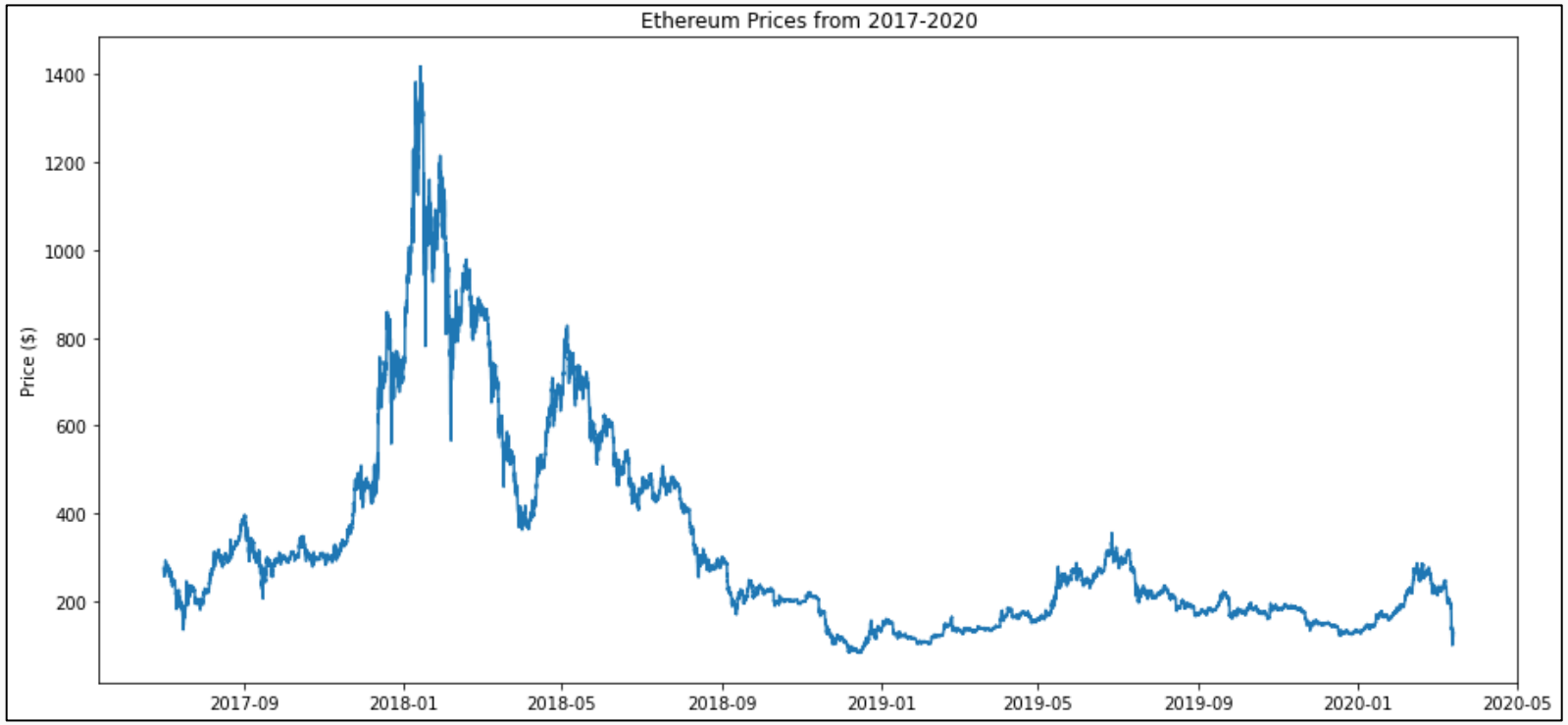
TIME SERIES

What is Time Series

- A time series is a sequence of data measured over regular time intervals.
- By analyzing a time series, we can identify the pattern in the data and develop a forecasting model.
- Python provides a lot of functionality to analyze a time series.



