stockforecastwithlstmanddl

December 9, 2023

Stock prices forecasting with LSTM and DL

In this assignment,we are using LSTM and DL to predict stock prices of Alphabet INC. The dataset has been obtained from Yahoo Finance and has the data of last 5 years which means around 1200 records. The dataset contains the following fields: Date, Open, High, Low, Close, Volume. Our aim in this assignment is to predict the stock prices of Alphabet. We will be using LSTM and Deep Learning to create a model to predict the future stock prices of Alphabet.

[46]:
```python
#Necessary imports for the project
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.stattools import adfuller
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense, LSTM
from sklearn.metrics import mean_absolute_error, median_absolute_error, r2_score
from sklearn.linear_model import LinearRegression
```

[6]:
```python
# Load the dataset
df = pd.read_csv('/content/sample_data/googlestock5Y.csv', index_col='Date',␣
↪parse_dates=True)
df.head()
```

[6]:

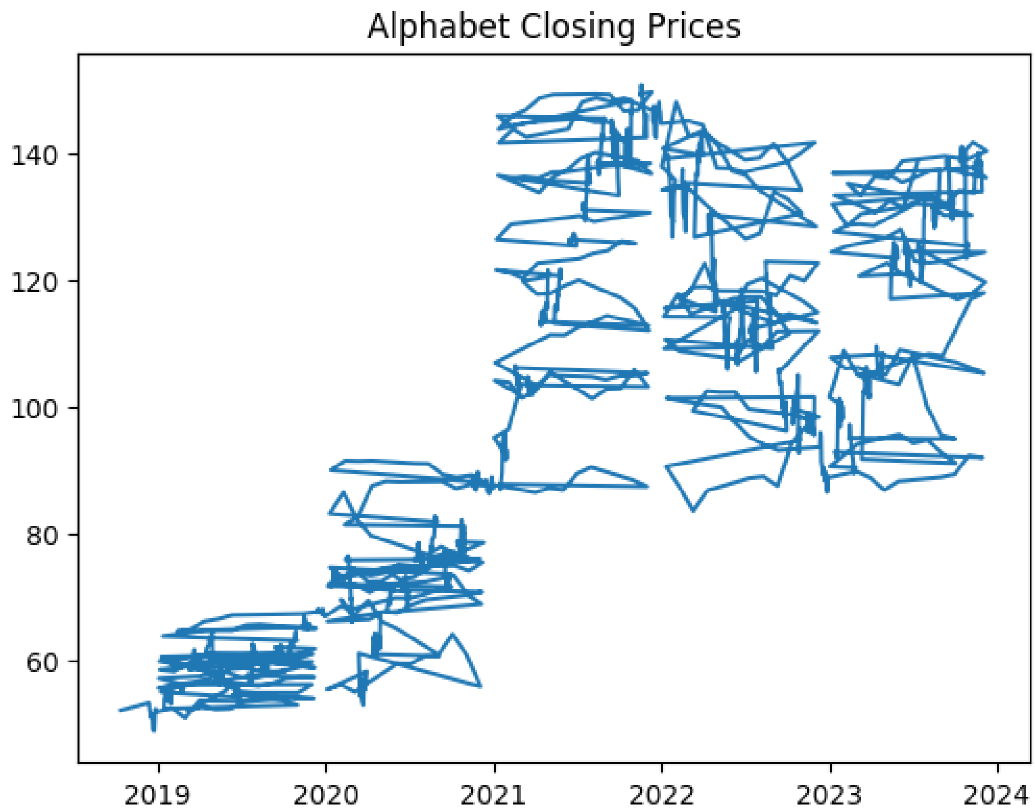
Date	Open	High	Low	Close	Volume
2018-10-12	51.752499	52.422501	51.164501	51.977501	36154000
2018-11-12	52.824501	53.029999	51.992001	52.587502	27894000
2018-12-12	53.400002	54.082500	53.139500	53.183998	30476000
2018-12-13	53.403500	53.987999	52.696499	53.095001	26596000
2018-12-14	52.499001	53.130001	52.039501	52.105000	33732000

