Python初級數據分析員證書

（五）進階Python數據分析及可視化技巧

12.Plotly套件 Part 2

plotly | Graphing Libraries

Recap

plotly | Graphing Libraries

In last chapter we had learnt

- Bubble chart
- Interactive graphing
- Discrete colour and continuous colour
- Facet plots
- Plotly Express
- Matplotlib vs Plotly

12.Plotly套件 Part 2

Chapter Summary

- Time Series Data and Plotly
- Series.isna, dropna, fillna, df.rank
- Range slider
- Multiple plot with graph_objects
- Candlestick chart
- OHLC chart

plotly | Graphing Libraries

Time Series Data

In Chapter 8 Pandas part 2, we learnt that the time series data is is a collection of observations obtained through repeated measurements over time. Plot the points on a graph, and one of your axes (usually x-axis) would always be time.

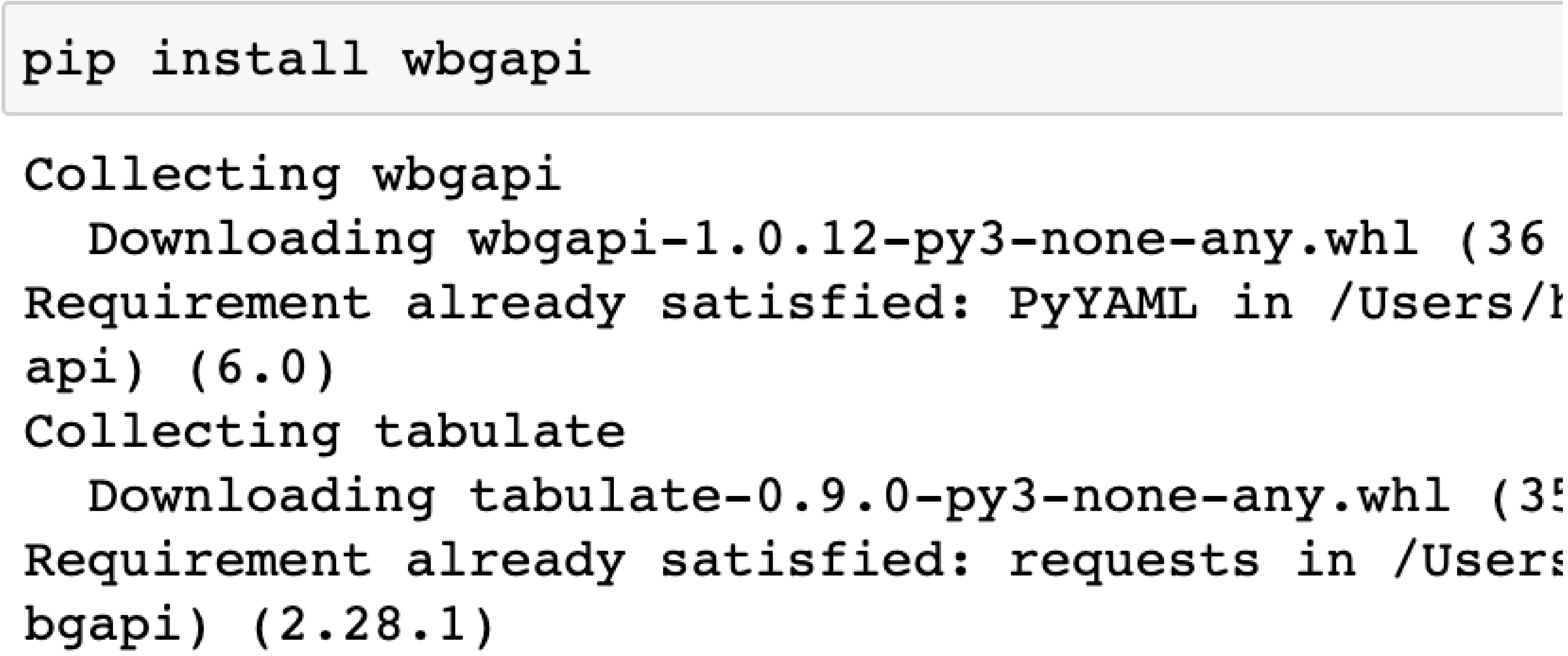
Time series data is an important form of structured data in many different fields, such as finance, economics, ecology, neuroscience, and physics.

Time Series Plots in Plotly

Time series can be represented using either plotly.express functions (px.line, px.scatter, px.bar etc) or plotly.graph_objects charts objects (go.Scatter, go.Bar etc).

API data from WorldBank

Many organization open their database with API, WorldBank.org is one of them and supports Python library as well.



```
pip install wbgapi
```

```
Collecting wbgapi
  Downloading wbgapi-1.0.12-py3-none-any.whl (36
Requirement already satisfied: PyYAML in /Users/
api) (6.0)
Collecting tabulate
  Downloading tabulate-0.9.0-py3-none-any.whl (35
Requirement already satisfied: requests in /Users
bgapi) (2.28.1)
```

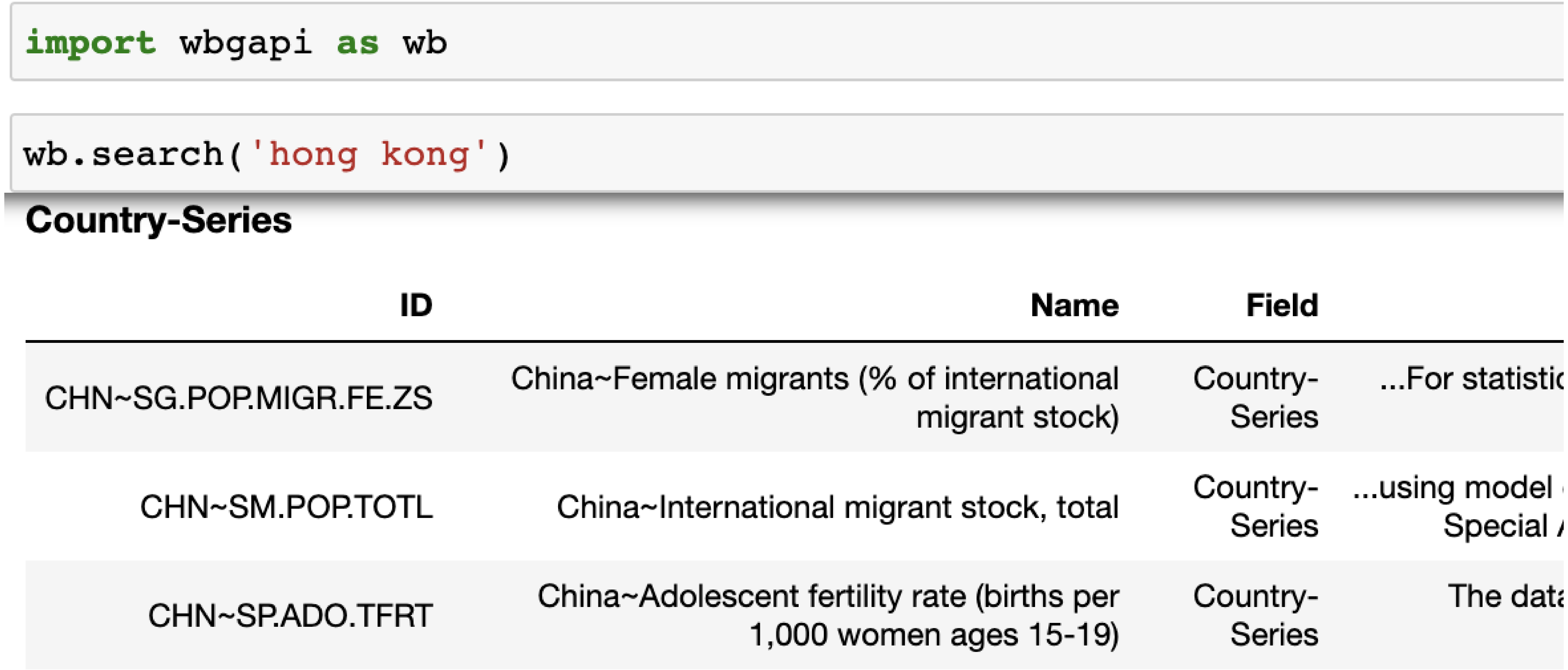
wbgapi 1.0.12

```
pip install wbgapi
```

https://blogs.worldbank.org/opendata/introducing-wbgapi-new-python-package-accessing-world-bank-data

API data from WorldBank

Firstly, import wbgapi as wb. Then we search ‘hong kong’



```
import wbgapi as wb
```

```
wb.search('hong kong')
```

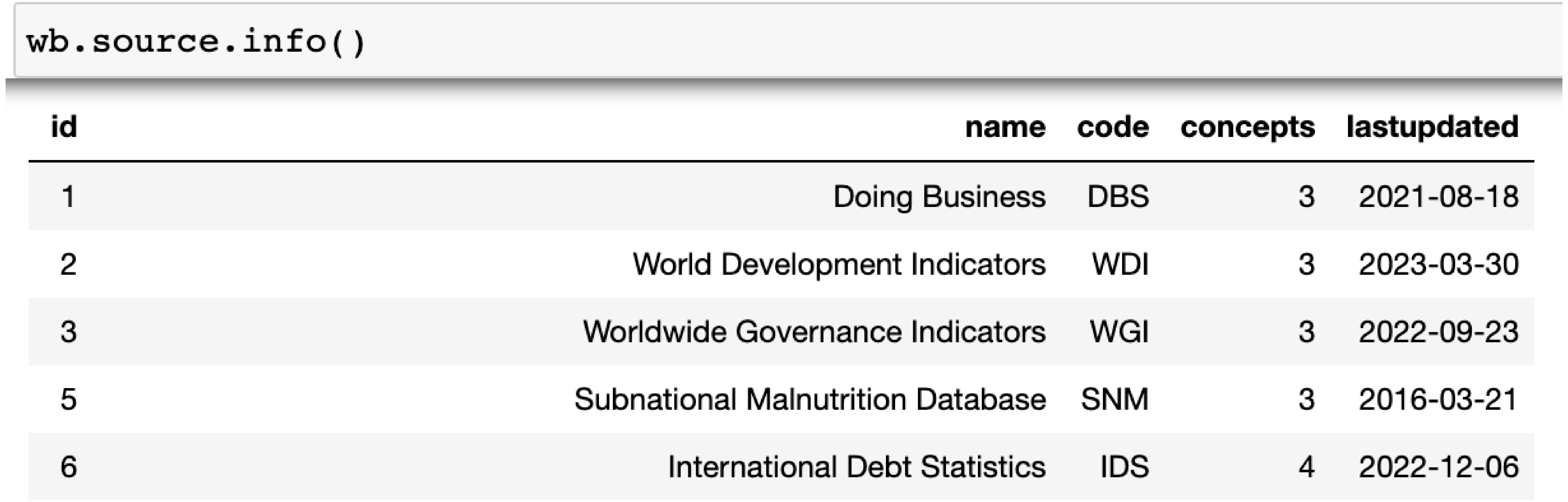
Country-Series

ID	Name	Field	
CHN~SG.POP.MIGR.FE.ZS	China~Female migrants (% of international migrant stock)	Country-Series	...For statistic
CHN~SM.POP.TOTL	China~International migrant stock, total	Country-Series	...using model Special
CHN~SP.ADO.TFRT	China~Adolescent fertility rate (births per 1,000 women ages 15-19)	Country-Series	The data

API data from WorldBank

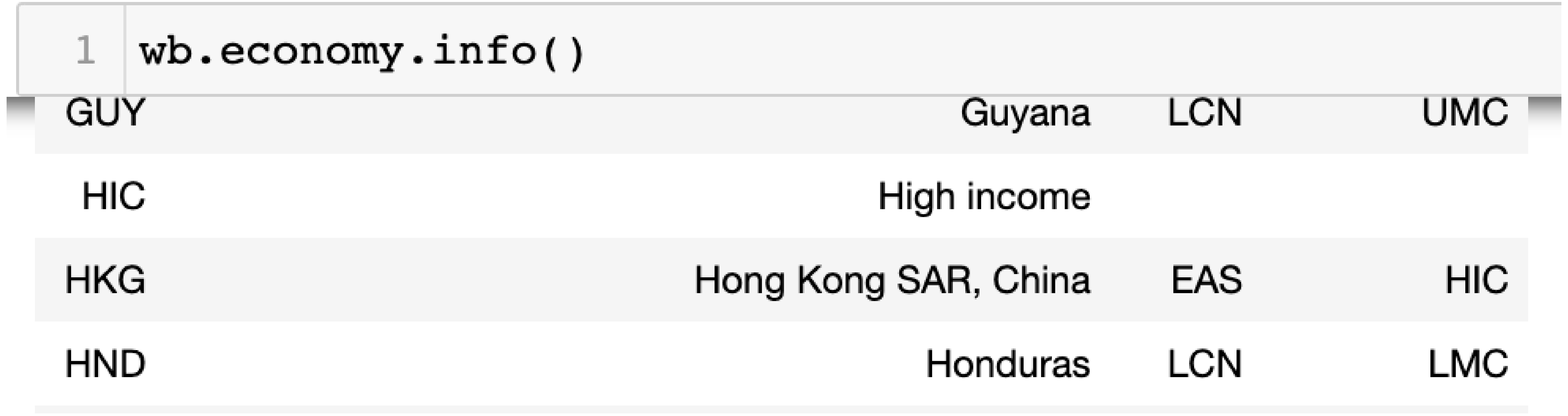
There are a lot of data over decades, well organised and FREE. And it’s served in Pandas DataFrame, even better than many paid services.

THE WORLD BANK
IBRD • IDA



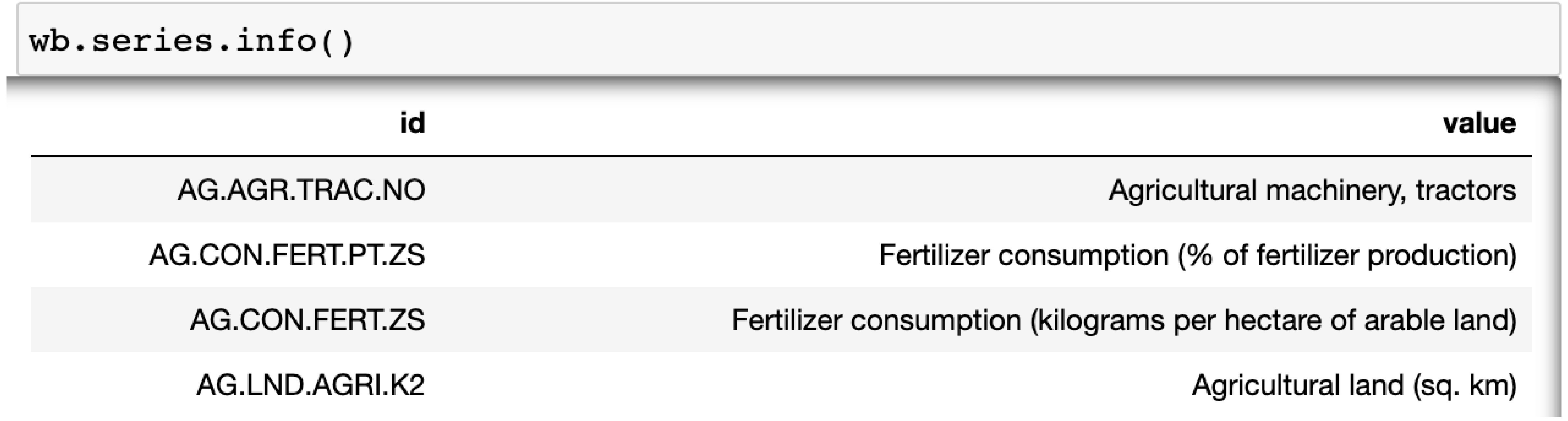
```
wb.source.info()
```

id	name	code	concepts	lastupdated
1	Doing Business	DBS	3	2021-08-18
2	World Development Indicators	WDI	3	2023-03-30
3	Worldwide Governance Indicators	WGI	3	2022-09-23
5	Subnational Malnutrition Database	SNM	3	2016-03-21
6	International Debt Statistics	IDS	4	2022-12-06



```
wb.economy.info()
```

GUY	Guyana	LCN	UMC
HIC	High income		
HKG	Hong Kong SAR, China	EAS	HIC
HND	Honduras	LCN	LMC

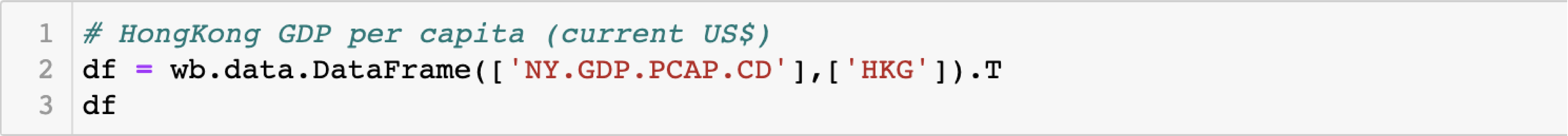


```
wb.series.info()
```

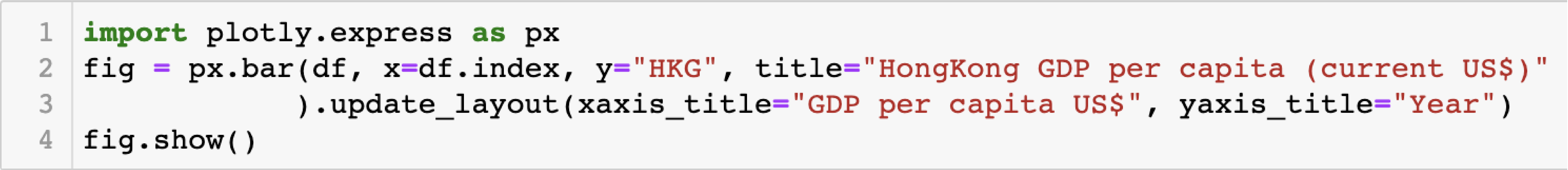
id	value
AG.AGR.TRAC.NO	Agricultural machinery, tractors
AG.CON.FERT.PT.ZS	Fertilizer consumption (% of fertilizer production)
AG.CON.FERT.ZS	Fertilizer consumption (kilograms per hectare of arable land)
AG.LND.AGRI.K2	Agricultural land (sq. km)

Capture Hong Kong GDP data and plot via plotly

Pull Hong Kong’s GDP data. Since the columns is in years, we use T(transpose) to switch them for rows.

