At the very bottom there is a paragraph explanation detailing the observations found during this analysis. The code below has explanations after each result with a quick explanation of what was found

```
import pandas as pd

df = pd.read_excel("Netflix Dataset Latest 2021.xlsx")

df.head()
```

```
                Title                                       Genre  \
0     Lets Fight Ghost  Crime, Drama, Fantasy, Horror, Romance
1  HOW TO BUILD A GIRL                                     Comedy
2     The Con-Heartist                          Comedy, Romance
3         Gleboka woda                                    Drama
4        Only a Mother                                    Drama

                                                Tags         Languages
\
0  Comedy Programmes,Romantic TV Comedies,Horror ...  Swedish, Spanish

1       Dramas,Comedies,Films Based on Books,British           English

2  Romantic Comedies,Comedies,Romantic Films,Thai...              Thai

3   TV Dramas,Polish TV Shows,Social Issue TV Dramas            Polish

4  Social Issue Dramas,Dramas,Movies Based on Boo...           Swedish


  Series or Movie  Hidden Gem Score  \
0          Series                4.3
1           Movie                7.0
2           Movie                8.6
3          Series                8.7
4           Movie                8.3

                                Country Availability       Runtime  \
0                                           Thailand  < 30 minutes
1                                             Canada      1-2 hour
2                                           Thailand       > 2 hrs
3                                             Poland  < 30 minutes
4  Lithuania,Poland,France,Italy,Spain,Greece,Bel...      1-2 hour

          Director                                             Writer
...  \
0  Tomas Alfredson                                John Ajvide Lindqvist
...
1    Coky Giedroyc                                        Caitlin Moran
```

```
...
2    Mez Tharatorn  Pattaranad Bhiboonsawade, Mez Tharatorn, Thods...
...
3              NaN                                                NaN
...
4      Alf Sjöberg                                  Ivar Lo-Johansson
...

  Netflix Release Date                        Production House  \
0           2021-03-04                Canal+, Sandrew Metronome
1           2021-03-04  Film 4, Monumental Pictures, Lionsgate
2           2021-03-03                                     NaN
3           2021-03-03                                     NaN
4           2021-03-03                                     NaN

                             Netflix Link  \
0  https://www.netflix.com/watch/81415947
1  https://www.netflix.com/watch/81041267
2  https://www.netflix.com/watch/81306155
3  https://www.netflix.com/watch/81307527
4  https://www.netflix.com/watch/81382068

                               IMDb Link  \
0   https://www.imdb.com/title/tt1139797
1   https://www.imdb.com/title/tt4193072
2  https://www.imdb.com/title/tt13393728
3   https://www.imdb.com/title/tt2300049
4   https://www.imdb.com/title/tt0041155

                                          Summary  IMDb Votes  \
0  A med student with a supernatural gift tries t...    205926.0
1  When nerdy Johanna moves to London, things get...      2838.0
2  After her ex-boyfriend cons her out of a large...       131.0
3  A group of social welfare workers led by their...        47.0
4  An unhappily married farm worker struggling to...        88.0

                                            Image  \
0  https://occ-0-4708-64.1.nflxso.net/dnm/api/v6/...
1  https://occ-0-1081-999.1.nflxso.net/dnm/api/v6...
2  https://occ-0-2188-64.1.nflxso.net/dnm/api/v6/...
3  https://occ-0-2508-2706.1.nflxso.net/dnm/api/v...
4  https://occ-0-2851-41.1.nflxso.net/dnm/api/v6/...

                                           Poster  \
0  https://m.media-amazon.com/images/M/MV5BOWM4NT...
1  https://m.media-amazon.com/images/M/MV5BZGUyN2...
2  https://m.media-amazon.com/images/M/MV5BODAzOG...
3  https://m.media-amazon.com/images/M/MV5BMTc0Nz...
4  https://m.media-amazon.com/images/M/MV5BMjVmMz...
```

```
                                   TMDb Trailer Trailer Site
0  https://www.youtube.com/watch?v=LqB6XJix-dM      YouTube
1  https://www.youtube.com/watch?v=eIbcxPy4okQ      YouTube
2  https://www.youtube.com/watch?v=md3CmFLGK6Y      YouTube
3  https://www.youtube.com/watch?v=5kyF2vy63r0      YouTube
4  https://www.youtube.com/watch?v=H0itWKFwMpQ      YouTube

[5 rows x 29 columns]
```

