 564: | learn: 30.9238391 | total: 529ms | remaining: 407ms |
| 565: | learn: 30.9186508 | total: 530ms | remaining: 406ms |
| 566: | learn: 30.9144196 | total: 531ms | remaining: 405ms |
| 567: | learn: 30.9100307 | total: 531ms | remaining: 404ms |
| 568: | learn: 30.9069771 | total: 532ms | remaining: 403ms |
| 569: | learn: 30.8963773 | total: 533ms | remaining: 402ms |
| 570: | learn: 30.8933898 | total: 533ms | remaining: 401ms |
| 571: | learn: 30.8910268 | total: 534ms | remaining: 400ms |
| 572: | learn: 30.8904750 | total: 535ms | remaining: 398ms |
| 573: | learn: 30.8827812 | total: 535ms | remaining: 397ms |
| 574: | learn: 30.8753474 | total: 536ms | remaining: 396ms |
| 575: | learn: 30.8684652 | total: 536ms | remaining: 395ms |
| 576: | learn: 30.8545126 | total: 537ms | remaining: 394ms |
| 577: | learn: 30.8535974 | total: 538ms | remaining: 393ms |
| 578: | learn: 30.8455631 | total: 538ms | remaining: 391ms |
| 579: | learn: 30.8432862 | total: 539ms | remaining: 390ms |
| 580: | learn: 30.8363853 | total: 540ms | remaining: 389ms |
| 581: | learn: 30.8322757 | total: 542ms | remaining: 389ms |
| 582: | learn: 30.8220638 | total: 542ms | remaining: 388ms |
| 583: | learn: 30.8166345 | total: 545ms | remaining: 388ms |
| 584: | learn: 30.8128003 | total: 546ms | remaining: 387ms |
| 585: | learn: 30.8083331 | total: 547ms | remaining: 386ms |
| 586: | learn: 30.8062543 | total: 547ms | remaining: 385ms |
| 587: | learn: 30.7964297 | total: 548ms | remaining: 384ms |
| 588: | learn: 30.7956953 | total: 549ms | remaining: 383ms |
| 589: | learn: 30.7951213 | total: 549ms | remaining: 382ms |
| 590: | learn: 30.7839856 | total: 550ms | remaining: 381ms |
| 591: | learn: 30.7705617 | total: 550ms | remaining: 379ms |
| 592: | learn: 30.7619271 | total: 551ms | remaining: 378ms |
| 593: | learn: 30.7486276 | total: 552ms | remaining: 377ms |
| 594: | learn: 30.7446523 | total: 552ms | remaining: 376ms |
| 595: | learn: 30.7426349 | total: 553ms | remaining: 375ms |
| 596: | learn: 30.7402335 | total: 554ms | remaining: 374ms |
| 597: | learn: 30.7346447 | total: 556ms | remaining: 374ms |
| 598: | learn: 30.7255246 | total: 557ms | remaining: 373ms |
| 599: | learn: 30.7199882 | total: 558ms | remaining: 372ms |
| 600: | learn: 30.7107365 | total: 559ms | remaining: 371ms |
| 601: | learn: 30.6977398 | total: 559ms | remaining: 370ms |
| 602: | learn: 30.6835200 | total: 560ms | remaining: 369ms |
| 603: | learn: 30.6790411 | total: 561ms | remaining: 368ms |
| 604: | learn: 30.6666611 | total: 562ms | remaining: 367ms |
| 605: | learn: 30.6660343 | total: 562ms | remaining: 366ms |
| 606: | learn: 30.6551163 | total: 563ms | remaining: 365ms |
| 607: | learn: 30.6455270 | total: 564ms | remaining: 363ms |
| 608: | learn: 30.6449089 | total: 564ms | remaining: 362ms |
| 609: | learn: 30.6402566 | total: 565ms | remaining: 361ms |
| 610: | learn: 30.6307777 | total: 566ms | remaining: 360ms |
| 611: | learn: 30.6242850 | total: 566ms | remaining: 359ms |

|      |                   |              |                  |
|------|-------------------|--------------|------------------|
| 612: | learn: 30.6171289 | total: 567ms | remaining: 358ms |
| 613: | learn: 30.6163715 | total: 567ms | remaining: 357ms |
| 614: | learn: 30.6118341 | total: 568ms | remaining: 356ms |
| 615: | learn: 30.6090967 | total: 569ms | remaining: 355ms |
| 616: | learn: 30.6007043 | total: 570ms | remaining: 354ms |
| 617: | learn: 30.5999417 | total: 571ms | remaining: 353ms |
| 618: | learn: 30.5941330 | total: 572ms | remaining: 352ms |
| 619: | learn: 30.5886278 | total: 573ms | remaining: 351ms |
| 620: | learn: 30.5879630 | total: 573ms | remaining: 350ms |
| 621: | learn: 30.5763047 | total: 574ms | remaining: 349ms |
| 622: | learn: 30.5752331 | total: 575ms | remaining: 348ms |
| 623: | learn: 30.5682454 | total: 576ms | remaining: 347ms |
| 624: | learn: 30.5564373 | total: 577ms | remaining: 346ms |
| 625: | learn: 30.5484216 | total: 577ms | remaining: 345ms |
| 626: | learn: 30.5419940 | total: 578ms | remaining: 344ms |
| 627: | learn: 30.5414165 | total: 579ms | remaining: 343ms |
| 628: | learn: 30.5360536 | total: 579ms | remaining: 342ms |
| 629: | learn: 30.5228873 | total: 580ms | remaining: 341ms |
| 630: | learn: 30.5221904 | total: 581ms | remaining: 340ms |
| 631: | learn: 30.5215821 | total: 581ms | remaining: 338ms |
| 632: | learn: 30.5170733 | total: 582ms | remaining: 337ms |
| 633: | learn: 30.5164029 | total: 583ms | remaining: 336ms |
| 634: | learn: 30.5070856 | total: 583ms | remaining: 335ms |
| 635: | learn: 30.4973895 | total: 584ms | remaining: 334ms |
| 636: | learn: 30.4901263 | total: 585ms | remaining: 333ms |
| 637: | learn: 30.4836556 | total: 586ms | remaining: 333ms |
| 638: | learn: 30.4749959 | total: 587ms | remaining: 332ms |
| 639: | learn: 30.4670514 | total: 587ms | remaining: 330ms |
| 640: | learn: 30.4571367 | total: 588ms | remaining: 329ms |
| 641: | learn: 30.4544749 | total: 589ms | remaining: 328ms |
| 642: | learn: 30.4434978 | total: 589ms | remaining: 327ms |
| 643: | learn: 30.4366754 | total: 590ms | remaining: 326ms |
| 644: | learn: 30.4250438 | total: 591ms | remaining: 325ms |
| 645: | learn: 30.4189717 | total: 591ms | remaining: 324ms |
| 646: | learn: 30.4060278 | total: 592ms | remaining: 323ms |
| 647: | learn: 30.4024530 | total: 593ms | remaining: 322ms |
| 648: | learn: 30.3879536 | total: 593ms | remaining: 321ms |
| 649: | learn: 30.3804499 | total: 594ms | remaining: 320ms |
| 650: | learn: 30.3728179 | total: 595ms | remaining: 319ms |
| 651: | learn: 30.3678193 | total: 595ms | remaining: 318ms |
| 652: | learn: 30.3555077 | total: 596ms | remaining: 317ms |
| 653: | learn: 30.3505536 | total: 597ms | remaining: 316ms |
| 654: | learn: 30.3462501 | total: 598ms | remaining: 315ms |
| 655: | learn: 30.3396822 | total: 598ms | remaining: 314ms |
| 656: | learn: 30.3285934 | total: 599ms | remaining: 313ms |
| 657: | learn: 30.3281600 | total: 600ms | remaining: 312ms |
| 658: | learn: 30.3231630 | total: 601ms | remaining: 311ms |
| 659: | learn: 30.3192258 | total: 601ms | remaining: 310ms |

|      |                   |              |                  |
|------|-------------------|--------------|------------------|
| 660: | learn: 30.3185441 | total: 602ms | remaining: 309ms |
| 661: | learn: 30.3152575 | total: 602ms | remaining: 308ms |
| 662: | learn: 30.3075237 | total: 603ms | remaining: 307ms |
| 663: | learn: 30.3044506 | total: 604ms | remaining: 306ms |
| 664: | learn: 30.2977360 | total: 605ms | remaining: 305ms |
| 665: | learn: 30.2968135 | total: 606ms | remaining: 304ms |
| 666: | learn: 30.2950982 | total: 606ms | remaining: 303ms |
| 667: | learn: 30.2945558 | total: 607ms | remaining: 302ms |
| 668: | learn: 30.2904528 | total: 608ms | remaining: 301ms |
| 669: | learn: 30.2828668 | total: 608ms | remaining: 300ms |
| 670: | learn: 30.2809145 | total: 609ms | remaining: 299ms |
| 671: | learn: 30.2729764 | total: 610ms | remaining: 298ms |
| 672: | learn: 30.2619328 | total: 610ms | remaining: 297ms |
| 673: | learn: 30.2443134 | total: 611ms | remaining: 295ms |
| 674: | learn: 30.2329945 | total: 612ms | remaining: 294ms |
| 675: | learn: 30.2196779 | total: 612ms | remaining: 293ms |
| 676: | learn: 30.2190786 | total: 613ms | remaining: 293ms |
| 677: | learn: 30.2142656 | total: 614ms | remaining: 292ms |
| 678: | learn: 30.2061330 | total: 615ms | remaining: 291ms |
| 679: | learn: 30.2008578 | total: 615ms | remaining: 290ms |
| 680: | learn: 30.1949765 | total: 616ms | remaining: 289ms |
| 681: | learn: 30.1903065 | total: 617ms | remaining: 288ms |
| 682: | learn: 30.1839230 | total: 617ms | remaining: 286ms |
| 683: | learn: 30.1833967 | total: 618ms | remaining: 285ms |
| 684: | learn: 30.1829894 | total: 619ms | remaining: 284ms |
| 685: | learn: 30.1778227 | total: 621ms | remaining: 284ms |
| 686: | learn: 30.1755435 | total: 625ms | remaining: 285ms |
| 687: | learn: 30.1727993 | total: 626ms | remaining: 284ms |
| 688: | learn: 30.1666773 | total: 626ms | remaining: 283ms |
| 689: | learn: 30.1608323 | total: 627ms | remaining: 282ms |
| 690: | learn: 30.1602154 | total: 628ms | remaining: 281ms |
| 691: | learn: 30.1566263 | total: 629ms | remaining: 280ms |
| 692: | learn: 30.1545179 | total: 630ms | remaining: 279ms |
| 693: | learn: 30.1539634 | total: 630ms | remaining: 278ms |
| 694: | learn: 30.1453341 | total: 631ms | remaining: 277ms |
| 695: | learn: 30.1324197 | total: 632ms | remaining: 276ms |
| 696: | learn: 30.1220193 | total: 632ms | remaining: 275ms |
| 697: | learn: 30.1109436 | total: 633ms | remaining: 274ms |
| 698: | learn: 30.1105167 | total: 633ms | remaining: 273ms |
| 699: | learn: 30.1042708 | total: 634ms | remaining: 272ms |
| 700: | learn: 30.1032092 | total: 635ms | remaining: 271ms |
| 701: | learn: 30.0996011 | total: 635ms | remaining: 270ms |
| 702: | learn: 30.0977212 | total: 636ms | remaining: 269ms |
| 703: | learn: 30.0882263 | total: 637ms | remaining: 268ms |
| 704: | learn: 30.0750473 | total: 638ms | remaining: 267ms |
| 705: | learn: 30.0746335 | total: 639ms | remaining: 266ms |
| 706: | learn: 30.0738209 | total: 640ms | remaining: 265ms |
| 707: | learn: 30.0734284 | total: 640ms | remaining: 264ms |

|      |                   |              |                  |
|------|-------------------|--------------|------------------|
| 708: | learn: 30.0704477 | total: 641ms | remaining: 263ms |
| 709: | learn: 30.0652820 | total: 642ms | remaining: 262ms |
| 710: | learn: 30.0461799 | total: 643ms | remaining: 261ms |
| 711: | learn: 30.0405570 | total: 644ms | remaining: 260ms |
| 712: | learn: 30.0254070 | total: 644ms | remaining: 259ms |
| 713: | learn: 30.0234675 | total: 645ms | remaining: 258ms |
| 714: | learn: 30.0110661 | total: 646ms | remaining: 257ms |
| 715: | learn: 30.0051871 | total: 647ms | remaining: 256ms |
| 716: | learn: 29.9959345 | total: 647ms | remaining: 255ms |
| 717: | learn: 29.9880125 | total: 648ms | remaining: 254ms |
| 718: | learn: 29.9874172 | total: 649ms | remaining: 254ms |
| 719: | learn: 29.9726204 | total: 649ms | remaining: 253ms |
| 720: | learn: 29.9564285 | total: 650ms | remaining: 252ms |
| 721: | learn: 29.9470388 | total: 651ms | remaining: 251ms |
| 722: | learn: 29.9408631 | total: 652ms | remaining: 250ms |
| 723: | learn: 29.9334956 | total: 652ms | remaining: 249ms |
| 724: | learn: 29.9330646 | total: 653ms | remaining: 248ms |
| 725: | learn: 29.9308467 | total: 654ms | remaining: 247ms |
| 726: | learn: 29.9258569 | total: 654ms | remaining: 246ms |
| 727: | learn: 29.9241754 | total: 655ms | remaining: 245ms |
| 728: | learn: 29.9237341 | total: 655ms | remaining: 244ms |
| 729: | learn: 29.9137408 | total: 657ms | remaining: 243ms |
| 730: | learn: 29.9121747 | total: 658ms | remaining: 242ms |
| 731: | learn: 29.9014207 | total: 659ms | remaining: 241ms |
| 732: | learn: 29.9010966 | total: 659ms | remaining: 240ms |
| 733: | learn: 29.8972024 | total: 660ms | remaining: 239ms |
| 734: | learn: 29.8967369 | total: 660ms | remaining: 238ms |
| 735: | learn: 29.8864502 | total: 661ms | remaining: 237ms |
| 736: | learn: 29.8764089 | total: 662ms | remaining: 236ms |
| 737: | learn: 29.8694082 | total: 663ms | remaining: 235ms |
| 738: | learn: 29.8690577 | total: 663ms | remaining: 234ms |
| 739: | learn: 29.8655654 | total: 664ms | remaining: 233ms |
| 740: | learn: 29.8626300 | total: 665ms | remaining: 232ms |
| 741: | learn: 29.8542606 | total: 666ms | remaining: 231ms |
| 742: | learn: 29.8454425 | total: 666ms | remaining: 230ms |
| 743: | learn: 29.8385342 | total: 667ms | remaining: 230ms |
| 744: | learn: 29.8347387 | total: 668ms | remaining: 229ms |
| 745: | learn: 29.8325951 | total: 668ms | remaining: 228ms |
| 746: | learn: 29.8240556 | total: 669ms | remaining: 227ms |
| 747: | learn: 29.8166013 | total: 670ms | remaining: 226ms |
| 748: | learn: 29.8080965 | total: 671ms | remaining: 225ms |
| 749: | learn: 29.8077585 | total: 672ms | remaining: 224ms |
| 750: | learn: 29.8018574 | total: 672ms | remaining: 223ms |
| 751: | learn: 29.7999858 | total: 673ms | remaining: 222ms |
| 752: | learn: 29.7929920 | total: 674ms | remaining: 221ms |
| 753: | learn: 29.7913190 | total: 674ms | remaining: 220ms |
| 754: | learn: 29.7872900 | total: 675ms | remaining: 219ms |
| 755: | learn: 29.7751692 | total: 676ms | remaining: 218ms |

|      |                   |              |                  |
|------|-------------------|--------------|------------------|
| 756: | learn: 29.7688203 | total: 676ms | remaining: 217ms |
| 757: | learn: 29.7625638 | total: 677ms | remaining: 216ms |
| 758: | learn: 29.7622738 | total: 678ms | remaining: 215ms |
| 759: | learn: 29.7619788 | total: 679ms | remaining: 214ms |
| 760: | learn: 29.7570563 | total: 679ms | remaining: 213ms |
| 761: | learn: 29.7566588 | total: 680ms | remaining: 212ms |
| 762: | learn: 29.7546674 | total: 681ms | remaining: 211ms |
| 763: | learn: 29.7469837 | total: 681ms | remaining: 210ms |
| 764: | learn: 29.7422366 | total: 682ms | remaining: 210ms |
| 765: | learn: 29.7401067 | total: 683ms | remaining: 209ms |
| 766: | learn: 29.7371594 | total: 683ms | remaining: 208ms |
| 767: | learn: 29.7368295 | total: 684ms | remaining: 207ms |
| 768: | learn: 29.7274163 | total: 685ms | remaining: 206ms |
| 769: | learn: 29.7226792 | total: 686ms | remaining: 205ms |
| 770: | learn: 29.7159945 | total: 687ms | remaining: 204ms |
| 771: | learn: 29.7103335 | total: 687ms | remaining: 203ms |
| 772: | learn: 29.7087465 | total: 688ms | remaining: 202ms |
| 773: | learn: 29.7012991 | total: 689ms | remaining: 201ms |
| 774: | learn: 29.6884189 | total: 689ms | remaining: 200ms |
| 775: | learn: 29.6824233 | total: 690ms | remaining: 199ms |
| 776: | learn: 29.6736351 | total: 691ms | remaining: 198ms |
| 777: | learn: 29.6648839 | total: 692ms | remaining: 198ms |
| 778: | learn: 29.6645111 | total: 693ms | remaining: 197ms |
| 779: | learn: 29.6585536 | total: 694ms | remaining: 196ms |
| 780: | learn: 29.6519661 | total: 695ms | remaining: 195ms |
| 781: | learn: 29.6463890 | total: 696ms | remaining: 194ms |
| 782: | learn: 29.6440115 | total: 696ms | remaining: 193ms |
| 783: | learn: 29.6435707 | total: 697ms | remaining: 192ms |
| 784: | learn: 29.6416914 | total: 698ms | remaining: 191ms |
| 785: | learn: 29.6345973 | total: 700ms | remaining: 190ms |
| 786: | learn: 29.6336766 | total: 701ms | remaining: 190ms |
| 787: | learn: 29.6333199 | total: 702ms | remaining: 189ms |
| 788: | learn: 29.6291840 | total: 703ms | remaining: 188ms |
| 789: | learn: 29.6275285 | total: 704ms | remaining: 187ms |
| 790: | learn: 29.6221421 | total: 704ms | remaining: 186ms |
| 791: | learn: 29.6176107 | total: 705ms | remaining: 185ms |
| 792: | learn: 29.6130017 | total: 707ms | remaining: 184ms |
| 793: | learn: 29.6058675 | total: 707ms | remaining: 184ms |
| 794: | learn: 29.6033799 | total: 708ms | remaining: 183ms |
| 795: | learn: 29.6000427 | total: 709ms | remaining: 182ms |
| 796: | learn: 29.5936376 | total: 710ms | remaining: 181ms |
| 797: | learn: 29.5844631 | total: 710ms | remaining: 180ms |
| 798: | learn: 29.5808165 | total: 711ms | remaining: 179ms |
| 799: | learn: 29.5736358 | total: 711ms | remaining: 178ms |
| 800: | learn: 29.5707852 | total: 712ms | remaining: 177ms |
| 801: | learn: 29.5651716 | total: 714ms | remaining: 176ms |
| 802: | learn: 29.5581160 | total: 715ms | remaining: 175ms |
| 803: | learn: 29.5469894 | total: 715ms | remaining: 174ms |

