Title: Task B-4 Report

Author: Robin Findlay-Marks, s103603871

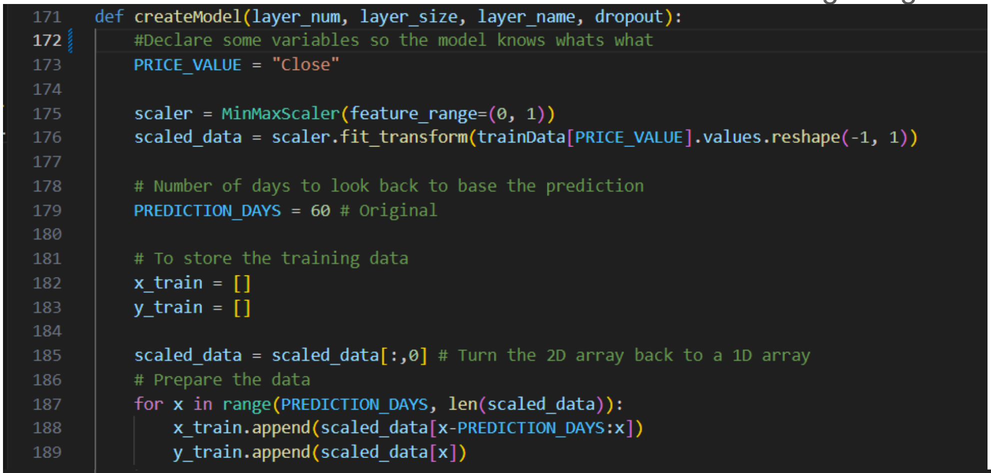
Task information:

Subtask 1:

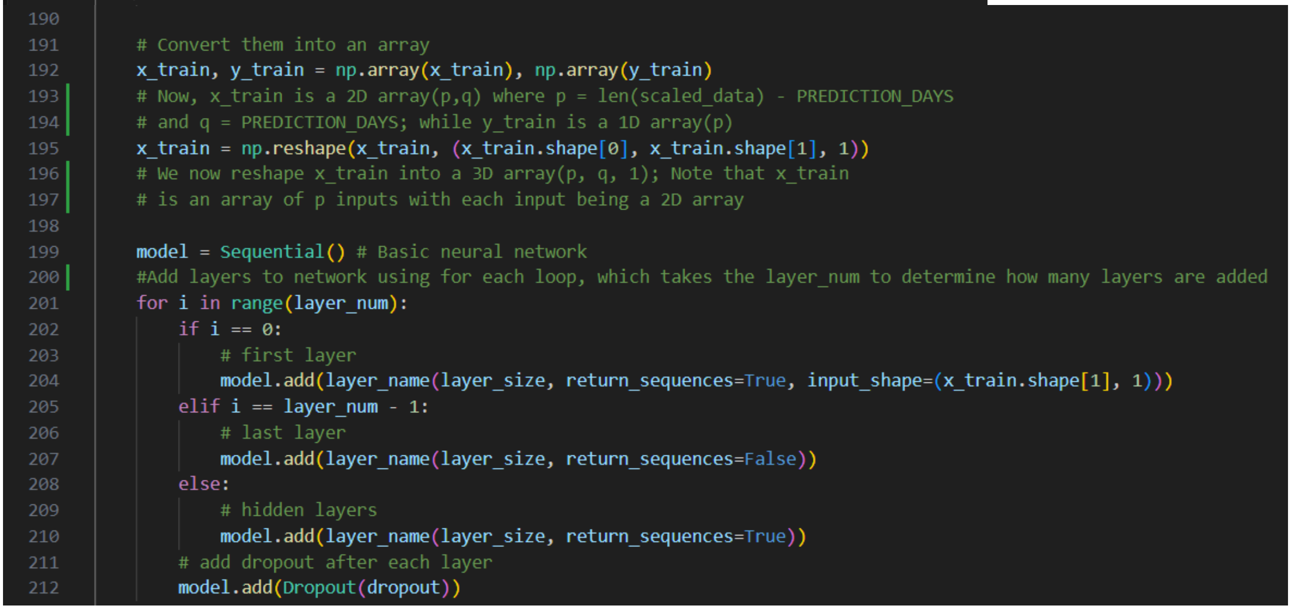
This subtask was surprisingly easy as I could use P1 as a guide for how to write this function. The first thing I did was separate the model creation code into it's own function. After a little bit of tweaking to make sure the it still worked I moved onto adding the function parameters. For this I used the ones listed in the assignment task; number of layers (layer_num), layer size (layer_size), deep learning network type (layer_name) as well as dropout rate (dropout).

