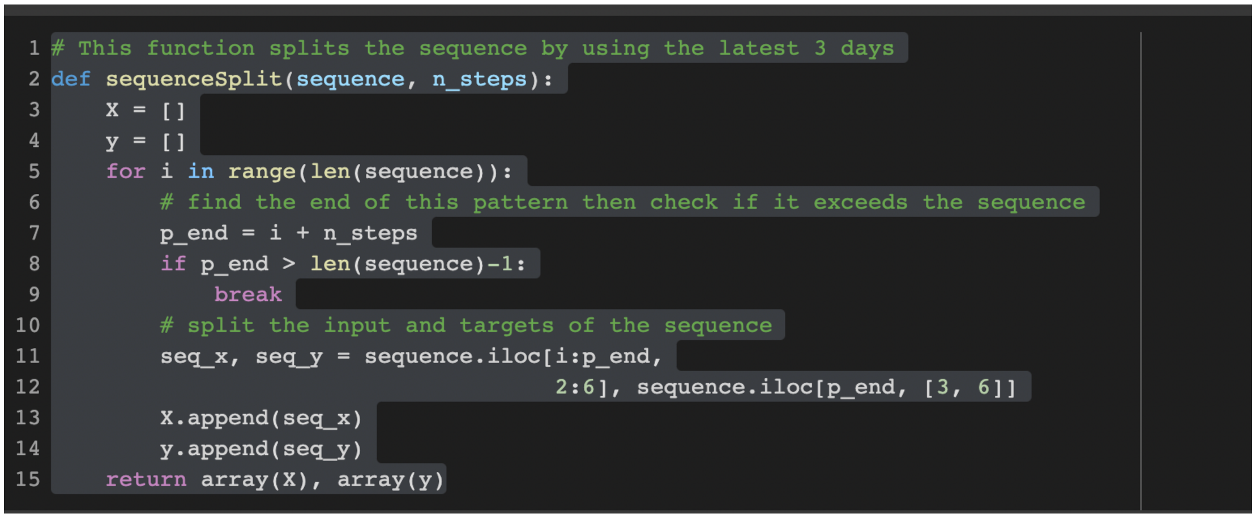
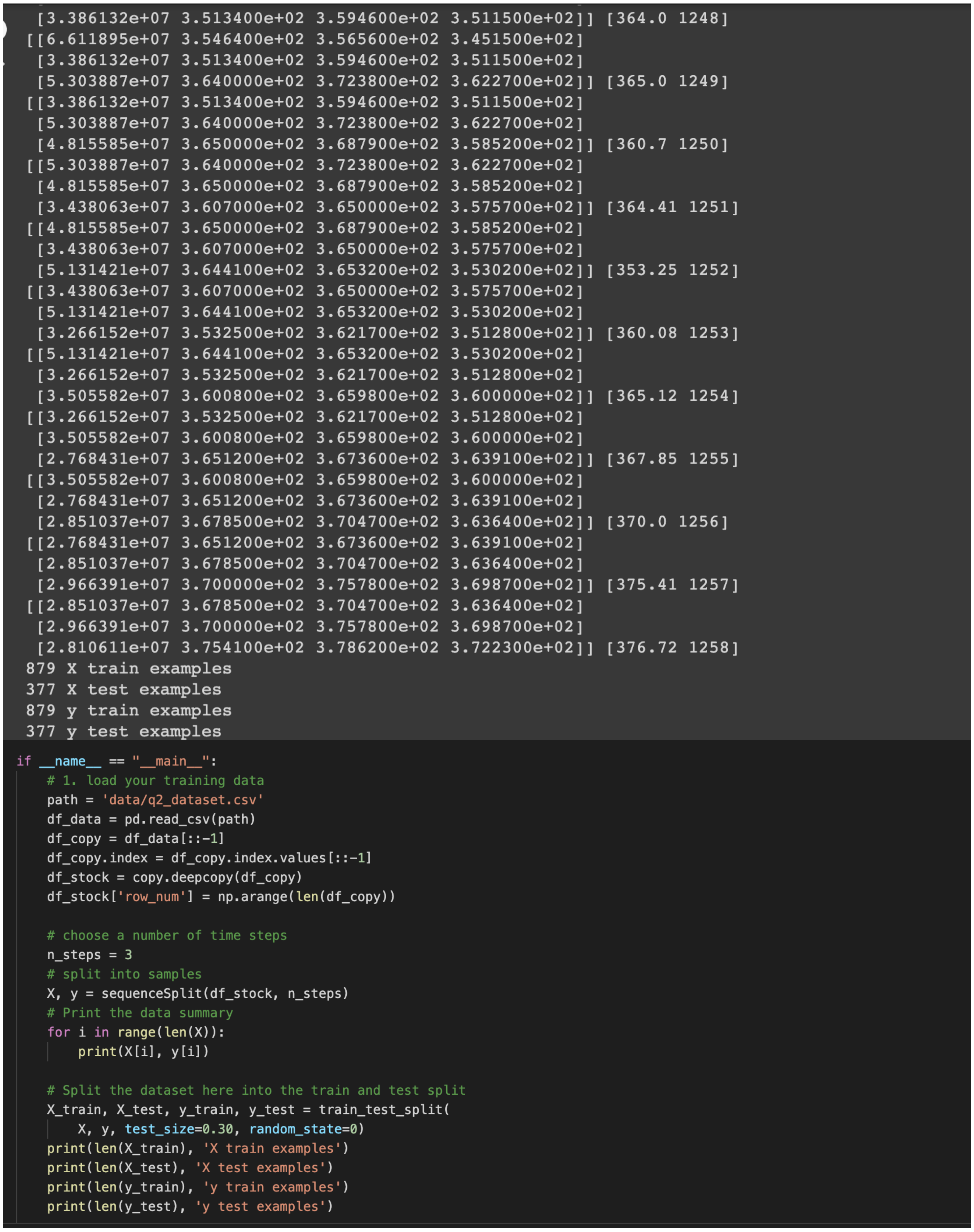
Question 2