|      |                   |              |                  |
|------|-------------------|--------------|------------------|
| 804: | learn: 29.5396366 | total: 716ms | remaining: 173ms |
| 805: | learn: 29.5392799 | total: 717ms | remaining: 173ms |
| 806: | learn: 29.5379857 | total: 717ms | remaining: 172ms |
| 807: | learn: 29.5344013 | total: 718ms | remaining: 171ms |
| 808: | learn: 29.5267967 | total: 719ms | remaining: 170ms |
| 809: | learn: 29.5131429 | total: 719ms | remaining: 169ms |
| 810: | learn: 29.5092255 | total: 720ms | remaining: 168ms |
| 811: | learn: 29.5007749 | total: 721ms | remaining: 167ms |
| 812: | learn: 29.4950492 | total: 722ms | remaining: 166ms |
| 813: | learn: 29.4946591 | total: 722ms | remaining: 165ms |
| 814: | learn: 29.4860444 | total: 723ms | remaining: 164ms |
| 815: | learn: 29.4836629 | total: 724ms | remaining: 163ms |
| 816: | learn: 29.4797137 | total: 724ms | remaining: 162ms |
| 817: | learn: 29.4794477 | total: 725ms | remaining: 161ms |
| 818: | learn: 29.4660008 | total: 726ms | remaining: 160ms |
| 819: | learn: 29.4636431 | total: 726ms | remaining: 159ms |
| 820: | learn: 29.4583733 | total: 727ms | remaining: 159ms |
| 821: | learn: 29.4512228 | total: 729ms | remaining: 158ms |
| 822: | learn: 29.4508776 | total: 730ms | remaining: 157ms |
| 823: | learn: 29.4454820 | total: 731ms | remaining: 156ms |
| 824: | learn: 29.4441071 | total: 731ms | remaining: 155ms |
| 825: | learn: 29.4437952 | total: 732ms | remaining: 154ms |
| 826: | learn: 29.4411288 | total: 733ms | remaining: 153ms |
| 827: | learn: 29.4387139 | total: 733ms | remaining: 152ms |
| 828: | learn: 29.4384632 | total: 734ms | remaining: 151ms |
| 829: | learn: 29.4308408 | total: 735ms | remaining: 151ms |
| 830: | learn: 29.4262173 | total: 736ms | remaining: 150ms |
| 831: | learn: 29.4185709 | total: 737ms | remaining: 149ms |
| 832: | learn: 29.4122712 | total: 738ms | remaining: 148ms |
| 833: | learn: 29.4070058 | total: 739ms | remaining: 147ms |
| 834: | learn: 29.4044579 | total: 740ms | remaining: 146ms |
| 835: | learn: 29.4029685 | total: 741ms | remaining: 145ms |
| 836: | learn: 29.4023145 | total: 741ms | remaining: 144ms |
| 837: | learn: 29.4017642 | total: 743ms | remaining: 144ms |
| 838: | learn: 29.3962563 | total: 744ms | remaining: 143ms |
| 839: | learn: 29.3958790 | total: 744ms | remaining: 142ms |
| 840: | learn: 29.3929130 | total: 746ms | remaining: 141ms |
| 841: | learn: 29.3926177 | total: 746ms | remaining: 140ms |
| 842: | learn: 29.3912022 | total: 747ms | remaining: 139ms |
| 843: | learn: 29.3909063 | total: 748ms | remaining: 138ms |
| 844: | learn: 29.3812683 | total: 749ms | remaining: 137ms |
| 845: | learn: 29.3730233 | total: 750ms | remaining: 136ms |
| 846: | learn: 29.3696314 | total: 750ms | remaining: 136ms |
| 847: | learn: 29.3662492 | total: 751ms | remaining: 135ms |
| 848: | learn: 29.3659739 | total: 752ms | remaining: 134ms |
| 849: | learn: 29.3599637 | total: 753ms | remaining: 133ms |
| 850: | learn: 29.3536131 | total: 754ms | remaining: 132ms |
| 851: | learn: 29.3516513 | total: 754ms | remaining: 131ms |

|      |                   |              |                   |
|------|-------------------|--------------|-------------------|
| 852: | learn: 29.3503230 | total: 755ms | remaining: 130ms  |
| 853: | learn: 29.3417926 | total: 755ms | remaining: 129ms  |
| 854: | learn: 29.3351595 | total: 756ms | remaining: 128ms  |
| 855: | learn: 29.3234317 | total: 758ms | remaining: 127ms  |
| 856: | learn: 29.3162281 | total: 758ms | remaining: 127ms  |
| 857: | learn: 29.3159358 | total: 759ms | remaining: 126ms  |
| 858: | learn: 29.3098989 | total: 760ms | remaining: 125ms  |
| 859: | learn: 29.3072534 | total: 760ms | remaining: 124ms  |
| 860: | learn: 29.2988762 | total: 761ms | remaining: 123ms  |
| 861: | learn: 29.2954456 | total: 762ms | remaining: 122ms  |
| 862: | learn: 29.2918159 | total: 762ms | remaining: 121ms  |
| 863: | learn: 29.2875076 | total: 763ms | remaining: 120ms  |
| 864: | learn: 29.2872870 | total: 764ms | remaining: 119ms  |
| 865: | learn: 29.2830256 | total: 765ms | remaining: 118ms  |
| 866: | learn: 29.2688239 | total: 766ms | remaining: 117ms  |
| 867: | learn: 29.2631554 | total: 766ms | remaining: 117ms  |
| 868: | learn: 29.2584973 | total: 767ms | remaining: 116ms  |
| 869: | learn: 29.2531837 | total: 768ms | remaining: 115ms  |
| 870: | learn: 29.2434788 | total: 769ms | remaining: 114ms  |
| 871: | learn: 29.2432026 | total: 769ms | remaining: 113ms  |
| 872: | learn: 29.2399178 | total: 770ms | remaining: 112ms  |
| 873: | learn: 29.2396324 | total: 771ms | remaining: 111ms  |
| 874: | learn: 29.2298932 | total: 772ms | remaining: 110ms  |
| 875: | learn: 29.2274802 | total: 773ms | remaining: 109ms  |
| 876: | learn: 29.2200966 | total: 774ms | remaining: 109ms  |
| 877: | learn: 29.2133286 | total: 775ms | remaining: 108ms  |
| 878: | learn: 29.2103404 | total: 776ms | remaining: 107ms  |
| 879: | learn: 29.2058285 | total: 776ms | remaining: 106ms  |
| 880: | learn: 29.2055522 | total: 777ms | remaining: 105ms  |
| 881: | learn: 29.2052187 | total: 777ms | remaining: 104ms  |
| 882: | learn: 29.2033959 | total: 779ms | remaining: 103ms  |
| 883: | learn: 29.2031697 | total: 781ms | remaining: 102ms  |
| 884: | learn: 29.1976684 | total: 782ms | remaining: 102ms  |
| 885: | learn: 29.1923492 | total: 783ms | remaining: 101ms  |
| 886: | learn: 29.1904315 | total: 784ms | remaining: 99.9ms |
| 887: | learn: 29.1901805 | total: 785ms | remaining: 99ms   |
| 888: | learn: 29.1859028 | total: 787ms | remaining: 98.3ms |
| 889: | learn: 29.1856470 | total: 788ms | remaining: 97.4ms |
| 890: | learn: 29.1838710 | total: 789ms | remaining: 96.5ms |
| 891: | learn: 29.1751318 | total: 790ms | remaining: 95.6ms |
| 892: | learn: 29.1686719 | total: 791ms | remaining: 94.8ms |
| 893: | learn: 29.1684337 | total: 791ms | remaining: 93.8ms |
| 894: | learn: 29.1624438 | total: 792ms | remaining: 92.9ms |
| 895: | learn: 29.1593074 | total: 793ms | remaining: 92.1ms |
| 896: | learn: 29.1528002 | total: 794ms | remaining: 91.2ms |
| 897: | learn: 29.1451051 | total: 795ms | remaining: 90.3ms |
| 898: | learn: 29.1397448 | total: 796ms | remaining: 89.4ms |
| 899: | learn: 29.1320899 | total: 796ms | remaining: 88.5ms |

|      |                   |              |                   |
|------|-------------------|--------------|-------------------|
| 900: | learn: 29.1318492 | total: 797ms | remaining: 87.6ms |
| 901: | learn: 29.1176778 | total: 798ms | remaining: 86.7ms |
| 902: | learn: 29.1151552 | total: 798ms | remaining: 85.8ms |
| 903: | learn: 29.1027065 | total: 799ms | remaining: 84.9ms |
| 904: | learn: 29.0957829 | total: 800ms | remaining: 84ms   |
| 905: | learn: 29.0946693 | total: 800ms | remaining: 83ms   |
| 906: | learn: 29.0943929 | total: 801ms | remaining: 82.1ms |
| 907: | learn: 29.0896526 | total: 802ms | remaining: 81.2ms |
| 908: | learn: 29.0894274 | total: 803ms | remaining: 80.3ms |
| 909: | learn: 29.0884749 | total: 803ms | remaining: 79.4ms |
| 910: | learn: 29.0809483 | total: 804ms | remaining: 78.5ms |
| 911: | learn: 29.0705627 | total: 805ms | remaining: 77.6ms |
| 912: | learn: 29.0680679 | total: 805ms | remaining: 76.7ms |
| 913: | learn: 29.0598676 | total: 806ms | remaining: 75.8ms |
| 914: | learn: 29.0549554 | total: 807ms | remaining: 75ms   |
| 915: | learn: 29.0425893 | total: 808ms | remaining: 74.1ms |
| 916: | learn: 29.0378088 | total: 808ms | remaining: 73.2ms |
| 917: | learn: 29.0375628 | total: 809ms | remaining: 72.3ms |
| 918: | learn: 29.0332618 | total: 810ms | remaining: 71.4ms |
| 919: | learn: 29.0273382 | total: 810ms | remaining: 70.5ms |
| 920: | learn: 29.0270460 | total: 811ms | remaining: 69.5ms |
| 921: | learn: 29.0254086 | total: 811ms | remaining: 68.6ms |
| 922: | learn: 29.0144747 | total: 812ms | remaining: 67.7ms |
| 923: | learn: 29.0118555 | total: 813ms | remaining: 66.8ms |
| 924: | learn: 29.0031270 | total: 813ms | remaining: 65.9ms |
| 925: | learn: 29.0020587 | total: 814ms | remaining: 65ms   |
| 926: | learn: 28.9990610 | total: 815ms | remaining: 64.2ms |
| 927: | learn: 28.9988392 | total: 816ms | remaining: 63.3ms |
| 928: | learn: 28.9951576 | total: 816ms | remaining: 62.4ms |
| 929: | learn: 28.9949681 | total: 817ms | remaining: 61.5ms |
| 930: | learn: 28.9860656 | total: 818ms | remaining: 60.6ms |
| 931: | learn: 28.9818758 | total: 818ms | remaining: 59.7ms |
| 932: | learn: 28.9776155 | total: 819ms | remaining: 58.8ms |
| 933: | learn: 28.9732002 | total: 819ms | remaining: 57.9ms |
| 934: | learn: 28.9656106 | total: 821ms | remaining: 57.1ms |
| 935: | learn: 28.9547222 | total: 822ms | remaining: 56.2ms |
| 936: | learn: 28.9483464 | total: 823ms | remaining: 55.3ms |
| 937: | learn: 28.9466855 | total: 823ms | remaining: 54.4ms |
| 938: | learn: 28.9407394 | total: 824ms | remaining: 53.5ms |
| 939: | learn: 28.9404784 | total: 825ms | remaining: 52.6ms |
| 940: | learn: 28.9318336 | total: 825ms | remaining: 51.7ms |
| 941: | learn: 28.9221611 | total: 826ms | remaining: 50.8ms |
| 942: | learn: 28.9184276 | total: 826ms | remaining: 50ms   |
| 943: | learn: 28.9108704 | total: 828ms | remaining: 49.1ms |
| 944: | learn: 28.9056695 | total: 829ms | remaining: 48.3ms |
| 945: | learn: 28.8980163 | total: 830ms | remaining: 47.4ms |
| 946: | learn: 28.8977151 | total: 831ms | remaining: 46.5ms |
| 947: | learn: 28.8898308 | total: 833ms | remaining: 45.7ms |

|      |                   |              |                   |
|------|-------------------|--------------|-------------------|
| 948: | learn: 28.8861961 | total: 834ms | remaining: 44.8ms |
| 949: | learn: 28.8812907 | total: 835ms | remaining: 43.9ms |
| 950: | learn: 28.8791039 | total: 836ms | remaining: 43.1ms |
| 951: | learn: 28.8706410 | total: 837ms | remaining: 42.2ms |
| 952: | learn: 28.8672401 | total: 837ms | remaining: 41.3ms |
| 953: | learn: 28.8546844 | total: 838ms | remaining: 40.4ms |
| 954: | learn: 28.8432139 | total: 839ms | remaining: 39.5ms |
| 955: | learn: 28.8391323 | total: 839ms | remaining: 38.6ms |
| 956: | learn: 28.8336290 | total: 840ms | remaining: 37.7ms |
| 957: | learn: 28.8306740 | total: 841ms | remaining: 36.8ms |
| 958: | learn: 28.8222114 | total: 841ms | remaining: 36ms   |
| 959: | learn: 28.8159360 | total: 842ms | remaining: 35.1ms |
| 960: | learn: 28.8135757 | total: 842ms | remaining: 34.2ms |
| 961: | learn: 28.8039269 | total: 844ms | remaining: 33.3ms |
| 962: | learn: 28.7966395 | total: 845ms | remaining: 32.4ms |
| 963: | learn: 28.7860954 | total: 845ms | remaining: 31.6ms |
| 964: | learn: 28.7804569 | total: 846ms | remaining: 30.7ms |
| 965: | learn: 28.7752447 | total: 847ms | remaining: 29.8ms |
| 966: | learn: 28.7733999 | total: 847ms | remaining: 28.9ms |
| 967: | learn: 28.7707185 | total: 848ms | remaining: 28ms   |
| 968: | learn: 28.7641518 | total: 849ms | remaining: 27.2ms |
| 969: | learn: 28.7591029 | total: 850ms | remaining: 26.3ms |
| 970: | learn: 28.7549542 | total: 850ms | remaining: 25.4ms |
| 971: | learn: 28.7520298 | total: 851ms | remaining: 24.5ms |
| 972: | learn: 28.7487635 | total: 852ms | remaining: 23.6ms |
| 973: | learn: 28.7420788 | total: 852ms | remaining: 22.8ms |
| 974: | learn: 28.7346196 | total: 853ms | remaining: 21.9ms |
| 975: | learn: 28.7343970 | total: 854ms | remaining: 21ms   |
| 976: | learn: 28.7336077 | total: 855ms | remaining: 20.1ms |
| 977: | learn: 28.7332974 | total: 856ms | remaining: 19.3ms |
| 978: | learn: 28.7161436 | total: 857ms | remaining: 18.4ms |
| 979: | learn: 28.7131120 | total: 858ms | remaining: 17.5ms |
| 980: | learn: 28.7079595 | total: 859ms | remaining: 16.6ms |
| 981: | learn: 28.7068893 | total: 860ms | remaining: 15.8ms |
| 982: | learn: 28.7018972 | total: 860ms | remaining: 14.9ms |
| 983: | learn: 28.7009314 | total: 861ms | remaining: 14ms   |
| 984: | learn: 28.6859964 | total: 862ms | remaining: 13.1ms |
| 985: | learn: 28.6803473 | total: 862ms | remaining: 12.2ms |
| 986: | learn: 28.6801183 | total: 864ms | remaining: 11.4ms |
| 987: | learn: 28.6709358 | total: 865ms | remaining: 10.5ms |
| 988: | learn: 28.6641891 | total: 866ms | remaining: 9.63ms |
| 989: | learn: 28.6639965 | total: 867ms | remaining: 8.76ms |
| 990: | learn: 28.6610188 | total: 868ms | remaining: 7.88ms |
| 991: | learn: 28.6565632 | total: 869ms | remaining: 7.01ms |
| 992: | learn: 28.6511666 | total: 871ms | remaining: 6.14ms |
| 993: | learn: 28.6433570 | total: 872ms | remaining: 5.26ms |
| 994: | learn: 28.6396976 | total: 873ms | remaining: 4.39ms |
| 995: | learn: 28.6394097 | total: 875ms | remaining: 3.51ms |

```
996: learn: 28.6355459      total: 876ms    remaining: 2.63ms
997: learn: 28.6353033      total: 879ms    remaining: 1.76ms
998: learn: 28.6323333      total: 883ms    remaining: 884us
999: learn: 28.6273099      total: 887ms    remaining: 0us
```

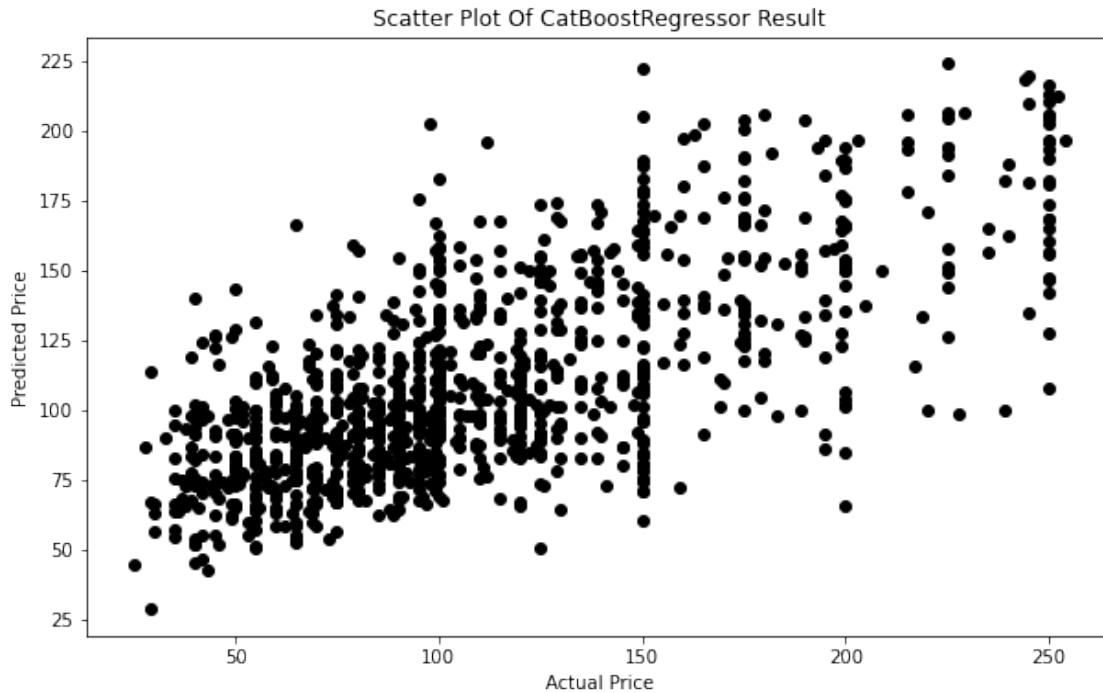
### Cat Boost Regressor Visuals

```
[30]: print(f"R-squared value for train data: {cat_model_train}")
print(f"R-squared value for test data: {cat_model_test}")
print(f"Average Missed Distance From Price: {mean_squared_error(y_test, y_pred_test, squared = False)}")
```

```
R-squared value for train data: 0.6911675671039983
R-squared value for test data: 0.4924641834108424
Average Missed Distance From Price: 36.71564130927711
```

```
[31]: plt.rcParams['figure.figsize'] = [10, 6]
```

```
plt.scatter(y_test, y_pred_test, color = 'black')
plt.title('Scatter Plot Of CatBoostRegressor Result')
plt.xlabel('Actual Price')
plt.ylabel('Predicted Price');
```



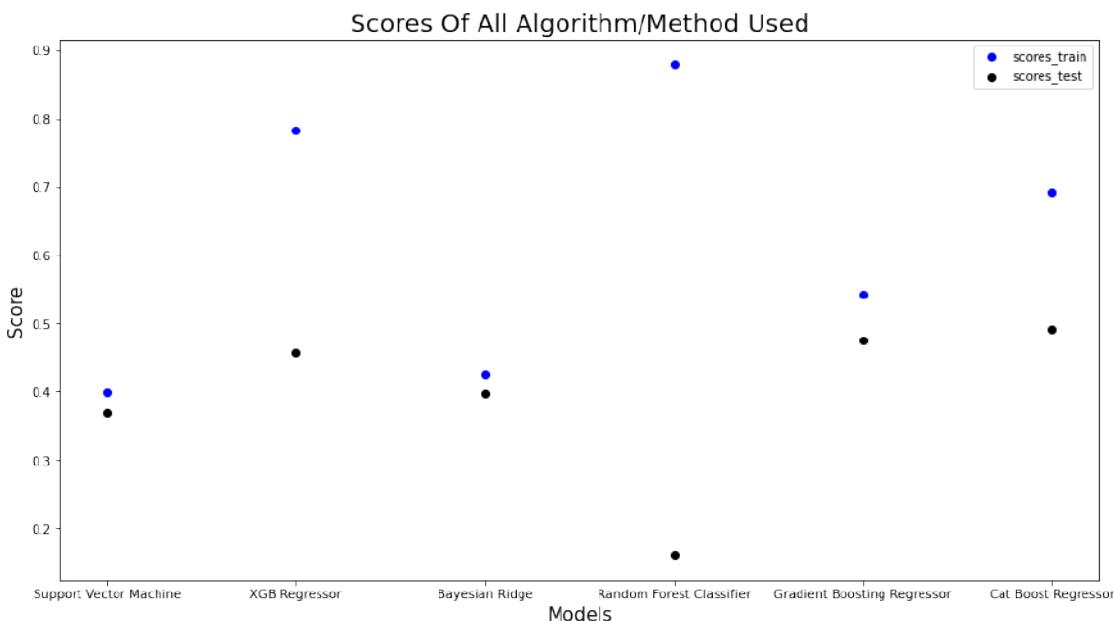
#### 1.1.4 Step 4: R-Squared Visualization

Now that we have obtained the R-squared scores of both data using different models, I will be comparing each one against each other.

```
[32]: scores_train = [svr_model_train, xgb_model_train, baye_model_train,
                     rfc_model_train, gradient_model_train, cat_model_train]
scores_test = [svr_model_test, xgb_model_test, baye_model_test,
               rfc_model_test, gradient_model_test, cat_model_test]
labels = ['Support Vector Machine', 'XGB Regressor', 'Bayesian Ridge',
          'Random Forest Classifier', 'Gradient Boosting Regressor', 'Cat BoostRegressor']
```

```
[34]: plt.rcParams['figure.figsize'] = [15, 8]

plt.scatter(labels, scores_train, color = 'blue')
plt.scatter(labels, scores_test, color = 'black')
plt.legend(['scores_train', 'scores_test'])
plt.ylabel('Score', fontsize = 15)
plt.xlabel('Models', fontsize = 15)
plt.title('Scores Of All Algorithm/Method Used', fontsize = 20)
plt.savefig('Scores Of All Models Used.png');
```



Here, we can see that the Support Method Machine and Bayesian Ridge Models have the smallest gap while the Random Forest Classifier has the widest one. However, our models fail at explaining prices for different independent variables. Therefore, I can conclude that we have not found the best model, but this is a good start.

