stock

December 3, 2024

[4]:
```python
import pandas as pd
import matplotlib.pyplot as plt
from alpha_vantage.timeseries import TimeSeries
```

[5]:
```python
# Function to retrieve the API key from a file
def get_api_key(file_path):
    try:
        with open(file_path, "r") as file:
            return file.read().strip()
    except FileNotFoundError:
        print(f"Error: API key file not found at {file_path}")
        exit(1)
    except Exception as e:
        print(f"Error reading API key file: {e}")
        exit(1)

# Your YouTube API key
API_KEY_FILE = r"C:\API\alphavantage.txt"
api_key = get_api_key(API_KEY_FILE)
```

[6]:
```python
ticker_symbol = 'GOOG'

# Fetching live stock data
ts = TimeSeries(key=api_key, output_format='pandas')
data, meta_data = ts.get_intraday(symbol=ticker_symbol, interval='1min',␣
↪outputsize='full')
```

[7]:
```python
print(data.head())
```

```
                     1. open  2. high    3. low  4. close  5. volume
date
2024-12-02 19:59:00   172.92  173.000  172.8400  172.9800       19.0
2024-12-02 19:58:00   172.98  172.980  172.9200  172.9200        3.0
2024-12-02 19:57:00   172.97  172.996  172.8406  172.8406      284.0
2024-12-02 19:56:00   172.97  172.970  172.8400  172.9700        9.0
2024-12-02 19:55:00   172.97  172.970  172.9700  172.9700        5.0
```

[8]:
```python
print(data.tail())
```

```
                     1. open  2. high  3. low  4. close  5. volume
date
2024-11-04 04:04:00   172.65   172.66  172.65    172.65      162.0
2024-11-04 04:03:00   172.73   172.73  172.70    172.70       15.0
2024-11-04 04:02:00   172.72   172.74  172.65    172.74      455.0
2024-11-04 04:01:00   172.67   172.77  172.59    172.72      189.0
2024-11-04 04:00:00   172.36   173.00  172.36    172.67      977.0
```

[9]:
```
# Reverse the DataFrame rows
data = data.iloc[::-1]

# Display the reversed DataFrame
print(data.head())
```

```
                     1. open  2. high  3. low  4. close  5. volume
date
2024-11-04 04:00:00   172.36   173.00  172.36    172.67      977.0
2024-11-04 04:01:00   172.67   172.77  172.59    172.72      189.0
2024-11-04 04:02:00   172.72   172.74  172.65    172.74      455.0
2024-11-04 04:03:00   172.73   172.73  172.70    172.70       15.0
2024-11-04 04:04:00   172.65   172.66  172.65    172.65      162.0
```