IN CREATING THE DATASET

- A function was created called sequenceSplit which takes in the stock as the sequence and loops through it till the pattern exceeds the length of the sequence
- The number of steps here is the number of previous days which is set to 3 when the function is called
- Afterwards we call the call main function where the preprocessing is done.
- In the preprocessing, we read the data file and set the n_steps which is the number of days to 3.
- Afterwards we call the sequence function and plug in the variables.
- Train , test split is then used to split the dataset into the training and testing data where it is then saved in a csv file in the data folder



```python
# This function splits the sequence by using the latest 3 days
def sequenceSplit(sequence, n_steps):
    X = []
    y = []
    for i in range(len(sequence)):
        # find the end of this pattern then check if it exceeds the sequence
        p_end = i + n_steps
        if p_end > len(sequence)-1:
            break
        # split the input and targets of the sequence
        seq_x, seq_y = sequence.iloc[i:p_end,
                                     2:6], sequence.iloc[p_end, [3, 6]]
        X.append(seq_x)
        y.append(seq_y)
    return array(X), array(y)
```



```
 [3.386132e+07 3.513400e+02 3.594600e+02 3.511500e+02]] [364.0 1248]
[[6.611895e+07 3.546400e+02 3.565600e+02 3.451500e+02]
 [3.386132e+07 3.513400e+02 3.594600e+02 3.511500e+02]
 [5.303887e+07 3.640000e+02 3.723800e+02 3.622700e+02]] [365.0 1249]
[[3.386132e+07 3.513400e+02 3.594600e+02 3.511500e+02]
 [5.303887e+07 3.640000e+02 3.723800e+02 3.622700e+02]
 [4.815585e+07 3.650000e+02 3.687900e+02 3.585200e+02]] [360.7 1250]
[[5.303887e+07 3.640000e+02 3.723800e+02 3.622700e+02]
 [4.815585e+07 3.650000e+02 3.687900e+02 3.585200e+02]
 [3.438063e+07 3.607000e+02 3.650000e+02 3.575700e+02]] [364.41 1251]
[[4.815585e+07 3.650000e+02 3.687900e+02 3.585200e+02]
 [3.438063e+07 3.607000e+02 3.650000e+02 3.575700e+02]
 [5.131421e+07 3.644100e+02 3.653200e+02 3.530200e+02]] [353.25 1252]
[[3.438063e+07 3.607000e+02 3.650000e+02 3.575700e+02]
 [5.131421e+07 3.644100e+02 3.653200e+02 3.530200e+02]
 [3.266152e+07 3.532500e+02 3.621700e+02 3.512800e+02]] [360.08 1253]
[[5.131421e+07 3.644100e+02 3.653200e+02 3.530200e+02]
 [3.266152e+07 3.532500e+02 3.621700e+02 3.512800e+02]
 [3.505582e+07 3.600800e+02 3.659800e+02 3.600000e+02]] [365.12 1254]
[[3.266152e+07 3.532500e+02 3.621700e+02 3.512800e+02]
 [3.505582e+07 3.600800e+02 3.659800e+02 3.600000e+02]
 [2.768431e+07 3.651200e+02 3.673600e+02 3.639100e+02]] [367.85 1255]
[[3.505582e+07 3.600800e+02 3.659800e+02 3.600000e+02]
 [2.768431e+07 3.651200e+02 3.673600e+02 3.639100e+02]