However, looking at the model and if I had to choose 2 models, I would go for Gradient Boosting Regressor(GBR) and Bayesian Ridge, in that order. Firstly, Bayesian Ridge has a similar distance to the Support Vector Machine model, but the r2 scores for the train and test data are higher.

In terms of percentage, GBR's r2 scores have a slightly larger distance between them, but both scores are higher than Bayesian Ridge. In other words, I would say that the distance is small enough for me to consider it as a usable model(out of all the ones presented).

[ ]:

[ ]:

[ ]:

[ ]:



**project**

**13**

**Python - IBM Recommendation  
Engine Project**

# IBM Recommendation Engine

## Overview

This project was designed to analyze the interactions that users have with articles on the IBM Watson Studio platform, and make recommendations to them about new articles that they might be interested in.

## Packages

List of packages used:

- Matplotlib
- Numpy
- Pandas
- Pickle

## Summary:

The project contains the following tasks:

- Exploratory Data Analysis: This part is for data exploration.
- Rank Based Recommendations: Here, I begin by finding the most popular articles based on the most interactions. These articles are the ones that we might recommend to new users.
- User-User Based Collaborative Filtering: In order to give better recommendations to the users of IBM's platform, I examine users that are similar in terms of the items they have interacted with. These items could then be recommended to similar users.
- Matrix Factorization: For the final step, I created a machine learning approach to building recommendations. Using the user-item interactions, I built out a matrix decomposition which helps me in predicting new articles an individual might interact with .

# Recommendations with IBM

In this notebook, you will be putting your recommendation skills to use on real data from the IBM Watson Studio platform.

You may either submit your notebook through the workspace here, or you may work from your local machine and submit through the next page. Either way assure that your code passes the project [RUBRIC](#). **Please save regularly.**

By following the table of contents, you will build out a number of different methods for making recommendations that can be used for different situations.

## Table of Contents

- I. [Exploratory Data Analysis](#)
- II. [Rank Based Recommendations](#)
- III. [User-User Based Collaborative Filtering](#)
- IV. [Content Based Recommendations \(EXTRA - NOT REQUIRED\)](#)
- V. [Matrix Factorization](#)
- VI. [Extras & Concluding](#)

At the end of the notebook, you will find directions for how to submit your work. Let's get started by importing the necessary libraries and reading in the data.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import project_tests as t
import pickle

%matplotlib inline

df = pd.read_csv('data/user-item-interactions.csv')
df_content = pd.read_csv('data/articles_community.csv')
del df['Unnamed: 0']
del df_content['Unnamed: 0']

# Show df to get an idea of the data
df.head()

   article_id                               title \
0      1430.0  using pixiedust for fast, flexible, and easier...
1      1314.0      healthcare python streaming application demo
2      1429.0          use deep learning for image classification
3      1338.0            ml optimization using cognitive assistant
4      1276.0           deploy your python model as a restful api
```

```

email
0 ef5f11f77ba020cd36e1105a00ab868bbdbf7fe7
1 083cbdfa93c8444beaa4c5f5e0f5f9198e4f9e0b
2 b96a4f2e92d8572034b1e9b28f9ac673765cd074
3 06485706b34a5c9bf2a0ecdac41daf7e7654ceb7
4 f01220c46fc92c6e6b161b1849de11faacd7ccb2

# Show df_content to get an idea of the data
df_content.head()

doc_body \
0 Skip navigation Sign in SearchLoading...\\r\\n\\r...
1 No Free Hunch Navigation * kaggle.com\\r\\n\\r\\n ...
2 ≡ * Login\\r\\n * Sign Up\\r\\n\\r\\n * Learning Pat...
3 DATALAYER: HIGH THROUGHPUT, LOW LATENCY AT SCA...
4 Skip navigation Sign in SearchLoading...\\r\\n\\r...

doc_description \
0 Detect bad readings in real time using Python ...
1 See the forest, see the trees. Here lies the c...
2 Here's this week's news in Data Science and Bi...
3 Learn how distributed DBs solve the problem of...
4 This video demonstrates the power of IBM DataS...

doc_full_name doc_status
article_id
0 Detect Malfunctioning IoT Sensors with Streami... Live
0
1 Communicating data science: A guide to present... Live
1
2 This Week in Data Science (April 18, 2017) Live
2
3 DataLayer Conference: Boost the performance of... Live
3
4 Analyze NY Restaurant data using Spark in DSX Live
4
```

## Part I : Exploratory Data Analysis

Use the dictionary and cells below to provide some insight into the descriptive statistics of the data.

1. What is the distribution of how many articles a user interacts with in the dataset? Provide a visual and descriptive statistics to assist with giving a look at the number of times each user interacts with an article.

```
# finding out the number of rows and columns in each dataset
df.shape, df_content.shape
```

```
((45993, 3), (1056, 5))

# find out the total nulls in df
df.isnull().sum()

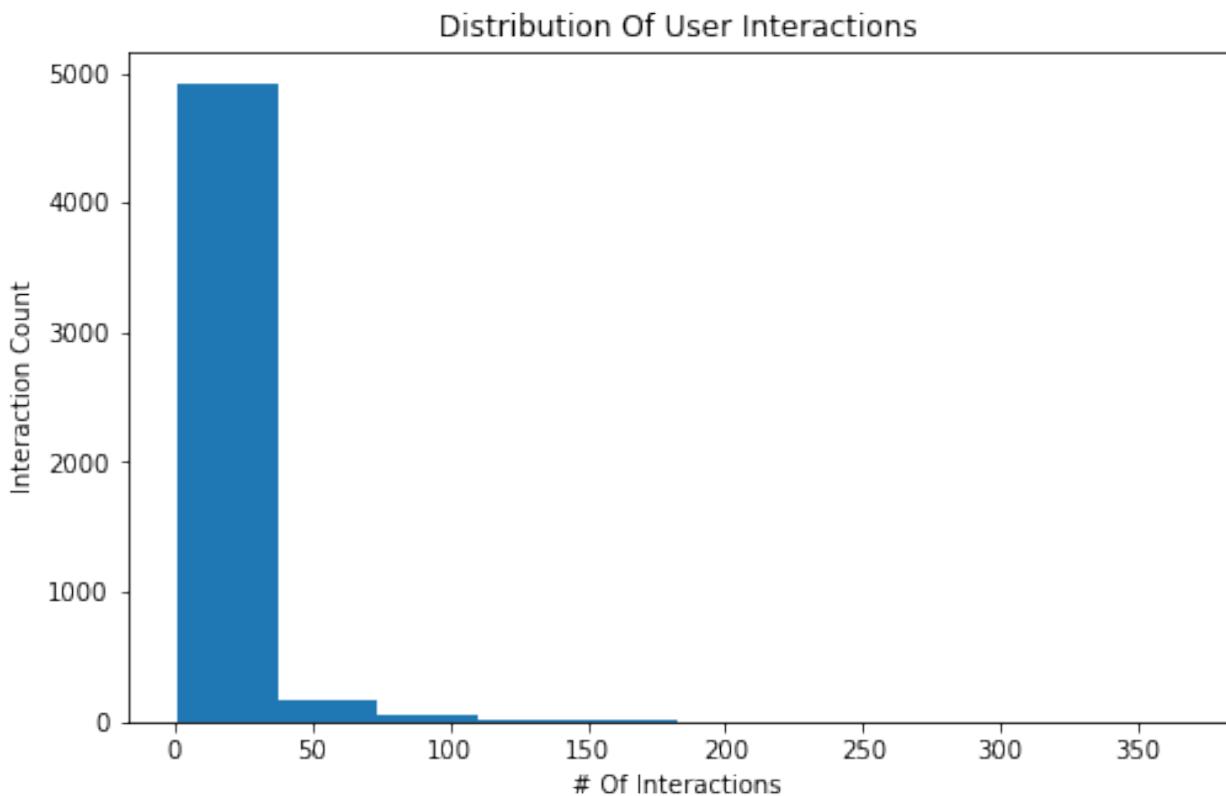
article_id      0
title          0
email         17
dtype: int64

df_content.isnull().sum()

doc_body        14
doc_description  3
doc_full_name   0
doc_status       0
article_id       0
dtype: int64

# first, let's count the number of interactions by users
# we will target the users by using the emails provided
# counts number of interactions by email
interaction_user = df['email'].value_counts(dropna = False)

plt.figure(figsize = (8, 5))
interaction_user.hist()
plt.grid(False)
plt.title('Distribution Of User Interactions')
plt.xlabel('# Of Interactions')
plt.ylabel('Interaction Count');
```



```
interaction_user.describe()

count      5149.000000
mean       8.932414
std        16.801011
min        1.000000
25%        1.000000
50%        3.000000
75%        9.000000
max       364.000000
Name: email, dtype: float64
```

# Fill in the median and maximum number of user\_article interactions below

median\_val = 3 # 50% of individuals interact with \_\_\_\_ number of articles or fewer.  
 max\_views\_by\_user = 364 # The maximum number of user-article interactions by any 1 user is \_\_\_\_.

2. Explore and remove duplicate articles from the **df\_content** dataframe.

```
# Find and explore duplicate articles
df_content['article_id'].duplicated().sum()
```

```
5
```

```
# viewing all duplicates in a dataframe
df_content[df_content['article_id'].duplicated()]

   doc_body \
365 Follow Sign in / Sign up Home About Insight Da...
692 Homepage Follow Sign in / Sign up Homepage * H...
761 Homepage Follow Sign in Get started Homepage *...
970 This video shows you how to construct queries ...
971 Homepage Follow Sign in Get started * Home\r\n...

   doc_description \
365 During the seven-week Insight Data Engineering...
692 One of the earliest documented catalogs was co...
761 Today's world of data science leverages data f...
970 This video shows you how to construct queries ...
971 If you are like most data scientists, you are ...

   doc_full_name doc_status
article_id
365                 Graph-based machine learning      Live
50
692 How smart catalogs can turn the big data flood...      Live
221
761 Using Apache Spark as a parallel processing fr...      Live
398
970                 Use the Primary Index      Live
577
971 Self-service data preparation with IBM Data Re...      Live
232

# viewing rows where article_id = 50
df_content[df_content['article_id'] == 50]

   doc_body \
50 Follow Sign in / Sign up Home About Insight Da...
365 Follow Sign in / Sign up Home About Insight Da...

   doc_description \
50                         Community Detection at Scale
365 During the seven-week Insight Data Engineering...

   doc_full_name doc_status  article_id
50 Graph-based machine learning      Live          50
365 Graph-based machine learning      Live          50

# Remove any rows that have the same article_id - only keep the first
df_content.drop_duplicates(subset='article_id', inplace = True)
```

```
# Check
df_content['article_id'].duplicated().sum()

0

# double-check
df_content[df_content['article_id'] == 50]

      doc_body \
50  Follow Sign in / Sign up Home About Insight Da...
                    doc_description          doc_full_name
doc_status \
50  Community Detection at Scale  Graph-based machine learning
Live

      article_id
50      50
```

3. Use the cells below to find:

- a. The number of unique articles that have an interaction with a user.
- b. The number of unique articles in the dataset (whether they have any interactions or not). c.
- The number of unique users in the dataset. (excluding null values) d. The number of user-article interactions in the dataset.

```
df.unique()

article_id    714
title         714
email        5148
dtype: int64

df_content.unique()

doc_body       1031
doc_description 1019
doc_full_name   1051
doc_status        1
article_id       1051
dtype: int64

unique_articles = 714 # The number of unique articles that have at
                     least one interaction
total_articles = 1051 # The number of unique articles on the IBM
                      platform
unique_users = 5148 # The number of unique users
user_article_interactions = 45993 # The number of user-article
                                interactions
```

4. Use the cells below to find the most viewed `article_id`, as well as how often it was viewed. After talking to the company leaders, the `email_mapper` function was deemed a reasonable way to map users to ids. There were a small number of null values, and it was found that all of these null values likely belonged to a single user (which is how they are stored using the function below).

```
df.groupby('article_id').count().sort_values(by = 'email', ascending = False).head()

      title   email
article_id
1429.0      937    937
1330.0      927    927
1431.0      671    671
1427.0      643    643
1364.0      627    627

# The most viewed article in the dataset as a string with one value following the decimal
most_viewed_article_id = '1429.0'
max_views = 937 # The most viewed article in the dataset was viewed how many times?

## No need to change the code here - this will be helpful for later parts of the notebook
# Run this cell to map the user email to a user_id column and remove the email column

def email_mapper():
    coded_dict = dict()
    cter = 1
    email_encoded = []

    for val in df['email']:
        if val not in coded_dict:
            coded_dict[val] = cter
            cter+=1

        email_encoded.append(coded_dict[val])
    return email_encoded

email_encoded = email_mapper()
del df['email']
df['user_id'] = email_encoded

# show header
df.head()

      article_id                      title
user_id
0      1430.0  using pixiedust for fast, flexible, and easier...
```

```

1      1314.0      healthcare python streaming application demo
2      1429.0      use deep learning for image classification
3      1338.0      ml optimization using cognitive assistant
4      1276.0      deploy your python model as a restful api
5

## If you stored all your results in the variable names above,
## you shouldn't need to change anything in this cell

sol_1_dict = {
    ``50% of individuals have ____ or fewer interactions.'': median_val,
    ``The total number of user-article interactions in the dataset is ____.'': user_article_interactions,
    ``The maximum number of user-article interactions by any 1 user is ____.'': max_views_by_user,
    ``The most viewed article in the dataset was viewed ____ times.'': max_views,
    ``The article_id of the most viewed article is ____.'': most_viewed_article_id,
    ``The number of unique articles that have at least 1 rating ____.'': unique_articles,
    ``The number of unique users in the dataset is ____.'': unique_users,
    ``The number of unique articles on the IBM platform''': total_articles
}

# Test your dictionary against the solution
t.sol_1_test(sol_1_dict)

It looks like you have everything right here! Nice job!

```

## Part II: Rank-Based Recommendations

Unlike in the earlier lessons, we don't actually have ratings for whether a user liked an article or not. We only know that a user has interacted with an article. In these cases, the popularity of an article can really only be based on how often an article was interacted with.

- Fill in the function below to return the **n** top articles ordered with most interactions as the top. Test your function using the tests below.

```
list(df.groupby('article_id').count().sort_values(by = 'user_id',
ascending = False).head().index)
```

```
[1429.0, 1330.0, 1431.0, 1427.0, 1364.0]
```

```

def get_top_articles(n, df=df):
    """
    INPUT:
    n - (int) the number of top articles to return
    df - (pandas dataframe) df as defined at the top of the notebook

    OUTPUT:
    top_articles - (list) A list of the top 'n' article titles

    ...
    # Your code here
    top_articles = list(df.groupby('title').count().sort_values(by =
'user_id',
ascending = False).head(n).index)
    return top_articles # Return the top article titles from df (not
df_content)

def get_top_article_ids(n, df=df):
    """
    INPUT:
    n - (int) the number of top articles to return
    df - (pandas dataframe) df as defined at the top of the notebook

    OUTPUT:
    top_articles - (list) A list of the top 'n' article titles

    ...
    # Your code here
    top_articles =
list(df.groupby('article_id').count().sort_values(by = 'user_id',
ascending = False).head(n).index)
    return top_articles # Return the top article ids

print(get_top_articles(10))
print(get_top_article_ids(10))

['use deep learning for image classification', 'insights from new york
car accident reports', 'visualize car data with brunel', 'use xgboost,
scikit-learn & ibm watson machine learning apis', 'predicting churn
with the spss random tree algorithm', 'healthcare python streaming
application demo', 'finding optimal locations of new store using
decision optimization', 'apache spark lab, part 1: basic concepts',
'analyze energy consumption in buildings', 'gosales transactions for
logistic regression model']
[1429.0, 1330.0, 1431.0, 1427.0, 1364.0, 1314.0, 1293.0, 1170.0,
1162.0, 1304.0]

```

```

# Test your function by returning the top 5, 10, and 20 articles
top_5 = get_top_articles(5)
top_10 = get_top_articles(10)
top_20 = get_top_articles(20)

# Test each of your three lists from above
t.sol_2_test(get_top_articles)

Your top_5 looks like the solution list! Nice job.
Your top_10 looks like the solution list! Nice job.
Your top_20 looks like the solution list! Nice job.

```

## Part III: User-User Based Collaborative Filtering

1. Use the function below to reformat the **df** dataframe to be shaped with users as the rows and articles as the columns.

- Each **user** should only appear in each **row** once.
- Each **article** should only show up in one **column**.
- **If a user has interacted with an article, then place a 1 where the user-row meets for that article-column.** It does not matter how many times a user has interacted with the article, all entries where a user has interacted with an article should be a 1.
- **If a user has not interacted with an item, then place a zero where the user-row meets for that article-column.**

Use the tests to make sure the basic structure of your matrix matches what is expected by the solution.

```

# create the user-article matrix with 1's and 0's

def create_user_item_matrix(df):
    """
    INPUT:
    df - pandas dataframe with article_id, title, user_id columns

    OUTPUT:
    user_item - user item matrix

    Description:
    Return a matrix with user ids as rows and article ids on the
    columns with 1 values where a user interacted with
    an article and a 0 otherwise
    """

    user_item = df.groupby(['user_id', 'article_id']).agg(lambda x:
1).unstack().fillna(0).astype('int')

```

```
    return user_item # return the user_item matrix

user_item = create_user_item_matrix(df)

## Tests: You should just need to run this cell. Don't change the
## code.
assert user_item.shape[0] == 5149, "Oops! The number of users in the
user-article matrix doesn't look right."
assert user_item.shape[1] == 714, "Oops! The number of articles in
the user-article matrix doesn't look right."
assert user_item.sum(axis=1)[1] == 36, "Oops! The number of articles
seen by user 1 doesn't look right."
print("You have passed our quick tests! Please proceed!")
```

You have passed our quick tests! Please proceed!

2. Complete the function below which should take a user\_id and provide an ordered list of the most similar users to that user (from most similar to least similar). The returned result should not contain the provided user\_id, as we know that each user is similar to him/herself. Because the results for each user here are binary, it (perhaps) makes sense to compute similarity as the dot product of two users.

Use the tests to test your function.

