Presentation Script

Slide 1:
Cover page with course code and name with team member details.

Slide 2:
Title: Introduction

Out of the given options for the final project we selected option 1 from the programming project which is predicting a single asset that is price or volatility related.
For investors and researchers, predicting stock prices has always been an interesting subject. Investors constantly wonder whether the price of a stock will increase or decrease; however, because there are numerous complex financial indicators that are only understood by investors and those with strong financial backgrounds, the stock market trend is inconsistent and appears to be completely random to the average person.
Machine learning offers non-experts a fantastic opportunity to forecast events accurately, make a constant profit, and assist professionals in obtaining the most useful indicators and improving their predictions. The purpose of our project is to develop a model that predicts the price of a stock in the future. The chosen stock was Amazon and the future window was set at 15 days in the future. We based our project on an already developed neural network. More specifically the code used a Recurrent neural network with LSTM cells. This was developed using Keras and TensorFlow 2. We also incorporated financial metrics to validate the quality and accuracy of the model.
The link to seed code is here.

Slide 3: We made 4 improvements to the seed program. In order to evaluate the results we incorporated financial metrics, such as Compound Annual Growth Rate (CAGR) and Sharpe ratio. We also made two more models and tested which model performed the best and used the best model to perform the predictions. Finally we used a White Reality Check to evaluate the obtained results.

Slide 4:
Title:Model from seed program

The seed program contains the following steps:
1) Data extraction
2) Data preprocessing
3) Model creation
4) Model training
5) Model testing
6) Target prediction

Slide 5:
Title:Data extraction

For this step we used the load_data function. This function utilizes the yahoo finance library. Our function takes as parameter ‘ticker’ which can be a string or a df. This is the parameter which will hold the stock symbol we are interested in. For example if we want to predict the price of Amazon stock, we will set the ticker to AMZN. If the parameter is a string, we use the get_data function to load the data from yahoo finance. If the parameter is a df, we assign it to our df variable. Otherwise our function throws an error. Once we extract our data, we will make a copy of it and assign it to a result dictionary with key ‘df’. After extracting our inputs, we will start preprocessing the data.

Slide 6:
Title:Data preprocessing

The preprocessing portion of the function handles tasks:
1) Data scaling
2) Identifying the inputs and target
3) Data cleaning
4) Splitting the data in testing set and training set
All these steps are done in the same function after extracting the data from yahoo finance. The load_data() functio does the following:

1. Using the stock info.get data() function in the yahoo fin module, it first loads the dataset.
2. If the "date" column from the index is missing, it is added; this will assist us in later retrieving the characteristics of the testing set.
3. Using the MinMaxScaler class from sklearn, it will scale all the prices (including the volume) from 0 to 1 if the scale argument is set to True. Be aware that there is a scaler for each column.
4. The adjusted close column is then shifted by lookup step to add the future column, which represents the goal values (the labels to predict, or the y's).
5. The data is then divided into training and testing sets and shuffled before the outcome is eventually returned.

Slide 7:
Title:Data scaling

This task was accomplished by using a boolean parameter scale and feeding it to the load_data function. If scale is set to True, the function creates a new dictionary and loops through the columns in the column_feature list. Each column is scaled using the MinMaxScaler function. The new scaled values are added to the corresponding column in the df. The scaled values are also saved in the new dictionary created.

Slide 8:
Title:Identifying inputs and target

For input features we have used all the available features extracted from yahoo_finance. The feature columns available are: adjclose, volume, open, high and low.
To add the target column to our DF, we use the shift method and apply it only to the adjclose feature. We apply the shift in the negative direction by the number of lookup_step, in this case we want to predict the stock price in 15 days, so the lookup_step is set to 15. This in turn creates a problem as the last 15 values in the target column will be set to NAN. Frequency of the target is ‘daily’.

Slide 9:
Title:Cleaning the data

The data obtained contains few NaN values. In order to proceed with further analysis we need to clean the data first. We need to remove the NaN values from the target however we do not want to lose the associated feature values as those can be used to test the model. To solve this issue we create an np array where we store the feature column values for the last 15 rows using the .tail() method. Once the inputs are saved we drop the rows containing NaN values from the df.

