Time series analysis means analyzing and finding patterns in a time series dataset. A time-series dataset is a sequence of data collected over an interval of time. Stock price data, monthly sales data, daily rainfall data, hourly website traffic data are some examples of time-series data that you will get to solve business problems as a data scientist

I will be using the plotly library in Python here as it is easy to analyze data with plotly because of less code and interactive results. I will recommend using a Jupyter Notebook or Google Colaboratory for Time series analysis instead of using a code editor or an IDE like VS Code or PyCharm.

In [1]:
```
# Importing required libraries
import pandas as pd
import plotly.express as px
import plotly.graph_objects as go
import yfinance as yf
from datetime import date, timedelta

# Getting today's date and formatting it
today=date.today()
end_date = today.strftime("%Y-%m-%d")

# Getting the date 720 days ago from today and formatting it
d1 = date.today() - timedelta(days=720)
start_date = d1.strftime("%Y-%m-%d")

# Fetching historical stock market data for AAPL (Apple Inc.)
data = yf.download('AAPL',
                   start=start_date,
                   end=end_date,
                   progress=False)
```

In [2]:
```
data.head(8)
```

Out[2]:

Date	Open	High	Low	Close	Adj Close	Volume
2022-04-04	174.570007	178.490005	174.440002	178.440002	176.423874	76468400
2022-04-05	177.500000	178.300003	174.419998	175.059998	173.082062	73401800
2022-04-06	172.360001	173.630005	170.130005	171.830002	169.888565	89058800
2022-04-07	171.160004	173.360001	169.850006	172.139999	170.195053	77594700
2022-04-08	171.779999	171.779999	169.199997	170.089996	168.168213	76575500
2022-04-11	168.710007	169.029999	165.500000	165.750000	163.877258	72246700
2022-04-12	168.020004	169.869995	166.639999	167.660004	165.765686	79265200
2022-04-13	167.389999	171.039993	166.770004	170.399994	168.474701	70618900

In [3]:
```
data.tail(8)
```

Out[3]:

Date	Open	High	Low	Close	Adj Close	Volume
2024-03-13	172.770004	173.190002	170.759995	171.130005	171.130005	52488700
2024-03-14	172.910004	174.309998	172.050003	173.000000	173.000000	72913500
2024-03-15	171.169998	172.619995	170.289993	172.619995	172.619995	121664700
2024-03-18	175.570007	177.710007	173.520004	173.720001	173.720001	75604200
2024-03-19	174.339996	176.610001	173.029999	176.080002	176.080002	55215200
2024-03-20	175.720001	178.669998	175.089996	178.669998	178.669998	53423100
2024-03-21	177.050003	177.490005	170.839996	171.369995	171.369995	106181300
2024-03-22	171.860001	173.050003	170.059998	172.279999	172.279999	70618124