```
user_item.shape, np.transpose(user_item).shape
((5149, 714), (714, 5149))

m = user_item.dot(np.transpose(user_item))
sorted_m = m.loc[1,:].sort_values(ascending=False).keys()

list(sorted_m)

[1,
 3933,
 23,
 3782,
 203,
 4459,
 3870,
 131,
 4201,
 46,
 5041,
 395,
 3697,
 49,
 322,
 242,
 3622,
```

3910,  
98,  
754,  
2982,  
290,  
3540,  
4642,  
3764,  
912,  
268,  
40,  
3775,  
4932,  
4134,  
52,  
621,  
5138,  
1355,  
4785,  
3651,  
3637,  
256,  
273,  
371,  
204,  
3784,  
3621,  
3596,  
21,  
135,  
4038,  
3532,  
64,  
186,  
184,  
3483,  
214,  
2926,  
696,  
249,  
3141,  
765,  
619,  
4755,  
4778,  
4892,  
3136,  
488,  
334,

4293,  
1897,  
4934,  
125,  
4484,  
445,  
4206,  
4774,  
3684,  
5013,  
60,  
38,  
656,  
2790,  
187,  
10,  
4209,  
665,  
3856,  
3632,  
288,  
5140,  
4706,  
5079,  
54,  
72,  
67,  
3740,  
58,  
3024,  
2430,  
409,  
733,  
4883,  
3354,  
3408,  
193,  
4277,  
4225,  
3485,  
195,  
3949,  
3358,  
4543,  
379,  
170,  
211,  
3693,  
807,

223,  
591,  
3794,  
3353,  
235,  
90,  
3818,  
3197,  
4404,  
295,  
261,  
263,  
3006,  
4471,  
3572,  
3879,  
362,  
28,  
69,  
3578,  
2981,  
324,  
689,  
2975,  
87,  
1040,  
8,  
3,  
113,  
3169,  
111,  
4595,  
244,  
330,  
1059,  
4824,  
511,  
2161,  
4933,  
1353,  
319,  
1330,  
4082,  
4088,  
304,  
312,  
659,  
3589,  
598,

750,  
280,  
4037,  
4167,  
365,  
3067,  
3329,  
1271,  
639,  
3118,  
640,  
4021,  
251,  
1244,  
647,  
215,  
4130,  
3264,  
4901,  
3057,  
383,  
3741,  
4900,  
3100,  
3710,  
641,  
155,  
4697,  
3005,  
4788,  
456,  
82,  
557,  
558,  
88,  
4495,  
3172,  
1567,  
3898,  
2908,  
4502,  
1062,  
2903,  
3801,  
114,  
4510,  
4792,  
4802,  
3523,

3305,  
926,  
3636,  
3829,  
11,  
510,  
907,  
22,  
24,  
26,  
4453,  
890,  
45,  
535,  
471,  
3500,  
63,  
65,  
3376,  
670,  
829,  
2423,  
3967,  
3966,  
168,  
4231,  
4248,  
5023,  
4526,  
1163,  
1162,  
3943,  
4725,  
3441,  
3535,  
418,  
3237,  
126,  
4515,  
4517,  
373,  
438,  
1633,  
3732,  
4449,  
3474,  
521,  
490,  
492,

321,  
2897,  
512,  
4241,  
4323,  
1705,  
4282,  
5034,  
4620,  
2144,  
607,  
668,  
4588,  
1426,  
669,  
3633,  
1386,  
3336,  
536,  
538,  
3705,  
347,  
3466,  
3614,  
4186,  
1446,  
420,  
4268,  
3611,  
343,  
4356,  
4272,  
1401,  
3061,  
3058,  
4255,  
4254,  
4274,  
4511,  
3292,  
3654,  
4522,  
3430,  
542,  
649,  
4497,  
253,  
5057,  
745,

3960,  
181,  
3421,  
3970,  
3971,  
763,  
197,  
3783,  
213,  
2305,  
995,  
5077,  
3877,  
57,  
56,  
3425,  
55,  
4843,  
4002,  
884,  
5080,  
4013,  
3486,  
2718,  
860,  
3417,  
2471,  
3802,  
1068,  
843,  
122,  
828,  
824,  
5124,  
845,  
4736,  
134,  
136,  
1137,  
140,  
4732,  
145,  
4882,  
794,  
3518,  
85,  
164,  
5078,  
4904,

4668,  
688,  
707,  
4648,  
5069,  
2586,  
698,  
277,  
896,  
697,  
3812,  
5143,  
4071,  
1301,  
927,  
9,  
6,  
681,  
296,  
3208,  
5062,  
4623,  
305,  
5059,  
5070,  
700,  
4768,  
41,  
722,  
1272,  
1281,  
37,  
726,  
2579,  
240,  
1671,  
532,  
4861,  
987,  
3196,  
938,  
4842,  
4504,  
885,  
842,  
4450,  
2993,  
565,  
1566,

4969,  
889,  
1668,  
1014,  
866,  
1018,  
4795,  
3007,  
4855,  
518,  
3618,  
555,  
4491,  
871,  
895,  
3191,  
4496,  
998,  
970,  
4786,  
977,  
4837,  
854,  
1038,  
2989,  
1574,  
1058,  
1513,  
3610,  
712,  
4943,  
4586,  
1262,  
1264,  
3069,  
631,  
3072,  
727,  
1413,  
1405,  
4909,  
721,  
4663,  
3108,  
3564,  
1437,  
4647,  
648,  
650,

651,  
3079,  
4921,  
3570,  
3093,  
662,  
687,  
1343,  
4618,  
4619,  
1354,  
1261,  
741,  
4506,  
785,  
833,  
4763,  
1076,  
4749,  
820,  
4876,  
798,  
791,  
790,  
4716,  
585,  
4711,  
3031,  
674,  
1499,  
1251,  
4707,  
1198,  
1213,  
596,  
3042,  
4552,  
4554,  
761,  
760,  
757,  
4685,  
1240,  
1247,  
4947,  
1673,  
924,  
260,  
92,  
354,

4145,  
5115,  
4142,  
2447,  
129,  
4137,  
346,  
335,  
4110,  
332,  
2128,  
132,  
326,  
4100,  
4098,  
4097,  
133,  
5052,  
4148,  
121,  
359,  
107,  
94,  
4204,  
3436,  
95,  
103,  
4197,  
3378,  
3324,  
2688,  
2318,  
3327,  
375,  
2845,  
3720,  
2029,  
4173,  
3434,  
120,  
5053,  
3335,  
3917,  
4015,  
2336,  
2248,  
2251,  
4020,  
245,

2335,  
241,  
3972,  
230,  
3968,  
199,  
225,  
200,  
2290,  
2324,  
2320,  
2312,  
205,  
182,  
4427,  
310,  
5103,  
3918,  
3749,  
2410,  
3338,  
294,  
3414,  
2174,  
2176,  
3757,  
3763,  
2197,  
2199,  
160,  
3363,  
163,  
169,  
4047,  
4045,  
3695,  
4161,  
209,  
4336,  
3835,  
3884,  
74,  
2953,  
4311,  
2510,  
3310,  
1747,  
2949,  
479,

2593,  
2899,  
475,  
3876,  
48,  
44,  
2934,  
1787,  
3807,  
1775,  
4367,  
35,  
487,  
2913,  
2898,  
86,  
4393,  
4245,  
3640,  
4,  
4390,  
13,  
5129,  
404,  
3404,  
89,  
1227,  
1187,  
2661,  
3951,  
975,  
2637,  
2370,  
2377,  
3195,  
2,  
2667,  
2376,  
4717,  
1229,  
3790,  
1230,  
1165,  
2374,  
925,  
974,  
2646,  
2621,  
2725,

3543,  
1214,  
3819,  
3833,  
1219,  
931,  
939,  
2728,  
950,  
4695,  
953,  
1208,  
930,  
3198,  
3147,  
1200,  
1199,  
955,  
1223,  
2342,  
4700,  
2350,  
1225,  
2355,  
1194,  
957,  
3947,  
2488,  
1154,  
2465,  
3159,  
3163,  
4800,  
2507,  
1003,  
3912,  
3883,  
3885,  
2450,  
2451,  
2682,  
3888,  
2458,  
1025,  
1026,  
2384,  
3889,  
1065,  
3520,

1063,  
3890,  
1061,  
1031,  
2475,  
4783,  
2479,  
2482,  
2484,  
2485,  
1035,  
3878,  
996,  
1098,  
1101,  
2387,  
2710,  
3938,  
2561,  
2406,  
3872,  
1139,  
2487,  
4809,  
2556,  
2552,  
984,  
2521,  
1132,  
3155,  
1129,  
1127,  
2516,  
988,  
4741,  
3528,  
1114,  
4746,  
4806,  
3190,  
2514,  
4747,  
1105,  
3913,  
2415,  
2974,  
1234,  
3691,  
4246,

4527,  
2879,  
1527,  
4250,  
4275,  
1543,  
1879,  
2901,  
4296,  
4300,  
4302,  
4503,  
3674,  
4307,  
2912,  
4316,  
1523,  
1512,  
4321,  
3600,  
1449,  
2007,  
3711,  
3052,  
3051,  
4559,  
4557,  
1476,  
4199,  
4553,  
1994,  
3046,  
4545,  
1988,  
4230,  
1498,  
4233,  
4319,  
2919,  
2850,  
1752,  
4456,  
2992,  
4378,  
4379,  
4444,  
1655,  
3629,  
4384,

1661,  
4386,  
4437,  
1665,  
4392,  
4401,  
4414,  
1685,  
3634,  
3650,  
4371,  
4494,  
2994,  
4329,  
2924,  
3664,  
919,  
1588,  
4485,  
2932,  
3009,  
1595,  
1780,  
4363,  
4364,  
2941,  
1766,  
1620,  
1627,  
1764,  
3713,  
4577,  
3986,  
1284,  
4658,  
4034,  
4652,  
4039,  
1302,  
3565,  
4049,  
3566,  
1313,  
2768,  
2208,  
2205,  
1320,  
2202,  
4057,

1329,  
2180,  
4660,  
3559,  
1340,  
3765,  
2304,  
3991,  
3554,  
2742,  
2288,  
4674,  
4673,  
2284,  
4004,  
4016,  
1265,  
2258,  
4667,  
2257,  
1274,  
4025,  
3768,  
4068,  
4073,  
3065,  
3745,  
4608,  
4107,  
2800,  
3737,  
4127,  
4128,  
2813,  
2815,  
3733,  
3581,  
1409,  
1411,  
2819,  
2826,  
2830,  
1427,  
4170,  
1389,  
3746,  
4076,  
1384,  
1347,

2164,  
1350,  
3088,  
2163,  
1352,  
2788,  
4624,  
4086,  
1359,  
1361,  
3087,  
4091,  
4096,  
1370,  
3082,  
3081,  
1807,  
4638,  
3824,  
233,  
706,  
709,  
4915,  
711,  
4914,  
227,  
714,  
226,  
3482,  
4910,  
220,  
219,  
723,  
5083,  
730,  
4907,  
734,  
3255,  
239,  
636,  
246,  
285,  
646,  
3341,  
3428,  
276,  
661,  
272,  
271,

666,  
667,  
673,  
266,  
264,  
262,  
3426,  
254,  
248,  
736,  
189,  
751,  
3420,  
149,  
3413,  
146,  
3238,  
5108,  
3368,  
3488,  
3235,  
809,  
810,  
4877,  
3373,  
4872,  
112,  
110,  
822,  
104,  
150,  
780,  
151,  
159,  
752,  
753,  
755,  
176,  
759,  
173,  
166,  
770,  
152,  
156,  
153,  
771,  
773,  
4891,  
778,  
3240,

```

638,
...]

def find_similar_users(user_id, user_item=user_item):
    """
    INPUT:
    user_id - (int) a user_id
    user_item - (pandas dataframe) matrix of users by articles:
                1's when a user has interacted with an article, 0
                otherwise

    OUTPUT:
    similar_users - (list) an ordered list where the closest users
    (largest dot product users)
                are listed first

    Description:
    Computes the similarity of every pair of users based on the dot
    product
    Returns an ordered
    ...
    # compute similarity of each user to the provided user
    matrix = user_item.dot(np.transpose(user_item))
    # sort by similarity
    sorted_matrix = matrix.loc[(user_id), :].sort_values(ascending =
False).keys()
    # create list of just the ids
    sorted_list = list(sorted_matrix)
    # remove the own user's id
    sorted_list.remove(user_id)
    most_similar_users = sorted_list

    return most_similar_users # return a list of the users in order
from most to least similar

# Do a spot check of your function
print("The 10 most similar users to user 1 are:
{}".format(find_similar_users(1)[:10]))
print("The 5 most similar users to user 3933 are:
{}".format(find_similar_users(3933)[:5]))
print("The 3 most similar users to user 46 are:
{}".format(find_similar_users(46)[:3]))

The 10 most similar users to user 1 are: [3933, 23, 3782, 203, 4459,
3870, 131, 4201, 46, 5041]
The 5 most similar users to user 3933 are: [1, 23, 3782, 203, 4459]
The 3 most similar users to user 46 are: [4201, 3782, 23]

```

3. Now that you have a function that provides the most similar users to each user, you will want to use these users to find articles you can recommend. Complete the functions below to return the articles you would recommend to each user.

```
# test space for get_article_names function
article_ids = [1314, 1429]
article_names = []
for i in article_ids:
    article_name = df[df['article_id'] == float(i)]['title'].values[0]
    article_names.append(article_name)
article_names

['healthcare python streaming application demo',
 'use deep learning for image classification']

user_item.head()

      title
 \
article_id 0.0    2.0    4.0    8.0    9.0   12.0   14.0   15.0
16.0
user_id

1          0      0      0      0      0      0      0      0
0
2          0      0      0      0      0      0      0      0
0
3          0      0      0      0      0      1      0      0
0
4          0      0      0      0      0      0      0      0
0
5          0      0      0      0      0      0      0      0
0

      ...
 \
article_id 18.0    ... 1434.0 1435.0 1436.0 1437.0 1439.0 1440.0
1441.0
user_id     ...

1          0  ...      0      0      1      0      1      0
0
2          0  ...      0      0      0      0      0      0
0
3          0  ...      0      0      1      0      0      0
0
4          0  ...      0      0      0      0      0      0
0
5          0  ...      0      0      0      0      0      0
```

```
0
```

```
article_id 1442.0 1443.0 1444.0
user_id
1          0      0      0
2          0      0      0
3          0      0      0
4          0      0      0
5          0      0      0
```

```
[5 rows x 714 columns]
```

```
def get_article_names(article_ids, df=df):
    ...
    INPUT:
        article_ids - (list) a list of article ids
        df - (pandas dataframe) df as defined at the top of the notebook

    OUTPUT:
        article_names - (list) a list of article names associated with the
        list of article ids
                                (this is identified by the title column)
    ...
    # Your code here
    article_names = []
    for i in article_ids:
        article_name = df[df['article_id'] == float(i)]
    ['title'].values[0]
        article_names.append(article_name)

    return article_names # Return the article names associated with
list of article ids

def get_user_articles(user_id, user_item=user_item):
    ...
    INPUT:
        user_id - (int) a user id
        user_item - (pandas dataframe) matrix of users by articles:
                    1's when a user has interacted with an article, 0
                    otherwise

    OUTPUT:
        article_ids - (list) a list of the article ids seen by the user
        article_names - (list) a list of article names associated with the
        list of article ids
                                (this is identified by the doc_full_name column in
        df_content)
```

```

Description:
Provides a list of the article_ids and article titles that have
been seen by a user
```
# Your code here
row_user = user_item.loc[user_id]
article_ids = [str(i[1]) for i in list(row_user[row_user >
0].index)]
article_names = get_article_names(article_ids)

return article_ids, article_names # return the ids and names

def user_user_recs(user_id, m=10):
```
INPUT:
user_id - (int) a user id
m - (int) the number of recommendations you want for the user

OUTPUT:
recs - (list) a list of recommendations for the user

Description:
Loops through the users based on closeness to the input user_id
For each user - finds articles the user hasn't seen before and
provides them as recs
Does this until m recommendations are found

Notes:
Users who are the same closeness are chosen arbitrarily as the
'next' user

For the user where the number of recommended articles starts below
m
and ends exceeding m, the last items are chosen arbitrarily
```
# Your code here
recs = []
similar_users = find_similar_users(user_id)
user_articles, article_names = get_user_articles(user_id)

for user in similar_users:
    article_ids, article_names = get_user_articles(user)
    for id in article_ids:
        if len(recs) >= m:
            break
        else:
            if id not in user_articles:

```

```

        recs.append(id)

    return recs # return your recommendations for this user_id

# test space for get_user_articles function
u_id = 5
row_user = user_item.loc[u_id]
a_id = [i[1] for i in list(row_user[row_user > 0].index)]
a_names = get_article_names(a_id)
a_id # must remember to convert this to string for assertion

[1166.0, 1276.0, 1351.0]

# making sure data types are strings
get_user_articles(20)

(['232.0', '844.0', '1320.0'],
 ['self-service data preparation with ibm data refinery',
  'use the cloudant-spark connector in python notebook',
  'housing (2015): united states demographic measures'])

# Check Results
get_article_names(user_user_recs(1, 10)) # Return 10 recommendations for user 1

['this week in data science (april 18, 2017)',
 'timeseries data analysis of iot events by using jupyter notebook',
 'got zip code data? prep it for analytics. – ibm watson data lab – medium',
 'higher-order logistic regression for large datasets',
 'using machine learning to predict parking difficulty',
 'deep forest: towards an alternative to deep neural networks',
 'experience iot with coursera',
 'using brunel in ipython/jupyter notebooks',
 'graph-based machine learning',
 'the 3 kinds of context: machine learning and the art of the frame']

# Test your functions here - No need to change this code - just run this cell
assert set(get_article_names(['1024.0', '1176.0', '1305.0', '1314.0',
 '1422.0', '1427.0'])) == set(['using deep learning to reconstruct high-resolution audio',
 'build a python app on the streaming analytics service',
 'gosales transactions for naive bayes model',
 'healthcare python streaming application demo',
 'use r dataframes & ibm watson natural language understanding',
 'use xgboost, scikit-learn & ibm watson machine learning apis']), "Oops! Your the get_article_names function doesn't work quite how we expect."
assert set(get_article_names(['1320.0', '232.0', '844.0'])) == set(['housing (2015): united states demographic measures',
 'self-service data preparation with ibm data refinery',
 'use the cloudant-spark connector in python notebook']), "Oops! Your the

```

```

get_article_names function doesn't work quite how we expect."
assert set(get_user_articles(20)[0]) == set(['1320.0', '232.0',
'844.0'])
assert set(get_user_articles(20)[1]) == set(['housing (2015): united
states demographic measures', 'self-service data preparation with ibm
data refinery', 'use the cloudant-spark connector in python notebook'])
assert set(get_user_articles(2)[0]) == set(['1024.0', '1176.0',
'1305.0', '1314.0', '1422.0', '1427.0'])
assert set(get_user_articles(2)[1]) == set(['using deep learning to
reconstruct high-resolution audio', 'build a python app on the
streaming analytics service', 'gosales transactions for naive bayes
model', 'healthcare python streaming application demo', 'use r
dataframes & ibm watson natural language understanding', 'use xgboost,
scikit-learn & ibm watson machine learning apis'])
print("If this is all you see, you passed all of our tests! Nice
job!")

```

If this is all you see, you passed all of our tests! Nice job!

4. Now we are going to improve the consistency of the `user_user_recs` function from above.

- Instead of arbitrarily choosing when we obtain users who are all the same closeness to a given user - choose the users that have the most total article interactions before choosing those with fewer article interactions.
- Instead of arbitrarily choosing articles from the user where the number of recommended articles starts below m and ends exceeding m, choose articles with the articles with the most total interactions before choosing those with fewer total interactions. This ranking should be what would be obtained from the `top_articles` function you wrote earlier.

```

def get_top_sorted_users(user_id, df=df, user_item=user_item):
    ...
    INPUT:
    user_id - (int)
    df - (pandas dataframe) df as defined at the top of the notebook
    user_item - (pandas dataframe) matrix of users by articles:
        1's when a user has interacted with an article, 0
    otherwise

    OUTPUT:
    neighbors_df - (pandas dataframe) a dataframe with:
        neighbor_id - is a neighbor user_id
        similarity - measure of the similarity of each
    user to the provided user_id
        num_interactions - the number of articles viewed
    by the user - if a u

```

*Other Details - sort the neighbors\_df by the similarity and then by number of interactions where highest of each is higher in the dataframe*

```
'''  
# Your code here  
neighbors_df = pd.DataFrame(columns = ['neighbor_id',  
'similarity', 'num_interactions'])  
  
for user in user_item.index:  
    if user == user_id:  
        continue  
    # help with dot product:  
https://numpy.org/doc/stable/reference/generated/numpy.dot.html  
    neighbors_df.loc[user] = [user,  
    np.dot(user_item.loc[user_id, :], user_item.loc[user, :]),  
                            df[df['user_id'] == user]  
    ['article_id'].count()]  
    neighbors_df = neighbors_df.sort_values(by = ['similarity',  
'num_interactions'], ascending = False)  
return neighbors_df # Return the dataframe specified in the  
doc_string
```

```
def user_user_recs_part2(user_id, m=10):  
    '''  
    INPUT:  
    user_id - (int) a user id  
    m - (int) the number of recommendations you want for the user  
  
    OUTPUT:  
    recs - (list) a list of recommendations for the user by article id  
    rec_names - (list) a list of recommendations for the user by  
    article title  
  
    Description:  
    Loops through the users based on closeness to the input user_id  
    For each user - finds articles the user hasn't seen before and  
    provides them as recs  
    Does this until m recommendations are found  
  
    Notes:  
    * Choose the users that have the most total article interactions  
    before choosing those with fewer article interactions.  
  
    * Choose articles with the articles with the most total  
    interactions  
    before choosing those with fewer total interactions.  
    ...
```

```

# very similair to get_similair_user function but slightly altered
recs = []
rec_names = []
neighbors_df = get_top_sorted_users(user_id)
user_articles, article_names = get_user_articles(user_id)

for user in neighbors_df['neighbor_id']:
    article_ids, article_names = get_user_articles(user)
    for id in [str(id) for id in get_top_article_ids(100)]:
        if len(recs) >= m:
            break
        else:
            if id not in user_articles:
                recs.append(id)

# get name of article
for name in recs:
    rec_name = df[df.article_id == float(name)].iloc[0,1]
    rec_names.append(rec_name)

return recs, rec_names

# Quick spot check - don't change this code - just use it to test your
functions
rec_ids, rec_names = user_user_recs_part2(20, 10)
print("The top 10 recommendations for user 20 are the following")
print(article ids:")
print(rec_ids)
print()
print("The top 10 recommendations for user 20 are the following")
print(article names:")
print(rec_names)

The top 10 recommendations for user 20 are the following article ids:
['1429.0', '1330.0', '1431.0', '1427.0', '1364.0', '1314.0', '1293.0',
'1170.0', '1162.0', '1304.0']

The top 10 recommendations for user 20 are the following article
names:
['use deep learning for image classification', 'insights from new york
car accident reports', 'visualize car data with brunel', 'use xgboost,
scikit-learn & ibm watson machine learning apis', 'predicting churn
with the spss random tree algorithm', 'healthcare python streaming
application demo', 'finding optimal locations of new store using
decision optimization', 'apache spark lab, part 1: basic concepts',
'analyze energy consumption in buildings', 'gosales transactions for
logistic regression model']

```

5. Use your functions from above to correctly fill in the solutions to the dictionary below. Then test your dictionary against the solution. Provide the code you need to answer each following the comments below.

```
# test space
get_top_sorted_users(1).iloc[0,0]
get_top_sorted_users(131).iloc[9,0]

242

### Tests with a dictionary of results

user1_most_sim = get_top_sorted_users(1).iloc[0,0] # Find the user
that is most similar to user 1
user131_10th_sim = get_top_sorted_users(131).iloc[9,0] # Find the 10th
most similar user to user 131

## Dictionary Test Here
sol_5_dict = {
    'The user that is most similar to user 1.': user1_most_sim,
    'The user that is the 10th most similar to user 131':
user131_10th_sim,
}

t.sol_5_test(sol_5_dict)

This all looks good! Nice job!
```

6. If we were given a new user, which of the above functions would you be able to use to make recommendations? Explain. Can you think of a better way we might make recommendations? Use the cell below to explain a better method for new users.

