IDS201 Assessment 3 - Programming Exercise

David Lawler

The Data science problem

The aim of this case study is to use statistical analysis, and machine learning algorithms, on a Netflix stock price dataset, with the objectives of trying to find patterns, identifying future trends and predicting stock values.

The Netflix Dataset

The dataset I be using is from the website kaggle and is called Netflix Stock Price Prediction. It contains five years of price data(5th May 2018 - 5th May 2022).

Source: **https://www.kaggle.com/datasets/jainilcoder/netflix-stock-price-prediction**

There are seven variables:

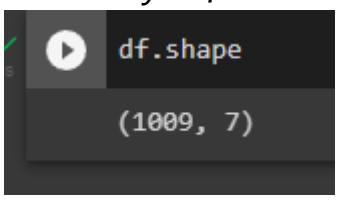
Date - The independent variable, format is YYYY-MM-DD
Open - The price the stock opened at
High - The intra-day highest price
Low - The intra-day lowest price
Close - The price the stock closed at
Adj Close - The close price adjusted for splits and dividend and/or capital gain distributions.
Volume - The number of stocks traded that day.

Head printout showing the first 5 columns of dataset. Date variable set to index

```
df.set_index('Date', inplace=True)
df.head()
```

Date	Open	High	Low	Close	Adj Close	Volume
2018-02-05	262.000000	267.899994	250.029999	254.259995	254.259995	11896100
2018-02-06	247.699997	266.700012	245.000000	265.720001	265.720001	12595800
2018-02-07	266.579987	272.450012	264.329987	264.559998	264.559998	8981500
2018-02-08	267.079987	267.619995	250.000000	250.100006	250.100006	9306700
2018-02-09	253.850006	255.800003	236.110001	249.470001	249.470001	16906900

1009 Days of Data



Analytical Approach

1. Imports
   First I imported the required packages for analysing the dataset.

```
[138] # Imports
      import numpy as np
      import pandas as pd
      import plotly.express as px
      import seaborn as sns
      import matplotlib.pyplot as plt
```

- NumPy is a package for scientific computing in Python, generally used for working with arrays.
- Pandas is a data analysis and manipulation package
- Plotly express is used for creating figures.
- Seaborn is another data visualisation library built on matplotlib, integrated with pandas data structures
- matplotlib.pyplot is used to make 2d graphs and plots

2. Read the CSV File

   Next I downloaded the NFLX dataset and read it using pandas, printing the head to visualise the dataset and variables.

```
df = pd.DataFrame(pd.read_csv("NFLX.csv"))
df.head()
```

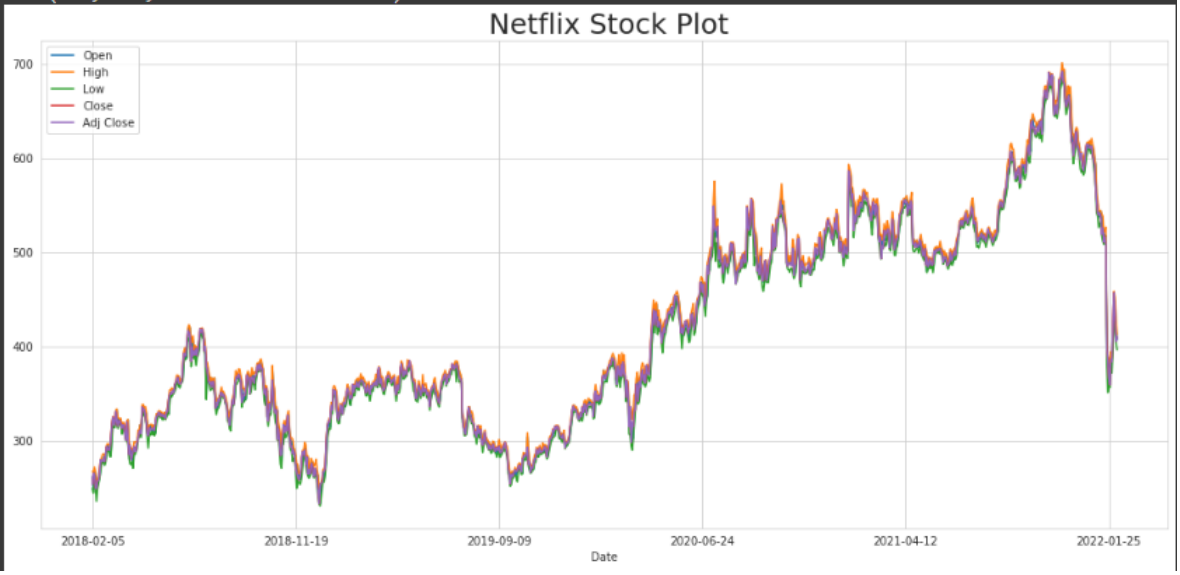
	Date	Open	High	Low	Close	Adj Close	Volume
0	2018-02-05	262.000000	267.899994	250.029999	254.259995	254.259995	11896100
1	2018-02-06	247.699997	266.700012	245.000000	265.720001	265.720001	12595800
2	2018-02-07	266.579987	272.450012	264.329987	264.559998	264.559998	8981500
3	2018-02-08	267.079987	267.619995	250.000000	250.100006	250.100006	9306700
4	2018-02-09	253.850006	255.800003	236.110001	249.470001	249.470001	16906900

