Title: Task B-7 Report
Author: Robin Findlay-Marks, s103603871

Task information:

For this extension task I decided to do the simple extension. This meant that I added some functionality to the program that has been documented and implemented elsewhere. To do this I decided to add two things to the program, random forest prediction and prophet prediction. I originally wanted to add a Convolutional Neural Network (CNN) to the program but I found this too difficult.

Subtask 1:
For the first subtask I added the random forest prediction to the program. Random forest is basically an ensemble of a large number of decision trees and uses a supervised learning dataset as an input.

Main Random forest function
```
def runTestForest():
    # get test + train data and make local variables
    global testData
    global trainData
    train = trainData["Close"].values
    test = testData["Close"].values

    # turn train and test data into supervised learning
    train = data_to_supervised(train)
    test = data_to_supervised(test)

    # make predictions
    forestPrediction = forest_validation(test, train)

    return forestPrediction
```

The first step in the setting up for the random forest is transforming the data into the supervised dataset. This is done by getting the input data, in this case training data. This data is then put through a sliding window to make the new samples for the supervised learning data. It is called a sliding window, because the window of inputs and expected outputs is shifted through time to create the new samples for a supervised learning dataset. This is then repeated for the test data.
Data to supervised function:
```
    # turn time series data into a supervised learning data
def data_to_supervised(data):
    df = pd.DataFrame(data)
    colunms = list()

    # sliding window technique used to make the new samples for the supervised learning data
    # input sequence (t-n, ... t-1)
    for i in range(1, 0, -1):
        colunms.append(df.shift(i))
    # forecast sequence (t, t+1, ... t+n)
    for i in range(0, 1):
        colunms.append(df.shift(-i))
    # put it all together
    newdf = concat(colunms, axis=1)
    # drop NaN values
    newdf.dropna(inplace=True)
    return newdf.values
```

For the next step, the test and train supervised datasets are then inputted into the forest loop function. This works by training the model on the training data then predicting the first day in the test dataset. We add the predicted data to predictions list, then add the real data for the day that was just predicted to the training data. Next we remake the model and use it to predict the next day from the test data. This process repeats until all the days from the test data have had a prediction.

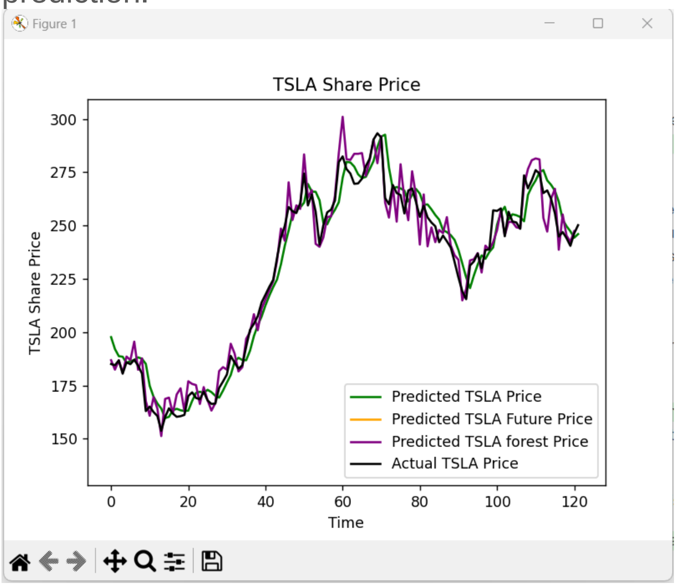
Forest loop function

```
    # loop for running the prediction on a number of days
def forest_loop(test, train):
    predictions = list()
    # seed history with training dataset
    history = [x for x in train]
    # step over each time-step in the test set
    for i in range(len(test)):
        # split test row into input and output columns
        testX = test[i, :-1]
        # fit model on history and make a prediction
        forestPrediction = forest_prediction(history, testX)
        # store forecast in list of predictions
        predictions.append(forestPrediction)
    return predictions
```

Prediction sub-function

```
    # fit an random forest model and make a one step prediction
def forest_prediction(train, testX):
    # turn list into an array
    train = asarray(train)
    # split into input and output columns
    trainX, trainy = train[:, :-1], train[:, -1]
    # make model
    model = RandomForestRegressor(n_estimators=FOREST_ESTIMATORS)
    model.fit(trainX, trainy)
    # make a single prediction
    forestPrediction = model.predict([testX])
    return forestPrediction[0]
```

A graph of the random forest prediction, compared with the actual data and LSTM prediction.



As you can see it works alright, but not as good as the LSTM model

Subtask 2:

For the next subtask, I decided to try adding the prophet model to my program. Prophet is an open source library that is created by facebook and made to do automatic forecasting of time series data. Unlike the random forest model, this all runs in a single function, which can be seen below.

```
def runTestProphet():
    #Get pre-split data
    # Only need to extract Close because the date is the index, and as such is automatically transfered over too
    test = fullData['Close']
    # reset index because it turns it back into a normal column which is referenced later
    test = test.reset_index()
    # turn date and close column into ds and y (necessary for prophet to work)
    test.columns = ['ds', 'y']
    # make sure ds column is a datatime
    test['ds']= pd.to_datetime(test['ds'])
    # remove offset days from the training data
    train = test.drop(test.index[-PROPHET_TRAIN_OFFSET:])

    # make the prophet model
    model = Prophet()
    # train the model from the train data
    model.fit(train)
    # setup dataframe from only the date date
    futuredays = list()
    futuredays = test['ds']
    futuredays = pd.DataFrame(futuredays)

    # use the model to make a forecast from the futuredays datetime range
    forecast = model.predict(futuredays)
    # set the actual and predicted values
    actualData = test['y'][-len(forecast):].values
    prophetPrediction = forecast['yhat'].values
    # plot actual vs Predicted Data
    plt.plot(actualData, label='Actual')
    plt.plot(prophetPrediction, label='Predicted')
    plt.legend()
    plt.show()
```