First we're gonna analyze if there is a correlation between certain words in the title of the movie with the movie's IMDb Score.

```
Y = df["IMDb Score"]

X = df["Title"]

list1 = [["z", '1', '1']]

for z in range(len(list1)):
    print(list1[z])
```

```
['z', '1', '1']
```

```
import string
import math

for x in range(X.size):
    txt = str(X[x])
    new_words = txt.split(" ")
    for y in new_words:
        new_string = y.translate(str.maketrans('', '',
string.punctuation)).lower()

        check1 = False
        for z in range(len(list1)):
            if list1[z][0] == new_string:
                if not math.isnan(Y[x]):
                    list1[z][1] = str(int(list1[z][1]) + int(Y[x]))
                    list1[z][2] = str(int(list1[z][2]) + 1)
                check1 = True

        if not check1:
            if not math.isnan(Y[x]):
                list1.append([new_string, str(int(Y[x])), str(1)])

print(list1[1:10])
```

```
[['lets', '34', '5'], ['fight', '67', '10'], ['ghost', '168', '25'],
['how', '175', '27'], ['to', '1238', '193'], ['build', '25', '4'],
['a', '2538', '391'], ['girl', '402', '64'], ['the', '16753', '2584']]
```

```
newlist1 = [["z", 1]]

for x in range(len(list1)):
    if (int(list1[x][2]) > 5):
        temp1 = float((int(list1[x][1]))/(int(list1[x][2])))
        res = "{:.4f}".format(temp1)
        newlist1.append([list1[x][0], float(res)])

newList3 = sorted(newlist1, key=lambda x: x[1], reverse=True)
newList3[1:25]
```

```
[['version', 7.5333],
 ['monty', 7.4667],
 ['files', 7.3333],
 ['terrace', 7.3333],
 ['masters', 7.3],
 ['python', 7.3],
 ['grand', 7.2857],
 ['wall', 7.2857],
 ['line', 7.2857],
 ['place', 7.2857],
 ['kenshin', 7.2857],
 ['shell', 7.2857],
 ['inside', 7.2727],
 ['oh', 7.25],
 ['future', 7.2308],
 ['mystery', 7.2222],
 ['universe', 7.2222],
 ['diaries', 7.2222],
 ['show', 7.1818],
 ['bill', 7.1818],
 ['lion', 7.1667],
 ['bridge', 7.1667],
 ['martin', 7.1667],
 ['inuyasha', 7.1667]]
```

This analyzes the words in the title of the show with the imdb score, finding key words that have the best imdb scores. We can see that movie titles that have the words, "version", "monty", "files", "terrace", and "masters" have an average IMDb score of "7.5333", "7.4667", "7.3333", "7.3333", and "7.3" respectively