[10]:
```
    # Simple Moving Average (SMA)
    data['SMA_20'] = data['4. close'].rolling(window=20).mean()
    data['SMA_50'] = data['4. close'].rolling(window=50).mean()

    # Exponential Moving Average (EMA)
    data['EMA_20'] = data['4. close'].ewm(span=20, adjust=False).mean()
    data['EMA_50'] = data['4. close'].ewm(span=50, adjust=False).mean()

    # Relative Strength Index (RSI)
    delta = data['4. close'].diff(1)
    gain = delta.where(delta > 0, 0)
    loss = -delta.where(delta < 0, 0)
    avg_gain = gain.rolling(window=14).mean()
    avg_loss = loss.rolling(window=14).mean()
    rs = avg_gain / avg_loss
    data['RSI'] = 100 - (100 / (1 + rs))

    # Bollinger Bands
    data['BB_upper'] = data['SMA_20'] + 2 * data['4. close'].rolling(window=20).
 ↪std()
    data['BB_lower'] = data['SMA_20'] - 2 * data['4. close'].rolling(window=20).
 ↪std()

    # Moving Average Convergence Divergence (MACD)
    data['MACD'] = data['EMA_12'] = data['4. close'].ewm(span=12, adjust=False).
 ↪mean() - data['4. close'].ewm(span=26, adjust=False).mean()
```

```
    data['MACD_signal'] = data['MACD'].ewm(span=9, adjust=False).mean()

    # Average True Range (ATR)
    data['High-Low'] = data['2. high'] - data['3. low']
    data['High-Close'] = abs(data['2. high'] - data['4. close'].shift(1))
    data['Low-Close'] = abs(data['3. low'] - data['4. close'].shift(1))
    data['TR'] = data[['High-Low', 'High-Close', 'Low-Close']].max(axis=1)
    data['ATR'] = data['TR'].rolling(window=14).mean()

    # Stochastic Oscillator
    data['L14'] = data['3. low'].rolling(window=14).min()
    data['H14'] = data['2. high'].rolling(window=14).max()
    data['%K'] = 100 * ((data['4. close'] - data['L14']) / (data['H14'] -␣
↪data['L14']))
    data['%D'] = data['%K'].rolling(window=3).mean()

    # Volume Weighted Average Price (VWAP)
    data['Cumulative_TP'] = (data['4. close'] + data['2. high'] + data['3.␣
↪low']) / 3
    data['Cumulative_Volume'] = data['5. volume'].cumsum()
    data['Cumulative_TPV'] = (data['Cumulative_TP'] * data['5. volume']).
↪cumsum()
    data['VWAP'] = data['Cumulative_TPV'] / data['Cumulative_Volume']

    # Drop rows with NaN values
    data.dropna(inplace=True)

    # Display the enriched data
    print(data[['4. close', 'SMA_20', 'SMA_50', 'EMA_20', 'RSI', 'BB_upper',␣
↪'BB_lower', 'MACD', 'ATR', '%K', '%D', 'VWAP']].head())
```

```
                     4. close    SMA_20    SMA_50      EMA_20        RSI  \
date
2024-11-04 04:56:00    172.55  172.3925  172.4136  172.430430  83.333333
2024-11-04 04:57:00    172.54  172.4070  172.4110  172.440865  80.000000
2024-11-04 04:58:00    172.54  172.4215  172.4074  172.450307  77.777778
2024-11-04 04:59:00    172.53  172.4315  172.4032  172.457897  75.000000
2024-11-04 05:00:00    172.53  172.4405  172.3998  172.464764  77.142857

                       BB_upper    BB_lower      MACD       ATR         %K  \
date
2024-11-04 04:56:00  172.588123  172.196877  0.040951  0.041429  92.592593
2024-11-04 04:57:00  172.601135  172.212865  0.044088  0.040000  86.956522
2024-11-04 04:58:00  172.609484  172.233516  0.046043  0.037857  86.956522
2024-11-04 04:59:00  172.620266  172.242734  0.046252  0.037857  82.608696
2024-11-04 05:00:00  172.630067  172.250933  0.045890  0.034286  81.818182
```

```
                           %D        VWAP
date
2024-11-04 04:56:00  85.308642  172.455547
2024-11-04 04:57:00  85.405260  172.455555
2024-11-04 04:58:00  88.835212  172.456611
2024-11-04 04:59:00  85.507246  172.456625
2024-11-04 05:00:00  83.794466  172.457857
```