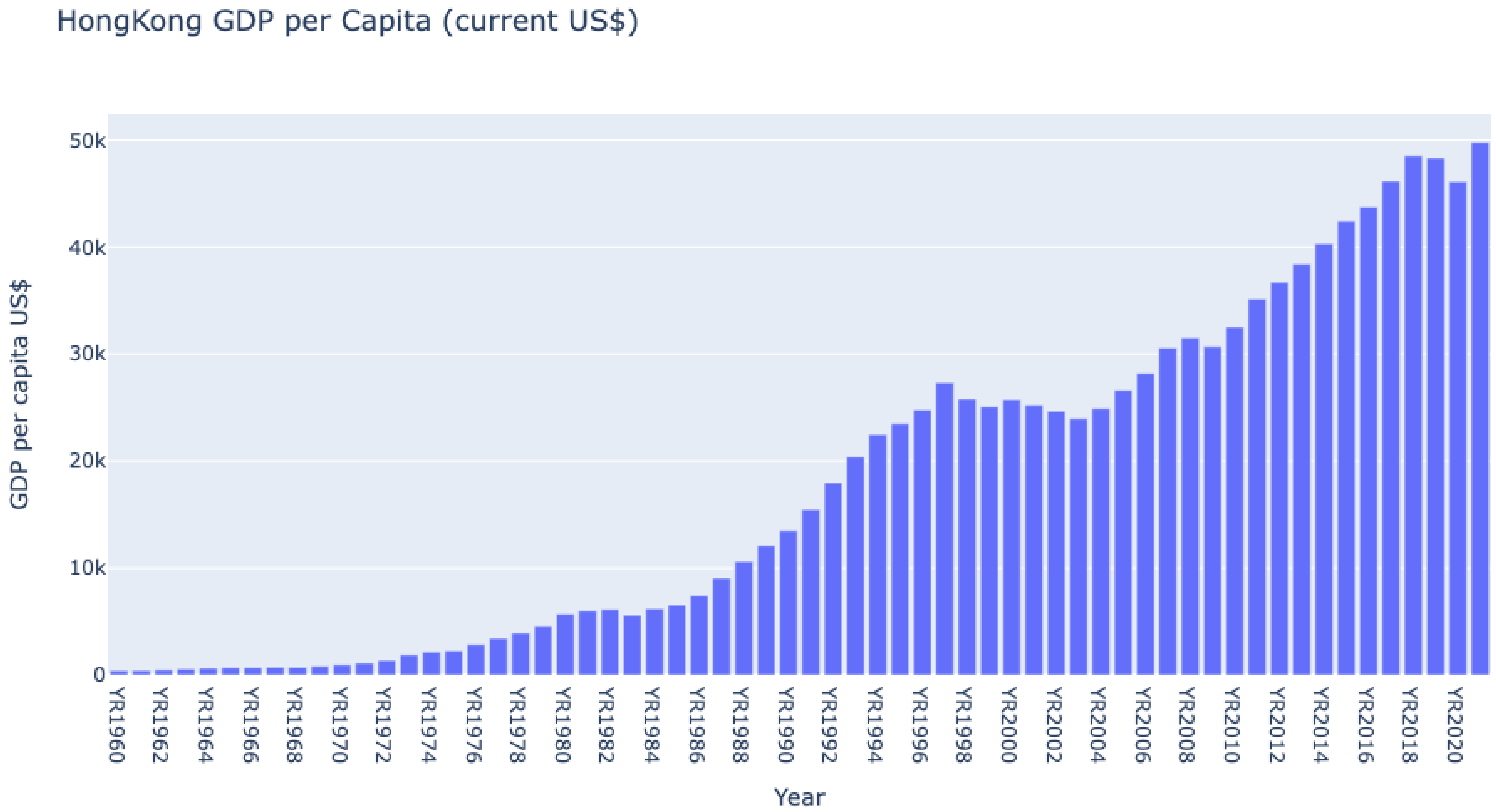


```
# HongKong GDP per capita (current US$)
df = wb.data.DataFrame(['NY.GDP.PCAP.CD'],['HKG']).T
df
```

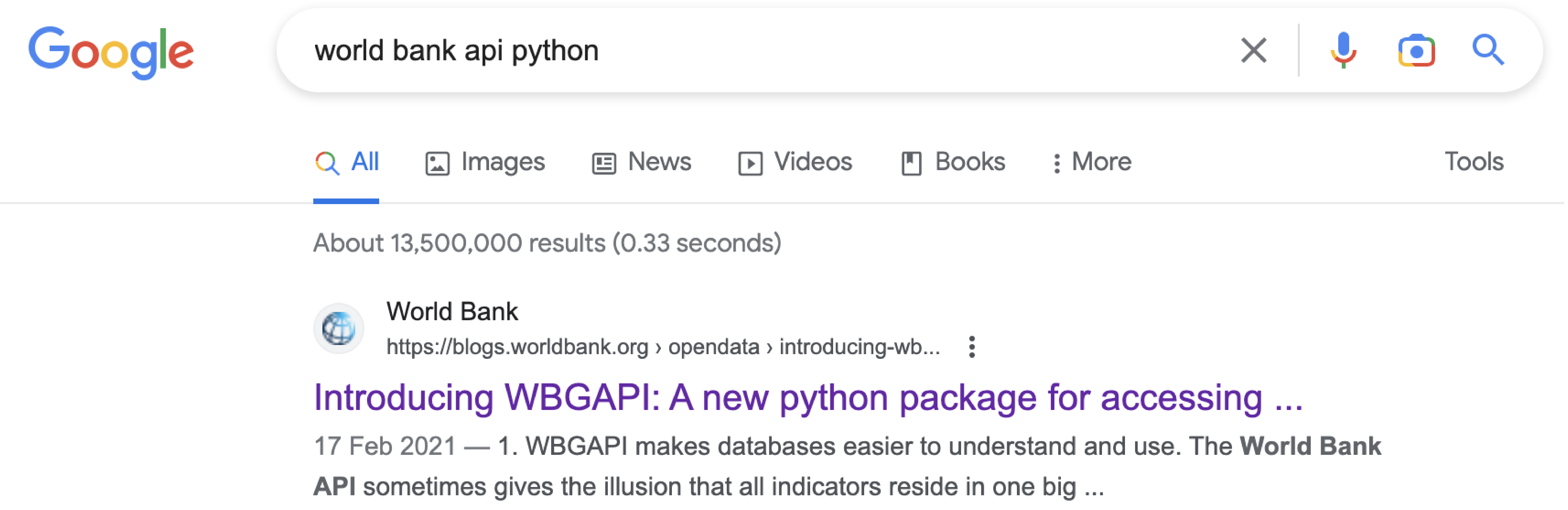


```
import plotly.express as px
fig = px.bar(df, x=df.index, y="HKG", title="HongKong GDP per capita (current US$)"
            ).update_layout(xaxis_title="GDP per capita US$", yaxis_title="Year")
fig.show()
```

Hong Kong GDP per Capita



WorldBank API Usage info



About 13,500,000 results (0.33 seconds)

World Bank
https://blogs.worldbank.org › opendata › introducing-wb...

Introducing WBGAPI: A new python package for accessing ...

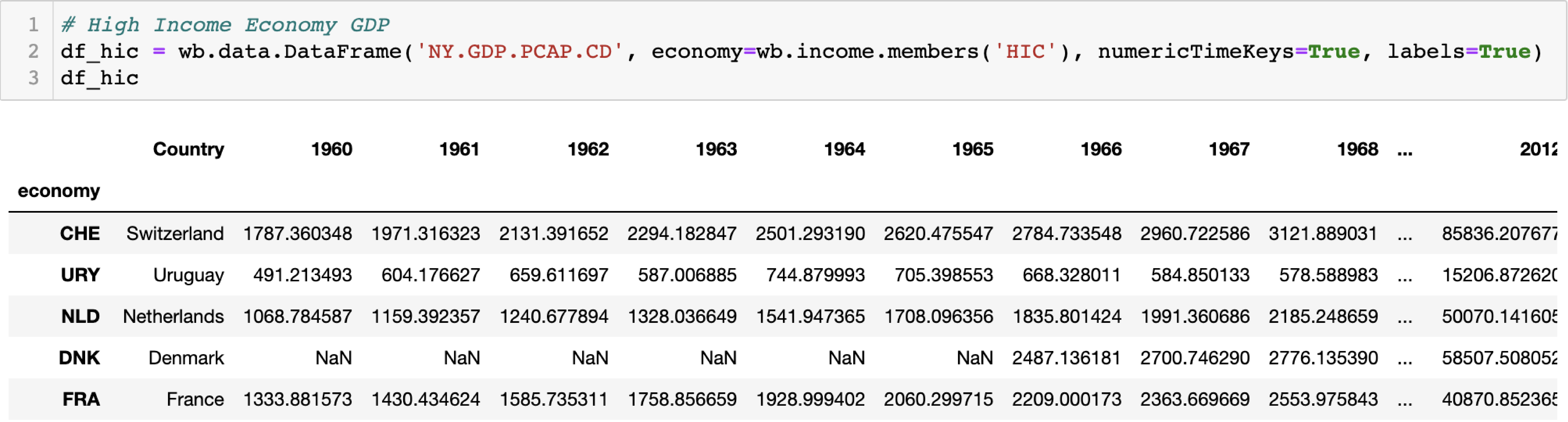
17 Feb 2021 — 1. WBGAPI makes databases easier to understand and use. The **World Bank API** sometimes gives the illusion that all indicators reside in one big ...

https://blogs.worldbank.org/opendata/introducing-wbgapi-new-python-package-accessing-world-bank-data

THE WORLD BANK
IBRD • IDA

High Income Econmy GDP

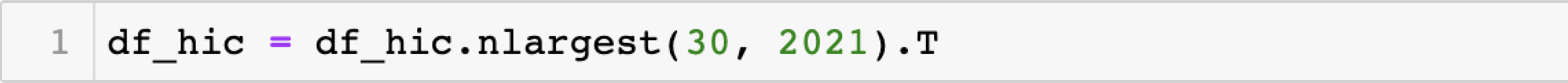
Pull High Income Countries.



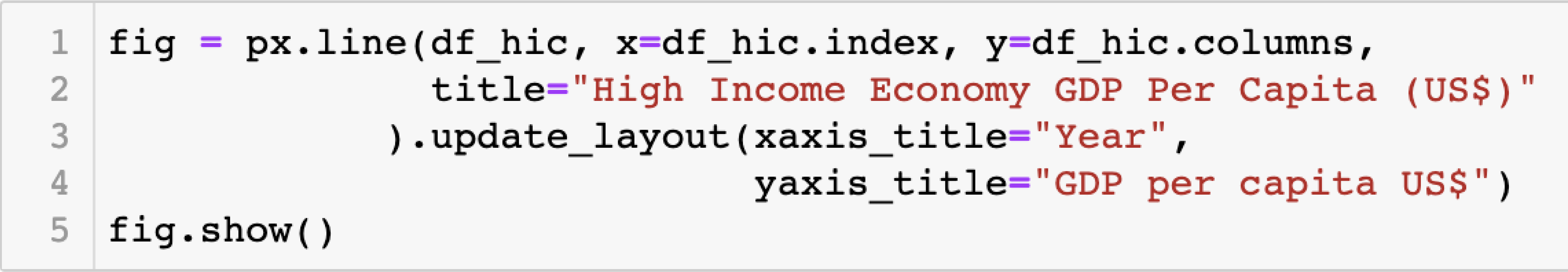
```python
# High Income Economy GDP
df_hic = wb.data.DataFrame('NY.GDP.PCAP.CD', economy=wb.income.members('HIC'), numericTimeKeys=True, labels=True)
df_hic
```

	Country	1960	1961	1962	1963	1964	1965	1966	1967	1968	...	2012
economy												
CHE	Switzerland	1787.360348	1971.316323	2131.391652	2294.182847	2501.293190	2620.475547	2784.733548	2960.722586	3121.889031	...	85836.207677
URY	Uruguay	491.213493	604.176627	659.611697	587.006885	744.879993	705.398553	668.328011	584.850133	578.588983	...	15206.872620
NLD	Netherlands	1068.784587	1159.392357	1240.677894	1328.036649	1541.947365	1708.096356	1835.801424	1991.360686	2185.248659	...	50070.141605
DNK	Denmark	NaN	NaN	NaN	NaN	NaN	NaN	2487.136181	2700.746290	2776.135390	...	58507.508052
FRA	France	1333.881573	1430.434624	1585.735311	1758.856659	1928.999402	2060.299715	2209.000173	2363.669669	2553.975843	...	40870.852365

Select top 30 largest GDP in 2021 and transpose the years columns to rows.

