MANUSCRIPT ID: 000000

Predicting QualComm Stock Price using Machine Learning Models

John W. Smutny, Ben C. Johnson, Anagha Mudki, James A. Ensminger

***Abstract* — In this paper, four different Machine Learning (ML) models are trained and tested to see which model could most accurately predict the closing stock price of the QualComm corporation twenty-eight days in the future. The four models tested were 1) Multivariate Linear Regression, 2) Logistic Regression, 3) Artificial Neural Network, and 4) a Random Forest (ensemble) model. The best model is determined by which model had the lowest average Mean-Square Error (MSE) value after training the model 100 unique times.**

The dataset used to train the model's predicted future stock price is a combination of the daily QualComm corporation stock behavior, the company's interpolated quarterly earnings, general long-term and short term US economic indicators, global trade indicators, and the values of these features from previous days. All of the four models described were trained using this identical dataset, but with different random seeds and different data entries taken as the Train and Test sets. After evaluating all four models, the Linear Regression model had the lowest average MSE at 89.

It is important to note that each model had a relatively acceptable accuracy prediction on the QualComm Stock price. Each model tackles the dataset differently. Most notably, logistic regression predicted a boolean value of whether or not to invest, rather than predicting the actual stock price. The ANN neural network model had a very large range of MSE which should be considered when making a decision on the best model

***Index Terms*— Linear Regression, Logistic Regression, Artificial Neural Network, Random Forest, Machine Learning.**

I. Introduction

Predicting the stock market is extremely risky as unforeseen events can create deep regret and extreme success. No one can predict real-life, but by understanding a company's fundamentals and the factors affecting its industry. This report attempts to explore how best to use known factors to make the stock market a little less risky. Specifically, when investing in the QualComm semiconductor corporation.

A. QualComm Corporation

The QualComm international corporation is an American company that specializes in the semiconductor and telecommunications industry. QualComm's self-described primary market is in the sale of integrated circuit chips for phone and wireless devices, giving access to wireless communications like fifth generation spectrum (5G), bluetooth and long-term evolution (LTE). They directly sell to telecom companies like T-Mobile and Verizon. The company uses 'fabless manufacturing', meaning that the company imports all of the semiconductors that it produces from overseas factories and thus are impacted by international law and trade relations [1]. In their industry, QualComm is a direct competitor with semiconductor producers Ericsson, NXP Semiconductors and Samsung [2].

II. Data Discovery

A. Methodology

Qualcomm stock price modeling consists of the careful collection and preparation of input feature data followed by an iterative model analysis and selection process. Per the specification, base datasets include the five core daily stock price metrics. To augment these values the researchers gathered available datasets of industry verticals, company performance trends, and global financial trends. Following data collection, data analysis is performed for compliance with the intended supervised machine learning models. This process is performed first by using the standard data quality report, and then by examining preliminary model performance achieved with different candidate data sets.

Given the error introduced into the model's highly correlated features; the previous step's goal is to create a small yet impactful feature set. Following final feature selection, the four candidate models are run repeatedly to better understand the target feature (price 28 days in the future) over time. As shown in appendix section *'B. Other Model 5 Year Stock Predictions Against Actual Close Price'*; the model performance provides feedback used to refine input data and select the final model. Using the final model, a simulation of daily investment decisions is run to exemplify the expected financial gains of an investor. Finally, results are analyzed for further understanding of the model architecture as well as the statistical nature of stock price prediction.

B. Datasets Used

Based on QualComm's company background and business competitors; several external datasets were added to the general daily stock trading information to improve the ML models' ability to predict the company's stock price.The datasets described below were chosen based on their perceived correlation with the company's future stock value. However, it is very likely that each model placed varying importance on the included features. Table DD1 on the following page highlights the exact list of datasets used. The list below describes the general categories of information added for analysis.

1. QualComm Stock Information
2. QualComm Quarterly Business Earnings
3. Competitor/Customer Stock Information
4. Economic Indicators for the United States of America
5. Economic Indicators affecting Global Trade

1. QualComm Stock Information

The basic statistics that are provided for every day on the NASDAQ stock exchange for every company. The daily QualComm stock 'Close Price' and 'Volume of Shares Traded statistics are valuable to understanding an outside investor's value of the company and where they see it headed. 'Close Price' is straight forward. 'Volume' indicates how people think the company will be in the future. More volume means more stock price volatility.

2. QualComm Quarterly Business Earnings

Quarterly earnings statistics represent the actual fundamentals of a business and (in theory) have the biggest impact on stock price. Items such as the amount of profit, debt, and revenue are included here. Since this is quarterly data, the researchers used linear interpolation to extend the data points from quarterly data to daily data..

3. Competitor/Customer Stock Information

A list of close competitors working against QualComm and companies buying QualComm's products. Only the close price of these company's stocks are included. The companies added include direct semiconductor competitors like Ericsson and Samsung, as well as customers like Verizon and Intel.

4. Economic Indicators for the United States of America

No company ever lives in a vacuum and thus are affected by every consumer that interacts with the company. The various United States of America bond rates and reported Consumer Price Index (CPI) by the United States of America government are used to gauge the overall sentiment and health of the country's economy. The rationale of these features is that as bond rates increase and CPI (also correlated with inflation rates) decrease, then it's a sign that the economy is thriving and consumers will spend more money. Thus increasing the stock price of QualComm.

5. Economic Indicators affecting Global Trade

Since QualComm is an international corporation that imports a majority of the supplies; global indicators like the price of Brent crude oil and the nominal broad US dollar index help understand the price of international business. As world (Brent) oil prices increase, importing/exporting from the United States becomes more expensive and influences a company's quarterly sales number (and future outlook). The Nominal Broad US Dollar Index is a US Treasury Department index that tracks how the value of the US Dollar compares to currencies in other countries. A high index indicates that the US Dollar is stronger than other currencies and thus importing goods to the US becomes cheaper. For QualComm, this means the cost of importing manufactured supplies would decrease.

TABLE DD1
LIST OF DATASETS AND FEATURES USED IN THE TRAINING OF ALL ML MODELS

Dataset	**Features Used**	**URL**	**Date Accessed**
QualComm Historical Data [3]	Date, Close/Last, Volume, Open, High, Low	https://www.nasdaq.com/market-activity/stocks/qcom/historical	4/29/2022
QualComm Company Quarterly Report [4]	Gross Profit, Net Income Available to Common Shareholders, Total Assets, Total Liabilities, Net Income/Starting Line, Dividends, Net Changes in Case	https://simfin.com/data/companies/85758	4/20/2022
CPI (Consumer Price Index) for all Urban Consumers [5]	InflationRate for all items less food and energy in U.S. city average, all urban consumers, not seasonally adjusted	https://data.bls.gov/timeseries/CUUR0000SA0L1E?output_view=pct_12mths	4/11/2022
Market Yield 3-Month US Treasury [6]	Daily bond yield	https://fred.stlouisfed.org/series/DGS3MO	4/29/2022
Market Yield 2-Year US Treasury [7]	Daily bond yield	https://fred.stlouisfed.org/series/DGS2	4/29/2022
Market Yield 10-Year US Treasury [8]	Daily bond yield	https://fred.stlouisfed.org/series/DGS10	4/29/2022
Nominal Broad US Dollar Index [9]	Daily value	https://fred.stlouisfed.org/series/DTWEXBGS	4/29/2022
Global Price of Brent Crude Oil [10]	Cost per barrel	https://fred.stlouisfed.org/series/POILBREUSDM	4/29/2022
Coinbase Bitcoin Price [11]	Price per Bitcoin	https://fred.stlouisfed.org/series/CBBTCUSD	4/29/2022
AAStock Price [12]	Daily close value	https://www.nasdaq.com/market-activity/stocks/aapl/historical	4/29/2022
Google Stock Price [13]	Daily close value	https://www.nasdaq.com/market-activity/stocks/googl/historical	4/29/2022
Ericsson Stock Price [14]	Daily close value	https://www.nasdaq.com/market-activity/stocks/erixf/historical	4/29/2022
NXP Semiconductors Stock Price [15]	Daily close value	https://www.nasdaq.com/market-activity/stocks/nxpi/historical	4/29/2022
Samsung Stock Price [16]	Daily close value	https://www.nasdaq.com/market-activity/stocks/ssnlf/historical	4/29/2022
Verizon Stock Price [17]	Daily close value	https://www.nasdaq.com/market-activity/stocks/vz/historical	4/29/2022
T-Mobile Stock Price [18]	Daily close value	https://www.nasdaq.com/market-activity/stocks/TMUS/historical	4/29/2022

C. Cleaning the Datasets (Data Quality Report)

Prior to Data Quality Report (DQR) generation, each input dataset required basic formatting and data cleaning. For consistency across data sets the ‘Date’ columns were converted to Pandas DateTime objects and sorted in ascending order. After date conversion, the majority of stock data was correct, except for the simple removal of the ‘$’ prefix for prices. For the remaining datasets, a few columns also required conversion from comma formatted prices to standard floating point numbers.

Please see section *‘A. Data Quality Report of the Final Dataset’* in the appendix for the Data Quality Report representing the modified datasets. Table DD2 below shows the initial DQR for Qualcomm pricing data.

TABLE DD2

DATA QUALITY REPORT OF THE QUALCOMM BASIC STOCK INFORMATION BEFORE PROCESSING.

stat	Date	Close/Last	Volume	Open	High	Low
count	1260	1260	1260	1260	1260	1260
cardinality	1260	1149	1259	1156	1180	1180
mean	*	94.34067	11185564	94.35892	95.64676	93.06873
median	*	77.91	9396205	77.595	78.9475	76.685
number at median	*	0	0	0	0	0
mode	5/1/2017	52.49	11301880	52	58.49	52.27
number at mode	1	4	2	3	4	3
stddev	*	39.76462	8104528	39.82371	40.48734	39.08962
min	*	49.4	2120165	49.52	49.8	48.56
number at min	*	1	1	1	1	1
max	*	189.28	1.56E+08	190.304	193.58	185.1852
number at max	*	1	1	1	1	1
number of zeros	*	0	0	0	0	0
number missing	*	0	0	0	0	0

All measurements annotated with ‘*’ represents an invalid mathematical operation with a non-numeric feature

III. Data Preparation

Before any of the models can be trained, several steps to clean and prepare the dataset can be done to improve the model's prediction accuracy. The initial Data Quality Report above helps guide the following steps in data preparation.

Daily stock data aligns perfectly across all included companies, but economic performance measures, such as CPI and financial reportings, have varying sample frequencies and dates. To create contiguous input feature data, lower frequency data is interpolated for all days the stock market was open within the applicable time window. Interpolation using sample-and-hold, zero-filling, and linear estimation offer various methods to fill in missing data points. To minimize the effect of generated data, linear approximation between data points is used.

Data from the Federal Reserve of St. Louis' Economic Data website contains several data points with '.' as a placeholder for missing values. Per analysis of the source, and the date interpolation approach, the previous day's value is used as a replacement. In total, the 3-month, 2-year, and 10-year Bond Maturity datasets each require 53 modifications while the Nominal Broad US Dollar Index requires 59. Bitcoin price only requires one such update. Following this step all missing or invalid values have been resolved.

With data prepared for model ingestion, the addition of a target feature and historical data are the final steps. The target feature, as mandated by the specification, requires the use of the stock price 28 days in the future of each day. To enable a valid future price for each data point, the last 28 days of the data are dropped as due to the lack of future values. Similarly, for some features such as the close price of the QualComm stock, the previous 30 days of data were used to give historical context to each data point. As a result, the 30 earliest data points are removed to account for the addition of historical data going thirty days in the past. Of note, due to the presence of multiple price values each day, the closing price is used as the future target variable.

Any additional modifications to data, such as normalization, were completed in the modeling stage due to variations in requirements for each model.

IV. Tested Models

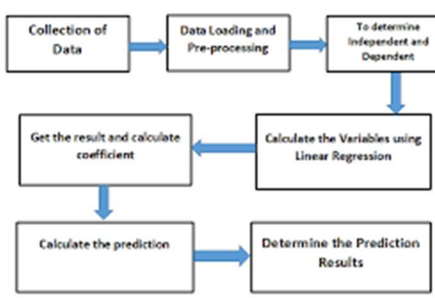
Four different ML models were compared to see which could predict the future QualComm company stock price the best; Multivariate Regression, Logistic Regression, Artificial Neural Network, and a Random Forest (ensemble) model. Each of these models were given the same dataset described in the previous section and trained to predict the closing stock price of the QualComm corporation twenty-eight days in the future. The best model is determined by which model has the lowest MSE.

A. Multivariate Regression

1. Summary of Approach

Linear regression is a linear approach for modeling the relationship between a scalar response and one or more explanatory variables (also known as dependent and independent variables). The case of one explanatory variable is called *simple linear regression*; for more than one, the process is called multiple linear regression. In multivariate linear regression multiple correlated dependent variables are predicted, rather than a single scalar variable. Linear regression is mostly used for prediction and forecasting purposes (such as stock price).

