PROJECT REPORT – GROUP 8

STOCK PRICE PREDICTION

Technical Report for Machine Learning Project

Under the guidance of

Raja Loganantharaj, Ph.D.

Submitted By

Teja Akula - 1862814

Vishnu Sai Inakollu - 2311922

Tanmai Veerapaneni - 2315355

Rohit Bejjam - 2138034

EDS 6342 INTRODUCTION TO MACHINE LEARNING

Department of Engineering Data Science

Cullen College of Engineering

University of Houston

ABSTRACT

Stock price forecasting is also known as time series forecasting. It is a process of predicting future values based on the historical data. The GE stock dataset shows information on the opening, volume, high, low, and other features on a day-to-day basis. It extends for decades and displays the trend and movement of these features. The motivation and objective for this study is to be able to capture some form of predictiveness in such an unpredictable environment, such as stock prices. The close feature is particularly important in this task and is thus the 'y' value. The steps taken towards this are modeling ARIMA, SARIMA, and PROPHET statistical models. Along with this, the team made models on LSTM (Long-short-term memory) and Random Forest. The findings were that Random Forest performed the best on the test data and in prediction, while the LSTM performed the best in predictions. This insight is based on the graphical representations of said predictions, where the LSTM offered more stable and believable visualizations. The following document will expound on these.

Table of Contents

INTRODUCTION

Observability basically refers to tracking or evaluating a system's condition based on its external outputs. Data observability is a DataOps method that aims to improve data quality and decrease data downtime by taking into account the five main pillars of data health namely freshness, distribution, volume, schema, and lineage. The adoption of a data observability solution enables data teams to optimize the computation, capacity, resources, cost, and performance of a data infrastructure while maintaining a high quality of their data.

Stock price prediction is handling time series data. Time series data aids businesses with data observability, and has recently become one of the most important commodities in technology. A time series is a collection of data points that are arranged and ordered according to timestamps. A collection of data points recorded at regular intervals is called a time series. These informational pieces may be recorded on an hourly, daily, weekly, monthly, quarterly, or annual basis. Time-Series Using a statistical model and historical data, forecasting entails predicting future values for a time series. It describes how to forecast how a time series will develop in the future. Time series can be achieved for both static as well as dynamic data which can be analyzed using statistical models such as ARIMA and SARIMA or Machine learning models such as LSTMs and random forest.

LITERATURE REVIEW

Aras et al. [1] provide an excellent literature review and comparison analysis on consumer spending forecasts using individual and combo methodologies. According to the facts presented in this article, combination tactics outperformed individual methods. A comparison of these approaches and the company's present system was also performed. Several additional noteworthy findings are addressed in the literature review part of this work, as follows. Ansuj et al. [2] examined sales data from 1979 to 1989 using the ARIMA (Autoregressive Integrated Moving Averages) model with inputs, as well as the ANN approach, used. Forecasts from the model were more accurate than those from the ARIMA model. Alon et al. [3] compared traditional techniques and multiple regressions for aggregated retail trade under stable economic conditions with winter's gradient descent. Women's apparel sales were developed by Frank et al. [4] using an ANN model (which yielded the best results in terms of R2 evaluation statistics), Winter's three-parameter model, and a single seasonal exponential smoothing. The best results were achieved by ANN models when attempting to predict future sales. The two-stage hybrid method used by Aburto and Weber [5] to create a restocking system for a Chilean supermarket surpassed the projections of both the ANN and ARIMA models. Hybrid methods resulted in lesser inventory and fewer unsuccessful sales attempts. In contrast to Ramos et al. [6], the findings showed no difference in projecting sales of women's footwear goods between state space models and ARIMA models with automated algorithms. Fabianová et al. [7] examined refrigerator sales at a retail shop. The results were improved by employing the identification of variables by Monte Carlo simulation and sensitivity analysis. Deep learning models can be fine-tuned and improved with more data and they can be trained to detect different types of anomalies, including point anomalies and collective anomalies, which improves the overall performance of the model. For instance, in a study by Malhotra et al. [8] the authors found that an LSTM network outperformed traditional time series models such as ARIMA.

BACKGROUND INFORMATION

A time series is a collection of data points that have been collected throughout time, time-indexed or time-graphed. Time series are used in practically all areas of applied research and engineering, including astronomy, econometrics, finance, meteorology, signal processing, pattern recognition, control engineering, and astrophysics. In order to determine their significant properties, time series analysis employs methods for studying time series data. Time series analysis is useful for continuous real-valued-data, discrete numeric-data, and discrete symbolic-data. Time series forecasting is a technique used in models to forecast future values based on observed values in the past. This section describes some of the concepts that are useful to know about time series modeling, as well as information about time series models such as ARIMA, SARIMA, and Prophet.

An Autoregressive_Integrated_Moving_Average (ARIMA) model is a time series data modeling and forecasting statistical approach The AR model is based on the idea that a time series' present value is a linear mixture of its previous values. The

MA model, on the other hand, is based on the premise that a time series' present value is a linear mixture of previous mistakes or shocks. ARIMA uses historical data to forecast future values (autoregressive, moving average). In the case of Seasonal_Autoregressive_Integrated_Moving_Average (SARIMA), seasonality trends are taken into account along with previous values, and because it incorporates seasonality as a parameter, it is far more effective than the prior for forecasting complex data spaces with cycles. High-quality forecasting has always faced tremendous challenges. Thus, there was a critical lack of analysts with the ability to provide reliable projections that may impact corporate choices. In order to address this demand/supply mismatch and make scaling forecasting much simpler, the Facebook Core Data Science team developed Prophet, a forecasting program for Python and R. Hyperparameter tuning is a crucial procedure in training machine learning models. It may be a challenging process due to the various parameters that must be adjusted, the lengthy training period, and several folds to avoid information leaks.
In this work, LSTM and Random Forest were run and the performance was recorded separately for comparison and analysis. The mean score error and mean absolute error were used. One important advantage of using the MAE to evaluate a time series forecasting model is that it is less sensitive to the scale of the data than other measures, such as mean squared error (MSE). This means that if the scale of the data changes, the MAE will not change as much as the MSE, even if the model's accuracy has not changed. This can be important when comparing the performance of different models, or when comparing the performance of a single model over different time periods.
Some background on the project itself, the main problem being addressed is predicting future closing prices of GE stock. It's a regression problem. The close stock is essential in understanding the market evaluation and movement. There are many different models that can be applied to do so, the team aims to compare some of the models to see which fits better. A brief outline of the document: EDA, Data preprocessing, model building, results, comparison, discussion, and references.