Meta Brains

Python Libraries for Time Series Analysis

- Python has a lot of libraries for time series analysis.
- Some of these libraries are used in Machine Learning and hence are out of the scope of this course.
- We will be using the following libraries for time series analysis;
  - Pandas
  - NumPy
  - datetime
  - Matplotlib

How to Get Time Series?

- As mentioned earlier, a time series is just a set of values measured over regular intervals of time.
- Generally, any dataset with values measured over time can be treated as a time series.
- Examples of such dataset could include but is not limited to;
  - Stocks data
  - Weather data
  - Patient health evolution data
- In Python, the data type of dates and times in a time series is either datetime or Timestamp.
- In this course, we will see how can we download a time series and how can we convert a dataset with values measured over time into a time series.

Getting Stocks Data using yfinance (1/5)

- yfinance is a Python library that allows downloading stocks data from Yahoo Finance for a specified period of time as a time series.
- You can also download a dataset from their website directly https://finance.yahoo.com/

Meta Brains

Getting Stocks Data using yfinance (2/5)

- To use yfinance Python library, we first have to install it via the following command;
  pip install yfinance
- Once the library has been installed, we can import it into the Jupyter Notebook.
- yfinance is commonly imported as yf.

```
pip install yfinance
```

```
Collecting yfinanceNote: you may need to restart the kernel to use updated packages.

  Downloading yfinance-0.1.70-py2.py3-none-any.whl (26 kB)
Requirement already satisfied: lxml>=4.5.1 in c:\users\22100199\anaconda3\lib\site-packages (from yfinance) (4.6.3)
Requirement already satisfied: numpy>=1.15 in c:\users\22100199\anaconda3\lib\site-packages (from yfinance) (1.20.3)
Requirement already satisfied: requests>=2.26 in c:\users\22100199\anaconda3\lib\site-packages (from yfinance) (2.26.0)
Collecting multitasking>=0.0.7
  Downloading multitasking-0.0.10.tar.gz (8.2 kB)
Requirement already satisfied: pandas>=0.24.0 in c:\users\22100199\anaconda3\lib\site-packages (from yfinance) (1.3.4)
Requirement already satisfied: python-dateutil>=2.7.3 in c:\users\22100199\anaconda3\lib\site-packages (from pandas>=0.24.0->yf
inance) (2.8.2)
Requirement already satisfied: pytz>=2017.3 in c:\users\22100199\anaconda3\lib\site-packages (from pandas>=0.24.0->yfinance) (2
021.3)
Requirement already satisfied: six>=1.5 in c:\users\22100199\anaconda3\lib\site-packages (from python-dateutil>=2.7.3->pandas>=
0.24.0->yfinance) (1.16.0)
Requirement already satisfied: idna<4,>=2.5 in c:\users\22100199\anaconda3\lib\site-packages (from requests>=2.26->yfinance)
(3.2)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\22100199\anaconda3\lib\site-packages (from requests>=2.26->yfinan
ce) (2021.10.8)
Requirement already satisfied: charset-normalizer~=2.0.0 in c:\users\22100199\anaconda3\lib\site-packages (from requests>=2.26-
>yfinance) (2.0.4)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in c:\users\22100199\anaconda3\lib\site-packages (from requests>=2.26->yfi
nance) (1.26.7)
Building wheels for collected packages: multitasking
  Building wheel for multitasking (setup.py): started
  Building wheel for multitasking (setup.py): finished with status 'done'
  Created wheel for multitasking: filename=multitasking-0.0.10-py3-none-any.whl size=8500 sha256=7d5ad1da18be483c99d9f99c43e6f4
58099dd09b5ec65cf632200300198faee6
  Stored in directory: c:\users\22100199\appdata\local\pip\cache\wheels\f2\b5\2c\59ba95dcf854e542944c75fe3da584e4e3833b319735a0
546c
Successfully built multitasking
Installing collected packages: multitasking, yfinance
Successfully installed multitasking-0.0.10 yfinance-0.1.70
```

```
import yfinance as yf
```

Meta Brains

Getting Stocks Data using yfinance (3/5)

- After importing the library, we can download stock data for different companies using the .download() method.
- We pass a start and an end date in the format 'YYYY-MM-DD'.
- If there is no data available for a certain date, that date would be skipped.

