# SMARTFOLIO

A Portfolio Management Tool

**By Ankita Mitkari, Aparna Sree, Anushree Bagwe, Sonam Bhatia**

## Table of Contents

## Proof of Concept

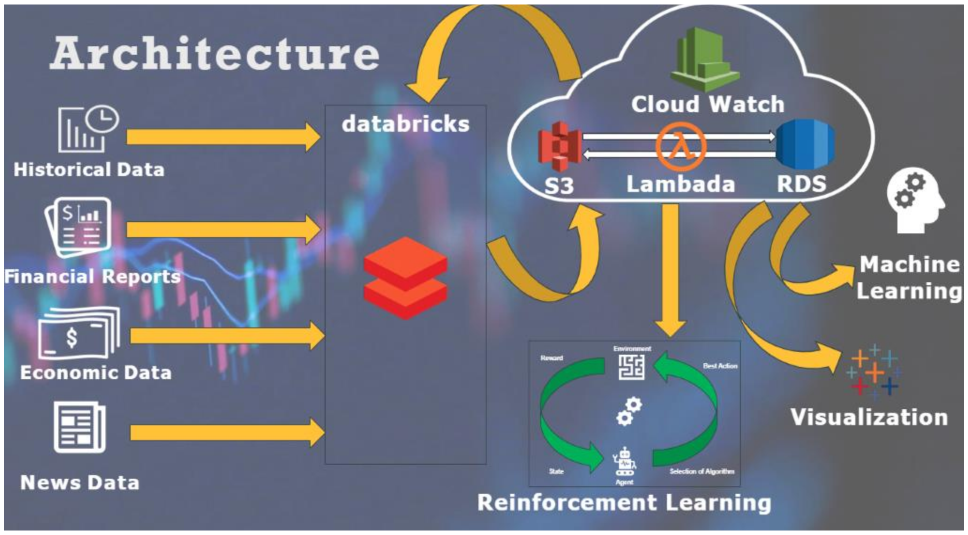
SmartFolio is a proof of concept of a Portfolio Management Tool that leverages data science techniques to stimulate and recommend the optimum strategy for the user based on their targeted objectives. This tool uses data from 4 different data sources such as historical stock prices, company's financial performance report and Economic data for democratization of intelligent Asset Management. It also uses four different types of model fitting such a Multivariable Ridge regression, Gradient boosted regression, Long Short-Term Memory networks, Reinforcement learning for predicting stock values.

The visualization of the product is a tableau front end which the user can use to view the analysis of the different stocks. It also uses a Sentiment analysis (NLP) quotient for decision making.

## Executive Summary

SmartFolio uses an array of the most trending cloud technology to meet the business demand of applying data science technology to portfolio management. To achieve this goal, the SmartFolio team came up with a design architecture which is scalable in terms of storage, processing and flexible to meet future changes. The architecture uses a functional design methodology to separate out codes into functional blocks to make future updates and upgrades seamless. For architecture we used S3, Databricks, Jupyter notebooks, Tableau. For modelling, we used 1) Multi Regression: Predicts +7 days into future, predicts for all Dow 30 stock, attains +0.80 r2 value for majority of Dow 30 tickets 2) LSTM: Predicts +7 days into future for 3 stocks, Attains +0.96 r2 value for all. 3) Reinforcement learning: An automatic bot that can trade to make at least 20% profit. For visualization we used Tableau to create a clean and neat visualization to engage the user to use a portfolio management.

## Technical Architecture



To accommodate the business requirement, we leveraged the best technologies available. We created three separate modules, each one of which are functionally independent of each other.

The first module has the scripts written in Databricks for pulling out the data from the data sources and uploading the raw data in the AWS S3 data repository.

The second module has the scripts for pulling the raw data from the AWS S3 to Databricks and performing data cleaning, transformation and feature engineering. Once the data is prepared for the models, it is again loaded back to AWS S3 at this stage.

The third module has scripts to pull the prepared data from AWS S3 and pass it though the different regression models built in Databricks. The results from the regression models are stored back in AWS S3.

Finally, the raw data and the regression results from AWS S3 are fed to Tableau dashboard where this data is used to make meaningful visualizations that the user can use to make right investment decisions.

The reason why we divided the scripts into different modules based on the functionality is because we wanted to make this product scalable. Right now, we are working on Dow30 index alone, but in the future if the company wishes to expand to include other index too then they should be able to do so by updating only the necessary scripts at a given point of time. This will ensure business continuity.

Also, in this project we have used AWS S3 alone, but we have proposed for the user to use other AWS services like RDS and Lambda for better performance when they plan to scale up.

## ETL

1. The data will be collected from four data sources:
   - Historical data from Alpha Vantage
   - Company Financial data from Wharton Research Data Services (WRDS)
   - Economic Data of the country from FRED – Federal Reserve Economic Data.
   - News API / StockTwits/Reddit data for sentiment analysis

2. Loading Data:

   - Amazon S3 will store all the data that will be used in the project.
   - Implementation of a script in Lambda in AWS to automate the process of populating data from files in S3 to DynamoDB and RDS.
   - Pulling data from DynamoDB and RDS into Databricks

3. Transforming / Cleaning the Data:

   - Removing null values from the dataset.
   - Merging stock data with financial data and economic data.
   - Cleaning news article data (removing unwanted characters and words).
   - Ensuring there is no redundant / duplicate data.

## DATA SOURCING & STORAGE:

### S3 Mounting

S3 is Amazon's data storage service which can be used to store the data. Due to the architecture in which S3 is built, it can store data which can comply with some of the strict regulatory requirements such as HIPPA and EU Data protection Directive.

The S3 mounting part had three components to it.

## 1) Creation of S3 bucket:

To create an S3 bucket, we created an amazon account and created a bucket with the following properties.

Region for S3: We chose the us-east-1 region. At this point since we are in experimentation part, keeping the data in this sector made more sense.

Type for S3: For now, since we are in experimentation stage, we are using the S3 One Zone-Infrequent Access. When we move the application from experimentation to production, we plan to use the S3 Standard-Infrequent Access.

Reason to choose One Zone-Infrequent Access

The reason we need only infrequent access is that our data collection happens only once in 4 hours and read/write access requires only 4 hours.

In addition, since we are not handling high frequency trading directly, but merely give suggestions to a human agent, the data rate and volume is not as high.  This choice of data storage will thus help us keep the cost down while still maintain the minimum requirements.

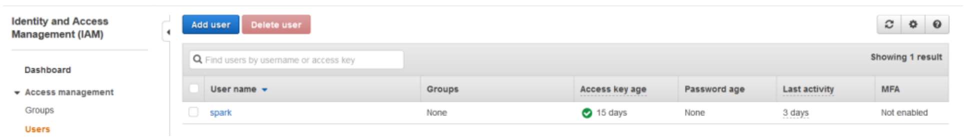
We initially thought of enabling the server access logging to see how our data is used. For now, we are just logging and not really using it for further analysis. Once we create a good enough client base, we might be able to use this logging to understand customer behavior better.

More details can be obtained in this link. [1]

## 2) Creating Access controls to S3

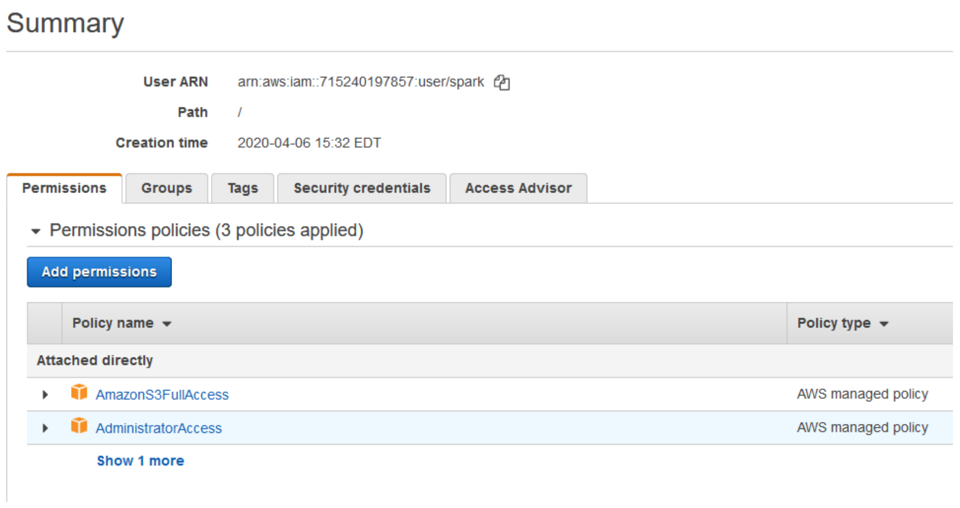
This was arguably the hardest part in getting S3 setup accessible from Databricks. To give the access we worked with the Identity and Access Management resources.

At first, we created a user in the IAM section of the AWS.


Identity and Access Management (IAM)
Dashboard
Access management
Groups
Users
Add user
Delete user
Find users by username or access key
Showing 1 result

| | User name | Groups | Access key age | Password age | Last activity | MFA |
|---|---|---|---|---|---|---|
| | spark | None | ✓ 15 days | None | 3 days | Not enabled |

Then we added permissions, for this user for S3 access.


Summary

| | |
|---|---|
| User ARN | arn:aws:iam::715240197857:user/spark |
| Path | / |
| Creation time | 2020-04-06 15:32 EDT |

| Policy name | Policy type |
|---|---|
| Attached directly | |
| AmazonS3FullAccess | AWS managed policy |
| AdministratorAccess | AWS managed policy |

After these steps, we had to go back to the bucket and specify the user just created in the IAM Section.

### 3) Accessing S3 from Databricks.

To access the data from databricks, we used the following keys: ACCESS key, Secret key. Both are available in the IAM section.

We used the urlib library and dbutils to mount the S3 as a drive in Databricks. After this, the data in S3 was available seamlessly from Databricks.

We had a couple of hiccups when we originally opened the bucket. We had to wait about 5 minutes, before the S3 bucket was accessible from data bricks.

## Data sources

In machine learning and predictive modeling, one of the most important steps is to collect good data, perform the necessary cleaning steps and recognize the limitations.

**Historical data:**

Historical data in the context of stock market analysis is the history of stock values over time.

For the purpose of this project we collected the data from 2010 to 2019. (10 years of data), for about 10 indicators.

Alphavantage is a leading provider of real-time Free APIs and stock historical data. Alphavantage is a leading provider of real-time Free APIs and stock historical data. The Alphavantage API was grouped into four sectors: 1) Stock Time Series Data, 2) Physical and Digital / Crypto Currencies (for example, Bitcoin), 3) Technical Indicators, and 4) Sector Performance. All APIs in Alphavantage are real time.i.e. the latest data points are based on current trading

For the first stage of implementation of the model, we have extracted the technical indicators and the time series data from the year 2010 to 2019.

The indicators are gathered from Alphvantage. The fields that we finally used in the prediction include: Close,  SMA , EMA , MACD , MACD_Hist , MACD_Signal , RSI , SlowK , SlowD , ADX , CCI. The close is the predicted stock value at the end of the day.

For extracting historical data , we have made use of API from Alphavantage. We used the API call get_daily to get the daily values. We also used the API TIME_SERIES_INTRADAY to

**Financial data for a company:**

The Financial data for the company is obtained from Wharton Research Data Service - WRDS (https://wrds-web.wharton.upenn.edu/wrds/). Wharton Research Data Services have company's financial data over the years from 1986 and it includes all Cash Flow Statements, Balance Sheets, Income Statements on a quarterly and annual basis.