EDA

The shape of this dataset is (15573, 7) with float64 and int64 data types. Its columns are date, open, high, low, close, adj close, and volume. It has no missing values. The selected output value is the closing evaluation. This close value is chosen because it offers a good basis for the opening for the next day, which further contributes to the closing of said day. Out of all the other features, this one is used the most to assess upcoming changes in stock. That being said, because this is stock data, the patterns and trends are very unpredictable. Exploration of the data is particularly important. To see the general direction, a plot of the output 'Close price' is shown below:

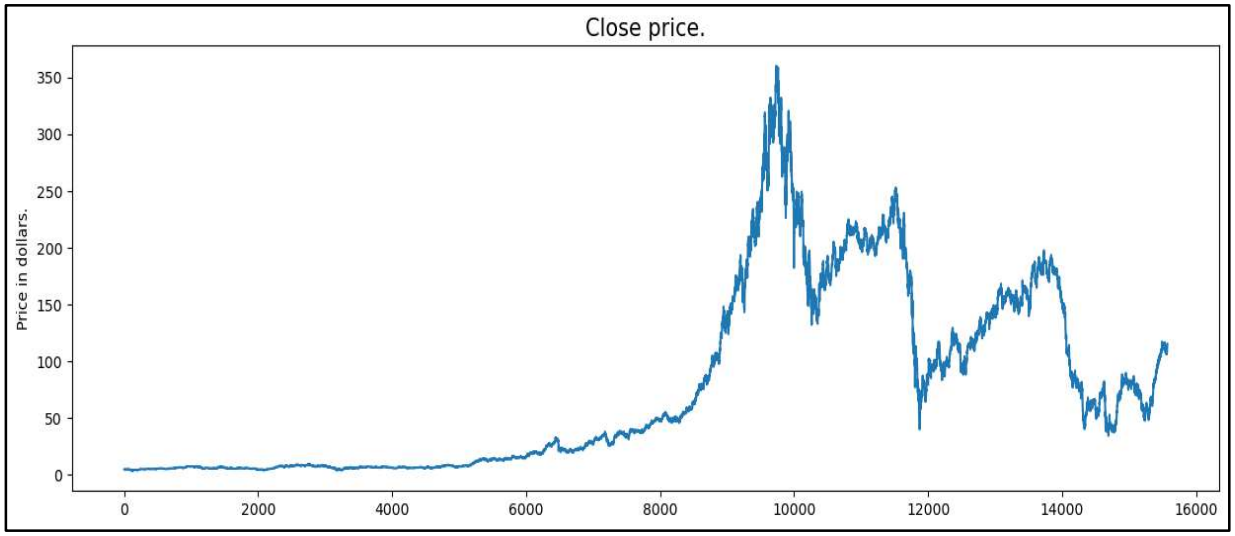

Fig – 1: 'Close price' plot

The team also plotted the distribution of the data against the density distribution to better understand the spread of data and to compare the different variable distributions. Most of them follow a similar distribution shape except for volume. In other

words, we can deduce similar central tendencies and region variations. The skew is alike. This can be a potentially good indicator that if the model predicts a value that widely differs from the inclusive cross-examination of the other variables, then something may be incorrect. Visually, this can be much simpler to identify.

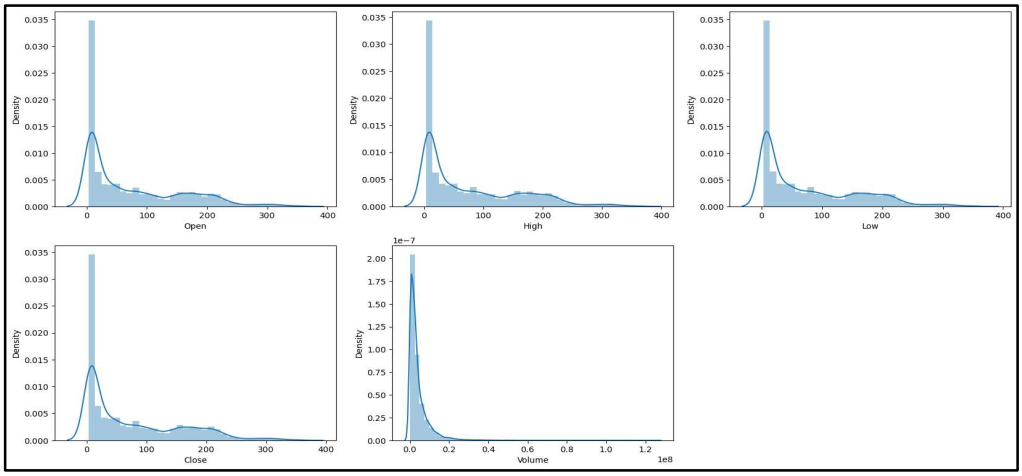

Fig – 2 Comparison of different variable distributions

This can be further tested and supported by plotting the stock prices for the overall period. As one can witness, the different columns' directionalities do in fact follow a near-parallel.

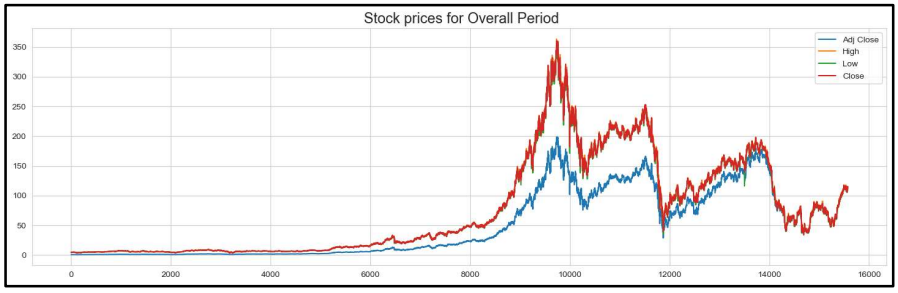

Fig – 3: Stock prices for an overall period

To check the correlation between each of these features, a correlation table was generated with the help of df.corr(). A visualized matrix was also created but it output rounded values. For better clarity and detail, here is both side by side:

	Open	High	Low	Close	Adj Close	Volume
Open	1.000000	0.999883	0.999865	0.999771	0.958082	0.244706
High	0.999883	1.000000	0.999806	0.999892	0.957589	0.247156
Low	0.999865	0.999806	1.000000	0.999874	0.958816	0.241092
Close	0.999771	0.999892	0.999874	1.000000	0.958225	0.243921
Adj Close	0.958082	0.957589	0.958816	0.958225	1.000000	0.351237
Volume	0.244706	0.247156	0.241092	0.243921	0.351237	1.000000

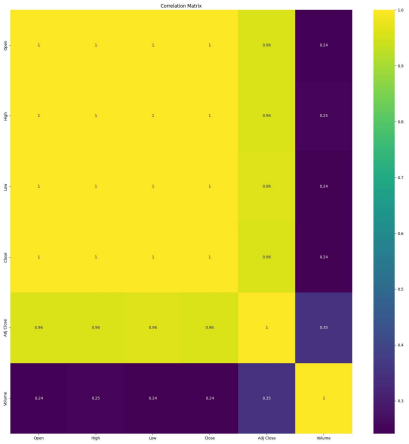

Fig – 4: Correlation data

Besides the volume feature, all the other features have a very high correlation with each other. This makes sense for stock data. For example, the opening stock price sets the base for where the stock is going to go at the end of the day. Likewise, the stock value at the end of the day influences the value for the opening of the next day.

DATA PREPROCESSING

The data chosen was already very clean it doesn't contain missing values and dates were in sequence. Reiterating from the EDA, there are no missing values. Box plots were initially created to detect outliers, as shown below.

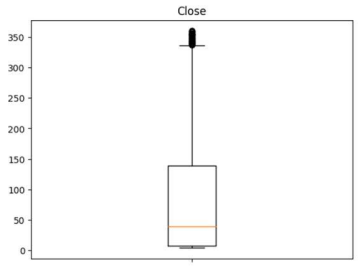

Fig – 5: Box Plot to detect outliers.

However, deleting outliers in stock is quite tricky as there are naturally many inconsistencies in the trends and patterns. Thus, those preceding steps are not included in this section. Scaling was not required for this dataset. In fact, for stock, it can be more appropriate to keep the raw values. The adjusted close value is also here to cover this just in case.

Model Building: ARIMA, SARIMA, PROPHET, LSTM, Random Forest

ARIMA:

The ARIMA model is a linear combination of three components: the Autoregressive (AR), the Integrated (I), and the Moving Average (MA). Since its inception, the ARIMA model has been a frequently used tool in time series analysis.
The ARIMA model is a linear combination of three components:
Autoregressive (AR) component: This component records the connection between the time series' current value and its previous values. It is represented by the "p" parameter in the ARIMA model. Integrated (I) component: This component accounts for the impact of previous shocks on the time series' current value. In the ARIMA model, it is represented by the "d" parameter. Moving Average (MA) component: This component represents the link between the time series' current value and previous mistakes or shocks. In the ARIMA model, it is represented by the "q" parameter. The ARIMA model has the following general form: ARIMA (p,d,q). The values of the general form "p," "d," and "q" parameters are determined through a process called model selection, this entails fitting multiple combinations of these parameters to the data and picking the optimal combination. After selecting the right ARIMA model, it may be used to anticipate future values of the time series. The model's integrated (I) component accounts for the impact of prior shocks on the present value of the time series, it is represented by the "d" parameter in the ARIMA model, which represents the number of differences needed to make the time series stationary. The moving average (MA) component of the model captures the relationship b/w the current value of the time series and the past errors or shocks.

SARIMA:

Time series and forecasting are two major issues in statistics for which there are algorithms like ARIMA and SARIMA. In SARIMA seasonality trends are taken into account along with previous values, and because it incorporates seasonality as a parameter, it is far more effective than the prior for forecasting complex data spaces with cycles. Taking seasonality into account is crucial for accurate demand forecasting techniques. SARIMA supports univariate time series data with a seasonal

component specifically. This support includes an additional parameter for the seasonality period as well as three new hyperparameters to calculate the autoregression (AR), differencing (I), and moving average (MA) for the seasonal component of the series. For SARIMA models, there are two categories of elements that can be observed:
Trend Elements (p, q, and d): p: Order of the autoregression trend, d: Order of trend difference and q: Order of the moving average trend
Seasonal Elements (p, d, q, m).: **p:** Order of the seasonal autoregressive component, **d:** Order of the seasonal difference, **q:** Order of the seasonal moving average component and **m:** Time steps for a single seasonal period.
For hyperparameter tuning we have used the Grid search technique where the p value ranges from 0 - 2, the d value ranges from 1-2, q value ranges from 0 - 3 same p, q, and d values are chosen for seasonality and m value is chosen as 5. The best parameters are chosen based on the lowest AIC score. Based on the best parameters the MSE and MAE values are calculated.

FBProphet:

Facebook Core Data Science team developed Prophet, a forecasting program for Python and R. As of the year 2017, you may get the source code for Prophet. The purpose of Prophet is to "allow both professionals and laypeople to make high-quality forecasts that keep pace with demand." Prophet regularly outperforms other traditional forecasting approaches by producing accurate projections with minimal user input. It also enables the use of domain expertise via easy parameter interpretation. It is an additive model-based method for forecasting univariate (one variable) time series data that takes into account trends, seasonality, and vacations. Equation of the model is y(t)= g(t)+h(t)+s(t)+et - (4)
Where y(t) = Additive Regressive Model, g(t)= Trend Factor h(t)= Holiday Component, s(t)= Seasonality Component
et = Error Team

Seasonality component s(t) adds flexibility to the model by including components on a daily, weekly, monthly, and bi-monthly basis. The influence of seasons is another variable that may be adjusted. Optuna is a Bayesian optimization toolset for improving machine learning model hyperparameters. We used Optuna for prophet hyperparameter tuning.
Fig – 6 Taxonomy of Statistical Model.

```
class ANOM_DET

  __init__()

  class PreProcessing
      __init__()
      extractColumn()
      sample()
      prophet_preprocessing()
      testStationarity()
      differentiate()
      trainTestSplit()

  class Arima
      tuneHyperparameters()
      generatePredictions()

  class SARIMA
      tuneHyperparameters()
      generatePredictions()

  class prophet1
      __init__()
      train_validate()
      objective()
      regression_report()
      prophet_requirments()

  make_data()

  choose_model()

  driver()
```