**Since we do not have prior information/data about the new user, we will not be able to recommend any articles based on similarities with other users' browsing history. This is called a cold start problem. Therefore, we can only recommend articles based on popularity which is a rank-based method. Therefore, the best function to use is 'get\_top\_article\_ids'.**

7. Using your existing functions, provide the top 10 recommended articles you would provide for the a new user below. You can test your function against our thoughts to make sure we are all on the same page with how we might make a recommendation.

```
new_user = '0.0'

# What would your recommendations be for this new user '0.0'? As a
# new user, they have no observed articles.
# Provide a list of the top 10 article ids you would give to
# make sure to convert to string for check
new_user_recs = [str(x) for x in get_top_article_ids(10)]

assert set(new_user_recs) ==
set(['1314.0','1429.0','1293.0','1427.0','1162.0','1364.0','1304.0','1
```

```
170.0', '1431.0', '1330.0'])], "Oops! It makes sense that in this case  
we would want to recommend the most popular articles, because we don't  
know anything about these users."  
  
print("That's right! Nice job!")  
  
That's right! Nice job!
```

## Part IV: Content Based Recommendations (EXTRA - NOT REQUIRED)

Another method we might use to make recommendations is to perform a ranking of the highest ranked articles associated with some term. You might consider content to be the `doc_body`, `doc_description`, or `doc_full_name`. There isn't one way to create a content based recommendation, especially considering that each of these columns hold content related information.

1. Use the function body below to create a content based recommender. Since there isn't one right answer for this recommendation tactic, no test functions are provided. Feel free to change the function inputs if you decide you want to try a method that requires more input values. The input values are currently set with one idea in mind that you may use to make content based recommendations. One additional idea is that you might want to choose the most popular recommendations that meet your 'content criteria', but again, there is a lot of flexibility in how you might make these recommendations.

This part is NOT REQUIRED to pass this project. However, you may choose to take this on as an extra way to show off your skills.

```
def make_content_recs():  
    ...  
  
    INPUT:  
  
    OUTPUT:  
  
    ...
```

2. Now that you have put together your content-based recommendation system, use the cell below to write a summary explaining how your content based recommender works. Do you see any possible improvements that could be made to your function? Is there anything novel about your content based recommender?

This part is NOT REQUIRED to pass this project. However, you may choose to take this on as an extra way to show off your skills.

**Write an explanation of your content based recommendation system here.**

3. Use your content-recommendation system to make recommendations for the below scenarios based on the comments. Again no tests are provided here, because there isn't one right answer that could be used to find these content based recommendations.

This part is NOT REQUIRED to pass this project. However, you may choose to take this on as an extra way to show off your skills.

```
# make recommendations for a brand new user
```

```
# make a recommendations for a user who only has interacted with  
article id '1427.0'
```

## Part V: Matrix Factorization

In this part of the notebook, you will build use matrix factorization to make article recommendations to the users on the IBM Watson Studio platform.

1. You should have already created a `user_item` matrix above in **question 1 of Part III** above. This first question here will just require that you run the cells to get things set up for the rest of **Part V** of the notebook.

```
# Load the matrix here  
user_item_matrix = pd.read_pickle('user_item_matrix.p')  
  
# quick look at the matrix  
user_item_matrix.head()  
  
article_id    0.0   100.0  1000.0  1004.0  1006.0  1008.0  101.0   1014.0  
1015.0 \  
user_id  
  
1           0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0  
0.0  
2           0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0  
0.0  
3           0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0  
0.0  
4           0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0  
0.0  
5           0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0  
0.0  
  
article_id  1016.0    ...    977.0   98.0    981.0   984.0   985.0   986.0  
990.0 \  
user_id      ...  
  
1           0.0    ...     0.0     0.0     1.0     0.0     0.0     0.0  
0.0  
2           0.0    ...     0.0     0.0     0.0     0.0     0.0     0.0  
0.0  
3           0.0    ...     1.0     0.0     0.0     0.0     0.0     0.0  
0.0
```

```

4          0.0 ...    0.0  0.0    0.0    0.0    0.0    0.0    0.0
0.0
5          0.0 ...    0.0  0.0    0.0    0.0    0.0    0.0    0.0
0.0

article_id 993.0  996.0  997.0
user_id
1          0.0      0.0      0.0
2          0.0      0.0      0.0
3          0.0      0.0      0.0
4          0.0      0.0      0.0
5          0.0      0.0      0.0

[5 rows x 714 columns]

```

2. In this situation, you can use Singular Value Decomposition from [numpy](#) on the user-item matrix. Use the cell to perform SVD, and explain why this is different than in the lesson.

```

# Perform SVD on the User-Item Matrix Here
# use the built in to get the three matrices
u, s, vt = np.linalg.svd(user_item_matrix, full_matrices = True)
s.shape, u.shape, vt.shape

((714,), (5149, 5149), (714, 714))

```

**This is different than the lesson because this time there's no missing values in the matrix. Furthermore, since the values in the matrix only consists of 1's and 0's, it's better to use the standard SVD rather than FunkSVD.**

3. Now for the tricky part, how do we choose the number of latent features to use? Running the below cell, you can see that as the number of latent features increases, we obtain a lower error rate on making predictions for the 1 and 0 values in the user-item matrix. Run the cell below to get an idea of how the accuracy improves as we increase the number of latent features.

```

num_latent_feats = np.arange(10, 700+10, 20)
sum_errs = []

for k in num_latent_feats:
    # restructure with k latent features
    s_new, u_new, vt_new = np.diag(s[:k]), u[:, :k], vt[:k, :]

    # take dot product
    user_item_est = np.around(np.dot(u_new, s_new), vt_new))

    # compute error for each prediction to actual value
    diffs = np.subtract(user_item_matrix, user_item_est)

    # total errors and keep track of them
    err = np.sum(np.sum(np.abs(diffs)))

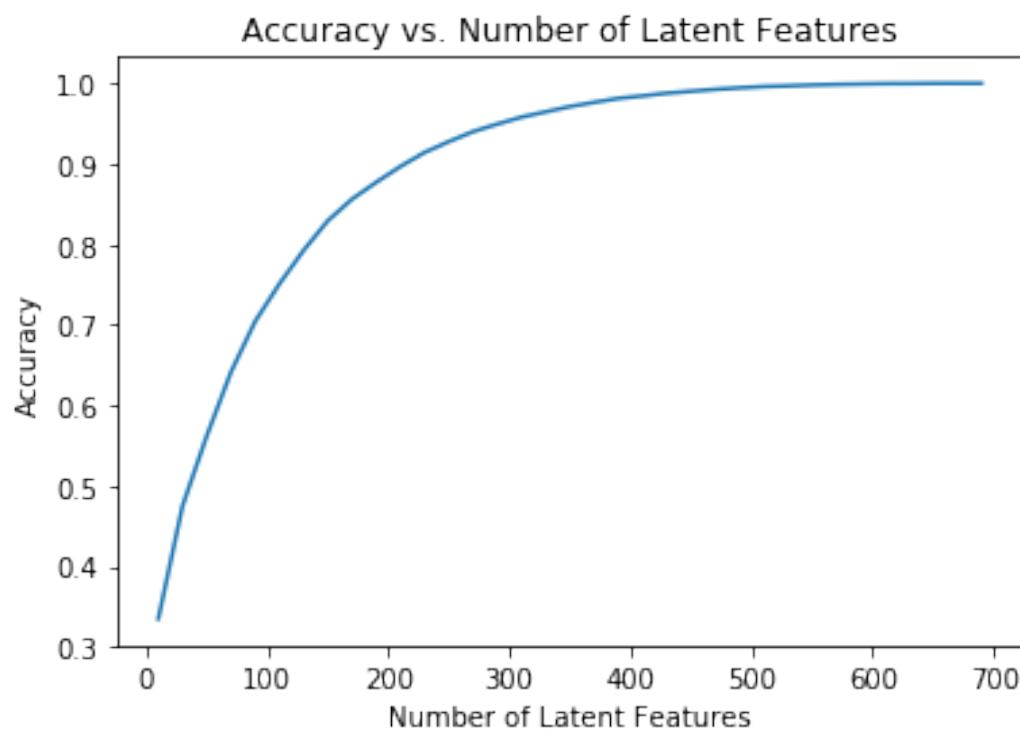
```

```

sum_errs.append(err)

plt.plot(num_latent_feats, 1 - np.array(sum_errs)/df.shape[0]);
plt.xlabel('Number of Latent Features');
plt.ylabel('Accuracy');
plt.title('Accuracy vs. Number of Latent Features');

```



4. From the above, we can't really be sure how many features to use, because simply having a better way to predict the 1's and 0's of the matrix doesn't exactly give us an indication of if we are able to make good recommendations. Instead, we might split our dataset into a training and test set of data, as shown in the cell below.

Use the code from question 3 to understand the impact on accuracy of the training and test sets of data with different numbers of latent features. Using the split below:

- How many users can we make predictions for in the test set?
- How many users are we not able to make predictions for because of the cold start problem?
- How many articles can we make predictions for in the test set?
- How many articles are we not able to make predictions for because of the cold start problem?

```

df_train = df.head(40000)
df_test = df.tail(5993)

```

```

def create_test_and_train_user_item(df_train, df_test):
    ...
    INPUT:
    df_train - training dataframe
    df_test - test dataframe

    OUTPUT:
    user_item_train - a user-item matrix of the training dataframe
                      (unique users for each row and unique articles
for each column)
    user_item_test - a user-item matrix of the testing dataframe
                      (unique users for each row and unique articles for
each column)
    test_idx - all of the test user ids
    test_arts - all of the test article ids

    ...
    user_item_train = create_user_item_matrix(df_train)
    user_item_test = create_user_item_matrix(df_test)

    test_idx = list(user_item_test.index)
    test_arts = list(user_item_test.columns)

    return user_item_train, user_item_test, test_idx, test_arts

user_item_train, user_item_test, test_idx, test_arts =
create_test_and_train_user_item(df_train, df_test)

train_idx = list(user_item_train.index)
train_arts = list(user_item_train.columns)

user_item_test.shape
(682, 574)

# test space
user_item_test.shape
len(np.intersect1d(test_idx, train_idx))
user_item_test.shape[0] - len(np.intersect1d(test_idx, train_idx))
len(test_arts) - user_item_test.shape[1]

0

# Replace the values in the dictionary below
a = 662
b = 574
c = 20
d = 0

# NOTE: 'articles' had to be changed to 'movies' to pass the check and

```

```

match the solution given by Udacity
sol_4_dict = {
    'How many users can we make predictions for in the test set?': c,
    'How many users in the test set are we not able to make
predictions for because of the cold start problem?': a,
    'How many movies can we make predictions for in the test set?': b,
    'How many movies in the test set are we not able to make
predictions for because of the cold start problem?': d
}

t.sol_4_test(sol_4_dict)

```

Awesome job! That's right! All of the test movies are in the training data, but there are only 20 test users that were also in the training set. All of the other users that are in the test set we have no data on. Therefore, we cannot make predictions for these users using SVD.

5. Now use the **user\_item\_train** dataset from above to find U, S, and V transpose using SVD. Then find the subset of rows in the **user\_item\_test** dataset that you can predict using this matrix decomposition with different numbers of latent features to see how many features makes sense to keep based on the accuracy on the test data. This will require combining what was done in questions 2 - 4.

Use the cells below to explore how well SVD works towards making predictions for recommendations on the test data.

```

# fit SVD on the user_item_train matrix
u_train, s_train, vt_train = np.linalg.svd(user_item_train)

u_train.shape, s_train.shape, vt_train.shape
((4487, 4487), (714,), (714, 714))

# Rows that match the test set
test_idx = user_item_test.index
row_idx = user_item_train.index.isin(test_idx)
u_test = u_train[row_idx, :]

# Columns that match the test set
test_col = user_item_test.columns
col_idx = user_item_train.columns.isin(test_col)
vt_test = vt_train[:, col_idx]

# Test data
train_idx = user_item_train.index
sub_user_item_test =
user_item_test.loc[user_item_test.index.isin(train_idx)]

num_latent_feats = np.arange(10, 700+10, 20)
sum_errs = []

```

```

for k in num_latent_feats:
    # restructure with k latent features
    s_train_new, u_train_new, vt_train_new = np.diag(s_train[:k]),
    u_train[:, :k], vt_train[:k, :]
    s_test_new, u_test_new, vt_test_new = s_train_new, u_test[:, :k],
    vt_test[:k, :]

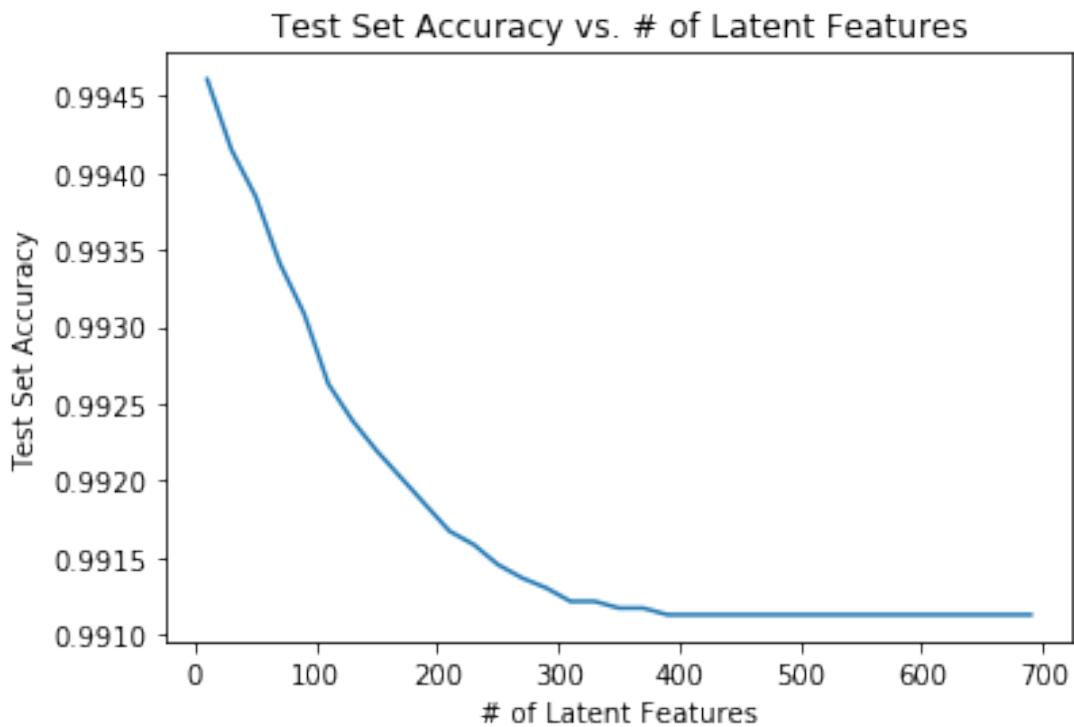
    # take dot product
    user_item_test_est = np.around(np.dot(np.dot(u_test_new,
    s_test_new), vt_test_new))

    # compute error for each prediction to actual value
    diff = np.subtract(sub_user_item_test, user_item_test_est)

    # total errors and keep track of them
    err = np.sum(np.sum(np.abs(diff)))
    sum_errs.append(err)

plt.plot(num_latent_feats, 1 - np.array(sum_errs)/df.shape[0])
plt.xlabel('# of Latent Features')
plt.ylabel('Test Set Accuracy')
plt.title('Test Set Accuracy vs. # of Latent Features')
plt.grid(False);

```



**6.** Use the cell below to comment on the results you found in the previous question. Given the circumstances of your results, discuss what you might do to determine if the recommendations you make with any of the above recommendation systems are an improvement to how users currently find articles?

From the above graph, we can see as the number of latent features increase, the accuracy gets worse. However, we should also keep in mind that we were working with a very small sample size, so we won't be able to make any solid conclusions. Furthermore, the accuracy rate is very high despite tapering off as the number of latent features increase. This can be explained by the imbalance of the two classes where there are many more 0's than 1's. Therefore, we can conclude that accuracy is not the best metric to use here. Even if the model predicts all 0's, the accuracy will still be high! Perhaps, the better metric to track here is clicks which is a cookie-based method.

Due to the low sample size, I believe that A/B testing would be more appropriate in this case. One group will have the new recommendation function and another will have the old one. We can then compute the mean interactions and use hypothesis testing to see if the new recommendation function is better than the old one. This first method that comes to mind is separating the user groups by cookies (both groups should have the same number of users). This experiment should be run for 3-5 months or until we hit a good sample size. As mentioned earlier, this project only offered a small sample size which provides insufficient data to compare predictions.

## Extras

Using your workbook, you could now save your recommendations for each user, develop a class to make new predictions and update your results, and make a flask app to deploy your results. These tasks are beyond what is required for this project. However, from what you learned in the lessons, you certainly capable of taking these tasks on to improve upon your work here!

## Conclusion

Congratulations! You have reached the end of the Recommendations with IBM project!

**Tip:** Once you are satisfied with your work here, check over your report to make sure that it satisfies all the areas of the [rubric](#). You should also probably remove all of the "Tips" like this one so that the presentation is as polished as possible.

## Directions to Submit

Before you submit your project, you need to create a .html or .pdf version of this notebook in the workspace here. To do that, run the code cell below. If it worked correctly, you should get a return code of 0, and you should see the generated .html file in the workspace directory (click on the orange Jupyter icon in the upper left).

Alternatively, you can download this report as .html via the **File > Download as** submenu, and then manually upload it into the workspace directory by clicking on the orange Jupyter icon in the upper left, then using the Upload button.

Once you've done this, you can submit your project by clicking on the "Submit Project" button in the lower right here. This will create and submit a zip file with this .ipynb doc and the .html or .pdf version you created. Congratulations!

```
from subprocess import call
call(['python', '-m', 'nbconvert', 'Recommendations_with_IBM.ipynb'])
0
```



**project**

**14**

**Python - Disaster Response Pipeline**

# Project Description

In this project, I will create a machine learning/NLP pipeline to categorize these events and build a model to classify messages that are sent during disasters. There are 36 pre-defined categories, and examples of these categories include Aid Related, Medical Help, Search And Rescue, etc. By classifying these messages, we can allow these messages to be sent to the appropriate disaster relief agency. The dataset -provided by Figure Eight- is used to build a model that classifies disaster messages, while the web app is where a respondent can input a new message and get classification results in several categories.

Finally, this project also contains a web app that allows you to input a message and get classification results.

## File Description

```
disaster_response_pipeline
    |-- app
        |-- templates
            |-- go.html
            |-- master.html
        |-- run.py
    |-- data
        |-- disaster_message.csv
        |-- disaster_categories.csv
        |-- DisasterResponse.db
        |-- process_data.py
    |-- models
        |-- classifier.pkl
        |-- train_classifier.py
    |-- Preparation
        |-- categories.csv
        |-- ETL Pipeline Preparation.ipynb
        |-- ETL_Preparation.db
        |-- messages.csv
        |-- ML Pipeline Preparation.ipynb
    |-- README
```

## Instructions:

1. Run the following commands in the project's root directory to set up your database and model.
  - o To run ETL pipeline that cleans data and stores in database `python data/process_data.py data/disaster_messages.csv data/disaster_categories.csv data/DisasterResponse.db`
  - o To run ML pipeline that trains classifier and saves `python models/train_classifier.py data/DisasterResponse.db models/classifier.pkl`
2. Run the following command in the app's directory to run your web app. `python run.py`
3. Go to <http://0.0.0.0:3001/>

# etl-pipeline-preparation

February 24, 2024

## 1 ETL Pipeline Preparation

Follow the instructions below to help you create your ETL pipeline.

