Introduction to Neural Networks Assignment 06
Stock Market Prediction (Don't Bet the Farm...)

Mangesh Sakordekar

Aim of this project is to use a stock market data to forecast stock market prices using Neural Net Models

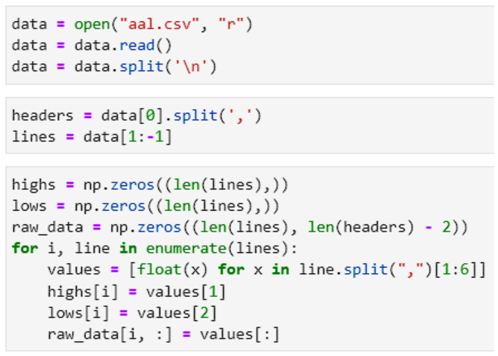
- Read and process stock data.
- Normalize the raw data.
- Validate using the naive model.
- Train neural net model to predict highs and lows of the stock.
- Predict and plot charts using the test dataset.
- Predict a different stock using the trained models.
- Define an investment strategy.
- Calculate possible profits/losses following the strategy.

Datasets

American Airlines (AAL) stock data was used to train, validate and test the model. The model was also tested on Delta Airlines (DAL) data.

Reading the data

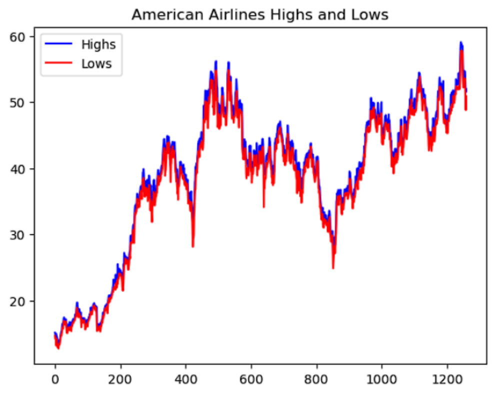
Sections were cut out from all_stocks_5yr.csv to make separate csv files fro each stock. The data was read in, processed and stored in an numpy array. Numpy arrays were created for the labels (highs and lows).



```
data = open("aal.csv", "r")
data = data.read()
data = data.split('\n')
```

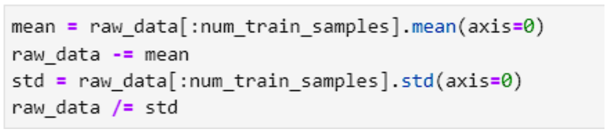
```
headers = data[0].split(',')
lines = data[1:-1]
```

```
highs = np.zeros((len(lines),))
lows = np.zeros((len(lines),))
raw_data = np.zeros((len(lines), len(headers) - 2))
for i, line in enumerate(lines):
    values = [float(x) for x in line.split(",")[1:6]]
    highs[i] = values[1]
    lows[i] = values[2]
    raw_data[i, :] = values[:]
```



Normalizing the Data

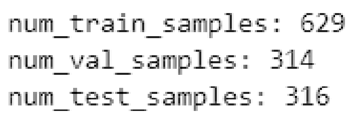
Each of the 5 features - open, high, low, close, volume - were normalized by subtracting the mean and divining by the standard deviation.



```
mean = raw_data[:num_train_samples].mean(axis=0)
raw_data -= mean
std = raw_data[:num_train_samples].std(axis=0)
raw_data /= std
```

Splitting the Data

The data was roughly split into 50-25-25 split. With total of 1259 samples, training dataset had 629 samples, validation data had 314 samples and test data had 316 samples.



```
num_train_samples: 629
num_val_samples: 314
num_test_samples: 316
```