[7]:
```python
# Checking for missing values
df.isnull().sum()
```

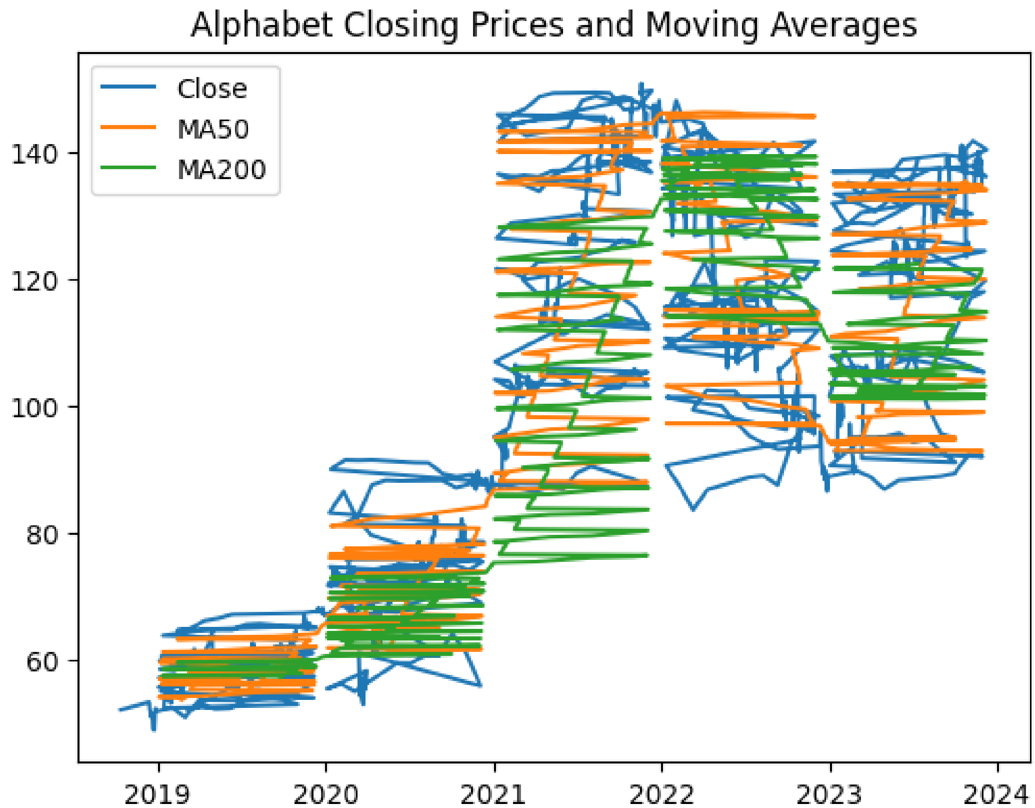
[7]:
```
Open      0
High      0
Low       0
```

```
Close     0
Volume    0
dtype: int64
```

[8]:
```python
# Plotting the closing prices
plt.plot(df['Close'])
plt.title('Alphabet Closing Prices')
plt.show()
```



[10]:
```python
# Calculating and plotting the moving average
df['MA50'] = df['Close'].rolling(50).mean()
df['MA200'] = df['Close'].rolling(200).mean()


plt.plot(df['Close'], label='Close')
plt.plot(df['MA50'], label='MA50')
plt.plot(df['MA200'], label='MA200')
plt.title('Alphabet Closing Prices and Moving Averages')
plt.legend()
plt.show()
```