Figure TM1
Schematic of a linear regression classifier [20]



2. Model Results

For the linear regression model, Table TM2 outlines the exact architecture used to train each model iteration.

TABLE TM2
Architecture of Linear regression model

Parameters	Values used (default values)
fit_intercept	True
copy_X	True
n_jobs	None
positive	False

After training, two measures were used to determine the error of the regression model: mean squared error (MSE) and R-squared error. The MSE shows how close a regression line is to a set of points. It does this by taking the distances from the points to the regression line (these distances are the “errors”) and squaring them. The squaring is necessary to remove any negative signs. It gives more weight to larger differences.

R-squared (R2) is a statistical measure that represents the proportion of the variance for a dependent variable that's explained by an independent variable or variables in a regression model.

The average of mean squared error for the model after 100 iterations was approximately 89 (from table TM3) which was lowest among all the models under consideration. Also the average value of R- squared error (table TM4) after 100 iterations was 0.94 which was best among the three models (For logistic MSE and R- squared was not calculated).

TABLE TM3
Linear Regression MSE Performance Metrics

MSE Mean	88.99348
MSE Max	157.91675
MSE Min	59.93440

TABLE TM4
Linear Regression R^2 Performance Metrics

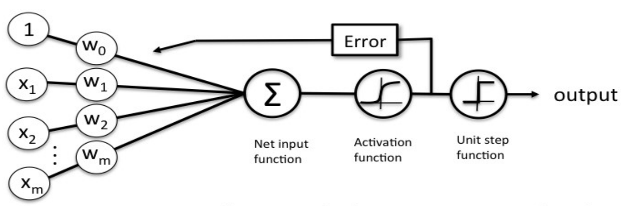
R^2 Mean	0.94295
R^2 Max	0.95995
R^2 Min	0.89916

B. Logistic Regression

1. Summary of Approach

Logistic regression is a process of modeling the probability of a discrete outcome given an input variable. The most common logistic regression models a binary outcome; something that can take two values such as true/false, yes/no etc. Logistic regression is used to predict the categorical dependent variable using a given set of independent variables. The full model architecture is shown below in Table TM8. The model was trained using a 0.0001 tolerance, giving no preference to weights, and using the lbfgs solver.

Figure TM5
SCHEMATIC OF A LOGISTIC REGRESSION CLASSIFIER [21]



Schematic of a logistic regression classifier.

2. Model Results

To determine the performance of the model Confusion matrix was used. There are four ways to check if the predictions are right or wrong: TN / True Negative: the case was negative and predicted negative. TP / True Positive: the case was positive and predicted positive.

On the training set; the logistic model made the correct investment decision 92% of the time. However, the model advised investors to invest when they shouldn't have 5% of the time. That means that when told to invest; the model would be correct 87% of the time.

As seen in the Classification Report (TM6); the model made accurate predictions from classification algorithms over 90% of the time.

TABLE TM6
LOGISTIC REGRESSION CLASSIFICATION REPORT

	Precision	Recall	f1-score	Support
0	0.93	0.87	0.90	158
1	0.90	0.95	0.92	201
accuracy			0.91	359
macro avg	0.92	0.91	0.91	359
weighted avg	0.91	0.91	0.91	359

TABLE TM7
LOGISTIC REGRESSION CONFUSION MATRIX (359 TEST SAMPLES)

True Positive Rate	38.4%
True Negative Rate	53.7%
False Positive Rate	5.6%
False Negative Rate	3.1%

TABLE TM8
ARCHITECTURE OF LOGISTIC REGRESSION MODEL

Parameters	Values used (default)
penalty	l2
Dual	False
Tolerance	0.0001
C	1.0
fit_intercept	True
intercept_scaling	1
class_weight	None
random_state	None
solver	lbfgs
max_iter	100
multi_class	auto
verbose	0

warm_start	False

C. Artificial Neural Network

1. Summary of Approach

The Artificial Neural Network (ANN) model relies on several stages of summing the results of inputs & weights together in order to find deeper meaning in large sets of information. In general an ANN model is divided into layers with a number of nodes that use a certain mathematical function (activation function) to analyze the outputs of other nodes on the same layer. Out of the four models tested, this model's architecture is the most configurable..

The model architecture chosen for the ANN model by comparing the MSE output for each possible combination of hidden layers (1-4), nodes per hidden layer (1-10), and activation function (relu, tanh & logistic).

2. Model Results

After initial testing, the 'relu' activation function with four hidden layers was the unanimous architecture for the top ten performing models. Another round of testing found the most optimal architecture (shown below in Table TM9) had a one time MSE of 117, an average MSE of 216, a max MSE of 1405, and a minimum MSE of 58.

TABLE TM9
ARCHITECTURE OF BEST PERFORMING ANN MODEL

Activation Function	relu
# of Hidden Layers	4
Nodes per Hidden Layer	[7, 5, 8, 9]
Learning Rate	0.001
Tolerance	0.001
Max Iteration	10000

Through 100 iterations, the ANN models had significant variation between MSE (58/1405) and R^2 (-0.008/0.956) performance metrics. However, this variance caused the average MSE to be higher than other models. It is interesting to see numerous models have MSE values under 100 and then to have one above 1000. It is believed that during training, there are random weight values that either are too far from their convergence location or that the learning rate was too high and the weights eventually diverged.

TABLE TM10
ANN MSE PERFORMANCE METRICS

MSE Mean	216.39571
MSE Max	1401.21127
MSE Min	58.45331

TABLE TM11
ANN R^2 PERFORMANCE METRICS

R^2 Mean	0.87369
R^2 Max	0.95584
R^2 Min	-0.00874

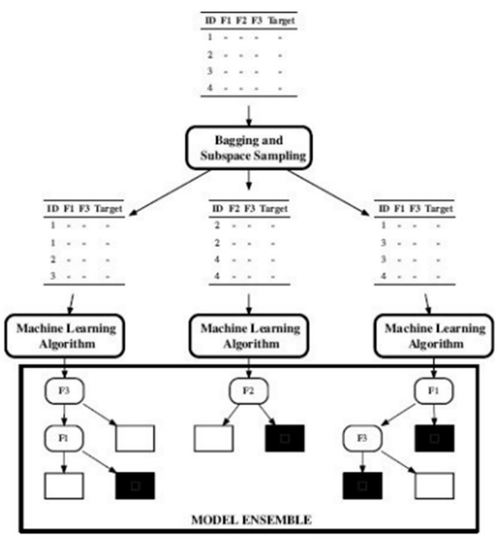
D. Ensemble Model (Random Forest)

1. Summary of Approach

An ensemble model is different from other types of machine learning prediction models, because it is composed of a set of models rather than just a single model. The goal of an ensemble model is to have multiple models work together to solve an issue rather than just one model working alone. However, it is important to avoid group-think, the event where the models began to change their predictions based on the other models. This is avoided by having each model make their predictions independently. The two general ensemble model techniques are boosting and bagging. The model created to tackle this issue was a simple random forest model. A random forest model is the combination of bagging, subspace sampling and a decision tree. This ensemble model makes its predictions based on only the information required. In this case, the target features are continuous, meaning the median is preferred to the mean since the mean is affected more by outliers than the median. A random forest architecture can be seen below in TM12.

TABLE TM12

BAGGING AND SUBSPACE SAMPLING EXAMPLE [22]



2. Model Results

The model's architecture was created from comparing the MSE output when varying the arguments available to the Random Forest Regressor while taking the average across 100 variations. The inputs that have changed from their default input are criterion and random_state. Increasing the n_estimators decreased the MSE output, where 10 to 100 are the available options so 100 was used which happens to be the default. The criterion can vary between squared_error, absolute_error and poisson. Squared_error was selected when attempting to determine the other arguments to use, but was switched to absolute_error since it gave a marginally lower MSE at the cost of an increase in calculation time. Random_state controls the randomness bootstrapping of the samples when creating the trees as well as the sampling of features to consider when making a split from a node. Random_state having a larger value decreased MSE reaching a saturation point around 100.

TABLE TM13

ARCHITECTURE OF BEST PERFORMING RANDOM FOREST MODEL

n_estimators	100
criterion	absolute_error
max_depth	None
random_State	100

The following MSE and R^2 performance metrics were calculated after training 100 unique models:

TABLE TM14

RANDOM FOREST MSE PERFORMANCE METRICS

MSE Mean	240.05837
MSE Max	209.25048
MSE Min	285.29670

TABLE TM15

RANDOM FOREST R^2 PERFORMANCE METRICS

R^2 Mean	0.84696
R^2 Max	0.82890
R^2 Min	0.86363

V. BEST PERFORMING MODEL

Based on the average MSE from 100 separately trained models, the Multivariate Linear Regression model was the best for predicting the stock price of the QualComm corporation 28 days in the future. The Linear model had an average MSE of 89 while the other regression models had average MSEs around 200-250 (as seen below in Table PM1).

TABLE PM1
SUMMARY OF MODEL PERFORMANCES

Model	Avg MSE after 100 Iterations
Multivariate Linear Regression	**89**
Logistic Regression	x
Artificial Neural Network	217
Random Forest (Ensemble)	240

The Logistic Regression Model was not considered for the 'best model' to predict stock price, because it is a classification model that does not predict the actual price of the stock in the future. The Logistic Regression model is a tool that can be used to support another model by plainly indicating when one should, or should, not invest in the QualComm corporation. A blanket 'Yes' or 'No' is a more risky investment tool by its nature of not being able to indicate to the user how much their investment is predicted to grow. Information with relative numbers yields more confidence than a plan boolean decision.

A. Model Architecture

Please see the section Multivariate Linear Regression' for the full details of the Multivariate Linear Regression model's architecture.

B. Predicting Stock Price

Line graphs detailing the Multivariate Linear Regression model's predicted future stock price and the actual price 28 days later for the last five years and the latest 3 months are available on the other page. Please see Figure PM3 and PM4.

As seen in both figures; the linear regression model's predictions are approximately following the general trends of the QualComm stock price. However, the model appears to be following the market instead of actively predicting ahead of the market. This is no surprise since the model is only able to predict what it has seen and cannot respond immediately to real-word results. The first year of model predictions in early 2017 is also an example of the model needing to re-align itself with reality before accurate predictions are made.

Prediction vs actual plots for the other three models (Logistic Regression, ANN, and Random Forest) can be seen in the Appendix section 'B. Other Model 5 Year Stock Predictions Against Actual Close Price'.

C. Income Generated

Mean Square Error is the determinant of which of the models performed the best, however, the ultimate goal of this investigation is to predict the price of QualComm's stock to make the most money. By determining a tolerance of when to invest ("Invest Tolerance"), each model has the ability to signal to an investor when they should have invested in the QualComm corporation. The amount of money that is gained/lost on an investment date is shown below in equation 1.

$$Income = \frac{\$1000}{ActualClose_n} * ActualClose_{n+28} - \$1000 \tag{1}$$

The 'Predicted Income' is the amount of money gained/lost based on the actual price of the stock when funds are invested (purchase price) compared to the model's predicted stock price in 28 days (sell price). The tally for which days to invest, the amount of stock purchased for a $1000 investment, predicted price, and actual price is available for review in the appendix, section *'C. Example Model Recommended Investment Record*'. The model that made the most money was also the model with the lowest MSE (Multivariate Linear Regression), followed by the Logistic Regression mode, then ANN, and Random Forest.

TABLE PM2

INCOME GENERATED FOR ONE INSTANCE* OF EACH MODEL

Model	Number of Investments	Predicted Income	Actual Income	Difference
Multivariate Linear Regression	355	$64347.90	$62126.86	3.5%
Logistic Regression	277	NA	$60047.88	NA
Artificial Neural Network	289	$48981.16	$52537.31	6.7%
Random Forest (Ensemble)	404	$91264.66	$43002.15	47.1%

*This is not an average. The values shown are the result of a single model training for example purposes only.

It is worth noting that all of the income values shown in Table PM2 reflect investments made in hynsite (looking back at how the stock has already performed, not on future dates).