There are only a few real differences from the original code. The first one of course is that parameters from the parameters.py file are being used, rather than the hard coded ones that were previously present. Another difference is that the layers are now being generated in a for each loop, dependent on the parameters.py and can now handle as many new layers as requested (even if that may take a lot more time to process), rather than being again, hard coded.

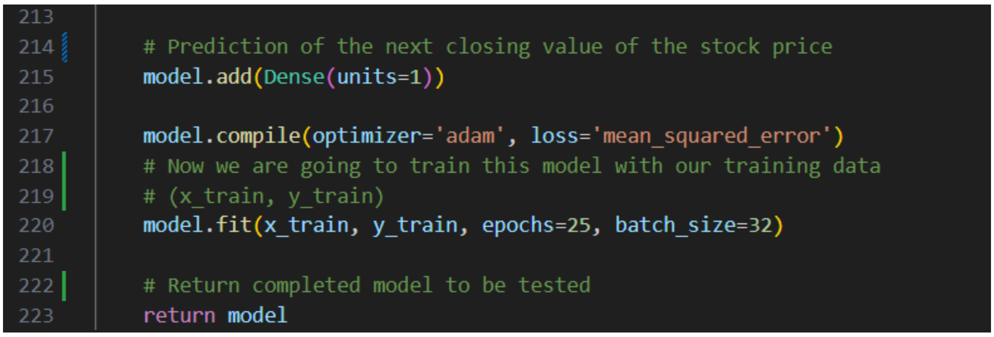
The code for the new function can be found in the following images:



```python
def createModel(layer_num, layer_size, layer_name, dropout):
    #Declare some variables so the model knows whats what
    PRICE_VALUE = "Close"

    scaler = MinMaxScaler(feature_range=(0, 1))
    scaled_data = scaler.fit_transform(trainData[PRICE_VALUE].values.reshape(-1, 1))

    # Number of days to look back to base the prediction
    PREDICTION_DAYS = 60 # Original

    # To store the training data
    x_train = []
    y_train = []

    scaled_data = scaled_data[:,0] # Turn the 2D array back to a 1D array
    # Prepare the data
    for x in range(PREDICTION_DAYS, len(scaled_data)):
        x_train.append(scaled_data[x-PREDICTION_DAYS:x])
        y_train.append(scaled_data[x])
```

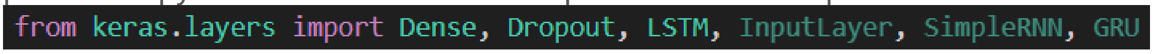


```python

    # Convert them into an array
    x_train, y_train = np.array(x_train), np.array(y_train)
    # Now, x_train is a 2D array(p,q) where p = len(scaled_data) - PREDICTION_DAYS
    # and q = PREDICTION_DAYS; while y_train is a 1D array(p)
    x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))
    # We now reshape x_train into a 3D array(p, q, 1); Note that x_train
    # is an array of p inputs with each input being a 2D array

    model = Sequential() # Basic neural network
    #Add layers to network using for each loop, which takes the layer_num to determine how many layers are added
    for i in range(layer_num):
        if i == 0:
            # first layer
            model.add(layer_name(layer_size, return_sequences=True, input_shape=(x_train.shape[1], 1)))
        elif i == layer_num - 1:
            # last layer
            model.add(layer_name(layer_size, return_sequences=False))
        else:
            # hidden layers
            model.add(layer_name(layer_size, return_sequences=True))
        # add dropout after each layer
        model.add(Dropout(dropout))
```



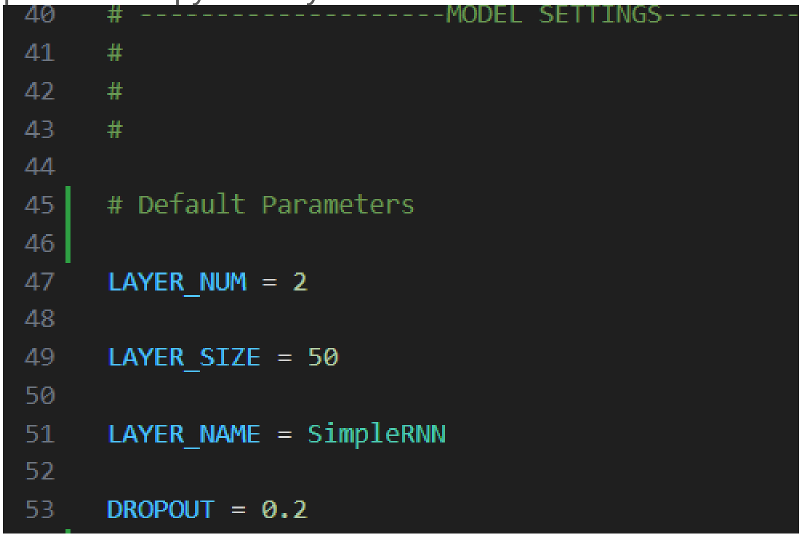
```

    # Prediction of the next closing value of the stock price
    model.add(Dense(units=1))

    model.compile(optimizer='adam', loss='mean_squared_error')
    # Now we are going to train this model with our training data
    # (x_train, y_train)
    model.fit(x_train, y_train, epochs=25, batch_size=32)

    # Return completed model to be tested
    return model
```