The stock is very bullish and has a general upward trend

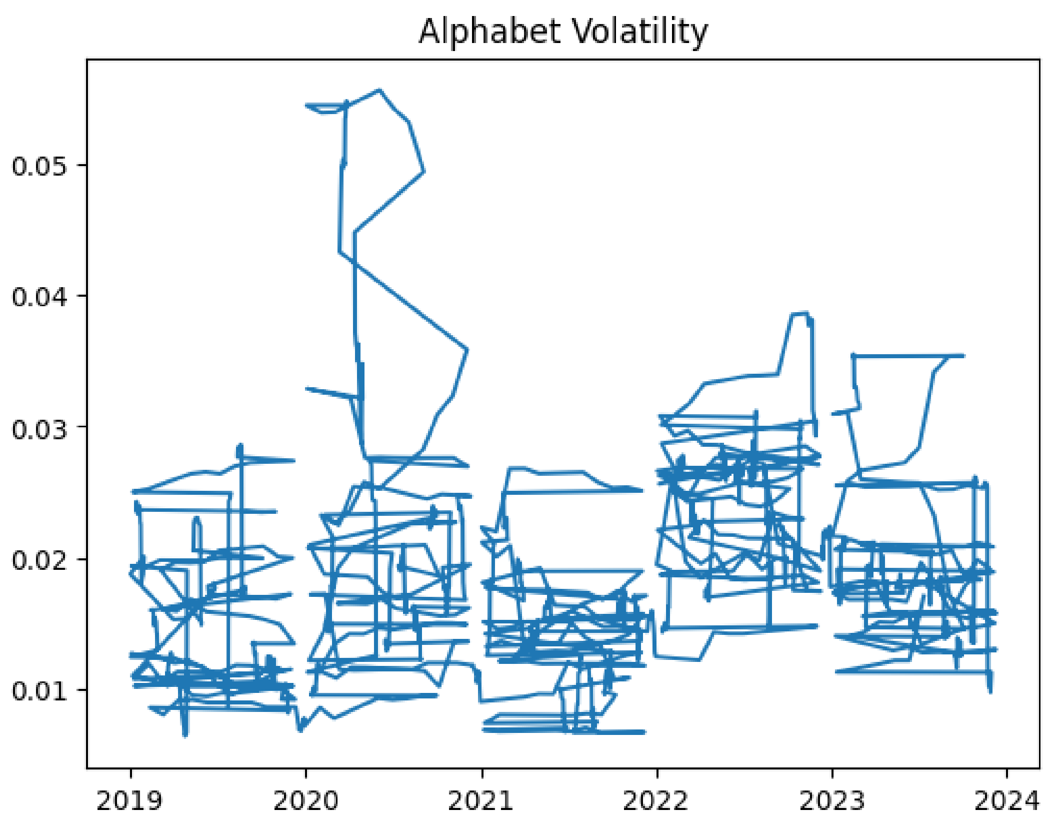
```
[12]: # Calculating the daily returns
df['Returns'] = df['Close'].pct_change()
df['Returns']
```

```
[12]: Date
2018-10-12         NaN
2018-11-12    0.011736
2018-12-12    0.011343
2018-12-13   -0.001673
2018-12-14   -0.018646
                ...
2023-01-12   -0.004480
2023-04-12   -0.020177
2023-05-12    0.013473
2023-06-12   -0.007251
2023-07-12    0.053412
Name: Returns, Length: 1258, dtype: float64
```

[13]:
```python
# Calculating the volatility
df['Volatility'] = df['Returns'].rolling(20).std()
df['Volatility']
```

[13]:
```
Date
2018-10-12         NaN
2018-11-12         NaN
2018-12-12         NaN
2018-12-13         NaN
2018-12-14         NaN
                  …
2023-01-12    0.011258
2023-04-12    0.011901
2023-05-12    0.012141
2023-06-12    0.012146
2023-07-12    0.017063
Name: Volatility, Length: 1258, dtype: float64
```

