Problem Statement:

Investors would become extremely wealthy if they knew which stocks would increase the next day and which stocks would decrease the next day. Current and future stock prices are affected by many different factors such as previous values, other companies, the overall economy, and even its own expected future value. This project exclusively utilizes a single stock's historical prices to predict future returns. By implementing a Recurrent Neural Network with Long Short Term Memory, the model is expected to learn trends and provide insight into whether a particular stock should be bought, held, or sold.

The client is any person with a particular interest in a single stock. The model is built to be used on a day to day basis to predict tomorrow's adjusted closing price. If the client is currently holding or intends to purchase a stock of a particular company, the model will predict tomorrow's returns and use the predicted change to predict tomorrow's stock price at the close of the day. With the insight provided by the model's prediction, the client will be better informed when deciding to buy, hold, sell their stock with a particular company.

Additionally, this model can serve as the foundation for Robotic trading/advising applications. Besides simply informing a single client about a particular stock, an application built around this predictive model could be scaled to automatically manage a stock portfolio with little to no human intervention. This means that one or many clients could simply fund a portfolio and provide a list of stocks for the model to manage based on its daily predictions. As a benefit to the user, this type of portfolio would help in the diversification of assets as its returns are uncorrelated to the buy and hold approach.

This project is intended to focus primarily on Time Series Analysis and implementing a Recurrent Neural Network with Keras to predict future changes. The project will showcase a strong understanding of Neural Networks and how to fine-tune the model architecture to maximize the accuracy of its predictions. This project will not yet go on to build trading software, but will instead provide a production level Machine Learning Model that provides insight to an individual client on a day to day basis. Predictions for the next trading day's closing price can be made immediately at the close of the current trading day.

Description of the Dataset:

Thanks to the yfinance library, collecting the necessary data for this project is rather simple and can be easily done from the command line. With the project files properly installed and with the project's TImeSeries module properly installed, the command

```
$ fetch_raw_data -ticker <TICKER>
```

will collect all of the historical stock data relevant to the stock ticker symbol provided as the command line argument. If no ticker symbol is provided as a command-line argument, the script will default to collecting Google's historical stock data.

The yfinance library makes collecting historical stock data from Yahoo Finance very simple. With the following code,

```
import pandas as pd
import yfinance as yf
historical_data_df = yf.download("GOOG", start=ipo_date, end=today)
```

all of a single stock's historical data can be retrieved from Yahoo Finance and returned as Pandas DataFrame. The historical data is then written to the file /TimeSeries/data/raw/raw.csv

Once the the data has been collected and written to the appropriate file, the historical stock data is ready to be cleaned, wrangled, and formatted for time series. The command

```
$ format_timeseries
```

will format the data in a way that the Recurrent Neural Network will be able to make effective predictions. In its raw format, the DataFrame holding the historical contains columns for [Date, Open, High, Low, Close, Adj Close, Volume] with each row containing the respective values at the close of each specified date. This project will use exclusively the Adj Close values for each day to provide insight about the next day's Adj Close.

To train a recurrent neural network on the time series formatted data, the command

```
$ predict_tomorrow
```