### 1. Import libraries and load datasets.

- Import Python libraries
- Load `messages.csv` into a dataframe and inspect the first few lines.
- Load `categories.csv` into a dataframe and inspect the first few lines.

```
[43]: # import libraries
import pandas as pd
import numpy as np
import sqlite3
```

```
[44]: # load messages dataset
messages = pd.read_csv("messages.csv")
messages.head()
```

```
[44]:      id                         message \
0    2  Weather update - a cold front from Cuba that c...
1    7                  Is the Hurricane over or is it not over
2    8                      Looking for someone but no name
3    9  UN reports Leogane 80-90 destroyed. Only Hospi...
4   12  says: west side of Haiti, rest of the country ...

                                original   genre
0  Un front froid se retrouve sur Cuba ce matin. ...  direct
1          Cyclone nan fini osinon li pa fini  direct
2  Patnm, di Maryani relem pou li bamn nouvel li ...  direct
3  UN reports Leogane 80-90 destroyed. Only Hospi...  direct
4  facade ouest d Haiti et le reste du pays aujou...  direct
```

```
[45]: # load categories dataset
categories = pd.read_csv("categories.csv")
categories.head()
```

```
[45]:      id                     categories
0    2  related-1;request-0;offer-0;aid_related-0;medi...
1    7  related-1;request-0;offer-0;aid_related-1;medi...
2    8  related-1;request-0;offer-0;aid_related-0;medi...
3    9  related-1;request-1;offer-0;aid_related-1;medi...
```

```
4 12 related-1;request-0;offer-0;aid_related-0;medi...
```

### 1.0.1 2. Merge datasets.

- Merge the messages and categories datasets using the common id
- Assign this combined dataset to df, which will be cleaned in the following steps

[46]: # merge datasets

```
df = messages.merge(categories, on= 'id')
df.head()
```

[46]:

	id	message \
0	2	Weather update - a cold front from Cuba that c...
1	7	Is the Hurricane over or is it not over
2	8	Looking for someone but no name
3	9	UN reports Leogane 80-90 destroyed. Only Hospi...
4	12	says: west side of Haiti, rest of the country ...

	original genre \
0	Un front froid se retrouve sur Cuba ce matin. ... direct
1	Cyclone nan fini osinon li pa fini direct
2	Patnm, di Maryani relem pou li bamn nouvel li ... direct
3	UN reports Leogane 80-90 destroyed. Only Hospi... direct
4	facade ouest d Haiti et le reste du pays aujou... direct

	categories
0	related-1;request-0;offer-0;aid_related-0;medi...
1	related-1;request-0;offer-0;aid_related-1;medi...
2	related-1;request-0;offer-0;aid_related-0;medi...
3	related-1;request-1;offer-0;aid_related-1;medi...
4	related-1;request-0;offer-0;aid_related-0;medi...

### 1.0.2 3. Split categories into separate category columns.

- Split the values in the categories column on the ; character so that each value becomes a separate column. You'll find `this method` very helpful! Make sure to set `expand=True`.
- Use the first row of categories dataframe to create column names for the categories data.
- Rename columns of categories with new column names.

[47]: # create a dataframe of the 36 individual category columns

```
categories = df.categories.str.split(";", expand = True)
categories.head()
```

[47]:

	0	1	2	3	4 \
0	related-1	request-0	offer-0	aid_related-0	medical_help-0
1	related-1	request-0	offer-0	aid_related-1	medical_help-0
2	related-1	request-0	offer-0	aid_related-0	medical_help-0

```

3 related-1 request-1 offer-0 aid_related-1 medical_help-0
4 related-1 request-0 offer-0 aid_related-0 medical_help-0

      5           6           7           8   \
0 medical_products-0 search_and_rescue-0 security-0 military-0
1 medical_products-0 search_and_rescue-0 security-0 military-0
2 medical_products-0 search_and_rescue-0 security-0 military-0
3 medical_products-1 search_and_rescue-0 security-0 military-0
4 medical_products-0 search_and_rescue-0 security-0 military-0

      9   ...       26           27   \
0 child_alone-0   ... aid_centers-0 other_infrastructure-0
1 child_alone-0   ... aid_centers-0 other_infrastructure-0
2 child_alone-0   ... aid_centers-0 other_infrastructure-0
3 child_alone-0   ... aid_centers-0 other_infrastructure-0
4 child_alone-0   ... aid_centers-0 other_infrastructure-0

      28       29       30       31       32       33   \
0 weather_related-0 floods-0 storm-0 fire-0 earthquake-0 cold-0
1 weather_related-1 floods-0 storm-1 fire-0 earthquake-0 cold-0
2 weather_related-0 floods-0 storm-0 fire-0 earthquake-0 cold-0
3 weather_related-0 floods-0 storm-0 fire-0 earthquake-0 cold-0
4 weather_related-0 floods-0 storm-0 fire-0 earthquake-0 cold-0

      34       35
0 other_weather-0 direct_report-0
1 other_weather-0 direct_report-0
2 other_weather-0 direct_report-0
3 other_weather-0 direct_report-0
4 other_weather-0 direct_report-0

```

[5 rows x 36 columns]

```
[48]: # select the first row of the categories dataframe
row = categories.iloc[[0]]

# use this row to extract a list of new column names for categories.
# one way is to apply a lambda function that takes everything
# up to the second to last character of each string with slicing
category_colnames = row.apply(lambda x: x.str[:-2]).values.tolist()
print(category_colnames)
```

```
[['related', 'request', 'offer', 'aid_related', 'medical_help',
'medical_products', 'search_and_rescue', 'security', 'military', 'child_alone',
'water', 'food', 'shelter', 'clothing', 'money', 'missing_people', 'refugees',
'death', 'other_aid', 'infrastructure_related', 'transport', 'buildings',
'electricity', 'tools', 'hospitals', 'shops', 'aid_centers',
```

```
'other_infrastructure', 'weather_related', 'floods', 'storm', 'fire',
'earthquake', 'cold', 'other_weather', 'direct_report']]
```

```
[49]: # rename the columns of `categories`  
categories.columns = category_colnames  
categories.head()
```

```
[49]:      related    request    offer    aid_related    medical_help  \  
0  related-1  request-0  offer-0  aid_related-0  medical_help-0  
1  related-1  request-0  offer-0  aid_related-1  medical_help-0  
2  related-1  request-0  offer-0  aid_related-0  medical_help-0  
3  related-1  request-1  offer-0  aid_related-1  medical_help-0  
4  related-1  request-0  offer-0  aid_related-0  medical_help-0  
  
      medical_products    search_and_rescue    security    military  \  
0  medical_products-0  search_and_rescue-0  security-0  military-0  
1  medical_products-0  search_and_rescue-0  security-0  military-0  
2  medical_products-0  search_and_rescue-0  security-0  military-0  
3  medical_products-1  search_and_rescue-0  security-0  military-0  
4  medical_products-0  search_and_rescue-0  security-0  military-0  
  
      child_alone ...  aid_centers    other_infrastructure  \  
0  child_alone-0 ...  aid_centers-0  other_infrastructure-0  
1  child_alone-0 ...  aid_centers-0  other_infrastructure-0  
2  child_alone-0 ...  aid_centers-0  other_infrastructure-0  
3  child_alone-0 ...  aid_centers-0  other_infrastructure-0  
4  child_alone-0 ...  aid_centers-0  other_infrastructure-0  
  
      weather_related    floods    storm    fire    earthquake    cold  \  
0  weather_related-0  floods-0  storm-0  fire-0  earthquake-0  cold-0  
1  weather_related-1  floods-0  storm-1  fire-0  earthquake-0  cold-0  
2  weather_related-0  floods-0  storm-0  fire-0  earthquake-0  cold-0  
3  weather_related-0  floods-0  storm-0  fire-0  earthquake-0  cold-0  
4  weather_related-0  floods-0  storm-0  fire-0  earthquake-0  cold-0  
  
      other_weather    direct_report  
0  other_weather-0  direct_report-0  
1  other_weather-0  direct_report-0  
2  other_weather-0  direct_report-0  
3  other_weather-0  direct_report-0  
4  other_weather-0  direct_report-0  
  
[5 rows x 36 columns]
```

### 1.0.3 4. Convert category values to just numbers 0 or 1.

- Iterate through the category columns in df to keep only the last character of each string (the 1 or 0). For example, `related-0` becomes 0, `related-1` becomes 1. Convert the string to a

numeric value.

- You can perform [normal string actions on Pandas Series](#), like indexing, by including `.str` after the Series. You may need to first convert the Series to be of type string, which you can do with `astype(str)`.

```
[50]: for column in categories:  
    # set each value to be the last character of the string  
    categories[column] = categories[column].apply(lambda x: x[-1:])  
  
    # convert column from string to numeric  
    categories[column] = pd.to_numeric(categories[column])  
categories.head()
```

```
[50]: related request offer aid_related medical_help medical_products \\\n0      1      0      0      0      0      0  
1      1      0      0      1      0      0  
2      1      0      0      0      0      0  
3      1      1      0      1      0      1  
4      1      0      0      0      0      0  
  
search_and_rescue security military child_alone ... aid_centers \\\n0          0      0      0      0 ...      0  
1          0      0      0      0 ...      0  
2          0      0      0      0 ...      0  
3          0      0      0      0 ...      0  
4          0      0      0      0 ...      0  
  
other_infrastructure weather_related floods storm fire earthquake cold \\\n0          0          0      0      0      0      0      0      0  
1          0          1      0      1      0      0      0      0  
2          0          0      0      0      0      0      0      0  
3          0          0      0      0      0      0      0      0  
4          0          0      0      0      0      0      0      0  
  
other_weather direct_report  
0          0          0  
1          0          0  
2          0          0  
3          0          0  
4          0          0  
  
[5 rows x 36 columns]
```

#### 1.0.4 5. Replace categories column in df with new category columns.

- Drop the categories column from the df datafram since it is no longer needed.
- Concatenate df and categories data frames.

```
[51]: # drop the original categories column from `df`  
df.drop(columns='categories', inplace = True)  
df.head()
```

```
[51]: id message \  
0 2 Weather update - a cold front from Cuba that c...  
1 7 Is the Hurricane over or is it not over  
2 8 Looking for someone but no name  
3 9 UN reports Leogane 80-90 destroyed. Only Hospi...  
4 12 says: west side of Haiti, rest of the country ...  
  
original genre  
0 Un front froid se retrouve sur Cuba ce matin. ... direct  
1 Cyclone nan fini osinon li pa fini direct  
2 Patnm, di Maryani relem pou li banm nouvel li ... direct  
3 UN reports Leogane 80-90 destroyed. Only Hospi... direct  
4 facade ouest d Haiti et le reste du pays aujou... direct
```

```
[52]: categories.head(1)
```

```
[52]: related request offer aid_related medical_help medical_products \  
0 1 0 0 0 0  
  
search_and_rescue security military child_alone ... aid_centers \  
0 0 0 0 0 ... 0  
  
other_infrastructure weather_related floods storm fire earthquake cold \  
0 0 0 0 0 0 0 0  
  
other_weather direct_report  
0 0  
  
[1 rows x 36 columns]
```

```
[53]: # concatenate the original dataframe with the new `categories` dataframe  
df = pd.concat([df, categories], axis = 1)  
df.head()
```

```
[53]: id message \  
0 2 Weather update - a cold front from Cuba that c...  
1 7 Is the Hurricane over or is it not over  
2 8 Looking for someone but no name  
3 9 UN reports Leogane 80-90 destroyed. Only Hospi...  
4 12 says: west side of Haiti, rest of the country ...  
  
original genre (related,) \  
0 Un front froid se retrouve sur Cuba ce matin. ... direct 1
```

```

1             Cyclone nan fini osinon li pa fini direct          1
2 Patnm, di Maryani relem pou li bamn nouvel li ... direct      1
3 UN reports Leogane 80-90 destroyed. Only Hospi... direct      1
4 facade ouest d Haiti et le reste du pays aujou... direct      1

    (request,)  (offer,)  (aid_related,)  (medical_help,)  (medical_products,) \
0            0           0               0                  0                   0
1            0           0               1                  0                   0
2            0           0               0                  0                   0
3            1           0               1                  0                   1
4            0           0               0                  0                   0

    ...  (aid_centers,)  (other_infrastructure,)  (weather_related,) \
0 ...            0                   0                  0                   0
1 ...            0                   0                  0                   1
2 ...            0                   0                  0                   0
3 ...            0                   0                  0                   0
4 ...            0                   0                  0                   0

    (floods,)  (storm,)  (fire,)  (earthquake,)  (cold,)  (other_weather,) \
0            0           0           0               0           0                   0
1            0           1           0               0           0                   0
2            0           0           0               0           0                   0
3            0           0           0               0           0                   0
4            0           0           0               0           0                   0

    (direct_report,)
0            0
1            0
2            0
3            0
4            0

[5 rows x 40 columns]

```

```
[59]: # df.columns[4][0]
# cleaning up the column names
for i in range(len(df.columns)):
    if i > 3:
        df.rename(columns = {df.columns[i] : df.columns[i][0]}, inplace = True)
```

```
[62]: df.head(1)
```

```
[62]:   id                      message \
0    2 Weather update - a cold front from Cuba that c...
                                         original   genre   related \
```

```
0 Un front froid se retrouve sur Cuba ce matin. ... direct      1
    request   offer  aid_related  medical_help  medical_products  ... \
0          0       0           0             0                   ...
0 aid_centers  other_infrastructure  weather_related  floods  storm  fire  \
0           0                   0             0             0       0     0     0
0 earthquake  cold  other_weather  direct_report
0           0       0           0             0
[1 rows x 40 columns]
```

### 1.0.5 6. Remove duplicates.

- Check how many duplicates are in this dataset.
- Drop the duplicates.
- Confirm duplicates were removed.

```
[63]: # check number of duplicates
df.duplicated().sum()
```

```
[63]: 170
```

```
[64]: # drop duplicates
df = df.drop_duplicates()
```

```
[65]: # check number of duplicates
df.duplicated().sum()
```

```
[65]: 0
```

### 1.0.6 7. Save the clean dataset into an sqlite database.

You can do this with pandas `to_sql` method combined with the SQLAlchemy library. Remember to import SQLAlchemy's `create_engine` in the first cell of this notebook to use it below.

```
[66]: from sqlalchemy import create_engine
engine = create_engine('sqlite:///ETLPipelinePrep.db')
df.to_sql('Disasters_ETL', engine, index=False)
```

### 1.0.7 8. Use this notebook to complete `etl_pipeline.py`

Use the template file attached in the Resources folder to write a script that runs the steps above to create a database based on new datasets specified by the user. Alternatively, you can complete `etl_pipeline.py` in the classroom on the Project Workspace IDE coming later.

```
[ ]:
```

[ ]:

[ ]:

[ ]:

[ ]:

# ml-pipeline-preparation

February 24, 2024

## 1 ML Pipeline Preparation

Follow the instructions below to help you create your ML pipeline.

### 1. Import libraries and load data from database.

- Import Python libraries
- Load dataset from database with `read_sql_table`
- Define feature and target variables X and Y

```
[1]: # download necessary NLTK data
import nltk
nltk.download(['punkt', 'wordnet'])
nltk.download('stopwords')

[nltk_data] Downloading package punkt to
[nltk_data]      /Users/danielchang/nltk_data...
[nltk_data]  Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package wordnet to
[nltk_data]      /Users/danielchang/nltk_data...
[nltk_data]  Unzipping corpora/wordnet.zip.
[nltk_data] Downloading package stopwords to
[nltk_data]      /Users/danielchang/nltk_data...
[nltk_data]  Unzipping corpora/stopwords.zip.
```

```
[1]: True
```

```
[2]: # import statements
import re
import numpy as np
import pandas as pd
import warnings
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from sqlalchemy import create_engine
import pickle

from sklearn.metrics import confusion_matrix
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline, FeatureUnion
```

```

from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer
from sklearn.multioutput import MultiOutputClassifier
from nltk.corpus import stopwords
from sklearn.metrics import classification_report
from sklearn.ensemble import AdaBoostClassifier

warnings.simplefilter('ignore')

```

[3]: # load data from database  
engine = create\_engine('sqlite:///ETLPipelinePrep.db')  
df = pd.read\_sql\_table('Disasters\_ETL', engine)  
X = df.message  
y = df[df.columns[4:]]

[10]: df.head(3)

```

[10]:   id                               message \
0    2  Weather update - a cold front from Cuba that c...
1    7                  Is the Hurricane over or is it not over
2    8                  Looking for someone but no name

                                         original   genre   related \
0  Un front froid se retrouve sur Cuba ce matin. ...  direct      1
1          Cyclone nan fini osinon li pa fini  direct      1
2  Patnm, di Maryani relem pou li banm nouvel li ...  direct      1

      request   offer  aid_related  medical_help  medical_products  ... \
0          0      0            0            0                  0      ...
1          0      0            1            0                  0      ...
2          0      0            0            0                  0      ...

      aid_centers  other_infrastructure  weather_related  floods  storm  fire \
0            0                      0                  0      0      0      0
1            0                      0                  1      0      1      0
2            0                      0                  0      0      0      0

      earthquake  cold  other_weather  direct_report
0            0      0            0            0
1            0      0            0            0
2            0      0            0            0

[3 rows x 40 columns]

```

### 1.0.1 2. Write a tokenization function to process your text data

```
[4]: def tokenize(text):
    # Converting everything to lower case
    text = re.sub(r"[^a-zA-Z0-9]", " ", text.lower())

    # Tokenize words
    tokens = word_tokenize(text)

    # normalization word tokens and remove stop words
    stop_words = stopwords.words("english")
    lemmatizer = WordNetLemmatizer()

    normalized = [lemmatizer.lemmatize(word) for word in tokens if word not in
    ↪stop_words]

    return normalized
```

### 1.0.2 3. Build a machine learning pipeline

This machine pipeline should take in the `message` column as input and output classification results on the other 36 categories in the dataset. You may find the `MultiOutputClassifier` helpful for predicting multiple target variables.

```
[5]: pipeline = Pipeline([
    ('vect', CountVectorizer(tokenizer=tokenize)),
    ('tfidf', TfidfTransformer()),
    ('clf', MultiOutputClassifier(RandomForestClassifier()))
])
```

### 1.0.3 4. Train pipeline

- Split data into train and test sets
- Train pipeline

```
[6]: X_train, X_test, y_train, y_test = train_test_split(X,y)
pipeline.fit(X_train, y_train)
```

```
[6]: Pipeline(steps=[('vect',
                     CountVectorizer(tokenizer=<function tokenize at 0x125ba3430>)),
                     ('tfidf', TfidfTransformer()),
                     ('clf',
                     MultiOutputClassifier(estimator=RandomForestClassifier()))])
```

