Connecting MongoDB to Python Notebook

```
!pip install pymongo
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-whe
Collecting pymongo
  Downloading pymongo-4.3.3-cp39-cp39-manylinux_2_17_x86_64.manylinux2014_x86_64
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 492.1/492.1 kB 4.0 MB/s eta 0:00:00
Collecting dnspython<3.0.0,>=1.16.0
  Downloading dnspython-2.3.0-py3-none-any.whl (283 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 283.7/283.7 kB 11.1 MB/s eta 0:00:00
Installing collected packages: dnspython, pymongo
Successfully installed dnspython-2.3.0 pymongo-4.3.3
```

```
import pandas as pd
import pymongo
import urllib.request
```

```
from pymongo.mongo_client import MongoClient
from pymongo.server_api import ServerApi

uri = "mongodb+srv://amhaske:1234@cluster0.miyzrzf.mongodb.net/?retryWrites=true&w=

# Create a new client and connect to the server
client = MongoClient(uri, server_api=ServerApi('1'))

# Send a ping to confirm a successful connection
try:
    client.admin.command('ping')
    print("Pinged your deployment. You successfully connected to MongoDB!")
except Exception as e:
    print(e)
```

```
Pinged your deployment. You successfully connected to MongoDB!
```

Data Exploration

```
# Download CSV file from internet
url = "https://raw.githubusercontent.com/plotly/datasets/master/finance-charts-appl
filename = "finance-charts-apple.csv"
urllib.request.urlretrieve(url, filename)

# Load CSV data into DataFrame
df = pd.read_csv(filename)
```

```
df.head()
```

	_id	AAPL.Open	AAPL.High	AAPL.Low	AAPL.Clos
0	6438cb603556ed1bc6bfdb23	127.489998	128.880005	126.919998	127.83000
1	6438cb603556ed1bc6bfdb24	127.629997	128.779999	127.449997	128.72000
2	6438cb603556ed1bc6bfdb25	128.479996	129.029999	128.330002	128.44999
3	6438cb603556ed1bc6bfdb26	128.619995	129.500000	128.050003	129.50000
4	6438cb603556ed1bc6bfdb27	130.020004	133.000000	129.660004	133.00000

```
print(df.columns.values)
```

```
['AAPL.Open' 'AAPL.High' 'AAPL.Low' 'AAPL.Close' 'AAPL.Volume'
 'AAPL.Adjusted' 'dn' 'mavg' 'up' 'direction']
```

```
# Convert data to pandas dataframe
#df = pd.DataFrame(data)

# Display basic information about the dataframe
print("Number of rows:", len(df))
print("Number of columns:", len(df.columns))
print("Data types:", df.dtypes)

# Display summary statistics of numerical columns
print(df.describe())
```

```
Number of rows: 17968
Number of columns: 7
Data types: _id           object
Open         float64
High         float64
Low          float64
Close        float64
Adj Close    float64
Volume         int64
dtype: object
               Open          High           Low         Close     Adj Close  \
count  17968.000000  17968.000000  17968.000000  17968.000000  17968.000000   
mean      49.390976     49.926744     48.844517     49.404797     43.865943   
std       69.853521     70.584939     69.085598     69.876112     70.115855   
min        0.088542      0.092014      0.088542      0.090278      0.056324   
25%        7.531250      7.583985      7.437500      7.537110      4.702391   
50%       27.350000     27.653125     27.127500     27.382500     19.430847   
75%       44.757812     45.219063     44.188125     44.795626     35.131214   
max      344.619995    349.670013    342.200012    343.109985    339.075562   

             Volume
```

```
count  1.796800e+04
mean   5.750879e+07
std    3.686449e+07
min    2.304000e+06
25%    3.368915e+07
50%    5.099990e+07
75%    7.123635e+07
max    1.031789e+09
```

Data Cleaning and Data preprocessing

```
# Clean and preprocess data
df["Date"] = pd.to_datetime(df["Date"])
df.set_index("Date", inplace=True)
df.sort_index(inplace=True)
```

MongoDB Exploration

```
# Connect to MongoDB database
client = pymongo.MongoClient(uri, server_api=ServerApi('1'))
db = client["financial_data"]
collection = db["stock_data"]

# Insert financial data into MongoDB collection
collection.insert_many(df.to_dict('records'))
```

```
<pymongo.results.InsertManyResult at 0x7f9ce5a1bfd0>
```

```
# Retrieve financial data from MongoDB collection
cursor = collection.find({})
df = pd.DataFrame(list(cursor))
```

```
cursor = collection.find_one({})
print(cursor.keys())
```

```
dict_keys(['_id', 'AAPL.Open', 'AAPL.High', 'AAPL.Low', 'AAPL.Close', 'AAPL.Volu
```