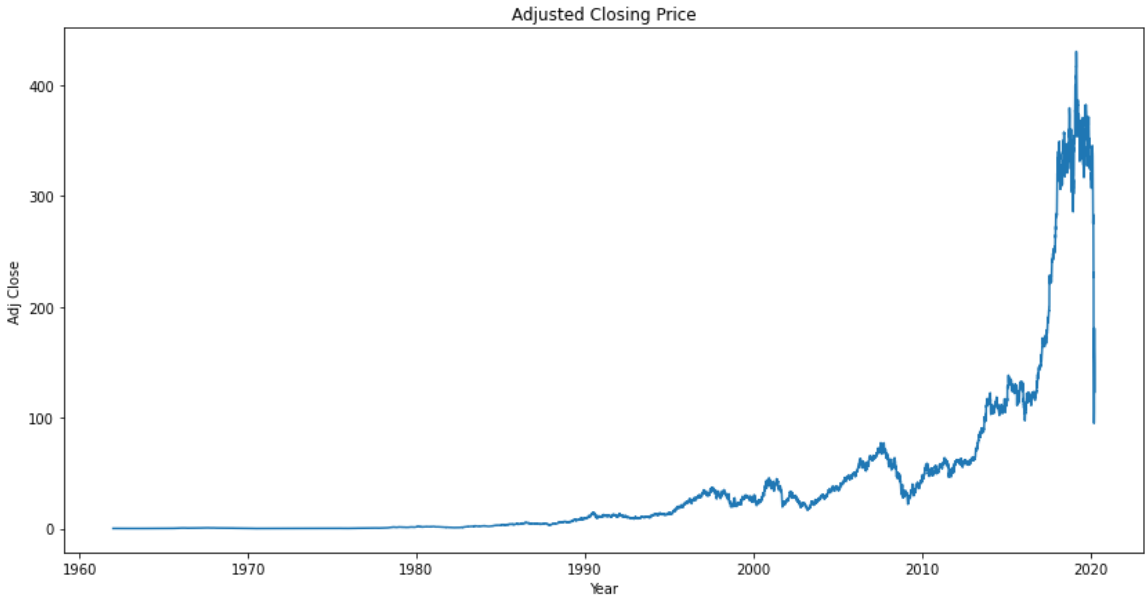
will train a Sequential Keras Model with LSTM layers and provide a prediction for tomorrow's expected change. For the stock corresponding to the command line argument provided in the fetch_raw_data command, the user will be advised on whether to buy, hold, or sell the particular stock.

To simultaneously run all of the previous steps and make a quick prediction from the command line, simply run

```
$ make prediction
```

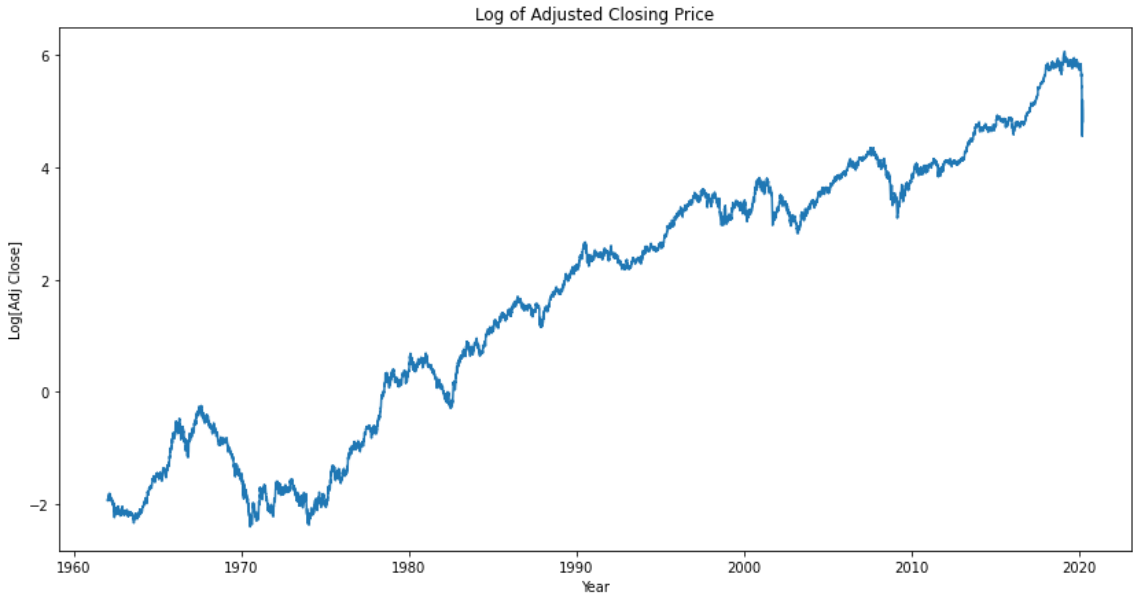
This command installs all of the necessary requirements for the virtual environment, retrieves the desired data with yfinance, prepares the data for a time series analysis, trains a recurrent neural network with Long Short Term Memory, scores the model's ability to make predictions on a holdout set, and provides a prediction for tomorrow's change in stock price to the user.