Now we're going to do the same but with the genre of the movies

```
X = df["Genre"]

list1 = [["z", '1', '1']]

import string
import math

for x in range(X.size):
```

```
    txt = str(X[x])
    new_words = txt.split(", ")
    for y in new_words:
        new_string = y.translate(str.maketrans('', '',
string.punctuation)).lower()

        check1 = False
        for z in range(len(list1)):
            if list1[z][0] == new_string:
                if not math.isnan(Y[x]):
                    list1[z][1] = str(int(list1[z][1]) + int(Y[x]))
                    list1[z][2] = str(int(list1[z][2]) + 1)
                check1 = True

        if not check1:
            if not math.isnan(Y[x]):
                list1.append([new_string, str(int(Y[x])), str(1)])

list1[1:10]
```

```
[['crime', '9814', '1512'],
 ['drama', '31692', '4803'],
 ['fantasy', '7978', '1228'],
 ['horror', '4239', '716'],
 ['romance', '11764', '1811'],
 ['comedy', '21235', '3304'],
 ['mystery', '6030', '936'],
 ['thriller', '12838', '2069'],
 ['short', '1111', '161']]
```

```
newlist1 = [["z", 1]]

for x in range(len(list1)):
    if (int(list1[x][2]) > 5):
        temp1 = float((int(list1[x][1]))/(int(list1[x][2])))
        res = "{:.4f}".format(temp1)
        newlist1.append([list1[x][0], float(res)])

newList3 = sorted(newlist1, key=lambda x: x[1], reverse=True)
newList3[1:25]
```

```
[['news', 7.1429],
 ['documentary', 6.9751],
 ['gameshow', 6.9375],
 ['realitytv', 6.9369],
 ['short', 6.9006],
 ['animation', 6.8653],
 ['nan', 6.8636],
 ['history', 6.7656],
 ['war', 6.7199],
 ['biography', 6.6836],
```

```
 ['drama', 6.5984],
 ['sport', 6.5952],
 ['music', 6.5812],
 ['western', 6.5402],
 ['fantasy', 6.4967],
 ['romance', 6.4959],
 ['crime', 6.4907],
 ['musical', 6.4634],
 ['mystery', 6.4423],
 ['comedy', 6.4271],
 ['adventure', 6.3882],
 ['scifi', 6.3602],
 ['family', 6.3485],
 ['action', 6.324]]
```

Looking at this data, we can see that the top 10 Genre's are News, Documentary, Gameshow, RealityTV, Short, Animation, History, War, Biography, Drama. Some of these genres are stereotypical high rated genres, such as "realitytv", "animation", "short", and "documentary"

Finally, we're doing the same for the movie tags

```
X = df["Tags"]
list1 = [["z", '1', '1']]

import string
import math

for x in range(X.size):
    txt = str(X[x])
    new_words = txt.split(",")
    for y in new_words:
        new_string = y.translate(str.maketrans('', '',
string.punctuation)).lower()

        check1 = False
        for z in range(len(list1)):
            if list1[z][0] == new_string:
                if not math.isnan(Y[x]):
                    list1[z][1] = str(int(list1[z][1]) + int(Y[x]))
                    list1[z][2] = str(int(list1[z][2]) + 1)
                check1 = True

        if not check1:
            if not math.isnan(Y[x]):
                list1.append([new_string, str(int(Y[x])), str(1)])

newlist1 = [["z", 1]]

for x in range(len(list1)):
    if (int(list1[x][2]) > 5):
```

```
        temp1 = float((int(list1[x][1]))/(int(list1[x][2])))
        res = "{:.4f}".format(temp1)
        newlist1.append([list1[x][0], float(res)])

newList3 = sorted(newlist1, key=lambda x: x[1], reverse=True)
newList3[1:25]
```

```
[['silent films', 7.8333],
 ['theater arts', 7.625],
 ['tv variety  talk shows', 7.5714],
 ['shoujo anime', 7.5],
 ['oscarwinning films', 7.5],
 ['classic westerns', 7.5],
 ['classic thrillers', 7.5],
 ['police tv shows', 7.5],
 ['miniseries', 7.5],
 ['asian programmes', 7.5],
 ['hindilanguage tv shows', 7.4615],
 ['australian tv programmes', 7.4615],
 ['anime horror films', 7.4444],
 ['classic scifi  fantasy', 7.4375],
 ['kids' programmes', 7.4286],
 ['british tv comedies', 7.4194],
 ['academy awardwinning films', 7.4043],
 ['british tv programmes', 7.3982],
 ['bafta awardwinning films', 7.3824],
 ['classic international movies', 7.375],
 ['classic films', 7.3652],
 ['military documentaries', 7.35],
 ['canadian tv programmes', 7.3333],
 ['nature  ecology documentaries', 7.3333]]
```

While the section above was quite general, this data point is interesting because there are very specific tags here. The top five are Silent Films, Theater Arts, TV Variety Talk Shows, Shoujo Anime, and Oscar Winning Films. This data point provides a more specific datapoint that could be used to promoted high rated shows in these genres over others, because shows of these tags are more likely to be higher rated.