Getting Stocks Data using yfinance (4/5)

- In the given example, we download the stocks data for Apple (abbreviated as AAPL in Yahoo Finance) from 1st Jan, 2022 to 31st Jan, 2022.

```
df = yf.download('AAPL', start='2022-01-01', end='2022-01-31')
df.head(10)
```

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

Date	Open	High	Low	Close	Adj Close	Volume
2021-12-31	178.089996	179.229996	177.259995	177.570007	177.344055	64062300
2022-01-03	177.830002	182.880005	177.710007	182.009995	181.778397	104487900
2022-01-04	182.630005	182.940002	179.119995	179.699997	179.471344	99310400
2022-01-05	179.610001	180.169998	174.639999	174.919998	174.697418	94537600
2022-01-06	172.699997	175.300003	171.639999	172.000000	171.781143	96904000
2022-01-07	172.889999	174.139999	171.029999	172.169998	171.950928	86580100
2022-01-10	169.080002	172.500000	168.169998	172.190002	171.970901	106765600
2022-01-11	172.320007	175.179993	170.820007	175.080002	174.857224	76138300
2022-01-12	176.119995	177.179993	174.820007	175.529999	175.306641	74805200
2022-01-13	175.779999	176.619995	171.789993	172.190002	171.970901	84505800

Meta Brains

Getting Stocks Data using yfinance (5/5)

- As you can see, the values of the index column (Date) of the dataset in the previous slide have the type Timestamp, indicating that it is a Python time series.

```
type(df.index[0])
```

```
pandas._libs.tslibs.timestamps.Timestamp
```

Meta Brains

Quiz Time

1. Stocks data over a period of regular time intervals is a time series?
    - True
    - False

Meta Brains

Quiz Time

1. Stocks data over a period of regular time intervals is a time series?
    - True
    - False

Meta Brains

Converting a Dataset To Time Series (1/6)

- Time Series data is generally stored as csv file having a column representing date or time or both.
- It can be imported in Python using the .read_csv() function of Pandas library.
- We can use the to_datetime() function of the Pandas library to convert the values of the date column from string to Timestamp.

Converting a Dataset To Time Series (2/6)

- Consider the following time series. The type of each value of ‘date’ (index) is a string as shown in the figure on the right.

```
df
```

date	value
1991-07-01	3.526591
1991-08-01	3.180891
1991-09-01	3.252221
1991-10-01	3.611003
1991-11-01	3.565869
...	...
2008-02-01	21.654285
2008-03-01	18.264945
2008-04-01	23.107677
2008-05-01	22.912510
2008-06-01	19.431740

204 rows × 1 columns

```
type(df.index[0])
```

```
str
```

Meta Brains

Converting a Dataset To Time Series (3/6)

- To be able to use the time series functionality available in Python, we have to convert the values of ‘date’ to Timestamp.
- This can be done using the to_datetime() function of Pandas.

```
df.index = pd.to_datetime(df.index)
df
```

date	value
1991-07-01	3.526591
1991-08-01	3.180891
1991-09-01	3.252221
1991-10-01	3.611003
1991-11-01	3.565869
...	...
2008-02-01	21.654285
2008-03-01	18.264945
2008-04-01	23.107677
2008-05-01	22.912510
2008-06-01	19.431740

204 rows × 1 columns

```
type(df.index[0])
```

```
pandas._libs.tslibs.timestamps.Timestamp
```

Meta Brains

Converting a Dataset To Time Series (4/6)

- In the previous example, the data in the ‘date’ column was in a format that Pandas could easily convert. However, this is not always the case.
- Consider the following dataframe for example. ‘Date’ values are strings.

