datavinci-private-ltd-assignment

June 23, 2023

1 System Requirements

1.0.1 Jupyter Notebook

[13]:
```
import notebook as nb
version=nb.__version__
print("Jupyter Notebook Version:",version)
```

```
Jupyter Notebook Version: 6.4.8
```

1.0.2 yfinance

[14]:
```
import yfinance as yf
version = yf.__version__
print("yfinance version:", version)
```

```
yfinance version: 0.2.21
```

1.0.3 Pandas

[15]:
```
import pandas as pd
version=pd.__version__
print("Pandas Version:",version)
```

```
Pandas Version: 2.0.1
```

1.0.4 Plotly

[16]:
```
pip show plotly
```

```
Name: plotly
Version: 5.14.1
Summary: An open-source, interactive data visualization library for Python
Home-page: https://plotly.com/python/
Author: Chris P
Author-email: chris@plot.ly
License: MIT
Location: c:\users\ritesh\anaconda3\lib\site-packages
Requires: tenacity, packaging
```

```
Required-by:
Note: you may need to restart the kernel to use updated packages.

WARNING: Ignoring invalid distribution -treamlit
(c:\users\ritesh\anaconda3\lib\site-packages)
```

2 Import the required libraries

[4]:
```
import yfinance as yf
import pandas as pd
import plotly.graph_objects as go
```

2.0.1 Step 1: Data Extraction

[17]:
```
def extract_stock_data(symbol, start_date, end_date):
    try:
        # Use yfinance to extract historical stock data
        data = yf.download(symbol, start=start_date, end=end_date)
        return data
    except Exception as e:
        print("Error occurred during data extraction:", e)
        return None

# Specify the stock symbol, start date, and end date for data extraction
stock_symbol = "RELIANCE.NS"
start_date = "2022-01-01"
end_date = "2023-01-01"

# Extract stock data
stock_data = extract_stock_data(stock_symbol, start_date, end_date)
```

```
[*********************100%***********************]  1 of 1 completed
```

2.0.2 Step 2: Data Processing

[20]:
```
if stock_data is not None:

    # Calculate average daily trading volume
    average_volume = stock_data['Volume'].mean()
    print("Average Daily Trading Volume:", average_volume)
```

```
Average Daily Trading Volume: 6399582.903225807
```

2.0.3 Step 3: Data Visualization

[21]:
```
# Create a candlestick chart
fig = go.Figure(data=[go.Candlestick(x=stock_data.index,
                                      open=stock_data['Open'],
                                      high=stock_data['High'],
                                      low=stock_data['Low'],
                                      close=stock_data['Close'])])

# Customize the chart layout
fig.update_layout(title="Stock Performance",
                  xaxis_title="Date",
                  yaxis_title="Price",
                  template="plotly_dark")

# Display the chart
fig.show()
```

2.0.4 Step 4: Documentation

This Python script accomplishes the task of extracting historical stock data for a publicly traded company listed on the National Stock Exchange (NSE) of India, processing the data to calculate the average daily trading volume, and visualizing the stock's performance over the selected period using a candlestick chart created with Plotly.

The script starts by defining the function extract_stock_data that utilizes the yfinance library to extract historical stock data. Error handling is implemented within the function to catch any exceptions that might occur during data extraction.

Next, the stock symbol, start date, and end date are specified for data extraction.

The script calls the extract_stock_data function and assigns the returned data to the stock_data variable.

If the data extraction is successful, the script proceeds to the data processing step. Using pandas, the script calculates the average daily trading volume from the extracted stock data.

In the data visualization step, a candlestick chart is created using Plotly. The chart's layout is customized with appropriate titles and a dark theme.

Finally, the chart is displayed using fig.show(), and the average daily trading volume is printed to the console.

Challenges that i faced during this task: One challenge encountered was handling potential errors during data extraction. To address this, error handling measures were implemented using try-except blocks to catch and handle any exceptions that might occur.