One last analysis, this time on the actors in the movie/series

```
X = df["Actors"]
list1 = [["z", '1', '1']]

import string
import math

for x in range(X.size):
    txt = str(X[x])
    new_words = txt.split(", ")
    for y in new_words:
        new_string = y.translate(str.maketrans('', '',
```

```
string.punctuation)).lower()

        check1 = False
        for z in range(len(list1)):
            if list1[z][0] == new_string:
                if not math.isnan(Y[x]):
                    list1[z][1] = str(int(list1[z][1]) + int(Y[x]))
                    list1[z][2] = str(int(list1[z][2]) + 1)
                check1 = True

        if not check1:
            if not math.isnan(Y[x]):
                list1.append([new_string, str(int(Y[x])), str(1)])

newlist1 = [["z", 1]]

for x in range(len(list1)):
    if (int(list1[x][2]) > 5):
        temp1 = float((int(list1[x][1]))/(int(list1[x][2])))
        res = "{:.4f}".format(temp1)
        newlist1.append([list1[x][0], float(res)])

newList3 = sorted(newlist1, key=lambda x: x[1], reverse=True)
newList3[1:25]
```

```
[['sathyaraj', 7.75],
 ['graham chapman', 7.7143],
 ['prabhas', 7.625],
 ['miyu irino', 7.6154],
 ['kwangsoo lee', 7.5714],
 ['charles chaplin', 7.5556],
 ['mamoru miyano', 7.5385],
 ['asami seto', 7.5],
 ['sue perkins', 7.5],
 ['aamir khan', 7.5],
 ['terry gilliam', 7.5],
 ['michelle ruff', 7.5],
 ['park seojoon', 7.4444],
 ['luci christian', 7.4444],
 ['tom kenny', 7.4286],
 ['atsumi tanezaki', 7.4286],
 ['takahiro sakurai', 7.4286],
 ['yui ishikawa', 7.4286],
 ['lee jieun', 7.4286],
 ['lee jongsuk', 7.4286],
 ['john cleese', 7.4],
 ['eric idle', 7.3846],
 ['koki uchiyama', 7.3333],
 ['ara go', 7.3333]]
```

Looking at this analysis, we can see that the top 5 highest rated IMDb actors are sathyaraj, graham chapman, prabhas, miyu irino, kwangsoo lee. This is interesting because the highest rated actors/actresses are foreign ones.

Now we're gonna perform the same analysis, but instead check it with the price of netflix stock at the time the movies drop rather than the IMDb rating of the movie

```
df1 = pd.read_csv('HistoricalPrices.csv')
```

```
dates = df1.iloc[:, 0]
prices = df1.iloc[:, 2]
```

```
dates[1:10]
```

```
1    09/13/22
2    09/12/22
3    09/09/22
4    09/08/22
5    09/07/22
6    09/06/22
7    09/02/22
8    09/01/22
9    08/31/22
Name: Date, dtype: object
```

```
show_dates = df['Netflix Release Date']
show_dates[1:10]
```

```
1   2021-03-04
2   2021-03-03
3   2021-03-03
4   2021-03-03
5   2021-03-03
6   2021-03-03
7   2021-03-03
8   2021-03-03
9   2021-03-03
Name: Netflix Release Date, dtype: datetime64[ns]
```