For the statistical model, we created a class in the name of ANOM_DET containing all the subclasses required for the statistical models. __init__ function constructs the object of the ANOM_DET class and is responsible for instantiating objects of the pre-processing, arima, sarima, prophet, and anomaly-detection classes, which are member attributes of the anom_det class. this function pre-processes the input dataset to suit the needs of the models and make the data ready for further processing. It makes use of the object of the class PreProcessing to perform these tasks. CHOOSE_MODEL() This function chooses the best-fit model based on the performances, which are determined by evaluation metrics such as MAE, MAPE, and MSE. The driver () function is the main module that runs the entire system by invoking all modules in a logical sequence. Class PreProcessing handles all the pre-processing involved with the three models. EXTRACTCOLUMN() This function extracts the required column from the dataset and ensures that the formatting for date-time is done correctly. SAMPLE () This function samples the dataset as per the variable specified by the user, using the in-built resample function. PROPHET_PREPROCESSING() This function is curated specifically for the Prophet model as the functionalities differ slightly. It is the same as the extractColumn() function, with an additional step of renaming the Date-time column to ‘ds’, and the required output/target column to ‘y’. TESTSTATIONARITY (). This function, as the name suggests, tests whether the data is stationary or not. Testing is done using the well-known AD-Fuller Testing algorithm, from the stats models.tsa.stattools library. DIFFERENTIATE() Upon stationarity, this function helps remove it by differentiation. CLASS ARIMA This class contains the functionalities associated with the Arima model. TUNEHYPERPARAMETERS() The in-built auto_arima() function is used to tune hyperparameters for the Arima model, i.e. to obtain the most optimal sequence of p,d,q parameters for modeling the given dataset. GENERATE PREDICTIONS () This function makes the prediction based on the

optimal hyperparameters of the model that were obtained after tuning, by fitting the model. CLASS SARIMA This class contains the functionalities associated with the Sarima model. GENERATE PREDICTIONS () This function makes the prediction based on the optimal hyperparameters of the model that were obtained after tuning, by fitting the model. CLASS PROPHET1 This class has 4 functions __init__() constructs the object of the prophet1 class and is responsible for instantiating objects of this class, train_validate() function splits the data into train, test, and validation, objective() function contains all the Prophet parameters and this function is optimized and it returns the best parameters for the prophet model, regression_report() function contains the evaluation metrics used for the prophet and prophet_requirments() function is the main function for prophet renames the date column and the prediction column into 'ds' and 'y' since those are the only columns the Prophet will take and makes the predictions based on the best parameters our model has chosen.

Random Forest:

Random Forest is a supervised learning algorithm whose architecture utilizes several decision trees to classify or perform regression. In this case, it is used to output a model for regression. Essentially, the combined result of several different decision tree predictions is used to produce the random forest prediction. This is called an ensemble, using several models and leveraging them to obtain a better result. The errors should differ and be independent from model to model, however. To detail further, the process is initiated as the algorithm takes a random set of data points from the training set and builds a decision tree based on those points specifically. Those selected data points are put in a bootleg sample data set. Sometimes, the instances are repeated. Even the features that the tree uses get sampled, in the sense that random parameters will be considered when performing the tree, not all of them. This step is repeated until a specified number of trees are built. Average all those predictions together. As a side note, the process of bootstrapping and combining results is termed "Bagging." Different predictions considered and amalgamated make for a more stable form of predicting. Below is a conceptual visual of these concepts.

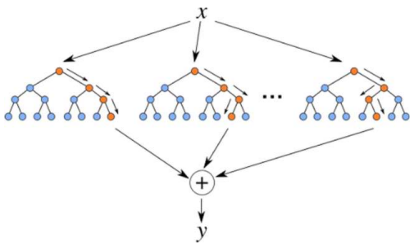

Fig –7 Diagram of random forest conceptual architecture [Citation 10]

Coming to the code, the team started by importing the necessary libraries. Some of these include but are not limited to StandardScaler, LinearRegression, and RandomForestRegressor. After performing the basic steps such as splitting the data into training and testing, scaling, etc. the following code was used:

```
param_grid = {
    'n_estimators': [100, 200, 300, 400, 500],
    'max_depth': [None, 10, 20, 30],

}

# Create a RandomForestRegressor
rf_model = RandomForestRegressor(random_state=42)

# Define mean squared error as the scoring metric
scorer = make_scorer(mean_squared_error, greater_is_better=False)

# Create GridSearchCV
grid_search = GridSearchCV(estimator=rf_model, param_grid=param_grid, scoring=scorer, cv=5)
grid_search.fit(X_train, y_train)

# Print the best parameters and best score
print("Best parameters: ", grid_search.best_params_)
print("Best MSE: ", -grid_search.best_score_)
```

Fig – 8: Hyperparameter tuning for random forest

The param grid provides a dictionary of the hyperparameter combinations n_estimators (number of trees) and max_depth (max depth of tree). These are the hyperparameters that are considered. Such hyperparameter optimization is more effiecient than trial and error with different ones. This ensures a greater reliability in the model. The tuning was used on the random rf_model. The team wanted to examine the mean squared error, wanting a low error of course. The grid search and fit are used for hyperparameter tuning. It examines each combination of parameters and the 'grid_search.best_params' and 'grid_search.best_score_' outputs the best working ones. It is important to note here, that the dataset did not come with prediction values, of course. The team had to create those. They were created with the

LSTM:

LSTM is a type of Recurrent Neural Network meant to avoid the exploding and vanishing gradient problems that may occur. This uses two separate paths, a long and a short one, to make its predictions. It uses sigmoid and tanh activation functions. These take the instance and make it fall within an interval of 0 and 1 or -1 and 1. The long-term memories don't have weight, but the short-term ones do. This is what helps avoid the previously mentioned gradient problem. These two different approaches interact with each other. The architecture is comprised of the forget gate, the input gate, the candidate memory, the output gate, the hidden state, the memory, and the tanh and sigmoid. The internal C-t basically allows the memory to flow, and this can be changed. It's almost like water valves. The forget value will pass old memory if open, and the new memory takes it in. Both are single layer neural networks. The output LSTM unit has an output that the new memory controls. The figure below depicts this architecture in a neat manner.