[12]:

```python
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM
from sklearn.metrics import mean_squared_error, mean_absolute_error
import matplotlib.pyplot as plt

# Step 1: Feature Engineering
data['Price_Change'] = data['4. close'] - data['1. open']
data['High_Low_Diff'] = data['2. high'] - data['3. low']
data['SMA_Diff'] = data['SMA_20'] - data['SMA_50']
data['EMA_Diff'] = data['EMA_20'] - data['EMA_50']

# Check for missing values and drop if needed
data.dropna(inplace=True)

# Select features and target
features = ['4. close', 'Price_Change', 'High_Low_Diff', 'SMA_Diff',␣
 ↪'EMA_Diff', 'RSI', 'VWAP']
target = '4. close'

# Ensure all selected columns exist
for feature in features:
    if feature not in data.columns:
        raise ValueError(f"Feature {feature} is missing in the dataset.")

# Scale the features
scaler = MinMaxScaler()
data_scaled = scaler.fit_transform(data[features])

# Step 2: Prepare data for LSTM
sequence_length = 90
X, y, indices = [], [], []

for i in range(sequence_length, len(data_scaled)):
    X.append(data_scaled[i-sequence_length:i])
    y.append(data_scaled[i, 0])  # Target column corresponds to '4. close'
    indices.append(i)
```

```python
X, y = np.array(X), np.array(y)
timestamps_test_raw = data.index[indices]  # Raw indices for validation

# Step 3: Train-Test Split
train_size = int(len(X) * 0.95)
X_train, X_test = X[:train_size], X[train_size:]
y_train, y_test = y[:train_size], y[train_size:]

# Directly use timestamps_test_raw for validation
timestamps_test_full = data.index[indices]  # Ensure valid indices
timestamps_test = pd.to_datetime(timestamps_test_full[-len(y_test):])  # Align with test set size


# Step 4: Build LSTM Model
model = Sequential([
    LSTM(50, return_sequences=True, input_shape=(X_train.shape[1], X_train.shape[2])),
    LSTM(50, return_sequences=False),
    Dense(25),
    Dense(1)
])

model.compile(optimizer='adam', loss='mean_squared_error')

# Train the Model
history = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=20, batch_size=32, verbose=1)

# Step 5: Evaluate the Model
y_pred = model.predict(X_test)

# Inverse transform the predictions and actual values
y_test_actual = scaler.inverse_transform(
    np.concatenate((y_test.reshape(-1, 1), np.zeros((len(y_test), len(features) - 1))), axis=1)
)[:, 0]

y_pred_actual = scaler.inverse_transform(
    np.concatenate((y_pred, np.zeros((len(y_pred), len(features) - 1))), axis=1)
)[:, 0]

# Ensure lengths match
assert len(timestamps_test) == len(y_test_actual), "Mismatch in timestamps and test set sizes."
```

```python
assert len(timestamps_test) == len(y_pred_actual), "Mismatch in timestamps and␣
 ↪predicted set sizes."

# Evaluation Metrics
mse = mean_squared_error(y_test_actual, y_pred_actual)
mae = mean_absolute_error(y_test_actual, y_pred_actual)

print(f"Mean Squared Error (MSE): {mse}")
print(f"Mean Absolute Error (MAE): {mae}")
```

```
Epoch 1/20
531/531 [==============================] - 27s 46ms/step - loss: 0.0026 -
val_loss: 4.1377e-05
Epoch 2/20
531/531 [==============================] - 24s 45ms/step - loss: 1.5587e-04 -
val_loss: 5.5934e-05
Epoch 3/20
531/531 [==============================] - 24s 44ms/step - loss: 1.6401e-04 -
val_loss: 3.5930e-05
Epoch 4/20
531/531 [==============================] - 24s 45ms/step - loss: 1.6002e-04 -
val_loss: 3.7698e-04
Epoch 5/20
531/531 [==============================] - 23s 44ms/step - loss: 1.6769e-04 -
val_loss: 3.4734e-05
Epoch 6/20
531/531 [==============================] - 24s 45ms/step - loss: 1.8308e-04 -
val_loss: 6.0909e-05
Epoch 7/20
531/531 [==============================] - 24s 45ms/step - loss: 1.4963e-04 -
val_loss: 2.9650e-05
Epoch 8/20
531/531 [==============================] - 25s 47ms/step - loss: 1.4095e-04 -
val_loss: 2.9583e-05
Epoch 9/20
531/531 [==============================] - 29s 54ms/step - loss: 1.6570e-04 -
val_loss: 7.1678e-05
Epoch 10/20
531/531 [==============================] - 26s 49ms/step - loss: 1.5180e-04 -
val_loss: 7.9843e-05
Epoch 11/20
531/531 [==============================] - 25s 47ms/step - loss: 1.5613e-04 -
val_loss: 2.4989e-05
Epoch 12/20
531/531 [==============================] - 25s 47ms/step - loss: 1.3831e-04 -
val_loss: 4.6963e-05
Epoch 13/20
531/531 [==============================] - 25s 47ms/step - loss: 1.6996e-04 -
```

```
val_loss: 2.3738e-05
Epoch 14/20
531/531 [==============================] - 24s 46ms/step - loss: 1.3383e-04 -
val_loss: 2.4472e-05
Epoch 15/20
531/531 [==============================] - 33s 61ms/step - loss: 1.3314e-04 -
val_loss: 6.8081e-05
Epoch 16/20
531/531 [==============================] - 25s 47ms/step - loss: 1.3494e-04 -
val_loss: 1.9014e-05
Epoch 17/20
531/531 [==============================] - 25s 47ms/step - loss: 1.3962e-04 -
val_loss: 4.6303e-05
Epoch 18/20
531/531 [==============================] - 25s 47ms/step - loss: 1.2426e-04 -
val_loss: 1.0449e-04
Epoch 19/20
531/531 [==============================] - 25s 47ms/step - loss: 1.3680e-04 -
val_loss: 3.7834e-05
Epoch 20/20
531/531 [==============================] - 28s 52ms/step - loss: 1.2397e-04 -
val_loss: 5.6900e-05
28/28 [==============================] - 1s 19ms/step
Mean Squared Error (MSE): 0.035392514068285 65
Mean Absolute Error (MAE): 0.160835339657064
```