3. Netflix Stock Price Plot

   Here I plotted the six dependant variables across the dependant variable showing the standard stock price plot.

```
# Time series plot of open,  high, low, close and adj close
df[['Open','High','Low','Close','Adj Close']].plot(figsize=(18,8))
plt.title(" Netflix Stock Plot", fontsize=25)
```

```
Text(0.5, 1.0, ' Netflix Stock Plot')
```

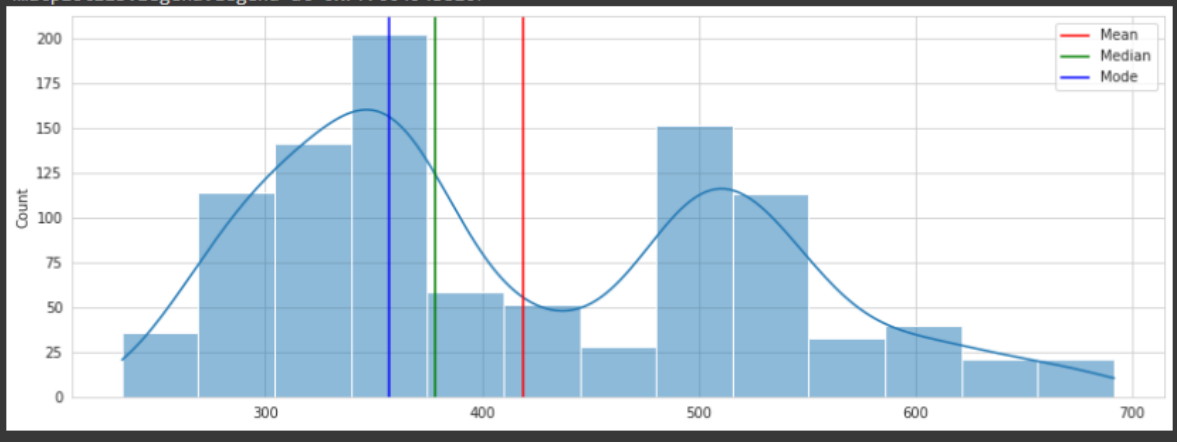


4. Mean, Median, Mode

   Next I created a subplot providing a graphical representation of the stock price mean, median and mode against a histogram of stock prices.

```
[197] f, (ax1) = plt.subplots(1, 1, figsize=(14, 5))
      v_dist_1 = df['Close'].values
      sns.histplot(v_dist_1, ax=ax1, kde=True)
      mean=df['Close'].mean()
      median=df['Close'].median()
      mode=df['Close'].mode().values[0]
      ax1.axvline(mean, color='r', linestyle='-', label="Mean")
      ax1.axvline(median, color='g', linestyle='-', label="Median")
      ax1.axvline(mode, color='b', linestyle='-', label="Mode")
      ax1.legend()

      <matplotlib.legend.Legend at 0x7f700494b810>
```



The mean gives us the average of all close prices. The median is the middle value of all close prices and mode is the most common value. Noting the difference between these values and the wide spread of close prices on the histogram.

5. Daily Returns

   Next I calculate daily returns based on Adjusted close price, as a percentage.