Training The Models

Creating Time Series Data

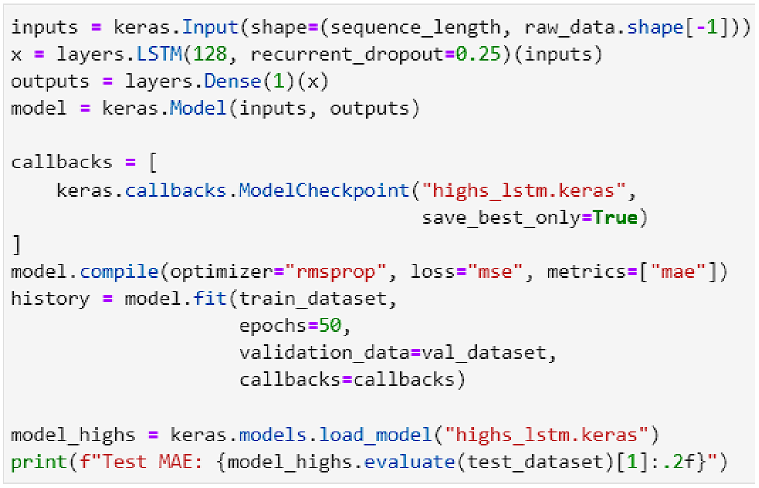
Time series data was created using the using the timeseries_dataset_from_array from the keras library. Since the readings are just one per day, the sampling rate was set to 1. The model uses 3 weeks of previous data to predict, thus the sequence length used was 21. Since we are predicting for the current day and not into the future, the delay is just the product of sampling rate and sequence length. The batch size was set to 4.

Naive Method

The naive method predicted the last element in the timeseries sequence array as the current value. This method was tested on the test and validation datasets for predicting highs and lows. Mean absolute error predicting the highs was approximately 0.56 for validation dataset and 0.62 for the test dataset. For the lows, the MAE for validation was 1.16 and for test data was 1.12.

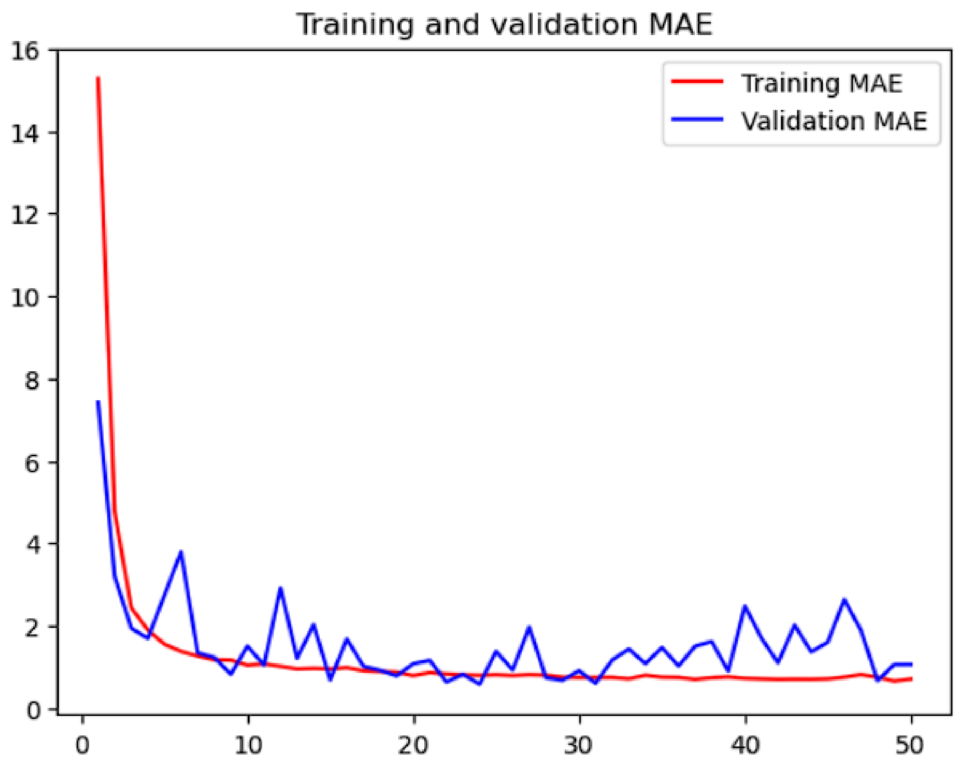
The Models

Both the models for highs and lows, have 1 hidden layer. This later is a LTSM layer with 128 neurons with a recurrent dropout rate of 0.25. The optimizer used is rmsprop, the loss is set to mean squared error and the metrics used is mean squared error. The models were trained fro 50 epochs.



```
inputs = keras.Input(shape=(sequence_length, raw_data.shape[-1]))
x = layers.LSTM(128, recurrent_dropout=0.25)(inputs)
outputs = layers.Dense(1)(x)
model = keras.Model(inputs, outputs)

callbacks = [
    keras.callbacks.ModelCheckpoint("highs_lstm.keras",
                                    save_best_only=True)
]
model.compile(optimizer="rmsprop", loss="mse", metrics=["mae"])
history = model.fit(train_dataset,
                    epochs=50,
                    validation_data=val_dataset,
                    callbacks=callbacks)

model_highs = keras.models.load_model("highs_lstm.keras")
print(f"Test MAE: {model_highs.evaluate(test_dataset)[1]:.2f}")
```