CRUD Operations

Insert: Inserting Data into the collection

```
# create a dictionary containing the data to insert
data = {
    'AAPL.Open': 142.11,
    'AAPL.High': 142.15,
    'AAPL.Low': 141.78,
    'AAPL.Close': 141.96,
    'AAPL.Volume': 54106570,
    'AAPL.Adjusted': 141.96,
    'dn': 139.21,
    'mavg': 140.88,
    'up': 142.56,
    'direction': 1
}

# insert the data into the collection
insert_result = collection.insert_one(data)

# print the ID of the inserted document
print(insert_result.inserted_id)

```

```
6438e2333556ed1bc6bfdd1d
```

Read: Reading Data from Collection

```
# find all documents in the collection
documents = collection.find()

# print each document
for document in documents:
    print(document)

```

```
{'_id': ObjectId('6438cb603556ed1bc6bfdb23'), 'AAPL.Open': 127.489998, 'AAPL.Hi
{'_id': ObjectId('6438cb603556ed1bc6bfdb24'), 'AAPL.Open': 127.629997, 'AAPL.Hi
{'_id': ObjectId('6438cb603556ed1bc6bfdb25'), 'AAPL.Open': 128.479996, 'AAPL.Hi
{'_id': ObjectId('6438cb603556ed1bc6bfdb26'), 'AAPL.Open': 128.619995, 'AAPL.Hi
{'_id': ObjectId('6438cb603556ed1bc6bfdb27'), 'AAPL.Open': 130.020004, 'AAPL.Hi
{'_id': ObjectId('6438cb603556ed1bc6bfdb28'), 'AAPL.Open': 132.940002, 'AAPL.Hi
{'_id': ObjectId('6438cb603556ed1bc6bfdb29'), 'AAPL.Open': 131.559998, 'AAPL.Hi
{'_id': ObjectId('6438cb603556ed1bc6bfdb2a'), 'AAPL.Open': 128.789993, 'AAPL.Hi
{'_id': ObjectId('6438cb603556ed1bc6bfdb2b'), 'AAPL.Open': 130.0, 'AAPL.High': 
{'_id': ObjectId('6438cb603556ed1bc6bfdb2c'), 'AAPL.Open': 129.25, 'AAPL.High':
{'_id': ObjectId('6438cb603556ed1bc6bfdb2d'), 'AAPL.Open': 128.960007, 'AAPL.Hi
{'_id': ObjectId('6438cb603556ed1bc6bfdb2e'), 'AAPL.Open': 129.100006, 'AAPL.Hi
{'_id': ObjectId('6438cb603556ed1bc6bfdb2f'), 'AAPL.Open': 128.580002, 'AAPL.Hi
{'_id': ObjectId('6438cb603556ed1bc6bfdb30'), 'AAPL.Open': 128.399994, 'AAPL.Hi
{'_id': ObjectId('6438cb603556ed1bc6bfdb31'), 'AAPL.Open': 127.959999, 'AAPL.Hi
{'_id': ObjectId('6438cb603556ed1bc6bfdb32'), 'AAPL.Open': 126.410004, 'AAPL.Hi
{'_id': ObjectId('6438cb603556ed1bc6bfdb33'), 'AAPL.Open': 124.75, 'AAPL.High':
{'_id': ObjectId('6438cb603556ed1bc6bfdb34'), 'AAPL.Open': 122.309998, 'AAPL.Hi
{'_id': ObjectId('6438cb603556ed1bc6bfdb35'), 'AAPL.Open': 124.400002, 'AAPL.Hi
{'_id': ObjectId('6438cb603556ed1bc6bfdb36'), 'AAPL.Open': 123.879997, 'AAPL.Hi
{'_id': ObjectId('6438cb603556ed1bc6bfdb37'), 'AAPL.Open': 125.900002, 'AAPL.Hi
```

```
{'_id': ObjectId('6438cb603556ed1bc6bfdb38'), 'AAPL.Open': 127.0, 'AAPL.High':
{'_id': ObjectId('6438cb603556ed1bc6bfdb39'), 'AAPL.Open': 128.75, 'AAPL.High':
{'_id': ObjectId('6438cb603556ed1bc6bfdb3a'), 'AAPL.Open': 128.25, 'AAPL.High':