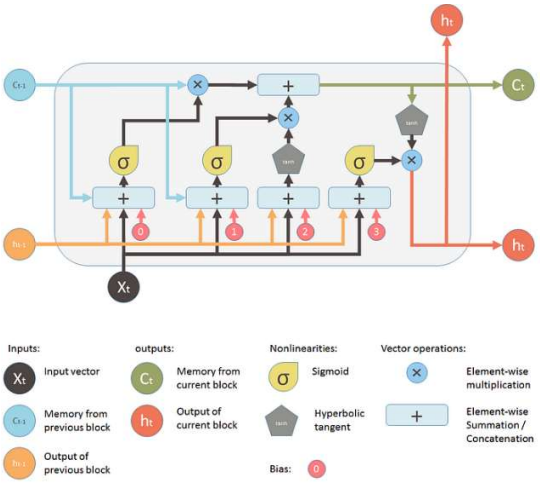

Fig – 9: Architecture of LSTM [9]

After cleaning, scaling, and setting up the test-train split and such, the team imported sklearn, Keras, and numpy.
As shown in the Random forest section, a similar form of choosing the hyperparameters was done. What primarily differed between param_grid was what was being incorporated. Here, the Param_grid considered neurons, dropouts, and dense units. The Gridsearch was also fit according to these and the neg_mean_squared_error. Again, the param_grid is the place to input the parameters that should be considered and at what value options.

```
# Function to create the LSTM model
def create_lstm_model(neurons=50, dropout_rate=0.3, dense_units=512):
    model = Sequential([
        LSTM(neurons, return_sequences=True, input_shape=(x_train.shape[1], 1)),
        LSTM(neurons, return_sequences=False),
        Dense(units=dense_units),
        Dropout(dropout_rate),
        Dense(units=256),
        Dropout(dropout_rate),
        Dense(units=64),
        Dense(1)
    ])
    model.compile(optimizer='adam', loss='mean_squared_error')
    return model

# Wrap Keras model so it can be used by scikit-learn
lstm_model = KerasRegressor(build_fn=create_lstm_model, epochs=25, batch_size=50, verbose=0)

# Define the hyperparameter grid
param_grid = {
    'neurons': [50, 100, 150],
    'dropout_rate': [0.3, 0.5],
    'dense_units': [512, 1024]
}

# Use time series cross-validation
tscv = TimeSeriesSplit(n_splits=5)

# Create and fit the grid search with progress and results printing
grid = GridSearchCV(estimator=lstm_model, param_grid=param_grid, scoring='neg_mean_squared_error', cv=tscv, verbo
grid_result = grid.fit(x_train, y_train)

# Print the best parameters
print("Best parameters: ", grid_result.best_params_)

# Print results for each set of hyperparameters
cv_results = grid_result.cv_results_
for mean_score, params in zip(cv_results["mean_test_score"], cv_results["params"]):
    print(f"Mean Test Score: {mean_score}, Parameters: {params}")
```

Fig – 10: Hyper parameter tuning for LSTM MODEL

The Gridsearch part does the actual optimization finding of these. The model itself was built with the following:

```
#Build the LSTM Model

model = Sequential()
model.add(LSTM(50, return_sequences=True, input_shape = (x_train.shape[1],1)))
model.add(LSTM(50, return_sequences=False))


model.add(Dense(units = 1024))
model.add(Dropout(0.3))
model.add(Dense(units = 256))
model.add(Dropout(0.3))
model.add(Dense(units = 64))
model.add(Dense(1))
```

Fig-11 Building LSTM – Code Snippet

Two LSTM layers with decreasing dense layer units lead up to a single dense layer for the output. The model was fit and compiled and the test dataset was tested on.

RESULTS

General info: The mean absolute error depicts the average of the absolute value of the differences between the actual and predictive values. For example, a value of 0.397 indicates how far off, on average, the model's predictions are. Errors and outliers are not punished as much as other error calculations. This means that if the scale of the data changes, the MAE will not change as much as the MSE, even if the model's accuracy has not changed. The MSE is similar but with a square instead of an absolute value. Because it is based on the squared disparities between the predicted and actual values, the MSE penalizes big mistakes more severely than small errors, and it is also reasonably straightforward to read and calculate. Therefore, the MSE will vary even if the model's accuracy remains the same if the scale of the data varies. The root mean squared error, as implied by the name, is the root of the mean squared error. This is the mean magnitude. This section of the report will merely display the outputs; the interpretation and importance of the outputs will be described in the following sections. Even if the model's accuracy has not altered,

a large increase in the stock price will cause the MSE to rise. This makes it difficult to evaluate the efficacy of several models, or even the efficacy of the same model over different time periods.

This table depicts the Mean squared error, mean absolute error, mean absolute percentage error, and root mean squared error for each of the models

Results	ARIMA	SARIMA	PROPHET	LSTM	Random Forest
MSE	3.00	478584.22	467.95	2.30	0.76
MAE	1.26	599.29	17.88	8.07	0.40
MAPE	5.73	3.17	0.247	-	-
RMSE	1.73	692.25	21.63	6.14	1.1205

The results for the ARIMA, SARIMA, PROPHET were merely in the form of the numbers displayed above. There were no further visualizations. The significance of the results and the results themselves will be described in the discussion section.

LSTM

After employing the code to best choose the hyperparameters, this was the result from it:

Best parameters: {'dense_units': 1024, 'dropout_rate': 0.5, 'neurons': 50}

The graph below portrays the model's prediction of the future stock price. It uses the most recent 60 days to do so. As shown, the upward trend of the close price is expected to continue for a bit.