The Company Financial Data is used to calculate Year over Year Revenue and Earnings Growth as required by our strategy.

**Economic data from Federal Reserve Economic Data:**

Gross Domestic Product (GDP) is the monetary value of all finished goods and services made within a country during a specific period. GDP provides an economic snapshot of a country which is used to estimate the size of an economy and growth rate.

We are fetching the GDP Data from https://fred.stlouisfed.org/series/GDP in a CSV File. GDP data for US since 1947 is available.

The stock market is often a sentiment indicator that can impact gross domestic product (GDP) either negatively or positively. Our requirement is to show the impact on GDP, hence only the required GDP data is fetched from macro-economic data.

**Unstructured Data for Sentiment Analysis:**

For this project apart from the primary data which focuses on the stock market and purely based on finance and numbers, we are collecting the unstructured data. The text data will allow us to figure out the current news about the Dow Jones companies we want to track and the sentiment and thoughts or opinion the general public has on the stock market.
We used three sources to collect this data:

1. NewsAPI (https://newsapi.org/)
Using this API, we are fetching news articles from over 50,000 news sources. We are querying this API to fetch news articles for each of the companies in the Dow 30 Index and any articles mentioning Dow 30 or Dow Jones 30. These articles are in JSON format. We are using the free version of this API, which has some limitations. The free version only gives access to data for the past one month from the date when the data is called.

2. Reddit (https://www.reddit.com/dev/api/)
For fetching data from this API we are using the PRAW library which is a Python Reddit API Wrapper. We are selecting five or more famous sub-reddits which are related to the stock market or the Dow Jones Index and the data pulled from these sub-reddits is queried accordingly for the companies we want. The fetched data is saved as a CSV file.

3. StockTwits (https://api.stocktwits.com/developers/docs)
StockTwits is a social media platform like Twitter but it is limited to investors, traders and entrepreneurs who want to share ideas about stocks, cryptocurrencies, forex etc. Through this API we

are fetching data for each of the posts or tweets containing words related to the companies in the Dow 30 Index or which references the Dow 30 index itself. The response formats for this API is JSON.
Data collected from all these sources is mounted onto S3 and is being used for sentiment analysis.

## Data cleaning & Transformation :

All the data cleaning and transformation was done in Databricks. Two columns had null values. We removed these columns. We also removed the columns which had a very high coefficient of correlation with the target “close”. Since we are predicting the close stock value, we removed the other features such as “open”, “high”, “low”. This helped us prevent any data leakage.

We transformed the date field as another column and we created new features such as the day of the week, month, date as double value, etc.

The date as a double value is a particular feature, this was done to make sure that we can represent the data as a time series.

We also created fields such as SMA7, SMA14, SMA 28, SMA50 and SMA 200. These are the slow-moving average calculated for 7 days, 14 days, 28 days, 50 days, and 200 days respectively.

NLP Cleaning:

The unstructured data fetched from news articles, forums and tweets is used for analyzing sentiments through natural language processing. This data will largely be used for visualizations but before performing any analysis, we needed to clean and pre-processes the data.
We have performed text normalization. The normalization includes converting all letters to lower case and removing numbers which are not useful for our analysis. We have removed punctuations and whitespaces. Then using the Natural Language Toolkit offered by Python we have tokenized the words and removed stop words that do not carry any important meaning. Using the Natural Language Toolkit we are also stemming the words to reduce them to their root forms and even performing lemmatization.

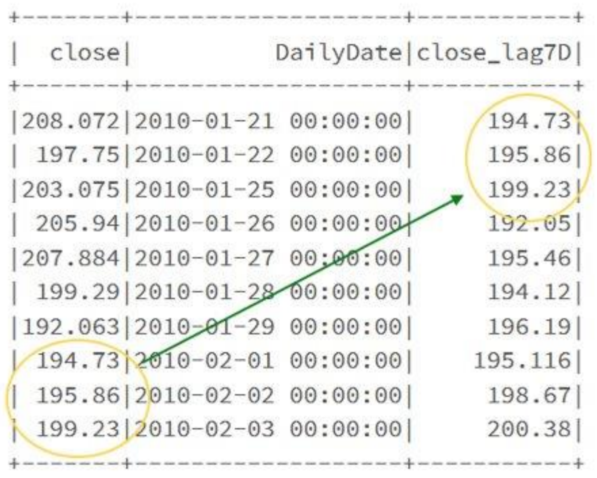
## Feature engineering

We have created a new feature close_lag7D from values that are 7 days into the future. This will help us with training and testing and the model would be able to predict 7 days into the future. We have predicted the close_lag7D using all the features. When we started predicting 7 days into the future , we noticed that the r2 value made more sense now. The r2 value for Apple was 0.82 . But, for some other tickers like Boeing Company and Travelers Company was getting an r2 value of 0.06 and 0.016. This means that when we started predicting into the future, the r2 values made more sense, although some of the stock performances went down. This was the initial step in feature engineering.

More features like date were added as part of feature engineering. It was converted to a double value first before added as a feature. Another feature we considered was the month. This is because we wanted to capture the seasonal variations if there are any. The year also was added as the feature, which was added as a relative value. This means that since our data starts in 2010, it is considered as year 0 and the 2019 was considered as year 9. The week was also added as another feature, so the first week of 2010 is week 0 and it runs to week 387(over 8 years).

We have also tried predicting 7-day, 14 day and 28 day into the future.

Other features added were SMA14, SMA28, SMA50 and SMA 200. Another feature calculated was a feature delta which is the difference between SMA200 and SMA50.



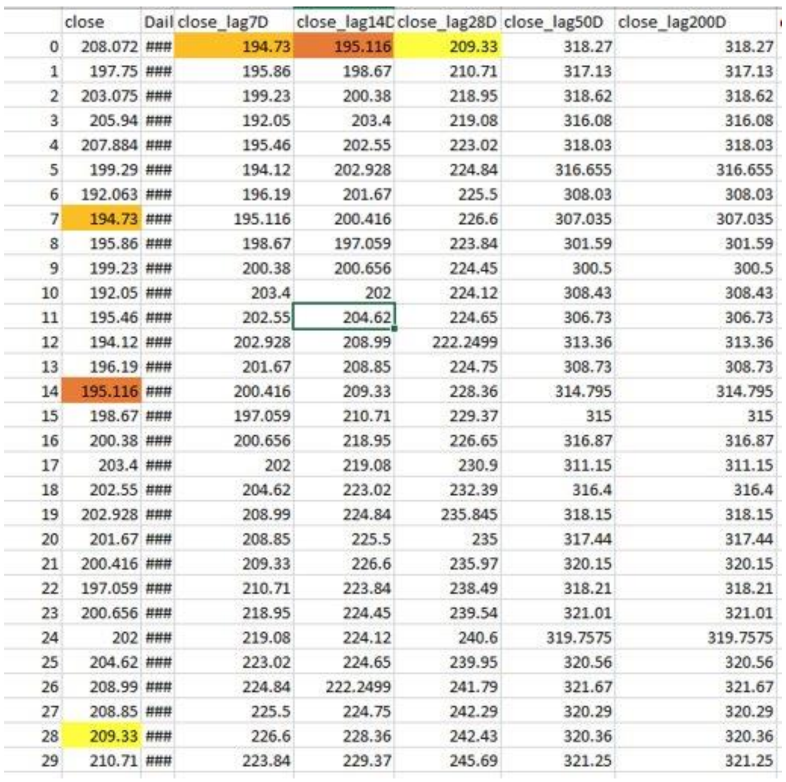
```
+-------+-------------------+-----------+
|  close|          DailyDate|close_lag7D|
+-------+-------------------+-----------+
|208.072|2010-01-21 00:00:00|     194.73|
| 197.75|2010-01-22 00:00:00|     195.86|
|203.075|2010-01-25 00:00:00|     199.23|
| 205.94|2010-01-26 00:00:00|     192.05|
|207.884|2010-01-27 00:00:00|     195.46|
| 199.29|2010-01-28 00:00:00|     194.12|
|192.063|2010-01-29 00:00:00|     196.19|
| 194.73|2010-02-01 00:00:00|    195.116|
| 195.86|2010-02-02 00:00:00|     198.67|
| 199.23|2010-02-03 00:00:00|     200.38|
+-------+-------------------+-----------+
```

To summarize, using feature engineering we started getting much better r2 values than without the added features. Some of the stocks which gave us the best r2 values are American Express, Walt Disney and Goldman Sachs. Walmart and Visa are some of the stocks which gave worst r2 values. Even the worst stocks still gave better r2 value than the linear regression models.

We have also checked whether adding all these features have improved the r2 value. **(FIG)**

We noticed that some of the ticker values improved drastically. For example, for BA (Boeing) the r2 value without the added features were terrible. But, after adding the engineered features, we got an r2 value close to 0.96. The same thing happened for most of the stocks where the r2 value increased.

However, we were surprised to find that the r2 value decreased for some other tickers. We have concluded that this might be due to over training. Another thing we observed was that



| | close | Dail | close_lag7D | close_lag14D | close_lag28D | close_lag50D | close_lag200D |
|---|---|---|---|---|---|---|---|
| 0 | 208.072 | ### | 194.73 | 195.116 | 209.33 | 318.27 | 318.27 |
| 1 | 197.75 | ### | 195.86 | 198.67 | 210.71 | 317.13 | 317.13 |
| 2 | 203.075 | ### | 199.23 | 200.38 | 218.95 | 318.62 | 318.62 |
| 3 | 205.94 | ### | 192.05 | 203.4 | 219.08 | 316.08 | 316.08 |
| 4 | 207.884 | ### | 195.46 | 202.55 | 223.02 | 318.03 | 318.03 |
| 5 | 199.29 | ### | 194.12 | 202.928 | 224.84 | 316.655 | 316.655 |
| 6 | 192.063 | ### | 196.19 | 201.67 | 225.5 | 308.03 | 308.03 |
| 7 | 194.73 | ### | 195.116 | 200.416 | 226.6 | 307.035 | 307.035 |
| 8 | 195.86 | ### | 198.67 | 197.059 | 223.84 | 301.59 | 301.59 |
| 9 | 199.23 | ### | 200.38 | 200.656 | 224.45 | 300.5 | 300.5 |
| 10 | 192.05 | ### | 203.4 | 202 | 224.12 | 308.43 | 308.43 |
| 11 | 195.46 | ### | 202.55 | 204.62 | 224.65 | 306.73 | 306.73 |
| 12 | 194.12 | ### | 202.928 | 208.99 | 222.2499 | 313.36 | 313.36 |
| 13 | 196.19 | ### | 201.67 | 208.85 | 224.75 | 308.73 | 308.73 |
| 14 | 195.116 | ### | 200.416 | 209.33 | 228.36 | 314.795 | 314.795 |
| 15 | 198.67 | ### | 197.059 | 210.71 | 229.37 | 315 | 315 |
| 16 | 200.38 | ### | 200.656 | 218.95 | 226.65 | 316.87 | 316.87 |
| 17 | 203.4 | ### | 202 | 219.08 | 230.9 | 311.15 | 311.15 |
| 18 | 202.55 | ### | 204.62 | 223.02 | 232.39 | 316.4 | 316.4 |
| 19 | 202.928 | ### | 208.99 | 224.84 | 235.845 | 318.15 | 318.15 |
| 20 | 201.67 | ### | 208.85 | 225.5 | 235 | 317.44 | 317.44 |
| 21 | 200.416 | ### | 209.33 | 226.6 | 235.97 | 320.15 | 320.15 |
| 22 | 197.059 | ### | 210.71 | 223.84 | 238.49 | 318.21 | 318.21 |
| 23 | 200.656 | ### | 218.95 | 224.45 | 239.54 | 321.01 | 321.01 |
| 24 | 202 | ### | 219.08 | 224.12 | 240.6 | 319.7575 | 319.7575 |
| 25 | 204.62 | ### | 223.02 | 224.65 | 239.95 | 320.56 | 320.56 |
| 26 | 208.99 | ### | 224.84 | 222.2499 | 241.79 | 321.67 | 321.67 |
| 27 | 208.85 | ### | 225.5 | 224.75 | 242.29 | 320.29 | 320.29 |
| 28 | 209.33 | ### | 226.6 | 228.36 | 242.43 | 320.36 | 320.36 |
| 29 | 210.71 | ### | 223.84 | 229.37 | 245.69 | 321.25 | 321.25 |