In [4]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 495 entries, 2022-04-04 to 2024-03-22
Data columns (total 6 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Open       495 non-null    float64
 1   High       495 non-null    float64
 2   Low        495 non-null    float64
 3   Close      495 non-null    float64
 4   Adj Close  495 non-null    float64
 5   Volume     495 non-null    int64  
dtypes: float64(5), int64(1)
memory usage: 27.1 KB
```

The reason we only have 494 values instead of the expected 720 days worth of data could be due to several factors:

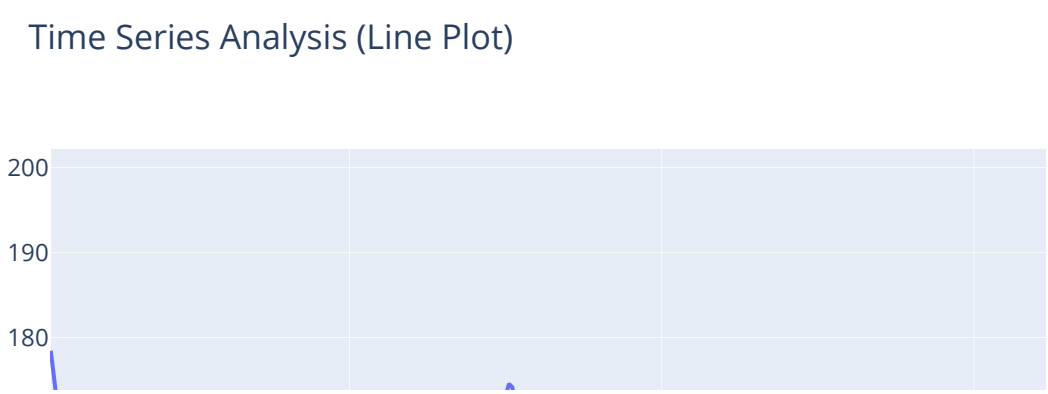
- Trading Days vs. Calendar Days: Stock markets are not open on weekends and holidays. Therefore, if you're looking for 720 calendar days, the number of trading days within that period might be less.
- Missing Data: Sometimes, there are missing data points for various reasons such as holidays, data errors, or the unavailability of trading data.

In [5]:
```
data.describe()
```

Out[5]:

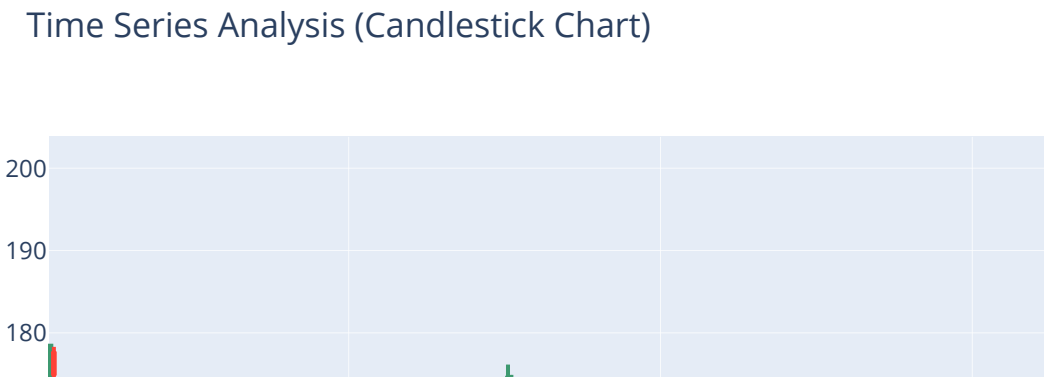
	Open	High	Low	Close	Adj Close	Volume
count	495.000000	495.000000	495.000000	495.000000	495.000000	4.950000e+02
mean	165.119213	166.881899	163.492808	165.266061	164.413930	6.953084e+07
std	18.827272	18.503530	19.051000	18.784394	19.079842	2.408641e+07
min	126.010002	127.769997	124.169998	125.019997	124.166634	2.404830e+07
25%	148.720001	150.760002	147.139999	149.075005	147.929649	5.234125e+07
50%	167.320007	168.880005	165.649994	167.229996	165.765198	6.508660e+07
75%	180.949997	182.384995	179.004997	180.849998	180.245224	8.030145e+07
max	198.020004	199.619995	197.000000	198.110001	197.857529	1.826020e+08

In [6]:
```
# Line Plot: Visualizing closing prices over time
figure = px.line(data, x = data.index,
                 y = "Close",
                 title = "Time Series Analysis (Line Plot)")
figure.show()
```

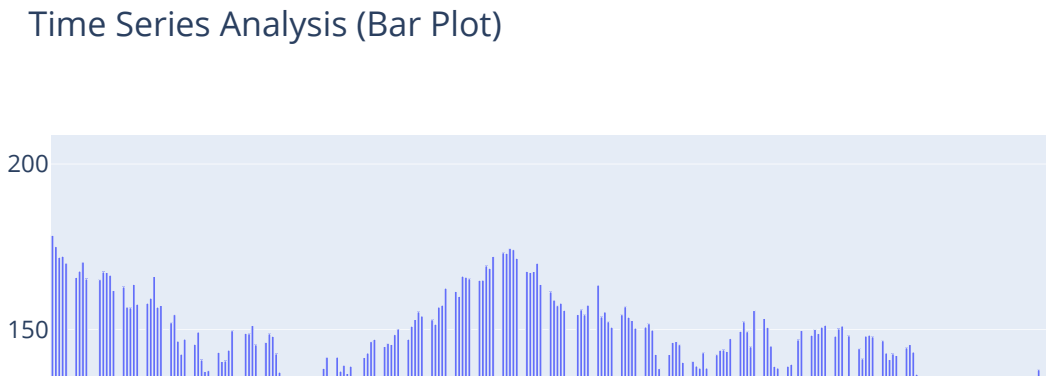


In [7]:
```
# Candlestick Chart: Visualizing open, high, low, and close prices over time
figure = go.Figure(data=[go.Candlestick(x = data.index,