Looking at the difference in how the data is presented between the netflix stock dataset and the netflix movie dataset, we need to do some preprocessing to convert the data to similar formats

```
new_list = []
```

```
for x in range(len(show_dates)):
    temp = str(show_dates[x])
    split_words = temp.split("-")
    new_date = split_words[1] + "/" + split_words[2][:2] + "/" +
split_words[0][2:4]
    new_list.append(str(new_date))
```

```
actor_price_list = [["z", 1.0, 1]]

def find_five(date):
    for x in range(len(dates)):
        if dates[x] == date and x > 5:
            val = 0.0
            for y in range(5):
                val += float(prices[x - y])
            return val/5

def find_ten(date):
    for x in range(len(dates)):
        if dates[x] == date and x > 5:
            val = 0.0
            for y in range(10):
                val += float(prices[x - y + 5])
            return val/10
```

The functions above:

find_five(date): This method takes in a date, and returns the five day rolling average of netflix after this date

find_ten(date): this method takes in a date, and returns the 10 day rolling average of netflix surrounding this date, meaning it checks 5 days afterwards and 4 days before as well.

```
def stock_five(date):
    for x in range(len(dates)):
        if dates[x] == date and x > 5:
            return prices[x - 4]

def stock_now(date):
    for x in range(len(dates)):
        if dates[x] == date and x > 5:
            return prices[x]
```

The functions above:

stock_five(date): returns the price of the stock 5 days after the given date

stock_now(date): returns the current price of the netflix stock

```
print(find_five("08/12/22"))
print(find_ten("08/12/22"))
print(stock_five("08/12/22"))
print(stock_now("08/12/22"))
```

```
248.28188
244.03492000000006
246.4794
249.41
```

Now we're doing the stock price analysis with the genre of the stock

```
X = df["Genre"]
```

```
import string
import math

for x in range(X.size):
    txt = str(X[x])
    new_words = txt.split(", ")
    if x % 500 == 0:
        print(x)
    for y in new_words:
        new_string = y.translate(str.maketrans('', '',
string.punctuation)).lower()

        check1 = False
        for z in range(len(actor_price_list)):
            if actor_price_list[z][0] == new_string:
                date = new_list[x]
                five = stock_five(date)
                ten = stock_now(date)
                if five is not None and ten is not None:
                    actor_price_list[z][1] += five/ten
                    actor_price_list[z][2] += 1
                check1 = True

        if not check1:
            date = new_list[x]
            five = stock_five(date)
            ten = stock_now(date)
            if five is not None and ten is not None:
                actor_price_list.append([new_string, five/ten, 1])
```

```
0
500
1000
1500
2000
2500
3000
3500
4000
4500
5000
5500
6000
6500
7000
7500
8000
```

```
8500
9000
```

```
actor_price_list[1:10]
```

```
[['crime', 3816.4523715012992, 3814],
 ['drama', 11921.459266066313, 11916],
 ['fantasy', 3128.9075221013736, 3125],
 ['horror', 1848.7416485502847, 1846],
 ['romance', 4497.565778911585, 4492],
 ['comedy', 8365.649542273219, 8357],
 ['mystery', 2381.7563756239515, 2378],
 ['thriller', 5220.90925943258, 5216],
 ['short', 391.75372146122646, 393]]
```

```
newlist1 = [["z", 1]]

for x in range(len(actor_price_list)):
    if (int(actor_price_list[x][2]) > 5):
        temp1 = float(actor_price_list[x][1])/(actor_price_list[x][2])
        res = "{:.4f}".format(temp1)
        newlist1.append([actor_price_list[x][0], float(res)])

newList3 = sorted(newlist1, key=lambda x: x[1], reverse=True)
newList3[1:25]
```

```
[['news', 1.0021],
 ['sport', 1.0018],
 ['mystery', 1.0016],
 ['western', 1.0016],
 ['horror', 1.0015],
 ['war', 1.0015],
 ['documentary', 1.0015],
 ['fantasy', 1.0013],
 ['romance', 1.0012],
 ['action', 1.0012],
 ['adventure', 1.0012],
 ['animation', 1.0011],
 ['comedy', 1.001],
 ['musical', 1.001],
 ['thriller', 1.0009],
 ['scifi', 1.0008],
 ['biography', 1.0007],
 ['history', 1.0007],
 ['crime', 1.0006],
 ['drama', 1.0005],
 ['family', 1.0005],
 ['nan', 1.0004],
 ['adult', 1.0003],
 ['z', 1]]
```