Date	Symbol	Open	High	Low	Close	Volume
2020-03-13 08-PM	ETHUSD	129.94	131.82	126.87	128.71	1940673.93
2020-03-13 07-PM	ETHUSD	119.51	132.02	117.10	129.94	7579741.09
2020-03-13 06-PM	ETHUSD	124.47	124.85	115.50	119.51	4898735.81
2020-03-13 05-PM	ETHUSD	124.08	127.42	121.63	124.47	2753450.92
2020-03-13 04-PM	ETHUSD	124.85	129.51	120.17	124.08	4461424.71
...	...	...	...	...	...	...
2017-07-01 03-PM	ETHUSD	265.74	272.74	265.00	272.57	1500282.55
2017-07-01 02-PM	ETHUSD	268.79	269.90	265.00	265.74	1702536.85
2017-07-01 01-PM	ETHUSD	274.83	274.93	265.00	268.79	3010787.99
2017-07-01 12-PM	ETHUSD	275.01	275.01	271.00	274.83	824362.87
2017-07-01 11-AM	ETHUSD	279.98	279.99	272.10	275.01	679358.87

23674 rows × 6 columns

Meta Brains

Converting a Dataset To Time Series (5/6)

- Converting the values of 'Date' (index) results in a ParserError.

```
df.index = pd.to_datetime(df.index)
df
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
File ~\Anaconda3\lib\site-packages\pandas\core\arrays\datetimes.py:2187, in objects_to_datetime64ns(data, dayfirst, yearfirst,
utc, errors, require_iso8601, allow_object, allow_mixed)
   2186 try:
-> 2187     values, tz_parsed = conversion.datetime_to_datetime64(data.ravel("K"))
   2188     # If tzaware, these values represent unix timestamps, so we
   2189     #  return them as i8 to distinguish from wall times

File ~\Anaconda3\lib\site-packages\pandas\_libs\tslibs\conversion.pyx:359, in pandas._libs.tslibs.conversion.datetime_to_dateti
me64()

TypeError: Unrecognized value type: <class 'str'>

During handling of the above exception, another exception occurred:

ParserError                               Traceback (most recent call last)
Input In [32], in <module>
----> 1 df.index = pd.to_datetime(df.index)
      2 df
```

Meta Brains

Converting a Dataset To Time Series (6/6)

- The error arises because Pandas does not know the format of the string values.
- We can use the ‘format’ parameter to resolve this issue.
- You can find a list of all format codes at: https://docs.python.org/3/library/datetime.html#strftime-and-strptime-behavior

```
df.index = pd.to_datetime(df.index, format='%Y-%m-%d %I-%p')
print(type(df.index[0]))
df
```

```
<class 'pandas._libs.tslibs.timestamps.Timestamp'>
```

Date	Symbol	Open	High	Low	Close	Volume
2020-03-13 20:00:00	ETHUSD	129.94	131.82	126.87	128.71	1940673.93
2020-03-13 19:00:00	ETHUSD	119.51	132.02	117.10	129.94	7579741.09
2020-03-13 18:00:00	ETHUSD	124.47	124.85	115.50	119.51	4898735.81
2020-03-13 17:00:00	ETHUSD	124.08	127.42	121.63	124.47	2753450.92
2020-03-13 16:00:00	ETHUSD	124.85	129.51	120.17	124.08	4461424.71
...	...	...	...	...	...	...
2017-07-01 15:00:00	ETHUSD	265.74	272.74	265.00	272.57	1500282.55
2017-07-01 14:00:00	ETHUSD	268.79	269.90	265.00	265.74	1702536.85
2017-07-01 13:00:00	ETHUSD	274.83	274.93	265.00	268.79	3010787.99
2017-07-01 12:00:00	ETHUSD	275.01	275.01	271.00	274.83	824362.87
2017-07-01 11:00:00	ETHUSD	279.98	279.99	272.10	275.01	679358.87

23674 rows × 6 columns

Meta Brains

Working With Time Series (1/7)