{'_id': ObjectId('6438cb603556ed1bc6bfdb3b'), 'AAPL.Open': 127.120003, 'AAPL.Hi
{'_id': ObjectId('6438cb603556ed1bc6bfdb3c'), 'AAPL.Open': 127.230003, 'AAPL.Hi
{'_id': ObjectId('6438cb603556ed1bc6bfdb3d'), 'AAPL.Open': 126.540001, 'AAPL.Hi
{'_id': ObjectId('6438cb603556ed1bc6bfdb3e'), 'AAPL.Open': 122.760002, 'AAPL.Hi
{'_id': ObjectId('6438cb603556ed1bc6bfdb3f'), 'AAPL.Open': 124.57, 'AAPL.High':
{'_id': ObjectId('6438cb603556ed1bc6bfdb40'), 'AAPL.Open': 124.050003, 'AAPL.Hi
{'_id': ObjectId('6438cb603556ed1bc6bfdb41'), 'AAPL.Open': 126.089996, 'AAPL.Hi
{'_id': ObjectId('6438cb603556ed1bc6bfdb42'), 'AAPL.Open': 124.82, 'AAPL.High':
{'_id': ObjectId('6438cb603556ed1bc6bfdb43'), 'AAPL.Open': 125.029999, 'AAPL.Hi
{'_id': ObjectId('6438cb603556ed1bc6bfdb44'), 'AAPL.Open': 124.470001, 'AAPL.Hi
{'_id': ObjectId('6438cb603556ed1bc6bfdb45'), 'AAPL.Open': 127.639999, 'AAPL.Hi
{'_id': ObjectId('6438cb603556ed1bc6bfdb46'), 'AAPL.Open': 125.849998, 'AAPL.Hi
{'_id': ObjectId('6438cb603556ed1bc6bfdb47'), 'AAPL.Open': 125.849998, 'AAPL.Hi
{'_id': ObjectId('6438cb603556ed1bc6bfdb48'), 'AAPL.Open': 125.949997, 'AAPL.Hi
{'_id': ObjectId('6438cb603556ed1bc6bfdb49'), 'AAPL.Open': 128.369995, 'AAPL.Hi
{'_id': ObjectId('6438cb603556ed1bc6bfdb4a'), 'AAPL.Open': 127.0, 'AAPL.High':
{'_id': ObjectId('6438cb603556ed1bc6bfdb4b'), 'AAPL.Open': 126.410004, 'AAPL.Hi
{'_id': ObjectId('6438cb603556ed1bc6bfdb4c'), 'AAPL.Open': 126.279999, 'AAPL.Hi
{'_id': ObjectId('6438cb603556ed1bc6bfdb4d'), 'AAPL.Open': 125.550003, 'AAPL.Hi
{'_id': ObjectId('6438cb603556ed1bc6bfdb4e'), 'AAPL.Open': 125.57, 'AAPL.High':
{'_id': ObjectId('6438cb603556ed1bc6bfdb4f'), 'AAPL.Open': 128.100006, 'AAPL.Hi
{'_id': ObjectId('6438cb603556ed1bc6bfdb50'), 'AAPL.Open': 126.989998, 'AAPL.Hi
{'_id': ObjectId('6438cb603556ed1bc6bfdb51'), 'AAPL.Open': 128.300003, 'AAPL.Hi
{'_id': ObjectId('6438cb603556ed1bc6bfdb52'), 'AAPL.Open': 130.490005, 'AAPL.Hi
{'_id': ObjectId('6438cb603556ed1bc6bfdb53'), 'AAPL.Open': 132.309998, 'AAPL.Hi
{'_id': ObjectId('6438cb603556ed1bc6bfdb54'), 'AAPL.Open': 134.460007, 'AAPL.Hi
{'_id': ObjectId('6438cb603556ed1bc6bfdb55'), 'AAPL.Open': 130.160004, 'AAPL.Hi
{'_id': ObjectId('6438cb603556ed1bc6bfdb56'), 'AAPL.Open': 128.639999, 'AAPL.Hi
{'_id': ObjectId('6438cb603556ed1bc6bfdb57'), 'AAPL.Open': 126.099998, 'AAPL.Hi
{'_id': ObjectId('6438cb603556ed1bc6bfdb58'), 'AAPL.Open': 129.5, 'AAPL.High':
{'_id': ObjectId('6438cb603556ed1bc6bfdb59'), 'AAPL.Open': 128.149994, 'AAPL.Hi
{'_id': ObjectId('6438cb603556ed1bc6bfdb5a'), 'AAPL.Open': 126.559998, 'AAPL.Hi
{'_id': ObjectId('6438cb603556ed1bc6bfdb5b'), 'AAPL.Open': 124.769997, 'AAPL.Hi
```