Subtask 2:
I believe this subtask to just a further implementation of the first subtask by doing two things. The first is to get the program to run with different DL networks, such as LSTM, RNN and GRU. The program already runs LSTM, so no need to do anything there, the only things left to do is run RNN and GRU. The general structure of the model creation already supports this, especially since I replaced the LSTM with a parameter from the parameter.py file. All I had to do was import GRU and SimpleRNN as can be seen here.



```
from keras.layers import Dense, Dropout, LSTM, InputLayer, SimpleRNN, GRU
```

The next step is to add them into the parameters so they can be changed. I did this in parameters.py and my work can be seen here

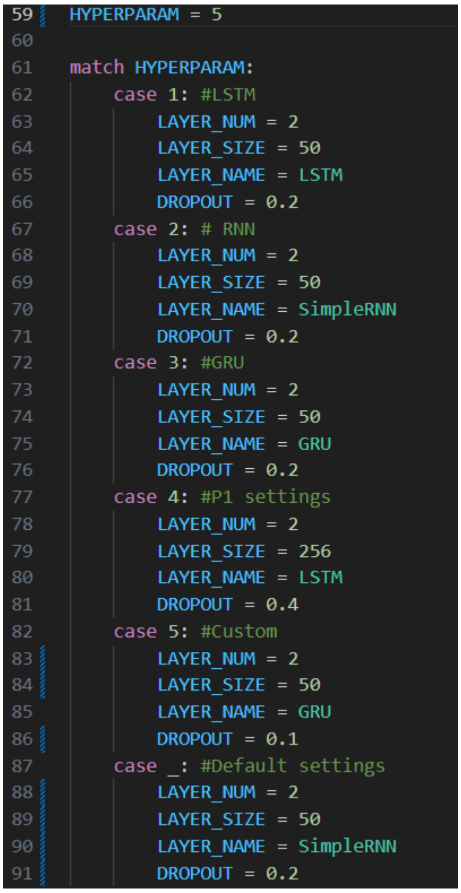


```
# --------------------MODEL SETTINGS---------
#
#
#

# Default Parameters

LAYER_NUM = 2

LAYER_SIZE = 50

LAYER_NAME = SimpleRNN

DROPOUT = 0.2
```

The second part of subtask 2 was I believe to make several hyperparameter configurations (like sets of previously declared parameters) which use different combinations of the parameters for the the DL network. To do this I made a simple switch which checks the HYPERPARAMETER variable in the parameters.py file and uses this to decide which parameter set to use.

For this I made 5 different hyperparameters. The first is the default LSTM, which is using the parameters that were used in v0.1, the second and third are the same but using SimpleRNN and GRU respectively, the fourth is LSTM but with the parameters from the P1 example code, and the final is my custom version, which uses the combination of parameters that I have done tests with to find the best one I can.

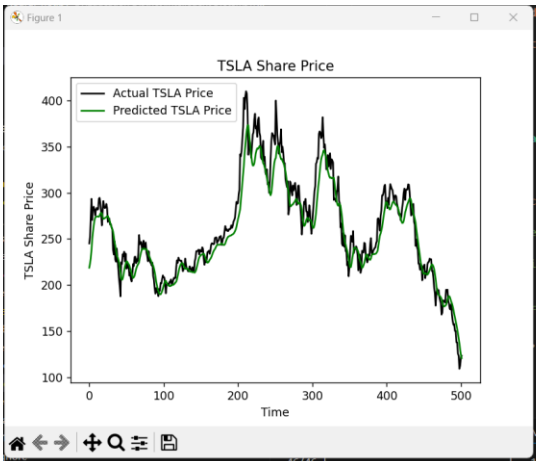
The code for the switch and hyperparameters can be seen below