The model to predict highs achieved a test mean absolute error of 0.68 and the one predicting the lows had an MAE of 0.61. The training and validation MAE plots can be seen below.



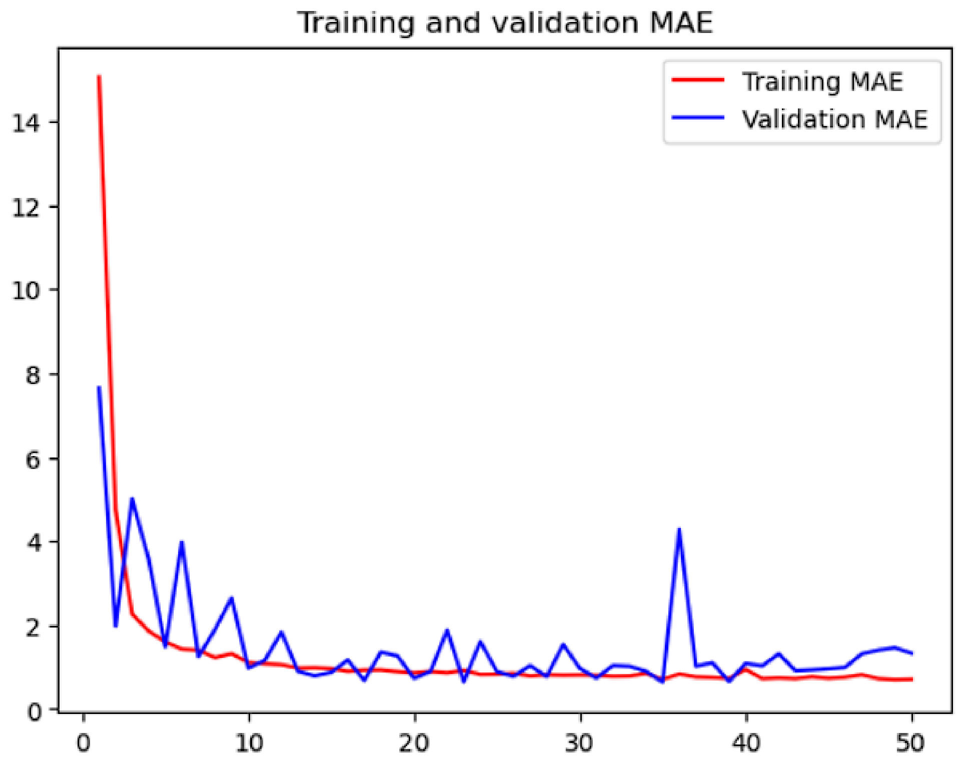
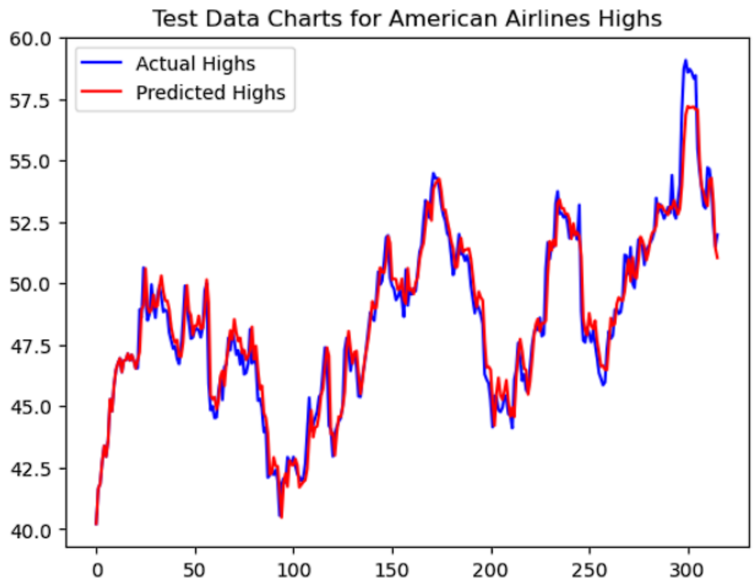