Update: Updating Data in the collection

```
# find a document by a query
query = {'AAPL.Open': 142.11}
document = collection.find_one(query)

# update the document
new_data = {'$set': {'AAPL.Open': 143.22}}
update_result = collection.update_one(query, new_data)

# print the number of documents updated
print(update_result.modified_count)

```

```
0
```

Delete: Deleting code from Collection

```
# delete a document by a query
query = {'AAPL.Open': 142.11}
delete_result = collection.delete_one(query)

# print the number of documents deleted
print(delete_result.deleted_count)
```

```
0
```

Data Pipeline

This pipeline first projects the month, year, and Close columns from the aapl collection. It then groups the data by month and year, and calculates the average close price for each group. Finally, it sorts the results by year and month.

This pipeline can be used to analyze the average close price for each month, which can be useful for identifying trends and patterns in the data.

```
# Pipeline to calculate the average close price for each month
pipeline = [
    {
        "$project": {
            "_id": 0,
            "month": {"$month": {"$dateFromString": {"dateString": "$Date", "format
            "year": {"$year": {"$dateFromString": {"dateString": "$Date", "format":
            "Close": 1
        }
    },
    {
        "$group": {
            "_id": {"month": "$month", "year": "$year"},
            "average_close": {"$avg": "$Close"}
        }
    },
    {
        "$sort": {"_id.year": 1, "_id.month": 1}
    }
]

result = db.aapl.aggregate(pipeline)

for doc in result:
    print(doc)
```

Connecting to a MongoDB server and database called financial_data. We then define a pipeline using the group and sort aggregation operators to group the data by the direction field and calculate the total volume for each group. The results are then stored in a Pandas DataFrame for further analysis or visualization.

```
# Aggregate pipeline
pipeline = [
    {"$group": {"_id": "$direction", "total_volume": {"$sum": "$AAPL.Volume"}}},
    {"$sort": {"total_volume": -1}}
]

# Execute the pipeline and store results in a dataframe
result = pd.DataFrame(list(collection.aggregate(pipeline)))

# Print the results
print(result)

```

```
Empty DataFrame
Columns: []
Index: []
```

Data Analysis

```
# Calculate daily stock returns
df["Returns"] = df["AAPL.Close"].pct_change()
```

```
# Calculate daily stock volatility
df["Volatility"] = df["Returns"].rolling(30).std() * (252**0.5)
```

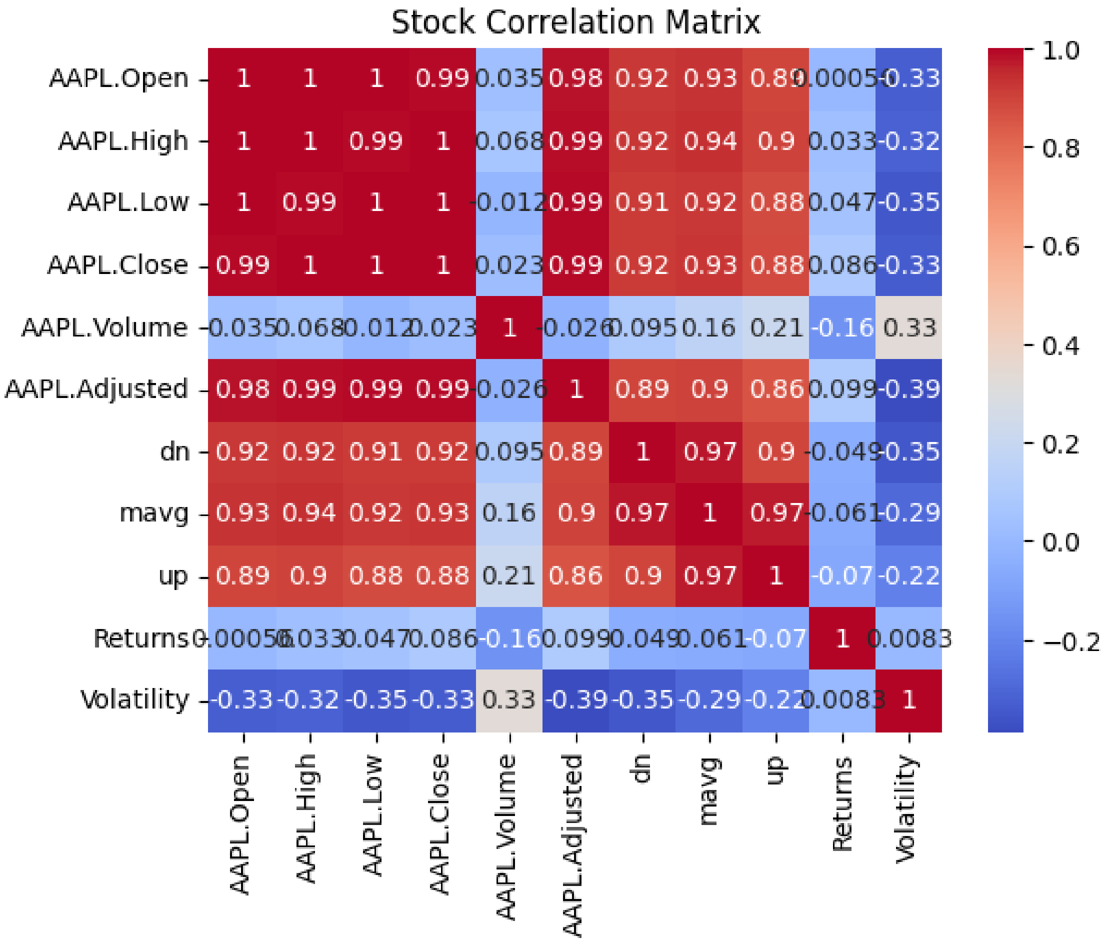
```
# Calculate correlation matrix
corr_matrix = df.corr()
```

```
<ipython-input-20-0e974cba71e3>:2: FutureWarning: The default value of numeric_o
  corr_matrix = df.corr()
```