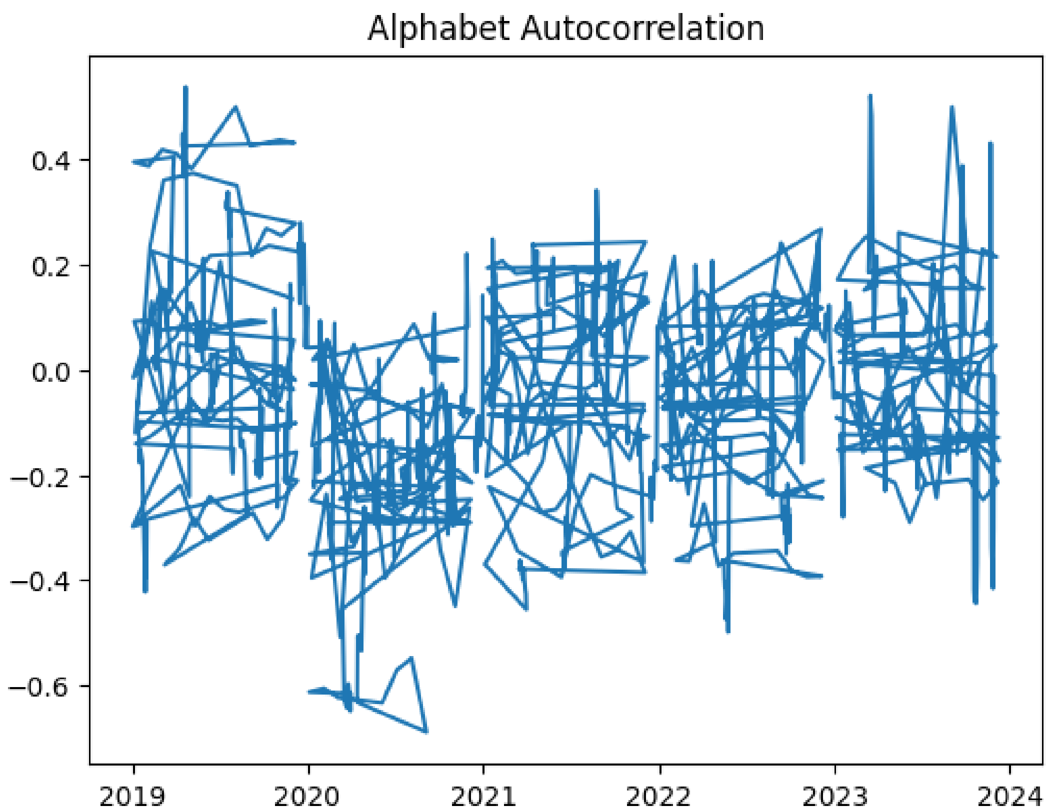
[14]:
```python
# Plotting the volatility
plt.plot(df['Volatility'])
plt.title('Alphabet Volatility')
plt.show()
```



[15]:
```python
# Calculating the autocorrelation
df['Autocorrelation'] = df['Returns'].rolling(20).corr(df['Returns'].shift(1))
df['Autocorrelation']
```

[15]:
```
Date
2018-10-12         NaN
2018-11-12         NaN
2018-12-12         NaN
2018-12-13         NaN
2018-12-14         NaN
                 ...
2023-01-12    0.014060
2023-04-12    0.039674
2023-05-12   -0.102479
2023-06-12   -0.155231
2023-07-12   -0.218289
Name: Autocorrelation, Length: 1258, dtype: float64
```

[16]:
```python
# Plotting the autocorrelation
plt.plot(df['Autocorrelation'])
plt.title('Alphabet Autocorrelation')
plt.show()
```



[18]:
```python
# Performing the Augmented Dickey-Fuller test for stationarity
adf_test = adfuller(df['Close'])
print('ADF test statistic:', adf_test[0])
print('p-value:', adf_test[1])
```

```
ADF test statistic: -1.2070558776104214
p-value: 0.6705531102119122
```

The ADF is in negative values which means that the time series could be non stationery but since p value is more than 5% we cant be conclusive.