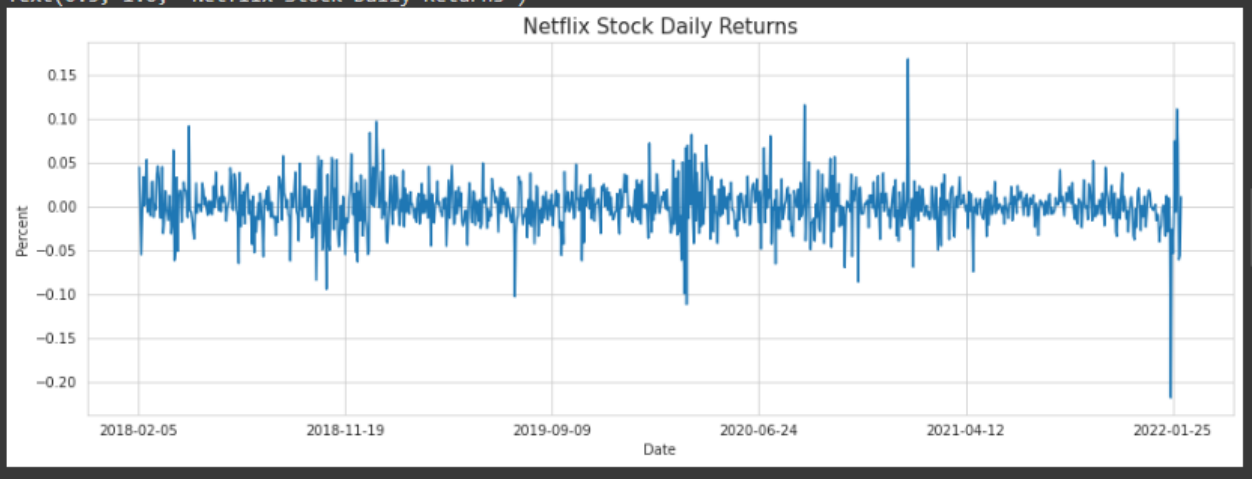
```
df['Daily_returns'] = df['Adj Close'].pct_change() #Pandas dataframe.pct_change() function cal
df.head()
```

Date	Open	High	Low	Close	Adj Close	Volume	Daily_returns
2018-02-05	262.000000	267.899994	250.029999	254.259995	254.259995	11896100	NaN
2018-02-06	247.699997	266.700012	245.000000	265.720001	265.720001	12595800	0.045072
2018-02-07	266.579987	272.450012	264.329987	264.559998	264.559998	8981500	-0.004366
2018-02-08	267.079987	267.619995	250.000000	250.100006	250.100006	9306700	-0.054657
2018-02-09	253.850006	255.800003	236.110001	249.470001	249.470001	16906900	-0.002519

We can see in the plot below most daily returns fall between 5% and -5%, with very few outliers outside 10% and -10%, and only 1 day where the daily returns went outside the 20% range.

```
[151] plt.figure(figsize=(15,5))
      df['Daily_returns'].plot()
      plt.xlabel("Date")
      plt.ylabel("Percent")
      plt.title("Netflix Stock Daily Returns",fontsize= 15 )
```

```
Text(0.5, 1.0, 'Netflix Stock Daily Returns')
```

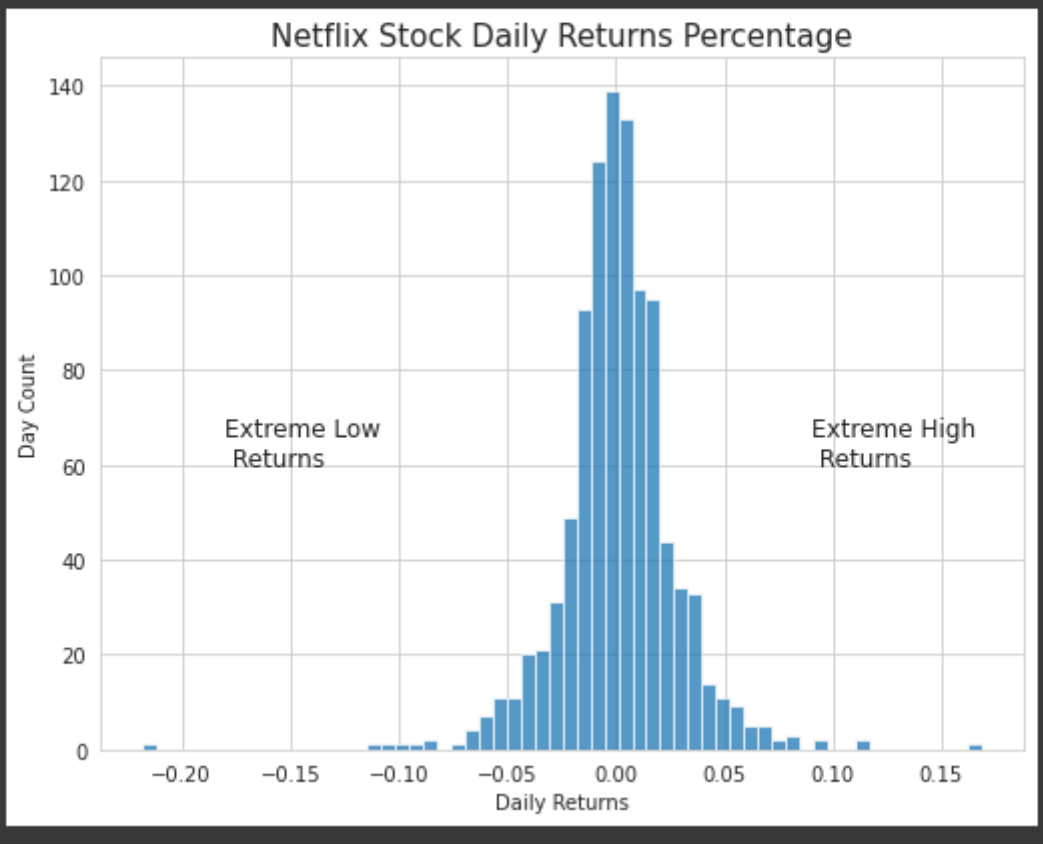


I then checked the distribution of daily returns. As per the above graph we see

few outliers.

```
sns.set_style('whitegrid')
fig = plt.figure(figsize=(8,6))
ax1 = fig.add_axes([0.1,0.1,0.8,0.8])
sns.histplot(data= df['Daily_returns'], bins=60)
ax1.set_xlabel("Daily Returns")
ax1.set_ylabel("Day Count")
ax1.set_title("Netflix Stock Daily Returns Percentage",fontsize= 15 )
ax1.text(-0.18,60,"Extreme Low\n Returns",fontsize= 12)
ax1.text(0.09,60,"Extreme High\n Returns", fontsize= 12)

plt.show()
```

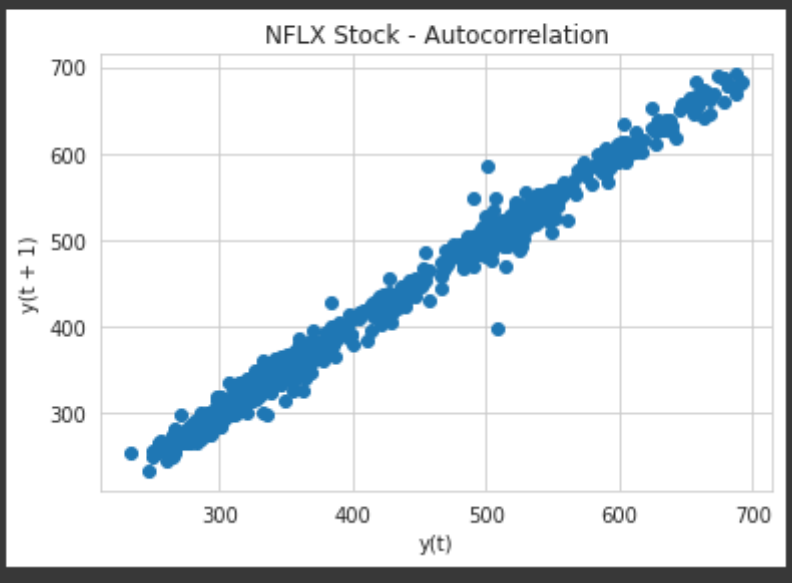


6. Autocorrelation

   Next we check autocorrelation - autocorrelation is used to represent how similar a value within a time series is to a previous value. We see a high

correlation between close values.

```
[198] plt.figure()
      lag_plot(df['Close'])
      plt.title('NFLX Stock - Autocorrelation')
      plt.show()
```

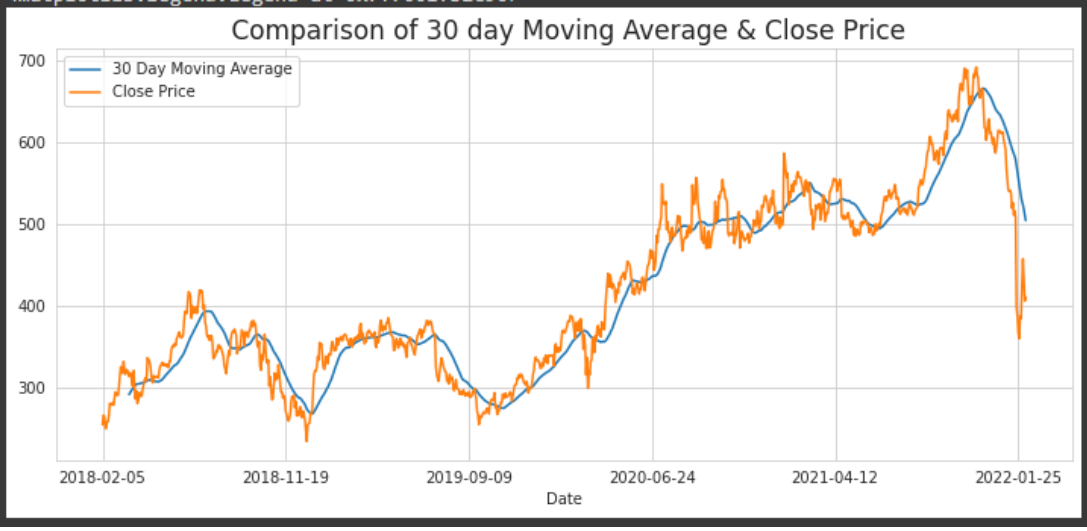


7. Compare 30 Day Moving Average and Close Price

   Looking at the close price to moving average we can see that the price tends towards the mean.

```
[200] f= plt.figure(figsize=(12,5))
      df['Close'].rolling(window=30).mean().plot(label='30 Day Moving Average')
      df['Close'].plot(label='Close Price')
      plt.title(" Comparison of 30 day Moving Average & Close Price", fontsize=17)
      plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f7001fb2c90>
```



8. 5 Day Prediction using Statistical Method: ARIMA

   ARIMA - Autoregressive Integrated Moving Average Model - converts non-stationary data to stationary data before working on it. It is one of

the most popular models to use to predict linear time series data. It has proven to be effective at short-term price prediction.

Imports

```
import pandas
import matplotlib.mlab as mlab
import matplotlib.pyplot as plt
import numpy as np
import math
from statsmodels.tsa.stattools import acf, pacf
import statsmodels.tsa.stattools as ts
from statsmodels.tsa.arima_model import ARIMA
```

```
[10] price_matrix=lnClose.to_numpy()
     model = ARIMA(price_matrix, order=(0,1,0))
     model_fit = model.fit(disp=0)
     print(model_fit.summary())
```

```
                             ARIMA Model Results
==============================================================================
Dep. Variable:                   D.y   No. Observations:                 1008
Model:                 ARIMA(0, 1, 0)   Log Likelihood                2219.566
Method:                           css   S.D. of innovations              0.027
Date:                Thu, 11 Aug 2022   AIC                          -4435.133
Time:                        05:16:35   BIC                          -4425.301
Sample:                             1   HQIC                         -4431.398

==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0005      0.001      0.563      0.574      -0.001       0.002
==============================================================================
```

```
# Prediction of the next five days
predictions=model_fit.predict(1009, 1013, typ='levels')
predictions
predictionsadjusted=np.exp(predictions)
predictionsadjusted
```

```
array([410.36465145, 410.55938227, 410.75420549, 410.94912116,
       411.14412932])
```

9. 5 Day Prediction using Machine Learning Method: Random Forest

The idea behind random forests is fairly simple; first, the data is split into different partitions, then a certain number of random features is used to create and train a decision tree. This will repeat *n* number of times, where *n* is the number of trees to grow. Each tree will then output a prediction. Each prediction will then be calculated for number of votes, and the prediction with the highest number of votes will be the final prediction.

```
[48] from sklearn.model_selection import train_test_split
     from sklearn import metrics
     from sklearn.preprocessing import StandardScaler
     from sklearn.linear_model import LinearRegression
     from sklearn.ensemble import RandomForestClassifier
     from sklearn.ensemble import RandomForestRegressor
```

```
[49] X= df[['Open', 'High', 'Low', 'Close', 'Volume']]
     y= df['Adj Close']
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

```
scale = StandardScaler()
X_train = scale.fit_transform(X_train)
X_test = scale.transform(X_test)
```

```
model = RandomForestRegressor(n_estimators=500, random_state=42, max_depth=10)
model.fit(X_train, y_train)
predict = model.predict(X_test)
print(predict)
print(predict.shape)
```

```
[553.8416925  379.17971491 361.33349299 282.50936889 260.77375919
 434.16278263 264.71795692 517.99407007 315.73065802 348.70286051
 520.36973569 321.46615905 489.83945534 298.52005655 510.72065161
 503.16192322 288.79939388 549.52489116 337.49877138 270.31087746
 338.96048506 294.16196666 462.23160614 369.07376958 290.25871525
 502.90477754 498.37161485 297.10562156 351.1763273  363.62071151
 280.67250402 542.85362106 316.8633096  540.65456756 303.56822135
 520.03440602 339.57363941 381.19329738 515.78588464 628.57645737
 482.38360772 298.34240141 305.6844204  264.90545262 515.93398785
 297.64492828 294.20649666 384.51621386 359.68261224 419.26142425
 365.91416813 607.00680919 480.68236577 512.51465393 364.59413437
 419.21392399 375.49670179 360.28943585 370.3928537  361.11599348
 295.16631208 493.15682611 491.8727955  406.09191563 325.18365563
 508.1887358  504.5033405  354.9995375  321.21234858 344.51623314
 656.18000094 351.34338604 263.12936646 267.7790058  426.99346204
 547.91465234 351.88714097 303.60157772 590.88517894 368.36846362
 501.74618395 502.94417431 519.53781643 644.61273492 359.95242387
 336.03428343 391.51586393 302.95646429 357.002556   519.26616112
 527.31512943 309.42027582 363.05155885 490.82235938 294.13425058
 513.69121975 362.65919124 534.74110402 333.074134   443.74326005
 541.14398409 553.11351702 309.29937323 548.03819645 450.63600547]
```

```
[55] predictions = pd.DataFrame({"Predictions": predict}, index=pd.date_range(start=df.index[-1], periods=len(p
     predictions.to_csv("Predicted-price-data.csv")

     #collecting future days from predicted values
     fivedays_df = pd.DataFrame(predictions[:5])
     fivedays_df.to_csv("five-days-predictions.csv")
```

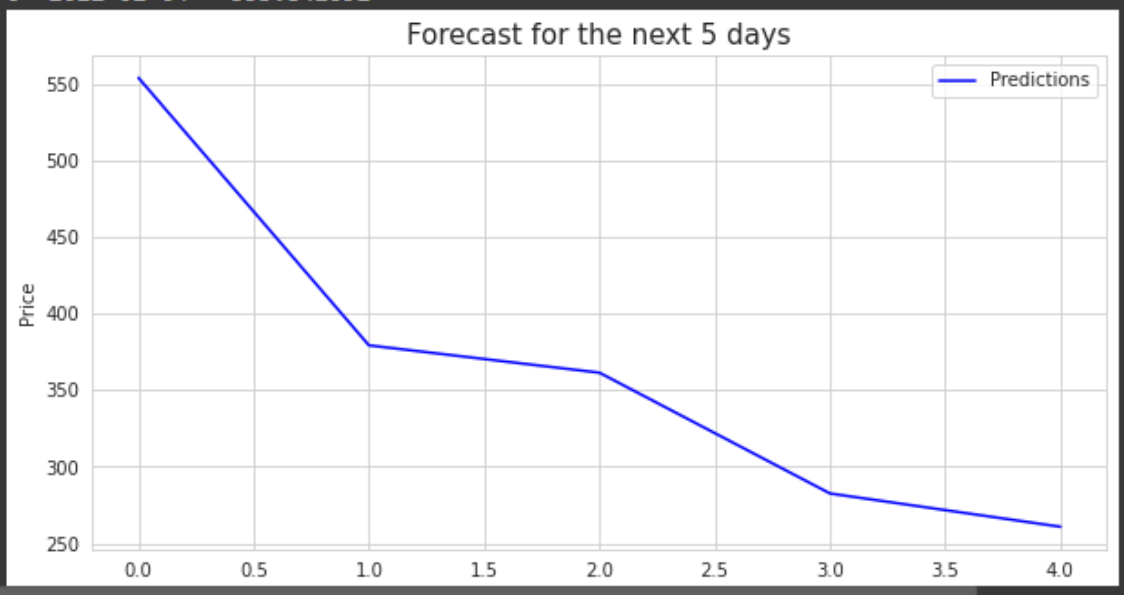
```
print("Mean Absolute Error:", round(metrics.mean_absolute_error(y_test, predict), 4))
print("Mean Squared Error:", round(metrics.mean_squared_error(y_test, predict), 4))
print("Root Mean Squared Error:", round(np.sqrt(metrics.mean_squared_error(y_test, predict)), 4))
print("(R^2) Score:", round(metrics.r2_score(y_test, predict), 4))
print(f'Train Score : {model.score(X_train, y_train) * 100:.2f}% and Test Score : {model.score(X_test, y_t
errors = abs(predict - y_test)
mape = 100 * (errors / y_test)
accuracy = 100 - np.mean(mape)
print('Accuracy:', round(accuracy, 2), '%.')
```

```
Mean Absolute Error: 0.6722
Mean Squared Error: 2.5565
Root Mean Squared Error: 1.5989
(R^2) Score: 0.9998
Train Score : 100.00% and Test Score : 99.98% using Random Tree Regressor.
Accuracy: 99.86 %.
```

```
[57] fivedays_df_pred = pd.read_csv("five-days-predictions.csv")
     buy_price = min(fivedays_df_pred["Predictions"])
     sell_price = max(fivedays_df_pred["Predictions"])
     fivedays_buy = fivedays_df_pred.loc[fivedays_df_pred["Predictions"] == buy_price]
     fivedays_sell = fivedays_df_pred.loc[fivedays_df_pred["Predictions"] == sell_price]
     print("Buy price and date")
     print(fivedays_buy,'\n')
     print("Sell price and date")
     print(fivedays_sell)
     fivedays_df_pred["Predictions"].plot(figsize=(10, 5), color="blue")
     plt.title("Forecast for the next 5 days",size=15)
     plt.xlabel("Date")
     plt.ylabel("Price")
     plt.legend()
     plt.show()
```

```
Buy price and date
   Unnamed: 0  Predictions
4  2022-02-08   260.773759

Sell price and date
   Unnamed: 0  Predictions
0  2022-02-04   553.841692
```



Results

First lets compare the five day predictions of the ARIMA method with what actually happened beyond the dataset - from February 7-11 2022.

Real Prices from Yahoo Stocks

Currency in USD

Download

Date	Open	High	Low	Close*	Adj Close**	Volume
Feb 15, 2022	403.79	409.16	401.01	407.46	407.46	5,392,300
Feb 14, 2022	387.59	409.36	386.89	396.57	396.57	7,202,200
Feb 11, 2022	405.33	411.61	387.65	391.31	391.31	7,558,900
Feb 10, 2022	402.10	408.00	396.36	406.27	406.27	8,452,900
Feb 09, 2022	408.65	412.98	398.79	412.89	412.89	7,738,200
Feb 08, 2022	398.18	406.61	395.83	403.53	403.53	6,818,500
Feb 07, 2022	410.17	412.35	393.55	402.10	402.10	8,232,900
Feb 04, 2022	407.31	412.77	396.64	410.17	410.17	7,789,800

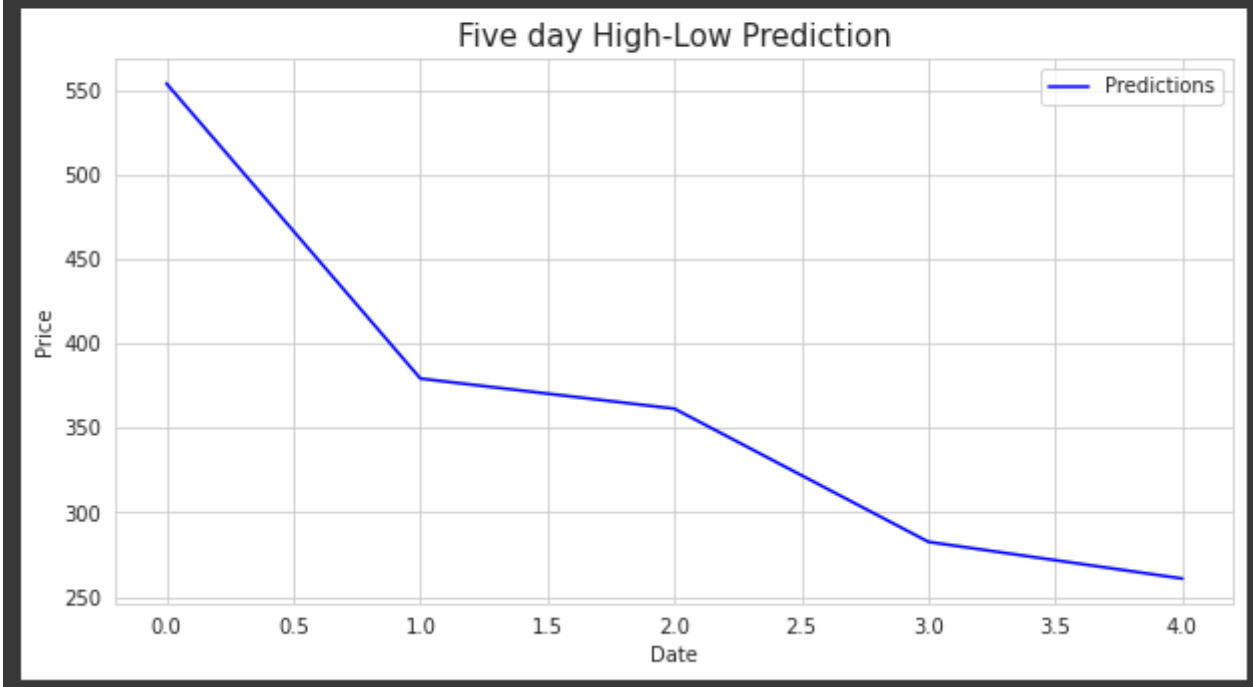
Day	Actual Close Price	ARIMA Prediction
Feb 07 2022	402.10	410.36
Feb 08 2022	403.53	410.56
Feb 09 2022	412.89	410.75
Feb 10 2022	406.27	410.95
Feb 11 2022	391.31	411.15

We can see from the table above that the ARIMA model was somewhat accurate within an error range of 5%. It also got the price movement direction correct 60% of the time.

Now lets have a look at the random forest prediction range.

```
Buy price and date
   Unnamed: 0  Predictions
4  2022-02-08   260.773759

Sell price and date
   Unnamed: 0  Predictions
0  2022-02-04   553.841692
```



The High ended up being 412.98 and the Low 387.65, well within the range of the random forest prediction. I assume the recent volatility led to this widened range and look forward to learning more about ML algorithms and being able to product more accurate predictions.

References

Jainil Shah (2022). Netflix Stock Price Prediction. Kaggle.
Source: https://www.kaggle.com/datasets/jainilcoder/netflix-stock-price-prediction/metadata?resource=download

TETSUYA SASAKI (2022). Netflix Stock Prediction Try by PyCaret TS Alpha. Kaggle.
Source: https://www.kaggle.com/code/sasakitetsuya/netflix-stock-prediction-try-by-pycaret-ts-alpha

VANSHIKHA ANGRISH (2022). Netflix Stock Prediction- Random Forest & Linear R. Kaggle.

Source: https://www.kaggle.com/code/vanshikhaangrish/netflix-stock-prediction-random-forest-linear-r/notebook

Christopher Bratkovics (2019). Exploratory Data Analysis Tutorial in Python. ## The best way to understand your data is by taking the time to explore it. Towards Data Science.
Source: https://towardsdatascience.com/exploratory-data-analysis-tutorial-in-python-15602b417445