Data Visualization

Visualize stock correlation matrix using heatmap

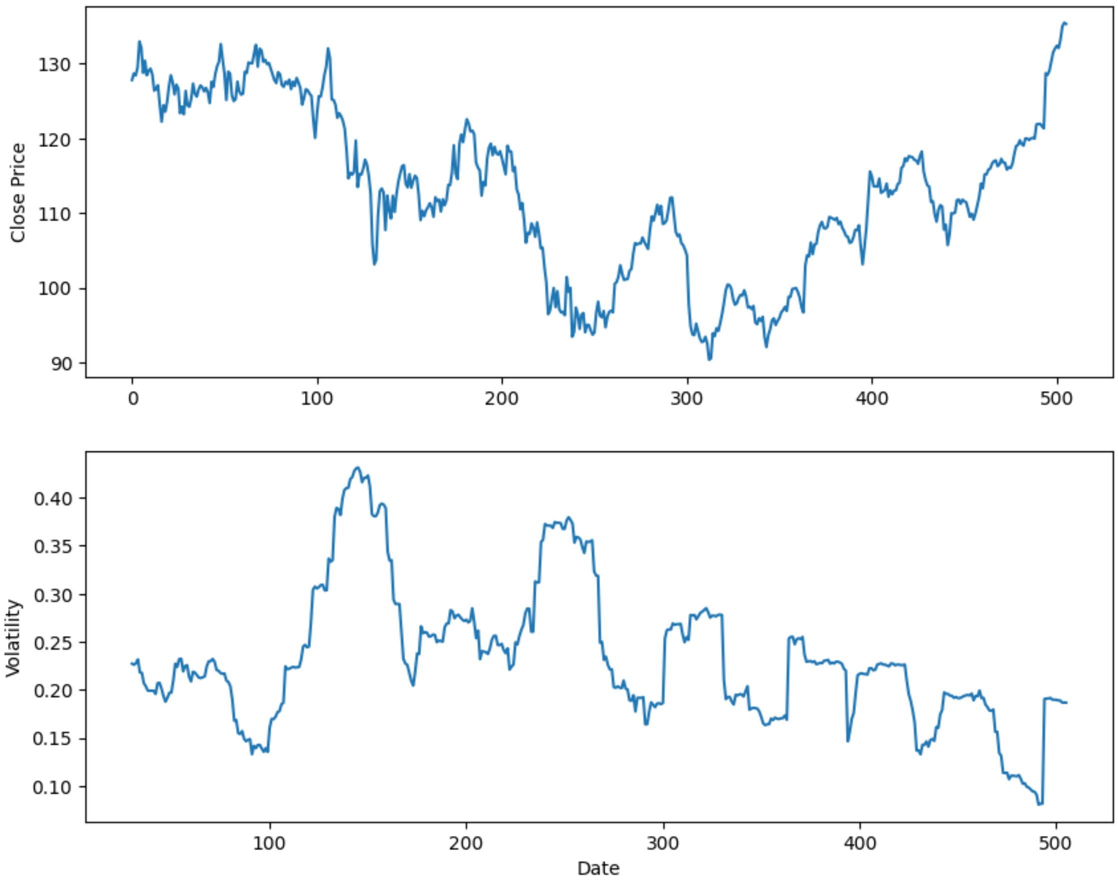
```
# Plot correlation matrix
import seaborn as sns
import matplotlib.pyplot as plt

sns.heatmap(corr_matrix, annot=True, cmap="coolwarm")
plt.title("Stock Correlation Matrix")
plt.show()
```



	AAPL.Open	AAPL.High	AAPL.Low	AAPL.Close	AAPL.Volume	AAPL.Adjusted	dn	mavg	up	Returns	Volatility
AAPL.Open	1	1	1	0.99	0.035	0.98	0.92	0.93	0.89	0.00056	-0.33
AAPL.High	1	1	0.99	1	0.068	0.99	0.92	0.94	0.9	0.033	-0.32
AAPL.Low	1	0.99	1	1	-0.012	0.99	0.91	0.92	0.88	0.047	-0.35
AAPL.Close	0.99	1	1	1	0.023	0.99	0.92	0.93	0.88	0.086	-0.33
AAPL.Volume	0.035	0.068	-0.012	0.023	1	-0.026	0.095	0.16	0.21	-0.16	0.33
AAPL.Adjusted	0.98	0.99	0.99	0.99	-0.026	1	0.89	0.9	0.86	0.099	-0.39
dn	0.92	0.92	0.91	0.92	0.095	0.89	1	0.97	0.9	-0.049	-0.35
mavg	0.93	0.94	0.92	0.93	0.16	0.9	0.97	1	0.97	-0.061	-0.29
up	0.89	0.9	0.88	0.88	0.21	0.86	0.9	0.97	1	-0.07	-0.22
Returns	0.00056	0.033	0.047	0.086	-0.16	0.099	-0.049	-0.061	-0.07	1	0.0083
Volatility	-0.33	-0.32	-0.35	-0.33	0.33	-0.39	-0.35	-0.29	-0.22	0.0083	1

