Report

December 8, 2024

1 Programming Assignment 3 - Predicting Closing Costs With Google Stock Prices

---

Author: Taylor Larrechea

Class: CSPB 4622 - Machine Learning

Professor: Dr. Hoang Truong

Due Date: 12/9/2024

1.1 Project Topic

This project aims to train a deep learning model to predict the closing price of Google stock based on historical stock prices. The key problem that is present is that the stock market is a highly volatile and unpredictable environment. Along with this, training a deep learning model to predict stock prices can be difficult due to the high dimensionality of the data and the non-linear relationships between the features and the target variable as well as other factors.

The motivation behind this project is to create a model that can accurately predict the closing price of Google stock based on historical stock prices so that others can use this model to make informed decisions about buying or selling Google stock. If a model like this can be developed for Google, it could possibly be adapted for other stocks as well.

The dataset for this project can be found here.

1.2 Dataset

The data that was provided for this project consists of two CSV files:

- Test.csv: This file contains the test data that will be used to evaluate the model.
- Train.csv: This file contains the training data that will be used to train the model.

Both CSV files contain the following columns:

- Date: The date of the stock price.
- Open: The opening price of the stock on that day.
- High: The highest price of the stock on that day.

- Low: The lowest price of the stock on that day.
- Close: The closing price of the stock on that day.
- Volume: The volume of the stock on that day.

The training file contains 1258 rows and the test file contains 20 rows. This means that the model will be trained on 1258 days of stock prices and then evaluated on 20 days of stock prices.

[2]:
```python
# Imports

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense, Dropout
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
import os

# Load Data

training_data = pd.read_csv('Data/Train.csv')
testing_data = pd.read_csv('Data/Test.csv')

# Display Data

print("\nTrain Dataset Info:")
print(training_data.info())

print("\nTest Dataset Info:")
print(testing_data.info())

print("\nTrain Dataset Description:")

print(training_data.describe())

print("\nTest Dataset Description:")

print(testing_data.describe())

print("\nTrain Dataset Head:")

print(training_data.head())

print("\nTest Dataset Head:")

print(testing_data.head())
```

```
Train Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1258 entries, 0 to 1257
Data columns (total 6 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Date    1258 non-null   object
 1   Open    1258 non-null   float64
 2   High    1258 non-null   float64
 3   Low     1258 non-null   float64
 4   Close   1258 non-null   object
 5   Volume  1258 non-null   object
dtypes: float64(3), object(3)
memory usage: 59.1+ KB
None

Test Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20 entries, 0 to 19
Data columns (total 6 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Date    20 non-null     object
 1   Open    20 non-null     float64
 2   High    20 non-null     float64
 3   Low     20 non-null     float64
 4   Close   20 non-null     float64
 5   Volume  20 non-null     object
dtypes: float64(4), object(2)
memory usage: 1.1+ KB
None

Train Dataset Description:
              Open         High          Low
count  1258.000000  1258.000000  1258.000000
mean    533.709833   537.880223   529.007409
std     151.904442   153.008811   150.552807
min     279.120000   281.210000   277.220000
25%     404.115000   406.765000   401.765000
50%     537.470000   540.750000   532.990000
75%     654.922500   662.587500   644.800000
max     816.680000   816.680000   805.140000

Test Dataset Description:
             Open        High         Low       Close
count   20.000000   20.000000   20.000000   20.000000
mean   807.526000  811.926500  801.949500  807.904500
```

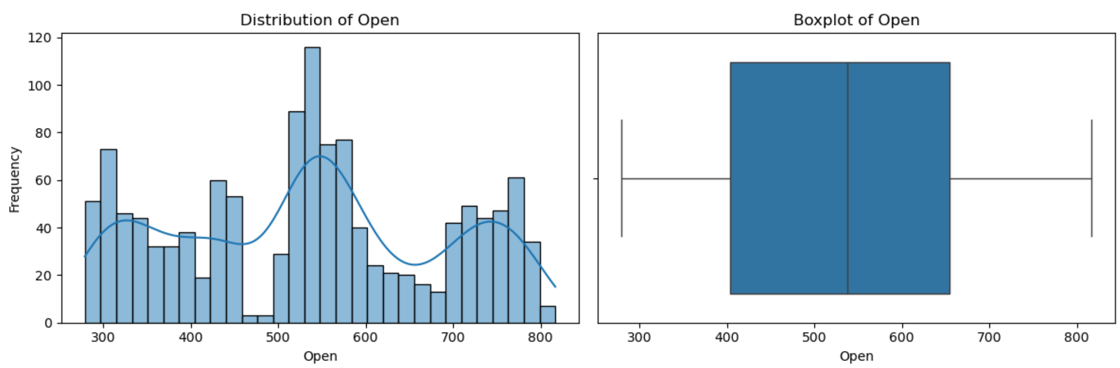
```
std     15.125428   14.381198   13.278607   13.210088
min    778.810000  789.630000  775.800000  786.140000
25%    802.965000  806.735000  797.427500  802.282500
50%    806.995000  808.640000  801.530000  806.110000
75%    809.560000  817.097500  804.477500  810.760000
max    837.810000  841.950000  827.010000  835.670000

Train Dataset Head:
       Date    Open    High     Low   Close      Volume
0  1/3/2012  325.25  332.83  324.97  663.59   7,380,500
1  1/4/2012  331.27  333.87  329.08  666.45   5,749,400
2  1/5/2012  329.83  330.75  326.89  657.21   6,590,300
3  1/6/2012  328.34  328.77  323.68  648.24   5,405,900
4  1/9/2012  322.04  322.29  309.46  620.76  11,688,800

Test Dataset Head:
       Date    Open    High     Low   Close     Volume
0  1/3/2017  778.81  789.63  775.80  786.14  1,657,300
1  1/4/2017  788.36  791.34  783.16  786.90  1,073,000
2  1/5/2017  786.08  794.48  785.02  794.02  1,335,200
3  1/6/2017  795.26  807.90  792.20  806.15  1,640,200
4  1/9/2017  806.40  809.97  802.83  806.65  1,272,400
```