                                        open = data["Open"],
                                        high = data["High"],
                                        low = data["Low"],
                                        close = data["Close"])])
figure.update_layout(title = "Time Series Analysis (Candlestick Chart)",
```

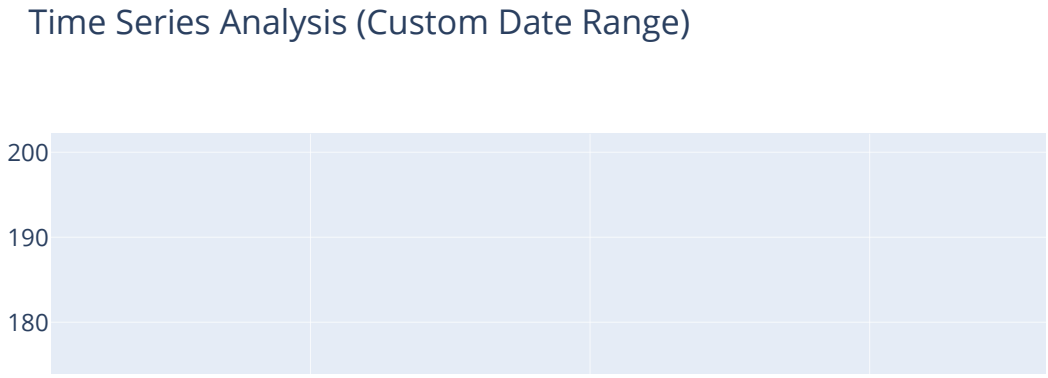
```
                    xaxis_rangeslider_visible = False)
figure.show()
```



In [8]:
```
# Bar Plot: Visualizing closing prices over time using bars
figure = px.bar(data, x = data.index,
                y = "Close",
                title = "Time Series Analysis (Bar Plot)" )
figure.show()
```



In [9]:
```
# Custom Date Range Line Plot: Visualizing closing prices for a specific date range
figure = px.line(data, x = data.index,
                 y = 'Close',
                 range_x = ['2022-04-11','2022-08-11'], # Input your desired date r
                 title = "Time Series Analysis (Custom Date Range)")
figure.show()
```



In [10]:
```
# Interactive Candlestick Chart with Buttons and Slider: Visualizing open, high, lo
figure = go.Figure(data = [go.Candlestick(x = data.index,
                                          open = data["Open"],
                                          high = data["High"],
                                          low = data["Low"],
                                          close = data["Close"])])
figure.update_layout(title = "Time Series Analysis (Candlestick Chart with Buttons

figure.update_xaxes(
    rangeslider_visible = True,
    rangeselector = dict(
        buttons = list([
            dict(count = 1, label = "1m", step = "month", stepmode = "backward"),
            dict(count = 6, label = "6m", step = "month", stepmode = "backward"),
            dict(count = 1, label = "YTD", step = "year", stepmode = "todate"),
            dict(count = 1, label = "1y", step = "year", stepmode = "backward"),
            dict(step = "all")
        ])
    )
)
figure.show()
```


Time Series Analysis (Candlestick Chart with Buttons and Slider)
1m
6m
YTD
1y
all
200
180