 [2.851037e+07 3.678500e+02 3.704700e+02 3.636400e+02]] [370.0 1256]
[[2.768431e+07 3.651200e+02 3.673600e+02 3.639100e+02]
 [2.851037e+07 3.678500e+02 3.704700e+02 3.636400e+02]
 [2.966391e+07 3.700000e+02 3.757800e+02 3.698700e+02]] [375.41 1257]
[[2.851037e+07 3.678500e+02 3.704700e+02 3.636400e+02]
 [2.966391e+07 3.700000e+02 3.757800e+02 3.698700e+02]
 [2.810611e+07 3.754100e+02 3.786200e+02 3.722300e+02]] [376.72 1258]
879 X train examples
377 X test examples
879 y train examples
377 y test examples
```

```python
if __name__ == "__main__":
    # 1. load your training data
    path = 'data/q2_dataset.csv'
    df_data = pd.read_csv(path)
    df_copy = df_data[::-1]
    df_copy.index = df_copy.index.values[::-1]
    df_stock = copy.deepcopy(df_copy)
    df_stock['row_num'] = np.arange(len(df_copy))

    # choose a number of time steps
    n_steps = 3
    # split into samples
    X, y = sequenceSplit(df_stock, n_steps)
    # Print the data summary
    for i in range(len(X)):
        print(X[i], y[i])

    # Split the dataset here into the train and test split
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.30, random_state=0)
    print(len(X_train), 'X train examples')
    print(len(X_test), 'X test examples')
    print(len(y_train), 'y train examples')
    print(len(y_test), 'y test examples')
```

IN TRAINING THE DATASET

Various parameters were used and tested , while most of them yielded little to no result , the best we could do was to reduce overfitting. However, some considerable difference was noticed when we worked with some certain parameters

- **EPOCHS** were set to 8000 because trying previously to set to a much lower number had huge loss values with the current architecture and yielded a straight linear line(High Bias) for the testing output, so not much was learnt from the data with low epochs as low as 500.

- **In terms of the architecture, GRU (Gated Recurring Units)** is currently used and as per its documentation well GRU has two gates (*reset and update gate*). Previously, **LSTM (Long Short Term Memory)** was used and as per it's documentation LSTM has three gates (*input, output and forget gate*). Running the code with GRU was not so different form LSTM albeit it ran faster than LSTM and it is probably suitable for larger datasets however LSTM had the upper hand because it gave a little lower Mean Squared loss value at 900 compared to GRU at 893 mean squared error

- **Batch size of 32** seemed efficient to prevent overfitting, it took some time to train but at that capacity we could capture sequences in a robust manner. It did also contribute to reduce the loss as not using batch size yielded a loss of 902

- **The mean squared error** was used to calculate the losses along the way.

