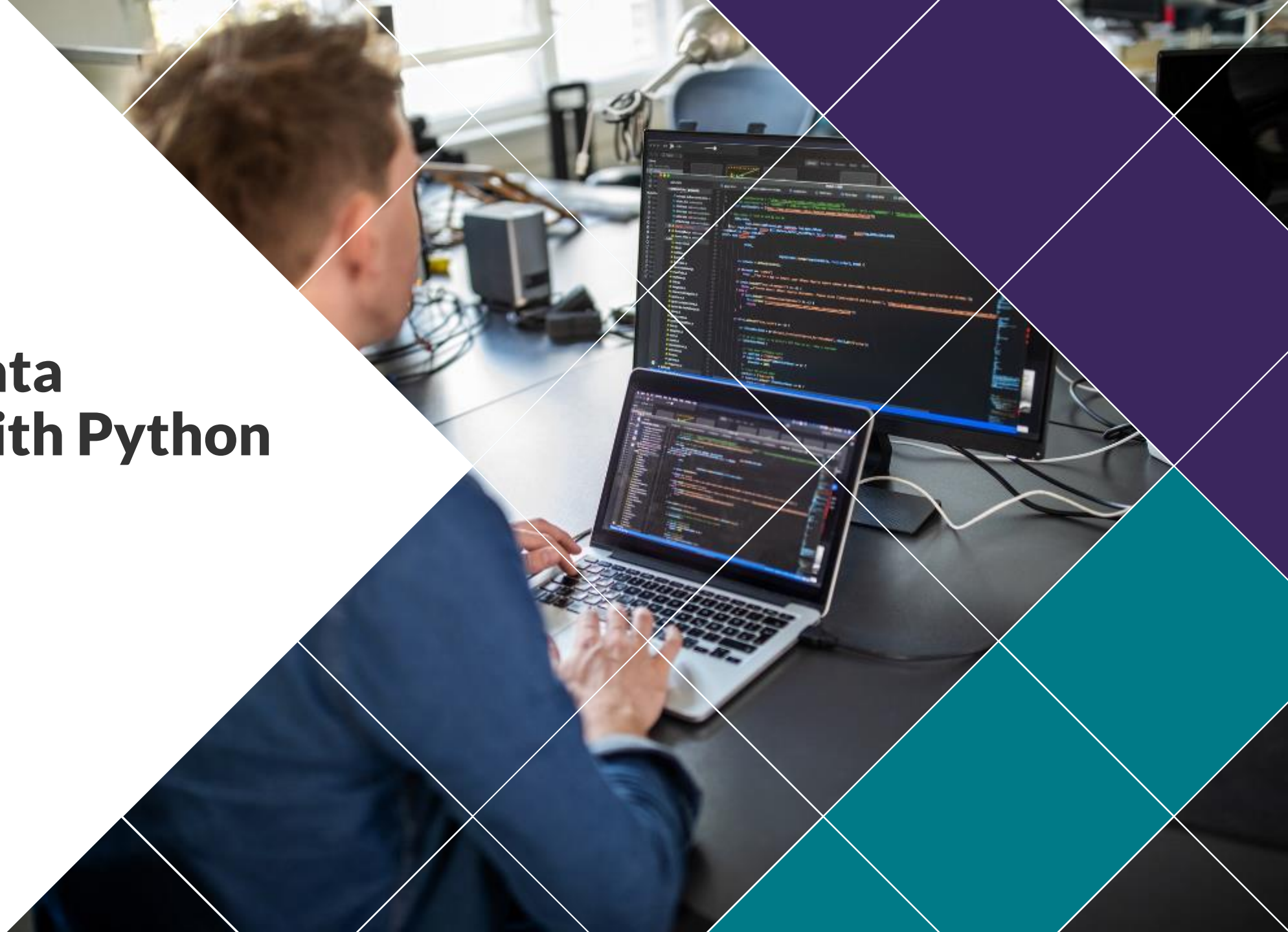
FitchLearning
a FitchSolutions Company

Financial Data Analytics with Python



Introduction to the Python Programming Language

FitchLearning

Outline includes:

- What is Python?
- Why is Python so prevalent?
- Why use Python in Finance?
- What are Jupyter Notebooks?
- Python's Syntax
- Data Types
- Why are Jupyter Notebooks useful?
- Python as a calculator
- Python Lists

What is Python?