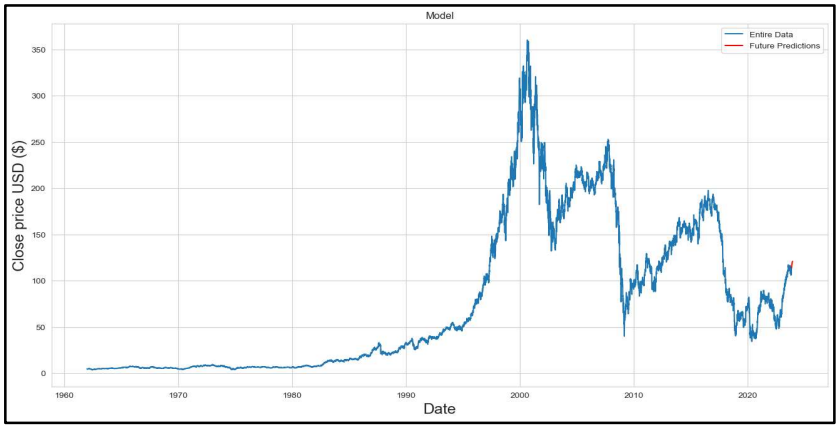

Fig-12: Upward trend of close price

Here it shows the same graph, zoomed in on the prediction part to get a better picture of it:

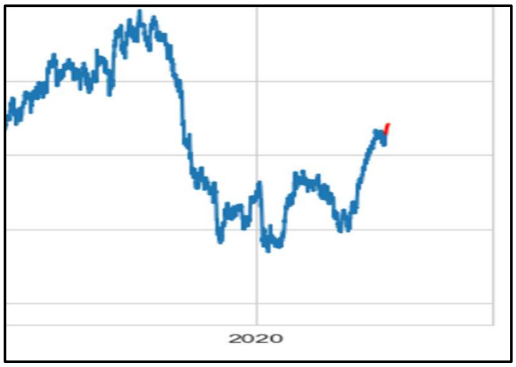

Fig-13: Zoomed version of predicted part

Random Forest

The output for the best max_depth is 20 while the best number of trees (n_estimators) is 300. The MSE from this process is 0.623. In other words, the model concluded that, on the training data, these hyperparameter values accomplished the best.

Future Predictions for Random Forest

The graph below depicts the forecasting prediction for Random Forest

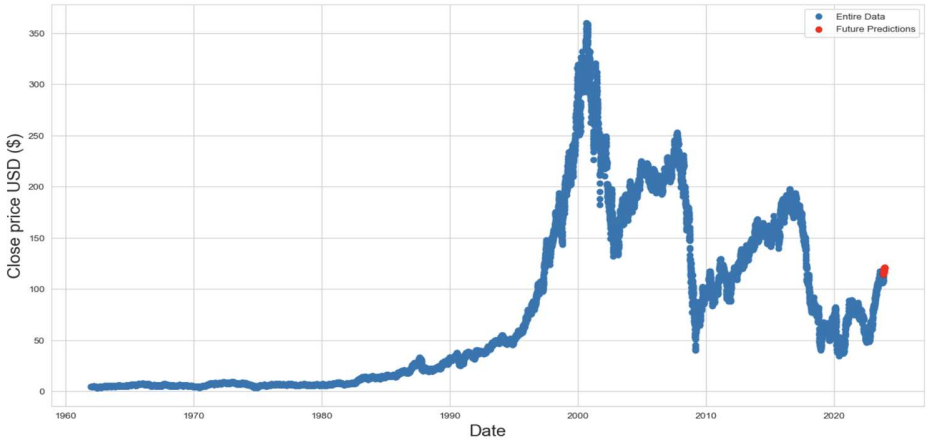

Fig-14: Trend for Forecasting Prediction

DISCUSSION

ARIMA

Testing Mean Squared Error: 3.002
Mean absolute percentage error: 5.73
Testing Mean Absolute Error: 1.260

The abnormally high Mean absolute percentage error indicates a potential problem with the model's predictions. Such a value suggests that the percentage difference is not taken properly, as the MAPE is sensitive to the scale and relative errors of the data. This could be an indication that such errors exist in the dataset. This makes sense due to the team not scaling

the data, as per given recommendation for a stock data such as ours. All the values and columns line up pretty well, as shown in figures 2 and 3. However, the volume column has values that are widely different than the rest. Being that this is one of the parameter's considered, it could be the potential reasoning. Interestingly enough, however, such large value differences did not affect the testing mean squared error, nor did it affect the testing absolute error. Both of these values are actually pretty reasonable. Additionally, although the ARIMA model can handle nonstationary series, it should be sationary to employ the parameters p and q. Because the project requirement involved the tuning and picking of parameters and such, the team proceeded with this approach, which may have impacted the results a bit. If given the chance to repeat this experiment again, the team would like to further examine this and even calculate other forms of error for better comparisons.

SARIMA

Testing Mean Squared Error: 478584.224
Mean absolute percentage error: 31706019469639224.000
Testing Mean Absolute Error: 599.293

Similar to the AIMA model, this model provided an extremely high MAPE percentage. Here, even the testing mean squared error and mean absolute error are much larger than what is deemed normal. A reason for this could be the SARIMA's assumption of stationarity. Stationarity time series are those that have a consistent mean, variance, and such over time. With stock prices, this is not the case, and the systematic rise and falling trends especially prove the data's non-stationarity. This could greatly impact the model's performance. Additionally, the SARIMA model takes into account of seasonal patterns. Stock data can exhibit seasonal trends and patterns. For example, a toy company could have a steep rise in stock around the holiday season. This trend occurs consistently. This idea could be applied here. The stock rises and falls in an almost oscillating pattern, as shown in Figure 3. At certain times the value rises, particularly right before the large even numbers from the interval from 10,000 onwards. Coincidingly, a large Mean squared error portrays a wide disparity of the datapoints from its central mean.

PROPHET

Testing Mean Squared Error: 467.950
Mean absolute percentage error: 0.247
Testing Mean Absolute Error: 17.877

The PROPHET model outputted a small mean absolute percentage error with a very large testing mean squared error. The testing mean absolute error may be perceived as quite large, even though it's not as large as the mean squared error. The mae of 17.87 means that on average the model is 17.87 units away from the actual. Considering that the close price goes well above 200, roughly 350 max, it's not that inaccurate. This is further backed up by the low MAPE value. The mean squared error is quite large, however. It's probable that the model picked up some of the outliers or inconsistencies and increased the weight of the difference. As suggested by Professor Loganantharaj, dealing with outliers is quite tricky in some datasets, and it's not a good idea to simply remove them. If given the opportunity to repeat this model again, the team would like to examine further ways and models that would better deal with these anomalies.