[18]:

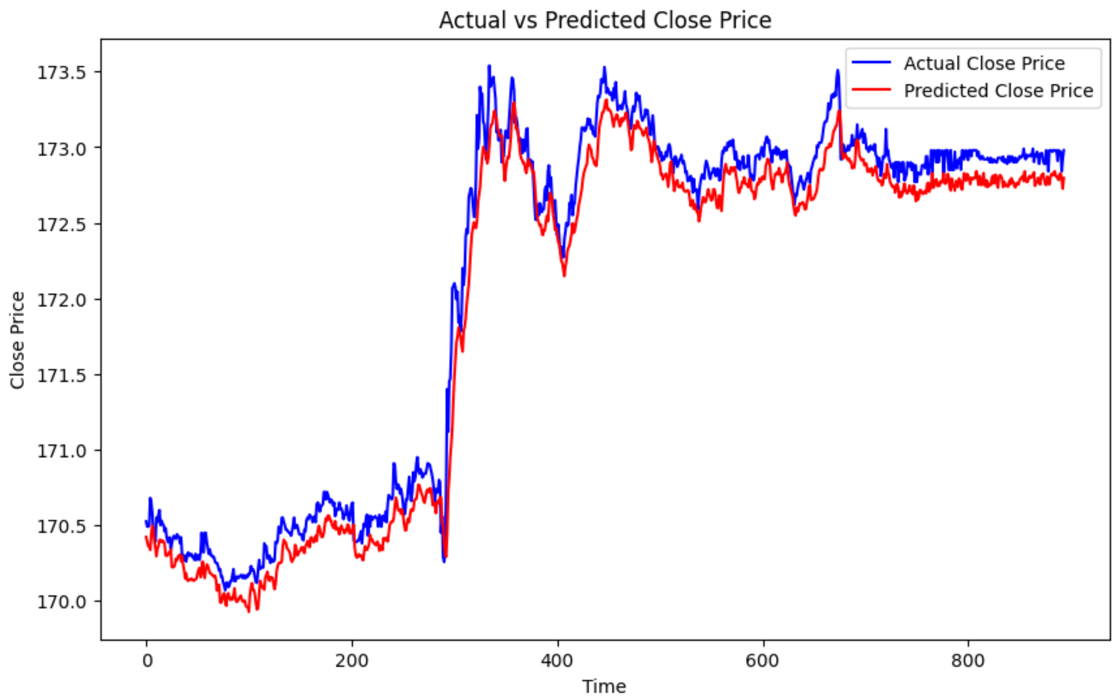
```python
from math import sqrt
from sklearn.metrics import (
    mean_squared_error,
    mean_absolute_error,
    r2_score,
    median_absolute_error,
    explained_variance_score
)
import numpy as np

# Evaluation Metrics
# 1. Mean Squared Error (MSE)
mse = mean_squared_error(y_test_actual, y_pred_actual)

# 2. Mean Absolute Error (MAE)
mae = mean_absolute_error(y_test_actual, y_pred_actual)

# 3. Root Mean Squared Error (RMSE)
rmse = sqrt(mse)

# 4. R-squared (R2)
```

```python
r2 = r2_score(y_test_actual, y_pred_actual)

# 5. Mean Absolute Percentage Error (MAPE)
mape = np.mean(np.abs((y_test_actual - y_pred_actual) / y_test_actual)) * 100

# 6. Symmetric Mean Absolute Percentage Error (SMAPE)
smape = 100 * np.mean(2 * np.abs(y_test_actual - y_pred_actual) / (np.
↪abs(y_test_actual) + np.abs(y_pred_actual)))

# 7. Median Absolute Error (MedAE)
medae = median_absolute_error(y_test_actual, y_pred_actual)

# 8. Explained Variance Score (EVS)
evs = explained_variance_score(y_test_actual, y_pred_actual)

# 9. Mean Bias Error (MBE)
mbe = np.mean(y_pred_actual - y_test_actual)

# Print all metrics
print(f"Mean Squared Error (MSE): {mse}")
print(f"Mean Absolute Error (MAE): {mae}")
print(f"Root Mean Squared Error (RMSE): {rmse}")
print(f"R-squared (R2): {r2}")
print(f"Mean Absolute Percentage Error (MAPE): {mape}%")
print(f"Symmetric Mean Absolute Percentage Error (SMAPE): {smape}%")
print(f"Median Absolute Error (MedAE): {medae}")
print(f"Explained Variance Score (EVS): {evs}")
print(f"Mean Bias Error (MBE): {mbe}")
```

```
Mean Squared Error (MSE): 0.03539251406828565
Mean Absolute Error (MAE): 0.160835339657064
Root Mean Squared Error (RMSE): 0.18812898253136237
R-squared (R2): 0.9745494048309522
Mean Absolute Percentage Error (MAPE): 0.09341532350217809%
Symmetric Mean Absolute Percentage Error (SMAPE): 0.09347394142568574%
Median Absolute Error (MedAE): 0.14776323421003212
Explained Variance Score (EVS): 0.9921448834507484
Mean Bias Error (MBE): -0.15642540122210452
```