This shows that for the top 5 genres (news, sport, mystery, western, horror), the stock price of netflix rose on average 1.0021, 1.0018, 1.0016, 1.0016, and 1.0015 times respectively in one week whenever a movie of that genre was released

Since we know that there are downturns in the stock market and that shouldn't negatively impact a genre's performance, we're going to run this test again but check the 5 day rolling average compared to the 10 day rolling average, and see if there was an uptick in the stock after the movie genre was released.

```
actor_price_list = [["z", 1.0, 1]]
X = df["Genre"]

import string
import math

for x in range(X.size):
    txt = str(X[x])
    new_words = txt.split(", ")
    if x % 500 == 0:
        print(x)
    for y in new_words:
        new_string = y.translate(str.maketrans('', '',
string.punctuation)).lower()

        check1 = False
        for z in range(len(actor_price_list)):
            if actor_price_list[z][0] == new_string:
                date = new_list[x]
                five = find_five(date)
                ten = find_ten(date)
                if five is not None and ten is not None:
                    actor_price_list[z][1] += five/ten
                    actor_price_list[z][2] += 1
                check1 = True

        if not check1:
            date = new_list[x]
            five = find_five(date)
            ten = find_ten(date)
            if five is not None and ten is not None:
                actor_price_list.append([new_string, five/ten, 1])
```

```
0
500
1000
1500
2000
2500
3000
3500
```

```
4000
4500
5000
5500
6000
6500
7000
7500
8000
8500
9000
```

```
newlist1 = [["z", 1]]

for x in range(len(actor_price_list)):
    if (int(actor_price_list[x][2]) > 5):
        temp1 = float(actor_price_list[x][1])/(actor_price_list[x][2])
        res = "{:.4f}".format(temp1)
        newlist1.append([actor_price_list[x][0], float(res)])
```

```
newList3 = sorted(newlist1, key=lambda x: x[1], reverse=True)
newList3[1:25]
```

```
[['family', 1.0361],
 ['action', 1.0337],
 ['adventure', 1.0333],
 ['thriller', 1.0325],
 ['war', 1.0317],
 ['horror', 1.0313],
 ['western', 1.0308],
 ['crime', 1.0306],
 ['musical', 1.0305],
 ['fantasy', 1.0304],
 ['sport', 1.0297],
 ['comedy', 1.0295],
 ['scifi', 1.0293],
 ['mystery', 1.0279],
 ['biography', 1.0272],
 ['romance', 1.027],
 ['talkshow', 1.0264],
 ['music', 1.0259],
 ['drama', 1.025],
 ['animation', 1.025],
 ['history', 1.024],
 ['gameshow', 1.0212],
 ['short', 1.0156],
 ['documentary', 1.0147]]
```

This is a better datapoint, because it shows that even at times when there is a downturn in netflix stops, when movies of genres family, action, adventure, thriller, and war are

released, there is on average a 3.61%, 3.37%, 3.33%, 3.25%, and 3.17% increase in the 5 day rolling average compared to the 10 day rolling average