TABLE PM3

5-YEAR PREDICTED VS ACTUAL STOCK PRICE FOR THE MULTIVARIATE LINEAR REGRESSION MODEL

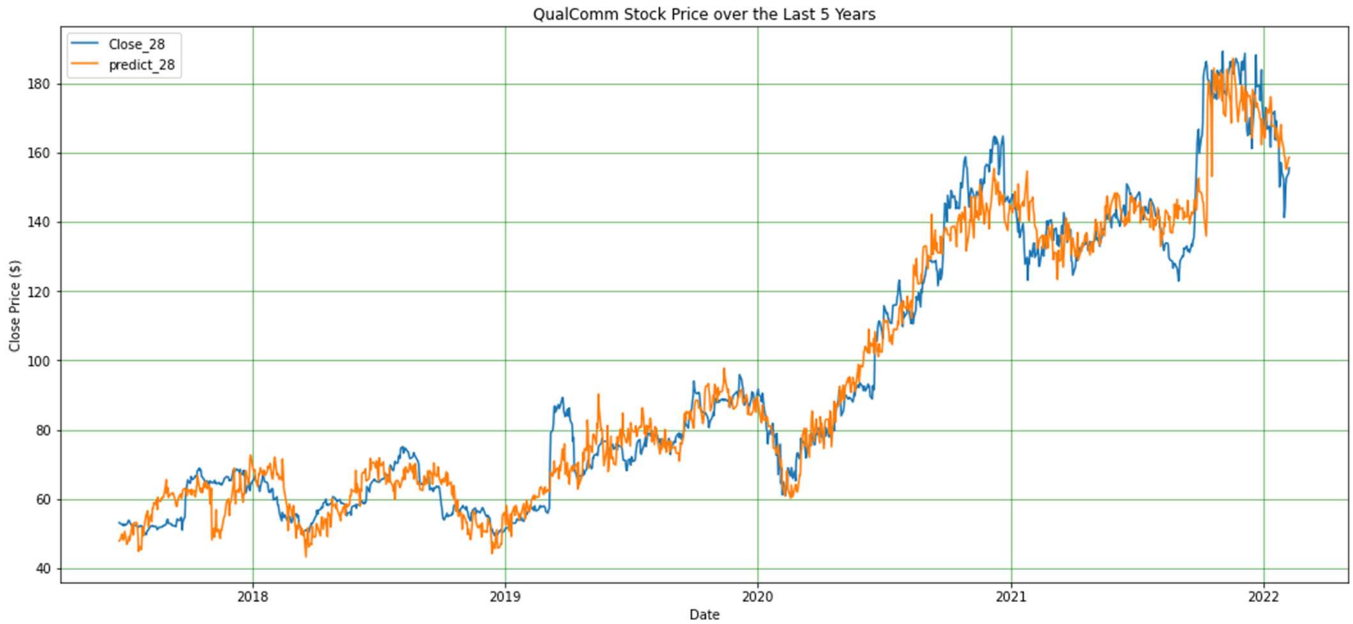
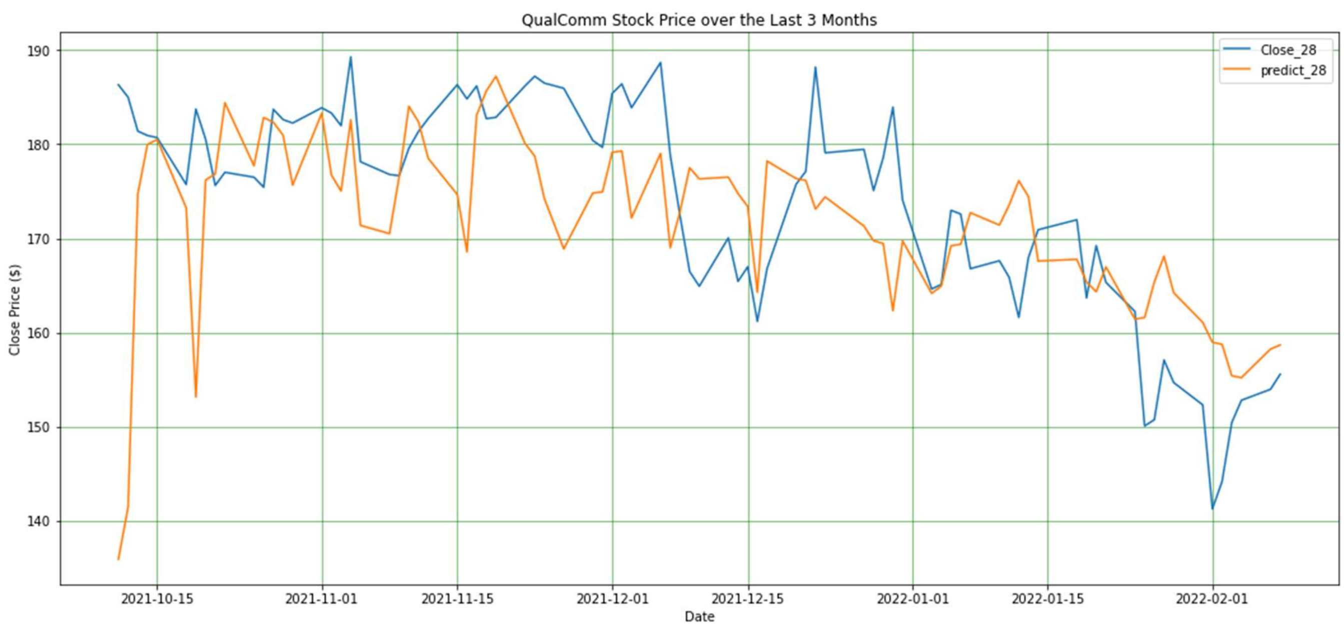


TABLE PM4

3-MONTH PREDICTED VS ACTUAL STOCK PRICE FOR THE MULTIVARIATE LINEAR REGRESSION MODEL



VI. Discussion

A. Why did you choose the model that you did?

By looking at Table PM1 & PM2, the model with the lowest MSE also made predictions that made the most money. The Multivariate Linear Regression model had the lowest MSE, being that of 89 compared to the other models and made the most actual money from investments ($62126.86). The MSE metric was chosen to decide the "best" model, because it assesses the average squared difference between the predicted and actual outcomes. When the model is closely aligned with the real-world, then investors can take full advantage of it.. See section V. Best Model for more information.

B. Is your model good? Why or why not?

The Linear Regression models success can be viewed clearly by looking at Graph(TABLE PM3) and Graph (TABLE PM4), these graphs help show the models accuracy in predicting the QualComm stock price. Since the variation between predicted and actual stock price are so close to each other it can be argued that the Linear Regression model is good.

C. Is the computed income a good deal? Why or why not?

The computed income was arguably a good idea because with each model the profit generated tended to be positive. It can be argued that this information may have influenced the decision making on which model is best, despite not having been how closely the model predicted the stock price. However the correlation cannot be denied.

D. What would you do to improve this model?

The models could possibly be improved by removing features that had little to no effect on the target variable, finding the outside variables that could have a major effect on QualComm's stock price that was not added into the dataset used for calculations, changing the models parameters to be best optimized.

VII. Conclusion

In conclusion; predicting the fluctuation of any particular stock can be a challenging task for anyone, which is why entire careers are based around attempting to do so. The outcome of a prediction is based on a large quantity of variables, some easier to see the connection over others.

For this prediction analysis; using Linear, Logistic, Neural-Network, and Ensemble models to predict the QualComm stock priced on a large range of features tended to lead to positive investing choices. They each approached the problem in their own way. Linear focused on modeling the correlation between dependent and independent variables. Logistic used an input variable to calculate a discrete outcome, due to its binary nature to focus on when to predict versus when not to predict. Neural-Network goal is to use several stages of summing the results caused by the inputs and weights together with the end goal being to find a deeper meaning between the features in a large set of information. Ensemble makes its prediction based on the cumulative prediction of other models merged together in order to make a more "informed" prediction.

In this analysis; it was determined that by looking at a model's MSE as well as income generated, the Linear Regression model was best fitted for predicting the stock market price. Each model's outcome could be swayed based on the parameters chosen, the data preparation and features chosen.

APPENDIX

A. Data Quality Report of the Final Dataset

The following Data Quality Report does not include previous day feature columns (IE: A column dedicated to the close value x days ago). See section: 'III. Data Preparation' for more details.

TABLE A1
DATA QUALITY REPORT OF FEATURES USED IN MODEL TRAINING BEFORE PROCESSING.

stat	Date	Close/Last	Volume	Open	High	Low
count	1260	1260	1260	1260	1260	1260
cardinality	1260	1149	1259	1156	1180	1180
mean	*	94.34067	11185564	94.35892	95.64676	93.06873
median	*	77.91	9396205	77.595	78.9475	76.685
number at median	*	0	0	0	0	0
mode	5/1/2017	52.49	11301880	52	58.49	52.27
number at mode	1	4	2	3	4	3
stddev	*	39.76462	8104528	39.82371	40.48734	39.08962
min	*	49.4	2120165	49.52	49.8	48.56
number at min	*	1	1	1	1	1
max	*	189.28	1.56E+08	190.304	193.58	185.1852
number at max	*	1	1	1	1	1
number of zeros	*	0	0	0	0	0

number missing	0	0	0	0	0	0

All measurements annotated with ‘*’ represents an invalid mathematical operation with a non-numeric feature

TABLE A2
DATA QUALITY REPORT OF FEATURES USED IN MODEL TRAINING BEFORE PROCESSING.

stat	Gross Profit	Net Income Available to Common Shareholders	InflationRate	Total Assets	Total Liabilities	Net Income/Starting Line
count	1260	1260	1260	1260	1260	1260
cardinality	1238	1239	589	1240	1241	1238
mean	3826.356	922.0514	2.548702	43858.82	31443.82	922.1066
median	3172.957	843.8261	2.149194	37408.94	30194.48	844.1992
number at median	0	0	0	0	0	0
mode	6402	3399	1.7	42820	31487	3399
number at mode	20	20	111	20	20	20
stddev	1185.773	1631.14	1.241059	12563.45	4011.146	1630.572
min	2654	-5913.26	1.2	31938	22341.87	-5912.93
number at min	1	1	21	1	1	1
max	7521	3399	5.5	65473.39	40425.68	3399
number at max	1	1	21	1	1	1
number of zeros	0	0	0	0	0	0
number missing	0	0	0	0	0	0

TABLE A3
DATA QUALITY REPORT OF FEATURES USED IN MODEL TRAINING BEFORE PROCESSING.

stat	Dividends Paid	Net Changes in Cash	BrentCrudeOil	DGS3MO	DGS2	DGS10
count	1260	1260	1260	1260	1260	1260
cardinality	1230	1241	1136	221	244	253
mean	-778.002	-64.1719	62.55909	1.069576	1.338688	1.923952
median	-763.234	-142.648	64.50149	1.05	1.47	1.9
number at median	0	0	0	12	7	6
mode	-765	-509	80.76636	0.05	0.16	1.63
number at mode	20	20	125	78	83	17
stddev	55.80532	6477.438	13.79855	0.904571	0.951453	0.751955
min	-911	-25586.4	23.33727	0	0.09	0.52
number at min	1	1	1	2	1	1
max	-705	19877.92	83.65	2.49	2.98	3.24
number at max	1	1	1	2	1	1
number of zeros	0	0	0	4	0	0
number missing	0	0	0	10	10	10

TABLE A4
DATA QUALITY REPORT OF FEATURES USED IN MODEL TRAINING BEFORE PROCESSING.

stat	DTWEXBGS	CBBTCUSD	Close_AAPL	Close_ERIXF	Close_GOOGL	Close_INTL
count	1260	1260	1260	1260	1260	1260
cardinality	1233	1260	1221	582	1250	968
mean	114.2736	18143.64	83.43226	9.307935	1562.801	50.62603
median	114.5039	9298.16	60.67375	8.955	1245.9	50.59
number at median	0	1	0	3	0	2
mode	113.6874	1436.5	43.125	9	1005.65	46.7
number at mode	2	1	3	19	2	4
stddev	3.336445	17716.88	45.11082	2.151958	633.0315	7.145869
min	106.4903	1436.5	35.5475	5.54	919.46	33.46
number at min	1	1	1	1	1	1
max	126.1428	67510.06	182.01	14.36	2996.77	68.47
number at max	1	1	1	1	1	1
number of zeros	0	0	0	0	0	0
number missing	24	1	0	0	0	0

TABLE A5
DATA QUALITY REPORT OF FEATURES USED IN MODEL TRAINING BEFORE PROCESSING.

stat	Close_NXPI	Close_SSNLF	Close_TMUS	Close_VZ	Close_28
count	1260	1260	1260	1260	1260
cardinality	1169	17	1123	821	1129
mean	132.6214	1961.236	90.9945	54.49279	95.1987
median	115.88	2210	78.59	55.27	78.615
number at median	3	940	2	2	0
mode	97.65	2210	59.75	55.78	52.49
number at mode	3	940	3	5	4
stddev	43.37271	671.3485	28.31623	4.373864	39.79902
min	64.56	57.75	55.36	42.89	49.4
number at min	1	138	1	1	1
max	238.9	2450	149.41	62.07	189.28
number at max	1	138	1	1	1
number of zeros	0	0	0	0	0
number missing	0	0	0	0	28