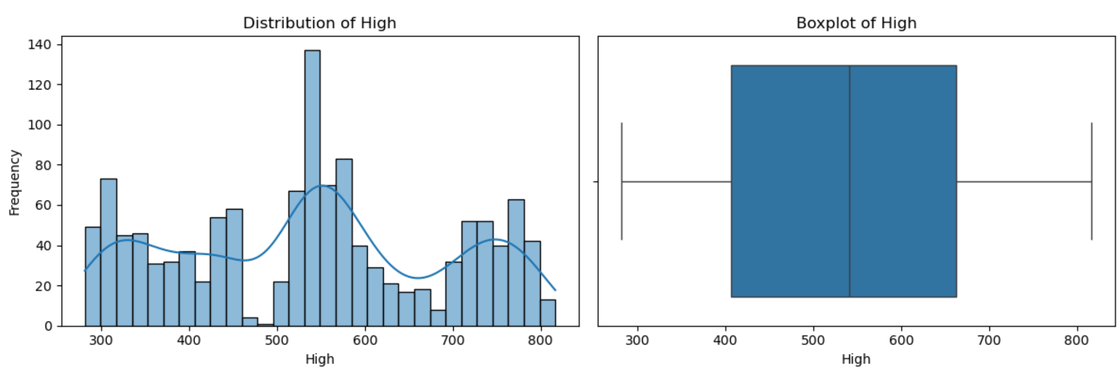
1.3 EDA

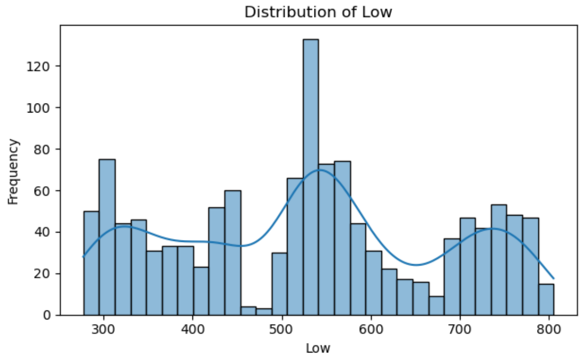
Now that we have the dataset, the next is to do some Exploratory Data Analysis (EDA) to understand the data better. We first begin by producing distribution and box plots for the Open, High, Low, Close, and Volume columns. Distribution plots were created so that we can see the distribution of the data and box plots were created so that we can see the spread of the data. Lastly, a correlation matrix was created to see how the features are correlated with each other.

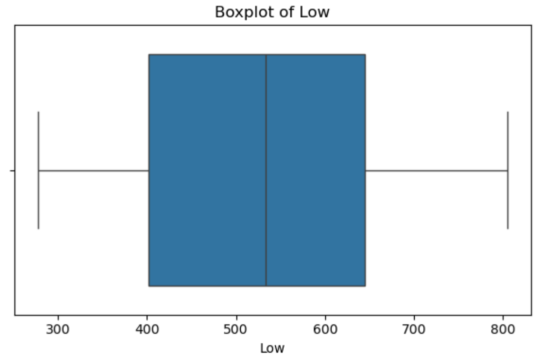
[8]:
```python
numerical_cols = ['Open', 'High', 'Low', 'Close', 'Volume']

# Ensure 'Date' column is not included in the conversion
training_data['Close'] = training_data['Close'].astype(str).str.replace(',',
 ↪'').astype(float)
training_data['Volume'] = training_data['Volume'].astype(str).str.replace(',',
 ↪'').astype(float)

# Distribution and Boxplots
for col in numerical_cols:
    plt.figure(figsize=(12, 4))

    # Distribution Plot
    plt.subplot(1, 2, 1)
    sns.histplot(training_data[col], kde=True, bins=30)
```

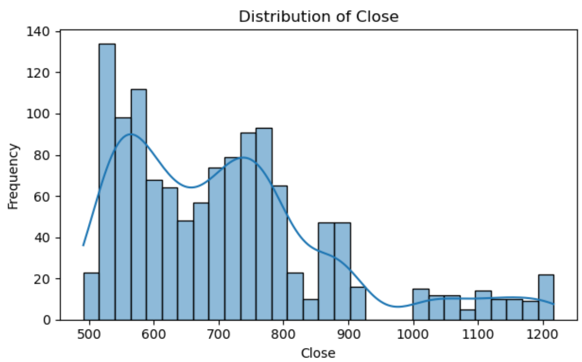
```
    plt.title(f"Distribution of {col}")
    plt.xlabel(col)
    plt.ylabel("Frequency")

    # Boxplot
    plt.subplot(1, 2, 2)
    sns.boxplot(x=training_data[col])
    plt.title(f"Boxplot of {col}")
    plt.xlabel(col)

    # Show combined plot
    plt.tight_layout()
    plt.show()

# Correlation heatmap
sns.heatmap(training_data.drop('Date', axis=1).corr(), annot=True, cmap='coolwarm')
plt.title("Correlation Heatmap")
plt.show()
```