- A computer programming language
- Used to build websites, automate tasks, and conduct data analysis
- I.E. A general-purpose language
- Designed as an OOP language
- (Named after the british comedy group monty python)

Why is Python so prevalent/popular?

- Simple (with respect to syntax)
  - Mimics natural language i.e. easy to read and understand (by design!)
- Versatile
  - Used for many different purposes
- Open source
  - It's free to use and distribute
- Supports modules and packages
  - third party user code that is used to expand the base Python language

Why use Python in finance?

- Productivity gains
- Easy to learn by non-programmers
- High-level built-in data structures, combined with dynamic typing and dynamic binding

Distributions of Python

A "collection" of resources i.e. what an end-user will download from the internet and install

- CPython - comes with the official installation and is written in C language
- PyPy
- Anaconda
- ActivePython
- Jython
- IronPython

Each of the implementations has their own features and trade-offs.

Jupyter Notebook

Open-source web application

- Allows you/user to author notebook documents (create & share)
- Contain:
  - Live code
  - Plots / Viz
  - Narrative text
  - Equations

Notebooks consist of three basic cell types

- Code cells
  - Input and output of live code that is run in the kernel
- Markdown cells
  - Narrative text with embedded LaTeX equations
- Raw cells
  - Unformatted text
    - included, without modification, when notebooks are converted to different formats using nbconvert

Narrative text with embedded LaTeX

PV = \frac{{CF_1}}{{(1 + r)^1}} + \frac{{CF_2}}{{(1 + r)^2}} + \ldots + \frac{{CF_n}}{{(1 + r)^n}}

$$PV = \frac{CF_1}{(1+r)^1} + \frac{CF_2}{(1+r)^2} + \ldots + \frac{CF_n}{(1+r)^n}$$

Displaying text

In [1]:
```
print("Kunal, Fitch Learning")
```

```
Kunal, Fitch Learning
```

In [2]:
```
# A 'hash' symbol denotes a comment
```

In [3]:
```
# This is a comment. Anything after the 'hash' symbol on the line is ignored by the Pyth
on interpreter

print("Hello")
print("World")  # commands are interpreted line-by-line
```

```
Hello
World
```

Variable assignment I

- A variable holds a value

```
In [4]: message = "Hello world!"
```

- A variable allows you to refer to a value with a name

```
In [5]: message = "Hello Earth!" # we can change the value of a variable
        print(message)

        Hello Earth!
```

Variable assignment II

- Simple Data Types

```
In [6]: name = "Kunal"
        type(name)

Out[6]: str

In [7]: age = 21
        type(age)

Out[7]: int

In [8]: GBPUSD = 1.19
        type(GBPUSD)

Out[8]: float

In [9]: is_alive = True
        type(is_alive)

Out[9]: bool
```

How do I display the value of variables on my screen?

```
In [10]: print(name)

         Kunal

In [11]: print(GBPUSD)

         1.19
```

- We use the print() function to display/write the value of a variable to the output/screen
  - Ref: https://docs.python.org/3/library/functions.html#print
    - From documentation: non-keyword arguments are converted to strings and written to the stream

Jupyter: display the values without needing print()

```
In [12]: name

Out[12]: 'Kunal'
```

- Jupyter: "print()s" the last line of EVERY cell

```
In [13]: print(GBPUSD)
         name

         1.19

Out[13]: 'Kunal'
```

Jupyter over MS excel, beyond simply display output without explicitly using the print function

- **Interactive** computing environment - write and execute code, visualize data, and preform EDA
- Reproducibility - notebooks can be saved and shared as files or even converted into **different formats**, such as HTML
- Detailed documentation capabilities - **markdown cells** allow you to write formatted text, equations, and include images or links
- Versatile - whilst MS Excel primarily focuses on formulas, Jupyter can leverage the **extensive libraries** and tools available in many languages for data manipulation, machine learning, visualization...
- Integration with **version control** systems like Git, enabling collaboration with others and facilitating the tracking of changes over time
- Integration with data science ecosystem
- **Tailor the environment** to your specific needs and preferences - via extensions such as code formatting, linting, debugging,...