## Modeling:

### INTRODUCTION

1. Our aim with the modeling section is to predict the close value of all the stocks in Dow 30. We aimed to predict the close value at the end of each day.
2. We used the following models to achieve this objective.
    a. Multi Regression model
        i. In the multi regression model, we predicted the close price 7 days into the future and we used the features we collected from the data collection stage and the additional features generated from the feature engineering stage. We used R2 value as a metric to evaluate the fit of our models.
    b. Gradient Boosted tree regression
        i. In gradient boosted regression, we predicted the close price 7 days into the future using the same features used for the linear regression. We compared how Gradient boosted tree regression worked compared to the linear regression model.
    c. Long Short Term Memory (LSTM) Network

        i. In LSTM, we predicted the close price 1 day and 7 days into the future. We used the close price as a feature. We used R2 value as a metric to evaluate the models.

   d. Reinforcement Learning
        i. In reinforcement learning, we predicted the close price 1 day into the future and developed a trading bot. We used “close price” as the target variable to be learned. The reinforcement learning agent learned to manage a stock portfolio of three companies (Microsoft, Apple, IBM). The goal of the agent here is to maximize the total value of the portfolio.

## MULTI REGRESSION

### Problem Description

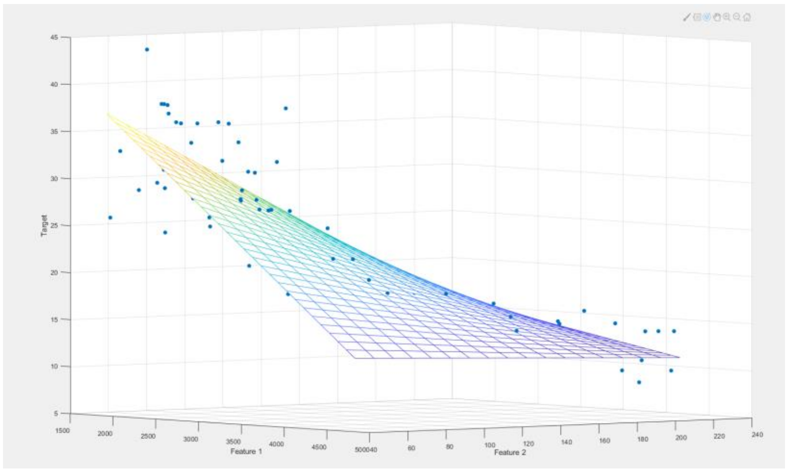
To predict the close value of all Dow 30 stocks, we created 30 different multi regression models. The behavior of the model depends on how the weights are distributed. Early on we found that the effect of the predictor on the target value varies from one ticker to another. Therefore, we created one model for each of the tickers. That is the reason why we went for 30 different models for each of the tickers.

We divided the training and test data such that the data from 2010 to July 2016 was taken as the test data and the data from July 2016 till end of 2019 was taken as the test data. The r2 value for most of the tickers were in the range of 0.94 to 0.99. Although a high r2 value is expected from a model of high performance, these values were extremely high. We explored more and found that some data was overlapping. For example, the way we have created our model is in such a way that, the model can predict close value if it is given all the other features such as SMA, EMA, MACD etc.

The idea was to strictly use the past data as features and future data as the target. So, to elaborate more on that idea, if we are trying to predict the close value of the market today, we need some of the indicators from today morning. However, the business requirement was to be able to predict the future. So, we decided to predict at least 7 days into future.

Keeping that in mind, we started looking into the concept of lead and lag in time series analysis. Lead moves values backward in time and lag moves forward in time. What this means is that, if we lag all the predictors in the dataset, say by 7 days, we found that we could do the fitting and predict 7 days into the future. Adding a lag to all the predictors is equivalent to adding a lead to the target output value.

## Mathematical modeling



In the figure, you can see how there are two variables Feature1 and Feature 2 which is used as independent variables to predict an output variable "Target". This is how the multi regression is different from a regular linear regression.

Technically the multi regression we are using is known as ridge regression. The reason to call it ridge regression is the regularization factor built into the mathematical model. To explain this in detail, we are showing the equation of the ridge regression we used.

$$\beta^{ridge} = minimize \sum_{i=1}^{N} \|y - w_i\|_2^2 + \lambda \sum_{i=1}^{N} w_i^2$$

## Testing and Evaluation with initial features

For the first run, we ran the linear regression with just the original features.

We got very high values for r2 value. We understood this is because we were predicting the closing value of the stock using features available on the same day. We should mention that we did a very good fitting because we did a parameter search using param grid. Since we used data bricks this was very fast and effective. However, the business value of predicting the stock value comes in predicting the future stock value using indicators from today. We have explained the lag and lead features we introduced in the feature engineering section. Although it was explained in the feature engineering part it is a necessary item to enable the business proposition.

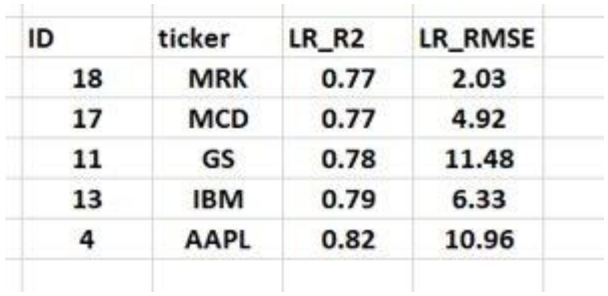
| ID | ticker | LR_R2 | LR_RMSE |
|---|---|---|---|
| 21 | WBA | 0.991765 | 0.552303 |
| 3 | GS | 0.992316 | 2.11921 |
| 6 | INT | 0.99263 | 0.259132 |
| 5 | IBM | 0.992898 | 1.143677 |
| 10 | MRK | 0.992951 | 0.390317 |

In the table you can see that the R2 value was close to 0.99 for majority of the stocks and will be close to 0.96 for the rest of the stocks. This is a trivial problem and something we failed to see at the start of the project. It was a good learning experience though to understand the reason and to modify the technical details to fit the business problem.

## Testing and Evaluation with initial features for close 7 days ahead

For predicting into the future, we started predicting the "close_7D" feature.

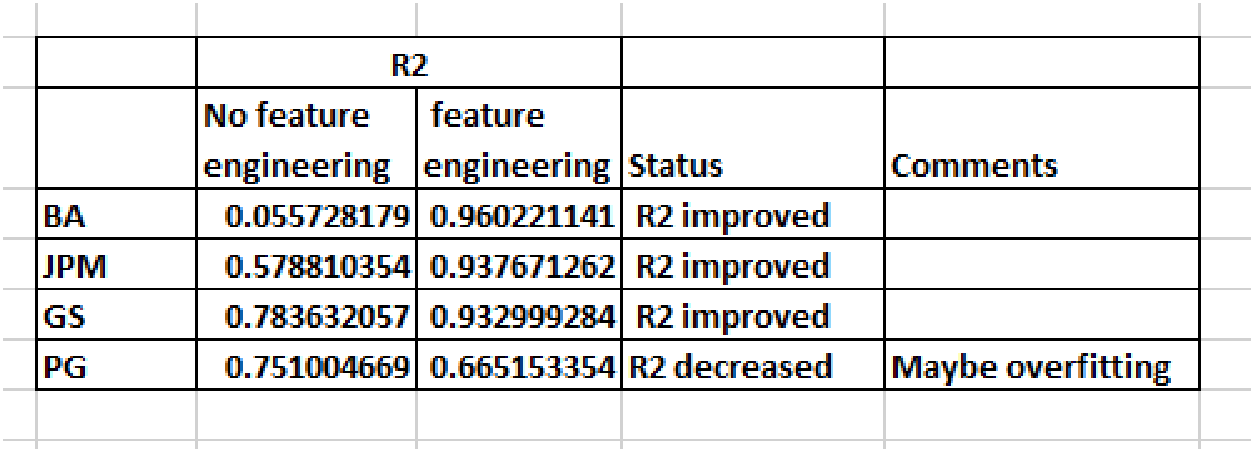
The R2 value for predicting 7 days into the future,



| ID | ticker | LR_R2 | LR_RMSE |
|---|---|---|---|
| 18 | MRK | 0.77 | 2.03 |
| 17 | MCD | 0.77 | 4.92 |
| 11 | GS | 0.78 | 11.48 |
| 13 | IBM | 0.79 | 6.33 |
| 4 | AAPL | 0.82 | 10.96 |

Once we realized that predicting into the future is the harder part, we started using the predictors which had a lead in them. At this point

## Testing and Evaluation with added features and close_7 days ahead



| | R2 | | | |
|---|---|---|---|---|
| | No feature engineering | feature engineering | Status | Comments |
| BA | 0.055728179 | 0.960221141 | R2 improved | |
| JPM | 0.578810354 | 0.937671262 | R2 improved | |
| GS | 0.783632057 | 0.932999284 | R2 improved | |
| PG | 0.751004669 | 0.665153354 | R2 decreased | Maybe overfitting |

To improve the R2 value, we added the features found during the feature engineering stage. The effects of these added new features was extremely interesting. In table above we are showing some of the stocks which improved the r2 value significantly. We also noticed that for some stocks, which already had a very good R2 value, the R2 value decrease. We think this is mostly due to over fitting. To get around this, we consider the following potential workarounds.

```
formula = "{} ~ {}".format("close_lag7D", " + ".join(columns))
print("Formula : {}".format(formula))
rformula = RFormula(formula = formula)
lr = LinearRegression()
pipeline = Pipeline(stages=[rformula, lr])
# Parameter grid
paramGrid = ParamGridBuilder()\
         .addGrid(lr.regParam,[0.01, .04])\
          .build()
cv = CrossValidator()\
      .setEstimator(pipeline)\
      .setEvaluator(RegressionEvaluator()\
                       .setMetricName("r2"))\
      .setEstimatorParamMaps(paramGrid)\
      .setNumFolds(3)

cvModel = cv.fit(train_data)
cvModel.avgMetrics
predictions = cvModel.transform(test_data)
evaluator = RegressionEvaluator(labelCol="label",
                                  predictionCol="prediction",
                                  metricName="rmse")

y_true = predictions.select('label').toPandas()
y_pred  = predictions.select('prediction').toPandas()

r2_score = sklearn.metrics.r2_score(y_true, y_pred)
print('r2_score: {0}'.format(r2_score))
```

## Potential Workarounds

One of the ideas we thought of was to use the best model that fits. In the table below you can see the R2 value for each of the tickers with initial features and with additional engineered features. As you can see, some models performed better with the added features while some other tickers got worse. Our recommendation is to use the model with the best r2 values.