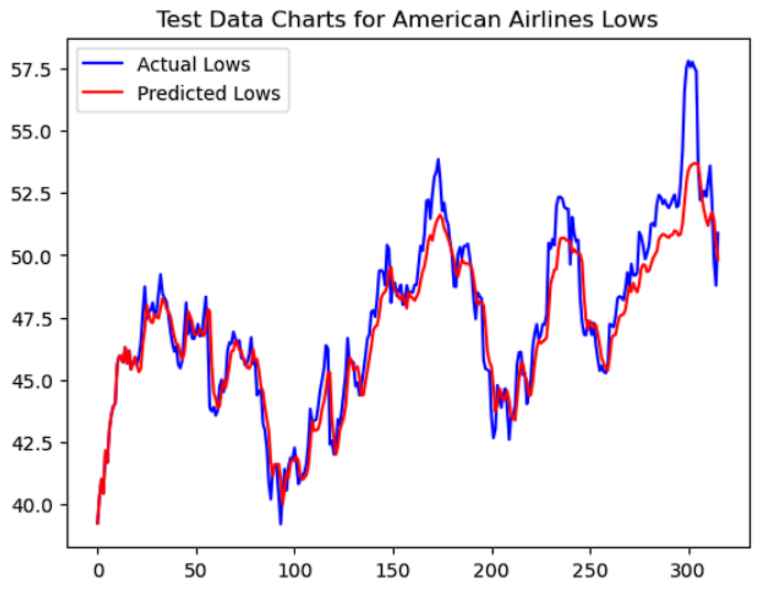
Figure 1: Top: MAE for highs Bottom: MAE for lows

Predicting Using The Models

American Airlines Data (316 Days)

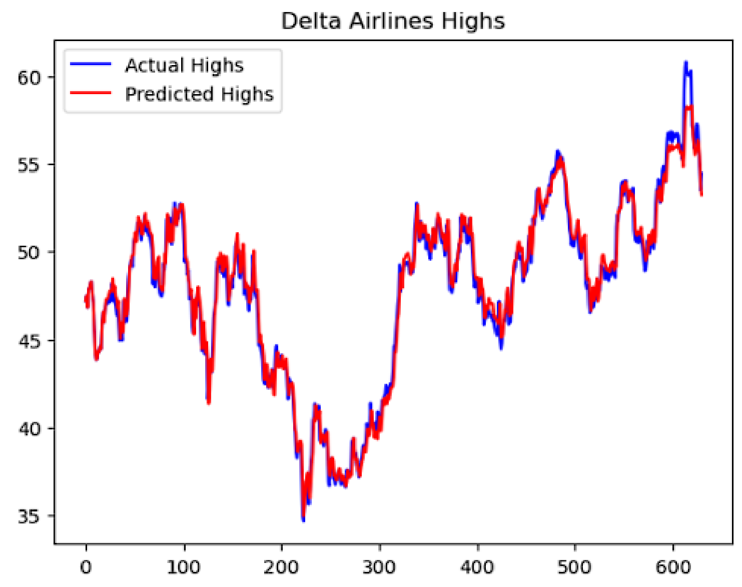
Highs and lows of the test dataset were predicted using the trained models over the course of last 316 days, the charts can be seen below