In order to implement a Recurrent Neural Network with Long Short Term Memory, a new DataFrame must first be created with only the historical Adj Close values. Predicting Adj Close values alone will not be enough to train the model. Doing so will lead to a model that predicts a price that is close to the mean, with no emphasis on the change of the Adj Close price with respect to the previous day. Before considering changes in price, taking the log of each data point can remove the exponential behavior in the values over time. Meaning we need to normalize the changes by accounting for the case when changes in stock price when the stock price is high can dominate small changes when the stock price was small. Before taking the log, the time series looks as follows:



This issue is resolved by taking the log of each individual adjusted close value.

$$value_i = Log_e(price_i)$$



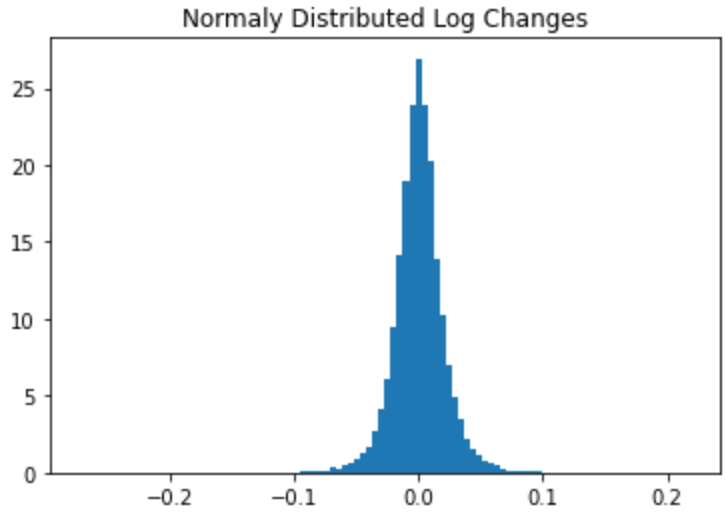
Because the model will be predicting the daily change of the Adj Close, a new column is created where each value is the difference between the next trading day's Adj Close price and the current day's Adj Close price.

$$delta_i = Log_e(price_{i+1}) - Log_e(price_i)$$

Or, a more simply

$$delta_i = Log_e\left(\frac{price_{i+1}}{price_i}\right)$$

This operation results in **normally distributed** values that are better suited for the model to predict. The Recurrent Neural Network will utilize Long Short Term Memory to predict the **log change** in stock price. Or more simply, the log tomorrow's expected returns. Find more about interpreting results in the Machine Learning and Initial Findings section of this document.



To format the data for LSTM, a loop can be written to create a column for each of the previous 5 days (1 trading week) before the current day. Any individual row will have the following shape and values.

DateTime Index	back_5 (predictor)	back_4 (predictor)	back_3 (predictor)	back_2 (predictor)	back_1 (predictor)	Log[delta(Adj Close)] **(target)**
date	$delta_{i-5}$	$delta_{i-4}$	$delta_{i-3}$	$delta_{i-2}$	$delta_{i-1}$	$delta_i$

The data is now prepared and ready to be split into training and testing sets and then split again into predictors and targets. The features/predictors are the log changes of the previous week. The target being predicted is the log of the change going into tomorrow.

With everything structured correctly, the first 5 rows will have missing values due to the absence of previous data. These rows can be dropped because they will be ineffective when training the model. Additionally, the row corresponding to the most recent date will have an empty value in the target column. **This is correct and the row should be kept, as this missing value is the one the model will predict.** The head and tail of the time-series DataFrame should resemble the following table.

