IBM Developer
SKILLS NETWORK

Application Programming Interface

Estimated time needed: **15** minutes

Objectives

After completing this lab you will be able to:

- Create and Use APIs in Python

Introduction

An API lets two pieces of software talk to each other. Just like a function, you don’t have to know how the API works only its inputs and outputs. An essential type of API is a REST API that allows you to access resources via the internet. In this lab, we will review the Pandas Library in the context of an API, we will also review a basic REST API

Table of Contents

In [1]:
```
!pip install pycoingecko
!pip install plotly
!pip install mplfinance
```

```
Collecting pycoingecko
  Downloading pycoingecko-2.0.0-py3-none-any.whl (7.7 kB)
Requirement already satisfied: requests in c:\programdata\anaconda3\lib\site-packages (from pycoingecko) (2.24.0)
Requirement already satisfied: idna<3,>=2.5 in c:\programdata\anaconda3\lib\site-packages (from requests->pycoingecko) (2.10)
Requirement already satisfied: certifi>=2017.4.17 in c:\programdata\anaconda3\lib\site-packages (from requests->pycoingecko) (2020.6.20)
Requirement already satisfied: chardet<4,>=3.0.2 in c:\programdata\anaconda3\lib\site-packages (from requests->pycoingecko) (3.0.4)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in c:\programdata\anaconda3\lib\site-packages (from requests->pycoingecko) (1.25.11)
Installing collected packages: pycoingecko
Successfully installed pycoingecko-2.0.0
Collecting plotly
  Downloading plotly-4.14.3-py2.py3-none-any.whl (13.2 MB)
Collecting retrying>=1.3.3
  Downloading retrying-1.3.3.tar.gz (10 kB)
Requirement already satisfied: six in c:\programdata\anaconda3\lib\site-packages (from plotly) (1.15.0)
Building wheels for collected packages: retrying
  Building wheel for retrying (setup.py): started
  Building wheel for retrying (setup.py): finished with status 'done'
  Created wheel for retrying: filename=retrying-1.3.3-py3-none-any.whl size=11434 sha256=95b4a7573e1f18d113cf61a3efa6bfe56e01c7f9bec55633824785d8738aa4bc
  Stored in directory: c:\users\user\appdata\local\pip\cache\wheels\c4\a7\48\0a434133f6d56e878ca511c0e6c38326907c0792f67b476e56
Successfully built retrying
Installing collected packages: retrying, plotly
Successfully installed plotly-4.14.3 retrying-1.3.3
Collecting mplfinance
  Downloading mplfinance-0.12.7a17-py3-none-any.whl (62 kB)
Requirement already satisfied: pandas in c:\programdata\anaconda3\lib\site-packages (from mplfinance) (1.1.3)
Requirement already satisfied: matplotlib in c:\programdata\anaconda3\lib\site-packages (from mplfinance) (3.3.2)
Requirement already satisfied: numpy>=1.15.4 in c:\programdata\anaconda3\lib\site-packages (from pandas->mplfinance) (1.19.2)
Requirement already satisfied: python-dateutil>=2.7.3 in c:\programdata\anaconda3\lib\site-packages (from pandas->mplfinance) (2.8.1)
Requirement already satisfied: pytz>=2017.2 in c:\programdata\anaconda3\lib\site-packages (from pandas->mplfinance) (2020.1)
Requirement already satisfied: pillow>=6.2.0 in c:\programdata\anaconda3\lib\site-packages (from matplotlib->mplfinance) (8.0.1)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\programdata\anaconda3\lib\site-packages (from matplotlib->mplfinance) (1.3.0)
Requirement already satisfied: certifi>=2020.06.20 in c:\programdata\anaconda3\lib\site-packages (from matplotlib->mplfinance) (2020.6.20)
Requirement already satisfied: cycler>=0.10 in c:\programdata\anaconda3\lib\site-packages (from matplotlib->mplfinance) (0.10.0)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.3 in c:\programdata\anaconda3\lib\site-packages (from matplotlib->mplfinance) (2.4.7)
Requirement already satisfied: six>=1.5 in c:\programdata\anaconda3\lib\site-packages (from python-dateutil>=2.7.3->pandas->mplfinance) (1.15.0)
Installing collected packages: mplfinance
Successfully installed mplfinance-0.12.7a17
```

Pandas is an API

Pandas is actually set of software components, much of which is not even written in Python.