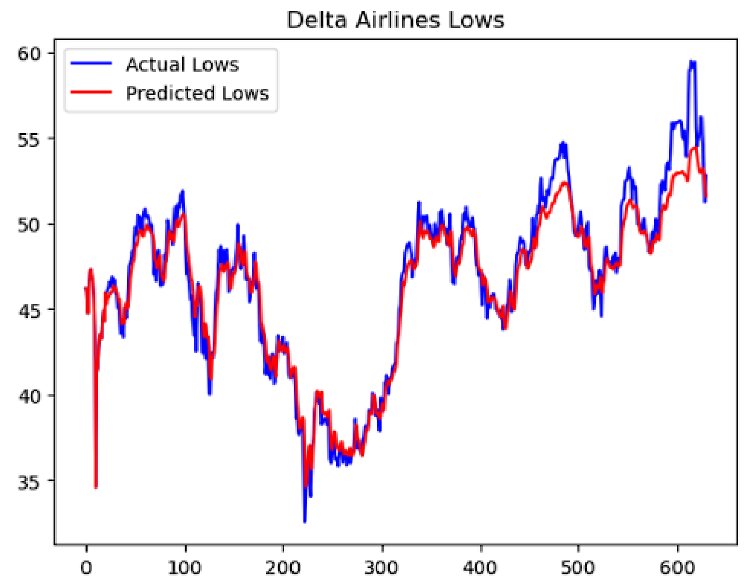




Delta Airlines Data (630 Days)

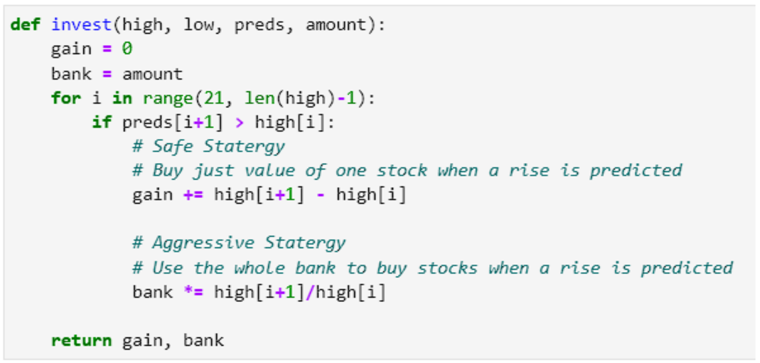
Highs and lows were predicted for the Delta Airline stocks using these models over the course of 630 days, the charts can be seen below. We only use the last 630 days as we want to avoid data from the same time period as training data as there might be some correlations between the price changes.





Investment Strategy

Only Using Highs

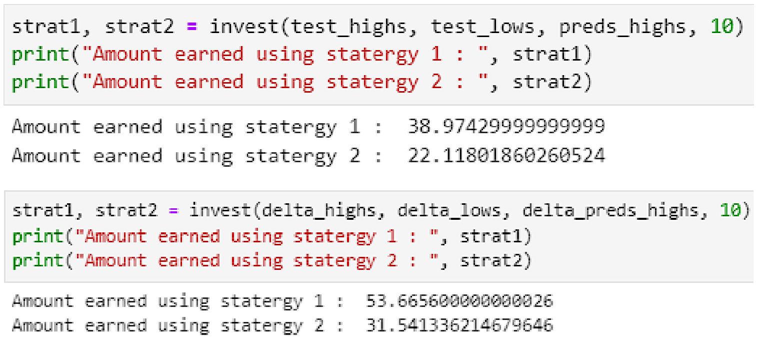


```python
def invest(high, low, preds, amount):
    gain = 0
    bank = amount
    for i in range(21, len(high)-1):
        if preds[i+1] > high[i]:
            # Safe Statergy
            # Buy just value of one stock when a rise is predicted
            gain += high[i+1] - high[i]

            # Aggressive Statergy
            # Use the whole bank to buy stocks when a rise is predicted
            bank *= high[i+1]/high[i]

    return gain, bank
```

Figure 2: Strategy 1 and 2 implementation

Strategy 1: Just 1 Stock
In this strategy, the predictions are used to make the decision. If the predicted high for the next day is grater than the we add the (high[i+1] - high[i]) to the profit. This strategy maximises the profit for owning only one stock at a time.

Strategy 2: Aggressive Investment
This strategy starts with a given balance amount. It uses similar strategy as the strategy 1 but is more risky. Given that the predictions made by the model are not 100% accurate, we will see this strategy yielding less profits compared to strategy 1.



```python
strat1, strat2 = invest(test_highs, test_lows, preds_highs, 10)
print("Amount earned using statergy 1 : ", strat1)
print("Amount earned using statergy 2 : ", strat2)
```