Slide 10:
Title:Train/ test split

Now that the data is clean after removing the Nan values, our inputs and targets are ready. The last step is to split the training and testing sets. We do this by using the train_test_split() function and set the test size to 20% of the data. 20% was found to be the best test size as 80% of the data is still remaining to have sufficient examples for the model to pick up on the patterns in the data so that it is complex enough. Thus, preventing underfitting. It was also important to not have the model too fitted to the data and ensure that it is still flexible enough to predict examples that were never seen before. Thus preventing overfitting.

Slide 11:
Title:Model Creation

After extracting and prepping the data, the next step is to create the model. The model is created using Keras. We utilize the sequential() method to initialize the model. The architecture of the model is dynamic. The layers are set through a parameter, n_layers. We then start a loop that counts through the number of layers. In the first layer we set the parameter batch_input_shape to set the number of neurons and the batch size we also set the return_sequences parameter to True such that the input to the following layer is the output of the previous layer. The following hidden layers have an output of the units parameter. Finally at the last layer, the return_sequences parameter is set to false. A simplified network is displayed

in the figure on the left. Due to the complexity of the network, the exact architecture cannot be captured in the figure.

Slide 12:
Title:Model training

We constructed the first model with loss as mean absolute error, adam optimizer, used 300 units and 3 layers. Second model with Huber loss, SGD optimizer, 300 units and 3 layers. The last model with Huber loss, adam optimizer, 256 units and 2 layers. We trained all these models on the training data and plotted their histories.

Slide 13:
Title:Model Testing

We plotted the history of all the 3 models together in a single plot for comparison and discovered that our model 3 performed the best with loss=0.000582 and MAE=0.016. Therefore, model 3 will be used for further analysis and prediction of stock price.

Slide 14:
Title:Target Prediction

The seed program included a custom predict() function that took model and data as inputs which were returned by create_model() and load_data() functions respectively. For making predictions, first the dimensions of the last sequence from data are expanded that were retrieved. The predictions are made using the model.predict() method that gives an output scaled from 0 to 1. After this step, the price is obtained by inverting the scaling and finally the function returns the predicted price. Then the model is evaluated by model.evaluate() method and mean absolute error is calculated by inverse scaling.Accuracy is calculated by counting the number of positive profits. Total profit is calculated by adding sell and buy profits together and finally profits per trade is calculated by dividing total profit by number of trades.

Slide 15:
Title:Explaining the metrics

Mean Absolute Error:We receive an error of around 20, which means that the model forecasts are, on average, more than $20 off from the genuine values. This will vary from ticker to ticker, and when prices rise, the error will rise as well. As a result, you should only use this metric to compare your models when the ticker is steady (e.g., AMZN).
Accuracy Score:This represents the level of accuracy of our predictions. This computation is based on all trades from the testing samples that resulted in positive profits.
Profit per trade:It is calculated as the total profit divided by the total samplings used for testing.
Buy/Sell Profit: Using the get final df() function, we computed the profit that would result from opening trades on each testing sample

Slide 16:
Title:Plot of actual price Vs predicted price

The red curve represents the expected prices, and the blue curve is the actual test set. As we predicted, the stock price rose and has been falling. This plot displays the prices of the testing set scattered across our whole dataset together with associated forecasted values because we set SPLIT_BY_DATE to False. The testing set will comprise the final TEST SIZE proportion of the entire dataset if SPLIT_BY_DATE is set to True.

Slide 17:
Title:Financial Metrics

We evaluated our predictions and model's performance against financial metrics: CAGR and Sharpe ratio.
The Compound Annual Growth rate(CAGR) is a statistic that represents the rate of growth that an investment would have experienced had it grown at the same pace each year and reinvested its earnings at the conclusion of each period.
When a new asset or asset class is introduced to a portfolio, the Sharpe ratio is used to analyze how the portfolio's overall risk-return characteristics have changed.
CAGR over training data is 41.7% ie we can get around 42% of return on our investment. We also get a 85.9% risk-adjusted performance on training data.
CAGR over test data is 18.9% ie we can get around 19% of return on our investment. We get 68.3% risk-adjusted performance over test data.

Slide 18:
Title:White Reality Check

One of the many techniques self-deceit traders employ while creating a trading strategy is data snooping. White's reality check, which is aptly titled, seeks to lessen the degree of self-deceit and offers a relatively straightforward but effective approach of sorting the good data-mined trading strategies/models/patterns/whatever from the bad.
The p-value that we get after using the white reality check is equal to 0.2992. The p-value should be as less as possible. Therefore, the null hypothesis cannot be rejected.