In [2]:
```
import pandas as pd
import numpy as np
import plotly.graph_objects as go
from plotly.offline import plot
import matplotlib.pyplot as plt
import datetime
from pycoingecko import CoinGeckoAPI
from mplfinance.original_flavor import candlestick2_ohlc
```

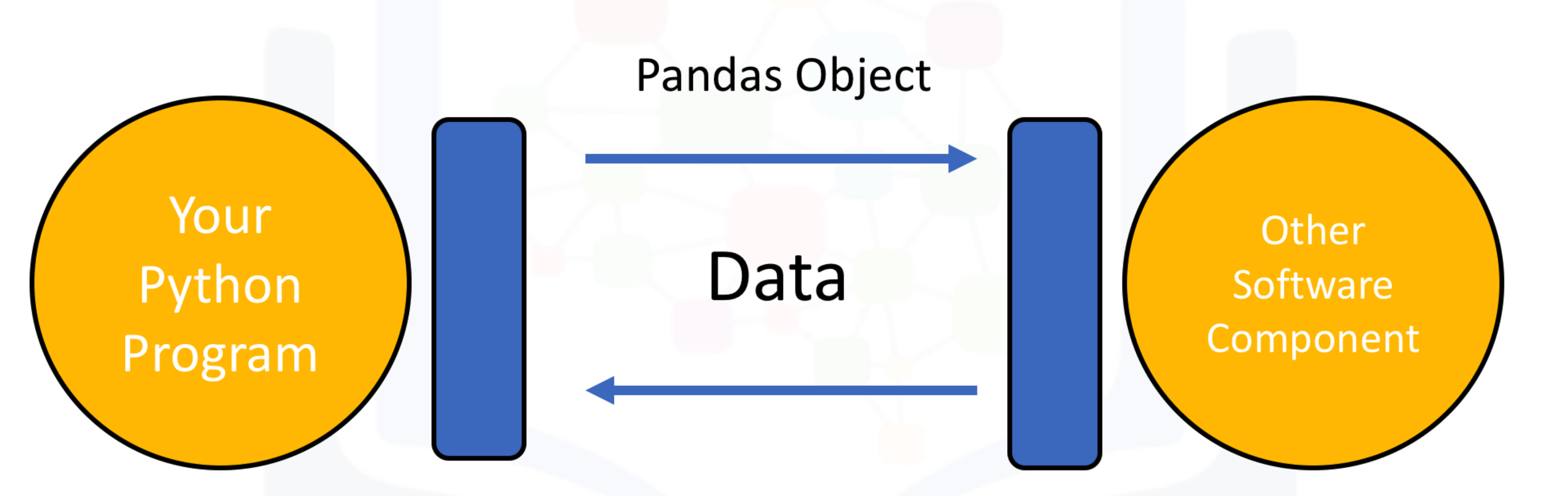
You create a dictionary, this is just data.

In [3]:
```
dict_={'a':[11,21,31],'b':[12,22,32]}
```

When you create a Pandas object with the Dataframe constructor in API lingo, this is an "instance". The data in the dictionary is passed along to the pandas API. You then use the dataframe to communicate with the API.

In [4]:
```
df=pd.DataFrame(dict_)
type(df)
```

Out[4]:
```
pandas.core.frame.DataFrame
```



When you call the method head the dataframe communicates with the API displaying the first few rows of the dataframe.

In [5]:
```
df.head()
```

Out[5]:

	a	b
0	11	12
1	21	22
2	31	32

When you call the method mean,the API will calculate the mean and return the value.