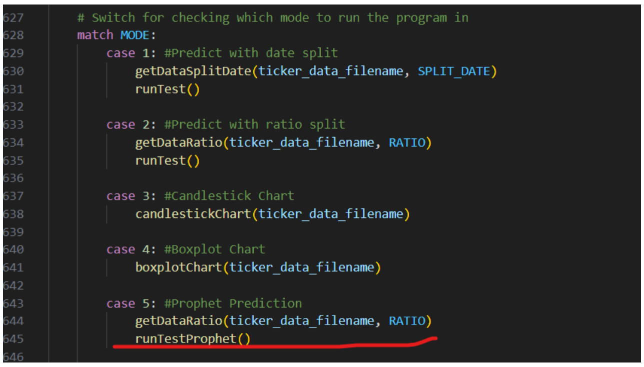
The first section of the function is transforming the data into a usable format for prophet, as prophet requires the data to be inputted in a specific format. This format is that the first column is for the datetime and must be named ‘ds’ while the second column contains all the observation data and must be named ‘y’.

You may notice that the input data is not ‘traindata’ and ‘testdata’ like it was for all the other functions. This is because it is not ideal for the data to be split by a ratio or split date. Instead, the training data is taken from the main dataset. Instead the prophet model is trained from the majority of the available data, except for a small portion that is withheld (the length of which is decided by the PROPHET_TRAIN_OFFSET from parameters). The next section of the function is where the prophet model is made. Then the futuredays dataframe is made, which is a list of the dates from the whole dataset. This list is then used as the input for the number of days for the prophet model to predict, and the predictions are then added to this dataframe. The predictions are then extracted from the dataframe and outputted onto a graph along with the real data. This prediction model is run separate from the other functions due to the difference in length of days the prediction is made from. As such it is run using a sperate mode from the other functions as can be seen here

From parameters:

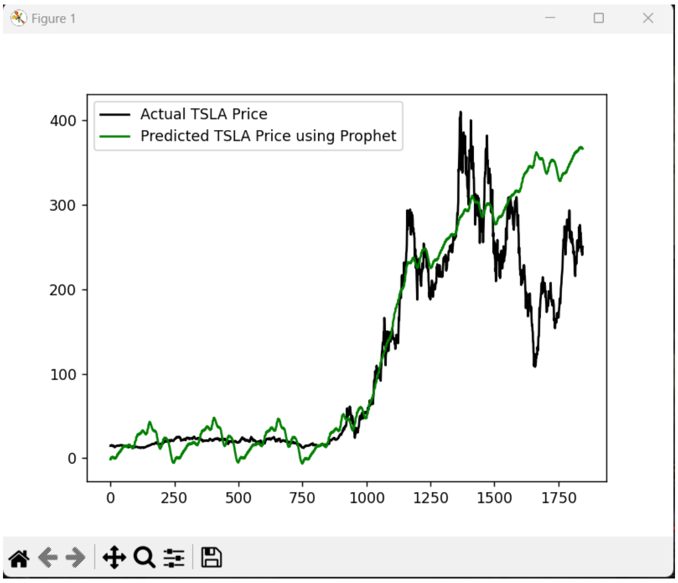
```
MODE = 5
# 1 = Split dataset into train/test sets by date, then predict
# 2 = Split dataset into train/test sets by ratio, then predict
# 3 = Make candlestick chart of data from past NDAYS
# 4 = Make boxplot chart of data from past NDAYS
# 5 = Run Prediction with Prophet model
# Other = Split dataset into train/test sets randomly, then predict
```

From the main function’s switch:



```
# Switch for checking which mode to run the program in
match MODE:
    case 1: #Predict with date split
        getDataSplitDate(ticker_data_filename, SPLIT_DATE)
        runTest()

    case 2: #Predict with ratio split
        getDataRatio(ticker_data_filename, RATIO)
        runTest()

    case 3: #Candlestick Chart
        candlestickChart(ticker_data_filename)

    case 4: #Boxplot Chart
        boxplotChart(ticker_data_filename)

    case 5: #Prophet Prediction
        getDataRatio(ticker_data_filename, RATIO)
        runTestProphet()
```

When plotted onto a graph and compared with the actual data, it can be seen below that though it does broadly follow the actual data, it clearly doesn't do a good job of predicting the data.



References:

Brownlee, J 2020, *Random Forest for Time Series Forecasting*, viewed 25/10/2023, <https://machinelearningmastery.com/random-forest-for-time-series-forecasting/>.

Brownlee, J 2020, *Time Series Forecasting With Prophet in Python* 25/10/2023, <https://machinelearningmastery.com/time-series-forecasting-with-prophet-in-python/>