[ASIDE]

Jupyter: Is it possible to amend the following, such that the output from the last line is NOT displayed ?

```
In [14]: age
         name

Out[14]: 'Kunal'
```

End the last line with ;

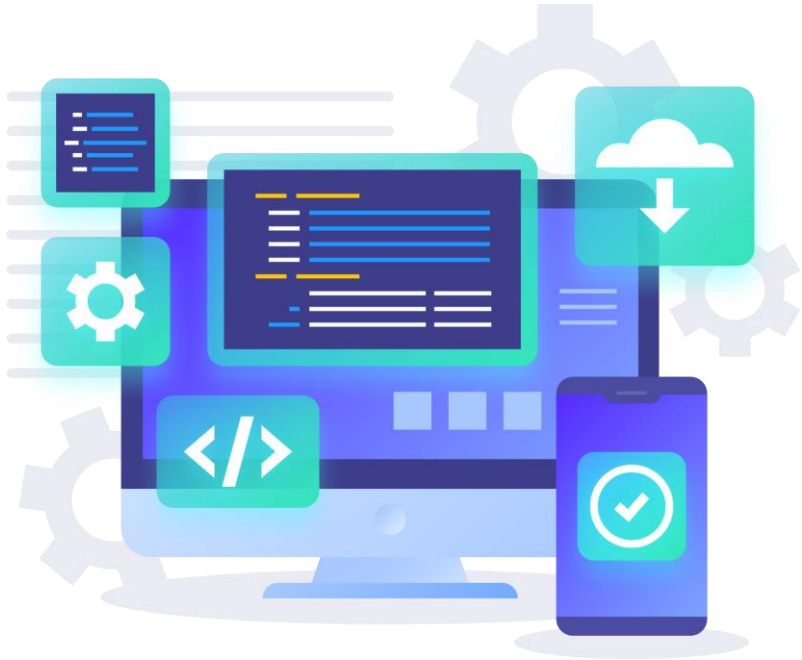
```
In [15]: age
         name;
```

Functions

The core Python programming language provides a small number of built-in functions.
https://docs.python.org/3/library/functions.html

Two that we have used intuitively:

- Print()
- Type()



- Series of statements or blocks of code that can be reused
  - Perform a single related action, i.e. a specific task
    - A function can return a result

Using functions

- e.g. sum()

In [16]:
```
# e.g. define a list of prices
prices = [1.12, 2.34, 4.56, 9.99, 9.99]
```

In [17]:
```
# step 1. name the function
# step 2. followed by parentheses
# step 3. pass prices in
```

Explicit type conversion

- Convert a number into a string

```
In [18]: GBPUSD # reminder from above: type(GBPUSD) => "float" (i.e. floating point number, i.e. decimal)

Out[18]: 1.19

In [19]: str(GBPUSD)

Out[19]: '1.19'
```

Python as a calculator

```
In [20]: 1 + 1 # any Python interpreter can be used as a calculator
Out[20]: 2

In [21]: 1 * 3
Out[21]: 3

In [22]: 100 / 3
Out[22]: 33.333333333333336

In [23]: 100 // 3 # divide 100 into thirds WITHOUT remainder (ie., integer division, aka floor division)
Out[23]: 33

In [24]: 10 ** 3
Out[24]: 1000
```

Python as a calculator

- Practice using base Python operators, such as assignment (=), addition (+), multiplication (*), and division (/)

Example

- You invest $20,000 in a debt instrument
  - Term = 5 years
  - Assume annual compounding
  - Annual interest rate is 4%

Q: How much do you receive in 5 years?