Plot stock returns and volatility

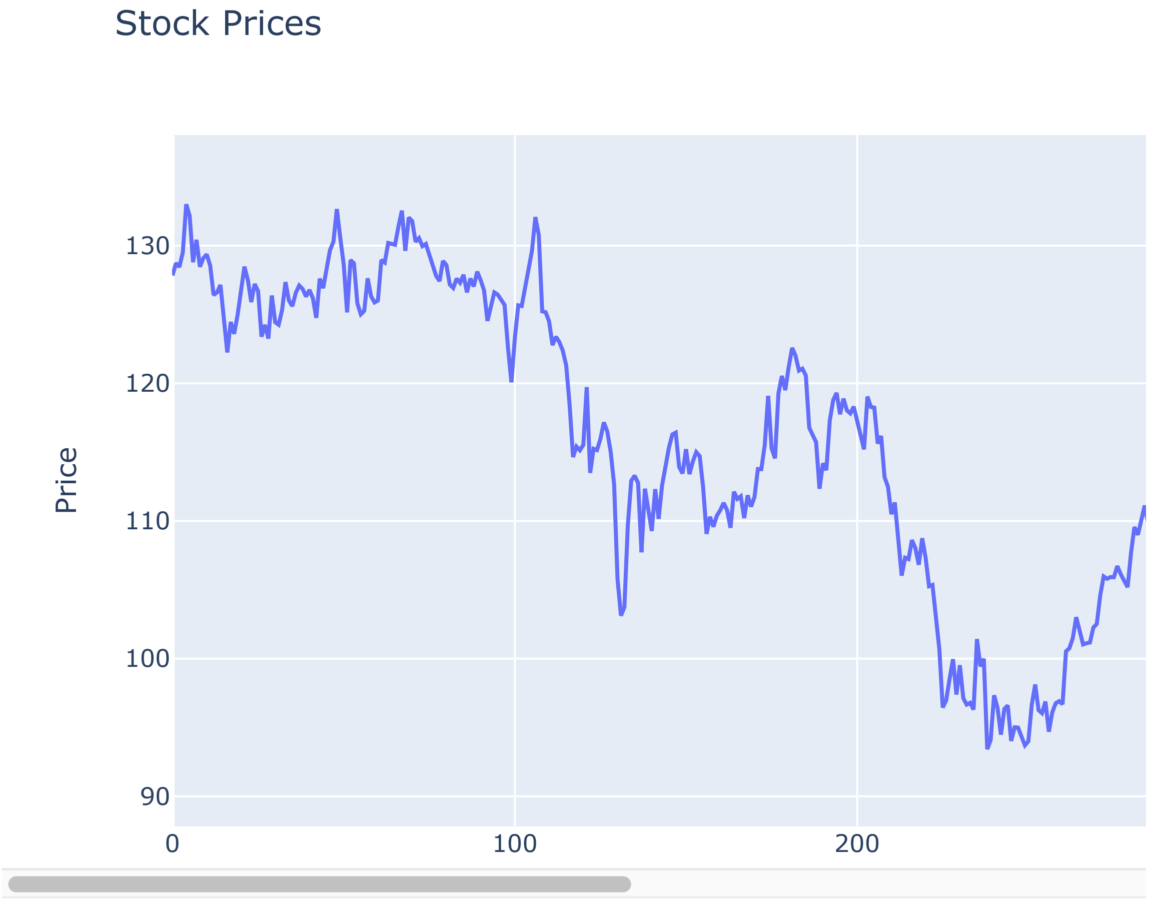
```
# Plot stock returns and volatility
fig, ax = plt.subplots(2,1, figsize=(10,8))
ax[0].plot(df.index, df["AAPL.Close"])
ax[0].set_ylabel("Close Price")

ax[1].plot(df.index, df["Volatility"])
ax[1].set_ylabel("Volatility")
ax[1].set_xlabel("Date")

plt.show()
```



Time series plot with hover information:

You can create an interactive time series plot of the stock prices, where hovering over a data point displays more information about that point, such as the date and the stock price.