B. Other Model 5 Year Stock Predictions Against Actual Close Price

Logistic Regression

Figure B1

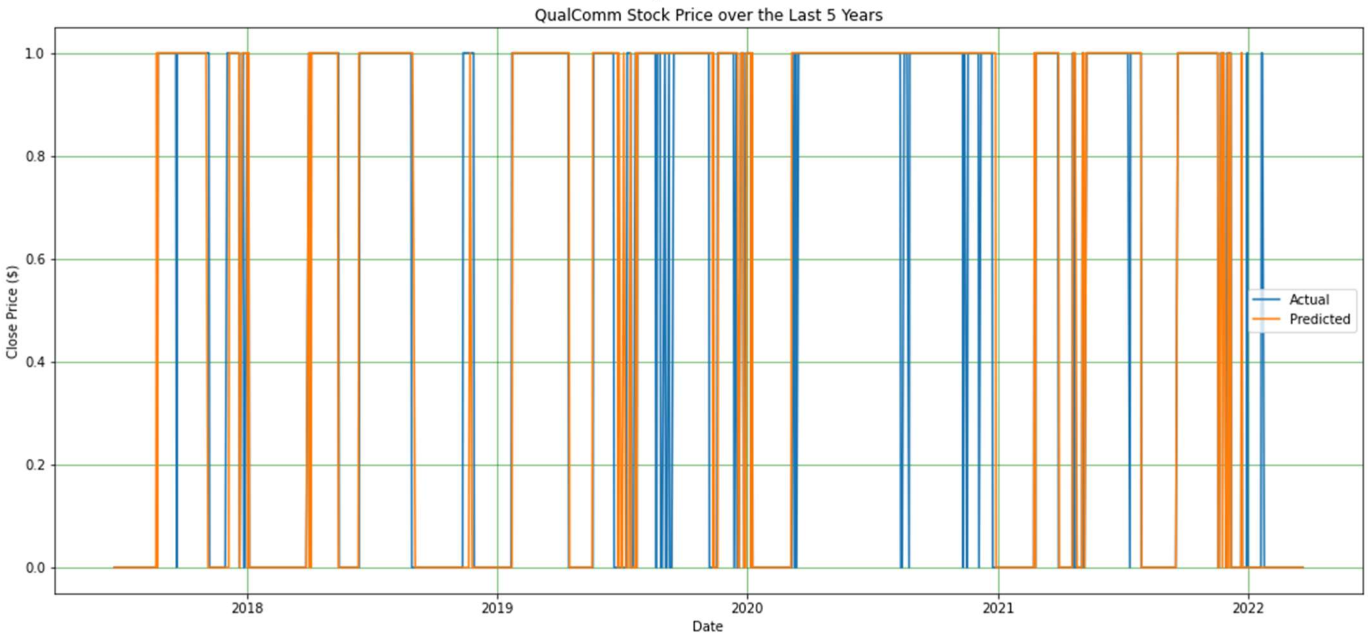
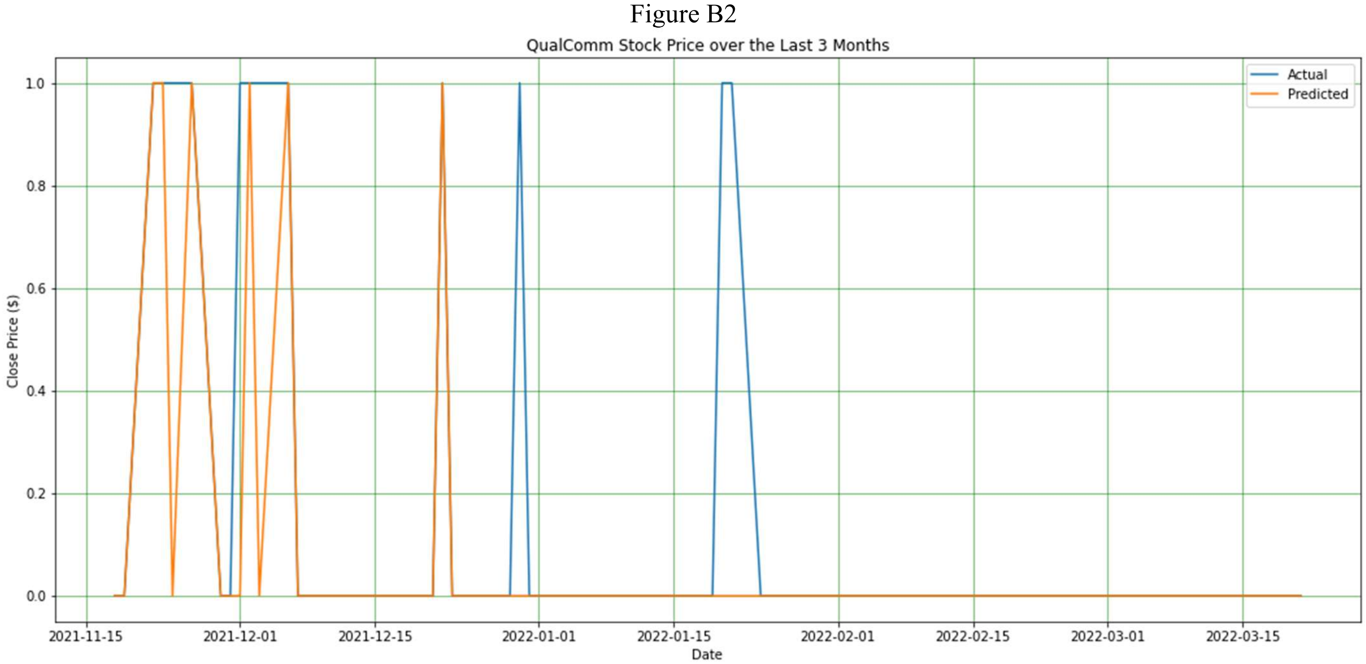


Figure B2



*Note: The Logistic Regression prediction vs actual close price plots are referenced in binary. '1 = You should Invest' and '0 = You should not Invest'. Therefore, an accurate model prediction will prejudice an identical 1/0 plot.

Artificial Neural Network

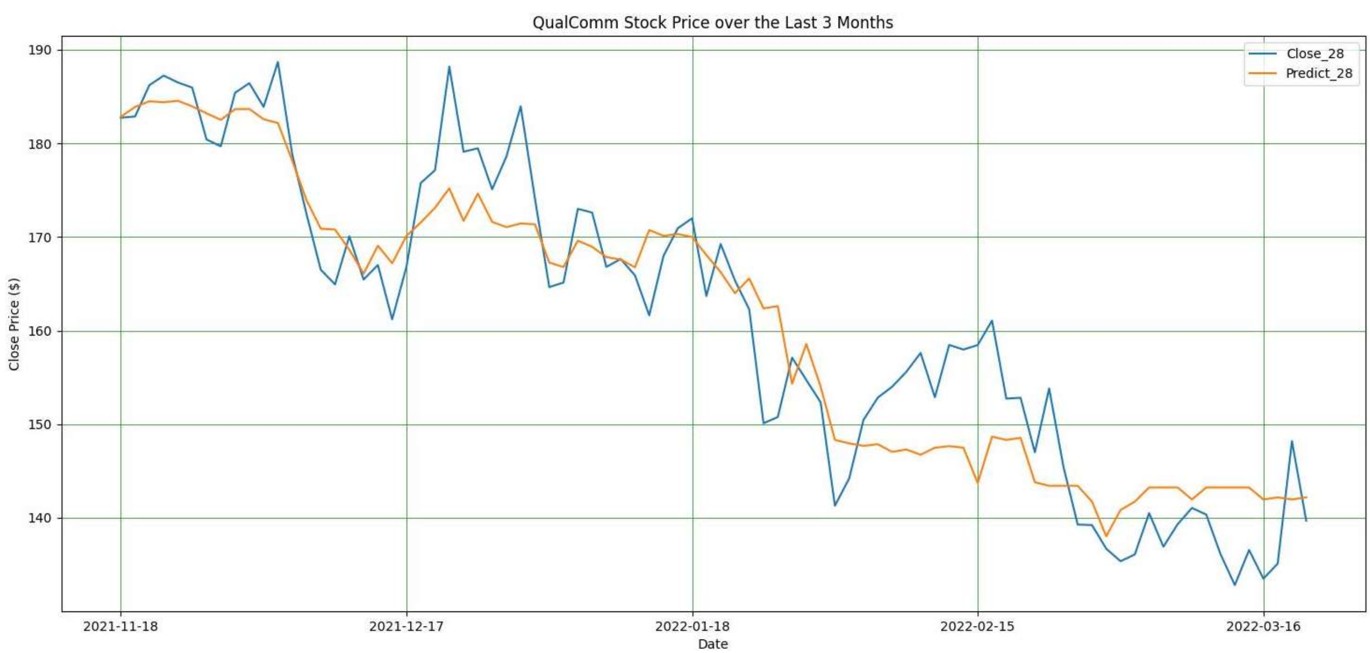
Figure B3

Figure B4

Random Forest

Figure B5

Figure B6



C. Example Model Recommended Investment Record

Index	Date	Quantity	Purchase Close	Sell Close	Model Sell Close Price	Predicted Income	Actual Income
40	2017-08-18	19.260	51.92	51.75	58.90	134.47	-3.27
41	2017-08-21	19.238	51.98	51.84	57.22	100.95	-2.69
43	2017-08-23	19.146	52.23	52.02	59.69	143.01	-4.02
44	2017-08-24	19.051	52.49	51.96	59.43	132.24	-10.09
45	2017-08-25	19.219	52.03	52.35	59.64	146.38	6.15
...	...	...	...	...	...	...	...
1100	2021-11-03	7.221	138.48	181.98	172.16	243.26	314.12
1101	2021-11-04	6.405	156.11	189.28	180.02	153.19	212.47
1105	2021-11-10	6.257	159.8	179.58	182.02	139.09	123.77
1106	2021-11-11	6.081	164.42	181.38	182.88	112.30	103.15
1107	2021-11-12	6.062	164.94	182.74	182.58	106.97	107.91

E. Python Code for Data Quality Analysis

Flow Controls and Data Handling

```
# Flow control and data handling settings
steps:
 ## Raw base data loading
 - module: process.prepare.Load
   input:
     name: input_raw
   output:
     name: raw
     print: true
 - module: process.dqr.DQR
   input:
     name: raw
   output:
     write: true
     name: qcom_stock_daily_raw
 - module: process.prepare.ConvertTypes
   args:
     columns:
       Date: date
       Close/Last: currency
       Open: currency
       High: currency
       Low: currency
   input:
     name: raw
   output:
     name: clean
     print: true
 - module: process.prepare.Sort
   args:
     sortColumn: Date
     ascending: True
   input:
     name: clean
   output:
     name: clean
 - module: process.dqr.DQR
   input:
     name: clean
   output:
     write: true
     name: qcom_stock_daily
 ## QCOM profit/loss data loading
 - module: process.prepare.Load
   input:
     name: input_qcom_profit_loss
   output:
     name: qcom_profit_loss_raw
 - module: process.prepare.ConvertTypes
   args:
     columns:
       Date: date
       Revenue: float
       "Cost of revenue": float
       "Gross Profit": float
       "Operating Expenses": float
       "Operating Income (Loss)": float
       "Non-Operating Income (Loss)": float
       "Pretax Income (Loss)": float
```

```
    "Income Tax (Expense) Benefit, net": float
    "Income (Loss) Including Minority Interest": float
    "Minority Interest": float
    "Net Income Available to Common Shareholders": float
  input:
    name: qcom_profit_loss_raw
  output:
    name: qcom_profit_loss_raw
- module: process.dqr.DQR
  input:
    name: qcom_profit_loss_raw
  output:
    write: true
    name: qcom_profit_loss
## Inflation rate data loading
- module: process.prepare.Load
  input:
    name: input_inflation
  output:
    name: inflation_raw
- module: process.prepare.ConvertTypes
  args:
    columns:
      Date: date
      InflationRate: float
  input:
    name: inflation_raw
  output:
    name: inflation_raw
- module: process.dqr.DQR
  input:
    name: inflation_raw
  output:
    name: inflation
# QCOM Balance Loading
- module: process.prepare.Load
  input:
    name: input_qcom_balance
  output:
    name: qcom_balance_raw
- module: process.prepare.ConvertTypes
  args:
    columns:
      Date: date
      Assets: float
      "Cash, Cash Equivalents & Short Term Investments": float
      "Accounts & Notes Receivable": float
      Inventories: float
      "Other Short Term Assets": float
      "Total Current Assets": float
      "Property, Plant & Equipment, Net": float
      "Long Term Investments & Receivables": float
      "Other Long Term Assets": float
      "Total Noncurrent Assets": float
      "Total Assets": float
      Liabilities: float
      "Payables & Accruals": float
      "Short Term Debt": float
      "Other Short Term Liabilities": float
      "Total Current Liabilities": float
      "Long Term Debt": float
      "Other Long Term Liabilities": float
      "Total Noncurrent Liabilities": float
      "Total Liabilities": float
```

```
        "Preferred Equity": float
        "Share Capital & Additional Paid-In Capital": float
        "Retained Earnings": float
        "Other Equity": float
        "Equity Before Minority Interest": float
        "Minority Interest": float
        "Total Equity": float
        "Total Liabilities & Equity": float
    input:
      name: qcom_balance_raw
    output:
      name: qcom_balance_raw
  - module: process.dqr.DQR
    input:
      name: qcom_balance_raw
    output:
      write: true
      name: qcom_balance
  # QCOM Cashflow loading
  - module: process.prepare.Load
    input:
      name: input_qcom_cashflow
    output:
      name: cashflow_raw
  - module: process.prepare.ConvertTypes
    args:
      columns:
        Date: date
        "Net Income/Starting Line": float
        "Depreciation & Amortization": float
        "Non-Cash Items": float
        "Change in Working Capital": float
        "Cash from Operating Activities": float
        "Change in Fixed Assets & Intangibles": float
        "Net Change in Long Term Investment": float
        "Net Cash From Acquisitions & Divestitures": float
        "Other Investing Activities": float
        "Cash from Investing Activities": float
        "Dividends Paid": float
        "Cash From (Repayment of) Debt": float
        "Cash From (Repurchase of) Equity": float
        "Other Financing Activities": float
        "Cash from Financing Activities": float
        "Net Cash Before Disc. Operations and FX": float
        "Change in Cash from Disc. Operations and Other": float
        "Net Cash Before FX": float
        "Effect of Foreign Exchange Rates": float
        "Net Changes in Cash": float
    input:
      name: cashflow_raw
    output:
      name: cashflow_raw
  - module: process.dqr.DQR
    input:
      name: cashflow_raw
    output:
      write: true
      name: cashflow
  # Brent crude oil loading
  - module: process.prepare.Load
    input:
      name: input_crude_oil
    output:
      name: crude_oil_raw
```

```
  - module: process.prepare.Rename
    args:
      columns:
        DATE: Date
        POILBREUSDM: BrentCrudeOil
    input:
      name: crude_oil_raw
    output:
      name: crude_oil_raw
  - module: process.prepare.ConvertTypes
    args:
      columns:
        Date: date
        BrentCrudeOil: float
    input:
      name: crude_oil_raw
    output:
      name: crude_oil_raw
  - module: process.dqr.DQR
    input:
      name: crude_oil_raw
    output:
      write: true
      name: crude_oil
  ## Final dataset
  - module: process.prepare.Load
    input:
      name: input_final_post
    output:
      name: final_raw
  - module: process.prepare.StartsWithDrop
    input:
      name: final_raw
    args:
      starts:
        - Prev
        - Unnamed
    output:
      name: final_raw
  - module: process.dqr.DQR
    input:
      name: final_raw
    output:
      write: true
      name: final_data_processed
```