- In this section we will learn to use Pandas time series functionality.
- We will use the following time series containing stock data of Apple from 1st of Jan, 2022 to 4th Feb, 2022.

```
df = yf.download('AAPL', start='2022-01-01', end='2022-02-04')
df.head(10)
```

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

Date	Open	High	Low	Close	Adj Close	Volume
2021-12-31	178.089996	179.229996	177.259995	177.570007	177.344055	64062300
2022-01-03	177.830002	182.880005	177.710007	182.009995	181.778397	104487900
2022-01-04	182.630005	182.940002	179.119995	179.699997	179.471344	99310400
2022-01-05	179.610001	180.169998	174.639999	174.919998	174.697418	94537600
2022-01-06	172.699997	175.300003	171.639999	172.000000	171.781143	96904000
2022-01-07	172.889999	174.139999	171.029999	172.169998	171.950928	86580100
2022-01-10	169.080002	172.500000	168.169998	172.190002	171.970901	106765600
2022-01-11	172.320007	175.179993	170.820007	175.080002	174.857224	76138300
2022-01-12	176.119995	177.179993	174.820007	175.529999	175.306641	74805200
2022-01-13	175.779999	176.619995	171.789993	172.190002	171.970901	84505800

Meta Brains

Working With Time Series (2/7)

.day_name()

- The first thing you might have noticed is that there is no data available for 1st and 2nd Jan, 2022.
- Let's see whether these two were business days or not.
- We will use the .day_name() function to find out what day was on 1st Jan.
- Remember that this function can only be applied on a value of time datetime or Timestamp.
- 1st Jan, 2022 was a Saturday and hence 2nd Jan, 2022 was a Sunday. No business!

```
jan1 = pd.to_datetime('2022, 1, 1')
jan1.day_name()
```

```
'Saturday'
```

Meta Brains

Working With Time Series (3/7)

Max/Min Date

- Sometimes, our timeseries data might not be in order.
- In such a case, we would be interested in what is the maximum and the minimum datetime (or Timestamp) in the dataset.
- This can be done using the .max() and .min() functions.

```
df.index.max()
```

```
Timestamp('2022-02-03 00:00:00')
```

```
df.index.min()
```

```
Timestamp('2021-12-31 00:00:00')
```

Meta Brains

Working With Time Series (4/7)

datetime Range

- Next, let's see what is the datetime range in our dataset.
- This is as simple as subtracting minimum value of datetime (Timestamp) from its maximum value.
- Note that this will only work if you have datetime or Timestamp values in your dataset and not strings.

```
timePeriod = df.index.max() - df.index.min()
timePeriod
```

```
Timedelta('34 days 00:00:00')
```

Meta Brains

Working With Time Series (5/7)

Time-based Indexing

- Pandas provides time-based indexing for time series, i.e., accessing data using datetime.
- We can select data on specific dates and times using .loc.
- In the given example, we get the data on 7th Jan, 2022.

```
df.loc['2022-01-07']
```

```
Open          1.728900e+02
High          1.741400e+02
Low           1.710300e+02
Close         1.721700e+02
Adj Close     1.719509e+02
Volume        8.658010e+07
Name: 2022-01-07 00:00:00, dtype: float64
```

Meta Brains

Working With Time Series (6/7)

Time-based Indexing

- Let's select multiple dates using slicing with .loc.

```
df.loc['2022-01-07':'2022-01-14']
```

Date	Open	High	Low	Close	Adj Close	Volume
2022-01-07	172.889999	174.139999	171.029999	172.169998	171.950928	86580100
2022-01-10	169.080002	172.500000	168.169998	172.190002	171.970901	106765600
2022-01-11	172.320007	175.179993	170.820007	175.080002	174.857224	76138300
2022-01-12	176.119995	177.179993	174.820007	175.529999	175.306641	74805200
2022-01-13	175.779999	176.619995	171.789993	172.190002	171.970901	84505800
2022-01-14	171.339996	173.779999	171.089996	173.070007	172.849792	80355000