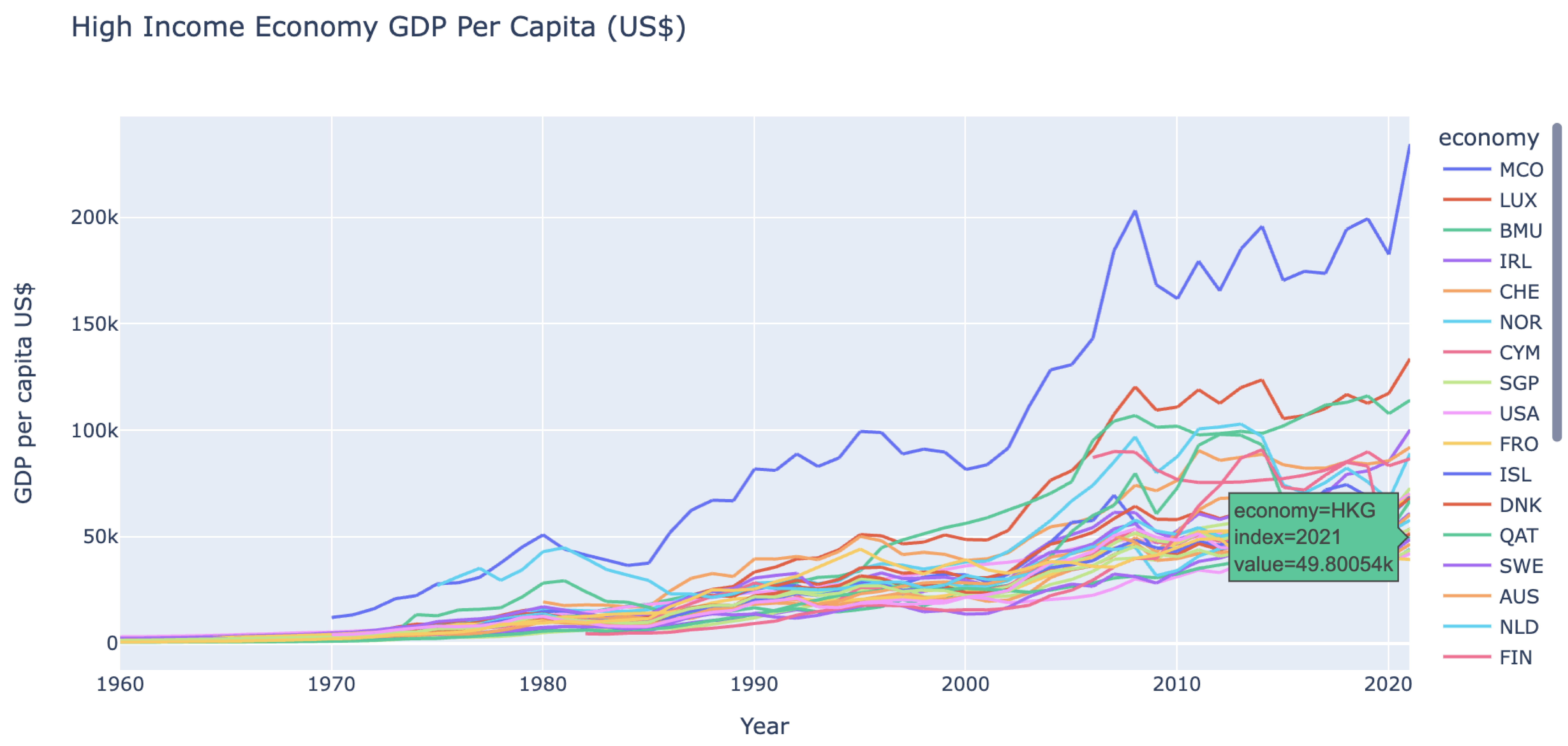


```python
df_hic = df_hic.nlargest(30, 2021).T
```

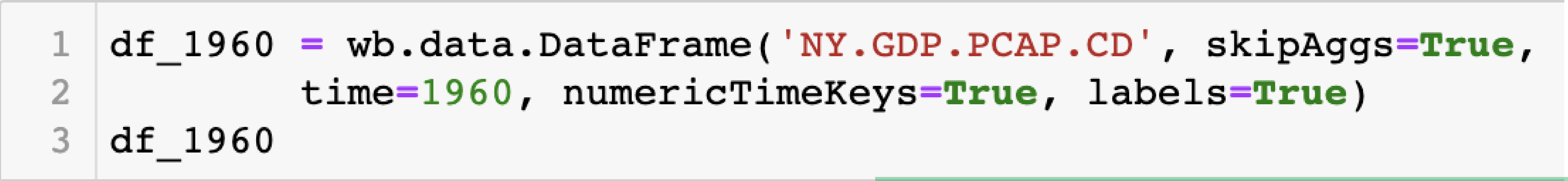


```python
fig = px.line(df_hic, x=df_hic.index, y=df_hic.columns,
              title="High Income Economy GDP Per Capita (US$)"
            ).update_layout(xaxis_title="Year",
                            yaxis_title="GDP per capita US$")
fig.show()
```

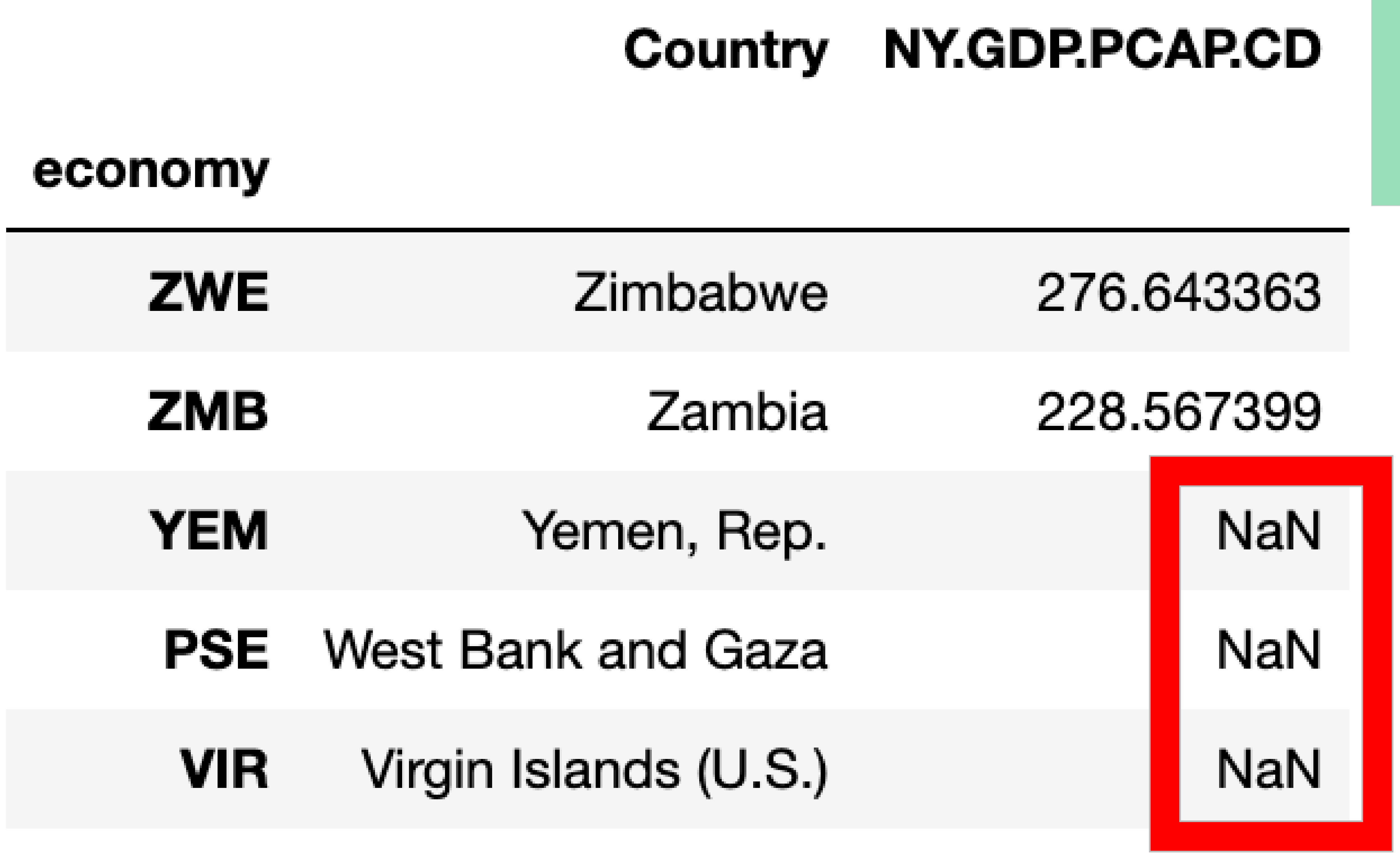
Hong Kong ranked Top 23 GDP in 2021



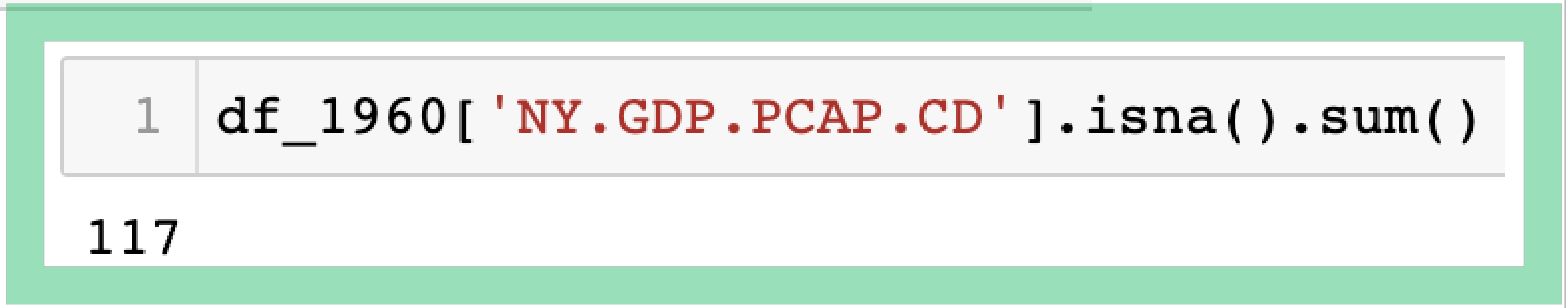
Hong Kong GDP rank in 1960



```
df_1960 = wb.data.DataFrame('NY.GDP.PCAP.CD', skipAggs=True,
          time=1960, numericTimeKeys=True, labels=True)
df_1960
```



economy	Country	NY.GDP.PCAP.CD
ZWE	Zimbabwe	276.643363
ZMB	Zambia	228.567399
YEM	Yemen, Rep.	NaN
PSE	West Bank and Gaza	NaN
VIR	Virgin Islands (U.S.)	NaN

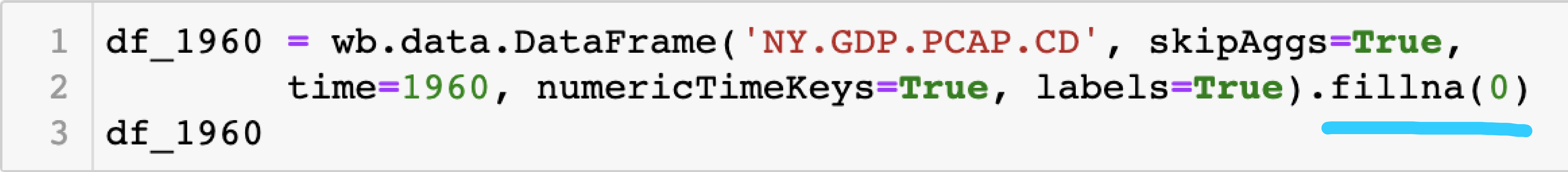


```
df_1960['NY.GDP.PCAP.CD'].isna().sum()
```

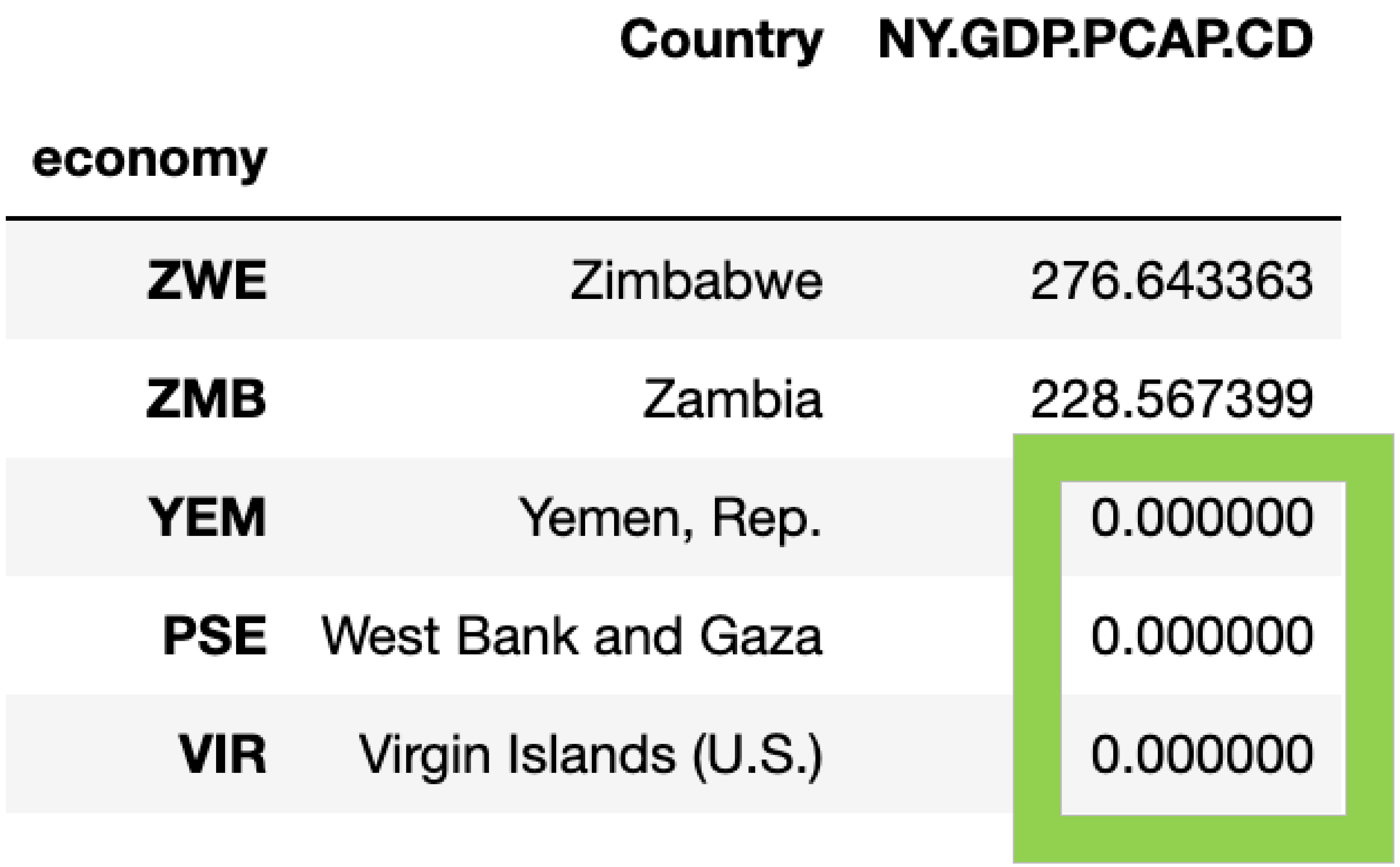
```
117
```

As there are NaN data in almost half the countries, we should **not** use dropna() to delete the data.

Hong Kong GDP rank in 1960



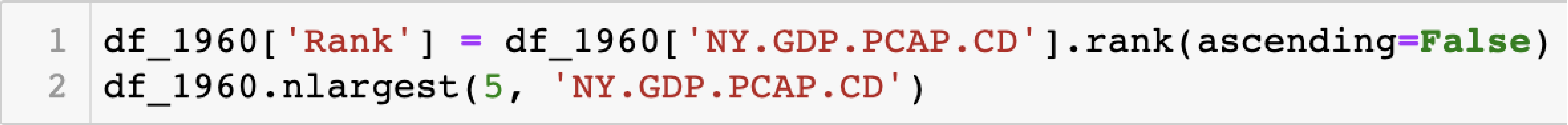
```
df_1960 = wb.data.DataFrame('NY.GDP.PCAP.CD', skipAggs=True,
          time=1960, numericTimeKeys=True, labels=True).fillna(0)
df_1960
```



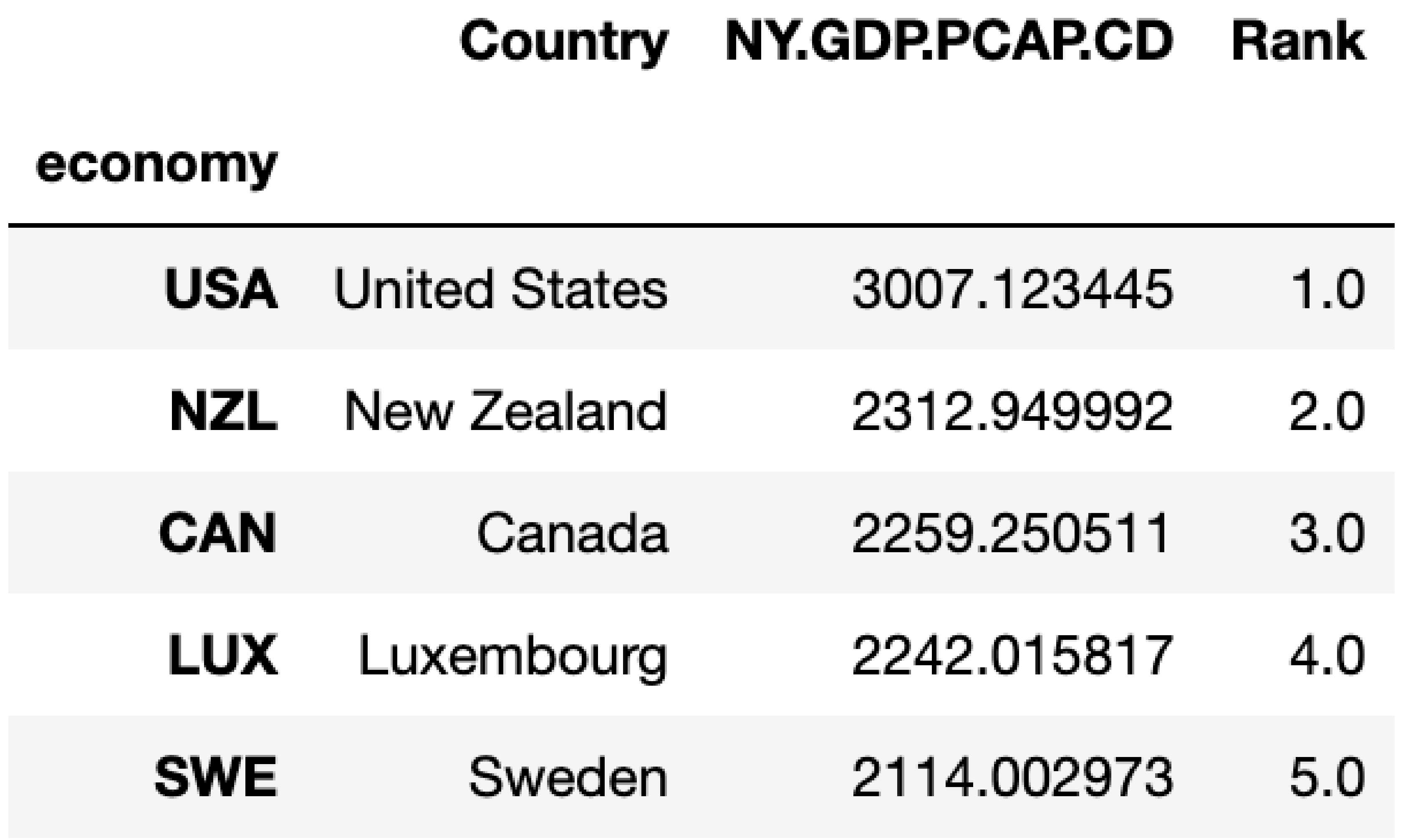
	Country	NY.GDP.PCAP.CD
economy		
ZWE	Zimbabwe	276.643363
ZMB	Zambia	228.567399
YEM	Yemen, Rep.	0.000000
PSE	West Bank and Gaza	0.000000
VIR	Virgin Islands (U.S.)	0.000000

We use fillna with zero to handle the NaN properly.