LSTM

Best parameters: {'dense_units': 1024, 'dropout_rate': 0.5, 'neurons': 50}
The output for the LSTM is as follows:
Root Mean Squared Error (RSME): 6.140506835818694
Mean Absolute Error (MAE): 4.934367591574843
Mean Squared Error (MSE): 37.7058242007361

The best parameters for this model are portrayed above. There will be 1024 neurons that are in the dense layer after the LSTM layer. These will help interpret the extracted features. Not to be confused by the 50 neurons value, which conveys the number of LSTM neurons in that respective recurrent layer. Relatively, these values are quite large, proving the complexity of the data's intricate patterns. The dropout rate will regularize and prevent overfitting. Here with 0.5, the model will randomly select half of the input units and set them to zero. The root mean absolute error and mean absolute error worked particularly well in this model when comparing it to the relatively high data instance values. The data points are on average roughly 5 units away from the mean. The mean squared error is decent as well. If given the chance to repeat this model, the team would like to repeat the steps on a sampled data. The model took multiple hours just to run once, and if there was an error in the code, it would not be indicated until after all that time has passed. Time-wise, it was quite inconvenient. Thus, along with working on a sample, the team would like to find a way to allow a better efficiency.

As an example for how the error values were calculated, here is a portion of the code that covers these:

```
    mae = mean_absolute_error(y_test_lstm, predictions)
    mae_mean = np.mean(mae)
    print(f"Mean Absolute Error (MAE): {mae_mean}")

    # Calculate Mean Squared Error (MSE)
    mse = mean_squared_error(y_test_lstm, predictions)
    mse_mean = np.mean(mse)
    print(f"Mean Squared Error (MSE): {mse_mean}")
9]

Mean Absolute Error (MAE): 4.934367591574843
Mean Squared Error (MSE): 37.7058242007361
```

Fig- 9: Mean Absolute and Mean Squared errors for LSTM– Code Snippet

This graph shows the training, predicted, and validation data compared to each other. Seeing that they fall along the same general path, means the model is learning pretty well at the appropriate complexity. This also portrays that the model did not overfit, as that would show very contrasting validation and prediction results. This can further lead to some more insight on the data and model, like that the parameters and hyper parameters were chosen well and accordingly. The complicity of the model matched the complexity of the data with sufficient number of instances for the model to train on. This may differ for different models, but we can interpret such a statement as valid for this case.

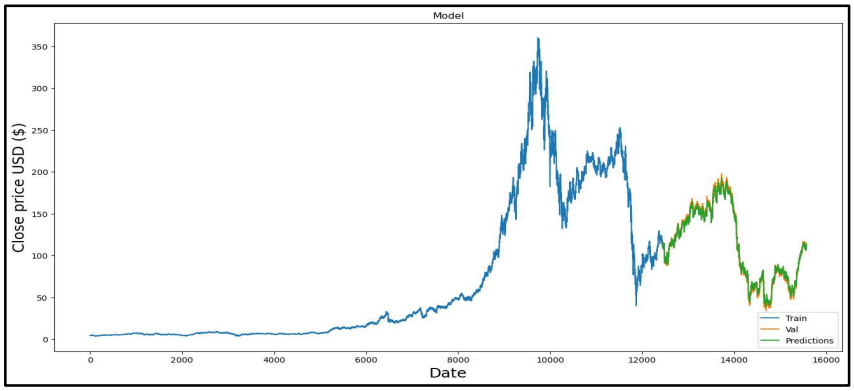

Fig-10: Comparison of training, predicted, and validation data

The graph below portrays the model's prediction of the future stock price. It uses the most recent 60 days to do so. As shown, the upward trend of the close price is expected to continue for a bit. This makes sense when visually witnessing the prior graphical trends. The close price rises after the large even increment, then it falls as it gets close to the next one. In terms of resources, the team was only able to predict that small orange segment. Even for that, the system took a lot of time and CPU. It became inefficient after a certain point. However, based on this initial trend itself, the team infers that

the model will create a similar trend pattern if the line continued. LSTM works well for time sequential data after all. The low error rates also give more confidence in this statement.

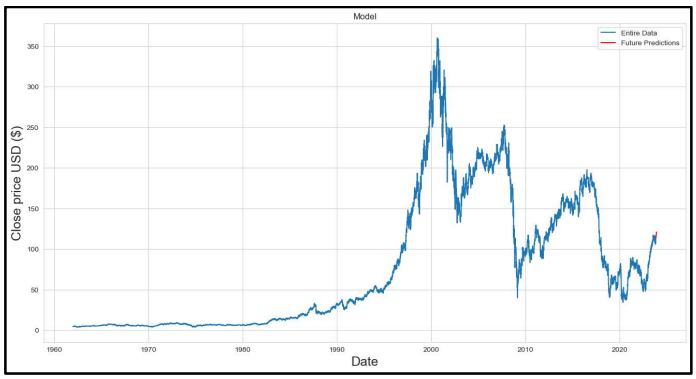

Fig-11: Upward trend of close price

Here it shows the same graph, zoomed in on the prediction part:

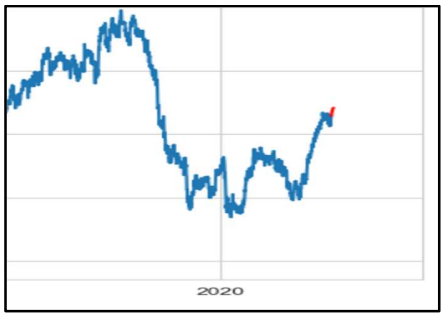

Fig-12: Zoomed version of predicted part

Random Forest

The output for the best max_depth is 20 while the best number of trees (n_estimators) is 300. A high max depth allows the trees to obtain a deeper connection, in a sense. A larger n_estimates value shows the complexity of the data as well. A large depth increases the risk of overfitting while a large number of trees can decrease the risk of overfitting. The MSE from this process is 0.623. In other words, the model concluded that, on the training data, these hyperparameter values accomplished the best. Using these hyperparameters, the prediction and actual values had the following:

Mean Absolute Error: 0.8542
Mean Squared Error: 1.2556
Root Mean Squared Error: 1.1205
Accuracy: 99.03 %