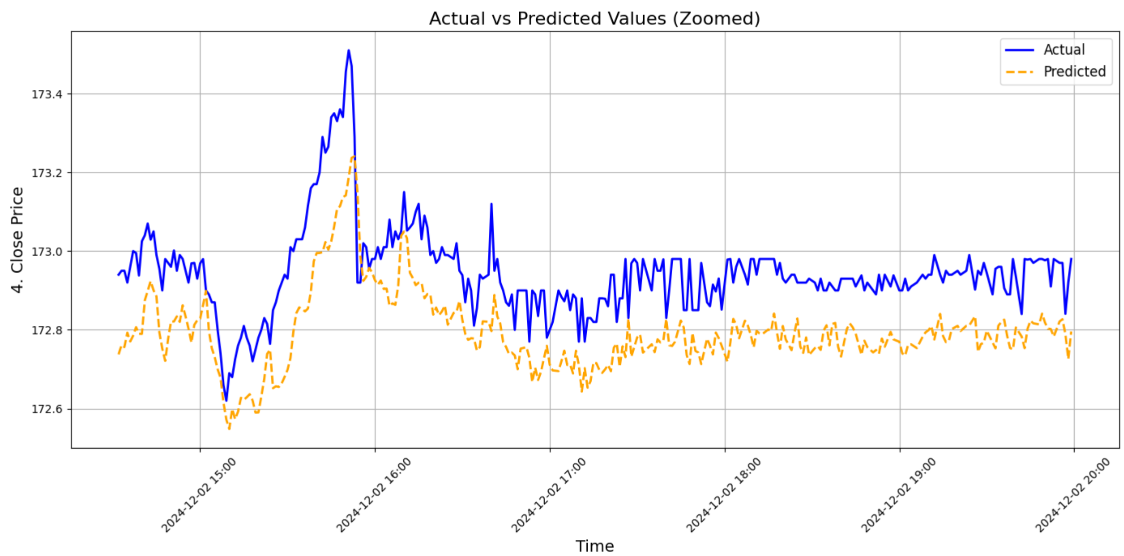
[13]:
```python
# Step 6: Visualize Predictions
plt.figure(figsize=(10, 6))
plt.plot(y_test_actual, label='Actual Close Price', color='blue')
plt.plot(y_pred_actual, label='Predicted Close Price', color='red')
plt.title('Actual vs Predicted Close Price')
plt.xlabel('Time')
plt.ylabel('Close Price')
plt.legend()
```

```
plt.show()
```