Hong Kong GDP rank in 1960

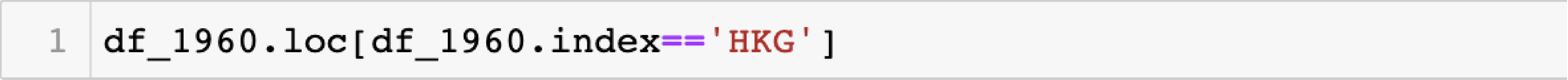


```
df_1960['Rank'] = df_1960['NY.GDP.PCAP.CD'].rank(ascending=False)
df_1960.nlargest(5, 'NY.GDP.PCAP.CD')
```

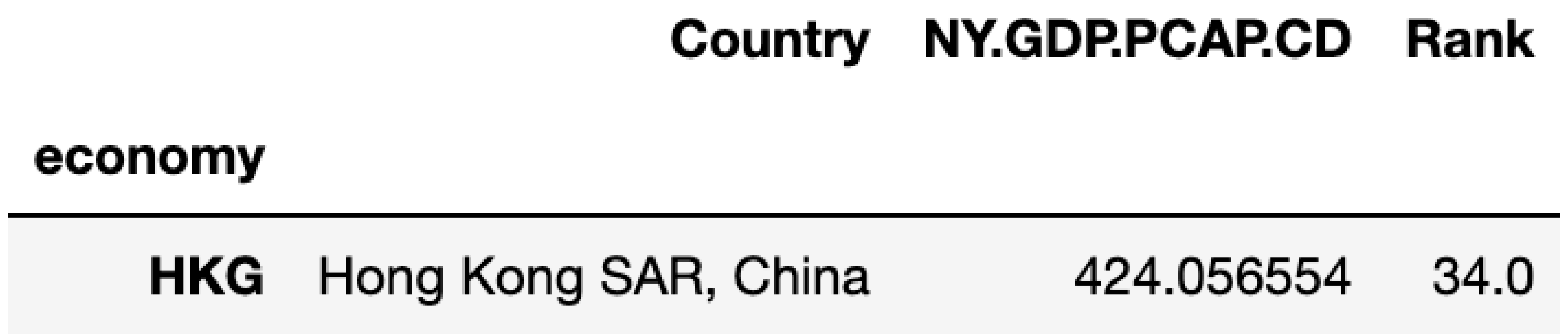


	Country	NY.GDP.PCAP.CD	Rank
economy			
USA	United States	3007.123445	1.0
NZL	New Zealand	2312.949992	2.0
CAN	Canada	2259.250511	3.0
LUX	Luxembourg	2242.015817	4.0
SWE	Sweden	2114.002973	5.0

We add a Rank column and .rank() based on GDP. ascending=False means the biggest number on top of the rank. This ranking process is used quite often in data analysis.



```
df_1960.loc[df_1960.index=='HKG']
```

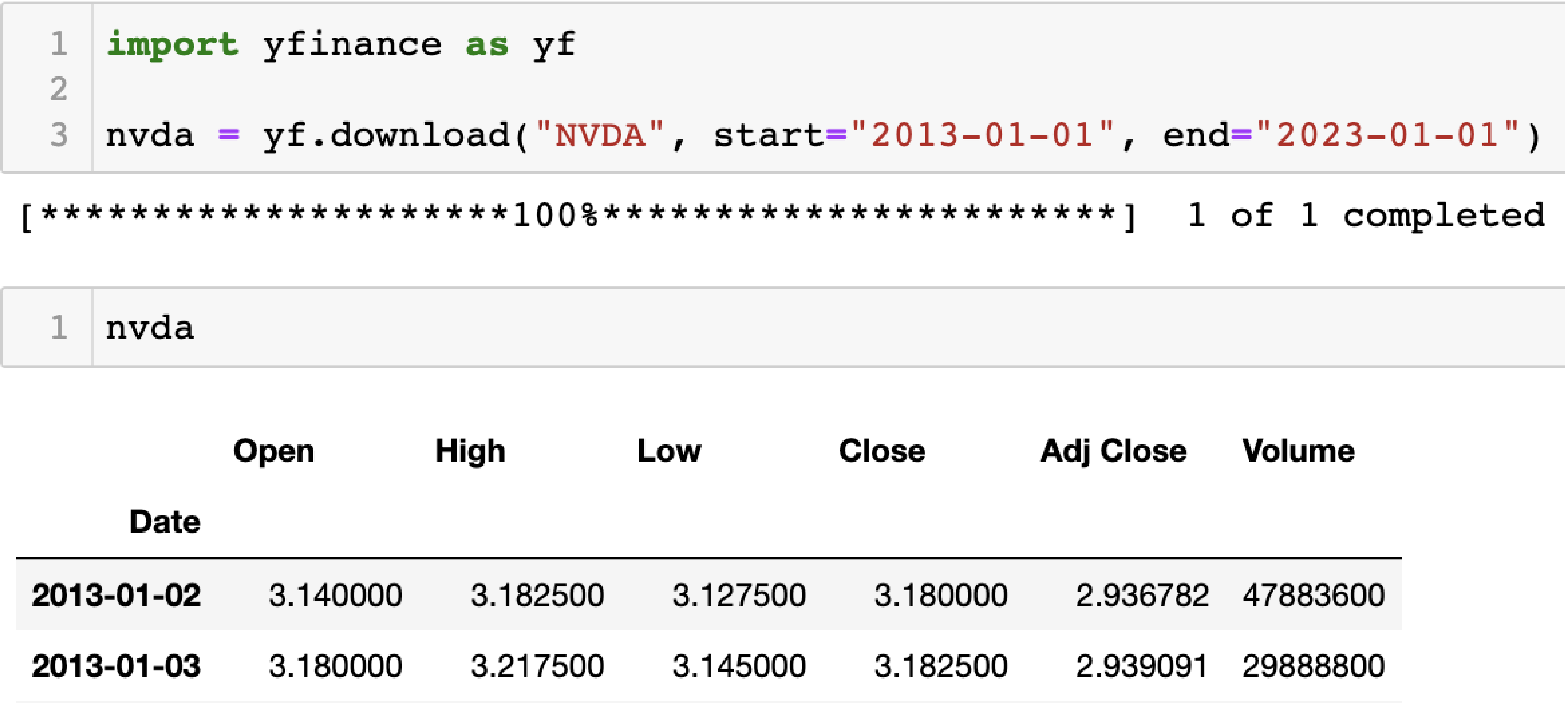


	Country	NY.GDP.PCAP.CD	Rank
economy			
HKG	Hong Kong SAR, China	424.056554	34.0

Filter the DF then we know Hong Kong rank top 34 in 1960. 👏 👏 👏

Time Series plot with Range Slider

Pull finance data from yfinance library.



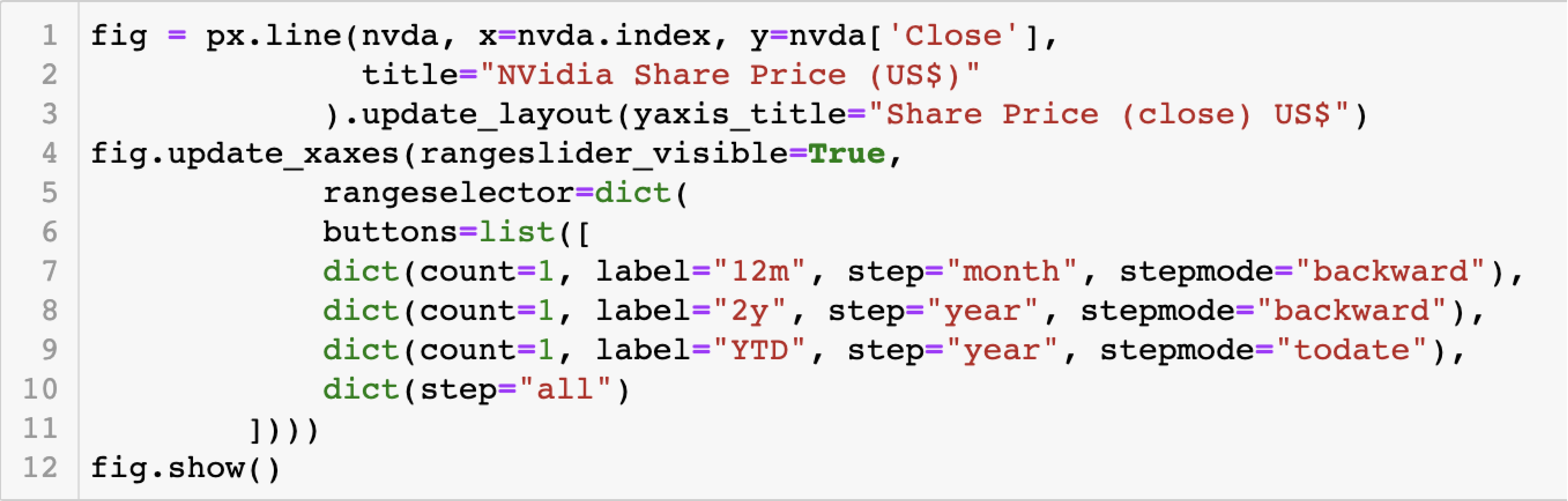
```python
import yfinance as yf

nvda = yf.download("NVDA", start="2013-01-01", end="2023-01-01")
```

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

```python
nvda
```

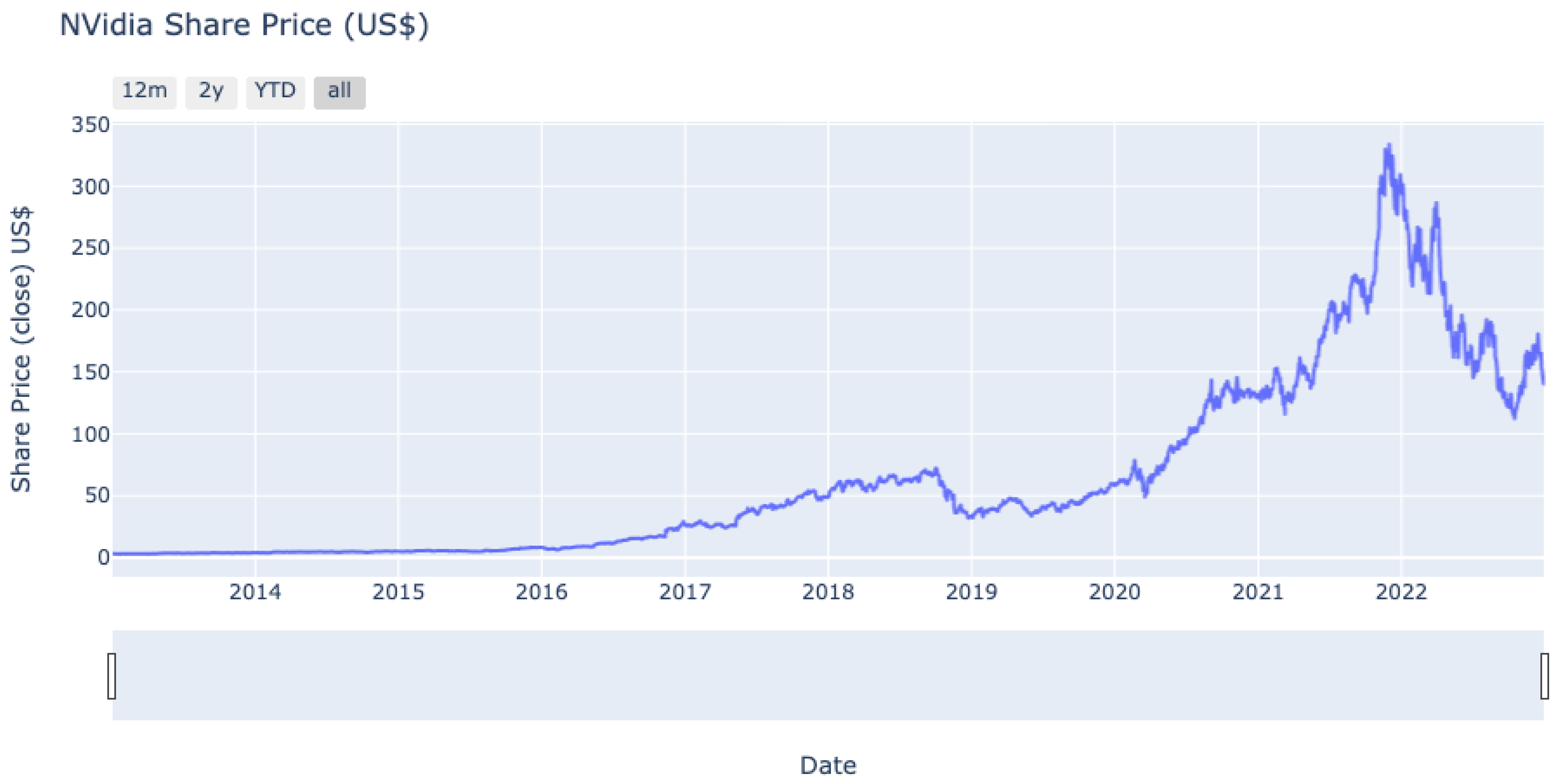
	Open	High	Low	Close	Adj Close	Volume
Date						
2013-01-02	3.140000	3.182500	3.127500	3.180000	2.936782	47883600
2013-01-03	3.180000	3.217500	3.145000	3.182500	2.939091	29888800

Time Series plot with Range Slider



```python
fig = px.line(nvda, x=nvda.index, y=nvda['Close'],
              title="NVidia Share Price (US$)"
            ).update_layout(yaxis_title="Share Price (close) US$")
fig.update_xaxes(rangeslider_visible=True,
            rangeselector=dict(
            buttons=list([
            dict(count=1, label="12m", step="month", stepmode="backward"),
            dict(count=1, label="2y", step="year", stepmode="backward"),
            dict(count=1, label="YTD", step="year", stepmode="todate"),
            dict(step="all")
        ])))
fig.show()
```

Time Series plot with Range Slider



Multiple plots with graph_objects

We can import multiple stock data into a single DF, but pay attention to the columns.

```
df_chip = yf.download(["NVDA","TSM"], start="2013-01-01", end="2023-01-01")
df_chip
```

```
[*********************100%***********************]  2 of 2
```