	back_5	**back_4**	**back_3**	**back_2**	**back_1**	**Adj Close**	
Date							
1966-07-05	**NaN**	**NaN**	**NaN**	**NaN**	**NaN**	0.052056	**drop row**
1966-07-06	**NaN**	**NaN**	**NaN**	**NaN**	0.052056	-0.036905	**drop row**
1966-07-07	**NaN**	**NaN**	**NaN**	0.052056	-0.036905	0.011215	**drop row**
1966-07-08	**NaN**	**NaN**	0.052056	-0.036905	0.011215	-0.003724	**drop row**
1966-07-11	**NaN**	0.052056	-0.036905	0.011215	-0.003724	-0.015037	**drop row**
...	**...**	...	...	...	...	...	**train/test**
2020-03-27	-0.079807	0.166577	0.006340	0.026460	-0.020160	0.024810	**train/test**
2020-03-30	0.166577	0.006340	0.026460	-0.020160	0.024810	-0.016673	**train/test**
2020-03-31	0.006340	0.026460	-0.020160	0.024810	-0.016673	-0.044394	**train/test**
2020-04-01	0.026460	-0.020160	0.024810	-0.016673	-0.044394	0.020835	**train/test**
2020-04-02	-0.020160	0.024810	-0.016673	-0.044394	0.020835	**NaN**	**prediction**

Initial Findings:

Interpreting Results:

Once the Recurrent Neural Network has been trained and predictions have been made on the test data, the results need to be interpreted in order to score the accuracy of the model's predictions. To create interpretable results, the inverse of the Log and Delta operations needs to be applied to the predictions. Currently, predictions resemble a Log of a change as follows.

$$prediction = Log_e[delta] \qquad \text{with} \qquad delta = \frac{Price_{today}}{Price_{tomorrow}}$$

To revert the log, we apply the exponential function to our prediction,

$$e^{prediction} = \frac{Price_{today}}{Price_{tomorrow}}$$

At this step, the exponential of the prediction resembles the expected return as a decimal. Finally, all that is left to do is multiply the exponential of the prediction by today's price to obtain the value of tomorrow's predicted price.

$$Price_{tomorrow} = Price_{today} * e^{prediction}$$

Therefore allowing us to use the models predicted change to determine the future price.

Scoring the Model:

The model can be evaluated on its ability to make predictions on the holdout set. **While the model is being fit to the data, the Gradient Descent algorithm adjusts the weights during backpropagation to minimize mean squared error.** But, for the purpose of this particular, the model only needs to be evaluated on its ability to predict if returns will be positive or negative. To score the model, the exponential of the predictions can be used instead of the expected price of the following day. This means we can compare the predicted returns to the returns calculated from the test data. Simply, an exponential of the prediction greater than one resembles and expected increase in stock price and an exponential of the prediction less than one resembles and expected decrease in stock price.

$$e^{prediction} > 1.0 \quad \Rightarrow \quad +positive\ returns\ expected$$

$$e^{prediction} = 1.0 \quad \Rightarrow \quad Price_{tomorrow} = Price_{today}$$

$$e^{prediction} < 1.0 \quad \Rightarrow \quad -\ negative\ returns\ expected$$

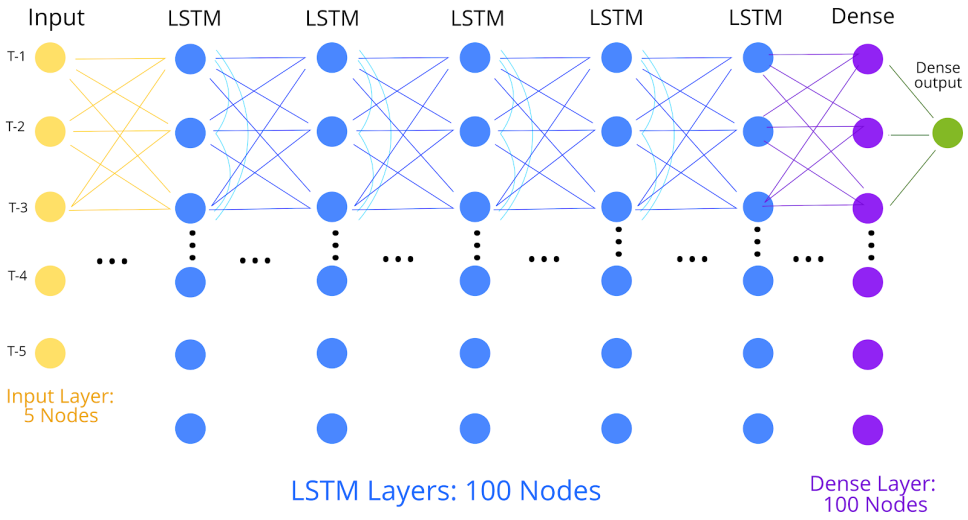
Improving Model Architecture:

With an effective method of scoring the model's ability to make valid predictions, Cross-Validation can be applied to improve model architecture and identify optimal hyperparameters. Through Cross-Validation, the following hyperparameters and Keras model architecture yielded the best results.

Model: "sequential_1"

Layer (type)	Output Shape	Param #
lstm_1 (LSTM)	(None, 1, 100)	42400
lstm_2 (LSTM)	(None, 1, 100)	80400
lstm_3 (LSTM)	(None, 1, 100)	80400
lstm_4 (LSTM)	(None, 100)	80400
dense_1 (Dense)	(None, 100)	10100
dense_2 (Dense)	(None, 1)	101

Total params: 293,801
Trainable params: 293,801
Non-trainable params: 0



Specific Model Parameters:

Input Layer:	5 nodes
LSTM Layers:	4 Layers, 100 Nodes
Dense Layers:	1 Layer, 100 Nodes
Dense Output Layer:	1 Node
Nodes in each Layer:	25
Training Epochs:	2
Optimizer:	"adam"
Loss function:	"mean_squared_error"
Mean accuracy of 10 samples:	53.23%

Determining the RNN's True Mean Accuracy:

During Cross-Validation, the above architecture returned the best mean accuracy when ten models were trained and evaluated against the test set. The accuracy in the table above is merely the mean of ten randomly sampled models and does not represent the true accuracy of the model. Because this architecture produced ten models that collectively outscored the mean accuracy of other models trained on respectively different architectures, this particular architecture will be used to explore the true mean accuracy.

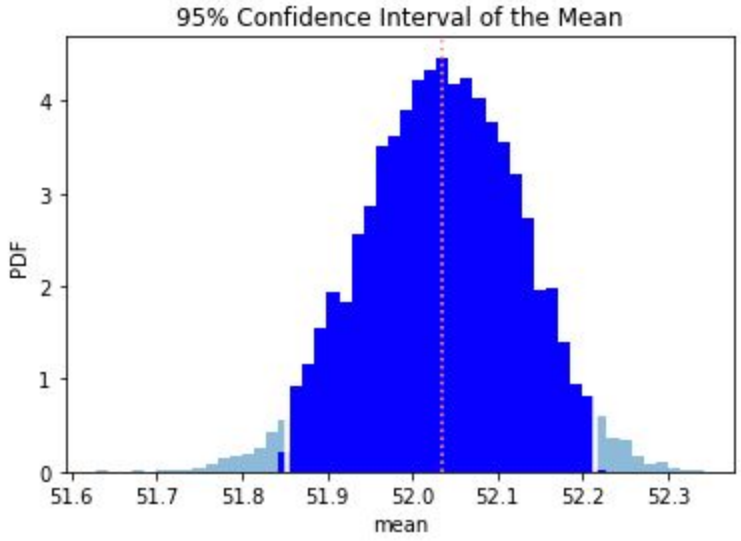
Ten samples were used to approximate the model's accuracy, but random chance and the train test split have a great impact on the scoring of the model. Luckily, it is not difficult to train and score models with this architecture, thus it is easy to obtain many more samples. To identify the true mean accuracy of models trained on this architecture, the accuracies that have been collected can then be **bootstrapped** to create a distribution of accuracies. Once bootstrapped, insight regarding the true mean can be obtained with less influence of random initial weights and where the data is split.

The best-case scenario of bootstrapping as it applies to the RNN would provide a true mean accuracy significantly larger than 50.0%. A result like this would describe the model's predictive power and its ability to outperform a random vector. A true mean accuracy of 50.0% or less would indicate that the model is no better than flipping a coin to determine whether to buy, hold, or sell the particular stock.