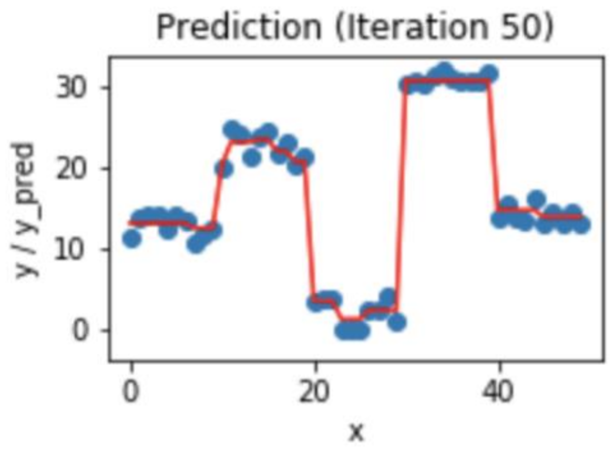
In the table below, we highlight the best R2 value obtained for each ticker.

| ticker | LR_R2 initial features | LR_R2 with initial + engineered features |
|---|---|---|
| AAPL | 0.816992108 | -0.770049639 |
| AXP | 0.260129662 | 0.896128953 |
| BA | 0.055728179 | 0.960221141 |
| CAT | 0.385196432 | 0.706895756 |
| CVX | 0.217265162 | 0.799392748 |
| DIS | -0.311825085 | 0.904728904 |
| GS | 0.783632057 | 0.932999284 |
| HD | 0.70902933 | 0.818385468 |
| IBM | 0.789141854 | 0.44410283 |

| | | |
|---|---|---|
| INT | 0.38143895 | 0.635999329 |
| JNJ | 0.361373758 | 0.601879609 |
| JPM | 0.578810354 | 0.937671262 |
| KO | 0.393412382 | 0.513343017 |
| MCD | 0.774372465 | 0.734922389 |
| MMM | 0.219752024 | 0.778889709 |
| MRK | 0.771377507 | 0.474992888 |
| MSF | 0.049433406 | 0.668460243 |
| NKE | 0.355689531 | -0.217194718 |
| PFE | 0.606532753 | 0.582555572 |
| PG | 0.751004669 | 0.665153354 |
| TRV | 0.160569486 | 0.563583992 |
| UNH | -0.141664327 | 0.860493536 |
| UTX | 0.717660602 | 0.790991837 |
| V | -0.20149655 | 0.404353203 |
| VZ | 0.630042679 | 0.625990842 |
| WBA | 0.582065659 | -0.045170119 |
| WMT | 0.294419311 | 0.352613372 |
| XOM | 0.57628482 | 0.659777842 |

## GRADIENT BOOSTING REGRESSION

Gradient boosting is a machine learning technique for problems of regression and classification, which generates a predictive model in the form of an ensemble of weak predictive models, usually decision trees. It constructs the model in a stage-wise manner as do other methods of boosting, and it generalizes them by allowing an arbitrary differentiable loss function to be optimized.



“Image source” - https://medium.com/mlreview/gradient-boosting-from-scratch-1e317ae4587d

In the image below, a gradient boosted technique graph is shown which fits into a step wise dataset with very good fitting. This is just to show the effectiveness of gradient boosted. From the literature, we know that gradient boosted techniques can easily overfit. So we were careful enough to add hyperparameters such as

We decided to use Gradient Boosted Regression because many winning solutions on Kaggle have made use of XgBoost for predictions. We added this to our Databricks pipeline. We ran the gradient boosted tree regression for only 5 stocks. The stocks we ran are United Technology Corporation, Home Depot, United Health Group, American Express and JP Morgan & Chase. We have also uploaded these models to S3.

The results we got from slightly higher for some stocks, but it was not something drastically different from the linear regression values.

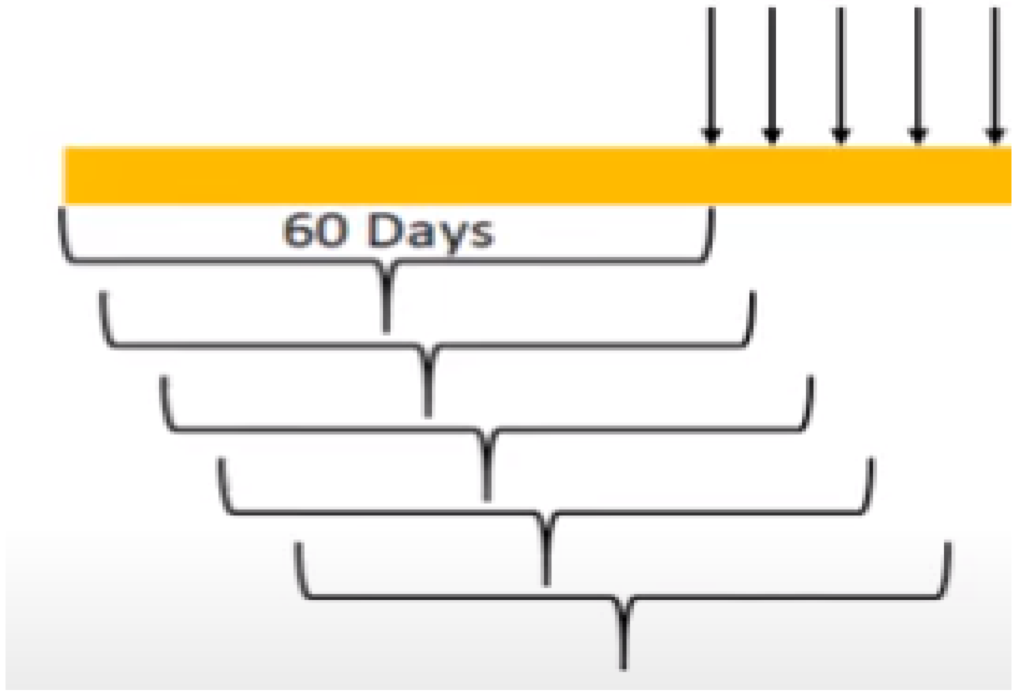
| ticker | LR_R2 | LR_RMSE |
|---|---|---|
| UTX | 0.90 | 2.18 |
| HD | 0.66 | 2.27 |
| UNH | 0.74 | 9.74 |
| AXP | 0.71 | 2.69 |
| JPM | 0.83 | 3.65 |

## Long Short-Term memory

### Introduction

Long Short-Term memory is one of the most successful RNNs architectures.
It not only processes a single datapoint, but it processes series of data points in sequence like speech and videos with great speed and accuracy. LSTM are capable of learning long term dependencies. It is explicitly designed to remember information for long term making it suitable to deal with time series dataset like the one used in this project.

### Data splitting



For the purposes of this project, we are predicting the closing price of the stock one day in future using the data for past 60 days. For first iteration, we take first 60 closing prices i.e. from index 1 to 60 and use that data to predict 61st day close price or data at index 61st index. The data for the first 60 days is added to X_train and the 61st day close price is added to Y_train.
For second iteration, the model will drop the data at index number 1 or 1st closing price and take the data for next 60 days to predict close price for 62nd day.
Similarly, all the data will be divided into test and train datasets over several such iterations.

### LSTM Architecture

We tried several combinations of number of layers and neurons to best fit the data we have.
After trying, multiple combinations, below is the architecture that gave us the best R2 values.

```
#Build the LSTM network model
model = Sequential()
model.add(LSTM(units=50, return_sequences=True,input_shape=(x_train.shape[1],1)))
model.add(LSTM(units=50, return_sequences=False))
model.add(Dense(units=25))
model.add(Dense(units=1))
```

We added two LSTM layers with 50 Neurons each and one dense layer with 25 neurons and another dense layer with 1 neuron. We referred to code online to create the LSTM [10] .
We compiled this model using Adam optimizer to improve upon the loss function. The loss function that we are using is Mean Squared error. This will let us know the performance of our model on training data set.

```
#Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')
```

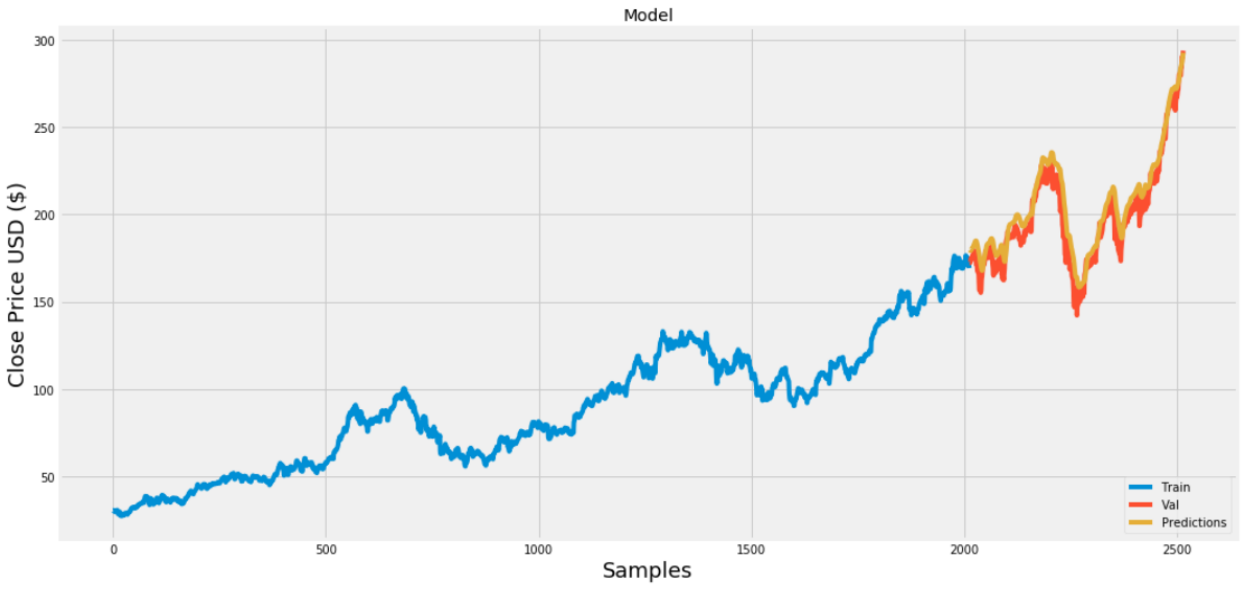
After compiling we rant his model for 1 epoch and it completed with loss function value as `4.1816e-040s.`

## Results

Since we are working with 30 tickers of Dow 30 index and this model needs to be fit individually for each of the ticker. The computation time taken for all the 30 tickers would be a lot that's why we choose to run this model for on three tickers.