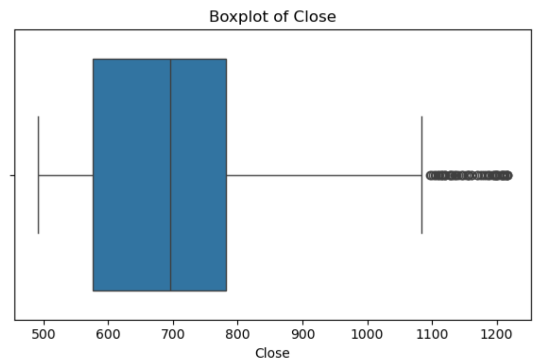


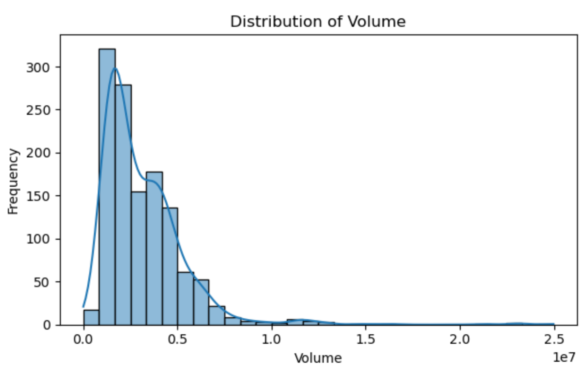


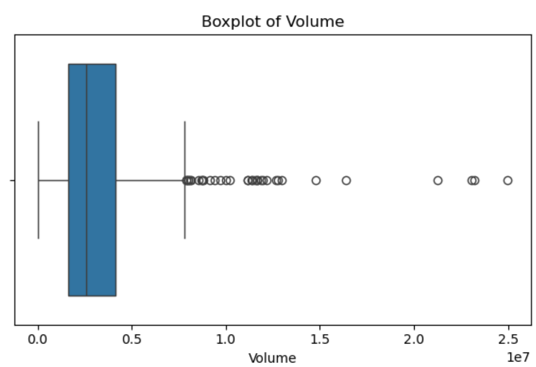

Distribution of Low
Frequency
0
20
40
60
80
100
120
300
400
500
600
700
800
Low

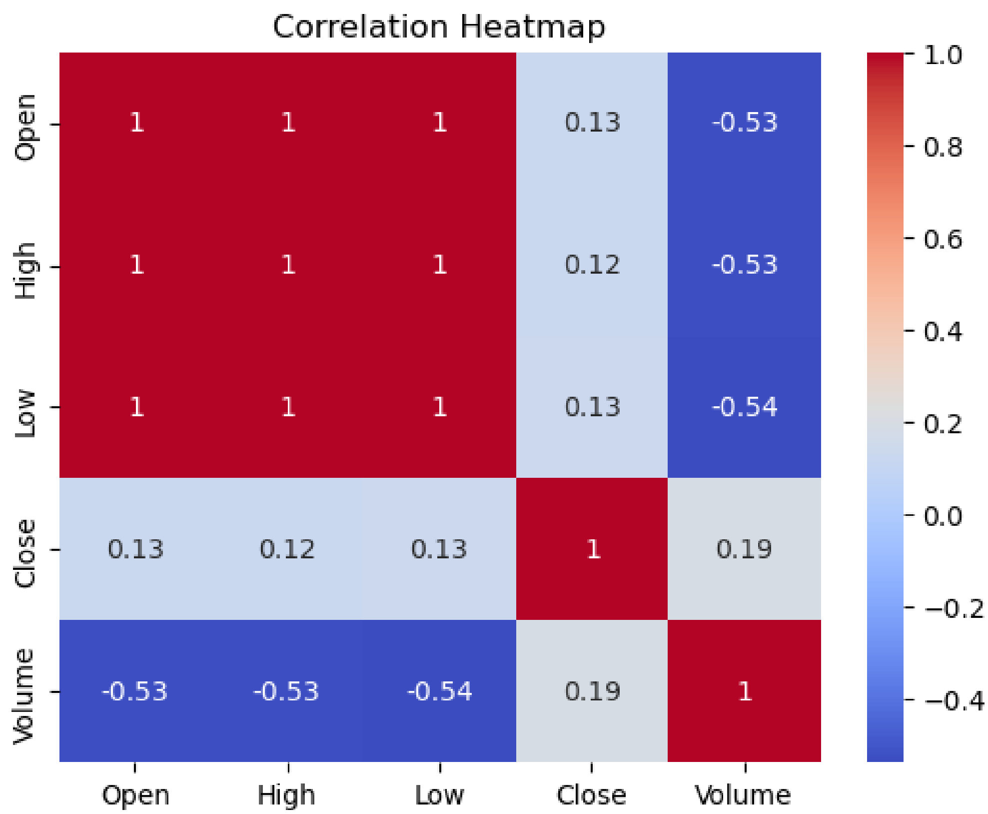

Boxplot of Low
300
400
500
600
700
800
Low


Distribution of Close
Frequency
0
20
40
60
80
100
120
140
500
600
700
800
900
1000
1100
1200
Close


Boxplot of Close
500
600
700
800
900
1000
1100
1200
Close


Distribution of Volume
Frequency
0
50
100
150
200
250
300
0.0
0.5
1.0
1.5
2.0
2.5
1e7
Volume


Boxplot of Volume
0.0
0.5
1.0
1.5
2.0
2.5
1e7
Volume



From the results above, if we are trying to predict the closing price of Google stock, it appears that the Volume and Open along with Low have the highest correlation with the closing price. From the results that we have, we can then decide to create a couple of models with different features to see which one performs the best. The models that we are going to create are:

- Model 1: Using all features
- Model 2: Using only the Open, Low, and Volume features
- Model 3: Using only the Volume and Open features
- Model 4: Using only the Open and Low features

We now are going to create these models and will evaluate them after.

1.4 Model

The models that are going to be created with a fully connected feedforward nueral network (FNN) or also known as Multilayer Perceptron (MLP). These model types are good for regression problems like the one that we have because they can learn complex non-linear relationships between the features and the target variable. A basic architecture for the each individual model was created for easy use and evaluation.

[ ]:

```
# Dictionary to store the features for each model
features_dictionary = {
    "Model1": training_data.columns.drop(['Date', 'Close']),
    "Model2": training_data.columns.drop(['Date', 'Close', 'High']),
    "Model3": training_data.columns.drop(['Date', 'Close', 'High', 'Low']),
    "Model4": training_data.columns.drop(['Date', 'Close', 'High', 'Low',␣
↪'Open']),
}

# Split the data into features and target
X = training_data.drop(['Date', 'Close'], axis=1)
Y = training_data['Close']

# Normalize the data
scaler = MinMaxScaler()
Scaled_X = scaler.fit_transform(X)

# Map column names to indices for feature selection
column_indices = {col: idx for idx, col in enumerate(X.columns)}

# Split the data
X_train, X_test, Y_train, Y_test = train_test_split(Scaled_X, Y, test_size=0.2,␣
↪random_state=42)

# Base Model Class
class DeepLearningModel:
    def __init__(self, model_name, selected_features):
        self.model_name = model_name
        self.selected_features = selected_features
        self.feature_indices = [column_indices[f] for f in selected_features]
        self.model = self._build_model(len(self.feature_indices))

    def _build_model(self, input_dim):
        model = Sequential([
            Dense(64, activation='relu', input_dim=input_dim),
            Dropout(0.2),
            Dense(32, activation='relu'),
            Dense(1, activation='linear')  # Assuming a regression task
        ])
        model.compile(optimizer='adam', loss='mse', metrics=['mae'])
        return model

    def train(self, X_train, y_train, epochs=50, batch_size=32,␣
↪validation_split=0.1):
        # Select features for the current model
        X_train_selected = X_train[:, self.feature_indices]
        self.history = self.model.fit(
```

```
            X_train_selected, y_train,
            epochs=epochs, batch_size=batch_size,
            validation_split=validation_split, verbose=1
        )

    def evaluate(self, X_test, y_test):
        X_test_selected = X_test[:, self.feature_indices]
        return self.model.evaluate(X_test_selected, y_test, verbose=0)

# Each model
models = {}
for name, features in features_dictionary.items():
    models[name] = DeepLearningModel(name, features)
```

```
/opt/homebrew/anaconda3/lib/python3.12/site-
packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

1.5 Model 1 - Every Feature Except For Date

[ ]:
```
model1 = models['Model1']
model1.train(X_train, Y_train)
m1loss, m1mae = model1.evaluate(X_test, Y_test)
print(f"Model1: Loss={m1loss}, MAE={m1mae}")
```

```
Epoch 1/50
29/29                0s 2ms/step - loss:
28370.3203 - mae: 126.5642 - val_loss: 22948.7051 - val_mae: 109.4151
Epoch 2/50
29/29                0s 1ms/step - loss:
25600.8652 - mae: 119.6071 - val_loss: 23122.9238 - val_mae: 111.1534
Epoch 3/50
29/29                0s 1ms/step - loss:
29785.6758 - mae: 127.1262 - val_loss: 22962.2676 - val_mae: 109.8654
Epoch 4/50
29/29                0s 1ms/step - loss:
24981.5664 - mae: 116.6116 - val_loss: 22882.4512 - val_mae: 109.0003
Epoch 5/50
29/29                0s 1ms/step - loss:
26943.7832 - mae: 123.7986 - val_loss: 23011.4824 - val_mae: 110.4552
Epoch 6/50
29/29                0s 1ms/step - loss:
27853.3359 - mae: 124.6374 - val_loss: 23013.8984 - val_mae: 110.5030
Epoch 7/50
29/29                0s 1ms/step - loss:
```