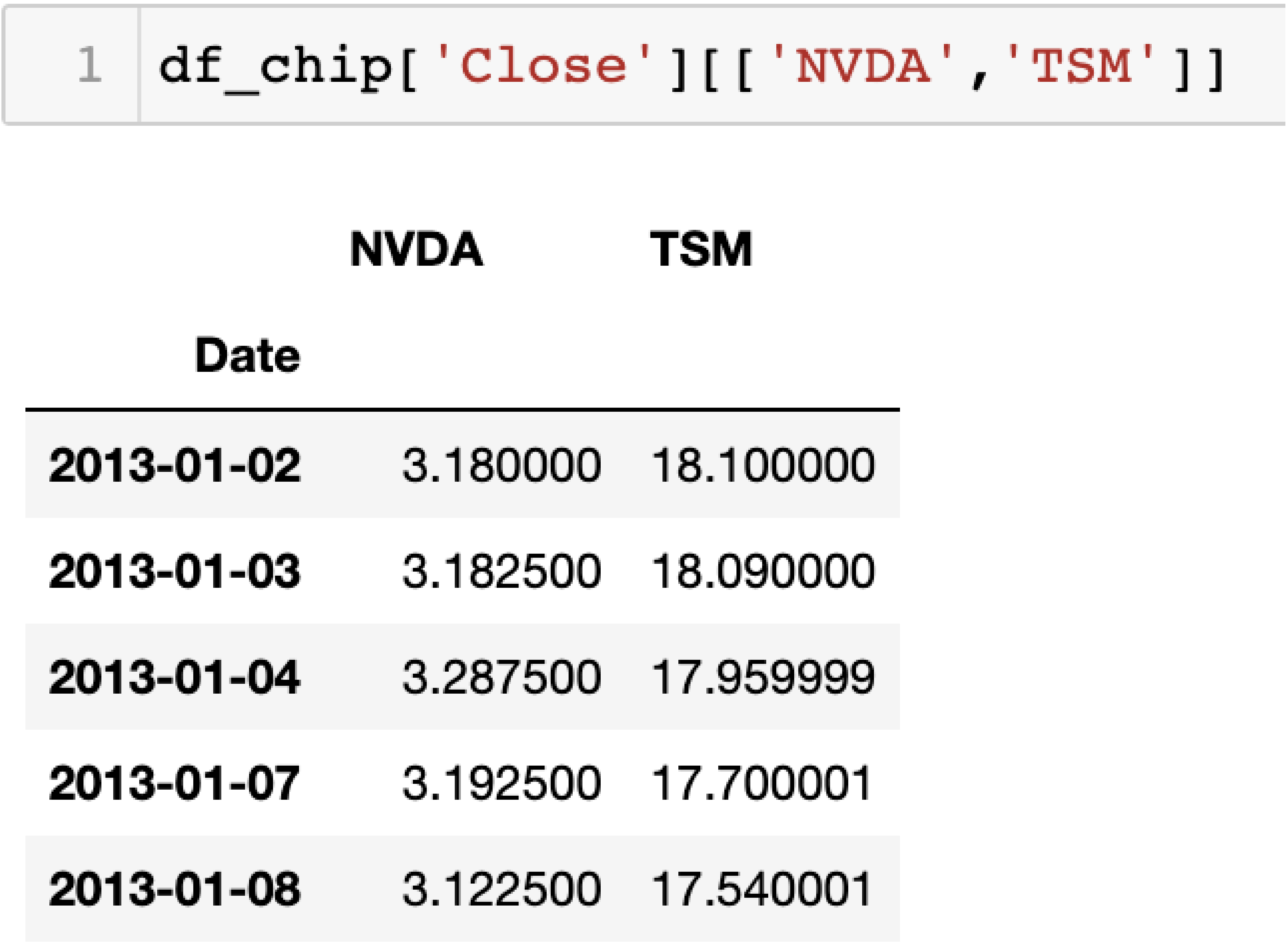
	Adj Close		Close		High		
	NVDA	TSM	NVDA	TSM	NVDA	TSM	
Date							
2013-01-02	2.936782	13.400276	3.180000	18.100000	3.182500	18.1200	6100
2013-01-03	2.939090	13.392871	3.182500	18.090000	3.217500	18.2999	8600
2013-01-04	3.036060	13.296628	3.287500	17.959999	3.297500	18.1200	4200
2013-01-07	2.948326	13.104136	3.192500	17.700001	3.295000	17.8600	9900
2013-01-08	2.883680	12.985683	3.122500	17.540001	3.210000	17.7199	2900

```
df_chip.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 2518 entries, 2013-01-02 to 2022-12-30
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   (Adj Close, NVDA)  2518 non-null   float64
 1   (Adj Close, TSM)   2518 non-null   float64
 2   (Close, NVDA)      2518 non-null   float64
 3   (Close, TSM)       2518 non-null   float64
 4   (High, NVDA)       2518 non-null   float64
 5   (High, TSM)        2518 non-null   float64
 6   (Low, NVDA)        2518 non-null   float64
 7   (Low, TSM)         2518 non-null   float64
 8   (Open, NVDA)       2518 non-null   float64
 9   (Open, TSM)        2518 non-null   float64
 10  (Volume, NVDA)     2518 non-null   int64  
 11  (Volume, TSM)      2518 non-null   int64  
dtypes: float64(10), int64(2)
memory usage: 255.7 KB
```

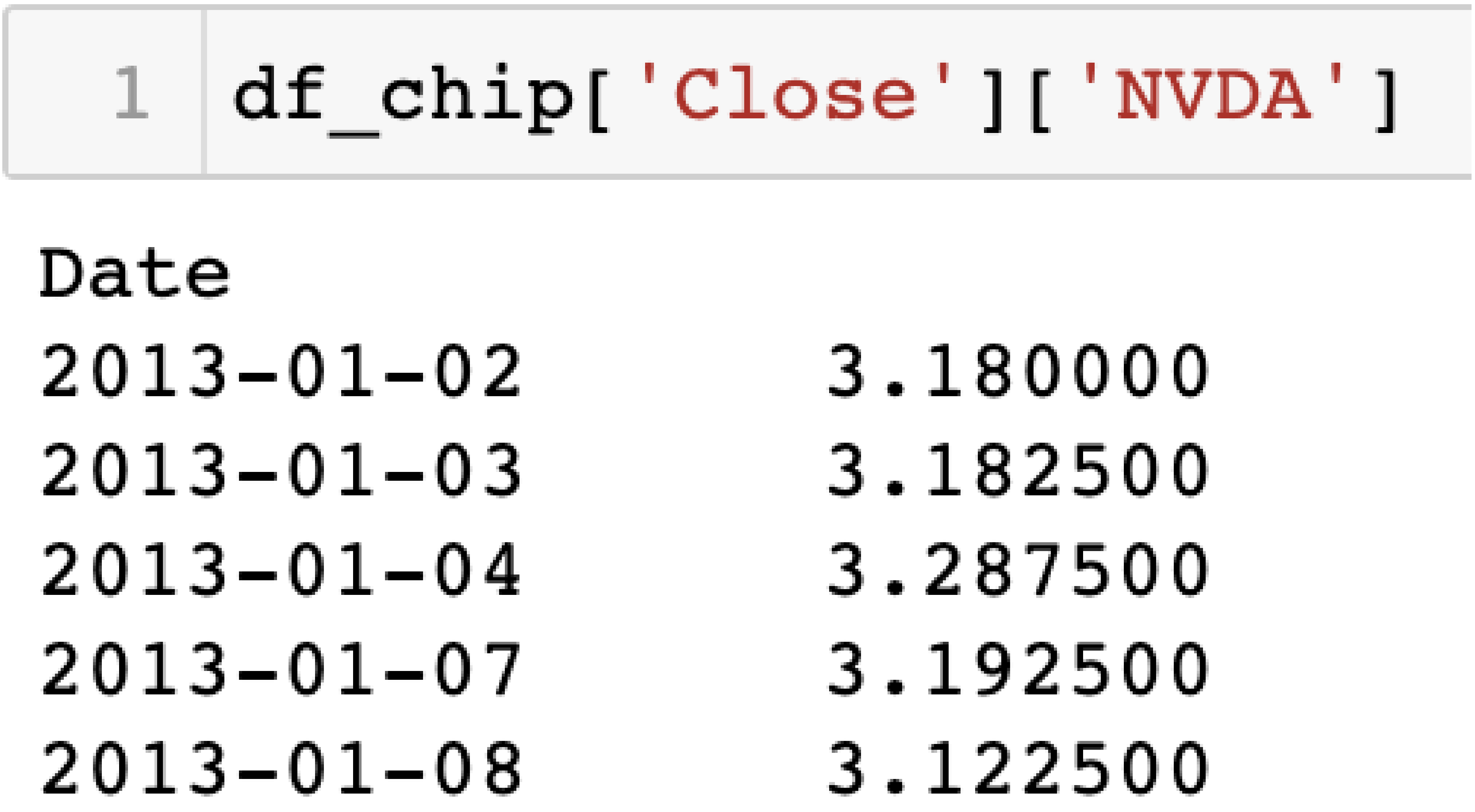
Access multiple series with a common column name

Access specific series like these.



```python
df_chip['Close'][['NVDA','TSM']]
```

Date	NVDA	TSM
2013-01-02	3.180000	18.100000
2013-01-03	3.182500	18.090000
2013-01-04	3.287500	17.959999
2013-01-07	3.192500	17.700001
2013-01-08	3.122500	17.540001



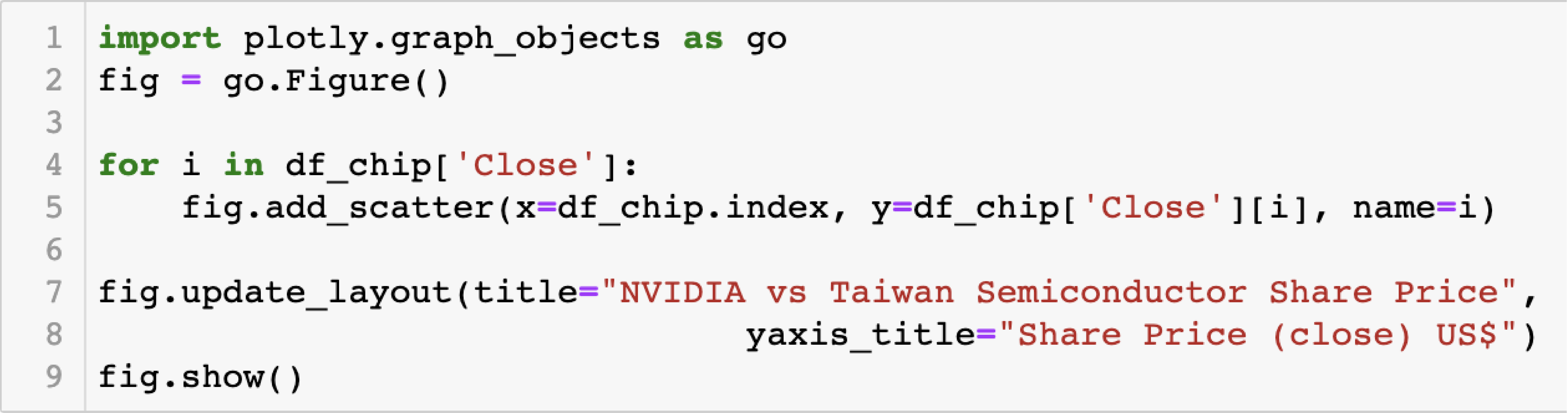
```python
df_chip['Close']['NVDA']
```

```
Date
2013-01-02        3.180000
2013-01-03        3.182500
2013-01-04        3.287500
2013-01-07        3.192500
2013-01-08        3.122500
```

Multiple plots with graph_objects

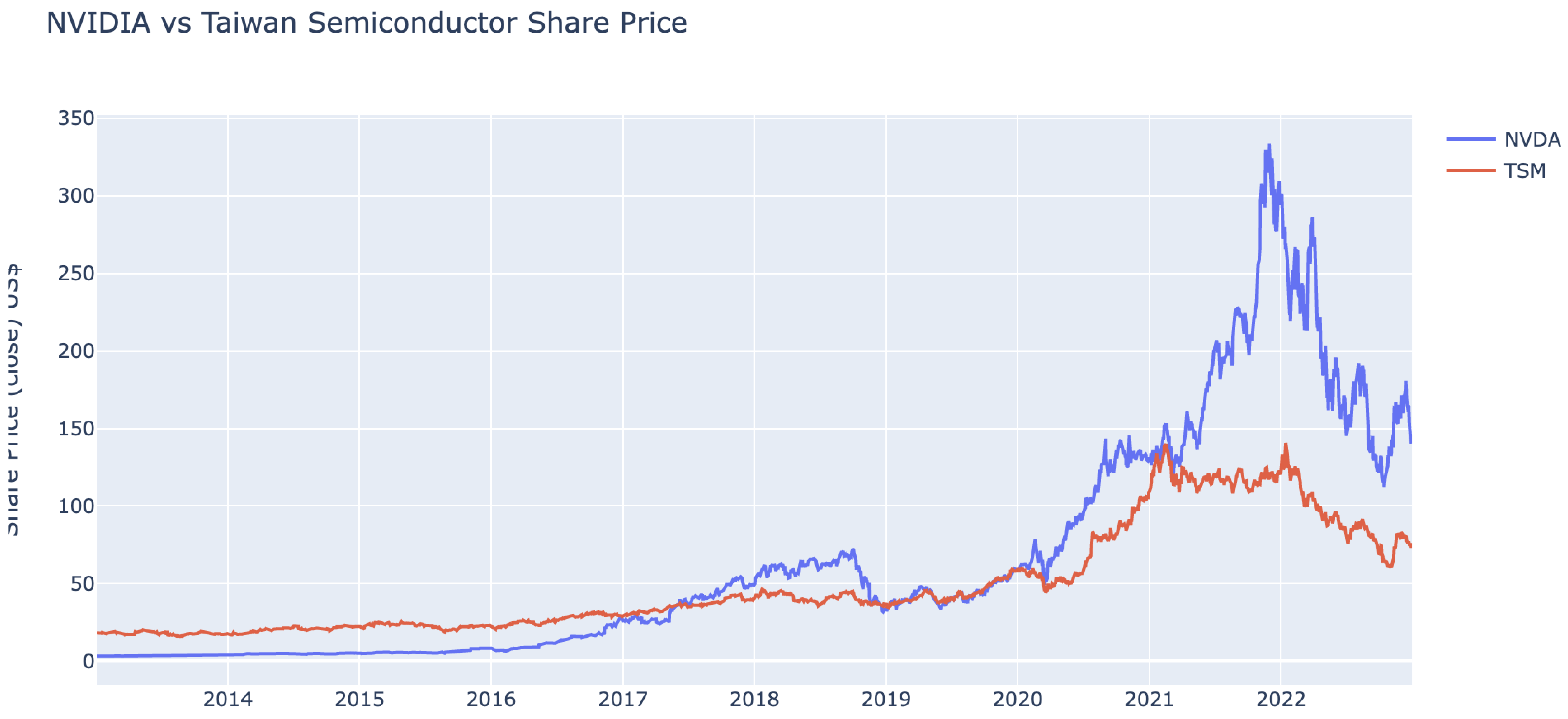
For easier customised setting, use graph_objects library.

Use for loop to plot each stock.



```
import plotly.graph_objects as go
fig = go.Figure()

for i in df_chip['Close']:
    fig.add_scatter(x=df_chip.index, y=df_chip['Close'][i], name=i)

fig.update_layout(title="NVIDIA vs Taiwan Semiconductor Share Price",
                                yaxis_title="Share Price (close) US$")
fig.show()
```

Multiple plots with graph_objects

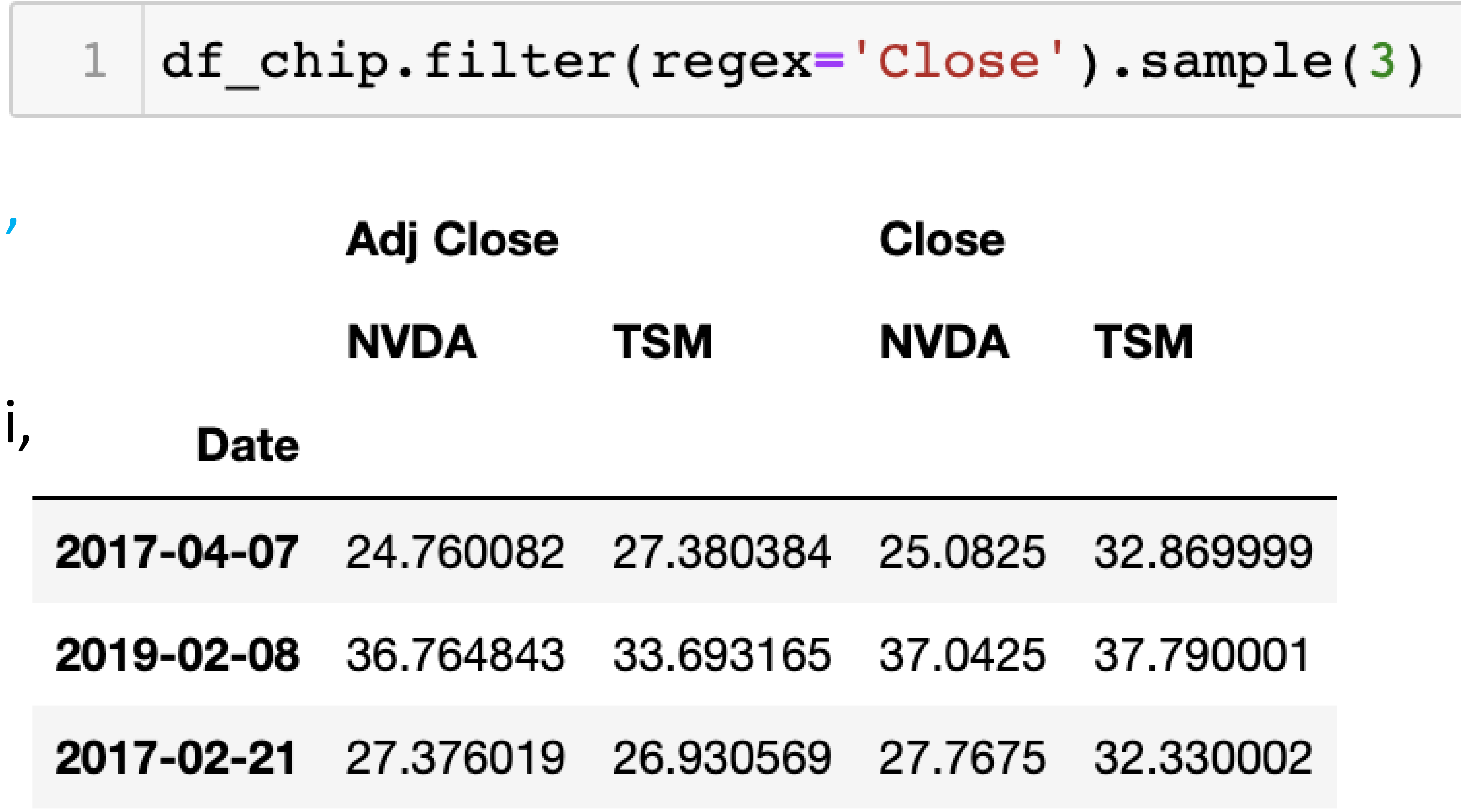


Close and Adj. Close

You may find out that there are two kinds of Close. Adjusted Close is weighted and adjusted price after any stock splits, dividend pay out, increase or decrease number of common stock, etc. Close is the actual market Close.

For analysing value issue, such as how earning and book value affect share price, we usually use Adj. Close.

For technical analysing, especially taken Open, Hi, Low factors into accounts, we tend to use Close. Since there are never adjusted Open, Hi, Low.

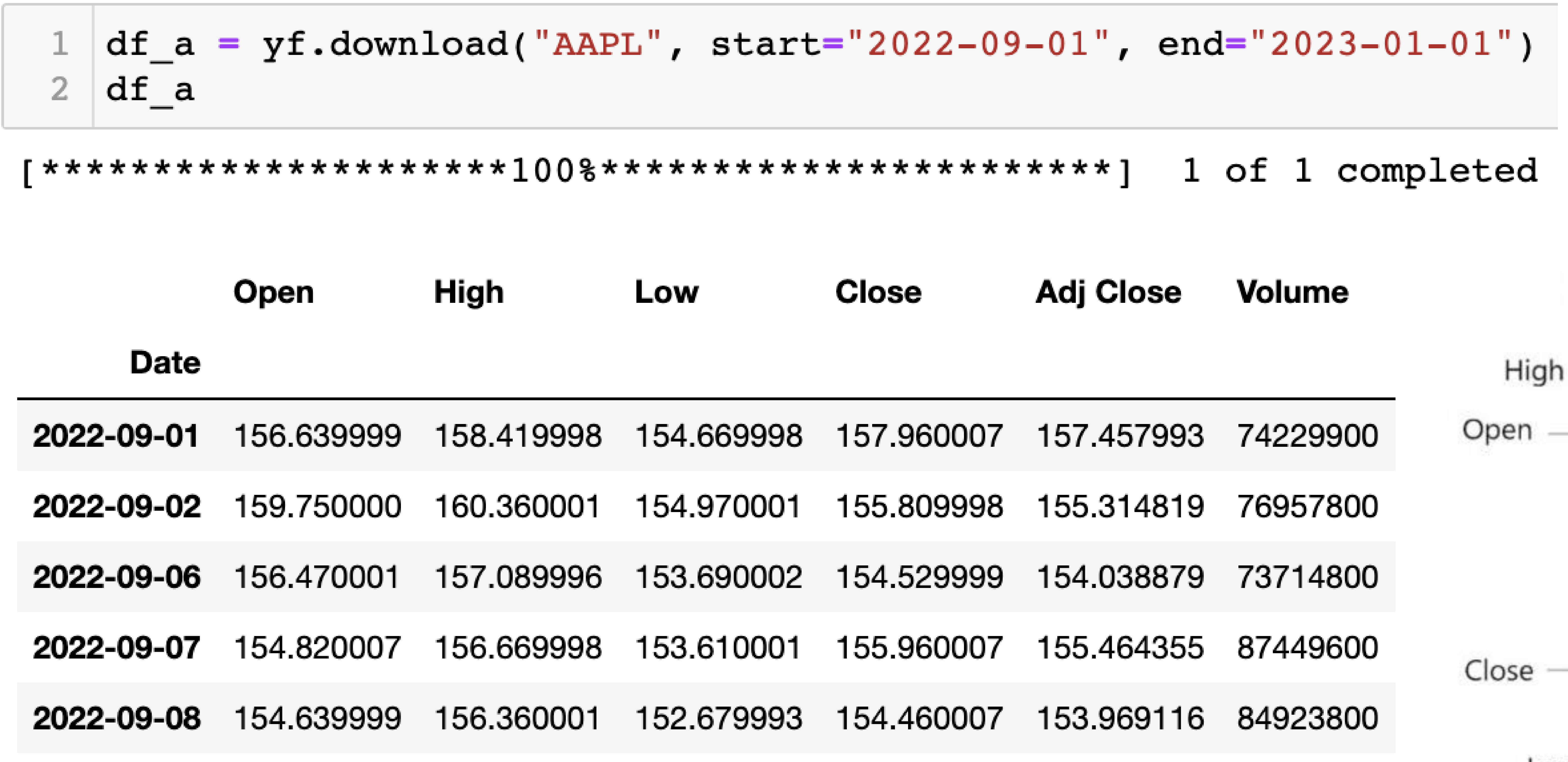


```
df_chip.filter(regex='Close').sample(3)
```

	Adj Close		Close	
	NVDA	TSM	NVDA	TSM
Date				
2017-04-07	24.760082	27.380384	25.0825	32.869999
2019-02-08	36.764843	33.693165	37.0425	37.790001
2017-02-21	27.376019	26.930569	27.7675	32.330002

Candlestick Chart

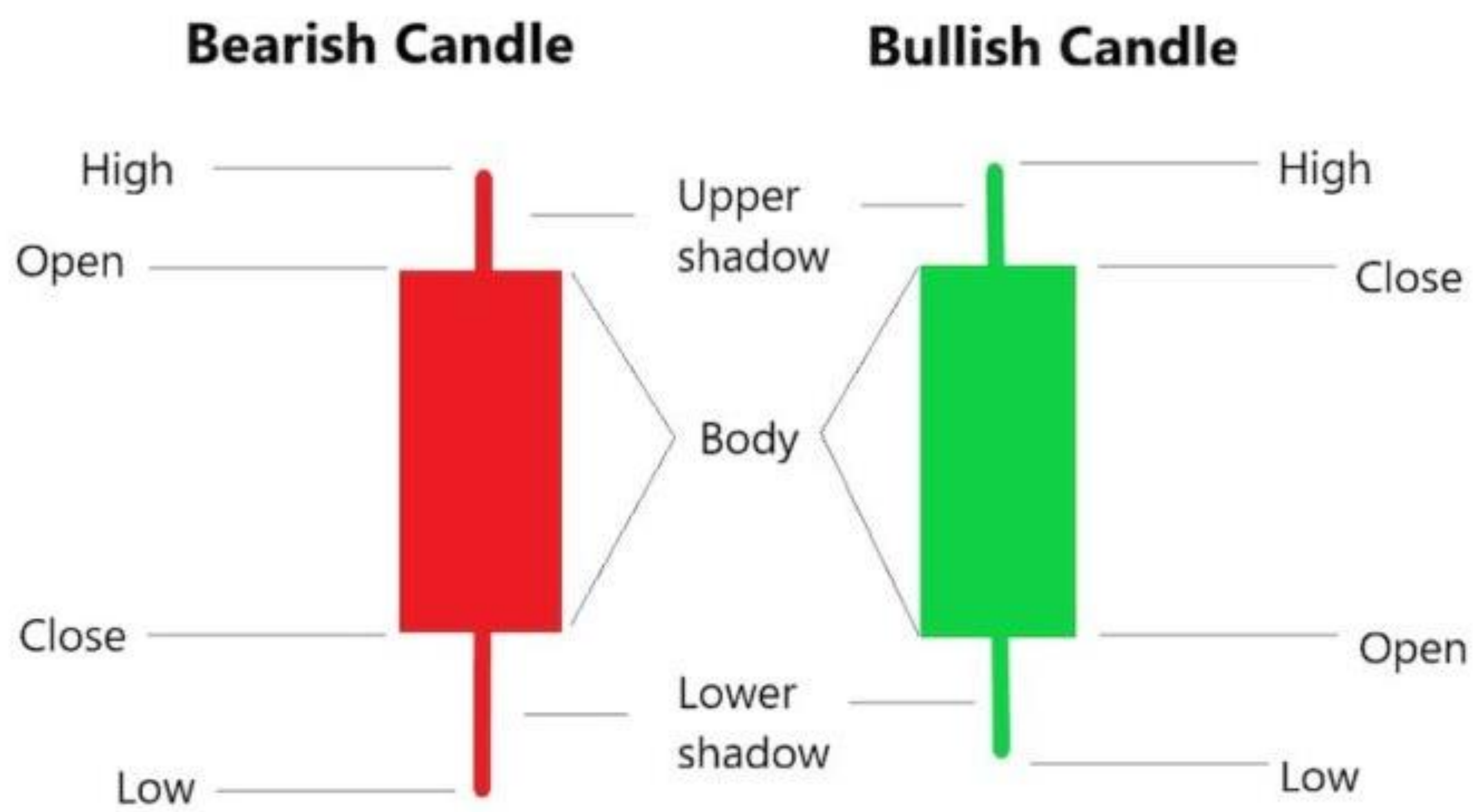
Plotly supports candlestick chart with easy coding.



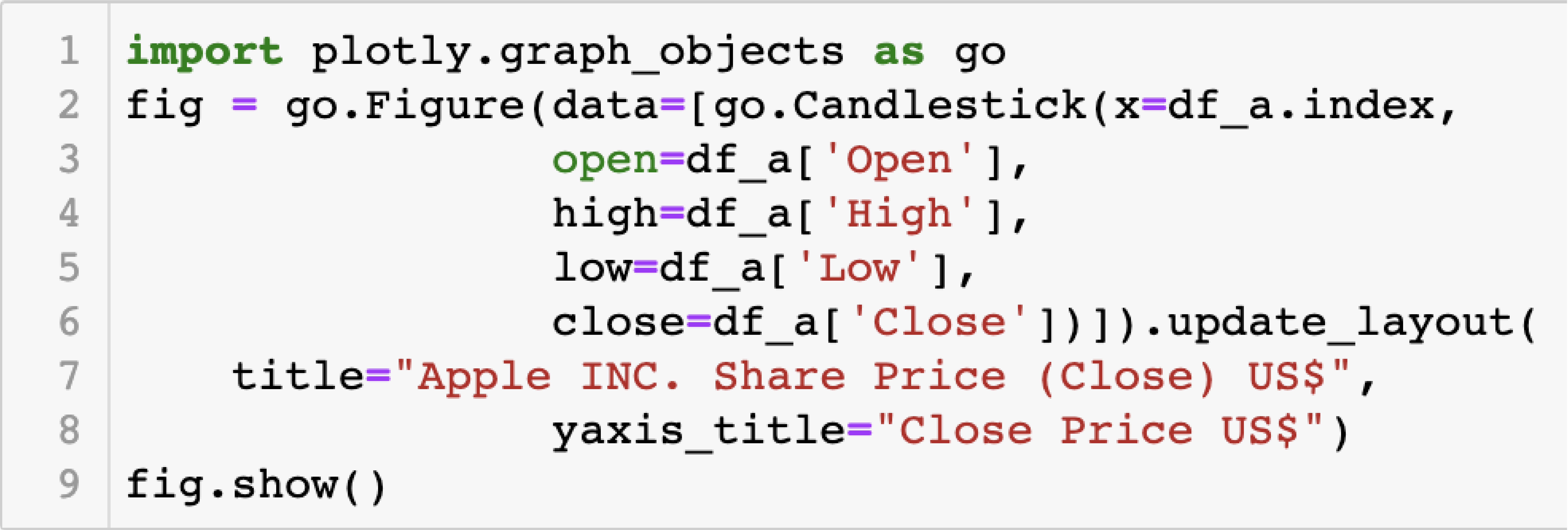
```
df_a = yf.download("AAPL", start="2022-09-01", end="2023-01-01")
df_a
```

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

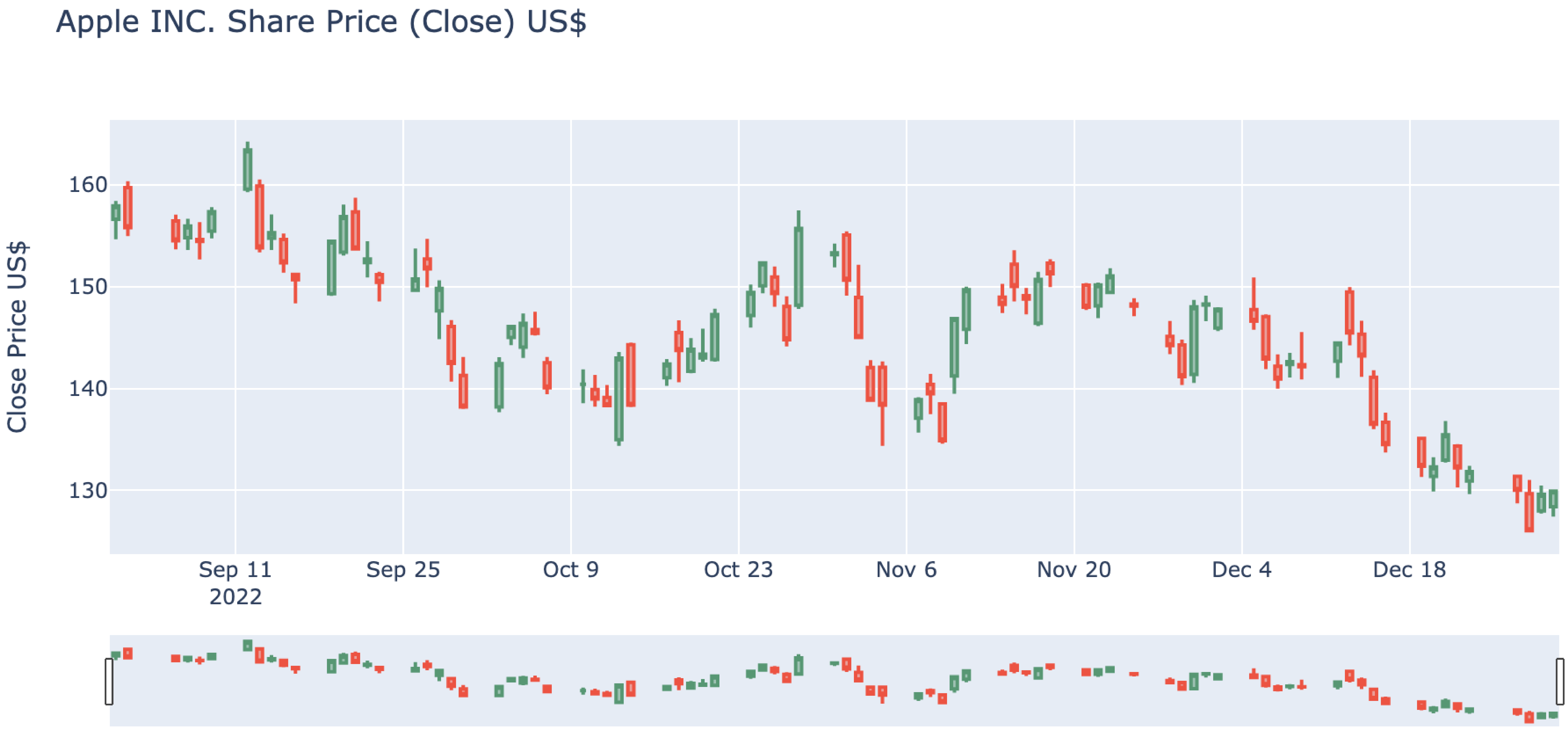
	Open	High	Low	Close	Adj Close	Volume
Date						
2022-09-01	156.639999	158.419998	154.669998	157.960007	157.457993	74229900
2022-09-02	159.750000	160.360001	154.970001	155.809998	155.314819	76957800
2022-09-06	156.470001	157.089996	153.690002	154.529999	154.038879	73714800
2022-09-07	154.820007	156.669998	153.610001	155.960007	155.464355	87449600
2022-09-08	154.639999	156.360001	152.679993	154.460007	153.969116	84923800



Candlestick Chart

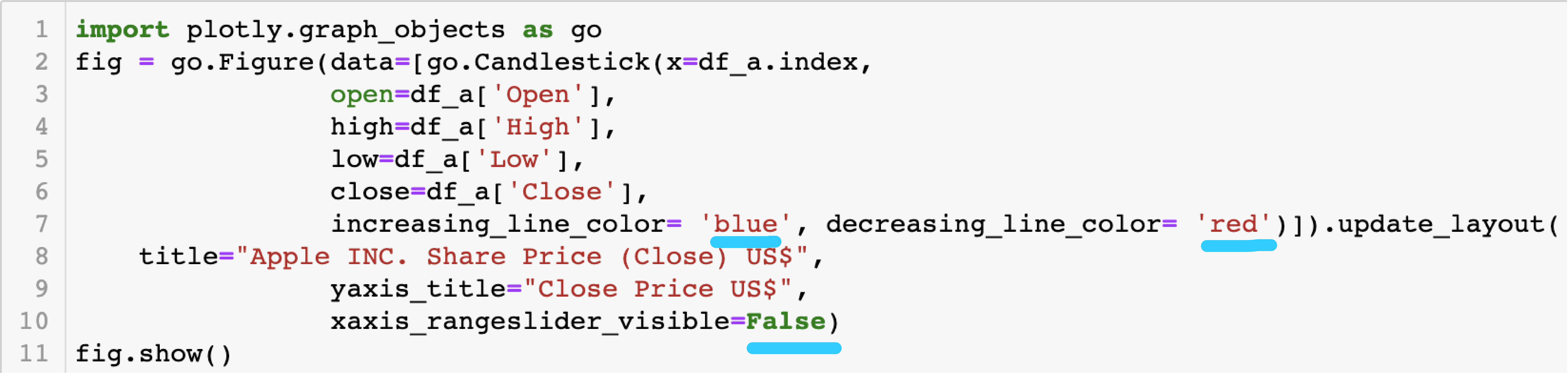


```python
import plotly.graph_objects as go
fig = go.Figure(data=[go.Candlestick(x=df_a.index,
                open=df_a['Open'],
                high=df_a['High'],
                low=df_a['Low'],
                close=df_a['Close'])]).update_layout(
    title="Apple INC. Share Price (Close) US$",
                yaxis_title="Close Price US$")
fig.show()
```


Apple INC. Share Price (Close) US$
Close Price US$
160
150
140
130
Sep 11
2022
Sep 25
Oct 9
Oct 23
Nov 6
Nov 20
Dec 4
Dec 18

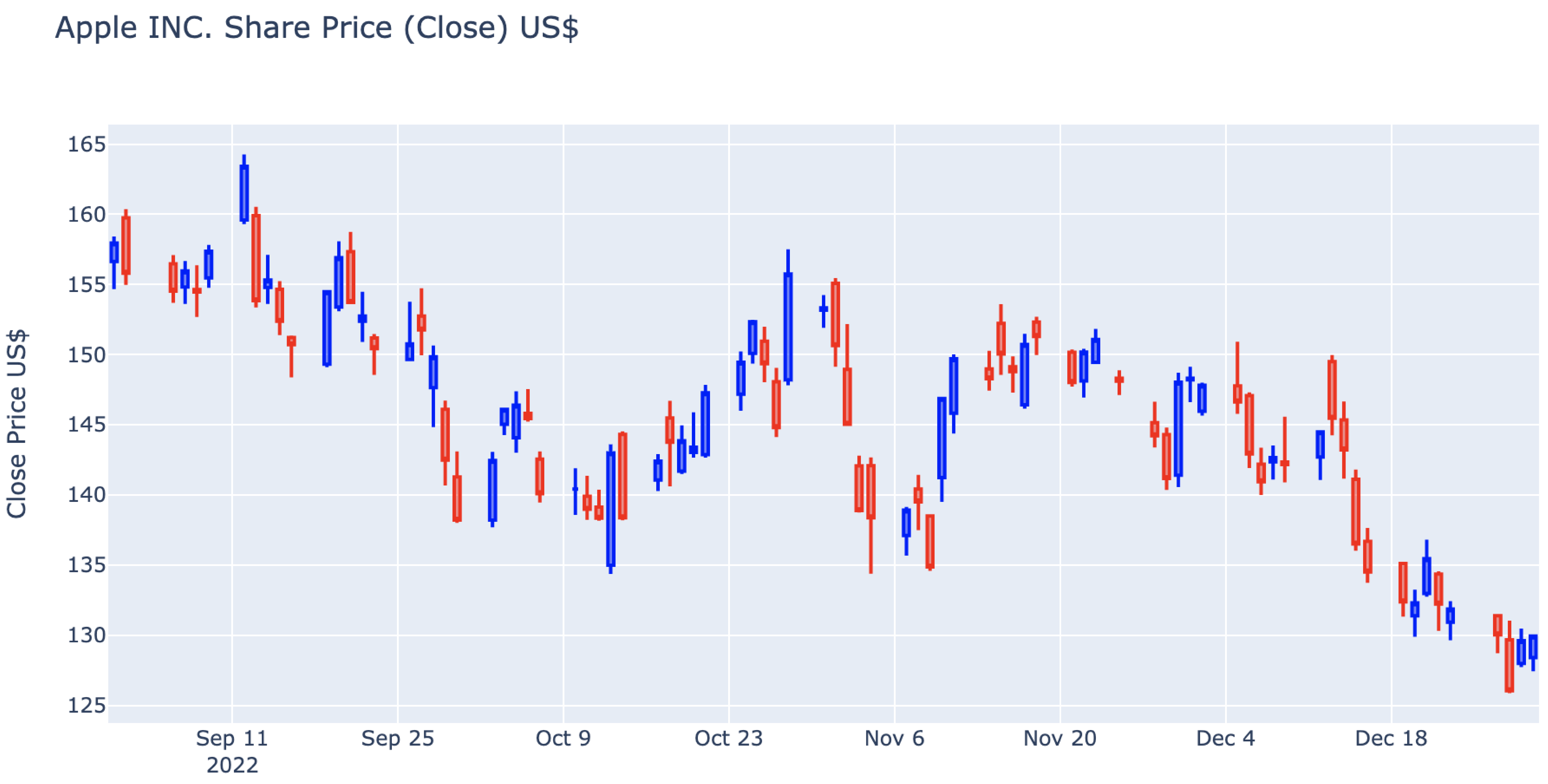
Change candle color and disable rangeslider

If we want to change the default plot setting, we could define as follow.



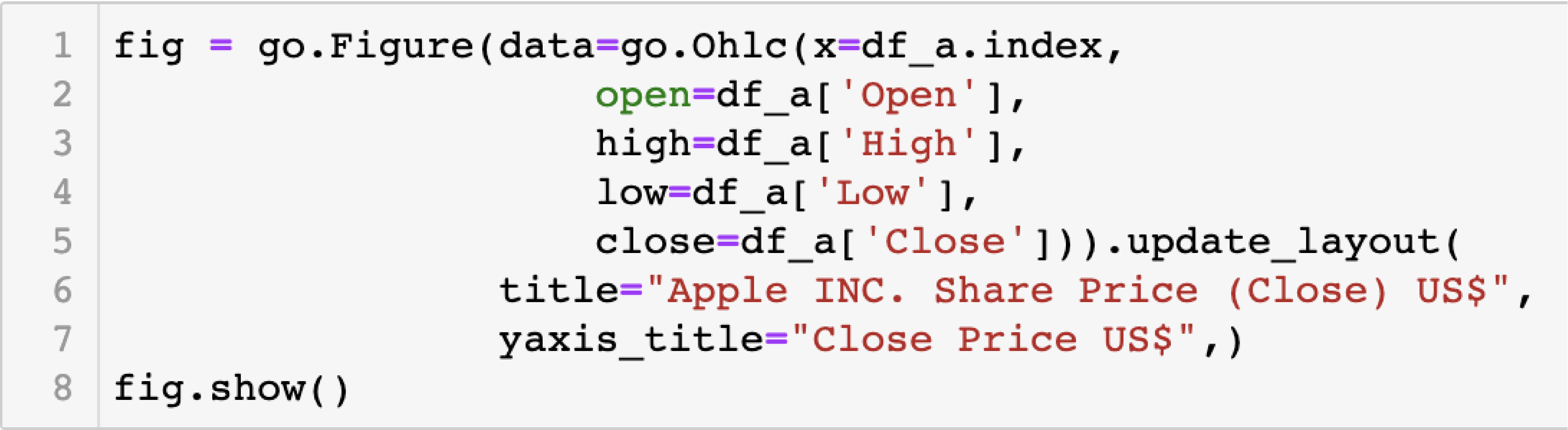
```
import plotly.graph_objects as go
fig = go.Figure(data=[go.Candlestick(x=df_a.index,
                open=df_a['Open'],
                high=df_a['High'],
                low=df_a['Low'],
                close=df_a['Close'],
                increasing_line_color= 'blue', decreasing_line_color= 'red')]).update_layout(
    title="Apple INC. Share Price (Close) US$",
                yaxis_title="Close Price US$",
                xaxis_rangeslider_visible=False)
fig.show()
```

Change candle color and disable rangeslider

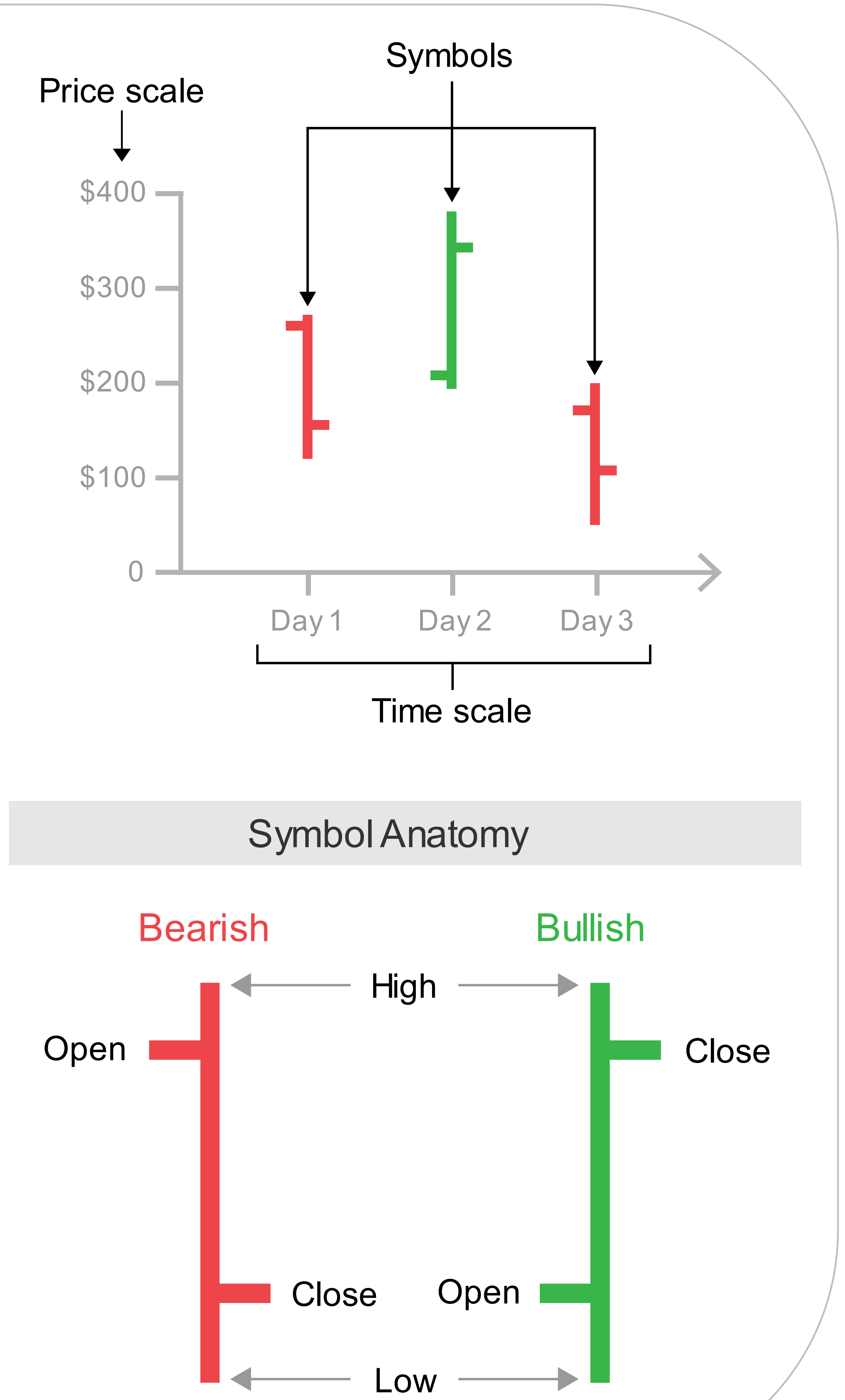


OHLC chart

OHLC (open high low close) chart is similar to candlestick chart.



```
fig = go.Figure(data=go.Ohlc(x=df_a.index,
                    open=df_a['Open'],
                    high=df_a['High'],
                    low=df_a['Low'],
                    close=df_a['Close'])).update_layout(
                title="Apple INC. Share Price (Close) US$",
                yaxis_title="Close Price US$",)
fig.show()
```

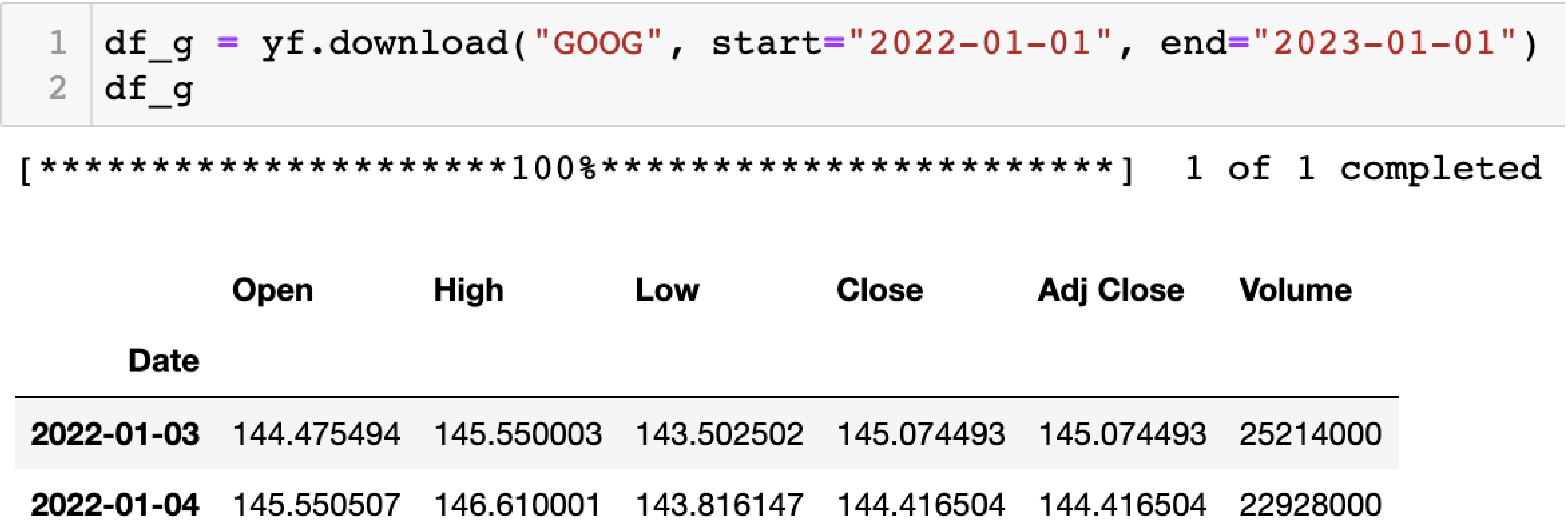


OHLC chart

Apple INC. Share Price (Close) US$
Close Price US$
160
150
140
130
Sep 11
2022
Sep 25
Oct 9
Oct 23
Nov 6
Nov 20
Dec 4
Dec 18

Moving Average and Exponential Moving Average

Moving Average (MA) sometimes mentioned as Simple Moving Average (SMA), is probably the most frequently used indicator. As well as Exponential Moving Average (EMA), which is sensitive to recent share price.



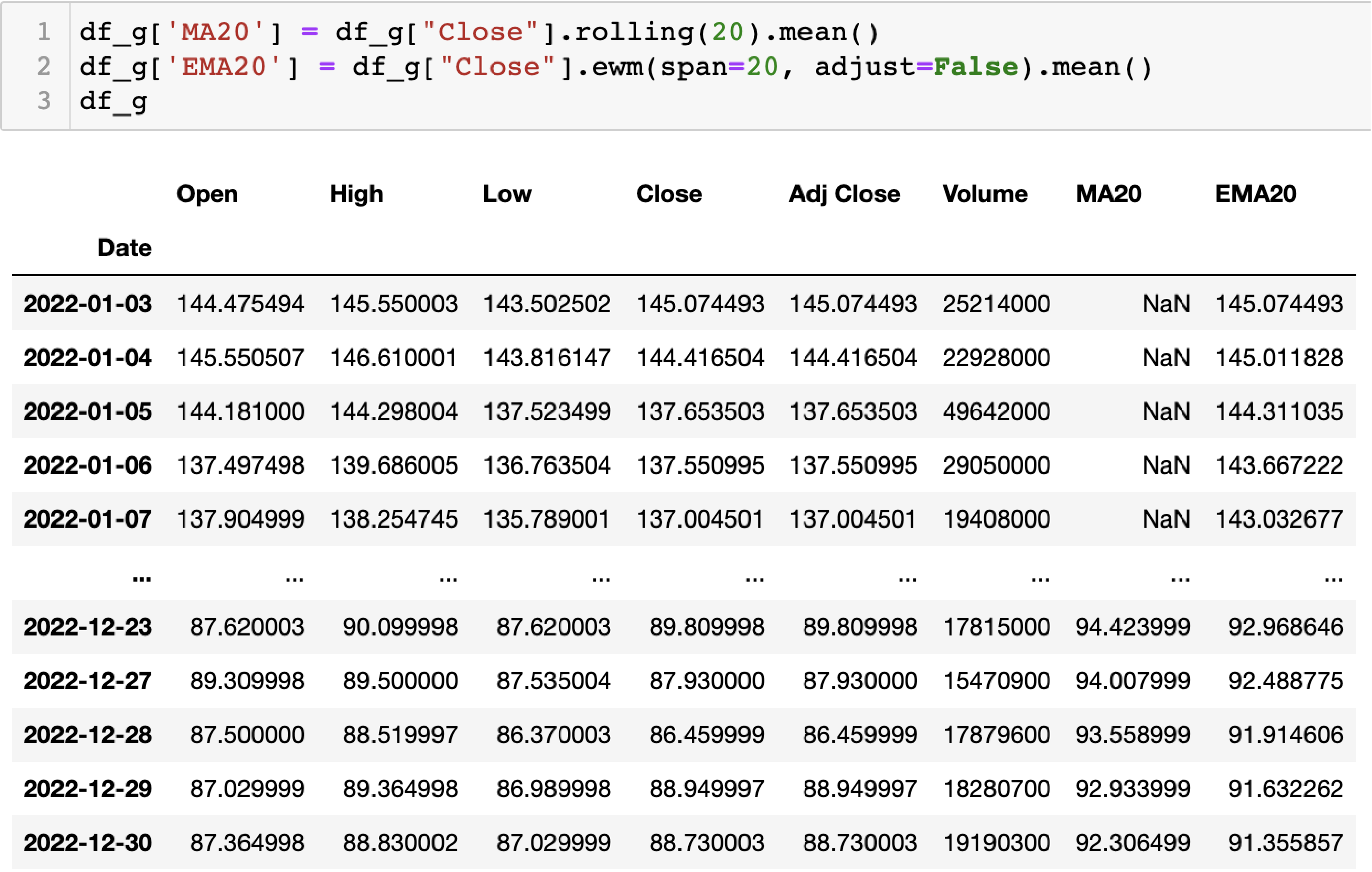
```python
df_g = yf.download("GOOG", start="2022-01-01", end="2023-01-01")
df_g
```

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

Date	Open	High	Low	Close	Adj Close	Volume
2022-01-03	144.475494	145.550003	143.502502	145.074493	145.074493	25214000
2022-01-04	145.550507	146.610001	143.816147	144.416504	144.416504	22928000

Create MA20 and EMA20 series

Create 20 days MA and EMA columns. Please note there are 20 NaN for MA20, for data analysis as a whole, we should drop all the NaN rows. If you MA20 on 2022-01-03, you should pull data from around 1 months earlier.

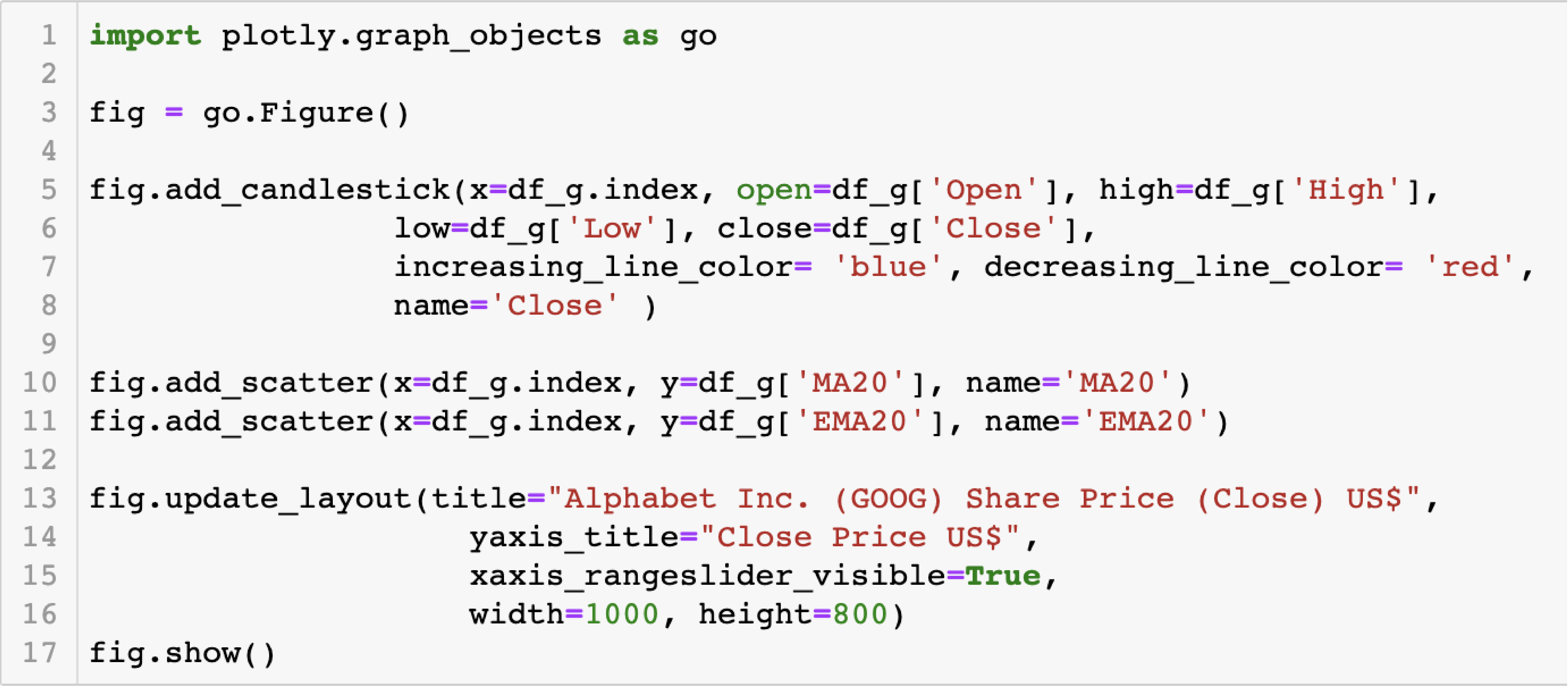


```
df_g['MA20'] = df_g["Close"].rolling(20).mean()
df_g['EMA20'] = df_g["Close"].ewm(span=20, adjust=False).mean()
df_g
```

	Open	High	Low	Close	Adj Close	Volume	MA20	EMA20
Date								
2022-01-03	144.475494	145.550003	143.502502	145.074493	145.074493	25214000	NaN	145.074493
2022-01-04	145.550507	146.610001	143.816147	144.416504	144.416504	22928000	NaN	145.011828
2022-01-05	144.181000	144.298004	137.523499	137.653503	137.653503	49642000	NaN	144.311035
2022-01-06	137.497498	139.686005	136.763504	137.550995	137.550995	29050000	NaN	143.667222
2022-01-07	137.904999	138.254745	135.789001	137.004501	137.004501	19408000	NaN	143.032677
...	...	...	...	...	...	...	...	...
2022-12-23	87.620003	90.099998	87.620003	89.809998	89.809998	17815000	94.423999	92.968646
2022-12-27	89.309998	89.500000	87.535004	87.930000	87.930000	15470900	94.007999	92.488775
2022-12-28	87.500000	88.519997	86.370003	86.459999	86.459999	17879600	93.558999	91.914606
2022-12-29	87.029999	89.364998	86.989998	88.949997	88.949997	18280700	92.933999	91.632262
2022-12-30	87.364998	88.830002	87.029999	88.730003	88.730003	19190300	92.306499	91.355857

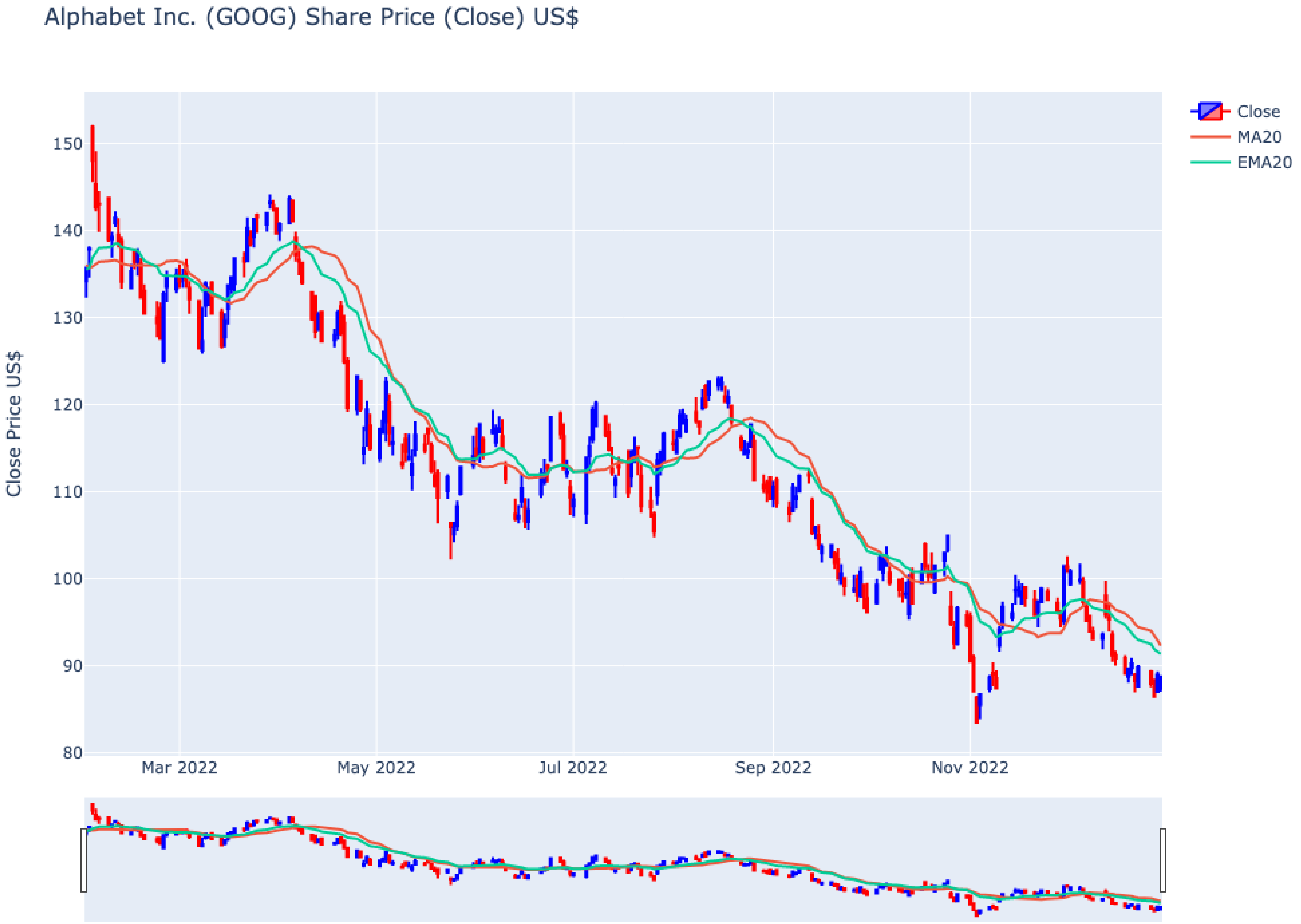
Candlestick close with MA20 & EMA20 line

```python
df_g = df_g.dropna()
```



```python
import plotly.graph_objects as go

fig = go.Figure()

fig.add_candlestick(x=df_g.index, open=df_g['Open'], high=df_g['High'],
                    low=df_g['Low'], close=df_g['Close'],
                    increasing_line_color= 'blue', decreasing_line_color= 'red',
                    name='Close' )

fig.add_scatter(x=df_g.index, y=df_g['MA20'], name='MA20')
fig.add_scatter(x=df_g.index, y=df_g['EMA20'], name='EMA20')

fig.update_layout(title="Alphabet Inc. (GOOG) Share Price (Close) US$",
                        yaxis_title="Close Price US$",
                        xaxis_rangeslider_visible=True,
                        width=1000, height=800)
fig.show()
```

Candlestick close with MA20 & EMA20 line



Critics on MA and EMA

Moving Average data is also used in financial accounting and marketing analysis. It is a simple way to compare historic data in certain period.

Exponential Moving Average is sensitive to recent data, therefore the average period should not be long, otherwise it is meaningless.

Although these two methods are used for prediction often by so-called columnist, it is not statistical way unless you can prove 95% correctness of trend in all circumstances.

We may discover this in later chapter.

Chapter Wrap Up

Plotly Graph Object is the library for customised plots.

Before you plot, make sure that data is in Pandas DF(less trouble) and NaN free.

Build your plots step by step and take reference on official documents.

API data format (json) and API pypi library make data more easy to access. No need web scrawling often nowadays.

Reference

Official Website:

https://plotly.com/python/

Plotly Graph Objects:

https://plotly.com/python/graph-objects/

WorldBank API:

- https://blogs.worldbank.org/opendata/introducing-wbgapi-new-python-package-accessing-world-bank-data
- https://nbviewer.org/github/tgherzog/wbgapi/blob/master/examples/wbgapi-cookbook.ipynb
- https://pypi.org/project/wbgapi/

GitHub Open Source Code:

https://github.com/plotly/plotly.py

plotly | Graphing Libraries