### 1.0.4 5. Test your model

Report the f1 score, precision and recall for each output category of the dataset. You can do this by iterating through the columns and calling sklearn's `classification_report` on each.

```
[13]: y_pred=pipeline.predict(X_test)
category_names = y.columns

for i in range(36):
    print(y_test.columns[i], ':')
    print(classification_report(y_test.iloc[:,i], y_pred[:,i])) #[:,i] is aimed
    ↪at the specific column and all rows
```

related :

	precision	recall	f1-score	support
0	0.67	0.41	0.51	1485
1	0.84	0.94	0.89	5020
2	0.74	0.29	0.41	49
accuracy			0.82	6554
macro avg	0.75	0.55	0.60	6554
weighted avg	0.80	0.82	0.80	6554

request :

	precision	recall	f1-score	support
0	0.90	0.98	0.94	5412
1	0.85	0.49	0.62	1142
accuracy			0.90	6554
macro avg	0.88	0.74	0.78	6554
weighted avg	0.89	0.90	0.88	6554

offer :

	precision	recall	f1-score	support
0	1.00	1.00	1.00	6529
1	0.00	0.00	0.00	25
accuracy			1.00	6554
macro avg	0.50	0.50	0.50	6554
weighted avg	0.99	1.00	0.99	6554

aid\_related :

	precision	recall	f1-score	support
0	0.79	0.84	0.81	3826
1	0.75	0.68	0.72	2728
accuracy			0.78	6554
macro avg	0.77	0.76	0.77	6554

weighted avg 0.77 0.78 0.77 6554

medical\_help :

	precision	recall	f1-score	support
0	0.92	1.00	0.96	6025
1	0.68	0.05	0.10	529
accuracy			0.92	6554
macro avg	0.80	0.53	0.53	6554
weighted avg	0.90	0.92	0.89	6554

medical\_products :

	precision	recall	f1-score	support
0	0.96	1.00	0.98	6245
1	0.65	0.06	0.10	309
accuracy			0.95	6554
macro avg	0.80	0.53	0.54	6554
weighted avg	0.94	0.95	0.94	6554

search\_and\_rescue :

	precision	recall	f1-score	support
0	0.97	1.00	0.99	6370
1	0.57	0.04	0.08	184
accuracy			0.97	6554
macro avg	0.77	0.52	0.53	6554
weighted avg	0.96	0.97	0.96	6554

security :

	precision	recall	f1-score	support
0	0.98	1.00	0.99	6441
1	0.00	0.00	0.00	113
accuracy			0.98	6554
macro avg	0.49	0.50	0.50	6554
weighted avg	0.97	0.98	0.97	6554

military :

	precision	recall	f1-score	support
0	0.97	1.00	0.98	6347
1	0.71	0.06	0.11	207

accuracy			0.97	6554
macro avg	0.84	0.53	0.55	6554
weighted avg	0.96	0.97	0.96	6554

child\_alone :

	precision	recall	f1-score	support
0	1.00	1.00	1.00	6554
accuracy			1.00	6554
macro avg	1.00	1.00	1.00	6554
weighted avg	1.00	1.00	1.00	6554

water :

	precision	recall	f1-score	support
0	0.96	1.00	0.98	6160
1	0.89	0.38	0.53	394
accuracy			0.96	6554
macro avg	0.92	0.69	0.75	6554
weighted avg	0.96	0.96	0.95	6554

food :

	precision	recall	f1-score	support
0	0.95	0.99	0.97	5845
1	0.84	0.59	0.69	709
accuracy			0.94	6554
macro avg	0.90	0.79	0.83	6554
weighted avg	0.94	0.94	0.94	6554

shelter :

	precision	recall	f1-score	support
0	0.94	0.99	0.97	5955
1	0.81	0.38	0.52	599
accuracy			0.94	6554
macro avg	0.88	0.69	0.74	6554
weighted avg	0.93	0.94	0.92	6554

clothing :

	precision	recall	f1-score	support
0	0.98	1.00	0.99	6444
1	0.89	0.07	0.13	110

accuracy			0.98	6554
macro avg	0.94	0.54	0.56	6554
weighted avg	0.98	0.98	0.98	6554

money :

	precision	recall	f1-score	support
0	0.98	1.00	0.99	6400
1	1.00	0.01	0.03	154

accuracy			0.98	6554
macro avg	0.99	0.51	0.51	6554
weighted avg	0.98	0.98	0.97	6554

missing\_people :

	precision	recall	f1-score	support
0	0.99	1.00	0.99	6477
1	0.75	0.04	0.07	77

accuracy			0.99	6554
macro avg	0.87	0.52	0.53	6554
weighted avg	0.99	0.99	0.98	6554

refugees :

	precision	recall	f1-score	support
0	0.97	1.00	0.98	6355
1	0.62	0.05	0.09	199

accuracy			0.97	6554
macro avg	0.80	0.52	0.54	6554
weighted avg	0.96	0.97	0.96	6554

death :

	precision	recall	f1-score	support
0	0.96	1.00	0.98	6235
1	0.86	0.19	0.32	319

accuracy			0.96	6554
macro avg	0.91	0.60	0.65	6554
weighted avg	0.96	0.96	0.95	6554

other\_aid :

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

0	0.87	1.00	0.93	5672
1	0.66	0.03	0.06	882
accuracy			0.87	6554
macro avg	0.76	0.51	0.49	6554
weighted avg	0.84	0.87	0.81	6554
<b>infrastructure_related :</b>				
	precision	recall	f1-score	support
0	0.93	1.00	0.97	6124
1	0.00	0.00	0.00	430
accuracy			0.93	6554
macro avg	0.47	0.50	0.48	6554
weighted avg	0.87	0.93	0.90	6554
<b>transport :</b>				
	precision	recall	f1-score	support
0	0.96	1.00	0.98	6242
1	0.77	0.11	0.19	312
accuracy			0.96	6554
macro avg	0.87	0.55	0.58	6554
weighted avg	0.95	0.96	0.94	6554
<b>buildings :</b>				
	precision	recall	f1-score	support
0	0.95	1.00	0.97	6196
1	0.72	0.12	0.21	358
accuracy			0.95	6554
macro avg	0.83	0.56	0.59	6554
weighted avg	0.94	0.95	0.93	6554
<b>electricity :</b>				
	precision	recall	f1-score	support
0	0.98	1.00	0.99	6416
1	0.56	0.04	0.07	138
accuracy			0.98	6554
macro avg	0.77	0.52	0.53	6554
weighted avg	0.97	0.98	0.97	6554
<b>tools :</b>				

	precision	recall	f1-score	support
0	0.99	1.00	1.00	6512
1	0.00	0.00	0.00	42
accuracy			0.99	6554
macro avg	0.50	0.50	0.50	6554
weighted avg	0.99	0.99	0.99	6554
hospitals :				
	precision	recall	f1-score	support
0	0.99	1.00	0.99	6486
1	0.00	0.00	0.00	68
accuracy			0.99	6554
macro avg	0.49	0.50	0.50	6554
weighted avg	0.98	0.99	0.98	6554
shops :				
	precision	recall	f1-score	support
0	0.99	1.00	1.00	6521
1	0.00	0.00	0.00	33
accuracy			0.99	6554
macro avg	0.50	0.50	0.50	6554
weighted avg	0.99	0.99	0.99	6554
aid_centers :				
	precision	recall	f1-score	support
0	0.99	1.00	0.99	6476
1	0.00	0.00	0.00	78
accuracy			0.99	6554
macro avg	0.49	0.50	0.50	6554
weighted avg	0.98	0.99	0.98	6554
other_infrastructure :				
	precision	recall	f1-score	support
0	0.96	1.00	0.98	6260
1	0.00	0.00	0.00	294
accuracy			0.95	6554
macro avg	0.48	0.50	0.49	6554
weighted avg	0.91	0.95	0.93	6554

**weather\_related :**

	precision	recall	f1-score	support
0	0.90	0.95	0.93	4776
1	0.86	0.73	0.79	1778
accuracy			0.89	6554
macro avg	0.88	0.84	0.86	6554
weighted avg	0.89	0.89	0.89	6554

**floods :**

	precision	recall	f1-score	support
0	0.95	1.00	0.97	6058
1	0.90	0.42	0.57	496
accuracy			0.95	6554
macro avg	0.93	0.71	0.77	6554
weighted avg	0.95	0.95	0.94	6554

**storm :**

	precision	recall	f1-score	support
0	0.96	0.98	0.97	5954
1	0.77	0.55	0.64	600
accuracy			0.94	6554
macro avg	0.87	0.77	0.81	6554
weighted avg	0.94	0.94	0.94	6554

**fire :**

	precision	recall	f1-score	support
0	0.99	1.00	0.99	6487
1	1.00	0.01	0.03	67
accuracy			0.99	6554
macro avg	0.99	0.51	0.51	6554
weighted avg	0.99	0.99	0.99	6554

**earthquake :**

	precision	recall	f1-score	support
0	0.98	0.99	0.98	5908
1	0.90	0.79	0.84	646
accuracy			0.97	6554

macro avg	0.94	0.89	0.91	6554
weighted avg	0.97	0.97	0.97	6554
 cold :				
	precision	recall	f1-score	support
0	0.98	1.00	0.99	6423
1	0.88	0.11	0.19	131
accuracy			0.98	6554
macro avg	0.93	0.55	0.59	6554
weighted avg	0.98	0.98	0.97	6554
 other_weather :				
	precision	recall	f1-score	support
0	0.95	1.00	0.97	6206
1	0.56	0.01	0.03	348
accuracy			0.95	6554
macro avg	0.75	0.51	0.50	6554
weighted avg	0.93	0.95	0.92	6554
 direct_report :				
	precision	recall	f1-score	support
0	0.86	0.98	0.92	5259
1	0.82	0.36	0.50	1295
accuracy			0.86	6554
macro avg	0.84	0.67	0.71	6554
weighted avg	0.85	0.86	0.83	6554

## 1.0.5 6. Improve your model

Use grid search to find better parameters.

[14]: pipeline.get\_params()

```
[14]: {'memory': None,
    'steps': [('vect',
        CountVectorizer(tokenizer=<function tokenize at 0x125ba3430>)),
    ('tfidf', TfidfTransformer()),
    ('clf', MultiOutputClassifier(estimator=RandomForestClassifier()))],
    'verbose': False,
    'vect': CountVectorizer(tokenizer=<function tokenize at 0x125ba3430>),
    'tfidf': TfidfTransformer(),
```

```

'clf': MultiOutputClassifier(estimator=RandomForestClassifier()),
'vect__analyzer': 'word',
'vect__binary': False,
'vect__decode_error': 'strict',
'vect__dtype': numpy.int64,
'vect__encoding': 'utf-8',
'vect__input': 'content',
'vect__lowercase': True,
'vect__max_df': 1.0,
'vect__max_features': None,
'vect__min_df': 1,
'vect__ngram_range': (1, 1),
'vect__preprocessor': None,
'vect__stop_words': None,
'vect__strip_accents': None,
'vect__token_pattern': '(?u)\\b\\w\\w+\\b',
'vect__tokenizer': <function __main__. tokenize(text)>,
'vect__vocabulary': None,
'tfidf__norm': 'l2',
'tfidf__smooth_idf': True,
'tfidf__sublinear_tf': False,
'tfidf__use_idf': True,
'clf__estimator__bootstrap': True,
'clf__estimator__ccp_alpha': 0.0,
'clf__estimator__class_weight': None,
'clf__estimator__criterion': 'gini',
'clf__estimator__max_depth': None,
'clf__estimator__max_features': 'auto',
'clf__estimator__max_leaf_nodes': None,
'clf__estimator__max_samples': None,
'clf__estimator__min_impurity_decrease': 0.0,
'clf__estimator__min_samples_leaf': 1,
'clf__estimator__min_samples_split': 2,
'clf__estimator__min_weight_fraction_leaf': 0.0,
'clf__estimator__n_estimators': 100,
'clf__estimator__n_jobs': None,
'clf__estimator__oob_score': False,
'clf__estimator__random_state': None,
'clf__estimator__verbose': 0,
'clf__estimator__warm_start': False,
'clf__estimator': RandomForestClassifier(),
'clf__n_jobs': None}

```

[19]: # GridSearchCV help: [https://scikit-learn.org/stable/modules/generated/sklearn.model\\_selection.GridSearchCV.html](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html)

```

parameters = {
    #'vect__ngram_range': ((1, 1),(1,2)),

```

```

        'clf__estimator__n_estimators': [50, 100],
        'clf__estimator__min_samples_split': [2, 3, 4],

    }

cv = GridSearchCV(pipeline, param_grid=parameters, verbose=10)

```

[20]: cv.fit(X\_train, y\_train)

```

Fitting 5 folds for each of 6 candidates, totalling 30 fits
[CV 1/5; 1/6] START clf__estimator__min_samples_split=2,
clf__estimator__n_estimators=50
[CV 1/5; 1/6] END clf__estimator__min_samples_split=2,
clf__estimator__n_estimators=50;, score=0.258 total time= 2.1min
[CV 2/5; 1/6] START clf__estimator__min_samples_split=2,
clf__estimator__n_estimators=50
[CV 2/5; 1/6] END clf__estimator__min_samples_split=2,
clf__estimator__n_estimators=50;, score=0.266 total time= 2.1min
[CV 3/5; 1/6] START clf__estimator__min_samples_split=2,
clf__estimator__n_estimators=50
[CV 3/5; 1/6] END clf__estimator__min_samples_split=2,
clf__estimator__n_estimators=50;, score=0.261 total time= 2.0min
[CV 4/5; 1/6] START clf__estimator__min_samples_split=2,
clf__estimator__n_estimators=50
[CV 4/5; 1/6] END clf__estimator__min_samples_split=2,
clf__estimator__n_estimators=50;, score=0.253 total time= 1.9min
[CV 5/5; 1/6] START clf__estimator__min_samples_split=2,
clf__estimator__n_estimators=50
[CV 5/5; 1/6] END clf__estimator__min_samples_split=2,
clf__estimator__n_estimators=50;, score=0.268 total time= 1.9min
[CV 1/5; 2/6] START clf__estimator__min_samples_split=2,
clf__estimator__n_estimators=100
[CV 1/5; 2/6] END clf__estimator__min_samples_split=2,
clf__estimator__n_estimators=100;, score=0.254 total time= 3.8min
[CV 2/5; 2/6] START clf__estimator__min_samples_split=2,
clf__estimator__n_estimators=100
[CV 2/5; 2/6] END clf__estimator__min_samples_split=2,
clf__estimator__n_estimators=100;, score=0.270 total time= 3.7min
[CV 3/5; 2/6] START clf__estimator__min_samples_split=2,
clf__estimator__n_estimators=100
[CV 3/5; 2/6] END clf__estimator__min_samples_split=2,
clf__estimator__n_estimators=100;, score=0.257 total time= 3.8min
[CV 4/5; 2/6] START clf__estimator__min_samples_split=2,
clf__estimator__n_estimators=100
[CV 4/5; 2/6] END clf__estimator__min_samples_split=2,
clf__estimator__n_estimators=100;, score=0.259 total time= 3.9min
[CV 5/5; 2/6] START clf__estimator__min_samples_split=2,

```

```

clf__estimator__n_estimators=100
[CV 5/5; 2/6] END clf__estimator__min_samples_split=2,
clf__estimator__n_estimators=100;, score=0.274 total time= 3.9min
[CV 1/5; 3/6] START clf__estimator__min_samples_split=3,
clf__estimator__n_estimators=50
[CV 1/5; 3/6] END clf__estimator__min_samples_split=3,
clf__estimator__n_estimators=50;, score=0.258 total time= 1.7min
[CV 2/5; 3/6] START clf__estimator__min_samples_split=3,
clf__estimator__n_estimators=50
[CV 2/5; 3/6] END clf__estimator__min_samples_split=3,
clf__estimator__n_estimators=50;, score=0.268 total time= 1.9min
[CV 3/5; 3/6] START clf__estimator__min_samples_split=3,
clf__estimator__n_estimators=50
[CV 3/5; 3/6] END clf__estimator__min_samples_split=3,
clf__estimator__n_estimators=50;, score=0.257 total time= 1.8min
[CV 4/5; 3/6] START clf__estimator__min_samples_split=3,
clf__estimator__n_estimators=50
[CV 4/5; 3/6] END clf__estimator__min_samples_split=3,
clf__estimator__n_estimators=50;, score=0.258 total time= 1.7min
[CV 5/5; 3/6] START clf__estimator__min_samples_split=3,
clf__estimator__n_estimators=50
[CV 5/5; 3/6] END clf__estimator__min_samples_split=3,
clf__estimator__n_estimators=50;, score=0.271 total time= 1.7min
[CV 1/5; 4/6] START clf__estimator__min_samples_split=3,
clf__estimator__n_estimators=100
[CV 1/5; 4/6] END clf__estimator__min_samples_split=3,
clf__estimator__n_estimators=100;, score=0.258 total time= 3.3min
[CV 2/5; 4/6] START clf__estimator__min_samples_split=3,
clf__estimator__n_estimators=100
[CV 2/5; 4/6] END clf__estimator__min_samples_split=3,
clf__estimator__n_estimators=100;, score=0.269 total time= 3.3min
[CV 3/5; 4/6] START clf__estimator__min_samples_split=3,
clf__estimator__n_estimators=100
[CV 3/5; 4/6] END clf__estimator__min_samples_split=3,
clf__estimator__n_estimators=100;, score=0.254 total time= 3.3min
[CV 4/5; 4/6] START clf__estimator__min_samples_split=3,
clf__estimator__n_estimators=100
[CV 4/5; 4/6] END clf__estimator__min_samples_split=3,
clf__estimator__n_estimators=100;, score=0.260 total time= 3.3min
[CV 5/5; 4/6] START clf__estimator__min_samples_split=3,
clf__estimator__n_estimators=100
[CV 5/5; 4/6] END clf__estimator__min_samples_split=3,
clf__estimator__n_estimators=100;, score=0.277 total time= 3.4min
[CV 1/5; 5/6] START clf__estimator__min_samples_split=4,
clf__estimator__n_estimators=50
[CV 1/5; 5/6] END clf__estimator__min_samples_split=4,
clf__estimator__n_estimators=50;, score=0.257 total time= 1.6min
[CV 2/5; 5/6] START clf__estimator__min_samples_split=4,

```

```

clf__estimator__n_estimators=50
[CV 2/5; 5/6] END clf__estimator__min_samples_split=4,
clf__estimator__n_estimators=50;, score=0.270 total time= 1.6min
[CV 3/5; 5/6] START clf__estimator__min_samples_split=4,
clf__estimator__n_estimators=50
[CV 3/5; 5/6] END clf__estimator__min_samples_split=4,
clf__estimator__n_estimators=50;, score=0.252 total time= 1.6min
[CV 4/5; 5/6] START clf__estimator__min_samples_split=4,
clf__estimator__n_estimators=50
[CV 4/5; 5/6] END clf__estimator__min_samples_split=4,
clf__estimator__n_estimators=50;, score=0.246 total time= 1.6min
[CV 5/5; 5/6] START clf__estimator__min_samples_split=4,
clf__estimator__n_estimators=50
[CV 5/5; 5/6] END clf__estimator__min_samples_split=4,
clf__estimator__n_estimators=50;, score=0.271 total time= 1.8min
[CV 1/5; 6/6] START clf__estimator__min_samples_split=4,
clf__estimator__n_estimators=100
[CV 1/5; 6/6] END clf__estimator__min_samples_split=4,
clf__estimator__n_estimators=100;, score=0.255 total time= 3.2min
[CV 2/5; 6/6] START clf__estimator__min_samples_split=4,
clf__estimator__n_estimators=100
[CV 2/5; 6/6] END clf__estimator__min_samples_split=4,
clf__estimator__n_estimators=100;, score=0.267 total time= 3.1min
[CV 3/5; 6/6] START clf__estimator__min_samples_split=4,
clf__estimator__n_estimators=100
[CV 3/5; 6/6] END clf__estimator__min_samples_split=4,
clf__estimator__n_estimators=100;, score=0.255 total time= 3.1min
[CV 4/5; 6/6] START clf__estimator__min_samples_split=4,
clf__estimator__n_estimators=100
[CV 4/5; 6/6] END clf__estimator__min_samples_split=4,
clf__estimator__n_estimators=100;, score=0.263 total time= 3.0min
[CV 5/5; 6/6] START clf__estimator__min_samples_split=4,
clf__estimator__n_estimators=100
[CV 5/5; 6/6] END clf__estimator__min_samples_split=4,
clf__estimator__n_estimators=100;, score=0.274 total time= 3.1min

```

```

[20]: GridSearchCV(estimator=Pipeline(steps=[('vect',
                                              CountVectorizer(tokenizer=<function
                                                tokenize at 0x125ba3430>)),
                                              ('tfidf', TfidfTransformer()),
                                              ('clf',
                                               MultiOutputClassifier(estimator=RandomForestClassifier()))]),
                           param_grid={'clf__estimator__min_samples_split': [2, 3, 4],
                                       'clf__estimator__n_estimators': [50, 100]},
                           verbose=10)

```

```
[21]: #Youtube help - https://www.youtube.com/watch?v=TvB_3jVIHhg
print(cv.best_params_)

{'clf__estimator__min_samples_split': 3, 'clf__estimator__n_estimators': 100}

[22]: print(cv.best_estimator_)

Pipeline(steps=[('vect',
                 CountVectorizer(tokenizer=<function tokenize at 0x125ba3430>)),
               ('tfidf', TfidfTransformer()),
               ('clf',
                 MultiOutputClassifier(estimator=RandomForestClassifier(min_samples_split=3)))])
```

### 1.0.6 7. Test your model

Show the accuracy, precision, and recall of the tuned model.