[28]:
```python
#keeping only the necessary columns
df=df[['Open','High','Low','Close','Volume']]
df.head()
```

[28]:

Date	Open	High	Low	Close	Volume
2018-10-12	51.752499	52.422501	51.164501	51.977501	36154000
2018-11-12	52.824501	53.029999	51.992001	52.587502	27894000
2018-12-12	53.400002	54.082500	53.139500	53.183998	30476000
2018-12-13	53.403500	53.987999	52.696499	53.095001	26596000
2018-12-14	52.499001	53.130001	52.039501	52.105000	33732000

[22]:
```python
#scaling the dataset
scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(df)
```

[29]:
```python
# Splitting the data into training and testing sets
X_train = scaled_data[:-12][:, :, np.newaxis]
X_test = scaled_data[-12:][:, :, np.newaxis]
y_train = df['Close'].iloc[:-12]
y_test = df['Close'].iloc[-12:]
```

[30]:
```python
# Creating the LSTM model
model = Sequential()
model.add(LSTM(128, input_shape=(X_train.shape[1], X_train.shape[2])))
model.add(Dense(1))
```

[31]:
```python
# Compiling and training the model
model.compile(loss='mse', optimizer='adam')
model.fit(X_train, y_train, epochs=100)
```

```
Epoch 1/100
39/39 [==============================] - 3s 17ms/step - loss: 9889.9072
Epoch 2/100
39/39 [==============================] - 1s 16ms/step - loss: 6983.2480
```

```
Epoch 3/100
39/39 [==============================] - 1s 15ms/step - loss: 5853.0430
Epoch 4/100
39/39 [==============================] - 1s 15ms/step - loss: 5085.2173
Epoch 5/100
39/39 [==============================] - 1s 15ms/step - loss: 4446.2241
Epoch 6/100
39/39 [==============================] - 1s 15ms/step - loss: 3899.5444
Epoch 7/100
39/39 [==============================] - 1s 16ms/step - loss: 3429.3528
Epoch 8/100
39/39 [==============================] - 0s 10ms/step - loss: 3025.3835
Epoch 9/100
39/39 [==============================] - 0s 10ms/step - loss: 2676.2312
Epoch 10/100
39/39 [==============================] - 0s 9ms/step - loss: 2377.8486
Epoch 11/100
39/39 [==============================] - 0s 9ms/step - loss: 2121.3608
Epoch 12/100
39/39 [==============================] - 0s 9ms/step - loss: 1901.8618
Epoch 13/100
39/39 [==============================] - 0s 9ms/step - loss: 1718.3921
Epoch 14/100
39/39 [==============================] - 0s 10ms/step - loss: 1563.4797
Epoch 15/100
39/39 [==============================] - 0s 9ms/step - loss: 1433.0625
Epoch 16/100
39/39 [==============================] - 0s 9ms/step - loss: 1326.2688
Epoch 17/100
39/39 [==============================] - 0s 9ms/step - loss: 1237.3922
Epoch 18/100
39/39 [==============================] - 0s 9ms/step - loss: 1165.9475
Epoch 19/100
39/39 [==============================] - 0s 9ms/step - loss: 1106.8405
Epoch 20/100
39/39 [==============================] - 0s 9ms/step - loss: 1059.9854
Epoch 21/100
39/39 [==============================] - 0s 9ms/step - loss: 1022.0915
Epoch 22/100
39/39 [==============================] - 0s 9ms/step - loss: 992.4019
Epoch 23/100
39/39 [==============================] - 0s 9ms/step - loss: 968.3958
Epoch 24/100
39/39 [==============================] - 0s 9ms/step - loss: 949.0529
Epoch 25/100