- **Linear activation function** is currently used. Previously, sigmoid, and tangent functions were used but it had similar issues such that the weights did not change much(Possible vanishing gradients). Rectified Linear Activation function was used and it wasn't so different from Linear activation function. Linear activation function helps model regression and in this case scenario the stock prediction is similar to a continuous regression.

- The **return sequences returns** the hidden state output for each input time step and was used previously but because we used one layer this time , we decided to go without it .

- Adam Optimizer is used which combines the best properties of the AdaGrad and RMSProp algorithms to provide an optimization algorithm that can handle sparse gradients on noisy problems

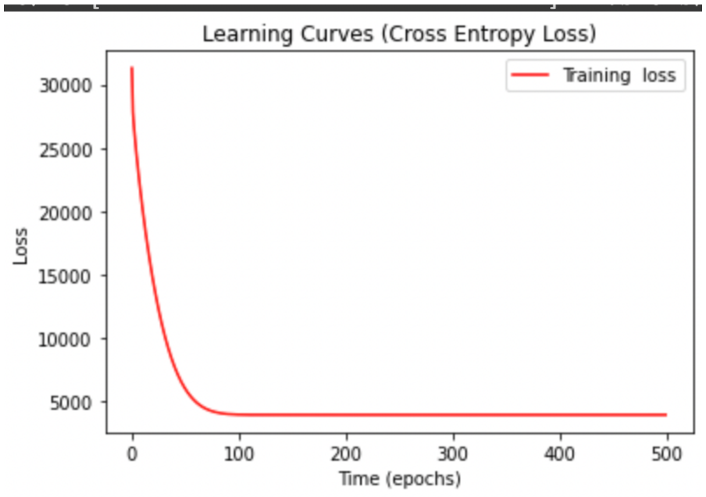
- **CURRENTLY** 40 nodes are being used

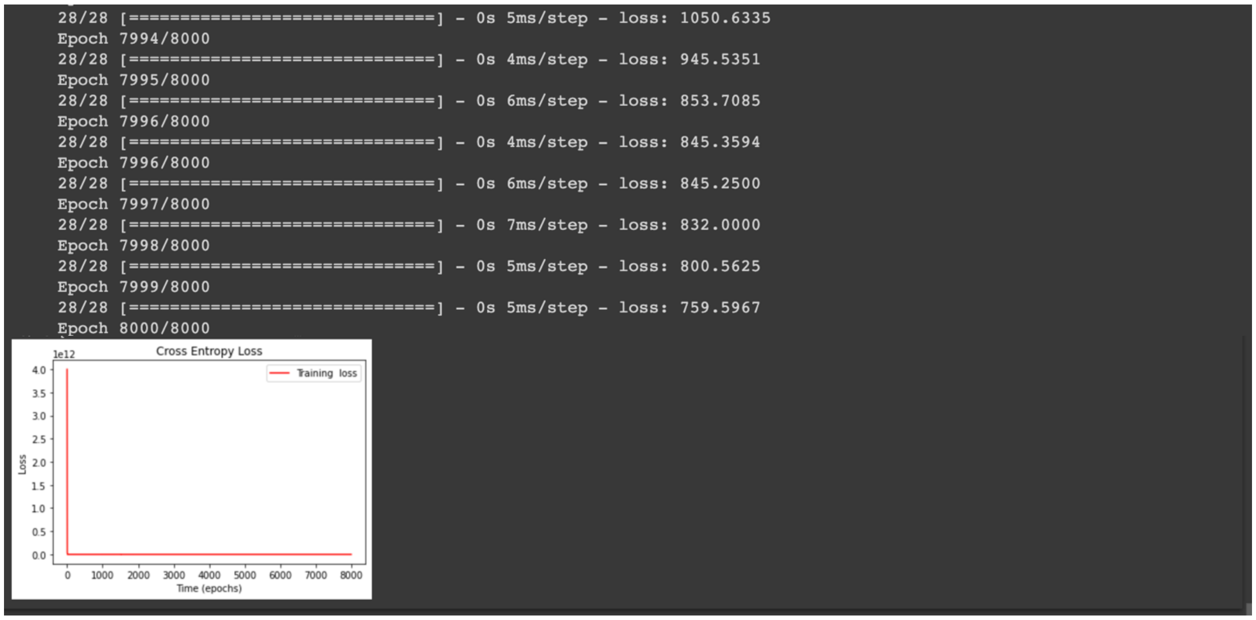
IN THE PAST

Using 128 nodes on the first layer and 64 neurons on the second layer made it possible to get a bit lower loss but it gave a high bias

Using 500 nodes on 3 LSTM layers took a lot longer time (possibly 8-10 minutes more) and overfitted heavily and a higher loss.