Meta Brains

Working With Time Series (7/7)

Time-based Indexing

- We can also partially match a datetime using .loc.
- In the given example, we get the data from the time series for the month of February only.
- We pass the string ‘2022-02’ to get the data of February.

```
df.loc['2022-02']
```

Date	Open	High	Low	Close	Adj Close	Volume
2022-02-01	174.009995	174.839996	172.309998	174.610001	174.387817	86213900
2022-02-02	174.750000	175.880005	173.330002	175.839996	175.616257	84914300
2022-02-03	174.479996	176.240005	172.119995	172.899994	172.679993	89418100

Meta Brains

Quiz Time

1. Consider the given dataframe called df. What value will the following command return?
   df.loc['2008-02-01']
   a) 29.665356
   b) 21.654285
   c) 18.264945
   d) Error

date	value
2008-01-01	29.665356
2008-02-01	21.654285
2008-03-01	18.264945
2008-04-01	23.107677
2008-05-01	22.912510
2008-06-01	19.431740

Meta Brains

Quiz Time

1. Consider the given dataframe called df. What value will the following command return?
   df.loc['2008-02-01']
   a) 29.665356
   b) 21.654285
   c) 18.264945
   d) Error

date	value
2008-01-01	29.665356
2008-02-01	21.654285
2008-03-01	18.264945
2008-04-01	23.107677
2008-05-01	22.912510
2008-06-01	19.431740

Meta Brains

Visualizing a Time Series (1/3)

- Using Matplotlib and seaborn, we can visualize a time series.
- Let's see how the stock prices of Apple varied over the last 5 years (2017-2021).
- We will begin by downloading the stock data from 2017 to 2021.

```
df = yf.download('AAPL', start='2017-01-01', end='2021-12-31')
df
```

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

Date	Open	High	Low	Close	Adj Close	Volume
2017-01-03	28.950001	29.082500	28.690001	29.037500	27.297691	115127600
2017-01-04	28.962500	29.127501	28.937500	29.004999	27.267141	84472400
2017-01-05	28.980000	29.215000	28.952499	29.152500	27.405800	88774400
2017-01-06	29.195000	29.540001	29.117500	29.477501	27.711329	127007600
2017-01-09	29.487499	29.857500	29.485001	29.747499	27.965153	134247600
...	...	...	...	...	...	...
2021-12-23	175.850006	176.850006	175.270004	176.279999	176.055695	68356600
2021-12-27	177.089996	180.419998	177.070007	180.330002	180.100540	74919600
2021-12-28	180.160004	181.330002	178.529999	179.289993	179.061859	79144300
2021-12-29	179.330002	180.630005	178.139999	179.380005	179.151749	62348900
2021-12-30	179.470001	180.570007	178.089996	178.199997	177.973251	59773000