39/39 [==============================] - 0s 9ms/step - loss: 929.7433
Epoch 26/100
39/39 [==============================] - 0s 10ms/step - loss: 671.0215
```

```
Epoch 27/100
39/39 [==============================] - 0s 9ms/step - loss: 540.9097
Epoch 28/100
39/39 [==============================] - 0s 9ms/step - loss: 475.3577
Epoch 29/100
39/39 [==============================] - 0s 9ms/step - loss: 419.2200
Epoch 30/100
39/39 [==============================] - 0s 9ms/step - loss: 370.2742
Epoch 31/100
39/39 [==============================] - 0s 9ms/step - loss: 327.7375
Epoch 32/100
39/39 [==============================] - 0s 9ms/step - loss: 290.9508
Epoch 33/100
39/39 [==============================] - 0s 9ms/step - loss: 258.6083
Epoch 34/100
39/39 [==============================] - 0s 10ms/step - loss: 230.8026
Epoch 35/100
39/39 [==============================] - 0s 12ms/step - loss: 205.4255
Epoch 36/100
39/39 [==============================] - 1s 15ms/step - loss: 183.4798
Epoch 37/100
39/39 [==============================] - 1s 15ms/step - loss: 164.2911
Epoch 38/100
39/39 [==============================] - 1s 15ms/step - loss: 146.6538
Epoch 39/100
39/39 [==============================] - 1s 15ms/step - loss: 131.3534
Epoch 40/100
39/39 [==============================] - 1s 16ms/step - loss: 118.0707
Epoch 41/100
39/39 [==============================] - 1s 15ms/step - loss: 106.1361
Epoch 42/100
39/39 [==============================] - 1s 15ms/step - loss: 95.2592
Epoch 43/100
39/39 [==============================] - 1s 13ms/step - loss: 85.4465
Epoch 44/100
39/39 [==============================] - 0s 8ms/step - loss: 76.8442
Epoch 45/100
39/39 [==============================] - 0s 9ms/step - loss: 68.9152
Epoch 46/100
39/39 [==============================] - 0s 9ms/step - loss: 62.1391
Epoch 47/100
39/39 [==============================] - 0s 9ms/step - loss: 56.0062
Epoch 48/100
39/39 [==============================] - 0s 9ms/step - loss: 50.5864
Epoch 49/100
39/39 [==============================] - 0s 9ms/step - loss: 46.2198
Epoch 50/100
39/39 [==============================] - 0s 9ms/step - loss: 41.5477
```

```
Epoch 51/100
39/39 [==============================] - 0s 10ms/step - loss: 37.7649
Epoch 52/100
39/39 [==============================] - 0s 9ms/step - loss: 34.4826
Epoch 53/100
39/39 [==============================] - 0s 9ms/step - loss: 31.1224
Epoch 54/100
39/39 [==============================] - 0s 9ms/step - loss: 28.4200
Epoch 55/100
39/39 [==============================] - 0s 10ms/step - loss: 26.0576
Epoch 56/100
39/39 [==============================] - 0s 9ms/step - loss: 23.8410
Epoch 57/100
39/39 [==============================] - 0s 10ms/step - loss: 21.8717
Epoch 58/100
39/39 [==============================] - 0s 9ms/step - loss: 20.2149
Epoch 59/100
39/39 [==============================] - 0s 9ms/step - loss: 18.5120
Epoch 60/100
39/39 [==============================] - 0s 9ms/step - loss: 17.0239
Epoch 61/100
39/39 [==============================] - 0s 9ms/step - loss: 15.6645
Epoch 62/100
39/39 [==============================] - 0s 9ms/step - loss: 14.5644