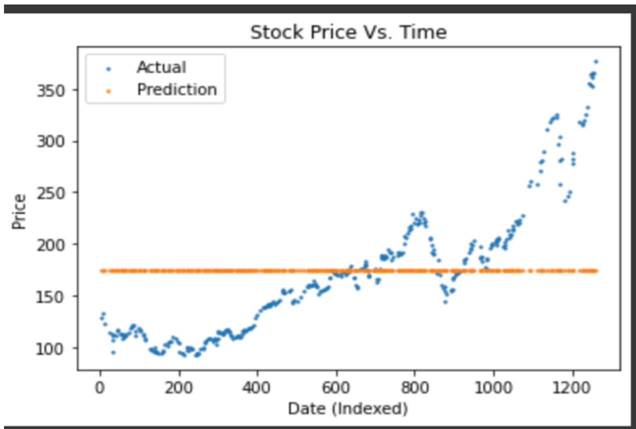
The 2 images below is the training epochs output for the previous network and final (Current) network respectively





```
28/28 [==============================] - 0s 5ms/step - loss: 1050.6335
Epoch 7994/8000
28/28 [==============================] - 0s 4ms/step - loss: 945.5351
Epoch 7995/8000
28/28 [==============================] - 0s 6ms/step - loss: 853.7085
Epoch 7996/8000
28/28 [==============================] - 0s 4ms/step - loss: 845.3594
Epoch 7996/8000
28/28 [==============================] - 0s 6ms/step - loss: 845.2500
Epoch 7997/8000
28/28 [==============================] - 0s 7ms/step - loss: 832.0000
Epoch 7998/8000
28/28 [==============================] - 0s 5ms/step - loss: 800.5625
Epoch 7999/8000
28/28 [==============================] - 0s 5ms/step - loss: 759.5967
Epoch 8000/8000
```

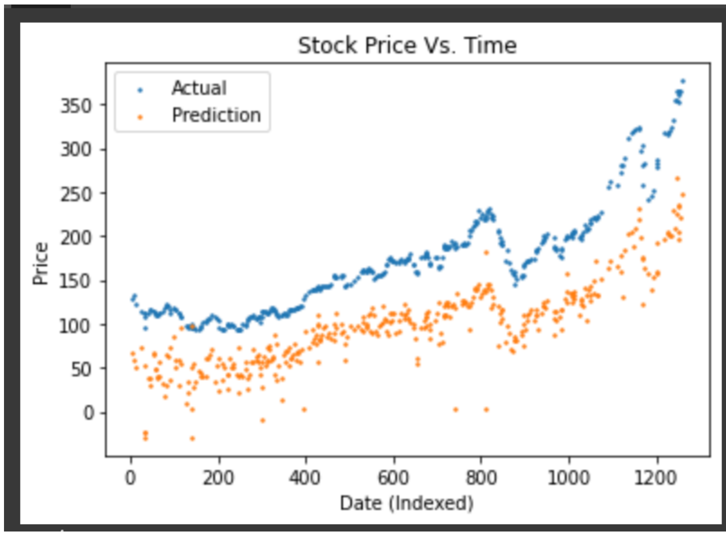
After TRAINING, THE MODEL WAS SAVED TO data/20923853_RNN_MODEL.h5



```
12/12 [==============================] - 1s 3ms/step - loss: 3805.5537
The loss of the test data is 3805.5537109375.
```

The prediction above was from the parameters I mentioned I used earlier in each part and yielded a high bias.

PREDICTION CHART



The prediction above is the final architecture and the prediction is close to the actual output.

- **Adding 5 days of data we noticed the loss increased towards 1000 then stagnated. Not much changed**