```
26416.4023 - mae: 120.9619 - val_loss: 23024.7480 - val_mae: 110.6226
Epoch 8/50
29/29            0s 1ms/step - loss:
26680.4160 - mae: 123.8184 - val_loss: 22992.4609 - val_mae: 110.2855
Epoch 9/50
29/29            0s 1ms/step - loss:
24689.7598 - mae: 117.4611 - val_loss: 23053.2676 - val_mae: 111.0174
Epoch 10/50
29/29            0s 1ms/step - loss:
27736.6641 - mae: 124.9723 - val_loss: 22973.5938 - val_mae: 110.1811
Epoch 11/50
29/29            0s 1ms/step - loss:
27460.8594 - mae: 124.2313 - val_loss: 22991.1367 - val_mae: 110.3552
Epoch 12/50
29/29            0s 1ms/step - loss:
25451.1641 - mae: 122.0816 - val_loss: 22978.2715 - val_mae: 110.1152
Epoch 13/50
29/29            0s 1ms/step - loss:
30661.5508 - mae: 129.4651 - val_loss: 22949.6016 - val_mae: 109.8653
Epoch 14/50
29/29            0s 1ms/step - loss:
26395.0254 - mae: 121.5385 - val_loss: 22932.6699 - val_mae: 109.6672
Epoch 15/50
29/29            0s 1ms/step - loss:
22904.7109 - mae: 112.6320 - val_loss: 23105.4141 - val_mae: 111.3926
Epoch 16/50
29/29            0s 1ms/step - loss:
26900.9141 - mae: 121.7389 - val_loss: 23022.2227 - val_mae: 110.6334
Epoch 17/50
29/29            0s 1ms/step - loss:
26581.6191 - mae: 121.3677 - val_loss: 23026.9570 - val_mae: 110.8974
Epoch 18/50
29/29            0s 1ms/step - loss:
24838.0840 - mae: 118.8808 - val_loss: 22919.3398 - val_mae: 109.6211
Epoch 19/50
29/29            0s 1ms/step - loss:
26217.3828 - mae: 121.3622 - val_loss: 22928.1699 - val_mae: 109.7341
Epoch 20/50
29/29            0s 1ms/step - loss:
26648.1836 - mae: 122.6112 - val_loss: 22952.8672 - val_mae: 110.0267
Epoch 21/50
29/29            0s 1ms/step - loss:
25808.5215 - mae: 119.7596 - val_loss: 22861.5391 - val_mae: 108.9964
Epoch 22/50
29/29            0s 1ms/step - loss:
28103.9043 - mae: 126.0977 - val_loss: 22956.8457 - val_mae: 110.1620
Epoch 23/50
29/29            0s 1ms/step - loss:
```

```
28121.7031 - mae: 128.0676 - val_loss: 22860.3340 - val_mae: 108.9378
Epoch 24/50
29/29            0s 1ms/step - loss:
26199.2676 - mae: 118.1943 - val_loss: 22947.9785 - val_mae: 110.0125
Epoch 25/50
29/29            0s 1ms/step - loss:
25807.4688 - mae: 121.8177 - val_loss: 23042.4102 - val_mae: 110.9462
Epoch 26/50
29/29            0s 1ms/step - loss:
24107.5508 - mae: 116.9614 - val_loss: 22959.0391 - val_mae: 110.0385
Epoch 27/50
29/29            0s 1ms/step - loss:
26202.9805 - mae: 122.5346 - val_loss: 22915.3613 - val_mae: 109.6608
Epoch 28/50
29/29            0s 1ms/step - loss:
26779.9961 - mae: 123.3850 - val_loss: 22881.5371 - val_mae: 109.2659
Epoch 29/50
29/29            0s 1ms/step - loss:
26001.2480 - mae: 120.2333 - val_loss: 22900.1328 - val_mae: 109.6122
Epoch 30/50
29/29            0s 1ms/step - loss:
26520.5898 - mae: 122.3495 - val_loss: 22982.6504 - val_mae: 110.6329
Epoch 31/50
29/29            0s 1ms/step - loss:
24313.9727 - mae: 116.0131 - val_loss: 23047.5430 - val_mae: 111.2873
Epoch 32/50
29/29            0s 1ms/step - loss:
25405.3203 - mae: 119.0674 - val_loss: 23039.3535 - val_mae: 111.2673
Epoch 33/50
29/29            0s 1ms/step - loss:
24970.8418 - mae: 120.5803 - val_loss: 22952.1543 - val_mae: 110.2273
Epoch 34/50
29/29            0s 1ms/step - loss:
25065.9688 - mae: 121.2952 - val_loss: 22882.5449 - val_mae: 109.3626
Epoch 35/50
29/29            0s 1ms/step - loss:
24319.6309 - mae: 119.5800 - val_loss: 22965.4785 - val_mae: 110.3684
Epoch 36/50
29/29            0s 1ms/step - loss:
24407.2500 - mae: 117.1395 - val_loss: 22864.7285 - val_mae: 109.2772
Epoch 37/50
29/29            0s 1ms/step - loss:
23856.1660 - mae: 115.9603 - val_loss: 23098.4453 - val_mae: 111.8045
Epoch 38/50
29/29            0s 1ms/step - loss:
27634.6133 - mae: 124.8530 - val_loss: 22929.3730 - val_mae: 110.0902
Epoch 39/50
29/29            0s 1ms/step - loss:
```

```
25658.7617 - mae: 118.8973 - val_loss: 22935.2676 - val_mae: 110.0645
Epoch 40/50
29/29              0s 1ms/step - loss:
24895.2363 - mae: 117.9678 - val_loss: 22916.5273 - val_mae: 109.7640
Epoch 41/50
29/29              0s 1ms/step - loss:
27581.6152 - mae: 126.9919 - val_loss: 22910.4883 - val_mae: 109.7579
Epoch 42/50
29/29              0s 1ms/step - loss:
26881.6250 - mae: 118.4613 - val_loss: 22882.5547 - val_mae: 109.3791
Epoch 43/50
29/29              0s 1ms/step - loss:
27825.2539 - mae: 128.3430 - val_loss: 22995.2773 - val_mae: 110.6199
Epoch 44/50
29/29              0s 1ms/step - loss:
28171.7793 - mae: 128.1667 - val_loss: 22906.2285 - val_mae: 109.6828
Epoch 45/50
29/29              0s 1ms/step - loss:
27073.0566 - mae: 122.3703 - val_loss: 22943.0000 - val_mae: 110.2126
Epoch 46/50
29/29              0s 1ms/step - loss:
27926.2715 - mae: 126.1402 - val_loss: 22937.9746 - val_mae: 110.1373
Epoch 47/50
29/29              0s 1ms/step - loss:
28205.0430 - mae: 127.6738 - val_loss: 22977.5215 - val_mae: 110.4208
Epoch 48/50
29/29              0s 1ms/step - loss:
25861.0586 - mae: 118.4258 - val_loss: 22868.9844 - val_mae: 109.2259
Epoch 49/50
29/29              0s 1ms/step - loss:
26199.7930 - mae: 118.0424 - val_loss: 23020.2695 - val_mae: 110.9060
Epoch 50/50
29/29              0s 1ms/step - loss:
25425.1113 - mae: 119.9558 - val_loss: 22949.1602 - val_mae: 110.3081
Model1: Loss=26112.140625, MAE=117.92471313476562
```

1.6 Model 2 - Open, Low, and Volume

---

[23]:
```python
model2 = models['Model2']
model2.train(X_train, Y_train)
m2loss, m2mae = model2.evaluate(X_test, Y_test)
print(f"Model2: Loss={m2loss}, MAE={m2mae}")
```