Now it's time to do the same but for actors

```
X = df["Actors"]
actor_price_list = [["z", 1.0, 1]]

import string
import math

for x in range(X.size):
    txt = str(X[x])
    new_words = txt.split(", ")
    if x % 250 == 0:
        print(x)
    for y in new_words:
        new_string = y.translate(str.maketrans('', '',
string.punctuation)).lower()

        check1 = False
        for z in range(len(actor_price_list)):
            if actor_price_list[z][0] == new_string:
                date = new_list[x]
                five = stock_five(date)
                ten = stock_now(date)
                if five is not None and ten is not None:
                    actor_price_list[z][1] += five/ten
                    actor_price_list[z][2] += 1
                check1 = True

        if not check1:
            date = new_list[x]
            five = stock_five(date)
            ten = stock_now(date)
            if five is not None and ten is not None:
                actor_price_list.append([new_string, five/ten, 1])
```

```
0
250
500
750
1000
1250
1500
1750
2000
2250
2500
2750
3000
```

```
3250
3500
3750
4000
4250
4500
4750
5000
5250
5500
5750
6000
6250
6500
6750
7000
7250
7500
7750
8000
8250
8500
8750
9000
9250
```

```
actor_price_list[1:10]
```

```
[['lina leandersson', 1.9669518113549378, 2],
 ['kåre hedebrant', 1.9669518113549378, 2],
 ['per ragnar', 1.9669518113549378, 2],
 ['henrik dahl', 1.9669518113549378, 2],
 ['cleo', 1.9567471955438813, 2],
 ['paddy considine', 6.455069005273012, 6],
 ['beanie feldstein', 1.9483259461409501, 2],
 ['dónal finn', 0.9764252116650989, 1],
 ['kathaleeya mcintosh', 0.9363235736900211, 1]]
```

```
newlist1 = [["z", 1]]

for x in range(len(actor_price_list)):
    if (int(actor_price_list[x][2]) > 10):
        temp1 = float(actor_price_list[x][1])/(actor_price_list[x][2])
        res = "{:.4f}".format(temp1)
        newlist1.append([actor_price_list[x][0], float(res)])

newList3 = sorted(newlist1, key=lambda x: x[1], reverse=True)
newList3[1:25]
```

```
[['jim carrey', 1.1571],
 ['cameron diaz', 1.1565],
```

```
 ['colin firth', 1.1532],
 ['paul rudd', 1.1492],
 ['stanley tucci', 1.1477],
 ['eddie murphy', 1.1453],
 ['tom wilkinson', 1.1443],
 ['kirsten dunst', 1.1428],
 ['matt damon', 1.1411],
 ['jack black', 1.1405],
 ['christopher walken', 1.1404],
 ['angelina jolie', 1.1403],
 ['jessica biel', 1.1395],
 ['johnny depp', 1.1391],
 ['ben stiller', 1.1388],
 ['tom cruise', 1.1382],
 ['josh brolin', 1.137],
 ['richard gere', 1.1363],
 ['gerard butler', 1.1341],
 ['will ferrell', 1.1326],
 ['vin diesel', 1.1317],
 ['owen wilson', 1.1314],
 ['leonardo dicaprio', 1.1306],
 ['morgan freeman', 1.1297]]
```

Now do the same with the rolling averages

```
X = df["Actors"]
actor_price_list = [["z", 1.0, 1]]

import string
import math

for x in range(X.size):
    txt = str(X[x])
    new_words = txt.split(", ")
    if x % 250 == 0:
        print(x)
    for y in new_words:
        new_string = y.translate(str.maketrans('', '',
string.punctuation)).lower()

        check1 = False
        for z in range(len(actor_price_list)):
            if actor_price_list[z][0] == new_string:
                date = new_list[x]
                five = find_five(date)
                ten = find_ten(date)
                if five is not None and ten is not None:
                    actor_price_list[z][1] += five/ten
                    actor_price_list[z][2] += 1
                check1 = True
```

```
        if not check1:
            date = new_list[x]
            five = find_five(date)
            ten = find_ten(date)
            if five is not None and ten is not None:
                actor_price_list.append([new_string, five/ten, 1])
```

```
0
250
500
750
1000
1250
1500
1750
2000
2250
2500
2750
3000
3250
3500
3750
4000
4250
4500
4750
5000
5250
5500
5750
6000
6250
6500
6750
7000
7250
7500
7750
8000
8250
8500
8750
9000
9250
```

```
newlist1 = [["z", 1]]

for x in range(len(actor_price_list)):
    if (int(actor_price_list[x][2]) > 10):
```

```
        temp1 = float(actor_price_list[x][1])/(actor_price_list[x][2])
        res = "{:.4f}".format(temp1)
        newlist1.append([actor_price_list[x][0], float(res)])

newList3 = sorted(newlist1, key=lambda x: x[1], reverse=True)
newList3[1:10]
```

```
[['jessica biel', 1.0709],
 ['jim carrey', 1.0707],
 ['jason statham', 1.0701],
 ['gerard butler', 1.0698],
 ['colin firth', 1.0692],
 ['paul rudd', 1.0686],
 ['jack black', 1.0685],
 ['tom wilkinson', 1.0668],
 ['kirsten dunst', 1.0666]]
```