**Below are the results for Apple Stock data–**

**R2 value: 0.911**

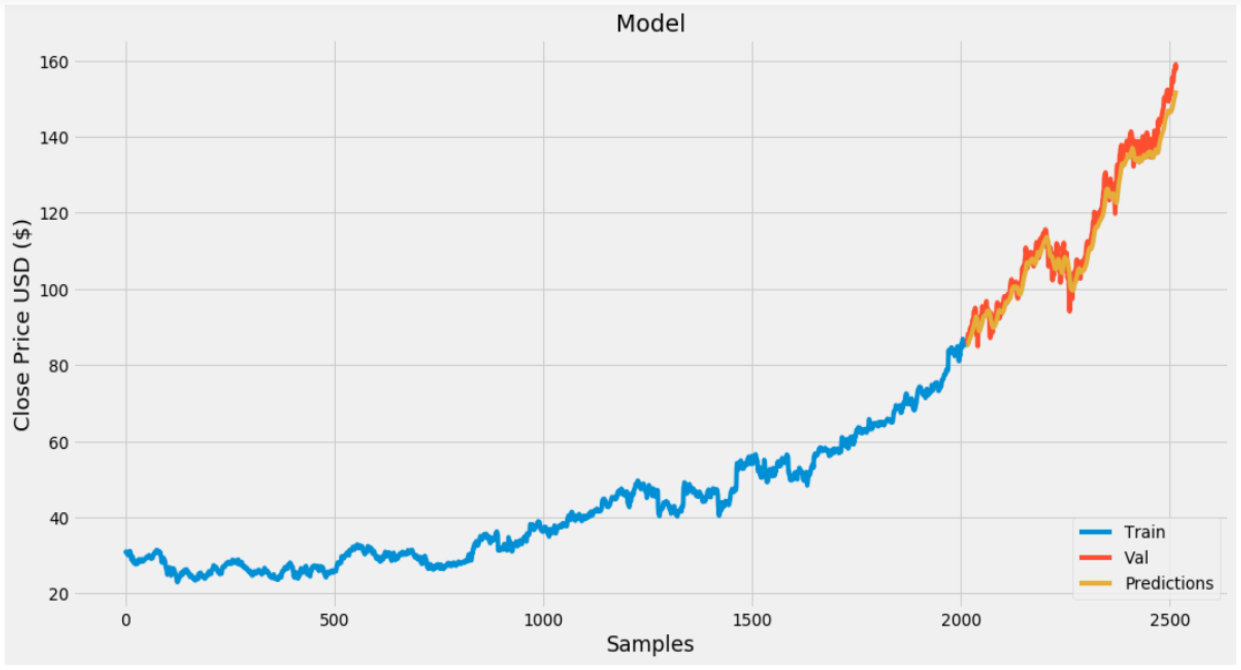


When back tested the data for few samples below were the results –

| Actual Close | Predictions |
|---|---|
| 172.259995 | 179.530884 |
| 172.229996 | 179.246902 |
| 173.029999 | 179.174316 |
| 175.000000 | 179.316193 |
| 174.350006 | 179.789886 |
| 174.330002 | 180.264404 |
| 174.289993 | 180.680969 |
| 175.279999 | 181.011276 |
| 177.089996 | 181.385956 |
| 176.190002 | 181.960480 |

**Below are the results for Microsoft Stock data–**

**R2 value: 0.96**



| Actual Close | Predictions |
|---|---|
| 85.540001 | 85.243675 |
| 85.949997 | 85.241837 |
| 86.349998 | 85.276344 |
| 87.110001 | 85.367226 |
| 88.190002 | 85.558388 |
| 88.279999 | 85.894699 |
| 88.220001 | 86.272308 |
| 87.820000 | 86.624039 |
| 88.080002 | 86.882477 |
| 89.599998 | 87.098511 |

As seen from the results, the actual close price data is closely matching to the predicted stock price data.

## REINFORCEMENT LEARNING

### Problem Description

An agent begins with a fixed sum of money and data on past stock prices, their past prices over 10 years of time. It will make a buy / sell / hold decision for the long-term maximization of the portfolio value (uninvested cash + present value of stocks). To simplify the problem, we have considered a group of 3 stocks in our portfolio. The training data consists of time series data of daily stock prices. Essentially, the agent needs to be able to understand how the stock price datapoints will vary in the future and take actions accordingly to maximize its portfolio.

### Mathematical Modelling

Considering the stochastic and interactive nature of the trading market, we model the stock trading process as a Markov Decision Process (MDP) as shown in Figure below and is specified as follows:

• States:

The state consists of three components:

1. The numbers of shares of the 3 stocks that can be bought, sold or held by the agent - n1, n2, n3.

2. The prices of these 3 stocks - p1, p2, p3

3. Uninvested cash amount c1

Full State Vector s = [n1, n2, n3, p1, p2, p3, c1]

• Action:

Actions is a set of actions on all three stocks. The available actions of each stock are buying, selling or holding which result in decreasing, increasing, and no change of the portfolio value respectively. In our case the number of stocks is three, so the number of possible actions is 3^3 = 27. For implementation of this project we simplify the actions – All the shares are sold when a sell decision is made. During a buy, we loop through every stock and buy one share in round robin fashion until the money is over. The shares are sold before buying.

•Reward:

The change of the portfolio value when action is taken at states and arriving at the new states.

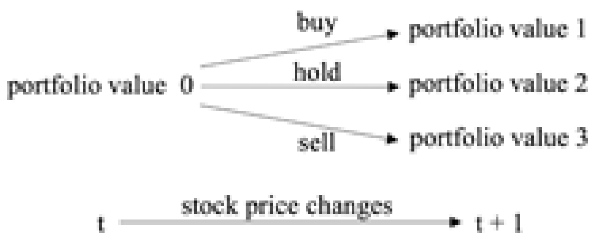
s = vector of #shares owned

p = vector of share prices

c = cash

Portfolio Value = s. p + c

With the states, actions and reward defined above, the goal is to maximize the total reward over time.



We are modelling the action value Q(s,a) using linear regression. However, the idea is to use the reinforcement learning with neural networks. Here the vector will only use the state as input and we'll have a separate output for each action. In our case the size of the state is 7. The number of actions is three to the power three which is twenty-seven. These represent the different permutations of buy sell and hold that we can perform for each stock.

Thus, our weight matrix will be of size seven by twenty-seven and our bias vector will be of size twenty-seven.

## Testing and Evaluation:

Testing and Evaluation is done as follows:

1) Provide the agent with day's stock prices from the test data and a cash amount, then get a predicted optimal action according to the learned policy.

2) Take the predicted action, then take the transition to the portfolio state on the next day. Revise the stock prices according to the test data to the next day's stock prices.

3) Repeat 1 and 2 over many days and track the value of the portfolio over time. This value should grow over time.

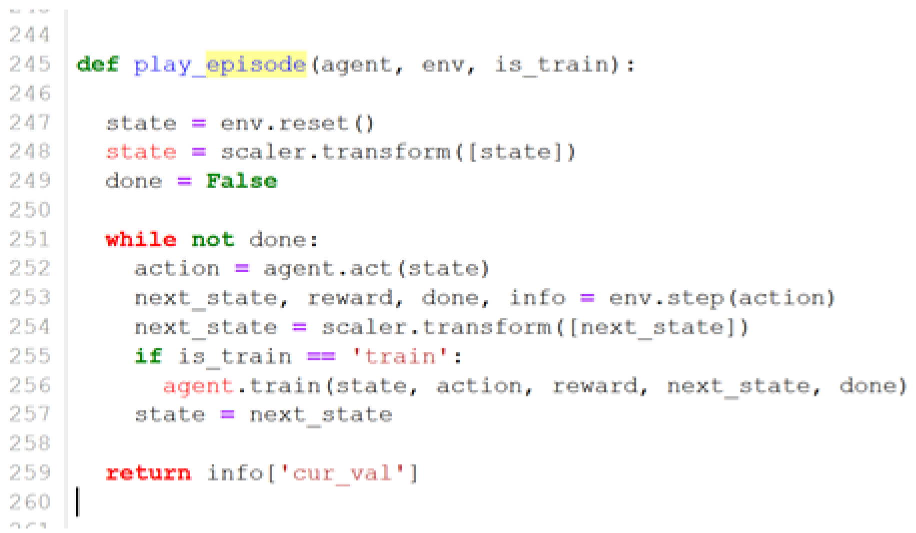
## Program Design

**B**elow is the high-level design of our RL program.

```
env = Env ()
agent = Agent ()
portfolio_values = []

for in range (number of episodes)
        value = play_episode (agent, env)
        portfolio_values. append (value )
```

The instance of the environment is first created and next an instance of the agent. Then in a loop a function called play_episode is executed which accepts the environment and the agent and returns the value of the portfolio at the end of the episode. When our loop is done, we're going to save the portfolio values for later so that we can plot them in the end and analyze them.



```python
def play_episode(agent, env, is_train):

  state = env.reset()
  state = scaler.transform([state])
  done = False

  while not done:
    action = agent.act(state)
    next_state, reward, done, info = env.step(action)
    next_state = scaler.transform([next_state])
    if is_train == 'train':
      agent.train(state, action, reward, next_state, done)
    state = next_state

  return info['cur_val']
```

The environment reset is done to get back to the initial stage. Then we initialize our done flag to false and enter a loop that only quits when done becomes true. Next an action is chosen which we receive from the Agent. The Step function perform the action and gets back the next day reward. This will be some variant of gradient descent as usual finally we'll set the current state to be the next day for the next iteration of this loop. When we're done, we'll return the value of our portfolio.

Data Normalization: Our state which is composed of three parts can have vastly different ranges. The first part consists of the number of shares owned, the second part consists of share prices and in the third part we have the uninvested cash. So, we want to normalize this data whenever we get to the next state. We will have a scalar object from scikit-learn which will take our state and standardize it to have zero mean and unit variance.

Environment: The environment objects will accept a time series of stock prices as input into its constructor will also have a pointer to tell us what day it is. We know the current stock prices while know how much cash we initially start with our initial investment.

The action will be taken to buy and sell the stocks as specified by the Agent. Next the pointer will be set to next day's process. We will also calculate the next day and the portfolio value. From this we can calculate the reward. If we reach the end of our time series so that's basically it for the environment.

Agent: The Agent class which has an action function accepts input a state and decides what actions you perform in the environment. Here it's going to use the Q learning or variant of it like Epsilon greedy. A gradient descent function which does the following.

First, it's going to take in a tuple of data – state, action, reward, next state and done flag. We can use this to calculate a supervised learning dataset which consists of input and target pairs. The input into our model is the state, the target will be the reward. If the next state is terminal the 'done' flag is set to true.

If the next state is not terminal, then we use the usual Q learning target reward. Once we have our data set, we can do one iteration of gradient descent as usual. Also, we will be incorporating momentum into our model as well which helps the model train faster.