1258 rows × 6 columns

Meta Brains

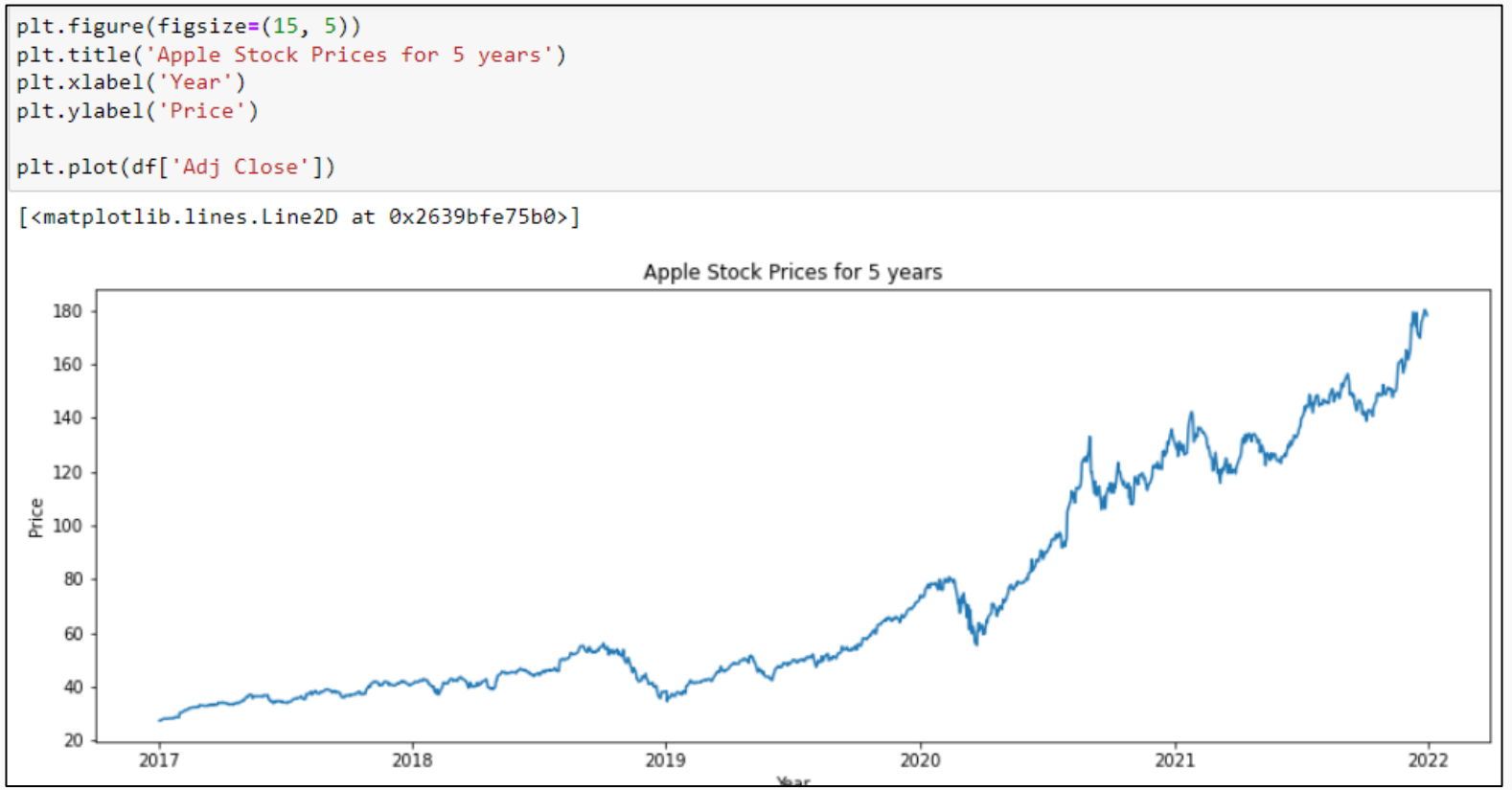
Visualizing a Time Series (2/3)

- Next, let's use the .plot() function of the matplotlib.pyplot to plot the 'Adj Close' time series.
- We can see that the stock prices of Apple have increased drastically since 2017.

```
plt.figure(figsize=(15, 5))
plt.title('Apple Stock Prices for 5 years')
plt.xlabel('Year')
plt.ylabel('Price')

plt.plot(df['Adj Close'])
```

```
[<matplotlib.lines.Line2D at 0x2639bfe75b0>]
```



Meta Brains

Visualizing a Time Series (3/3)

- We can also see how the Apple stock prices have varied over 2021 alone.
- We use the time-based indexing to get the year 2021 from the dataset.
- We then plot the ‘Adj Close’ time series.

```
plt.figure(figsize=(15, 5))
plt.title('Apple Stock Prices for 5 years')
plt.xlabel('Year')
plt.ylabel('Price')

plt.plot(df.loc['2021']['Adj Close'])
```

```
[<matplotlib.lines.Line2D at 0x2639c3010d0>]
```

Meta Brains

Thank You!

Meta Brains