```
Epoch 1/50
29/29              0s 2ms/step - loss:
28812.0938 - mae: 132.7459 - val_loss: 25141.5898 - val_mae: 128.1075
```

```
Epoch 2/50
29/29            0s 1ms/step - loss:
25018.9824 - mae: 123.9633 - val_loss: 24909.5059 - val_mae: 126.3309
Epoch 3/50
29/29            0s 1ms/step - loss:
28974.1172 - mae: 131.0616 - val_loss: 24743.3594 - val_mae: 125.3700
Epoch 4/50
29/29            0s 1ms/step - loss:
28234.8984 - mae: 132.7963 - val_loss: 24537.8340 - val_mae: 123.6904
Epoch 5/50
29/29            0s 1ms/step - loss:
26673.2070 - mae: 126.7331 - val_loss: 24366.0000 - val_mae: 122.2310
Epoch 6/50
29/29            0s 1ms/step - loss:
28965.2559 - mae: 129.6909 - val_loss: 24258.4570 - val_mae: 121.6341
Epoch 7/50
29/29            0s 1ms/step - loss:
28853.4512 - mae: 131.3919 - val_loss: 24069.1660 - val_mae: 119.3996
Epoch 8/50
29/29            0s 1ms/step - loss:
28721.3281 - mae: 128.5680 - val_loss: 23962.7988 - val_mae: 118.6927
Epoch 9/50
29/29            0s 1ms/step - loss:
26195.9980 - mae: 124.1904 - val_loss: 23877.9062 - val_mae: 118.0883
Epoch 10/50
29/29            0s 1ms/step - loss:
27404.4492 - mae: 127.7584 - val_loss: 23763.9570 - val_mae: 116.7863
Epoch 11/50
29/29            0s 1ms/step - loss:
26638.1504 - mae: 123.8092 - val_loss: 23668.3164 - val_mae: 115.7944
Epoch 12/50
29/29            0s 1ms/step - loss:
24786.9883 - mae: 117.8913 - val_loss: 23742.8398 - val_mae: 117.2542
Epoch 13/50
29/29            0s 1ms/step - loss:
28117.9844 - mae: 129.0065 - val_loss: 23590.3730 - val_mae: 115.5622
Epoch 14/50
29/29            0s 1ms/step - loss:
26501.8281 - mae: 123.1734 - val_loss: 23526.1680 - val_mae: 114.9254
Epoch 15/50
29/29            0s 1ms/step - loss:
26132.5977 - mae: 122.0284 - val_loss: 23570.8945 - val_mae: 115.7408
Epoch 16/50
29/29            0s 1ms/step - loss:
26979.1250 - mae: 124.6934 - val_loss: 23480.8496 - val_mae: 114.8063
Epoch 17/50
29/29            0s 1ms/step - loss:
25599.8945 - mae: 122.1290 - val_loss: 23468.6191 - val_mae: 114.7901
```

```
Epoch 18/50
29/29              0s 1ms/step - loss:
28792.7188 - mae: 128.3037 - val_loss: 23385.7051 - val_mae: 113.9261
Epoch 19/50
29/29              0s 1ms/step - loss:
27172.7344 - mae: 124.0678 - val_loss: 23307.5352 - val_mae: 113.0948
Epoch 20/50
29/29              0s 1ms/step - loss:
26274.9043 - mae: 118.9787 - val_loss: 23290.7793 - val_mae: 113.1159
Epoch 21/50
29/29              0s 1ms/step - loss:
24223.8867 - mae: 118.3001 - val_loss: 23300.9980 - val_mae: 113.3343
Epoch 22/50
29/29              0s 1ms/step - loss:
26783.7324 - mae: 122.5470 - val_loss: 23338.5547 - val_mae: 113.8645
Epoch 23/50
29/29              0s 1ms/step - loss:
27474.3828 - mae: 126.8213 - val_loss: 23293.7852 - val_mae: 113.4266
Epoch 24/50
29/29              0s 1ms/step - loss:
28137.0352 - mae: 126.9800 - val_loss: 23152.9570 - val_mae: 111.9901
Epoch 25/50
29/29              0s 1ms/step - loss:
25550.9707 - mae: 120.9170 - val_loss: 23165.2578 - val_mae: 112.2064
Epoch 26/50
29/29              0s 1ms/step - loss:
24265.0566 - mae: 117.0819 - val_loss: 23227.9004 - val_mae: 112.8464
Epoch 27/50
29/29              0s 1ms/step - loss:
25825.3730 - mae: 123.2617 - val_loss: 23267.1094 - val_mae: 113.1884
Epoch 28/50
29/29              0s 1ms/step - loss:
25684.3066 - mae: 122.4128 - val_loss: 23081.3555 - val_mae: 111.4500
Epoch 29/50
29/29              0s 1ms/step - loss:
26322.2285 - mae: 123.1851 - val_loss: 23106.2500 - val_mae: 111.7892
Epoch 30/50
29/29              0s 1ms/step - loss:
26193.2969 - mae: 120.5727 - val_loss: 23075.2578 - val_mae: 111.6170
Epoch 31/50
29/29              0s 1ms/step - loss:
24575.8613 - mae: 117.8963 - val_loss: 23051.8027 - val_mae: 111.4498
Epoch 32/50
29/29              0s 1ms/step - loss:
25463.4707 - mae: 120.8961 - val_loss: 23080.3555 - val_mae: 111.8008
Epoch 33/50
29/29              0s 1ms/step - loss:
25723.9180 - mae: 120.7477 - val_loss: 23196.6113 - val_mae: 112.8449
```

```
Epoch 34/50
29/29              0s 1ms/step - loss:
26351.0371 - mae: 122.0052 - val_loss: 23072.7676 - val_mae: 111.7646
Epoch 35/50
29/29              0s 1ms/step - loss:
26528.6777 - mae: 123.1343 - val_loss: 23020.1328 - val_mae: 111.3543
Epoch 36/50
29/29              0s 1ms/step - loss:
24186.1719 - mae: 116.6348 - val_loss: 22935.7559 - val_mae: 110.5165
Epoch 37/50
29/29              0s 1ms/step - loss:
26199.2559 - mae: 121.8520 - val_loss: 22906.5684 - val_mae: 110.4315
Epoch 38/50
29/29              0s 1ms/step - loss:
28507.8164 - mae: 126.7017 - val_loss: 23022.0566 - val_mae: 111.7272
Epoch 39/50
29/29              0s 1ms/step - loss:
26163.8262 - mae: 123.2475 - val_loss: 22978.6172 - val_mae: 111.3882
Epoch 40/50
29/29              0s 1ms/step - loss:
28532.2969 - mae: 128.3905 - val_loss: 22890.8535 - val_mae: 110.6656
Epoch 41/50
29/29              0s 1ms/step - loss:
26423.8281 - mae: 121.9996 - val_loss: 22847.1934 - val_mae: 110.2067
Epoch 42/50
29/29              0s 1ms/step - loss:
25212.4629 - mae: 119.8012 - val_loss: 22756.8223 - val_mae: 109.4396
Epoch 43/50
29/29              0s 1ms/step - loss:
24192.3066 - mae: 115.3305 - val_loss: 22917.0312 - val_mae: 111.1686
Epoch 44/50
29/29              0s 1ms/step - loss:
27524.4316 - mae: 125.9858 - val_loss: 22757.1855 - val_mae: 109.8325
Epoch 45/50
29/29              0s 1ms/step - loss:
27809.5254 - mae: 125.9975 - val_loss: 22787.2383 - val_mae: 110.2964
Epoch 46/50
29/29              0s 1ms/step - loss:
24456.4766 - mae: 118.8977 - val_loss: 22914.8242 - val_mae: 111.5793
Epoch 47/50
29/29              0s 1ms/step - loss:
25234.9551 - mae: 120.4147 - val_loss: 23007.9785 - val_mae: 112.4383
Epoch 48/50
29/29              0s 1ms/step - loss:
24926.8555 - mae: 119.4357 - val_loss: 22794.2676 - val_mae: 110.5454
Epoch 49/50
29/29              0s 1ms/step - loss:
25640.1152 - mae: 119.1767 - val_loss: 22801.1406 - val_mae: 110.6440
```