## Training Process:

Training happens through a series of episodes. Each episode cycles through stock data from a specific contiguous time period, and the following steps happen in an episode:

1) Observe the current state and gets the Q-Values for all actions in the corresponding state-action pairs

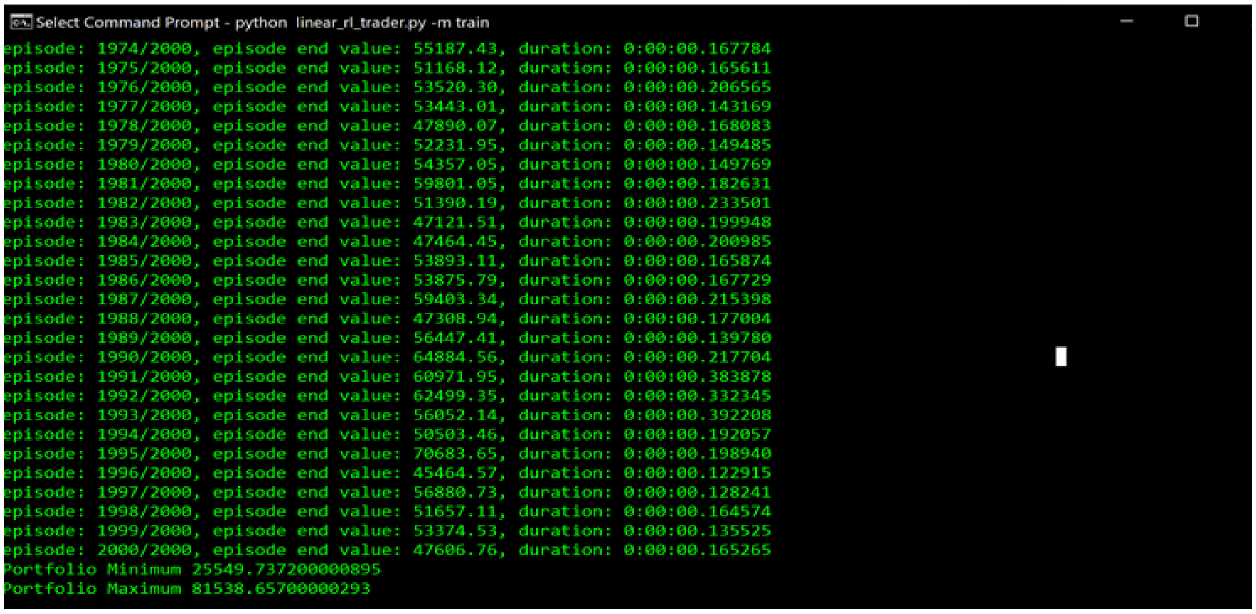
2) Pick the action with the highest Q-value using classic-greedy approach to exploration.

3) Take the action chosen in Step 2. This results in a transition to state t+1, with an immediate reward

4) Update(s,a) using the newly obtained reward and the transitioned state.

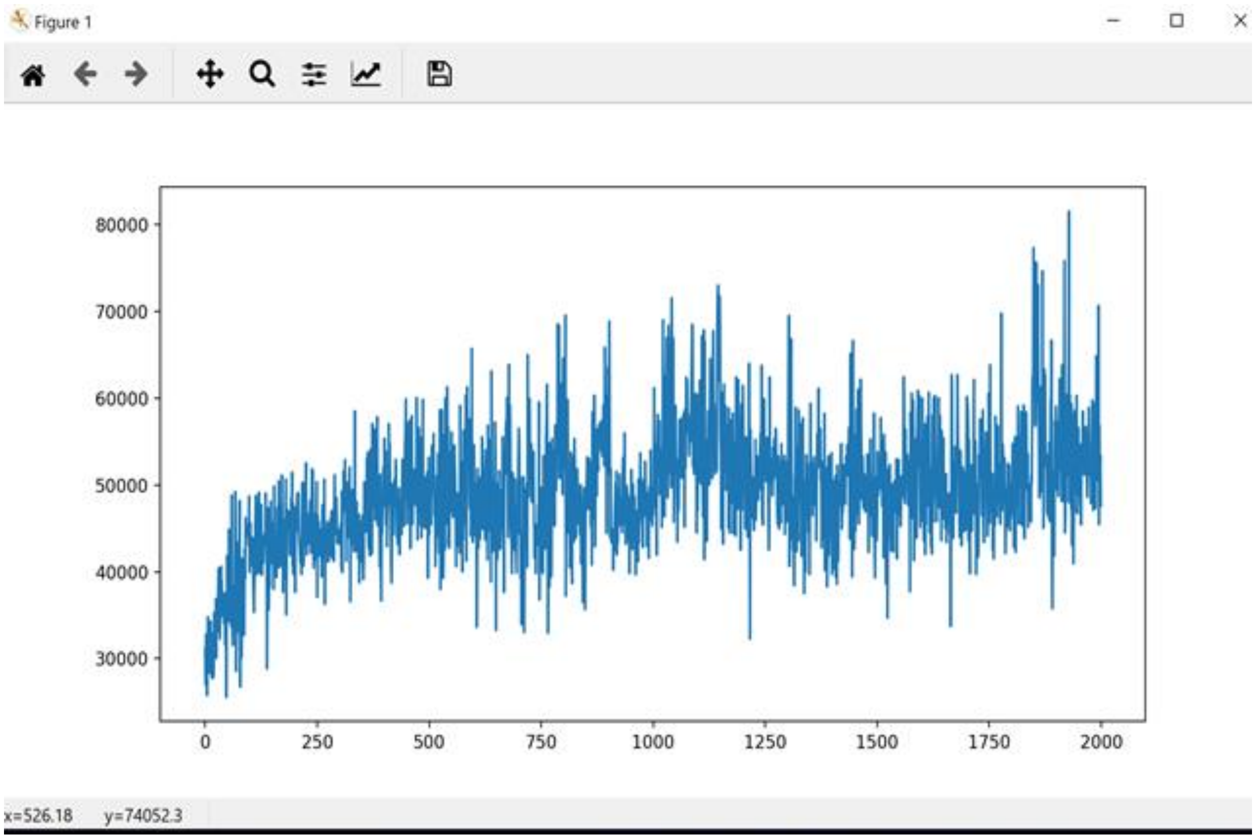
5) Repeat steps 2, 3 and 4 until the end of the episode. After the episode ends, we use the Q-Values for all actions in all the states observed so far to train neural network. The update equation reduces to $Q_{t+1}(s,a) = r+\gamma \max Q(s_{t+1}, a)$

In our case we used 2000 episodes for training for our model.


Select Command Prompt - python  linear_rl_trader.py -m train

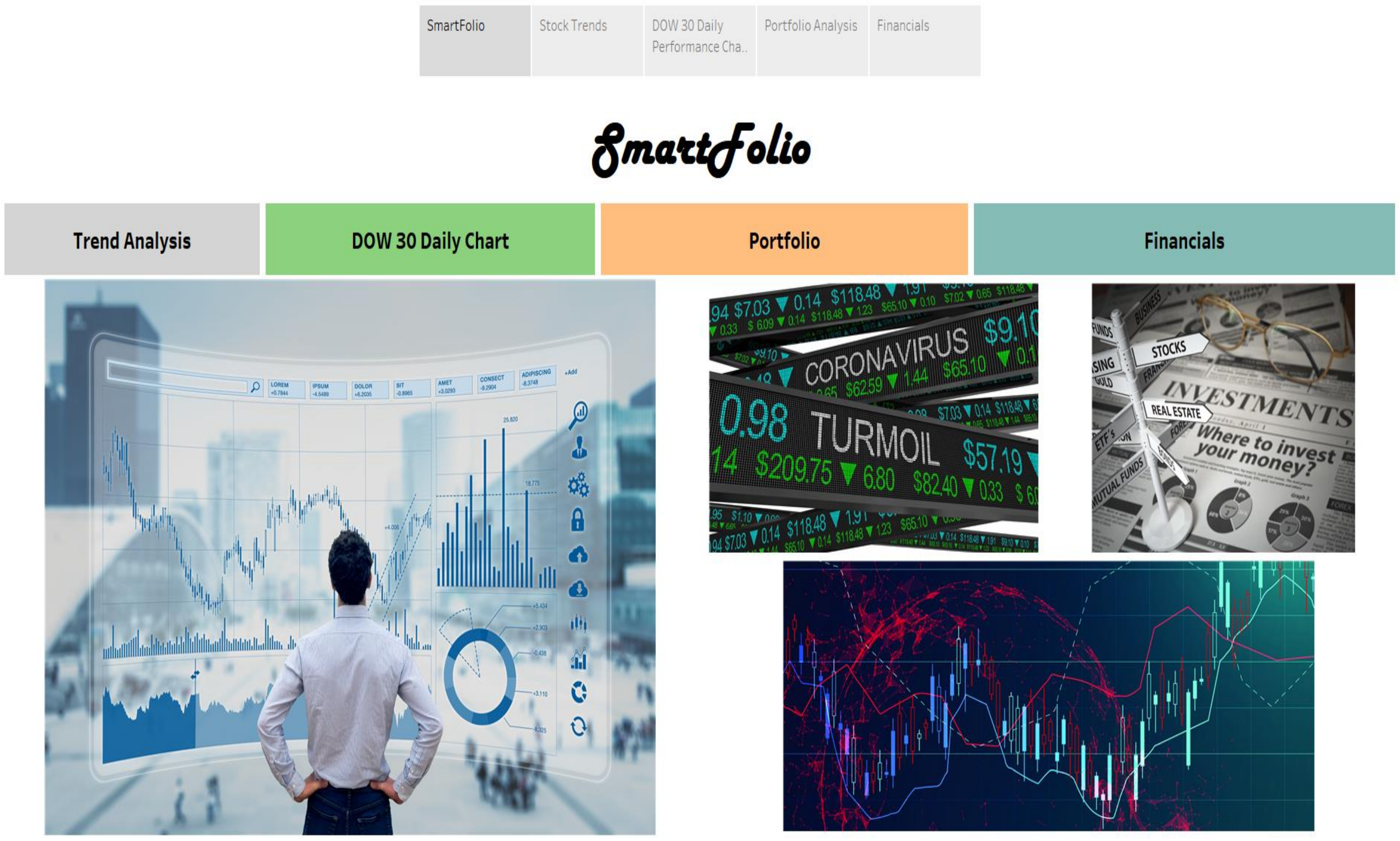
```
episode: 1974/2000, episode end value: 55187.43, duration: 0:00:00.167784
episode: 1975/2000, episode end value: 51168.12, duration: 0:00:00.165611
episode: 1976/2000, episode end value: 53520.30, duration: 0:00:00.206565
episode: 1977/2000, episode end value: 53443.01, duration: 0:00:00.143169
episode: 1978/2000, episode end value: 47890.07, duration: 0:00:00.168083
episode: 1979/2000, episode end value: 52231.95, duration: 0:00:00.149485
episode: 1980/2000, episode end value: 54357.05, duration: 0:00:00.149769
episode: 1981/2000, episode end value: 59801.05, duration: 0:00:00.182631
episode: 1982/2000, episode end value: 51390.19, duration: 0:00:00.233501
episode: 1983/2000, episode end value: 47121.51, duration: 0:00:00.199948
episode: 1984/2000, episode end value: 47464.45, duration: 0:00:00.200985
episode: 1985/2000, episode end value: 53893.11, duration: 0:00:00.165874
episode: 1986/2000, episode end value: 53875.79, duration: 0:00:00.167729
episode: 1987/2000, episode end value: 59403.34, duration: 0:00:00.215398
episode: 1988/2000, episode end value: 47308.94, duration: 0:00:00.177004
episode: 1989/2000, episode end value: 56447.41, duration: 0:00:00.139780
episode: 1990/2000, episode end value: 64884.56, duration: 0:00:00.217704
episode: 1991/2000, episode end value: 60971.95, duration: 0:00:00.383878
episode: 1992/2000, episode end value: 62499.35, duration: 0:00:00.332345
episode: 1993/2000, episode end value: 56052.14, duration: 0:00:00.392208
episode: 1994/2000, episode end value: 50503.46, duration: 0:00:00.192057
episode: 1995/2000, episode end value: 70683.65, duration: 0:00:00.198940
episode: 1996/2000, episode end value: 45464.57, duration: 0:00:00.122915
episode: 1997/2000, episode end value: 56880.73, duration: 0:00:00.128241
episode: 1998/2000, episode end value: 51657.11, duration: 0:00:00.164574
episode: 1999/2000, episode end value: 53374.53, duration: 0:00:00.135525
episode: 2000/2000, episode end value: 47606.76, duration: 0:00:00.165265
Portfolio Minimum 25549.737200000895
Portfolio Maximum 81538.65700000293
```

Following figure shows the distribution of our portfolio value over the number of episodes. The policy learned by the agent is generally low-risk and medium returns. We made 40 trials on data not previously seen by the model.