Model Functionality

```
#!/usr/bin/env python3
'''
@module qcommodel.model
@info   Parent module for running end-to-end stock price modeling for the ECE
        5984 group K project2 effort. More to come
'''

# Python libraries
import sys
import os
import yaml
import traceback

# Third party libraries
import pandas as pd
from importlib import import_module
```

```
# Local libraries
from utils.logger import Logger
from utils.dataframe import Summarize
from utils.dataframe import Write

## Model implementation class
class Model():
  ''' This class performs configuration loading, data processing, model
    training and final model storage with abstracted methods for
    intermediate states
  '''

  def __init__(self, config_file):
    ''' Constructor
    :param config_file: String path to the configuration file
    '''
    # Parse the configuration file
    self._config = self.readConfig(config_file)
    assert self._config != None, 'Failed to parse configuration data'

    # Initialize class member data
    self.initVariables()

    # Setup logging
    self.initLogger()

    # Initialization complete
    self.logger.info('Initialization complete')

  ## Data modeling functionality
  ##############################################################################
  @staticmethod
  def loadSteps(steps):
    ''' Load and validate modeling steps
    :param steps: List of step configuration dicts
    :return List: Step objects
    '''
    result = []
    for idx, step in enumerate(steps):
      print(f'Loading step-{idx}')
      result.append(Step(step))
    return result

  def run(self):
    ''' Data prepration wrapper method
    This function uses configuration fields to load, analyze, and prepare
      input data for further modeling
    :return None:
    '''
    # Setup input data/preparation settings
    self.logger.info(f"Processing input files: {self._config_input.keys()}")
    steps = self.loadSteps(self._config_steps)

    outputs = {}
    for key in self._config_input.keys():
      outputs[f'input_{key}'] = self._config_input[key]

    for idx, step in enumerate(steps):
      # Run the step
      self.logger.info('')
      self.logger.info(f'Running step-{idx}: {step.module.__name__}')

      # Handle multiple input objects prior to positional arguments
```

```
        inputs = []
        if type(step.input.name) in [list, tuple]:
            inputs += step.input.name
        else:
            inputs = [step.input.name]

        # Get the object mapping to the input for each specified input
        for idx in range(len(inputs)):
            print(inputs[idx])
            inputs[idx] = outputs[inputs[idx]]

        # Call the function with kwargs if specified
        if step.args:
            output = step.module(*inputs, **step.args)
        else:
            output = step.module(*inputs)
        self.logger.info('==================================================')

        # Process the output of the step
        outputs[step.output.name] = output
        if step.output.write:
            ofn = self._config_output['prefix'].replace('%suffix', step.output.name.lower())
            self.logger.info(f'Writing data to: {ofn}')
            Write(output, ofn)
        if step.output.summarize:
            Summarize(output, prefix=step.output.name)
        if step.output.print:
            print(output)
        self.logger.info('==================================================')

    ## Class support functions
    ##########################################################################
    def initVariables(self):
        ''' Class member variable initialization
        Initializes all class member variables with defaults/config fields
            => Configuration field requirements are defined by the method of
                access. .get for optional, direct access for requried
        :return None:
        '''
        # Utilities
        self.name = self.__class__.__name__
        self._config_logging = self._config.get('logging', {})

        # Debug
        self._config_debug = self._config.get('debug', {})
        self._summarize = self._config_debug.get('summarize', False)

        # Input data configuration
        self._config_input = self._config['input']

        # Output data configuration
        self._config_output = self._config['output']

        # Modeling steps configuration
        self._config_steps = self._config['steps']

    def initLogger(self):
        ''' Logger initialization function
        Initializes class member 'logger'
        :return None:
        '''
        # Just pass config file logging params through
        self._config_logging['name'] = self.name
        self.logger = Logger(**self._config_logging)
```

```
    @staticmethod
    def readConfig(fn):
        ''' Configuration file parser
        :param fn: String path to the config file
        :return dict: Config file contents
        '''
        result = None
        assert os.path.exists(fn), 'Missing/invalid configuration: {fn}'
        try:
            with open(fn, 'r') as fd:
                result = yaml.safe_load(fd)
        except Exception as exc:
            print(f'Failed to parse configuration: {fn}')
            print(f'\n{traceback.format_exc()}')
        return result

## Modeling 'Step' wrapper class/object
class Step:
    class Input:
        def __init__(self, config):
            self.validate(config)
            self.name = config['name']

        def validate(self, config):
            ''' Validate input settings '''
            assert type(config['name']) in [str, list, tuple], f"Invalid input name: {config['name']}"

    class Output:
        def __init__(self, config):
            self.validate(config)
            self.name = str(config['name']).lower()
            self.write = bool(config.get('write', False))
            self.print = bool(config.get('print', False))
            self.summarize = bool(config.get('summarize', False))

        def validate(self, config):
            ''' Validate input settings '''
            assert isinstance(config['name'], str), f"Invalid output name: {config['name']}"

    def __init__(self, config):
        ''' Constructor
        :param method: String path to the python module
        :param input: Python dict with data descriptor fields
            - name:
        :param output: Python dict with data descriptor fields
            - write: Bool write to file flag
            - print: Bool print object flag
            - summarize: Bool pd.dataframe summary flag
            - name: String name of the output for mapping to other steps
        '''
        self.validate(config)
        module = config['module'].split('.')
        method = module[-1]
        module = '.'.join(module[:-1])
        try:
            self.module = import_module(module)
            self.module = getattr(self.module, method)
        except Exception:
            print(traceback.format_exc())
            print(f"Step - Failed to load module: {config['module']}")
            raise ImportError(f"Failed to load module: {config['module']}")
        self.args = config.get('args', None)
        self.input = self.Input(config['input'])
```

```
        self.output = self.Output(config['output'])

    def validate(self, config):
        ''' Step configuration validation '''
        assert 'module' in config, 'Step missing required field "module"'
        assert 'input' in config, 'Step missing required field "input"'
        assert 'output' in config, 'Step missing required field "output"'

if __name__ == '__main__':
    # Create the model class
    if len(sys.argv[1:]):
        fn = sys.argv[1]
    else:
        #fn = os.path.dirname(os.path.realpath(__file__)) + '/conf/ann.yaml'
        #fn = os.path.dirname(os.path.realpath(__file__)) + '/conf/model.yaml'
        fn = os.path.dirname(os.path.realpath(__file__)) + '/conf/ensemble.yaml'
    print(f'Using configuration file: {fn}')
    model = Model(fn)

    # Run the modeling steps
    model.run()

'''
@module     qualcomm.process.prepare
@info       General data processing support
@author     ece5984-groupk
'''

# Python libraries
import yaml
import os
import sys

# Third party libraries
import pandas as pd
import numpy as np

## Data preparation functions
################################################################################
def Load(data_file):
    ''' Static data loading method
    :param data_file: String path to the data file
    :return pd.DataFrame: DataFrame housing data file contents
    '''
    if not os.path.exists(data_file):
        print(f'{data_file} not found - Searching in parent directory', file=sys.stderr)
        data_file = os.path.dirname(__file__) + '/../../' + data_file
    assert os.path.exists(data_file), f'Invalid data file: {data_file}'
    dtype = data_file.split('.')[-1].upper()
    result = None
    if dtype == 'CSV':
        result = pd.read_csv(data_file)
    else:
        raise TypeError(f'Unsupported file type: {dtype}')
    return result

def Rename(df: pd.DataFrame, columns: dict):
    ''' Rename columns
    :param df: Pandas dataframe
    :param columns: Dict of format {old_name: new_name}
    :return pd.DataFrame: Updated dataframe
    '''
    result = df.copy()
    result = result.rename(columns=columns)
```

```
    return result

def CapitalizeColumns(df: pd.DataFrame, upper: bool=True):
    ''' Capitalize column names
    :param df: Pandas dataframe
    :param upper: Boolean uppercase flag (default True)
    :return pd.DataFrame: Updated dataframe
    '''
    if upper:
        df.columns = [col.upper() for col in df.columns]
    else:
        df.columns = [col.lower() for col in df.columns]
    return df

def ExpandDate(df: pd.DataFrame, column: str):
    ''' Expand a date string column into day month year
    :param df: Pandas dataframe
    :param column: String name of the column to update
    :return pd.DataFrame: Updated dataframe
    '''
    result = df.copy()
    date = pd.to_datetime(result[column])
    result['year'] = date.dt.year
    result['month'] = date.dt.month
    result['day'] = date.dt.day
    return result

def ConvertTypes(df: pd.DataFrame, columns: dict):
    ''' Convert problematic data types to target formats
    :param df: Pandas dataframe
    :param columns: Dict of format {column_name: column_type}
    :return pd.DataFrame: Updated dataframe
    '''
    result = df.copy()
    for name, type in columns.items():
        type = type.lower().strip()
        if type == 'currency':
            result[name] = result[name].replace('[\$,]', '', regex=True).replace(',', '').astype(float)
        elif type == 'date':
            result[name] = pd.to_datetime(result[name]).dt.date
        elif type == 'float':
            result[name] = result[name].replace(',', '').replace(',', '', regex=True).astype(float)
        else:
            raise ValueError(f'Unsupported data type: {type}')
    return result

def Sort(df: pd.DataFrame, sortColumn: str, ascending: bool=True):
    ''' Sort a pd dataframe in ascending/descending fashion based on a single
            column's values
    :param df: Pandas DataFrame object
    :param sortColumn: String name of the column to sort with
    :param ascending: Boolean [a/de]scending flag (default: True/Ascending)
    :return pd.DataFrame: Output dataframe object
    '''
    result = df.copy()
    result = result.sort_values(by=sortColumn, ascending=ascending, ignore_index=True)
    return result

def InterpolateAndConcatByDate(
    target: pd.DataFrame,
    source: pd.DataFrame,
    columns: list,
    method: str='linear'
):
```

```
    ''' Method for interpolating a data set to match a target and then add
            selected columns to the original
    :param: target: DataFrame to match and append new columns to
    :param source: DataFrame with new values
    :param columns: List of string column names to append
    :param method: String pandas interpolation method
    :return pd.DataFrame: resulting DataFrame
    '''
    # Copy the target and source so we don't update the inputs
    result = target.copy()
    src = source.copy()

    # Reindex the target to it's Date column and the source by the full date range
    result.index = result.reindex(target['Date']).index
    src = src.set_index('Date')

    # Reindex the source or new data by the target range
    #   Note: The min/max functions handle deltas in start and stop date for the
    #       two datasets. The interpolate function handles fitting missing data
    startDate = min(result.index[0], src.index[0])
    stopDate = max(result.index[-1], src.index[-1])
    fullIndex = pd.date_range(startDate, stopDate, freq='1D')
    src = src.reindex(fullIndex, fill_value=np.nan)

    # Interpolate nan values
    for column in columns:
        src[column] = src[column].interpolate(method=method)

    # Add the applicable date value to the target and return
    for column in columns:
        result.insert(len(result.columns), column, src[column])

    # Reset the output index back to normal
    result.index = result.reindex(target.index).index
    return result

def StartsWithDrop(df: pd.DataFrame, starts: list):
    result = df.copy()
    toDrop = []
    for start in starts:
        for column in result.columns:
            if column.startswith(start):
                toDrop.append(column)
    return result.drop(columns=toDrop)


'''
@module     qualcomm.dqr.DQR
@info       Wrapper method for returning a data quality report dataframe given
            a standard pandas dataframe
@author     ece5984_groupk
'''

# Python libraries

# Third party libraries
import pandas as pd
import numpy as np

def DQR(data: pd.DataFrame) -> pd.DataFrame:
    ''' Given a pandas dataframe, generated a DQR table
    :param data: Pandas DataFrame object
    :return pd.DataFrame: Data quality report
    '''
```

```
    dqr = pd.DataFrame()
    dqr['statistic'] = [
        'count',
        'cardinality',
        'mean',
        'median',
        'n_at_median',
        'mode',
        'n_at_mode',
        'stddev',
        'min',
        'n_at_min',
        'max',
        'n_at_max',
        'n_zero',
        'n_missing'
    ]
    for column in data.columns:
        mode = data[column].mode()
        if not len(mode):
            continue
        mode = mode[0]
        value_counts = data[column].value_counts()
        if data.dtypes[column] in [np.object_]:
            entry = [
                data[column].size,
                len(data[column].unique()),
                np.nan,
                np.nan,
                np.nan,
                mode,
                value_counts.get(mode, 0),
                np.nan,
                np.nan,
                np.nan,
                np.nan,
                np.nan,
                np.nan,
                data[column].isnull().sum()
            ]
        else:
            median = data[column].median()
            min = data[column].min()
            max = data[column].max()
            entry = [
                data[column].size,
                len(data[column].unique()),
                data[column].mean(),
                median,
                value_counts.get(median, 0),
                mode,
                value_counts.get(mode, 0),
                data[column].std(),
                min,
                value_counts.get(min, 0),
                max,
                value_counts.get(min, 0),
                value_counts.get(0, 0) + value_counts.get(0.0, 0),
                data[column].isnull().sum()
            ]
        dqr[column] = entry
    return dqr
```