```
Epoch 50/50
29/29              0s 1ms/step - loss:
24360.5195 - mae: 115.4747 - val_loss: 22675.9941 - val_mae: 109.7248
Model2: Loss=25806.84765625, MAE=117.37318420410156
```

1.7 Model 3 - Open and Volume

---

[24]:
```
model3 = models['Model3']
model3.train(X_train, Y_train)
m3loss, m3mae = model3.evaluate(X_test, Y_test)
print(f"Model3: Loss={m3loss}, MAE={m3mae}")
```

```
Epoch 1/50
29/29              0s 3ms/step - loss:
534739.8125 - mae: 711.7370 - val_loss: 519191.2188 - val_mae: 704.2909
Epoch 2/50
29/29              0s 1ms/step - loss:
538378.6875 - mae: 713.7792 - val_loss: 516197.2188 - val_mae: 702.1772
Epoch 3/50
29/29              0s 1ms/step - loss:
532242.6875 - mae: 709.5441 - val_loss: 509407.3125 - val_mae: 697.3585
Epoch 4/50
29/29              0s 1ms/step - loss:
525493.4375 - mae: 705.5302 - val_loss: 496545.0000 - val_mae: 688.1323
Epoch 5/50
29/29              0s 1ms/step - loss:
498449.5938 - mae: 687.9258 - val_loss: 475374.3438 - val_mae: 672.6631
Epoch 6/50
29/29              0s 1ms/step - loss:
480538.4375 - mae: 673.4208 - val_loss: 444301.1875 - val_mae: 649.2700
Epoch 7/50
29/29              0s 1ms/step - loss:
440878.5938 - mae: 645.4056 - val_loss: 402918.4688 - val_mae: 616.7004
Epoch 8/50
29/29              0s 1ms/step - loss:
398694.5938 - mae: 609.5698 - val_loss: 351785.1562 - val_mae: 573.8356
Epoch 9/50
29/29              0s 1ms/step - loss:
348017.7188 - mae: 567.0738 - val_loss: 293352.0625 - val_mae: 520.4122
Epoch 10/50
29/29              0s 1ms/step - loss:
284353.5625 - mae: 509.1370 - val_loss: 232428.8281 - val_mae: 457.8993
Epoch 11/50
29/29              0s 1ms/step - loss:
224588.5469 - mae: 445.4697 - val_loss: 173628.7969 - val_mae: 387.7497
Epoch 12/50
```

```
29/29             0s 1ms/step - loss:
173057.6250 - mae: 380.8571 - val_loss: 122478.4062 - val_mae: 313.6195
Epoch 13/50
29/29             0s 1ms/step - loss:
118559.6797 - mae: 300.8306 - val_loss: 82782.5625 - val_mae: 239.8885
Epoch 14/50
29/29             0s 1ms/step - loss:
84942.8359 - mae: 232.5396 - val_loss: 56353.0547 - val_mae: 174.9237
Epoch 15/50
29/29             0s 1ms/step - loss:
60354.6562 - mae: 179.3221 - val_loss: 41477.3945 - val_mae: 142.4140
Epoch 16/50
29/29             0s 1ms/step - loss:
44714.1953 - mae: 151.4915 - val_loss: 34497.3633 - val_mae: 140.5484
Epoch 17/50
29/29             0s 1ms/step - loss:
38623.9609 - mae: 148.9069 - val_loss: 31478.9180 - val_mae: 145.8142
Epoch 18/50
29/29             0s 1ms/step - loss:
35532.0078 - mae: 149.5918 - val_loss: 30276.9746 - val_mae: 148.5980
Epoch 19/50
29/29             0s 1ms/step - loss:
35083.1016 - mae: 153.4113 - val_loss: 29740.5117 - val_mae: 149.6856
Epoch 20/50
29/29             0s 1ms/step - loss:
31338.7715 - mae: 147.5528 - val_loss: 29401.0527 - val_mae: 149.7660
Epoch 21/50
29/29             0s 1ms/step - loss:
32089.6914 - mae: 150.3046 - val_loss: 29133.9570 - val_mae: 149.7255
Epoch 22/50
29/29             0s 1ms/step - loss:
33456.4883 - mae: 151.1467 - val_loss: 28905.2676 - val_mae: 149.3125
Epoch 23/50
29/29             0s 1ms/step - loss:
31427.2227 - mae: 148.3245 - val_loss: 28631.6602 - val_mae: 147.8110
Epoch 24/50
29/29             0s 1ms/step - loss:
31293.4043 - mae: 145.7733 - val_loss: 28375.3027 - val_mae: 146.5458
Epoch 25/50
29/29             0s 1ms/step - loss:
31088.0605 - mae: 145.0971 - val_loss: 28174.4824 - val_mae: 146.0423
Epoch 26/50
29/29             0s 1ms/step - loss:
33408.9648 - mae: 151.1647 - val_loss: 27952.0488 - val_mae: 145.0937
Epoch 27/50
29/29             0s 1ms/step - loss:
31906.4590 - mae: 145.8375 - val_loss: 27658.8027 - val_mae: 143.1359
Epoch 28/50
```