Epoch 63/100
39/39 [==============================] - 0s 9ms/step - loss: 13.6848
Epoch 64/100
39/39 [==============================] - 0s 9ms/step - loss: 12.6697
Epoch 65/100
39/39 [==============================] - 0s 9ms/step - loss: 11.7470
Epoch 66/100
39/39 [==============================] - 0s 9ms/step - loss: 10.8352
Epoch 67/100
39/39 [==============================] - 0s 9ms/step - loss: 10.5146
Epoch 68/100
39/39 [==============================] - 0s 9ms/step - loss: 9.5838
Epoch 69/100
39/39 [==============================] - 0s 9ms/step - loss: 9.3659
Epoch 70/100
39/39 [==============================] - 0s 9ms/step - loss: 8.9499
Epoch 71/100
39/39 [==============================] - 1s 15ms/step - loss: 8.3363
Epoch 72/100
39/39 [==============================] - 1s 15ms/step - loss: 8.0735
Epoch 73/100
39/39 [==============================] - 1s 16ms/step - loss: 7.3185
Epoch 74/100
39/39 [==============================] - 1s 14ms/step - loss: 6.5265
```

```
Epoch 75/100
39/39 [==============================] - 1s 16ms/step - loss: 6.6566
Epoch 76/100
39/39 [==============================] - 1s 16ms/step - loss: 6.0532
Epoch 77/100
39/39 [==============================] - 1s 15ms/step - loss: 5.7607
Epoch 78/100
39/39 [==============================] - 1s 15ms/step - loss: 5.4822
Epoch 79/100
39/39 [==============================] - 0s 9ms/step - loss: 5.5695
Epoch 80/100
39/39 [==============================] - 0s 9ms/step - loss: 5.0816
Epoch 81/100
39/39 [==============================] - 0s 9ms/step - loss: 5.0486
Epoch 82/100
39/39 [==============================] - 0s 9ms/step - loss: 4.7620
Epoch 83/100
39/39 [==============================] - 0s 9ms/step - loss: 4.4777
Epoch 84/100
39/39 [==============================] - 0s 9ms/step - loss: 4.2099
Epoch 85/100
39/39 [==============================] - 0s 9ms/step - loss: 4.0638
Epoch 86/100
39/39 [==============================] - 0s 9ms/step - loss: 3.9091
Epoch 87/100
39/39 [==============================] - 0s 9ms/step - loss: 3.7963
Epoch 88/100
39/39 [==============================] - 0s 9ms/step - loss: 3.5614
Epoch 89/100
39/39 [==============================] - 0s 9ms/step - loss: 3.5968
Epoch 90/100
39/39 [==============================] - 0s 9ms/step - loss: 3.5625
Epoch 91/100
39/39 [==============================] - 0s 8ms/step - loss: 3.4491
Epoch 92/100
39/39 [==============================] - 0s 9ms/step - loss: 3.5614
Epoch 93/100
39/39 [==============================] - 0s 9ms/step - loss: 3.2759
Epoch 94/100
39/39 [==============================] - 0s 9ms/step - loss: 3.2742
Epoch 95/100
39/39 [==============================] - 0s 9ms/step - loss: 3.1752
Epoch 96/100
39/39 [==============================] - 0s 9ms/step - loss: 2.8885
Epoch 97/100
39/39 [==============================] - 0s 10ms/step - loss: 3.0483
Epoch 98/100
39/39 [==============================] - 0s 9ms/step - loss: 2.9983
```

```
Epoch 99/100
39/39 [==============================] - 0s 9ms/step - loss: 2.9948
Epoch 100/100
39/39 [==============================] - 0s 9ms/step - loss: 2.8467
```