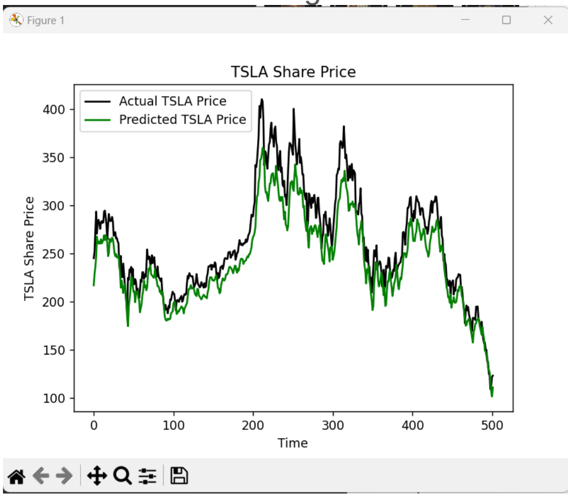
```
HYPERPARAM = 5

match HYPERPARAM:
    case 1: #LSTM
        LAYER_NUM = 2
        LAYER_SIZE = 50
        LAYER_NAME = LSTM
        DROPOUT = 0.2
    case 2: # RNN
        LAYER_NUM = 2
        LAYER_SIZE = 50
        LAYER_NAME = SimpleRNN
        DROPOUT = 0.2
    case 3: #GRU
        LAYER_NUM = 2
        LAYER_SIZE = 50
        LAYER_NAME = GRU
        DROPOUT = 0.2
    case 4: #P1 settings
        LAYER_NUM = 2
        LAYER_SIZE = 256
        LAYER_NAME = LSTM
        DROPOUT = 0.4
    case 5: #Custom
        LAYER_NUM = 2
        LAYER_SIZE = 50
        LAYER_NAME = GRU
        DROPOUT = 0.1
    case _: #Default settings
        LAYER_NUM = 2
        LAYER_SIZE = 50
        LAYER_NAME = SimpleRNN
        DROPOUT = 0.2
```

Below can be seen the graph results for the different setting presets

Default LSTM settings:

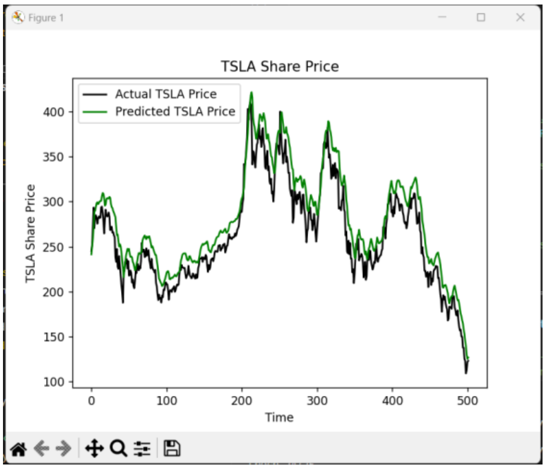


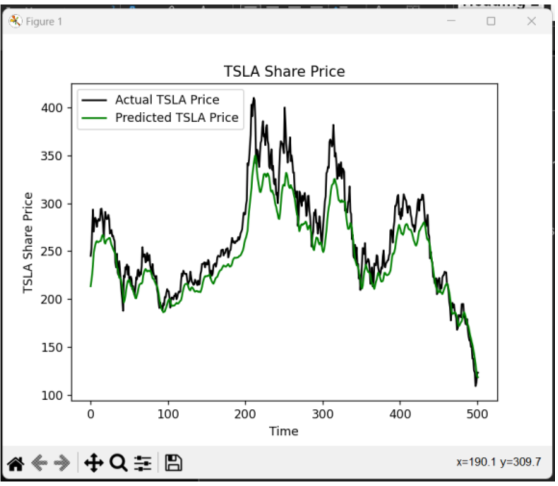
Default RNN settings:



Default GRU settings:

P1 Settings:





Robin’s custom settings:

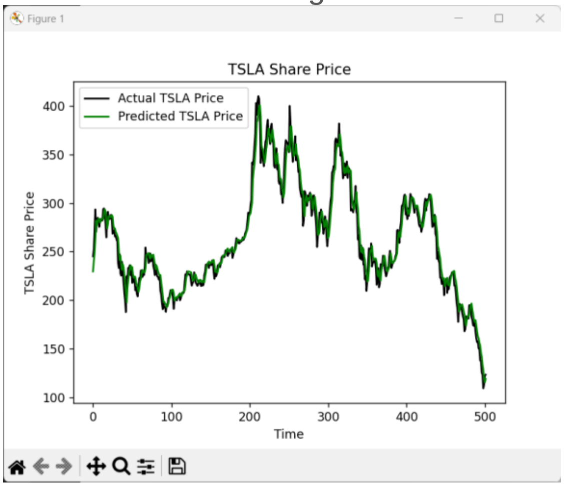


Figure 1
TSLA Share Price
Actual TSLA Price
Predicted TSLA Price
TSLA Share Price
400
350
300
250
200
150
100
0
100
200
300
400
500
Time