We can see that the actors with the highest change in rolling averages was jessica biel, jim carrey, jason statham, garard butler, and colin firth.

During this analysis, I split it into two goals. The first was to see if there were specific features of the movies that led to higher IMDb ratings. The second part was to see if there were any features of movies that hed to a positive gain in stock price.

In the first section of the analysis, like stated above, the goal was to find the any specific feature that correlated with higher IMDb ratings. We started off by analyzing the titles of movies. By splitting the words in the titles, converting them to lowercase, and ignoring words like "the", "and", "by", etc. that provide no additional meaning, we were able to calculate a list of words that appeared in titles with their average IMDb score. One that the had to be adjusted was that we only displayed words that appeared more than 5 times, to avoid words that only appeared in 1 movie title that had an insanely high IMDb score and skewed the results.

However, this analysis doesn't really help any data scientists, because there aren't many conclusions you can draw with random words in titles vs IMDb scores. Thus, we proceeded to analyze the different genres. Since each movie/series had 1 or more genres, we were able to find the average IMDb scores of each genre. This provided interesting information that could be useful in analysis.

But we can do better. The data column of "tags" provided very specific information about each movie/series. Using this data point, we can find very specific information about different movies/series that correlated with higher IMDb scores. After completing this analysis, we now possess important data that can be used to see what types of shows to promote. If we know that movies of one tag in general are rated higher than movies of another tag, then it allows netflix to choose which types of movies to promote to gain better user engagement. This type of data is useful and is what data scientists seek.

Another useful data point that I analyzed is the relationship between the actors in a movie with the IMDb score of the movie. This data point shows which actors had the highest

overall rating for IMDb scores. This data point is also quite useful, because it tells directors of new movies what actor's are promising to add to a movie to get higher scores.

In the second section of this analysis, we try to find specific features of movies that led to the highest stock gains. Yet again, we split this section into two different parts.

1. Find the features with the highest overall 5 day stock gain.

This data point is useful because pure stock gain is what investment funds seek. However, it leads to issues when the stock market/netflix stock is at a down turn. In situations like this, even if a movie is great and leads to positivity of the stock, the overall condition of the market might cause the stock to still decrease over 5 days, negatively affecting the features of this stock. This is why I felt the second metric was more useful

1. Compare the 5 day rolling average with the 10 day rolling average, to see if a feature caused the 5 day average to be larger than the 10 day average meaning there was a positive uptick in the stock

This data point I felt is more useful because even in downturns of the market, a less sharp decline in the stock caused by the release of a movie leads to the 5 day average being greater than the 10 day average, positively benefiting the features of that stock.

I won't go into details about each of the features we tested for in this section (because they are the same as the ones in the previous section), but for each of the features tested, I went through the two different stock measurements for each. Althought I felt that the rolling average datapoint is more useful, having both that and the stock gain datapoint is useful for any future gains.

One positive of the very specific features like tags and author names is that when movies with these features are released, one can be relatively confident that they can follow the data and make an accurate stock purchase.

A note for this section is that these movies and series were all released during an overall bull market for netflix. It's been relatively positive gains for the most part up until November 2021. After that date, the Netflix stock has dropped drastically. This means that the analysis of the data before the major drop may not be completely useful in the future, as the change from a bull market to a bear market can mean the data behind each feature isn't accurate anymore.