```
29/29              0s 1ms/step - loss:
32596.8711 - mae: 146.3936 - val_loss: 27407.3965 - val_mae: 141.5439
Epoch 29/50
29/29              0s 1ms/step - loss:
31882.5410 - mae: 144.1437 - val_loss: 27193.5156 - val_mae: 140.6736
Epoch 30/50
29/29              0s 1ms/step - loss:
29694.1797 - mae: 141.1842 - val_loss: 26964.6465 - val_mae: 139.3521
Epoch 31/50
29/29              0s 1ms/step - loss:
30385.2461 - mae: 140.5966 - val_loss: 26735.7520 - val_mae: 137.9059
Epoch 32/50
29/29              0s 1ms/step - loss:
29236.7695 - mae: 138.4479 - val_loss: 26582.6133 - val_mae: 137.6965
Epoch 33/50
29/29              0s 1ms/step - loss:
29812.7324 - mae: 139.4223 - val_loss: 26422.5371 - val_mae: 136.9933
Epoch 34/50
29/29              0s 1ms/step - loss:
29501.7324 - mae: 139.7239 - val_loss: 26165.0684 - val_mae: 135.0531
Epoch 35/50
29/29              0s 1ms/step - loss:
30519.2109 - mae: 138.0943 - val_loss: 25988.9473 - val_mae: 134.0342
Epoch 36/50
29/29              0s 1ms/step - loss:
31496.9355 - mae: 141.7415 - val_loss: 25809.9102 - val_mae: 132.8464
Epoch 37/50
29/29              0s 1ms/step - loss:
28508.4102 - mae: 135.6502 - val_loss: 25602.5293 - val_mae: 131.2348
Epoch 38/50
29/29              0s 1ms/step - loss:
30191.9883 - mae: 139.2510 - val_loss: 25453.5078 - val_mae: 130.4232
Epoch 39/50
29/29              0s 1ms/step - loss:
26128.9922 - mae: 127.5790 - val_loss: 25330.5117 - val_mae: 129.8986
Epoch 40/50
29/29              0s 1ms/step - loss:
28374.1953 - mae: 132.3744 - val_loss: 25183.9805 - val_mae: 128.7983
Epoch 41/50
29/29              0s 1ms/step - loss:
28696.2930 - mae: 132.7467 - val_loss: 25068.0996 - val_mae: 128.3271
Epoch 42/50
29/29              0s 1ms/step - loss:
30165.2051 - mae: 134.0542 - val_loss: 24946.9238 - val_mae: 127.6223
Epoch 43/50
29/29              0s 1ms/step - loss:
28655.7305 - mae: 132.3934 - val_loss: 24810.6230 - val_mae: 126.6913
Epoch 44/50
```

```
29/29              0s 1ms/step - loss:
29218.2812 - mae: 134.0314 - val_loss: 24648.6367 - val_mae: 125.1712
Epoch 45/50
29/29              0s 1ms/step - loss:
30085.6602 - mae: 136.0771 - val_loss: 24517.1602 - val_mae: 124.0780
Epoch 46/50
29/29              0s 1ms/step - loss:
28739.8770 - mae: 132.5713 - val_loss: 24448.5898 - val_mae: 124.0446
Epoch 47/50
29/29              0s 1ms/step - loss:
27728.3809 - mae: 129.7123 - val_loss: 24328.3691 - val_mae: 123.0174
Epoch 48/50
29/29              0s 1ms/step - loss:
26509.7207 - mae: 125.4872 - val_loss: 24225.7773 - val_mae: 122.1881
Epoch 49/50
29/29              0s 1ms/step - loss:
27530.9922 - mae: 128.3953 - val_loss: 24213.8457 - val_mae: 122.6899
Epoch 50/50
29/29              0s 1ms/step - loss:
30163.9180 - mae: 132.0680 - val_loss: 24183.1309 - val_mae: 122.6883
Model3: Loss=27664.771484375, MAE=129.22500610351562
```

1.8 Model 4 - Open and Low

---

[25]:
```
model4 = models['Model4']
model4.train(X_train, Y_train)
m4loss, m4mae = model4.evaluate(X_test, Y_test)
print(f"Model4: Loss={m4loss}, MAE={m4mae}")
```

```
Epoch 1/50
29/29              0s 3ms/step - loss:
522854.9375 - mae: 705.0274 - val_loss: 520071.4062 - val_mae: 704.9104
Epoch 2/50
29/29              0s 1ms/step - loss:
536846.8750 - mae: 713.8518 - val_loss: 518864.5625 - val_mae: 704.0608
Epoch 3/50
29/29              0s 1ms/step - loss:
537043.1875 - mae: 712.9481 - val_loss: 516101.6875 - val_mae: 702.1096
Epoch 4/50
29/29              0s 1ms/step - loss:
536468.0625 - mae: 712.8058 - val_loss: 510775.6875 - val_mae: 698.3300
Epoch 5/50
29/29              0s 1ms/step - loss:
516571.0312 - mae: 700.8719 - val_loss: 501823.9062 - val_mae: 691.9273
Epoch 6/50
29/29              0s 1ms/step - loss:
```

```
508882.7500 - mae: 694.7305 - val_loss: 488299.0625 - val_mae: 682.1353
Epoch 7/50
29/29            0s 1ms/step - loss:
492150.3438 - mae: 683.1104 - val_loss: 469354.0625 - val_mae: 668.1716
Epoch 8/50
29/29            0s 1ms/step - loss:
488359.4688 - mae: 678.7653 - val_loss: 444364.4375 - val_mae: 649.2853
Epoch 9/50
29/29            0s 1ms/step - loss:
450677.1250 - mae: 650.1018 - val_loss: 413627.8125 - val_mae: 625.2618
Epoch 10/50
29/29            0s 1ms/step - loss:
431923.9688 - mae: 635.4297 - val_loss: 377086.9062 - val_mae: 595.4255
Epoch 11/50
29/29            0s 1ms/step - loss:
376748.1875 - mae: 592.1757 - val_loss: 335933.9688 - val_mae: 559.8982
Epoch 12/50
29/29            0s 1ms/step - loss:
341636.9062 - mae: 559.5388 - val_loss: 290868.2500 - val_mae: 518.1704
Epoch 13/50
29/29            0s 1ms/step - loss:
283839.0938 - mae: 508.0380 - val_loss: 243913.9531 - val_mae: 470.7217
Epoch 14/50
29/29            0s 1ms/step - loss:
248884.6562 - mae: 470.7483 - val_loss: 197492.2188 - val_mae: 418.4786
Epoch 15/50
29/29            0s 1ms/step - loss:
195994.1562 - mae: 413.3004 - val_loss: 153822.3594 - val_mae: 362.4085
Epoch 16/50
29/29            0s 1ms/step - loss:
158118.1250 - mae: 361.8278 - val_loss: 115428.6641 - val_mae: 305.6562
Epoch 17/50
29/29            0s 1ms/step - loss:
120895.8047 - mae: 303.7696 - val_loss: 84227.0625 - val_mae: 250.6843
Epoch 18/50
29/29            0s 1ms/step - loss:
90861.0156 - mae: 252.0874 - val_loss: 60674.7617 - val_mae: 199.2888
Epoch 19/50
29/29            0s 1ms/step - loss:
62075.5039 - mae: 195.0382 - val_loss: 44264.3633 - val_mae: 156.9230
Epoch 20/50
29/29            0s 1ms/step - loss:
49812.8164 - mae: 164.0208 - val_loss: 34383.6914 - val_mae: 134.3285
Epoch 21/50
29/29            0s 1ms/step - loss:
41335.8203 - mae: 147.9767 - val_loss: 28969.4043 - val_mae: 124.5107
Epoch 22/50
29/29            0s 1ms/step - loss:
```