Multivariate Linear Regression

```
##############################################
#Multivariate Linear Regression
##############################################
import pandas as pd
import numpy as np
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
import seaborn as sn
from sklearn import utils
from sklearn import preprocessing as preproc
import helperFunctions
df= pd.read_csv("C:/Users/anagh/OneDrive/Desktop/ML/FINAL_MODEL_DATASET.csv")
df.head()
z=['Close/Last', 'Open', 'High', 'Low']
df= reformatDailyDates(df, True)
df
df= appendPastData( df, 2, ['Close/Last'], True )
df= addTarget(df, 'Close_28', 28, True)
df
X= df.drop(['Date','Close_28'], axis=1).to_numpy()
y= df['Close_28'].to_numpy()
scaler= MinMaxScaler(feature_range=(-1,1))

scalertrain = scaler.fit(X)
X = scalertrain.transform(X)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state= 22222)

from sklearn.linear_model import LinearRegression
model= LinearRegression()
model.fit(X_train, y_train)
prediction= model.predict(X_test)
prediction


from sklearn import metrics
from sklearn.metrics import mean_squared_error
print("Mean squared error: %.2f" % mean_squared_error(y_test, prediction))
from sklearn.metrics import r2_score
print("Coefficient of determination: %.2f" % r2_score(y_test, prediction))
import sklearn.model_selection as modelsel
mse_results = []
rs_results = []
for i in range(100):
    X_train, X_test, y_train, y_test =  modelsel.train_test_split(X, y, test_size=0.30)

    model= LinearRegression()
    model.fit(X_train, y_train)
    prediction= model.predict(X_test)
    print("Mean squared error: %.2f" % mean_squared_error(y_test, prediction))
    mse_results.append(metrics.mean_squared_error(y_test, prediction))
    print("Coefficient of determination: %.2f" % r2_score(y_test, prediction))
    rs_results.append(metrics.r2_score(y_test, prediction))
mseMean = np.mean(mse_results)
mseMin = np.min(mse_results)
```

```
mseMax = np.max(mse_results)
print("\n\rLinear Regression:\nMean = {}\nMax = {}\nMin = {}".format(mseMean, mseMax, mseMin))
rsMean = np.mean(rs_results)
rsMin = np.min(rs_results)
rsMax = np.max(rs_results)
print("\n\rLinear Regression:\nMean = {}\nMax = {}\nMin = {}".format(rsMean, rsMax, rsMin))
model.predict(X)
df['predict_28'] = model.predict(X)
df
calcIncome(df,'Close_28', 1000, 28, 1.1)
plot_5yr(df, 'Date', 'Close_28', 'predict_28')
plot_3month(df, 'Date', 'Close_28', 'predict_28')


'
```

Logistic Regression

```
###############################################
#Logistic Regression
###############################################
import helperFunctions
import pandas as pd
import numpy as np
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
import seaborn as sn
from sklearn import utils
from sklearn import preprocessing as preproc
df= pd.read_csv("C:/Users/anagh/OneDrive/Desktop/ML/FINAL_MODEL_DATASET.csv")
df.head()
z=['Close/Last', 'Open', 'High', 'Low']
df= reformatDailyDates(df, True)
df
df= appendPastData( df, 2, ['Close/Last'], True )
TOL= 1
df["result"] = np.where(df["Close_28"]>df['Close/Last']*TOL, 1,0)
X= df.drop(['Date','result'], axis=1).to_numpy()
y= df['result'].to_numpy()
scaler= MinMaxScaler(feature_range=(-1,1))
scalertrain = scaler.fit(X)
X = scalertrain.transform(X)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state= 22222)
from sklearn.linear_model import LogisticRegression
model= LogisticRegression()
model.fit(X_train, y_train)
prediction= model.predict(X_test)
prediction
from sklearn.metrics import classification_report
report = classification_report(y_test,prediction)
print(report)
from sklearn import metrics
confusion= metrics.confusion_matrix(y_test, prediction)
print(confusion)
model.predict(X)
df['predict_28'] = model.predict(X)
df
calcIncome(df,'Close_28', 1000, 28, 1.5)

plot_5yr(df, 'Date', 'result', 'predict_28')
plot_3month(df, 'Date', 'result', 'predict_28')
```

```
'''
ann.py
@author:    John Smutny
@team:      James Ensminger, Ben Johnson, Anagha Mudki, John Smutny
@info:      Regression artificial neural network (ann) model to predict the
           future stock price of the Qualcomm semiconductor company.

           The ann model cycles through several model frameworks and then
           chooses the best architecture to maximize profit from a $1000
           investment 28 days prior to sale.
'''

from sklearn import neural_network as ann
from sklearn import metrics
from sklearn import preprocessing as preproc
import sklearn.model_selection as modelsel
import numpy as np
import pandas as pd

import helperFunctions as hf

### Set Constants
#######################################

OUTPUT_FILES = True
ASCENDING_DATES = True

INCOME_TOLERANCE = 1.10
PREDICT_FUTURE_DAY = 28
INVESTMENT = 1000

INPUT_QUALCOMM = '../../data/QCOM_HistoricalData_5yr.csv'
INPUT_QUALCOMM_FINAL = \
   '../../data/final_extended_data_no_past_data_clean_extended.csv'
INPUT_FINAL = '../../data/FINAL_MODEL_DATASET.csv'
APPLE_FILE = '../../data/AAPL_HistoricalData_5yr.csv'
GOOGLE_FILE = '../../data/GOOGL_HistoricalData_5yr.csv'
ERICSSON_FILE = '../../data/ERIXF_HistoricalData_5yr.csv'
INTEL_FILE = '../../data/INTL_HistoricalData_5yr.csv'
NXP_FILE = '../../data/NXPI_HistoricalData_5yr.csv'
SAMSUNG_FILE = '../../data/SSNLF_HistoricalData_5yr.csv'
TMOBILE_FILE = '../../data/TMUS_HistoricalData_5yr.csv'
VERIZON_FILE = '../../data/VZ_HistoricalData_5yr.csv'
INPUT_BOND03m = '../../data/marketYield_3monthUsTreasureySecurity_5yr.csv'
INPUT_BOND02 = '../../data/marketYield_2YrUsTreasureySecurity_5yr.csv'
INPUT_BOND10 = '../../data/marketYield_10YrUsTreasureySecurity_5yr.csv'
INPUT_DOLLAR = '../../data/nominalBroadUSDollarIndex-5yr.csv'
INPUT_BITCOIN = '../../data/CoinbaseBitcoin_5yr.csv'
INPUT_COMPANY = '../../data/QCOM-SimFin-data-REFORMATTED.xlsx'

OUTPUT_INCOME = '../../artifacts/annIncome-TOL{}.xlsx'.format(INCOME_TOLERANCE)

IDName = "Date"
TARGET_NAME = "Close_{}".format(PREDICT_FUTURE_DAY)


# Artificial Neural Network Settings
```

```
#######################################

TRAIN_RATIO = 0.8
TEST_RATIO = 0.2
VALID_DATA_FROM_TRAIN = 0.25
RANDOM_SEED = 10

HIDDEN_LAYERS = [7, 5, 8, 9]
ACTIVATION_FCT = 'relu'
SOLVER = 'adam'
MAX_ITER = 10000
LEARNING_RATE = 0.0001 * 10
TOLERANCE = 0.0001* 100
EARLY_STOPPING = True


### Data Processing
#######################################

def prepData(df: pd.DataFrame) -> pd.DataFrame:
   FINANCIAL_FEATURES = ['Close/Last', 'Open', 'High', 'Low']

   # Data cleaning of the main QualComm stock data
       # Below is Commented out b/c of ymal ann.py script
   #df = hf.removeDollarSign(df, FINANCIAL_FEATURES)
   df = hf.reformatDailyDates(df, ASCENDING_DATES)  # Re-order dates

   # Add new independent variables to help model stock price.
   df = hf.addFedData(df, 'DGS3MO', INPUT_BOND03m, ASCENDING_DATES)
   df = hf.addFedData(df, 'DGS2', INPUT_BOND02, ASCENDING_DATES)
   df = hf.addFedData(df, 'DGS10', INPUT_BOND10, ASCENDING_DATES)
   df = hf.addFedData(df, 'DTWEXBGS', INPUT_DOLLAR, True)
   df = hf.addFedData(df, 'CBBTCUSD', INPUT_BITCOIN, True)
   df = hf.addStockClosePrice(df, 'AAPL', APPLE_FILE, True)
   df = hf.addStockClosePrice(df, 'ERIXF', ERICSSON_FILE, True)
   df = hf.addStockClosePrice(df, 'GOOGL', GOOGLE_FILE, True)
   df = hf.addStockClosePrice(df, 'INTL', INTEL_FILE, True)
   df = hf.addStockClosePrice(df, 'NXPI', NXP_FILE, True)
   df = hf.addStockClosePrice(df, 'SSNLF', SAMSUNG_FILE, True)
   df = hf.addStockClosePrice(df, 'TMUS', TMOBILE_FILE, True)
   df = hf.addStockClosePrice(df, 'VZ', VERIZON_FILE, True)

   EXPAND30 = ['Close/Last', 'Volume']
   df = hf.appendPastData(df, 30, EXPAND30, ASCENDING_DATES)

   EXPAND05 = ['DGS3MO', 'DGS2', 'DGS10', 'DTWEXBGS',
               'Close_AAPL', 'Close_ERIXF', 'Close_GOOGL', 'Close_INTL',
               'Close_NXPI', 'Close_SSNLF', 'Close_TMUS', 'Close_VZ']
   df = hf.appendPastData(df, 5, EXPAND05, ASCENDING_DATES)

   # Add the future price target.
   df = hf.addTarget(df, TARGET_NAME, PREDICT_FUTURE_DAY, ASCENDING_DATES)

   return df


def doANN(df: pd.DataFrame):
   # Normalize and separate data into Independent & Dependent Variables.
   X = df.drop([IDName, TARGET_NAME], axis=1).to_numpy()
   scalerX = preproc.MinMaxScaler()
```

```
    scalerX.fit(X)
    X = scalerX.transform(X)

    Y = df[TARGET_NAME].to_numpy()
    trainX, testX, trainY, testY = \
        modelsel.train_test_split(X, Y, test_size=TEST_RATIO,
                                  random_state=RANDOM_SEED)
    # Record and report the average mse of the ANN model
    print("doANN: Start modeling loop.")
    mse_results = []
    r2_results = []
    for i in range(100):
        trainX, testX, trainY, testY = \
            modelsel.train_test_split(X, Y, test_size=TEST_RATIO,
                                      random_state=RANDOM_SEED)

        # Define Artificial Neural Network parameters
        clf = ann.MLPRegressor(hidden_layer_sizes=HIDDEN_LAYERS,
                               activation=ACTIVATION_FCT,
                               solver=SOLVER,
                               alpha=LEARNING_RATE,
                               early_stopping=EARLY_STOPPING,
                               max_iter=MAX_ITER,
                               validation_fraction=VALID_DATA_FROM_TRAIN)

        # Train and Evaluate the ANN
        clf.fit(trainX, trainY)
        annPredY = clf.predict(testX)
        mse_results.append(metrics.mean_squared_error(testY, annPredY))
        r2_results.append(metrics.r2_score(testY, annPredY))
        print(i)


    if OUTPUT_FILES:
        mseMean = np.mean(mse_results)
        mseMin = np.min(mse_results)
        mseMax = np.max(mse_results)
        print("\n\rANN: MSE = %f" % mseMean)
        df_mse = pd.DataFrame({'MSE':mse_results, 'Mean':"", 'Max':"", 'Min':""})
        df_mse.loc[0, 'Mean'] = mseMean
        df_mse.loc[0, 'Max'] = mseMax
        df_mse.loc[0, 'Min'] = mseMin
        df_mse.to_csv('ANN_MSE_Results.csv')

        r2Mean = np.mean(r2_results)
        print("\n\rANN: AUROC = %f" % r2Mean)
        r2Mean = np.mean(r2_results)
        r2Min = np.min(r2_results)
        r2Max = np.max(r2_results)
        print("\n\rANN: MSE = %f" % mseMean)
        df_mse = pd.DataFrame(
            {'MSE': r2_results, 'Mean': "", 'Max': "", 'Min': ""})
        df_mse.loc[0, 'Mean'] = r2Mean
        df_mse.loc[0, 'Max'] = r2Max
        df_mse.loc[0, 'Min'] = r2Min
        df_mse.to_csv('ANN_r2_Results.csv')

        newLabel = 'predict_{}'.format(PREDICT_FUTURE_DAY)
        df[newLabel] = clf.predict(X)
```

```
        hf.plot_5yr(df, IDName, TARGET_NAME, newLabel)
        hf.plot_3month(df, IDName, TARGET_NAME, newLabel)

        df_income = hf.calcIncome(df, TARGET_NAME, INVESTMENT,
                                  PREDICT_FUTURE_DAY,
                                  INCOME_TOLERANCE)
        df_income.to_excel(OUTPUT_INCOME)


# Used in the automated .yaml files in the the 'Data Handling' section
def SimpleANNModel(
    trainTestSplit: tuple,
    layers: list=HIDDEN_LAYERS,
    activation: str=ACTIVATION_FCT,
    solver: str=SOLVER,
    alpha: float=LEARNING_RATE,
    earlyStopping: bool=EARLY_STOPPING,
    maxIters: int=MAX_ITER,
    validationFraction=VALID_DATA_FROM_TRAIN
):
    ''' ANN Model wrapper that takes train-test split data and returns a trained model
        See SciKitLearn docs for more information
    :param trainTestSplit: Tuple of pd.DataFrames (trainX, testX, trainY, testY)
    :param layers: List of node depths of length 'hiddenLayerCount' (default:
ann.HIDDEN_LAYERS)
    :param activation: String name of the ANN activation function (default:
ann.ACTIVATION_FCT)
    :param solver: String name of the ANN solver function (default: ann.SOLVER)
    :param aplha: Float learning rate for the model (default: ann.LEARNING_RATE)
    :param earlyStopping: Boolean flag for early stop of learning (default:
ann.EARLY_STOPPING)
    :param maxIters: Integer stop limit for learning iterations (default: ann.MAX_ITER)
    :param validationFraction: Float fraction of training points to validate
with(default: ann.VALID_DATA_FROM_TRAIN)
    :return sklean.Model: Trained ANN model object
    '''
    # Define Artificial Neural Network parameters
    trainX, testX, trainY, testY = trainTestSplit
    model = ann.MLPRegressor(hidden_layer_sizes=layers,
                             activation=activation,
                             solver=solver,
                             alpha=alpha,
                             early_stopping=earlyStopping,
                             max_iter=maxIters,
                             validation_fraction=validationFraction)

    # Train and Evaluate the ANN
    model.fit(trainX.to_numpy(), trainY.to_numpy())
    annPredY = model.predict(testX)
    print(f'{__name__} MSE = {metrics.mean_squared_error(testY, annPredY)}')
    return model

### Main Processing
#######################################

# load data and add columns to expand data as necessary.
if __name__ == "__main__":

    if False:
        df_raw = pd.read_csv(INPUT_QUALCOMM_FINAL)
```

```
        # make changes/additions to the loaded base stock data.
        df_edit = prepData(df_raw)

        if OUTPUT_FILES:
            df_edit.to_csv("postDataPrep-ModelDataUsed-Preprocessing.csv")
    else:
        df_edit = pd.read_csv(INPUT_FINAL)


    # Do model evaluation.
    doANN(df_edit)
```

```
#############################################
#RANDOM FOREST MODE by James E
#############################################
Python Code for RandomForestModel
from operator import mod
from sklearn.ensemble import RandomForestRegressor
from sklearn.datasets import make_classification
from sklearn import metrics
import models.helperFunctions as hf
#import helperFunctions as hf

from sklearn.model_selection import train_test_split


### Set Constants
######################################


oneHundredCalcs = 100
nEstimatorsValue = 100
criterionToUse = 'squared_error'
maxDepthToUse = 10
randomStateToUse = 100


#code based on ANN model to ensure it works with yaml
def SimpleEnsembleModel(
    X, Y,
    #trainTestSplit: tuple,
    nEstimators: int = nEstimatorsValue,
    criterionInUse: str = criterionToUse,
    maxDepthInUse: int = maxDepthToUse,
    randomStateInUse: int = randomStateToUse
):
    '''
    Ensemble model, is RandomForestRegressor, code and documentation found at
    https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html

    class sklearn.ensemble.RandomForestRegressor(n_estimators=100,
    *,
    criterion='squared_error',
    max_depth=None,
    min_samples_split=2,
    min_samples_leaf=1,
    min_weight_fraction_leaf=0.0,
    max_features='auto',
    max_leaf_nodes=None,
    min_impurity_decrease=0.0,
    bootstrap=True,
    oob_score=False,
    n_jobs=None,
    random_state=None,
    verbose=0,
    warm_start=False,
    ccp_alpha=0.0,
    max_samples=None)

    :param trainX: pd.DataFrame containing training predictors
    :param testX: pd.DataFrame containing test predictors
    :param trainY: pd.DataFrame containing training target(s)
    :param testY: pd.DataFrame containing test target(s)
    '''
```

```
    #trainX, testX, trainY, testY = trainTestSplit
    # Define RandomForestRegressor parameters
    model = RandomForestRegressor(criterion=criterionInUse, random_state=100)

    mseOneHundred = []
    rtwoOneHundred = []
    lowestMSEModel = 99999
    highestMSEmodel = -1
    lowestR2Model = 99999
    highestR2model = -1
    lowestModel = model
    # Train and Evaluate the Ensamble
    for x in range(oneHundredCalcs):
        trainX, testX, trainY, testY = train_test_split(X, Y, test_size=0.30, random_state= x+100)
        model.fit(trainX.to_numpy(), trainY.to_numpy())
        ensemblePredY = model.predict(testX)
        mseOneHundred.append(metrics.mean_squared_error(testY, ensemblePredY))
        if(lowestMSEModel > metrics.mean_squared_error(testY, ensemblePredY)):
            lowestMSEModel = metrics.mean_squared_error(testY, ensemblePredY)
            lowestModel = model
        if(highestMSEmodel < metrics.mean_squared_error(testY, ensemblePredY)):
            highestMSEmodel = metrics.mean_squared_error(testY, ensemblePredY)

        if(lowestR2Model > metrics.r2_score(testY, ensemblePredY)):
            lowestR2Model = metrics.r2_score(testY, ensemblePredY)

        if(highestR2model <metrics.r2_score(testY, ensemblePredY)):
             highestR2model = metrics.r2_score(testY, ensemblePredY)

    print(f'{__name__} MSE AVERAGE = {FindAverage(mseOneHundred)}')
    print(f'{__name__} MSE MIN = {lowestMSEModel}')
    print(f'{__name__} MSE MAX = {highestMSEmodel}')

    print(f'{__name__} R2 AVERAGE = {FindAverage(rtwoOneHundred)}')
    print(f'{__name__} R2 MIN = {lowestR2Model}')
    print(f'{__name__} R2 MAX = {highestR2model}')
    return lowestModel

def FindAverage(list):
    return sum(list)/len(list)
```