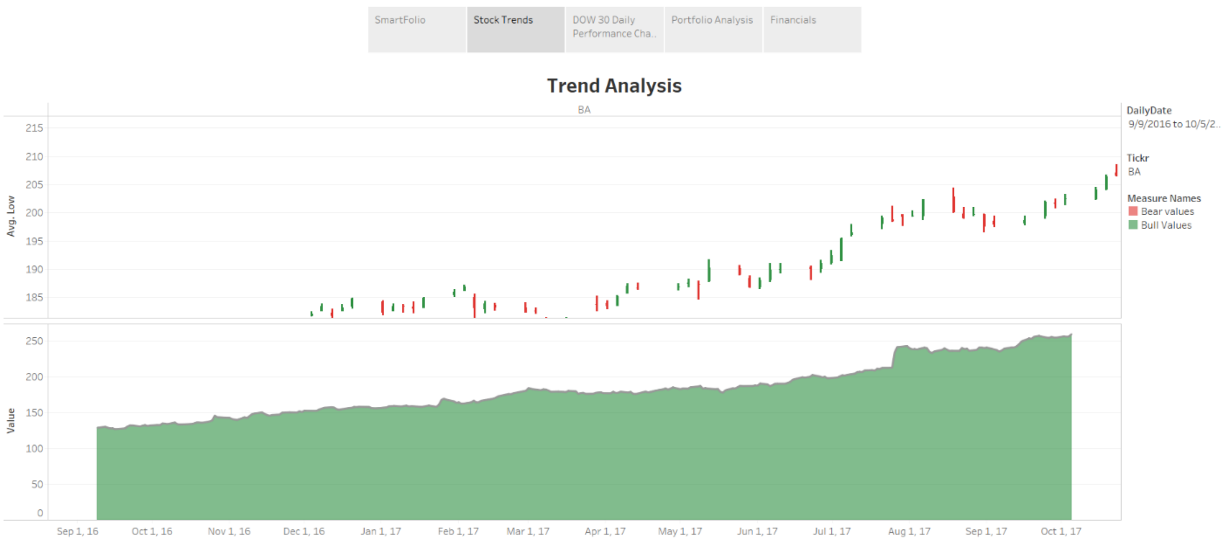
# SmartFolio Tool: Tableau Visualizations

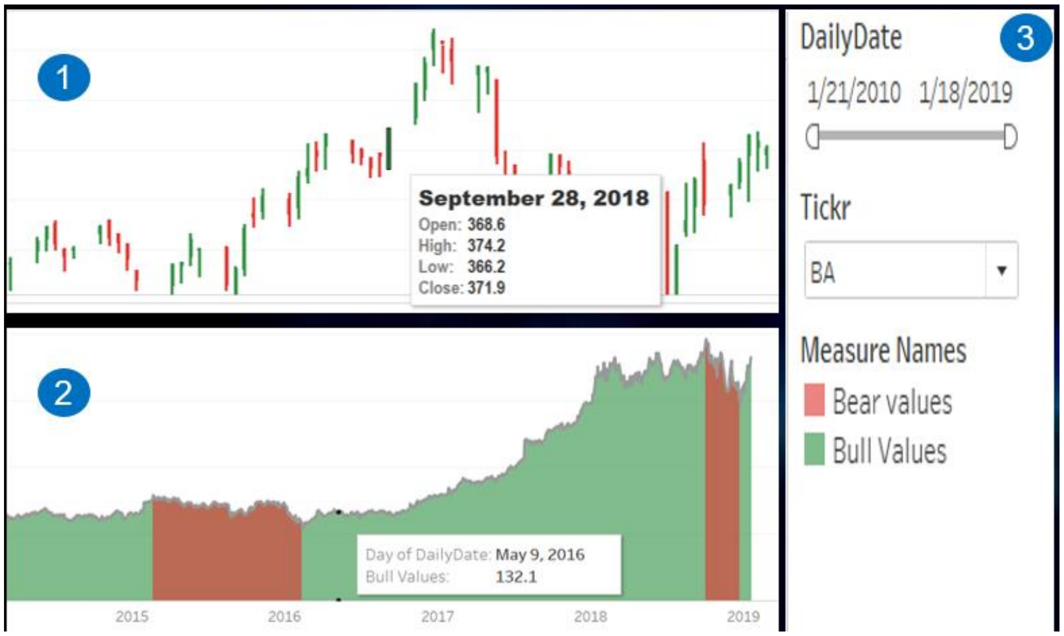
## Tableau Home page



This is the first page the user will see when using SmartFolio. The Home page essentially has four buttons to easily navigate on-click to other pages which will display the visualizations. The user can navigate to any of the following four screens: Trend analysis, Dow 30 Daily Chart, Portfolio and Financials.

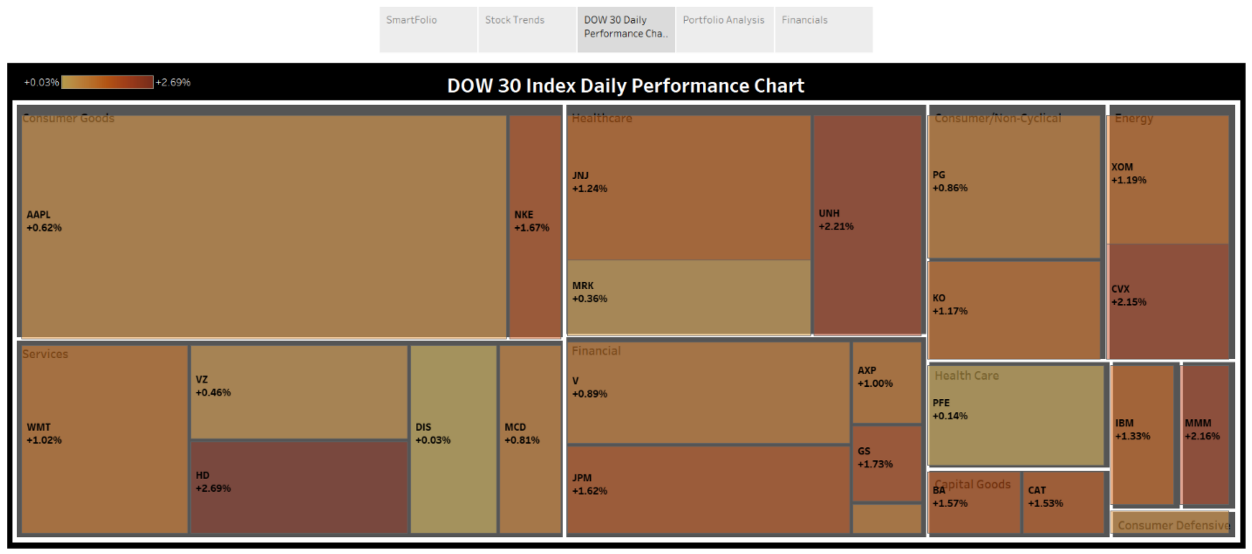
## Trend analysis page

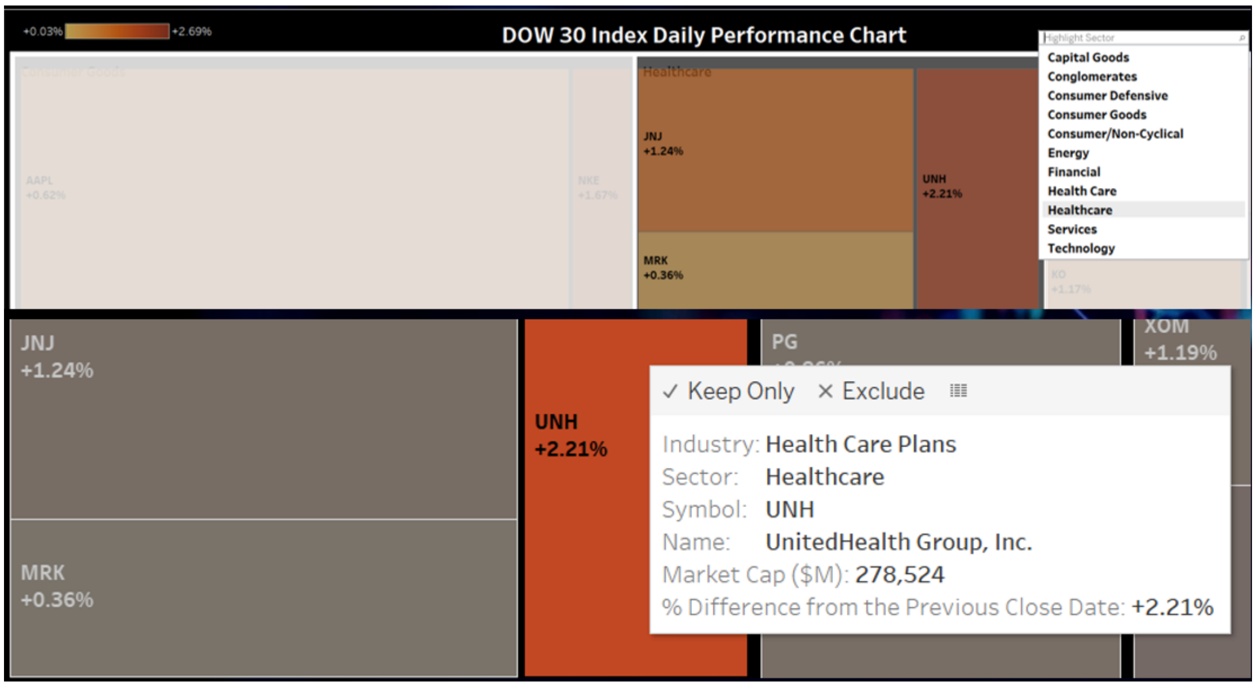




The first page shows a stock's general trend over the years. The first chart is a candlestick chart which shows the price movement over a time range and the bottom graph shows if the market was bullish or bearish. In the candlestick chart, every line is a candle, which on hover displays data, open, close, high, low prices. Through the height of the candle one can estimate if there is strong buyer control, taller the height means more buyer control. It is better to not consider candles in isolation or in a range when making buy/sell decisions. The next chart shows the trend when the close price falls below 20%. The red range shows when the market was bearish that means when there was a decline and the green range shows when the market was bearish or when the market was rising. The other component is the filter's where the user can select the time range and the company to display data.