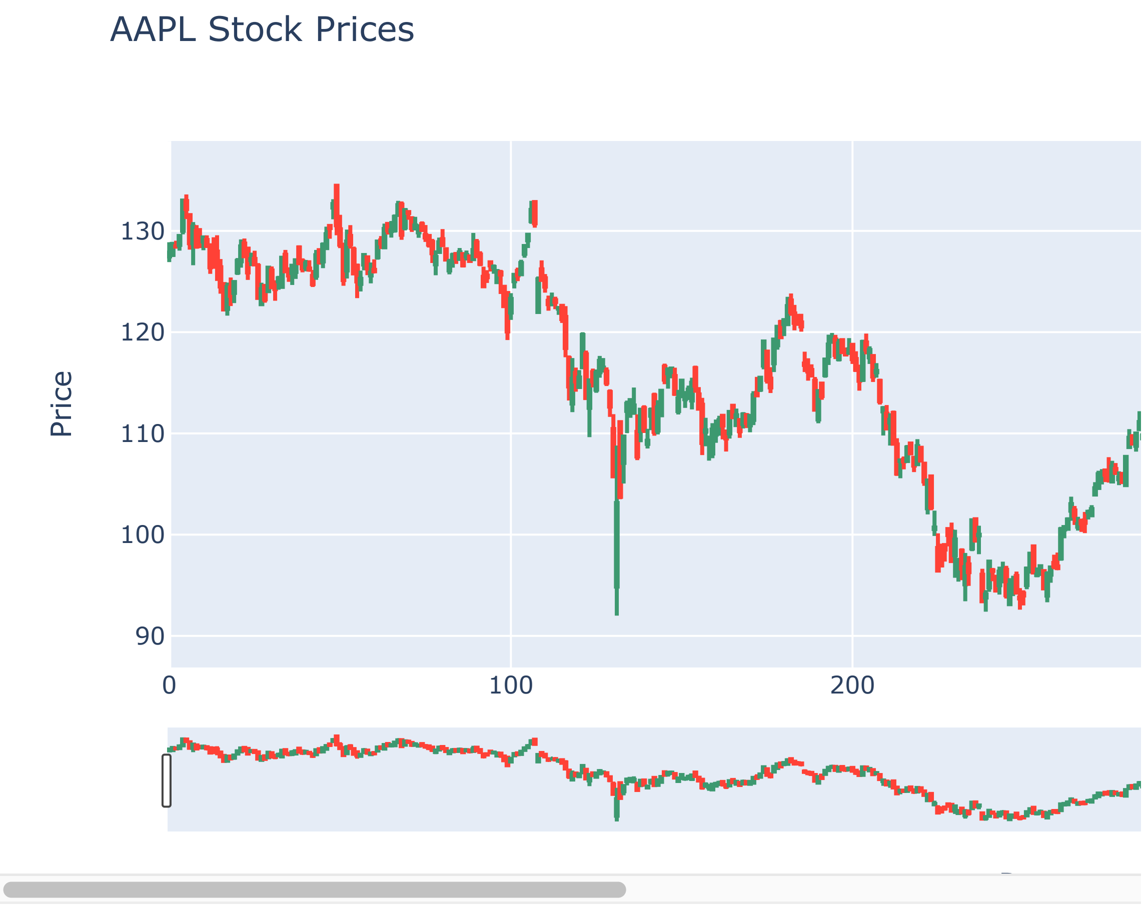
```
import plotly.graph_objs as go

# Create trace for time series plot
trace = go.Scatter(x=df.index, y=df["AAPL.Close"], mode="lines")

# Define layout for time series plot
layout = go.Layout(title="Stock Prices", xaxis=dict(title="Date"), yaxis=dict(title

# Create figure and add trace and layout to it
fig = go.Figure(data=[trace], layout=layout)

# Add hover information to time series plot
fig.update_traces(hovertemplate="Price: %{y:.2f}<br>Date: %{x|%Y-%m-%d}")

# Show time series plot
fig.show()

```



Candlestick chart:

You can create an interactive candlestick chart for the AAPL stock prices, where each data point represents the opening, high, low, and closing prices for a given time period. The chart can be zoomed and panned for a more detailed view of the data.

```
import plotly.graph_objs as go
# Create trace for candlestick chart
trace = go.Candlestick(x=df.index, open=df["AAPL.Open"], high=df["AAPL.High"], low=

# Define layout for candlestick chart
layout = go.Layout(title="AAPL Stock Prices", xaxis=dict(title="Date"), yaxis=dict(

# Create figure and add trace and layout to it
fig = go.Figure(data=[trace], layout=layout)

# Show candlestick chart
fig.show()

```



Line chart with moving averages:

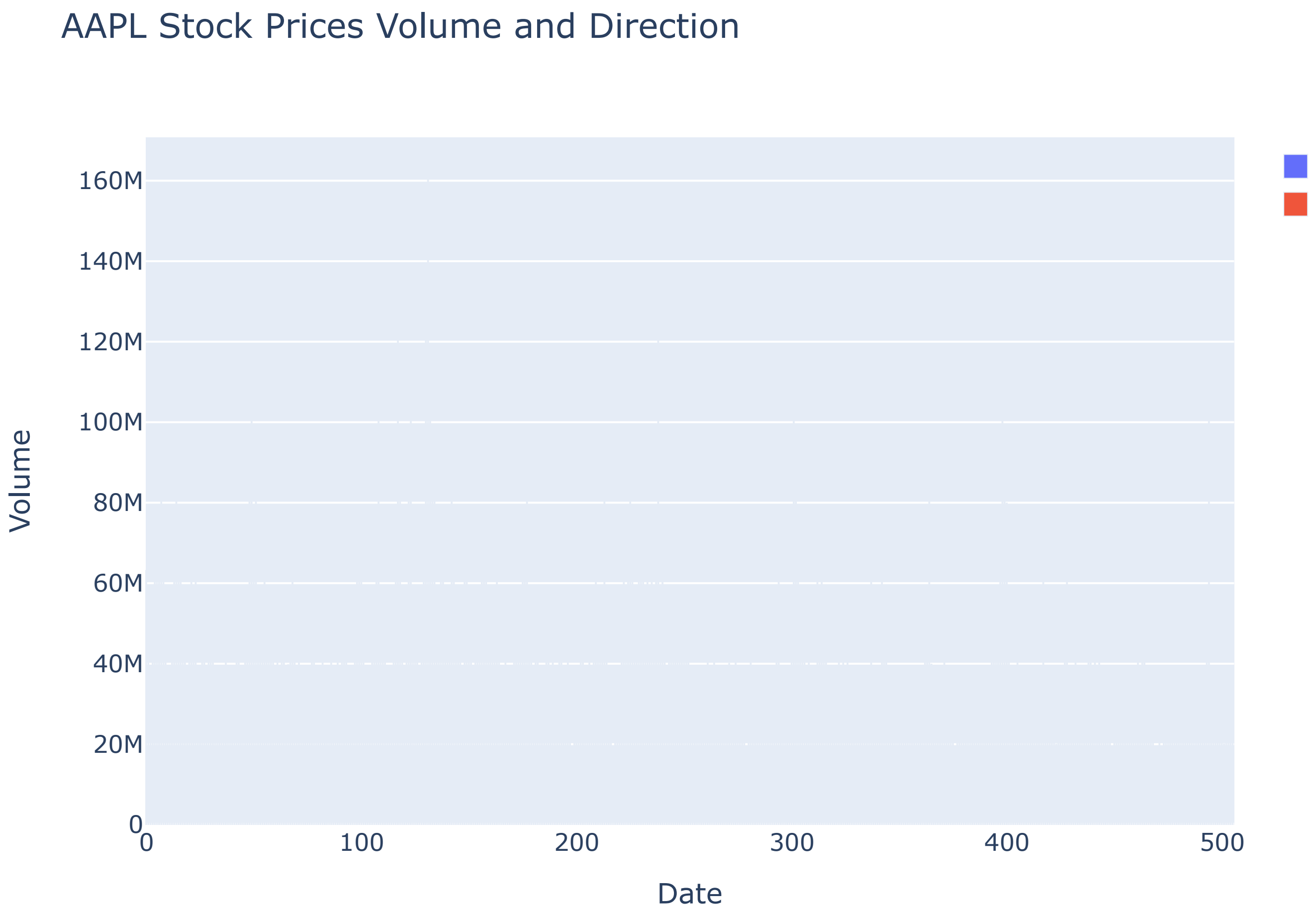
You can create an interactive line chart that shows the AAPL stock prices over time, along with the moving average values. The moving averages can be calculated using the mavg column in the dataset. The chart can be zoomed and panned for a more detailed view of the data.

```
# Create trace for stock prices
trace1 = go.Scatter(x=df.index, y=df["AAPL.Close"], mode="lines", name="AAPL")

# Create trace for moving average
trace2 = go.Scatter(x=df.index, y=df["mavg"], mode="lines", name="Moving Average")

# Define layout for line chart
layout = go.Layout(title="AAPL Stock Prices with Moving Average", xaxis=dict(title=

# Create figure and add traces and layout to it
fig = go.Figure(data=[trace1, trace2], layout=layout)

# Show line chart
fig.show()

```

Bar chart with volume and direction:

You can create an interactive bar chart that shows the volume and direction of the AAPL stock prices over time. The chart can be zoomed and panned for a more detailed view of the data.

```
# Create trace for volume bar chart
trace1 = go.Bar(x=df.index, y=df["AAPL.Volume"], name="Volume")

# Create trace for direction bar chart
trace2 = go.Bar(x=df.index, y=df["direction"], name="Direction")

# Define layout for bar chart
layout = go.Layout(title="AAPL Stock Prices Volume and Direction", xaxis=dict(title

# Create figure and add traces and layout to it
fig = go.Figure(data=[trace1, trace2], layout=layout)

# Show bar chart
fig.show()
```



+ Code + Text