Since this project focuses on code quality, process, and pipelines, there is no minimum performance metric needed to pass. However, make sure to fine tune your models for accuracy, precision and recall to make your project stand out - especially for your portfolio!

```
[24]: # built a new model, using best_estimator_
# https://stackoverflow.com/questions/45074698/
    ↪how-to-pass-elegantly-sklearns-gridsearchcv-best-parameters-to-another-model
tuned_model = cv.best_estimator_
tuned_pred = tuned_model.predict(X_test)

for i in range(36):
    print(y_test.columns[i], ':')
    print(classification_report(y_test.iloc[:,i], tuned_pred[:,i]))
```

related :

	precision	recall	f1-score	support
0	0.68	0.40	0.50	1485
1	0.84	0.94	0.89	5020
2	0.67	0.29	0.40	49
accuracy			0.81	6554
macro avg	0.73	0.54	0.60	6554
weighted avg	0.80	0.81	0.80	6554

request :

	precision	recall	f1-score	support
0	0.90	0.98	0.94	5412
1	0.84	0.48	0.61	1142
accuracy			0.89	6554

macro avg	0.87	0.73	0.77	6554
weighted avg	0.89	0.89	0.88	6554

offer :

	precision	recall	f1-score	support
0	1.00	1.00	1.00	6529
1	0.00	0.00	0.00	25
accuracy			1.00	6554
macro avg	0.50	0.50	0.50	6554
weighted avg	0.99	1.00	0.99	6554

aid\_related :

	precision	recall	f1-score	support
0	0.79	0.84	0.82	3826
1	0.75	0.69	0.72	2728
accuracy			0.78	6554
macro avg	0.77	0.76	0.77	6554
weighted avg	0.78	0.78	0.78	6554

medical\_help :

	precision	recall	f1-score	support
0	0.92	1.00	0.96	6025
1	0.65	0.08	0.14	529
accuracy			0.92	6554
macro avg	0.79	0.54	0.55	6554
weighted avg	0.90	0.92	0.89	6554

medical\_products :

	precision	recall	f1-score	support
0	0.96	1.00	0.98	6245
1	0.63	0.06	0.10	309
accuracy			0.95	6554
macro avg	0.79	0.53	0.54	6554
weighted avg	0.94	0.95	0.94	6554

search\_and\_rescue :

	precision	recall	f1-score	support
0	0.97	1.00	0.99	6370
1	0.67	0.05	0.10	184

accuracy			0.97	6554
macro avg	0.82	0.53	0.54	6554
weighted avg	0.96	0.97	0.96	6554

security :

	precision	recall	f1-score	support
0	0.98	1.00	0.99	6441
1	0.00	0.00	0.00	113

accuracy			0.98	6554
macro avg	0.49	0.50	0.50	6554
weighted avg	0.97	0.98	0.97	6554

military :

	precision	recall	f1-score	support
0	0.97	1.00	0.98	6347
1	0.64	0.08	0.14	207

accuracy			0.97	6554
macro avg	0.81	0.54	0.56	6554
weighted avg	0.96	0.97	0.96	6554

child\_alone :

	precision	recall	f1-score	support
0	1.00	1.00	1.00	6554

accuracy			1.00	6554
macro avg	1.00	1.00	1.00	6554
weighted avg	1.00	1.00	1.00	6554

water :

	precision	recall	f1-score	support
0	0.96	1.00	0.98	6160
1	0.88	0.41	0.56	394

accuracy			0.96	6554
macro avg	0.92	0.70	0.77	6554
weighted avg	0.96	0.96	0.95	6554

food :

	precision	recall	f1-score	support
0	0.96	0.98	0.97	5845

1	0.83	0.63	0.72	709
accuracy			0.95	6554
macro avg	0.90	0.81	0.84	6554
weighted avg	0.94	0.95	0.94	6554
<b>shelter :</b>				
	precision	recall	f1-score	support
0	0.94	0.99	0.96	5955
1	0.79	0.36	0.49	599
accuracy			0.93	6554
macro avg	0.87	0.67	0.73	6554
weighted avg	0.93	0.93	0.92	6554
<b>clothing :</b>				
	precision	recall	f1-score	support
0	0.98	1.00	0.99	6444
1	0.83	0.05	0.09	110
accuracy			0.98	6554
macro avg	0.91	0.52	0.54	6554
weighted avg	0.98	0.98	0.98	6554
<b>money :</b>				
	precision	recall	f1-score	support
0	0.98	1.00	0.99	6400
1	1.00	0.01	0.01	154
accuracy			0.98	6554
macro avg	0.99	0.50	0.50	6554
weighted avg	0.98	0.98	0.97	6554
<b>missing_people :</b>				
	precision	recall	f1-score	support
0	0.99	1.00	0.99	6477
1	0.75	0.04	0.07	77
accuracy			0.99	6554
macro avg	0.87	0.52	0.53	6554
weighted avg	0.99	0.99	0.98	6554
<b>refugees :</b>				
	precision	recall	f1-score	support

0	0.97	1.00	0.99	6355
1	0.77	0.05	0.09	199
accuracy			0.97	6554
macro avg	0.87	0.52	0.54	6554
weighted avg	0.96	0.97	0.96	6554
 death :				
	precision	recall	f1-score	support
0	0.96	1.00	0.98	6235
1	0.88	0.20	0.32	319
accuracy			0.96	6554
macro avg	0.92	0.60	0.65	6554
weighted avg	0.96	0.96	0.95	6554
 other_aid :				
	precision	recall	f1-score	support
0	0.87	1.00	0.93	5672
1	0.65	0.03	0.06	882
accuracy			0.87	6554
macro avg	0.76	0.52	0.50	6554
weighted avg	0.84	0.87	0.81	6554
 infrastructure_related :				
	precision	recall	f1-score	support
0	0.93	1.00	0.97	6124
1	0.00	0.00	0.00	430
accuracy			0.93	6554
macro avg	0.47	0.50	0.48	6554
weighted avg	0.87	0.93	0.90	6554
 transport :				
	precision	recall	f1-score	support
0	0.96	1.00	0.98	6242
1	0.72	0.11	0.19	312
accuracy			0.96	6554
macro avg	0.84	0.55	0.58	6554
weighted avg	0.95	0.96	0.94	6554

buildings :				
	precision	recall	f1-score	support
0	0.95	1.00	0.97	6196
1	0.69	0.11	0.20	358
accuracy			0.95	6554
macro avg	0.82	0.56	0.59	6554
weighted avg	0.94	0.95	0.93	6554
electricity :				
	precision	recall	f1-score	support
0	0.98	1.00	0.99	6416
1	0.83	0.04	0.07	138
accuracy			0.98	6554
macro avg	0.91	0.52	0.53	6554
weighted avg	0.98	0.98	0.97	6554
tools :				
	precision	recall	f1-score	support
0	0.99	1.00	1.00	6512
1	0.00	0.00	0.00	42
accuracy			0.99	6554
macro avg	0.50	0.50	0.50	6554
weighted avg	0.99	0.99	0.99	6554
hospitals :				
	precision	recall	f1-score	support
0	0.99	1.00	0.99	6486
1	0.00	0.00	0.00	68
accuracy			0.99	6554
macro avg	0.49	0.50	0.50	6554
weighted avg	0.98	0.99	0.98	6554
shops :				
	precision	recall	f1-score	support
0	0.99	1.00	1.00	6521
1	0.00	0.00	0.00	33
accuracy			0.99	6554
macro avg	0.50	0.50	0.50	6554

weighted avg 0.99 0.99 0.99 6554

aid\_centers :

	precision	recall	f1-score	support
0	0.99	1.00	0.99	6476
1	0.00	0.00	0.00	78
accuracy			0.99	6554
macro avg	0.49	0.50	0.50	6554
weighted avg	0.98	0.99	0.98	6554

other\_infrastructure :

	precision	recall	f1-score	support
0	0.96	1.00	0.98	6260
1	0.00	0.00	0.00	294
accuracy			0.95	6554
macro avg	0.48	0.50	0.49	6554
weighted avg	0.91	0.95	0.93	6554

weather\_related :

	precision	recall	f1-score	support
0	0.90	0.95	0.93	4776
1	0.85	0.73	0.78	1778
accuracy			0.89	6554
macro avg	0.88	0.84	0.86	6554
weighted avg	0.89	0.89	0.89	6554

floods :

	precision	recall	f1-score	support
0	0.96	1.00	0.98	6058
1	0.90	0.44	0.59	496
accuracy			0.95	6554
macro avg	0.93	0.72	0.78	6554
weighted avg	0.95	0.95	0.95	6554

storm :

	precision	recall	f1-score	support
0	0.96	0.98	0.97	5954
1	0.77	0.55	0.64	600

accuracy			0.94	6554
macro avg	0.86	0.77	0.80	6554
weighted avg	0.94	0.94	0.94	6554

fire :

	precision	recall	f1-score	support
0	0.99	1.00	1.00	6487
1	1.00	0.03	0.06	67

accuracy			0.99	6554
macro avg	1.00	0.51	0.53	6554
weighted avg	0.99	0.99	0.99	6554

earthquake :

	precision	recall	f1-score	support
0	0.98	0.99	0.99	5908
1	0.90	0.83	0.87	646

accuracy			0.97	6554
macro avg	0.94	0.91	0.93	6554
weighted avg	0.97	0.97	0.97	6554

cold :

	precision	recall	f1-score	support
0	0.98	1.00	0.99	6423
1	0.87	0.10	0.18	131

accuracy			0.98	6554
macro avg	0.92	0.55	0.58	6554
weighted avg	0.98	0.98	0.97	6554

other\_weather :

	precision	recall	f1-score	support
0	0.95	1.00	0.97	6206
1	0.74	0.04	0.08	348

accuracy			0.95	6554
macro avg	0.84	0.52	0.52	6554
weighted avg	0.94	0.95	0.93	6554

direct\_report :

	precision	recall	f1-score	support
0	0.86	0.98	0.92	5259

1	0.81	0.37	0.50	1295
accuracy			0.86	6554
macro avg	0.84	0.67	0.71	6554
weighted avg	0.85	0.86	0.84	6554

### 1.0.7 8. Try improving your model further. Here are a few ideas:

- try other machine learning algorithms
- add other features besides the TF-IDF

```
[25]: # AdaBoost Classifier help: https://scikit-learn.org/stable/modules/generated/
      ↪sklearn.ensemble.AdaBoostClassifier.html
new_pipeline = Pipeline([
    ('features', FeatureUnion([
        ('text_pipeline', Pipeline([
            ('vect', CountVectorizer(tokenizer=tokenize)),
            ('tfidf', TfidfTransformer())
        ])),
        ('clf', MultiOutputClassifier(AdaBoostClassifier(n_estimators = 100)))
    ])
])
```

```
[26]: X_train, X_test, y_train, y_test = train_test_split(X,y)
new_pipeline.fit(X_train, y_train)
```

```
[26]: Pipeline(steps=[('features',
                      FeatureUnion(transformer_list=[('text_pipeline',
                                                     Pipeline(steps=[('vect',
                                                                    CountVectorizer(tokenizer=<function tokenize at 0x125ba3430>)),
                                                     ('tfidf',
                                                      TfidfTransformer()))]]),
                      ('clf',
                       MultiOutputClassifier(estimator=AdaBoostClassifier(n_estimators=100)))]))
```

```
[27]: y_pred_new = new_pipeline.predict(X_test)

for i in range(36):
    print(y_test.columns[i], ':')
    print(classification_report(y_test.iloc[:,i], y_pred_new[:,i]))
```

```
related :
precision    recall   f1-score   support
```

0	0.61	0.16	0.25	1506
1	0.79	0.97	0.87	4994
2	0.36	0.17	0.23	54
accuracy			0.78	6554
macro avg	0.59	0.43	0.45	6554
weighted avg	0.74	0.78	0.72	6554

request :

	precision	recall	f1-score	support
0	0.91	0.96	0.94	5442
1	0.75	0.55	0.63	1112
accuracy			0.89	6554
macro avg	0.83	0.75	0.79	6554
weighted avg	0.89	0.89	0.89	6554

offer :

	precision	recall	f1-score	support
0	1.00	1.00	1.00	6522
1	0.00	0.00	0.00	32
accuracy			0.99	6554
macro avg	0.50	0.50	0.50	6554
weighted avg	0.99	0.99	0.99	6554

aid\_related :

	precision	recall	f1-score	support
0	0.78	0.86	0.82	3851
1	0.77	0.65	0.70	2703
accuracy			0.77	6554
macro avg	0.77	0.76	0.76	6554
weighted avg	0.77	0.77	0.77	6554

medical\_help :

	precision	recall	f1-score	support
0	0.94	0.98	0.96	6056
1	0.55	0.29	0.38	498
accuracy			0.93	6554
macro avg	0.75	0.64	0.67	6554
weighted avg	0.91	0.93	0.92	6554

medical\_products :

	precision	recall	f1-score	support
0	0.96	0.99	0.98	6204
1	0.65	0.33	0.44	350
accuracy			0.95	6554
macro avg	0.81	0.66	0.71	6554
weighted avg	0.95	0.95	0.95	6554

search\_and\_rescue :

	precision	recall	f1-score	support
0	0.98	0.99	0.99	6373
1	0.48	0.17	0.25	181
accuracy			0.97	6554
macro avg	0.73	0.58	0.62	6554
weighted avg	0.96	0.97	0.97	6554

security :

	precision	recall	f1-score	support
0	0.98	1.00	0.99	6451
1	0.05	0.01	0.02	103
accuracy			0.98	6554
macro avg	0.52	0.50	0.50	6554
weighted avg	0.97	0.98	0.98	6554

military :

	precision	recall	f1-score	support
0	0.98	0.99	0.98	6346
1	0.56	0.30	0.39	208
accuracy			0.97	6554
macro avg	0.77	0.65	0.69	6554
weighted avg	0.96	0.97	0.97	6554

child\_alone :

	precision	recall	f1-score	support
0	1.00	1.00	1.00	6554
accuracy			1.00	6554
macro avg	1.00	1.00	1.00	6554
weighted avg	1.00	1.00	1.00	6554

water :

	precision	recall	f1-score	support
0	0.98	0.98	0.98	6128
1	0.74	0.64	0.68	426
accuracy			0.96	6554
macro avg	0.86	0.81	0.83	6554
weighted avg	0.96	0.96	0.96	6554

food :

	precision	recall	f1-score	support
0	0.96	0.98	0.97	5845
1	0.79	0.69	0.74	709
accuracy			0.95	6554
macro avg	0.88	0.83	0.85	6554
weighted avg	0.94	0.95	0.94	6554

shelter :

	precision	recall	f1-score	support
0	0.96	0.98	0.97	5986
1	0.73	0.52	0.61	568
accuracy			0.94	6554
macro avg	0.84	0.75	0.79	6554
weighted avg	0.94	0.94	0.94	6554

clothing :

	precision	recall	f1-score	support
0	0.99	1.00	0.99	6455
1	0.63	0.45	0.53	99
accuracy			0.99	6554
macro avg	0.81	0.73	0.76	6554
weighted avg	0.99	0.99	0.99	6554

money :

	precision	recall	f1-score	support
0	0.98	0.99	0.99	6407
1	0.48	0.28	0.35	147
accuracy			0.98	6554

macro avg	0.73	0.64	0.67	6554
weighted avg	0.97	0.98	0.97	6554

missing\_people :

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

0	0.99	1.00	0.99	6491
1	0.35	0.13	0.19	63

accuracy			0.99	6554
macro avg	0.67	0.56	0.59	6554
weighted avg	0.99	0.99	0.99	6554

refugees :

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

0	0.97	0.99	0.98	6308
1	0.58	0.26	0.36	246

accuracy			0.97	6554
macro avg	0.78	0.63	0.67	6554
weighted avg	0.96	0.97	0.96	6554

death :

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

0	0.97	0.99	0.98	6266
1	0.67	0.40	0.50	288

accuracy			0.96	6554
macro avg	0.82	0.70	0.74	6554
weighted avg	0.96	0.96	0.96	6554

other\_aid :

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

0	0.88	0.97	0.93	5674
1	0.49	0.17	0.25	880

accuracy			0.87	6554
macro avg	0.69	0.57	0.59	6554
weighted avg	0.83	0.87	0.84	6554

infrastructure\_related :

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

0	0.94	0.99	0.97	6134
1	0.45	0.13	0.20	420

accuracy			0.93	6554
macro avg	0.69	0.56	0.58	6554
weighted avg	0.91	0.93	0.92	6554

transport :

	precision	recall	f1-score	support
0	0.97	0.99	0.98	6264
1	0.63	0.24	0.35	290
accuracy			0.96	6554
macro avg	0.80	0.62	0.66	6554
weighted avg	0.95	0.96	0.95	6554

buildings :

	precision	recall	f1-score	support
0	0.96	0.99	0.98	6207
1	0.68	0.36	0.47	347
accuracy			0.96	6554
macro avg	0.82	0.67	0.72	6554
weighted avg	0.95	0.96	0.95	6554

electricity :

	precision	recall	f1-score	support
0	0.99	0.99	0.99	6425
1	0.38	0.25	0.30	129
accuracy			0.98	6554
macro avg	0.68	0.62	0.64	6554
weighted avg	0.97	0.98	0.97	6554

tools :

	precision	recall	f1-score	support
0	1.00	1.00	1.00	6522
1	0.08	0.03	0.05	32
accuracy			0.99	6554
macro avg	0.54	0.51	0.52	6554
weighted avg	0.99	0.99	0.99	6554

hospitals :

precision	recall	f1-score	support
-----------	--------	----------	---------

	0	0.99	1.00	0.99	6479
	1	0.17	0.05	0.08	75
accuracy				0.99	6554
macro avg		0.58	0.53	0.54	6554
weighted avg		0.98	0.99	0.98	6554
<b>shops :</b>					
		precision	recall	f1-score	support
	0	1.00	1.00	1.00	6527
	1	0.11	0.04	0.06	27
accuracy				0.99	6554
macro avg		0.55	0.52	0.53	6554
weighted avg		0.99	0.99	0.99	6554
<b>aid_centers :</b>					
		precision	recall	f1-score	support
	0	0.99	1.00	0.99	6458
	1	0.25	0.08	0.12	96
accuracy				0.98	6554
macro avg		0.62	0.54	0.56	6554
weighted avg		0.98	0.98	0.98	6554
<b>other_infrastructure :</b>					
		precision	recall	f1-score	support
	0	0.96	0.99	0.98	6281
	1	0.28	0.10	0.15	273
accuracy				0.95	6554
macro avg		0.62	0.54	0.56	6554
weighted avg		0.93	0.95	0.94	6554
<b>weather_related :</b>					
		precision	recall	f1-score	support
	0	0.89	0.95	0.92	4714
	1	0.84	0.69	0.76	1840
accuracy				0.88	6554
macro avg		0.87	0.82	0.84	6554
weighted avg		0.87	0.88	0.87	6554
<b>floods :</b>					

	precision	recall	f1-score	support
0	0.96	0.99	0.97	6025
1	0.79	0.54	0.65	529
accuracy			0.95	6554
macro avg	0.88	0.77	0.81	6554
weighted avg	0.95	0.95	0.95	6554
<b>storm :</b>				
	precision	recall	f1-score	support
0	0.95	0.98	0.97	5936
1	0.74	0.54	0.62	618
accuracy			0.94	6554
macro avg	0.84	0.76	0.80	6554
weighted avg	0.93	0.94	0.93	6554
<b>fire :</b>				
	precision	recall	f1-score	support
0	0.99	1.00	0.99	6483
1	0.47	0.21	0.29	71
accuracy			0.99	6554
macro avg	0.73	0.60	0.64	6554
weighted avg	0.99	0.99	0.99	6554
<b>earthquake :</b>				
	precision	recall	f1-score	support
0	0.98	0.99	0.98	5925
1	0.87	0.80	0.83	629
accuracy			0.97	6554
macro avg	0.92	0.89	0.91	6554
weighted avg	0.97	0.97	0.97	6554
<b>cold :</b>				
	precision	recall	f1-score	support
0	0.98	1.00	0.99	6418
1	0.64	0.25	0.36	136
accuracy			0.98	6554
macro avg	0.81	0.62	0.68	6554
weighted avg	0.98	0.98	0.98	6554

<b>other_weather :</b>				
	precision	recall	f1-score	support
0	0.95	0.99	0.97	6191
1	0.39	0.13	0.20	363
accuracy			0.94	6554
macro avg	0.67	0.56	0.58	6554
weighted avg	0.92	0.94	0.93	6554
<b>direct_report :</b>				
	precision	recall	f1-score	support
0	0.87	0.96	0.91	5270
1	0.70	0.43	0.53	1284
accuracy			0.85	6554
macro avg	0.79	0.69	0.72	6554
weighted avg	0.84	0.85	0.84	6554

### 1.0.8 9. Export your model as a pickle file

```
[28]: # https://machinelearningmastery.com/
    ↪save-load-machine-learning-models-python-scikit-learn/
#https://stackoverflow.com/questions/34143829/
    ↪sklearn-how-to-save-a-model-created-from-a-pipeline-and-gridsearchcv-using-joblib
#pickling the best model and saving data(I believe tuned_model is the best one.
    ↪)
pickle.dump(tuned_model, open('disaster_model.sav', 'wb'))

# loading data
loaded_model = pickle.load(open('disaster_model.sav', 'rb'))
result = loaded_model.score(X_test, y_test)
print(result)
```

0.797985962770827

```
[29]: # out of curiosity, I wanted to check out the result for the new model as well
pickle.dump(new_pipeline, open('new_disaster_model.sav', 'wb'))

# loading data
loaded_model = pickle.load(open('new_disaster_model.sav', 'rb'))
result = loaded_model.score(X_test, y_test)
print(result)
```

0.22566371681415928

### **1.0.9 10. Use this notebook to complete train.py**

Use the template file attached in the Resources folder to write a script that runs the steps above to create a database and export a model based on a new dataset specified by the user.

[ ]:

[ ]: