Algorithm Assisted Day Trading

1. The problem at hand

According to etoro, a social trading platform, 80% of private day traders are unprofitable over the course of a year. This is due to many different reasons such as lack of patience, lack of knowledge, poor risk management, etc.

There are many factors that one has to look at when deciding to buy, hold or sell a certain stock, my main purpose in this project was not to look at all those factors but to build a function that will help investors get the idea of the future stock price and advice them to hold or sell the stock they already have.

In summary, day traders can use this function alongside other important factors in order for them to decide whether to sell or hold the stock they possess.

2. Data Acquisition and Management

Stock data used in this project can be found in various platforms that offer financial data such as yahoo finance, google finance, Bloomberg, etc. For this project, yahoo finance was used because it is easy to import data (we don't need an API key) and it is commonly used among python (the programming language that was used) users.

To import data into python few things are needed:

- The ticker of the stock you need to import
- Dates: from when to when
- Data source: yahoo finance was used as the source (for this case JPMorgan stock data was used)

1. Importing Data ¶

In [4]:
```
# Setting Dates
start= date(2005,1,1)
end= date(2020,1,1)
```

In [5]:
```
# importing data
## JP Morgan
jpm=DataReader('JPM', data_source='yahoo',start=start, end=end)
jpm.tail(3)
```

Out[5]:

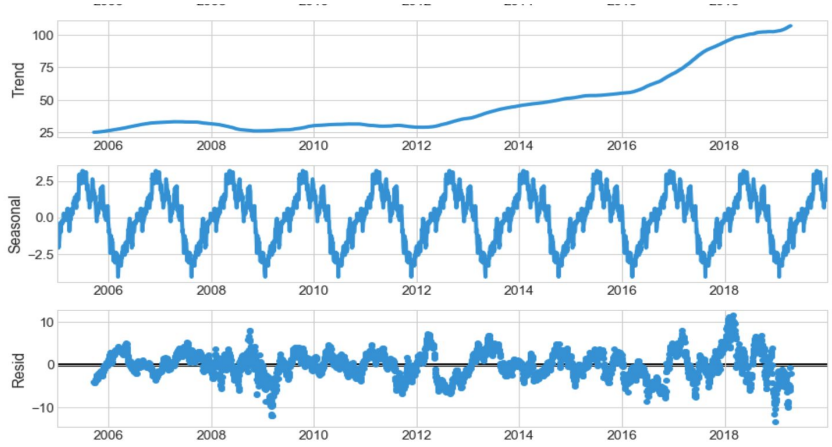
Date	High	Low	Open	Close	Volume	Adj Close
2019-12-27	139.770004	138.669998	139.300003	139.139999	7868200.0	134.264221
2019-12-30	140.080002	138.470001	139.899994	138.630005	6963000.0	133.772079
2019-12-31	139.479996	138.289993	138.509995	139.399994	7201600.0	134.515091

The data that was imported was in CSV format.

3. Exploratory Data Analysis

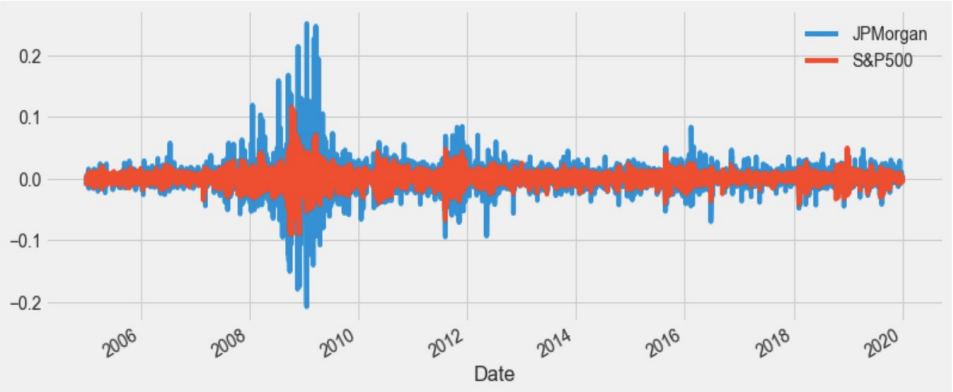
Many traders look at different factors such as the trend and seasonality, how to stock is performing compared with the market, moving average, etc in order to decide what actions they should consider taking. For this project, I looked also at some of those factors in order to understand how the JPMorgan stock was doing.

Trend and Seasonality



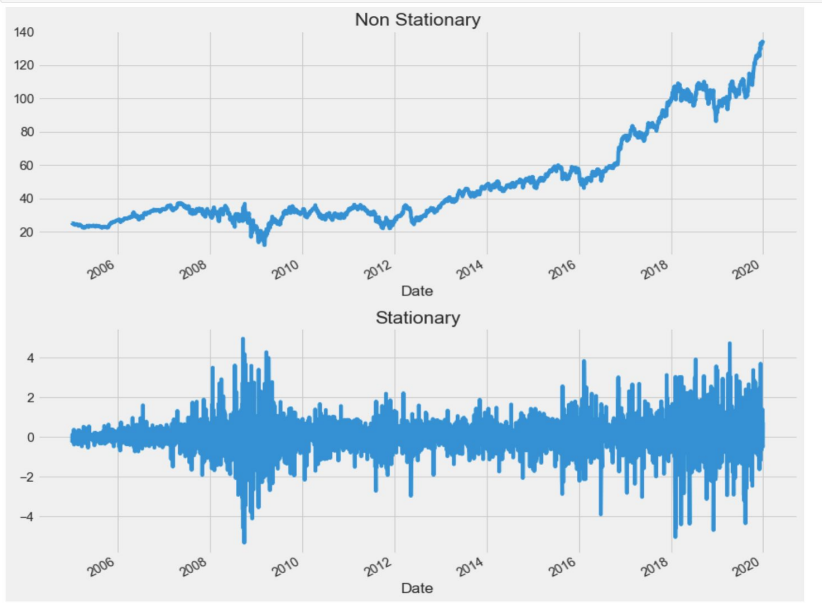
14 Days Moving Average

How the stock is performing compared to the market (S&P 500)



4. Stationarity and Random Walk

Many time series models such as ARIMA models require the data to be stationary but when using Neural networks Models such as LSTM networks, one does not need to transform data into stationary data.



Stationarity, in this case, means the time series do not depend on the time at which it was observed which allows the model not to depend on the trend and seasonality which can affect the value of time series at different times.

Random walk, on the other hand, we use it to know if the current price (today's price) depends on the previous price by setting a null hypothesis that our stock data is a random walk.

```
In [28]: from statsmodels.tsa.stattools import adfuller

In [29]: # Is our stock a random walk

         all_results=adfuller(jpm['Adj Close'])
         all_results
         ## if a stock returns is a random walk
         ## then todays price depends on the previous price plus some noise (white noise)

Out[29]: (2.351935691574812,
          0.9989858643500918,
          30,
          3744,
          {'1%': -3.432097806919567,
           '5%': -2.862312284650772,
           '10%': -2.5671810978277496},
          9445.466702364864)

In [30]: p_value=all_results[1]
         test_statistic=all_results[0]
         (p_value,test_statistic)
         ##if the p_value is less than 5% (0.05) we can reject the null hypothesis
         ## which says that the stock is a random walk
         ## in this case the p_value is higher than 5% which means the stock is a random walk

Out[30]: (0.9989858643500918, 2.351935691574812)
```

The code above shows that we failed to reject the null hypothesis which means that the current price does depend on the previous price.

5. Pre-Processing and Modeling

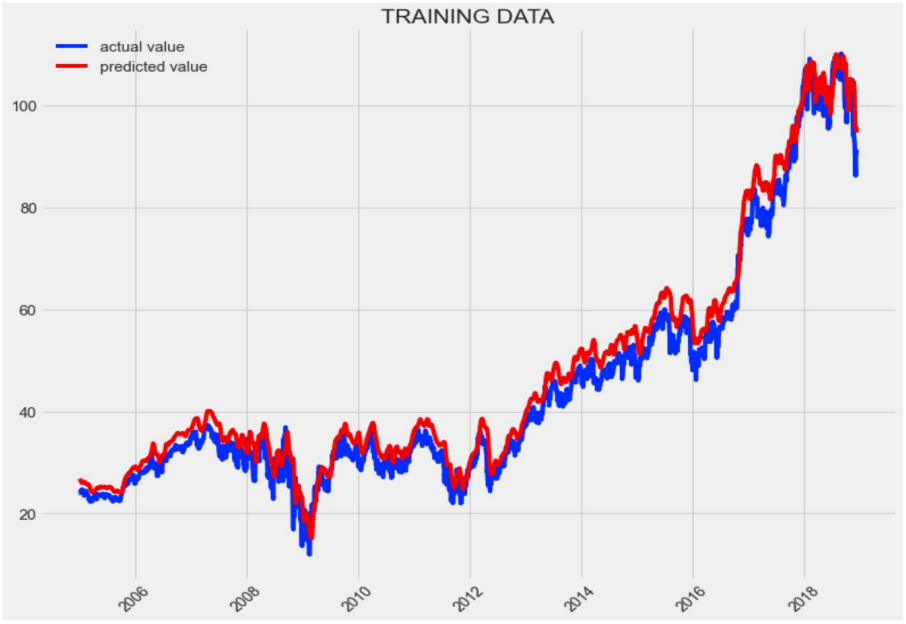
Whenever we are dealing with data in time series or sequential form, in most cases, it is preferable to use Long Short Term Memory(LSTM) networks. LSTM networks are a type of Recurrent Neural Network(a family of neural networks for processing sequential data) capable of learning order dependence in sequence prediction problems. LSTM networks have been used in resolving many different problems such as natural language processing, handwriting recognition, etc.

LSTMs allow previous outputs to be used as inputs, for our case, while the model was trained on a large scale of data, only 14 previous days were used to predict the next day (following those 14 days) price. To implement LSTM networks on our data, Keras/Tensorflow library was used.

```
In [39]: #this kind of shape
         X_train.shape

Out[39]: (3509, 14, 1)

In [40]: X_test.shape

Out[40]: (238, 14, 1)

         9. Modeling

In [41]: # Calling the model
         model = Sequential()
         model.add(LSTM(50,activation='relu', return_sequences=True,input_shape=(X_train.shape[1], X_train.shape[2])))
         model.add(LSTM(50, activation='relu'))
         model.add(Dense(1))
         model.compile(loss='mean_squared_error', optimizer='adam')
         model.summary()

         Model: "sequential1"
         _________________________________________________________________
         Layer (type)                 Output Shape              Param #
         =================================================================
         lstm (LSTM)                  (None, 14, 50)            10400
         _________________________________________________________________
         lstm_1 (LSTM)                (None, 50)                20200
         _________________________________________________________________
         dense (Dense)                (None, 1)                 51
         =================================================================
         Total params: 30,651
         Trainable params: 30,651
         Non-trainable params: 0
         _________________________________________________________________

In [43]: # fitting/training the data
         model.fit(X_train, Y_train, epochs=200, batch_size=100, validation_data=(X_test, Y_test), verbose=0, shuffle=False)

Out[43]: <tensorflow.python.keras.callbacks.History at 0x1f3527c7910>
```

As you can see on the image above, the shape for the output is `(batch_size, timesteps, units)` which shows that 14 days were used to predict the future price. One has to be careful when specifying the kind of shape the input data should have.

The image below demonstrates how the model performed on our JPMorgan data.




```
Train Mean Absolute Error: 3.095375721187597
Train Root Mean Squared Error: 3.6323006090589587
Test Mean Absolute Error: 2.891670611726136
Test Root Mean Squared Error: 3.5791633090698
```

6. The Function

To use the function in order to predict the next day future price, two inputs are needed:

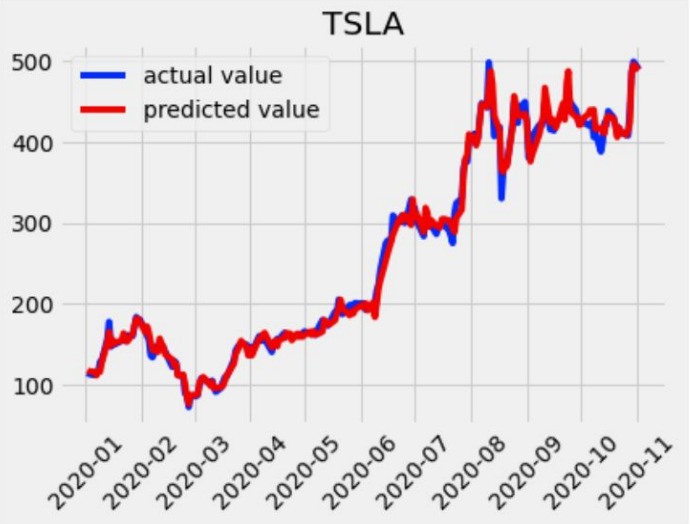
- The ticker of the stock you want to predict
- And the price of shares a person has

The function will predict the next future price and it will compare with the current day and if the future price is bigger than the current price, the function will advise you to hold your shares, if the function predicts that you will lose money the next day it will advise you to sell your shares.

The image below shows an example where inputs into our function are the ticker (Tesla) and the price of shares. As you can see the output has a future price that is

higher than the current price, therefore the function advises us to hold the shares we already have.

```
In [3]: forecast_stock('TSLA',584.5)

Out[3]: {'Status': ['HOLD'],
         'Future Price': [633.7675054881902],
         'Number of shares': [1.1938073517440664]}
```



References:

- Stock Forecasting Code (https://github.com/ursus123/Forecasting-Stocks/blob/main/Stock%20Forecast.ipynb)
- Function Code (https://github.com/ursus123/Forecasting-Stocks/blob/main/Function.ipynb)
- LSTM networks (https://arxiv.org/pdf/1503.04069.pdf)
- Recurrent Neural Networks (https://www.deeplearningbook.org/contents/rnn.html)
- Stationarity and Random walk (https://otexts.com/fpp2/stationarity.html)