Helper Functions

```
'''
helperfunctions.py
@author:    John Smutny
@team:      James Ensminger, Ben Johnson, Anagha Mudki, John Smutny
@info:      Various functions used to standardize the input of datasets,
            plot results, and calculate the income from various ML Stock
            predicting models.

            Used as a part of the QualComm Group Project 2 Stock Predictor.
'''

import pandas as pd
from matplotlib import pyplot as plt


### Helper Functions
########################################
'''
@brief Function to expand a single data entry by x columns to include
       previous data entry values in time.

@input  df      Pandas DataFrame data to be extrapolated.
@input  numPrevData  The number of earlier entries that will be appended to
                       the last entry in the dataframe.
@input labels   The specific labels that are going to be extrapolated
@input ASCENDING Whether the 'Dates' used are ascending or descending.
@return df      Pandas DataFrame with new data columns to represent original df
                labels for a previous day. NOTE: The numPrevData first
                entries in the df DataFame are deleted

'''


def appendPastData(df: pd.DataFrame, numPrevDays, labels, ASCENDING) -> \
       pd.DataFrame:
   # Error check that inputted 'labels' are all in df input
   notInDF = False
   for label in labels:
       if list(df.columns.values).count(label) == 0:
           notInDF = True
           print("ERROR: Label not in df.")

   if notInDF:
       print("Do not append any df columns with previous day's data. Some of "
             "the labels in the of inputted list does not exist in DataFrame "
             "df.")

   # Execute the Function's purpose.
   else:
       for label in labels:

           # Create columns to extrapolate data too
           for i in range(numPrevDays):
               # Create the new columns for each desired day
               addedColName = "Prev{}_{}".format(i + 1, label)
               zeros = [0] * len(df.index)
               df[addedColName] = zeros
```

```
                ########################################

                # Add previous day's data to new columns
                # Isolate one column at a time.
                for entry in range(len(df.index)):
                    if ASCENDING:
                        if entry >= numPrevDays:
                            df.loc[entry, addedColName] = \
                                df.loc[entry - i - 1, label]
                    else:
                        if entry < (len(df.index) - numPrevDays):
                            df.loc[entry, addedColName] = \
                                df.loc[entry + i + 1, label]

        # Delete the first x number of entries to prevent an indexing exception.
        if numPrevDays > 0:
            print("::appendPastData - Deleted {} yearlest dates to avoid "
                  "segFaults.".format(numPrevDays))
            if ASCENDING:
                df = df.drop(range(numPrevDays), axis=0)
            else:
                df = df.drop(range(len(df.index) - numPrevDays, len(df.index)),
                             axis=0)

            df = df.reset_index(drop=True)

    return df


'''
@brief Function to remove any '$' characters from a pandas dataframe column.

@input  df      Pandas DataFrame data to be reviewed.
@input labels   The specific labels that are going to be extrapolated
@return df      Pandas DataFrame with replaced values.
'''


def removeDollarSign(df: pd.DataFrame, labels) -> pd.DataFrame:
    for label in labels:
        df[label] = df[label].str.replace('$', '', regex=True)

    return df


'''
@brief  Universal function to take a dataset's DATE column and reformat it to
       a consistent style based on the datetime python object.
       Then sort the data based on the desired order.
       Style = yyyy-mm-dd
@input df DataFrame of the full data to be reformatted.
@input ASCENDING Whether the 'Dates' used are ascending or descending.
@output a dataframe of UNCHANGED data, only re-formatted.
'''


def reformatDailyDates(df: pd.DataFrame, ASCENDING) -> pd.DataFrame:
    df['Date'] = pd.to_datetime(df['Date'])
    df['Date'] = df['Date'].dt.date
```

```
    df = df.sort_values(by='Date', ascending=ASCENDING, ignore_index=True)
    return df


'''
@brief Function that will generate the target variable of
       'stock price x days in the future'.
       NOTE: This function will REMOVE data from the dataset to prevent
       exceptions or predicting the future.
@input TARGET label name of the target variable you are trying to model.
@input FUTURE_DAY How many days in the future are you looking at stock prices.
@input ASCENDING Whether the 'Dates' used are ascending or descending.
@output New data frame with the actual stock price after x days.
'''


def addTarget(df: pd.DataFrame, TARGET, FUTURE_DAY, ASCENDING) -> pd.DataFrame:
    # Add new column for the target.
    df[TARGET] = ""

    print("::addTarget - {} newest days will be dropped to predict {} days in "
          "the future ".format(FUTURE_DAY, FUTURE_DAY))

    listOfDropEntries = []

    for x in range(len(df['Date'])):
        # (earliest first)
        if ASCENDING:
            if x < len(df['Date']) - FUTURE_DAY:
                df.loc[x, TARGET] = df.loc[x + FUTURE_DAY,
                                           "Close/Last"]
            else:
                listOfDropEntries.append(x)

        # Descending (most recent first)
        else:
            if x > FUTURE_DAY:
                df.loc[x, TARGET] = df.loc[x - FUTURE_DAY,
                                           "Close/Last"]
            else:
                listOfDropEntries.append(x)

    # Drop indices to prevent segfault and are out of range of prediction.
    df = df.drop(index=listOfDropEntries, axis=0)
    df = df.reset_index(drop=True)
    return df


'''
@brief Function to calculate how much you would make based on a minimum gain
       predicted by the given model.
@input df The dataframe of data inputs used to train your model.
           MUST INCLUDE THE PREDICTION OF THE MODEL
@input TARGET The target variable in the input 'df' that the model trained on.
@input INVESTMENT How much are you investing each time the model tells you.
@input FUTURE_DAYS How many days in the future will you sell your stock.
@input TOL the tolerance of when you should invest to get a minimum return.
       Ex: TOL = 1.05 means that the model must predict 5% profit to invest.
@output Dataframe record of the investments made and the conditions on that day.
```

```
'''


def calcIncome(df: pd.DataFrame, TARGET, INVESTMENT, FUTURE_DAYS, TOL) -> \
        pd.DataFrame:
    print("WARN: You must include the model predictions for 'Close Price 28 "
          "Days Later' for this fct to work. Please insert the following code "
          "before calling this function:\n"
          "\t\tdf['predict_28'] = clf.predict(X)'")

    df_invest = pd.DataFrame(columns=['Date', 'quantity', 'close',
                                      'sell_price',
                                      'model_price', 'predIncome',
                                      'actualIncome'])

    for i in range(len(df['Date'])):
        close = float(df.loc[i, 'Close/Last'])
        modelClose = float(df.loc[i, 'predict_{}'.format(FUTURE_DAYS)])
        modelGain = modelClose / close

        if modelGain > TOL:
            actualClose = float(df.loc[i, TARGET])

            quantity = INVESTMENT / close
            predIncome = (modelClose - close) * quantity
            actualIncome = (actualClose - close) * quantity

            df_invest.loc[i, 'Date'] = df.loc[i, 'Date']
            df_invest.loc[i, 'quantity'] = quantity
            df_invest.loc[i, 'close'] = close
            df_invest.loc[i, 'sell_close'] = actualClose
            df_invest.loc[i, 'model_close'] = modelClose
            df_invest.loc[i, 'predIncome'] = predIncome
            df_invest.loc[i, 'actualIncome'] = actualIncome

    print("TOTAL INCOME FROM {} INVESTMENTS (PREDICT/ACTUAL): "
          "${:.2f}/${:.2f}".format(len(df_invest),
                                   df_invest['predIncome'].sum(),
                                   df_invest['actualIncome'].sum()))

    return df_invest


def plot_5yr(df: pd.DataFrame, labelx, labelActual, labelPredicted):
    df = pd.DataFrame({'Date': df[labelx],
                       'Close_28': df[labelActual].astype(float),
                       #'Todays_Close': df['Close/Last'].astype(float),
                       'Predict_28': df[labelPredicted].astype(float)
                       })

    df.plot(x='Date', y=[
                         'Close_28',
                         #'Todays_Close',
                         'Predict_28'
                        ],
            kind="line", figsize=(18, 8))
    plt.grid(which='major', linestyle='-', linewidth='0.5', color='green')
    plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
    plt.xlabel("Date")
    plt.ylabel("Close Price ($)")
```

```
    plt.title("QualComm Stock Price over the Last 5 Years")
    # plt.show()

    plt.savefig("StockPrice-5Year.jpeg")


def plot_3month(df: pd.DataFrame, labelx, labelActual, labelPredicted):
    df = pd.DataFrame(
        {'Date': df[labelx],
         #'Close': df['Close/Last'].astype(float),
         'Close_28': df[labelActual].astype(float),
         'Predict_28': df[labelPredicted].astype(float)})

    # Slice data to plot only the last 3 months
    df = df.tail(28 * 3)

    df.plot(x='Date', y=['Close_28',
                         #'Close',
                         'Predict_28'],
            kind="line", figsize=(18, 8))
    plt.grid(which='major', linestyle='-', linewidth='0.5', color='green')
    plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
    plt.xlabel("Date")
    plt.ylabel("Close Price ($)")
    plt.title("QualComm Stock Price over the Last 3 Months")
    # plt.show()

    plt.savefig("StockPrice-3Month.jpeg")


################################################################################
################################################################################
################################################################################
# Functions to add datasets
################################################################################
################################################################################
################################################################################

'''
@brief  Function to add daily bond yields of the desired year to an existing
       dataset for modeling.
       Covers Bond yields of 3-months, 2-years, and 10-years.
@input  BOND label given by the Treasury department stating the bond data.
               See the datafile used for the label.
@input FILE Input file of bond data.
@input ASCENDING Whether the 'Dates' used are ascending or descending.
@output Dataframe now including x yr daily bond yields.
'''


def addBondPrice(df: pd.DataFrame, BOND, FILE, ASCENDING) -> pd.DataFrame:
    df_bond = pd.read_csv(FILE)
    df_bond = df_bond.rename(columns={'DATE': 'Date'})

    # Ensure that the information is the correct order
    df_bond = reformatDailyDates(df_bond, ASCENDING)

    # Clean data

    valuesChanged = len(df_bond.index[df_bond[BOND] == '.'].tolist())
```

```
    print("::addBondPrice - {} values were changed to clean data.".format(
        valuesChanged))

    for x in df_bond.index[df_bond[BOND] == '.'].tolist():
        if x != 0:
            df_bond.loc[x, BOND] = df_bond.loc[x - 1, BOND]
        else:
            df_bond.loc[x, BOND] = df_bond.loc[x + 1, BOND]

    # Add ready values to main dataframe for models
    df = pd.merge(df, df_bond, on='Date', how='left', validate='one_to_one')

    return df


'''
@brief Use for data from fred.stlouisfed.org
       General function to add an economic indicator to the dataframe of data.
@input FILE Input file of data.
@input ASCENDING Whether the 'Dates' used are ascending or descending.
@output Dataframe now including x yr price of that indicator at the close.
'''


def addFedData(df: pd.DataFrame, SYM, FILE, ASCENDING) -> pd.DataFrame:
    df_Fed = pd.read_csv(FILE)
    df_Fed = df_Fed.rename(columns={'DATE': 'Date'})

    # Ensure that the information is the correct order
    df_Fed = reformatDailyDates(df_Fed, ASCENDING)

    # Clean data

    valuesChanged = len(df_Fed.index[df_Fed[SYM] == '.'].tolist())
    print("::addBondPrice - {} values were changed to clean data.".format(
        valuesChanged))

    for x in df_Fed.index[df_Fed[SYM] == '.'].tolist():
        if x != 0:
            df_Fed.loc[x, SYM] = df_Fed.loc[x - 1, SYM]
        else:
            df_Fed.loc[x, SYM] = df_Fed.loc[x + 1, SYM]

    # Add ready values to main dataframe for models
    df = pd.merge(df, df_Fed, on='Date', how='left', validate='one_to_one')

    return df


'''
@brief Use for data from NASDAQ.com
       General function to add a stock ticker to the dataframe of data.
@input FILE Input file of data.
@input ASCENDING Whether the 'Dates' used are ascending or descending.
@output Dataframe now including x yr stock price at the close.
'''

def addStockClosePrice(df: pd.DataFrame, SYM, FILE, ASCENDING) -> pd.DataFrame:
    EXTRACTED_FEATURE = 'Close/Last'
    ADDED_FEATURE = 'Close_{}'.format(SYM)
```

```
    df_stock = pd.read_csv(FILE)
    df_stock = df_stock[['Date', EXTRACTED_FEATURE]]
    df_stock = df_stock.rename(columns={EXTRACTED_FEATURE:ADDED_FEATURE})
    df_stock = removeDollarSign(df_stock, [ADDED_FEATURE])


    # Ensure that the information is the correct order
    df_stock = reformatDailyDates(df_stock, ASCENDING)

    # Add ready values to main dataframe for models
    df = pd.merge(df, df_stock, on='Date', how='left', validate='one_to_one')

    return df


'''
@brief
@input FILE Input file of bond data.
@input ASCENDING Whether the 'Dates' used are ascending or descending.
@output Dataframe now including x yr daily bond yields.
'''


def addSimFin(df: pd.DataFrame, FILE, ASCENDING) -> pd.DataFrame:
    df_QualComm = pd.read_csv(FILE)
    df_QualComm = df_QualComm.rename(columns={'DATE': 'Date'})

    # Ensure that the information is the correct order
    df_QualComm = reformatDailyDates(df_QualComm, ASCENDING)

    return df
```