[16]:
```python
import matplotlib.dates as mdates

# Step 6: Improved Plot for Actual vs Predicted Values
plt.figure(figsize=(14, 7))

# Focus on relevant data (optional - adjust slicing to zoom in)
relevant_range = slice(-300, None)  # Customize the slice based on your data
timestamps_zoomed = timestamps_test[relevant_range]
y_test_actual_zoomed = y_test_actual[relevant_range]
y_pred_actual_zoomed = y_pred_actual[relevant_range]

# Plot actual and predicted values
plt.plot(timestamps_zoomed, y_test_actual_zoomed, label='Actual', linewidth=2,␣
 ↪color='blue')
plt.plot(timestamps_zoomed, y_pred_actual_zoomed, label='Predicted',␣
 ↪linestyle='--', linewidth=2, color='orange')

# Formatting the plot
plt.title('Actual vs Predicted Values (Zoomed)', fontsize=16)
plt.xlabel('Time', fontsize=14)
plt.ylabel('4. Close Price', fontsize=14)
plt.legend(fontsize=12)
```

```
plt.grid(True)

# Rotate x-axis labels and format them
plt.xticks(rotation=45)
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-%d %H:%M'))
plt.gca().xaxis.set_major_locator(mdates.AutoDateLocator(maxticks=6))  #␣
↪Further reduce ticks

# Ensure proper layout
plt.tight_layout()

# Show the plot
plt.show()
```



[17]:
```
import plotly.graph_objects as go
from plotly.subplots import make_subplots

# Focus on relevant data (optional - adjust slicing to zoom in)
relevant_range = slice(-300, None)  # Customize the slice based on your data
timestamps_zoomed = timestamps_test[relevant_range]
y_test_actual_zoomed = y_test_actual[relevant_range]
y_pred_actual_zoomed = y_pred_actual[relevant_range]

# Create the figure
fig = make_subplots()

# Add actual values
fig.add_trace(go.Scatter(
```

```
    x=timestamps_zoomed,
    y=y_test_actual_zoomed,
    mode='lines',
    name='Actual',
    line=dict(color='blue', width=2)
))

# Add predicted values
fig.add_trace(go.Scatter(
    x=timestamps_zoomed,
    y=y_pred_actual_zoomed,
    mode='lines',
    name='Predicted',
    line=dict(color='orange', width=2, dash='dash')
))

# Update layout
fig.update_layout(
    title='Actual vs Predicted Values (Zoomed)',
    xaxis_title='Time',
    yaxis_title='4. Close Price',
    legend=dict(font=dict(size=12)),
    xaxis=dict(
        showgrid=True,
        showline=True,
        tickformat='%Y-%m-%d %H:%M',  # Format for x-axis ticks
        tickangle=45  # Rotate ticks for better readability
    ),
    yaxis=dict(showgrid=True),
    margin=dict(l=40, r=20, t=40, b=40),
    height=600,
    width=1000,
)

# Show the figure
fig.show()
```