Bootstrapping the mean produced the following results. Bootstrapping the means corresponding to a sample of trained models, resulted in a mean accuracy of 52.035%

```
1 bootstrapping = bootstrap_mean(accuracy_samples)
```

```
52.035 52.035
```




```
1 replicates, int_min, int_max = bootstrapping
2 np.mean(replicates)
```

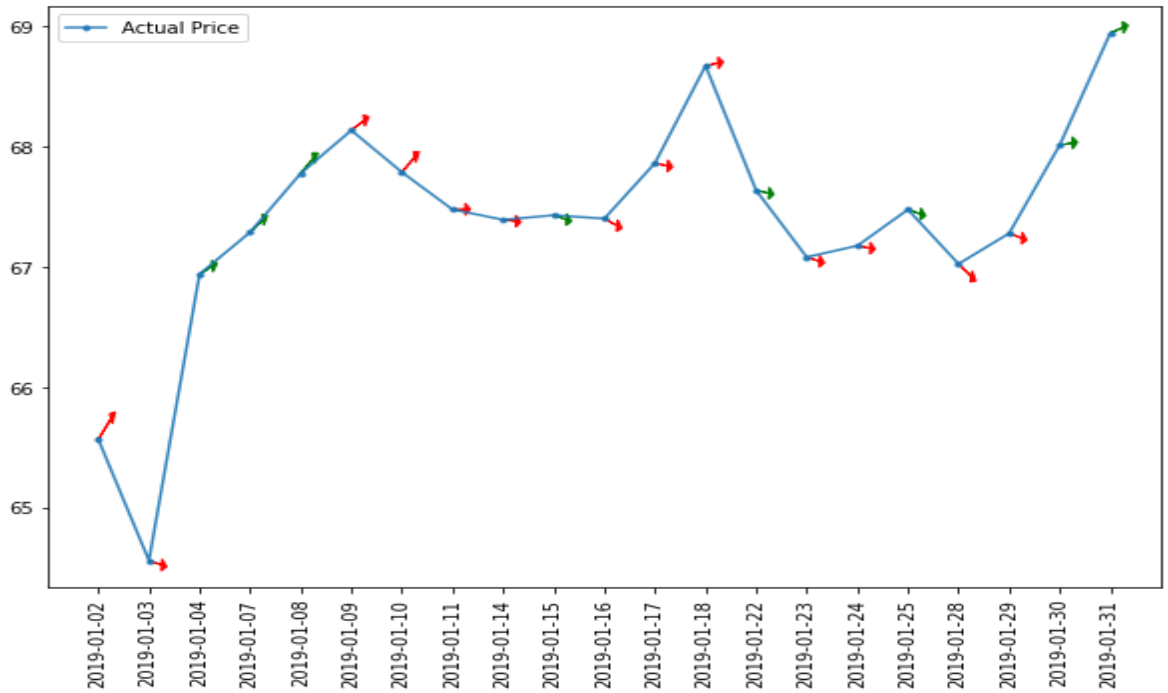
```
52.03477468541332
```

Sample Output:

```
Predicting change in stock price with 51.362398% accuracy

Date            price_today       price_tomorrow     Action
2020-04-15       90.79             90.93             BUY
```

The following visualization provides a series of sequential daily predictions made by the model for test data from the month of January 2019. The blue line depicts the actual price curve during the month of January 2019. The direction of each arrow depicts the model's prediction of the price the following day. The color of the arrow depicts whether the model correctly or incorrectly predicted the expected direction of change. The visualization for a perfectly trained model that performs with 100% accuracy and Zero mean squared error would show green arrows for each day that are exactly parallel to line plot depicting actual price.



Significance of Results:

Though these results may not seem significant, this predictive model can generate a tremendous amount of wealth when scaled with enough capital. This model provides a slight edge over a client randomly choosing whether the price will go up or down. Though not by much, the model produces correct predictions more often than not.

Link to Project: https://ahalarewicz.github.io/TimeSeriesAnalysis/