[31]:
```
<keras.src.callbacks.History at 0x7fccfd1dd0f0>
```

[32]:
```python
# Making predictions on the test set
y_pred = model.predict(X_test)
y_pred
```

```
1/1 [==============================] - 1s 751ms/step
```

[32]:
```
array([[139.79343],
       [140.95493],
       [140.74307],
       [139.72331],
       [139.43202],
       [140.45444],
       [137.35425],
       [133.48978],
       [131.18095],
       [131.24704],
       [132.9559 ],
       [139.20847]], dtype=float32)
```

[39]:
```python
# Reshaping y_pred to a 1D array
y_pred = y_pred.reshape(-1)
y_pred
```

[39]:
```
array([139.79343, 140.95493, 140.74307, 139.72331, 139.43202, 140.45444,
       137.35425, 133.48978, 131.18095, 131.24704, 132.9559 , 139.20847],
      dtype=float32)
```

[41]:
```python
# Let us calculate the mean squared error
mse = np.mean(np.square(y_pred - y_test))
print('Mean Squared Error:', mse)
```

```
Mean Squared Error: 3.737603401851672
```

[43]:
```python
# Let us calculate the mean absolute error
mae = mean_absolute_error(y_test, y_pred)
print('Mean Absolute Error:', mae)
```

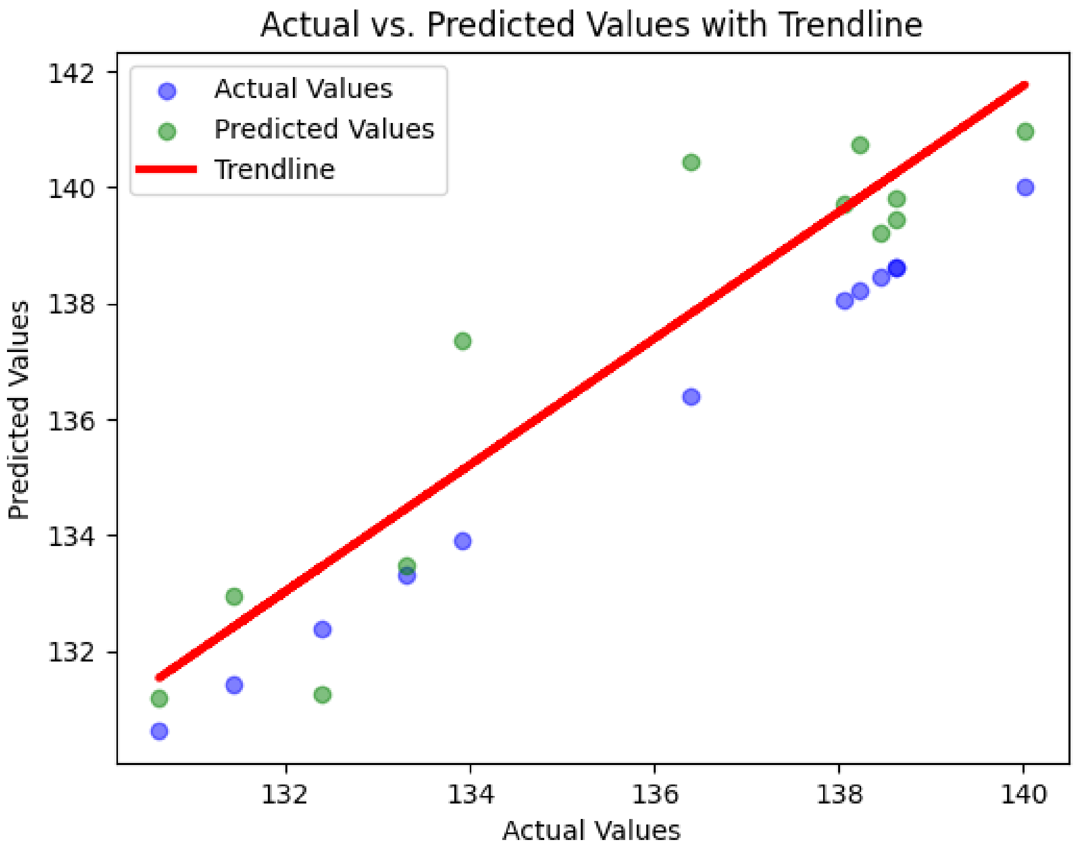
```
Mean Squared Error: 1.5627924839681004
```

[44]:
```python
# Let us calculate the R squared value
r2 = r2_score(y_test, y_pred)
```

```python
print('R-squared:', r2)
```

```
R-squared: 0.6235254479335808
```

[48]:
```python
# Let us plot predicted vs. actual values

plt.scatter(y_test, y_test, label='Actual Values', color='blue', alpha=0.5)


plt.scatter(y_test, y_pred, label='Predicted Values', color='green', alpha=0.5)


regression_line = LinearRegression()
regression_line.fit(y_test.values.reshape(-1, 1), y_pred)


plt.plot(y_test, regression_line.predict(y_test.values.reshape(-1, 1)),␣
↪color='red', linewidth=3, label='Trendline')

plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Actual vs. Predicted Values with Trendline')
plt.legend()
plt.show()
```


Actual vs. Predicted Values with Trendline
Actual Values
Predicted Values
Trendline
142
140
138
136
134
132
Predicted Values
132
134
136
138
140
Actual Values