In [6]:
```
df.mean()
```

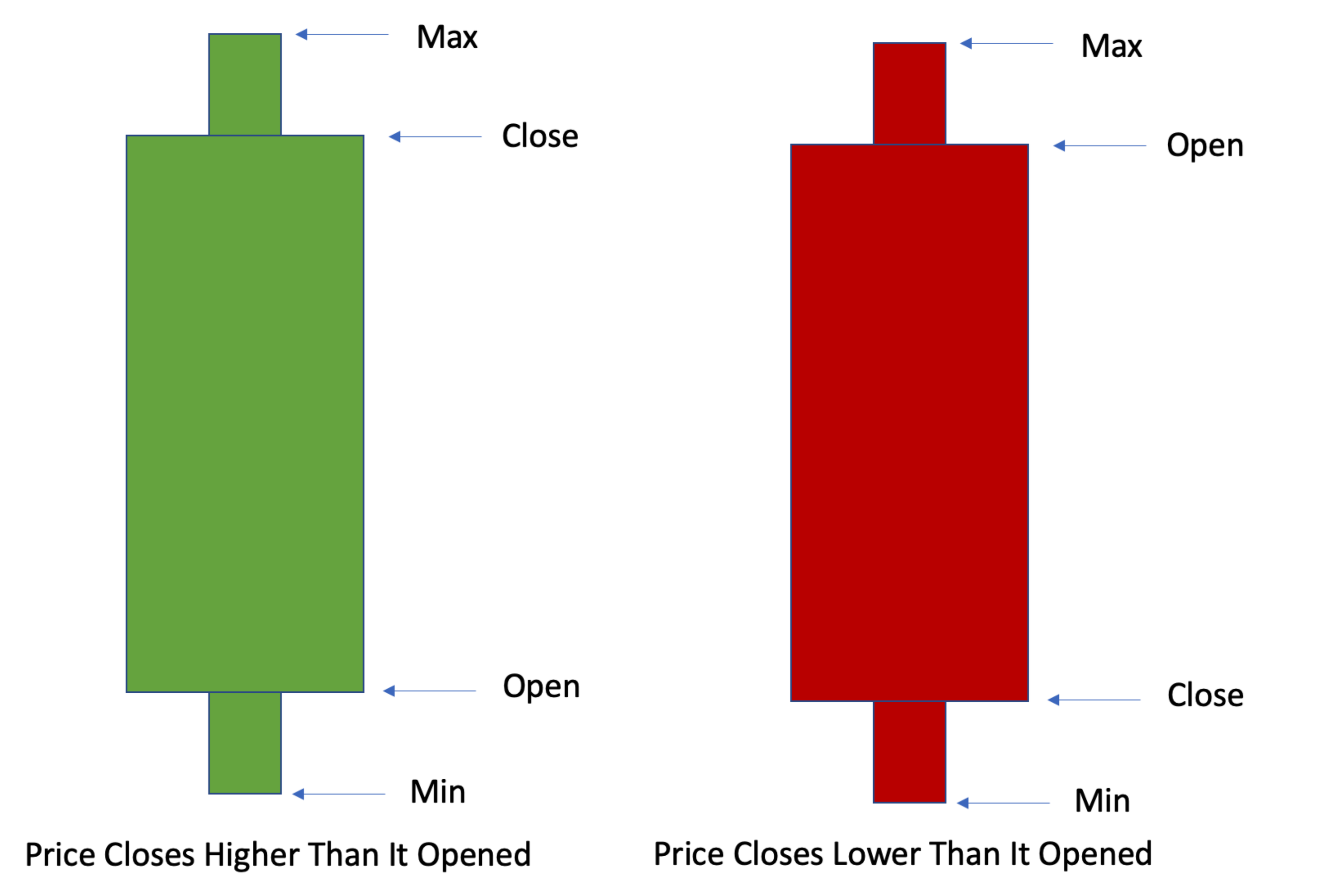
Out[6]:
```
a    21.0
b    22.0
dtype: float64
```

REST APIs

Rest API’s function by sending a request, the request is communicated via HTTP message. The HTTP message usually contains a JSON file. This contains instructions for what operation we would like the service or resource to perform. In a similar manner, API returns a response, via an HTTP message, this response is usually contained within a JSON.

In cryptocurrency a popular method to display the movements of the price of a currency.

Here is a description of the candle sticks.



In this lab, we will be using the CoinGecko API to create one of these candlestick graphs for Bitcoin. We will use the API to get the price data for 30 days with 24 observation per day, 1 per hour. We will find the max, min, open, and close price per day meaning we will have 30 candlesticks and use that to generate the candlestick graph. Although we are using the CoinGecko API we will use a Python client/wrapper for the API called PyCoinGecko. PyCoinGecko will make performing the requests easy and it will deal with the enpoint targeting.

Lets start off by getting the data we need. Using the `get_coin_market_chart_by_id(id, vs_currency, days)`. `id` is the name of the coin you want, `vs_currency` is the currency you want the price in, and `days` is how many days back from today you want.

In [7]:
```
cg = CoinGeckoAPI()

bitcoin_data = cg.get_coin_market_chart_by_id(id='bitcoin', vs_currency='usd', days=30)
```

In [8]:
```
type(bitcoin_data )
```

Out[8]:
```
dict
```

The response we get is in the form of a JSON which includes the price, market caps, and total volumes along with timestamps for each observation. We are focused on the prices so we will select that data.

In [9]:
```
bitcoin_price_data = bitcoin_data['prices']

bitcoin_price_data[0:5]
```

Out[9]:
```
[[1617901315072, 57678.64518365409],
 [1617904931469, 57707.785635819404],
 [1617908759730, 57957.14832967089],
 [1617912168837, 57739.52846765616],
 [1617916006984, 57645.28777969739]]
```

Finally lets turn this data into a Pandas DataFrame.

In [10]:
```
data = pd.DataFrame(bitcoin_price_data, columns=['TimeStamp', 'Price'])
```

In [11]:
```
data
```

Out[11]:

	TimeStamp	Price
0	1617901315072	57678.645184
1	1617904931469	57707.785636
2	1617908759730	57957.148330
3	1617912168837	57739.528468
4	1617916006984	57645.287780
...	...	...
717	1620481122011	59271.441750
718	1620482463942	59228.121916
719	1620486014013	59111.943634
720	1620489741773	57945.792913
721	1620492968000	58324.604926

722 rows × 2 columns

Now that we have the DataFrame we will convert the timestamp to datetime and save it as a column called `Date`. We will map our `unix_to_datetime` to each timestamp and convert it to a readable datetime.

In [12]:
```
data['Date'] = pd.to_datetime(data['TimeStamp'], unit='ms')
```

In [13]:
```
data
```

Out[13]:

	TimeStamp	Price	Date
0	1617901315072	57678.645184	2021-04-08 17:01:55.072
1	1617904931469	57707.785636	2021-04-08 18:02:11.469
2	1617908759730	57957.148330	2021-04-08 19:05:59.730
3	1617912168837	57739.528468	2021-04-08 20:02:48.837
4	1617916006984	57645.287780	2021-04-08 21:06:46.984
...	...	...	...
717	1620481122011	59271.441750	2021-05-08 13:38:42.011
718	1620482463942	59228.121916	2021-05-08 14:01:03.942
719	1620486014013	59111.943634	2021-05-08 15:00:14.013
720	1620489741773	57945.792913	2021-05-08 16:02:21.773
721	1620492968000	58324.604926	2021-05-08 16:56:08.000

722 rows × 3 columns

Using this modified dataset we can now group by the `Date` and find the min, max, open, and close for the candlesticks.

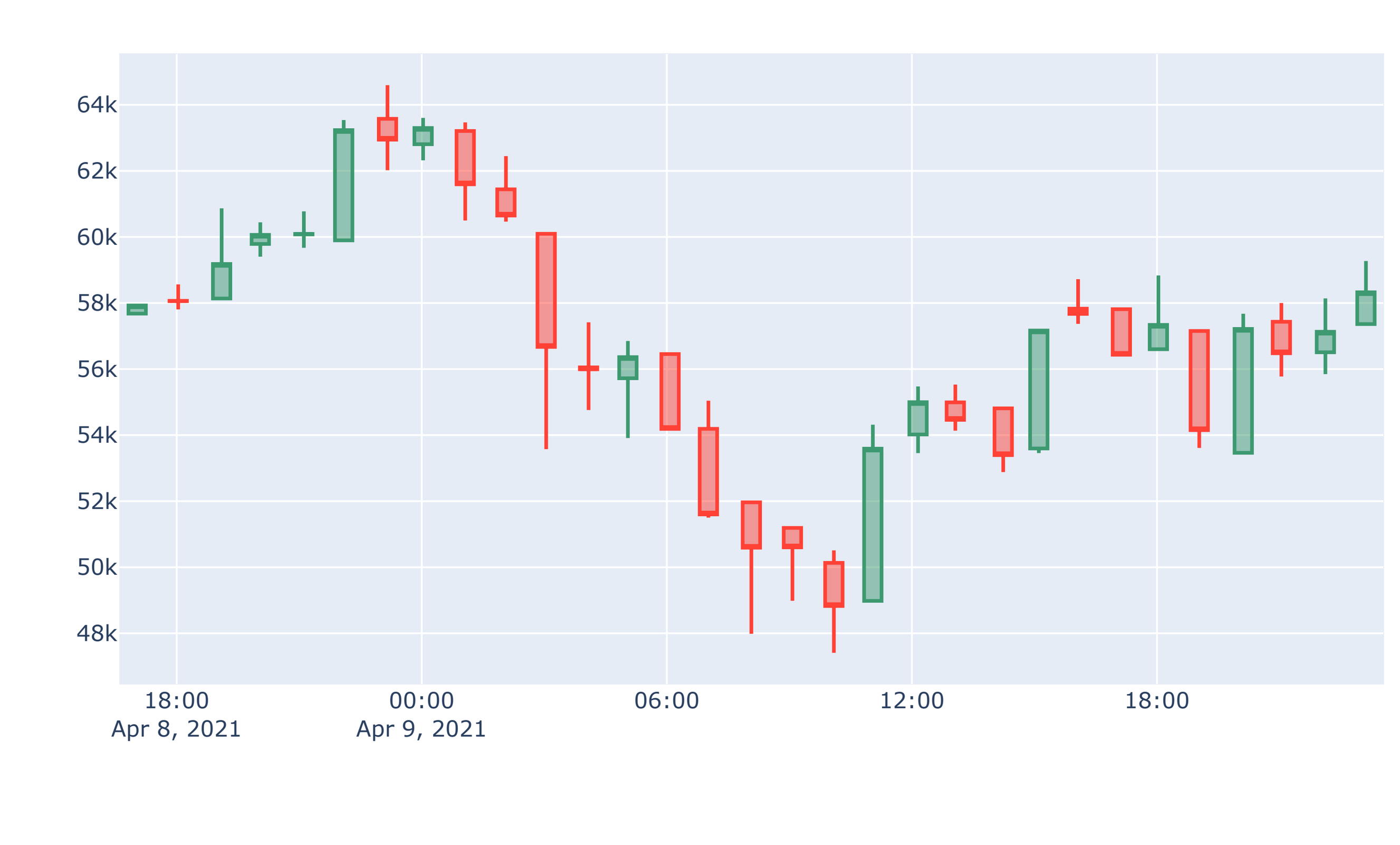
In [14]:
```
candlestick_data = data.groupby(data.Date.dt.date, as_index=False).agg({"Price": ['min', 'max', 'first', 'last'
```

Finally we are now ready to use plotly to create our Candlestick Chart.

In [15]:
```
fig = go.Figure(data = [go.Candlestick(x = data['Date'],
                open = candlestick_data['Price']['first'],
                high = candlestick_data['Price']['max'],
                low = candlestick_data['Price']['min'],
                close = candlestick_data['Price']['last'])
                ])

fig.update_layout(xaxis_rangeslider_visible = False)

fig.show()
```



Authors:

Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2020-11-23	3.0	Azim Hirjani	New API
2020-09-09	2.1	Malika Singla	Spell Check
2020-08-26	2.0	Lavanya	Moved lab to course repo in GitLab