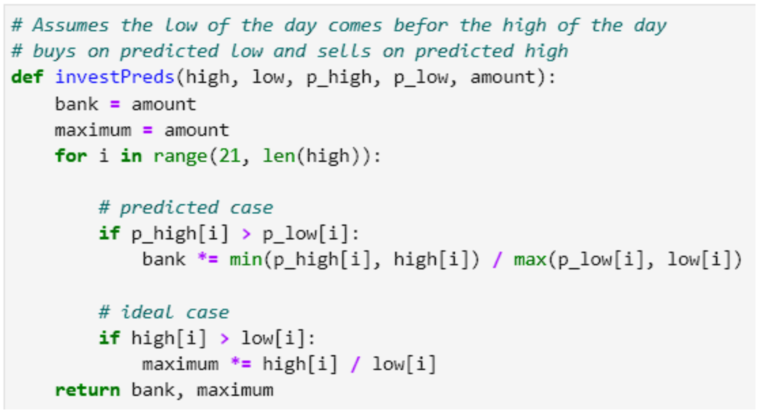
```
Amount earned using statergy 1 :  38.97429999999999
Amount earned using statergy 2 :  22.11801860260524
```

```python
strat1, strat2 = invest(delta_highs, delta_lows, delta_preds_highs, 10)
print("Amount earned using statergy 1 : ", strat1)
print("Amount earned using statergy 2 : ", strat2)
```

```
Amount earned using statergy 1 :  53.665600000000026
Amount earned using statergy 2 :  31.541336214679646
```

As we can see in the test data, strategy 1 yields more than strategy 2, thus proving our assumption that the predictions are not 100% accurate and it sometimes predicts a rise even when the stock falls. But, in the calculated profit for Delta airlines is higher is strategy 2.

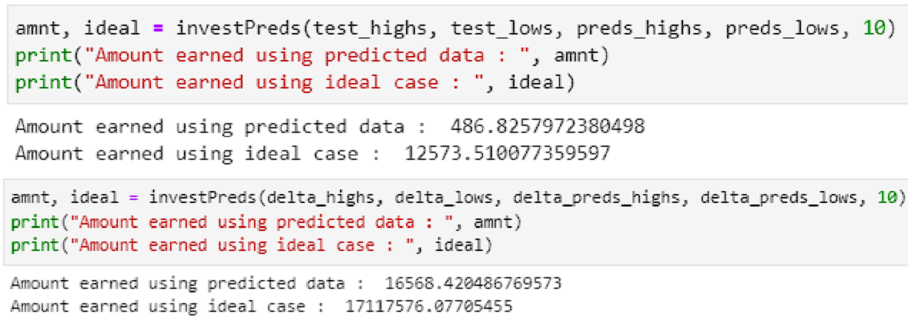
Using Highs and Lows



```
# Assumes the low of the day comes befor the high of the day
# buys on predicted low and sells on predicted high
def investPreds(high, low, p_high, p_low, amount):
    bank = amount
    maximum = amount
    for i in range(21, len(high)):

        # predicted case
        if p_high[i] > p_low[i]:
            bank *= min(p_high[i], high[i]) / max(p_low[i], low[i])

        # ideal case
        if high[i] > low[i]:
            maximum *= high[i] / low[i]
    return bank, maximum
```

Figure 3: Strategy 1 and 2 implementation

In this strategy we assume that the low of the day is always before high of the day. The basic idea is to buy during the low and sell when the stock is high. If the predicted low is greater than or equal to the predicted high, we conclude that the values predicted are inaccurate and we do not take any action. Otherwise, we buy at the stock at the predicted low and sell at the predicted high. To compare to the maximum possible profit, we also simulate buying at actual low and selling at actual high.



```
amnt, ideal = investPreds(test_highs, test_lows, preds_highs, preds_lows, 10)
print("Amount earned using predicted data : ", amnt)
print("Amount earned using ideal case : ", ideal)
```

```
Amount earned using predicted data :  486.8257972380498
Amount earned using ideal case :  12573.510077359597
```

```
amnt, ideal = investPreds(delta_highs, delta_lows, delta_preds_highs, delta_preds_lows, 10)
print("Amount earned using predicted data : ", amnt)
print("Amount earned using ideal case : ", ideal)
```

```
Amount earned using predicted data :  16568.420486769573
Amount earned using ideal case :  17117576.07705455
```

As we can see, this strategy performs better, but even if we compare it to the total possible profit, it only makes around 3% of it. Using this strategy and starting with 10 dollars, a profit of 476 investing in American Airlines for 316 days and a profit of 16558 dollars can be made investing in Delta Airlines over the course of 630 days.