Here, the mean absolute error is very good given how large the numbers in the data get. Even the mean squared error and root mean are quite impressive. Going back to the overfitting scenario, this was able to predict and model properly without overfitting. Overfitting would have lead to a very large number. This provides further evidence that the hyperparameter optimization worked quite well.

```
In [265]: print("Mean Absolute Error:", round(metrics.mean_absolute_error(y_test, predict), 4))
          print("Mean Squared Error:", round(metrics.mean_squared_error(y_test, predict), 4))
          print("Root Mean Squared Error:", round(np.sqrt(metrics.mean_squared_error(y_test, predict)), 4))
          print("(R^2) Score:", round(metrics.r2_score(y_test, predict), 4))
          print(f'Train Score : {model_RF.score(X_train, y_train) * 100:.2f}% and Test Score : {model.score(X_test, y_test) *
          errors = abs(predict - y_test)
          mape = 100 * (errors / y_test)
          accuracy = 100 - np.mean(mape)
          print('Accuracy:', round(accuracy, 2), '%.')

          Mean Absolute Error: 0.8542
          Mean Squared Error: 1.2556
          Root Mean Squared Error: 1.1205
          (R^2) Score: 0.9994
          Train Score : 100.00% and Test Score : 97.47% using Random Tree Regressor.
          Accuracy: 99.03 %.
```

Fig-13: Mean Absolute and Mean Squared errors for RF– Code Snippet

COMPARISON

Tabel-1: Comparison of MSE and MAE scores among models

Evaluation Metrics/Model	MSE	MAE
ARIMA	3.00	1.260
SARIMA	478584.224	599.293
PROPHET	834.966	25.757
LSTM	37.705824	4.934367
RANDOM FOREST	1.2556	0.8542

In terms of comparing models, Mean squared error and mean absolute error are the most common measures. Reiterating, MSE considers the square of the summed differences while MAE considers the absolute value of the differences. Comparing all the models, the Random Forest had the lowest mean squared error and mean absolute error. The training was the best of the bunch and it performed well on the testing data. Thus, the team expected it to perform better in forecasting. LSTM was better, however, for the prediction of future instances. The trend is far more believable and stable than that of Random Forest. In terms of order for the MSE, the most effective models are listed as Random Forest, ARIMA, LSTM, PROPHET, then SARIMA. For MAE, the order from best to worst goes Random Forest, ARIMA, LSTM, PROPHET, and finally SARIMA. Interesting to note here, regardless of the order, the mean absolute error values are smaller than the mean squared value of the respective model. This observation can lead to a few insights. For example, there could have been outliers and anomalies in the data whose presence was increased in the squared difference process. The mean absolute error is less susceptible to noise. The differences for each data instance, between the predictive and actual values, are treated equally. Even without the presence of outliers, the disparity itself is shown more in the mean squared error. It also reiterates that the working data is nonlinear, as such nonlinearities lead to the results that were outputted.

CONCLUSION

The team examined the GE stock data set to perform future predictions on the close data. This close data conveys the price of the stock at the end of the day. The data holds the following properties:
Shape = (15573, 7), with float64 and int64 data types. Its columns are date, open, high, low, close, adj close, and volume. It has no missing values. Three statistical models were made: Arima, SARIMA, and Prophet. These helps identify the patterns and trends of the data. They can also be used for forecasting, the mean squared error and mean absolute error shed more light on the productivity of said predictions. Then, LSTM and Random Forest models were created, trained, tested, and applied. Despite the Random Forest model performing better on the test data (lower MSE and MAE), the LSTM performed better in predicting the future. The conclusion of LSTM being better than Random Forest, in future predictions, leads to some insight on the data itself. LSTM works better on sequential dependencies, so time or sequence-based data works better with this model. Additionally, the relationships in the data are quite complex to relate to each other in this case. Such intricacies and non-linearities are captured better by LSTM. It is more flexible to such circumstances than Random Forest. The goal of the project is to line up better with LSTM, given the goal of using the prior data instances to make future predictions. Random Forest offers limitations in these.

REFERENCES

[1] Aras, S., Deveci Kocakoc, I. and Polat, C. (2017). Comparative study on retail sales forecasting between single and combination methods. J. Bus. Econ. Manag. https://doi.org/10.3846/16111699.2017.1367324

[2] Ansuj, A. P., Camargo, M. E., Radharamanan, R., and Petry, D. G. (1996). Sales forecasting using time series and neural networks. Computers & Industrial Engineering 31(1): 421–424. https://doi.org/10.1016/0360-8352(96)00166-0

[3] Alon, I., Qi, M., and Sadowski, R. J. (2001). Forecasting aggregate retail sales: a comparison of artificial neural networks and traditional methods. Journal of Retailing and Consumer Services 8(3): 147–156. https://doi.org/10.1016/S0969-6989(00)00011-4

[4] Frank, C., Garg, A., Sztandera, L., and Raheja, A. (2003). Forecasting women's apparel sales using mathematical modeling. International Journal of Clothing Science and Technology 15(2): 107–125. https://doi.org/10.1108/09556220310470097

[5] Aburto, L., and Weber, R. (2007). Improved supply chain management based on hybrid demand forecasts. Apl. Soft Computing 7(1): 136–144. https://doi.org/10.1016/j.asoc.2005.06.001

[6] Ramos, P., Santos, N., and Rebelo, R. (2015). Performance of state space and ARIMA models for consumer retail sales forecasting. Robotics and ComputerIntegrated Manufacturing 34: 151–163. https://doi.org/10.1016/j.rcim.2014.12.015

[7] Fabianova, J., Kacmary, P., Molnar, V., and Michalik, P. (2016). Using a software tool in forecasting: a case study of sales forecasting taking into account data uncertainty. Open Engineering 6(1): 270–279. https://doi.org/10.1515/eng2016-0033

[8] Malhotra, Pankaj, et al. "Long short-term memory networks for anomaly detection in time series." Proceedings of the 21st ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, 2015. sting. International Journal of Forecasting 32(3): 788–803. https://doi.org/10.1016/j.ijforecast.2015.12.004

[9]Yan,S.(2017,November15). *Understanding LSTM and its diagrams*. Medium. https://blog.mlreview.com/understanding-lstm-and-its-diagrams-37e2f46f1714

[10] Chaya. (2022, April 14). *Random Forest regression*. Medium. https://levelup.gitconnected.com/random-forest-regression-209c0f354c84

TEAMS VIDEO RECORDING:

https://uofh-my.sharepoint.com/:v:/g/personal/vinakoll_cougarnet_uh_edu/EV7XRGA2V6tFg59axw6zuioBjz-UQ02-dmLC_aB4Y33zGw