```
33737.8711 - mae: 134.4945 - val_loss: 26422.7969 - val_mae: 120.6935
Epoch 23/50
29/29            0s 1ms/step - loss:
30595.9648 - mae: 131.3508 - val_loss: 25397.0293 - val_mae: 119.7730
Epoch 24/50
29/29            0s 1ms/step - loss:
32732.0312 - mae: 136.7651 - val_loss: 24995.1309 - val_mae: 120.0872
Epoch 25/50
29/29            0s 1ms/step - loss:
30705.0410 - mae: 137.5934 - val_loss: 24936.4316 - val_mae: 120.5562
Epoch 26/50
29/29            0s 1ms/step - loss:
29864.5723 - mae: 133.2247 - val_loss: 24953.1055 - val_mae: 121.0548
Epoch 27/50
29/29            0s 1ms/step - loss:
29174.4141 - mae: 132.3843 - val_loss: 24971.8535 - val_mae: 121.3483
Epoch 28/50
29/29            0s 1ms/step - loss:
30274.9297 - mae: 137.6172 - val_loss: 24963.7754 - val_mae: 121.4824
Epoch 29/50
29/29            0s 1ms/step - loss:
32615.9121 - mae: 142.7236 - val_loss: 25006.5996 - val_mae: 121.8545
Epoch 30/50
29/29            0s 1ms/step - loss:
30440.6523 - mae: 136.7662 - val_loss: 25002.4336 - val_mae: 121.9570
Epoch 31/50
29/29            0s 1ms/step - loss:
33398.1016 - mae: 138.6189 - val_loss: 24995.3242 - val_mae: 122.0710
Epoch 32/50
29/29            0s 1ms/step - loss:
28207.3887 - mae: 129.9369 - val_loss: 24969.7676 - val_mae: 122.0367
Epoch 33/50
29/29            0s 1ms/step - loss:
32753.3145 - mae: 140.1547 - val_loss: 24949.5352 - val_mae: 122.0525
Epoch 34/50
29/29            0s 1ms/step - loss:
32103.3008 - mae: 139.6828 - val_loss: 24858.2207 - val_mae: 121.7155
Epoch 35/50
29/29            0s 1ms/step - loss:
34476.4102 - mae: 144.8006 - val_loss: 24832.5488 - val_mae: 121.6957
Epoch 36/50
29/29            0s 1ms/step - loss:
28053.9707 - mae: 131.2436 - val_loss: 24822.1582 - val_mae: 121.7716
Epoch 37/50
29/29            0s 1ms/step - loss:
33067.2656 - mae: 138.7504 - val_loss: 24794.4180 - val_mae: 121.7766
Epoch 38/50
29/29            0s 1ms/step - loss:
```

```
28766.7832 - mae: 130.0310 - val_loss: 24789.2012 - val_mae: 121.8370
Epoch 39/50
29/29           0s 1ms/step - loss:
30018.0098 - mae: 133.5793 - val_loss: 24733.1660 - val_mae: 121.6782
Epoch 40/50
29/29           0s 1ms/step - loss:
28176.5957 - mae: 131.3183 - val_loss: 24675.8184 - val_mae: 121.5245
Epoch 41/50
29/29           0s 1ms/step - loss:
30523.2012 - mae: 135.3908 - val_loss: 24656.8242 - val_mae: 121.5559
Epoch 42/50
29/29           0s 1ms/step - loss:
30852.5293 - mae: 139.4847 - val_loss: 24620.7500 - val_mae: 121.4856
Epoch 43/50
29/29           0s 1ms/step - loss:
29331.7461 - mae: 137.7069 - val_loss: 24561.7012 - val_mae: 121.3100
Epoch 44/50
29/29           0s 1ms/step - loss:
29798.8340 - mae: 131.5747 - val_loss: 24571.1758 - val_mae: 121.4791
Epoch 45/50
29/29           0s 1ms/step - loss:
31009.0684 - mae: 135.6317 - val_loss: 24501.8711 - val_mae: 121.2642
Epoch 46/50
29/29           0s 1ms/step - loss:
30125.7129 - mae: 138.2657 - val_loss: 24413.7754 - val_mae: 120.8971
Epoch 47/50
29/29           0s 1ms/step - loss:
29309.6230 - mae: 131.0885 - val_loss: 24365.1387 - val_mae: 120.7674
Epoch 48/50
29/29           0s 1ms/step - loss:
29104.0332 - mae: 132.6562 - val_loss: 24358.3672 - val_mae: 120.8662
Epoch 49/50
29/29           0s 1ms/step - loss:
30403.9668 - mae: 135.0929 - val_loss: 24357.4316 - val_mae: 120.9717
Epoch 50/50
29/29           0s 1ms/step - loss:
29599.6836 - mae: 135.5686 - val_loss: 24342.4609 - val_mae: 121.0313
Model4: Loss=28335.056640625, MAE=132.73684692382812
```

1.9 Evaluation

Now that we have trained our models, we can evaluate them and see which model performed the best. For starters, here were the shared parameters for all models that were trained:

- Model Type: Fully Connected Feedforward Neural Network
- Optimizer: Adam
- Epochs: 50

- Batch Size: 32
- Validation Split: 0.1
- Loss Function: Mean Squared Error
- Metrics: Mean Absolute Error

The Mean Squared Error was chosen for the loss function because it is typically a good choice for regression models. The Mean Absolute Error was chosen for the metrics because it is easy to interpret and understand. Each model was trained under these same parameters but with different features. The results for each model can be seen in the table below:

Model	Mean Squared Error	Mean Absolute Error
1	26112.140625	117.92471313476562
2	25806.84765625	117.37318420410156
3	27664.771484375	129.22500610351562
4	28335.056640625	132.73684692382812

From this, we can see that the model that had the lowest Mean Squared Error was Model 2 with a value of 25806.84765625. This model also had the lowest Mean Absolute Error as well. This means that Model 2 is the best model out of the four that were created.

The model that performed the worst was Model 4 with a Mean Squared Error of 28335.056640625. This model also had the highest Mean Absolute Error as well. This means that Model 4 is the worst model out of the four that were created.

1.10 Conclusion

The average mean squared error of the four models that were created is 26954.2041015625. The average mean absolute error of the four models came out to be 124.96466159820557. In terms of understanding these results, the mean squared error is a measure of how close the predictions are to the actual values. The mean absolute error is a measure of how far off the predictions are from the actual values. The lower the mean squared error and mean absolute error, the better the model is performing. In this case, Model 2 performed the best out of the four models that were created. This means that if we were to use a model to predict the closing price of Google stock, we would use Model 2.

Reasons for why the error rates are so high could be due to the high volatility of the stock market and the non-linear relationships between the features and the target variable. The stock market is a highly unpredictable environment and it can be difficult to predict stock prices with a high degree of accuracy. This is a potential reason as to why the error rates are so high.

Another reason for the high error could be due to the volume of data that was used to train the model and the volume of data that was used to evaluate the model. If more data could be collected for training and evaluation, the model could potentially perform better with the current architecture that was used.

Other factors include parameters that were used to train the model such as the number of epochs, batch size, and validation split. If these parameters were changed, the model could potentially perform better as well.

Further work could be done with different data sets and higher volume of data with potentially different features as well.