## Dow 30 Daily Performance Chart

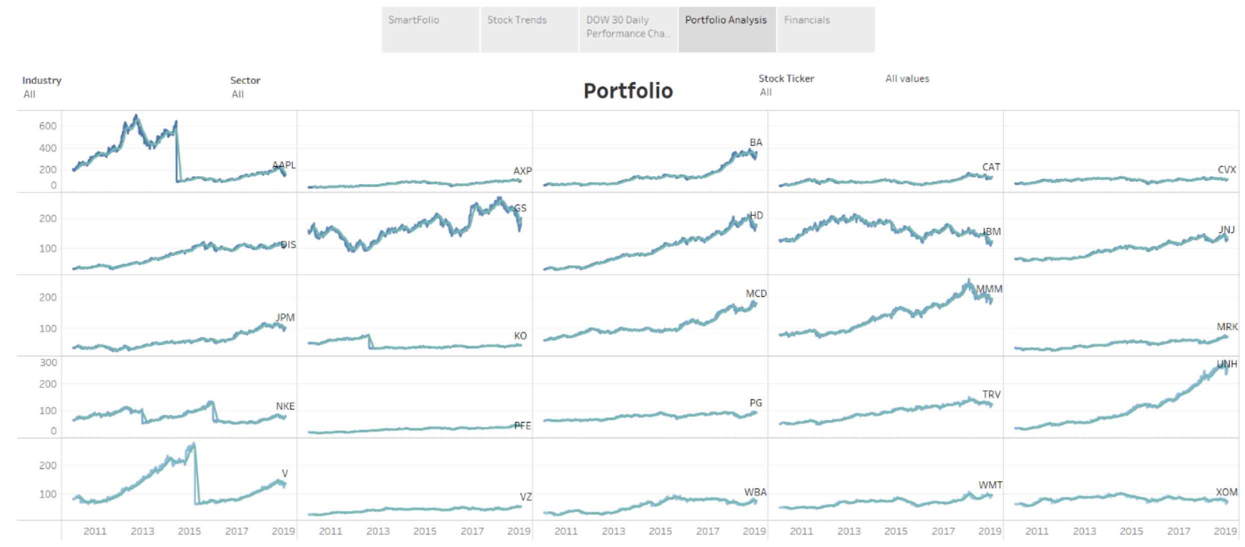


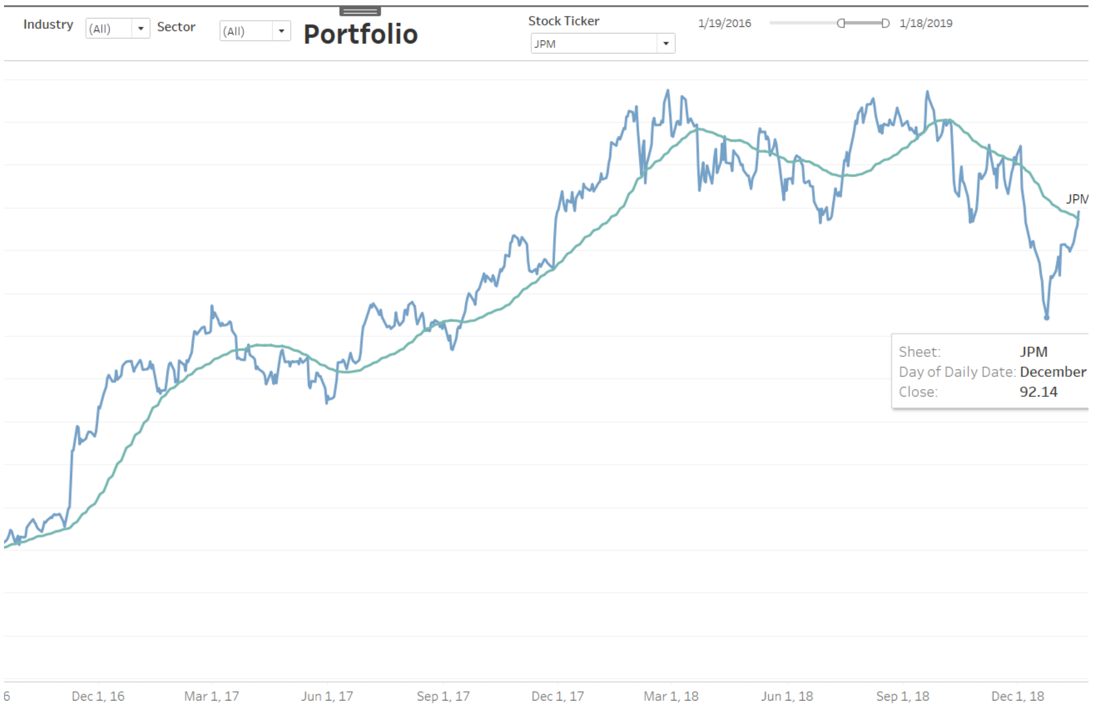


Industry: Health Care Plans
Sector: Healthcare
Symbol: UNH
Name: UnitedHealth Group, Inc.
Market Cap ($M): 278,524
% Difference from the Previous Close Date: +2.21%

The second screen or page shows the daily performance of all the 30 companies in the Dow Jones Index. The heatmap divides the companies based on sector and to highlight all companies in the sector, the user can select the sector from the drop down.  The daily performance is tracked based on market capitalization and the percentage difference from previous day's close price. This information can be seen on hovering on a company. The size of the block depends on the market capitalization, meaning bigger the market cap bigger the size of the block. The color range shows if the company is performing better or worse. Brown color indicates the company's stock is profitable or performing well while yellow color means it's not performing well.

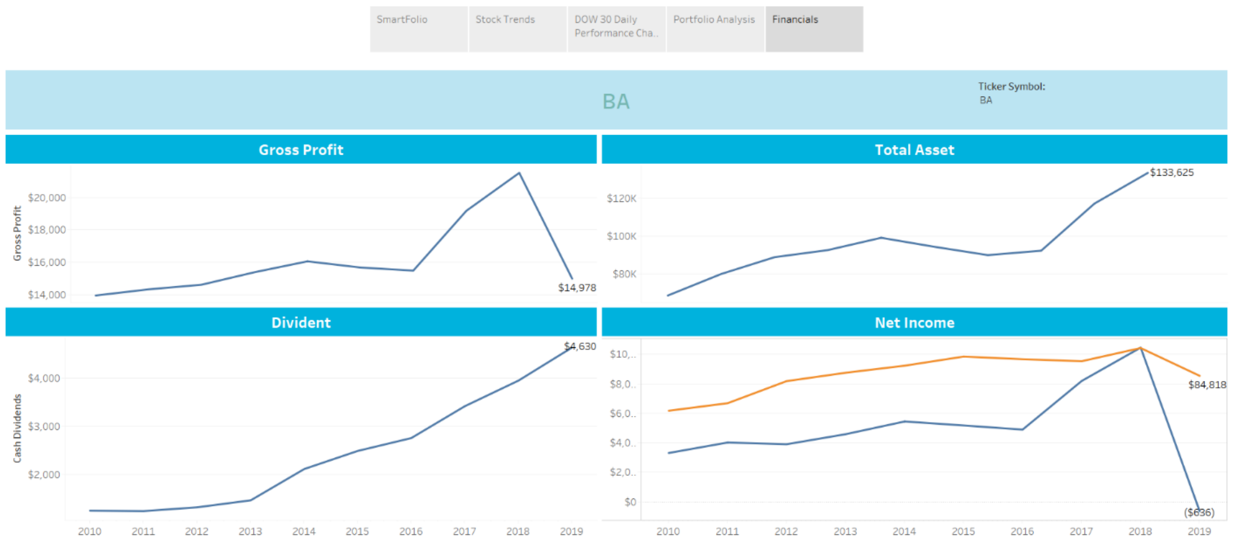
## Portfolio Analysis Page





The third page gives a portfolio analysis of all companies in the Dow 30 Index. Depending on the industry and sector selected the companies displayed will keep on changing. On selecting a ticker, the graph of only that company will be displayed. SMA 50(50 Day Moving Average) which is a stock price average over the last 50 days which often acts as a support for trading. The graph tracks SMA 50 and the close price of a company over the years, which can be altered through the date range filter provided. The blue line indicates the close price and the teal line represents SMA 50 for a date which is made visible on hover. A close price dropping below the SMA 50 line could inferred as the trader should sell the stock in the hope that the close price will rise again to meet the SMA 50.

## Financials Page



The final screen in SmartFolio tracks each company's financial data which was taken from Wharton. There is dropdown filter to select the company name. The screen is divided into four parts namely, gross profit earned, total assets owned, dividends given out and the net income earned over a period of ten years from 2009 to 2019.

Gross profit is the profit a company makes after subtracting the costs of making and selling its products or cost of providing services. Higher the gross profit, better is the process efficiency of the company. Lower the gross profit, poorer is better is the process efficiency of the company.

Total asset means the amount of assets a company owns. This can help the trader understand the valuation of the firm. Dividend is a small sum of money a shareholder receives when the company profits. When the market is thriving, it is most likely that a company will increase its' dividend. and uncertain market or during times of recession a company may decrease or not give out any dividends at all.

In net income chart, the yellow line tracks the sales net and the blue tracks the net income which is dependent on the sales net. Net income is the money earned by a company after removing taxes and other such deductions. Net income basically indicates profit. Net sales is the money earned through interacting with customers alone. Net income is dependent on net sales.

A dip in net sales affects net income hugely as can be seen in the image and which may also be a result of Boeing's 737 Max crashes in 2018 and 2019. Such dips and highs in the company financials graph can usually be associated with a certain incident that occurred with the company.

## Challenges with Sentiment Analysis

After obtaining the data from NewsAPI we realized there were certain limitations and problems.

First of the free version of NewsAPI only provides data for the last thirty days, which was not enough. To solve this problem and obtain lot of data we decided to call the API by using search queries and passing the name of each of the 30 companies in the Dow Jones Index. This created another problem which is that lot of the data was duplicated and irrelevant. To overcome this challenge, we decided to try other News API's like Finnhub.io and StockNews API but of them had the same issues. Since we would use the free version, the free version of both had limitations. StockeNews API returned only 50 items in a response for a ticker which were repeated and Finnhub.io would block out after making a few calls to the API.

For StockTwits API, an authentication key was needed, which we had applied for, but which was received very late. Additionally, we had the realization NewsAPI would always give us valid news because it fetched it from well-known news journals. But since StockTwits and Reddit are both public forums with anonymous user's posting their thoughts and opinions, laced with trolls who would misguide people with misinformation, these APIs could not be trusted as sources of valid information. To weed out the valid information, analysis would need to be performed to identify which post is fake or not.

Additionally, the news data was only to be used in visualization's and not in any of the models we had built. Hence, due to all these challenges, we decided to focus more on building accurate models like RL and drop sentiment analysis and include it as a future development which can also be used in RL where sentiment score can be passed as a feature.

## Challenges with Connecting Tableau to S3

All the visualizations performed for Tableau were by fetching data saved on the local disk. This was because there were port issues to connect Tableau to Amazon S3. To connect Tableau to S3, two other AWS services called Glue and Athena would be needed as well. AWS Glue essentially crawls all data in S3 using a crawler and automatically distinguishes the type of file, the type of data etc. basically auto-creating a schema. Amazon Athena then can see these crawled databases and can be used to query them. Though we were successfully able to crawl the S3 data through AWS Glue and query it through AWS Athena and view the table names in Tableau, the data would not be visible due to port 444 error which is needed by Amazon Athena.

## Link to the code

**https://www.dropbox.com/sh/fookb4zmzh2hnnk/AAD0NO2KTJeGhaHbYCxxA1DCa?dl=0**

## References:

[1] https://aws.amazon.com/s3/faqs/

[2] https://www.investopedia.com/

[3] https://www.alphavantage.co/documentation/#intraday

[4] https://newsapi.org/

[5] https://stocktwits.com/

[6] https://www.tableau.com/about/blog/2017/5/connect-your-s3-data-amazon-athena-connector-tableau-103-71105

[7 ] https://towardsdatascience.com/ridge-regression-for-better-usage-2f19b3a202db

[8] https://scikit-learn.org/stable/auto_examples/ensemble/plot_gradient_boosting_quantile.html

[9] Xiong et al., Practical Deep Reinforcement Learning Approach for Stock Trading https://arxiv.org/pdf/1811.07522.pdf

[10] https://medium.com/@randerson112358/stock-price-prediction-using-python-machine-learning-e82a039ac2bb