REFERENCES

[1] "Qualcomm CDMA Technologies - Investor Relations." QUALCOMM Incorporated, Qualcomm Technologies, Inc, https://investor.qualcomm.com/segments/qct. Section 'PRODUCTION & MANUFACTURING - Fabless Model'. May 3, 2022.

[2] "Qualcomm Competitors." Comparably, Comparably Inc, https://www.comparably.com/companies/qualcomm-incorporated/competitors. May 3, 2022

[3] Nasdaq. 2022. QCOM Historical Data [Dataset]. https://www.nasdaq.com/market-activity/stocks/qcom/historical

[4] Flassbeck, Thomas. 2022. QUALCOMM INC/DE Financial statements (Original) [Dataset]. SimFin. https://simfin.com/data/companies/85758

[5] Division of Information and Marketing Services, CPI for All Urban Consumers [CUUR0000SA0L1E], retrieved from the U.S. Bureau of Labor Statistics. https://data.bls.gov/timeseries/CUUR0000SA0L1E?output_view=pct_12mths (accessed Apr. 11, 2022)

[6] Board of Governors of the Federal Reserve System (US), Market Yield on U.S. Treasury Securities at 3-Month Constant Maturity [DGS3MO], retrieved from FRED, Federal Reserve Bank of St. Louis; https://fred.stlouisfed.org/series/DGS3MO, April 29, 2022.

[7] Board of Governors of the Federal Reserve System (US), Market Yield on U.S. Treasury Securities at 3-Month Constant Maturity [DGS3MO], retrieved from FRED, Federal Reserve Bank of St. Louis; https://fred.stlouisfed.org/series/DGS3MO, April 29, 2022.

[8] Board of Governors of the Federal Reserve System (US), Market Yield on U.S. Treasury Securities at 2-Year Constant Maturity [DGS2], retrieved from FRED, Federal Reserve Bank of St. Louis; https://fred.stlouisfed.org/series/DGS2, April 29, 2022.

[9] Board of Governors of the Federal Reserve System (US), Market Yield on U.S. Treasury Securities at 10-Year Constant Maturity [DGS10], retrieved from FRED, Federal Reserve Bank of St. Louis; https://fred.stlouisfed.org/series/DGS10, April 29, 2022.

[10] Board of Governors of the Federal Reserve System (US), Nominal Broad U.S. Dollar Index [DTWEXBGS], retrieved from FRED, Federal Reserve Bank of St. Louis; https://fred.stlouisfed.org/series/DTWEXBGS, April 29, 2022.

[11] International Monetary Fund, Global price of Brent Crude [POILBREUSDM], retrieved from FRED, Federal Reserve Bank of St. Louis; https://fred.stlouisfed.org/series/POILBREUSDM, April 29, 2022.

[12] Coinbase, Coinbase Bitcoin [CBBTCUSD], retrieved from FRED, Federal Reserve Bank of St. Louis; https://fred.stlouisfed.org/series/CBBTCUSD, April 29, 2022.

[13] Nasdaq. 2022. AAPL Historical Data [Dataset]. https://www.nasdaq.com/market-activity/stocks/aapl/historical. April 29, 2022.

[14] Nasdaq. 2022. GOOGL Historical Data [Dataset]. https://www.nasdaq.com/market-activity/stocks/googl/historical. April 29, 2022.

[15] Nasdaq. 2022. ERIXF Historical Data [Dataset]. https://www.nasdaq.com/market-activity/stocks/erixf/historical. April 29, 2022.

[16] Nasdaq. 2022. NXPI Historical Data [Dataset]. https://www.nasdaq.com/market-activity/stocks/nxpi/historical. April 29, 2022.

[17] Nasdaq. 2022. SSNLF Historical Data [Dataset]. https://www.nasdaq.com/market-activity/stocks/ssnlf/historical. April 29, 2022.

[18] Nasdaq. 2022. VZ Historical Data [Dataset]. https://www.nasdaq.com/market-activity/stocks/vz/historical. April 29, 2022.

[19] Nasdaq. 2022. TMUS Historical Data [Dataset]. https://www.nasdaq.com/market-activity/stocks/tmus/historical. April 29, 2022.

[20] N. N. Ghosalkar and S. N. Dhage, "Real Estate Value Prediction Using Linear Regression," 2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA), 2018, pp. 1-5, doi: 10.1109/ICCUBEA.2018.8697639.

[21] Raschka, Sebastian. "Logistic Regression Schematic". mlxtend. 03 May 2022

[22] John D. Kelleher, Brian Mac Namee, Aoife D'Arcy, Fundamentals of Machine Learning for Predictive Data Analytics_ Algorithms, Worked Examples, and Case Studies, 2nd ed. Cambridge, Massachusetts, (London, England): The MIT Press, 2015