Python as a calculator

In [26]:
```
# Solution

principal = 20000 # USD
rate = 4 # %
t = 5 # years
n = 1 # compounded annually
```

In [27]:
```
# Q: How much do you receive in 5 years?

FV = principal * (1+(rate/100)/n) ** (n*t)
FV
```

Out[27]:
```
24333.058048000003
```

Complex data types

aka: collections of Simple Data Types

- **List**
- Dictionary
- Tuple
- Set

Lists

```
In [28]: tickers = ['GOOG', 'AMZN', 'AAPL', 'MSFT'] # a list is created with "[" and "]"

In [29]: tickers

Out[29]: ['GOOG', 'AMZN', 'AAPL', 'MSFT']
```

- Groups of data (collections)
- Ordered
- Mutable
  - Editable
  - Expandable

Lists

- Element Access

```
In [30]: tickers

Out[30]: ['GOOG', 'AMZN', 'AAPL', 'MSFT']

In [31]: tickers[0] # FIRST element

Out[31]: 'GOOG'

In [32]: tickers[1] # second element

Out[32]: 'AMZN'

In [33]: tickers[2] # 3rd element

Out[33]: 'AAPL'

In [34]: tickers[-1] # LAST element

Out[34]: 'MSFT'
```

Lists

- Mutable

```
In [35]: tickers[-2] # penultimate element

Out[35]: 'AAPL'

In [36]: tickers[-2] = "anything" # update by variable assignment

In [37]: tickers # editable

Out[37]: ['GOOG', 'AMZN', 'anything', 'MSFT']

In [38]: tickers.append("META") # uses a method

In [39]: tickers # expandable

Out[39]: ['GOOG', 'AMZN', 'anything', 'MSFT', 'META']
```

Lists

In [40]:
```
# NB: objects do not have to be the same type
my_list = ['GOOG', 999, 'AMZN', 9.11, 'AAPL', "we can mutliple words"]
```

Multiple elements?

- From index 0 until index 2? i.e. a range of items

In [41]:
```
tickers # reminder
```

Out[41]:
```
['GOOG', 'AMZN', 'anything', 'MSFT', 'META']
```

In [42]:
```
tickers[  0:2  ] # from the first element to the second

# indexing starts at ZERO and the last element is not included
# i.e. inclusive of first index, exclusive of second index
```

Out[42]:
```
['GOOG', 'AMZN']
```

Lists

```
In [43]: tickers[1:3] # from the second element to the third

Out[43]: ['AMZN', 'anything']

In [44]: tickers[0:3]  # from the beginning

Out[44]: ['GOOG', 'AMZN', 'anything']

In [45]: tickers[ :3]  # from the beginning

Out[45]: ['GOOG', 'AMZN', 'anything']

In [46]: tickers[3: ]  # to the end

Out[46]: ['MSFT', 'META']

In [47]: tickers[3:-1]  # to the end (exclusive of the element at the last index)

Out[47]: ['MSFT']
```

Introduction to Data Analysis with Python

FitchLearning

Outline includes:

- Extend base Python by utilizing data analytical libraries
- The Pandas library
- How to read in CSV data
- Data Structure: Series vs DataFrames
- Sub setting a DataFrame
- Generate descriptive statistics
- Visualize time-series data utilizing the Matplotlib library

Packages

- Most of the high-level functions you will want for data analysis are NOT built-in
- A package contains functions that can be shared with other users
- To utilize packages, we need to first import them into your current programing environment

```
In [48]: import statistics # utilise a library -- provides functions for statistical calculations
```

Ref: https://docs.python.org/3/library/statistics.html

Calling/using functions within a library

- Period(.) to call

```
In [49]: statistics.mean(prices)

Out[49]: 5.6

In [50]: statistics.harmonic_mean(prices)

Out[50]: 2.874048807070016
```

Utilizing aliases

```
In [51]: import statistics as stats

In [52]: stats.mean(prices)

Out[52]: 5.6

In [53]: stats.harmonic_mean(prices)

Out[53]: 2.874048807070016
```

Python is inefficient

Python lists:

- **Flexible:** can contain ANY types of objects
- **Versatile:** items can be added into a list; and items can be removed
- **NOT** suitable for storing and computing **LARGE** amount of numbers

Because INEFFICIENT use of cache

- Refreshes cache a lot
  - (NB: cache = internal temporary storage area inside the CPU itself)
  - Interpreted language (converted into bytecode that is then executed)
  - (NB: compiled language {e.g. C and C++} - code is built and linked)

Python inefficiencies - overcome via packages

- NumPy and SciPy – Fundamental Scientific Computing
- pandas – Data Manipulation and Analysis
- Matplotlib – Plotting and Visualisation
- scikit-learn – Machine Learning and Data Mining
- statsmodels – Statistical Modeling, Testing, and Analysis
- seaborn – Statistical Data Visualisation
- Plotly - Interactive Visualsations

Implemented in FASTER languages (such as C or Fortran)

– compiled C code --> executes very quickly (& scales better as data size increases)

What is data analysis?

- The process of collecting data and then processing that data into useful information (aka insights)
- The insights can be used to make data driven decisions

Pandas

Python library for data analysis

Exploration

Cleaning

Transformation

Ref: http://pandas.pydata.org/pandas-docs/stable/

Import libraries

```
In [54]: import pandas as pd # using community agreed alias
```

- Question: Are aliases useful?
- Answer: yes, easier to reference the package downstream in your code
- Question: Does the name of this package refer to a 🐼 ?
- Answer: No, it refers to PANel DAta (i.e. multidimensional data)

Python Inefficiencies - overcome via packages

- NumPy and SciPy – Fundamental Scientific Computing
- Pandas – Data Manipulation and Analysis

```
import numpy as np # using community agreed alias
```

```
import pandas as pd # using community agreed alias
```

Read data into a DataFrame

In [56]:
```
df = pd.read_csv("https://raw.githubusercontent.com/matplotlib/sample_data/master/aapl.csv")
```

In [57]:
```
df = pd.read_csv("aapl.csv") # assumes the file is saved in the same location as this notebook
```

In [58]:
```
df
```

Out[58]:

	Date	Open	High	Low	Close	Volume	Adj Close
0	2008-10-14	116.26	116.40	103.14	104.08	70749800	104.08
1	2008-10-13	104.55	110.53	101.02	110.26	54967000	110.26
2	2008-10-10	85.70	100.00	85.00	96.80	79260700	96.80
3	2008-10-09	93.35	95.80	86.60	88.74	57763700	88.74
4	2008-10-08	85.91	96.33	85.68	89.79	78847900	89.79
...	...	...	...	...	...	...	...
6076	1984-09-13	27.50	27.62	27.50	27.50	7429600	3.14
6077	1984-09-12	26.87	27.00	26.12	26.12	4773600	2.98
6078	1984-09-11	26.62	27.37	26.62	26.87	5444000	3.07
6079	1984-09-10	26.50	26.62	25.87	26.37	2346400	3.01
6080	1984-09-07	26.50	26.87	26.25	26.50	2981600	3.02

6081 rows × 7 columns

DataFrame

- Considered as a "table" of data (i.e. a two-dimensional data structure)
- An object for storing related columns of data
- Each column in a DataFrame is a Series (i.e. a one-dimensional data structure)

```
In [59]: df.head() # a method is a function that "belongs to"/"associated with" an object
```

Out[59]:

	Date	Open	High	Low	Close	Volume	Adj Close
0	2008-10-14	116.26	116.40	103.14	104.08	70749800	104.08
1	2008-10-13	104.55	110.53	101.02	110.26	54967000	110.26
2	2008-10-10	85.70	100.00	85.00	96.80	79260700	96.80
3	2008-10-09	93.35	95.80	86.60	88.74	57763700	88.74
4	2008-10-08	85.91	96.33	85.68	89.79	78847900	89.79

- https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.head.html
  - Return the first n rows, default 5

DataFrame

```
In [60]: df.tail()
```

Out[60]:

	Date	Open	High	Low	Close	Volume	Adj Close
6076	1984-09-13	27.50	27.62	27.50	27.50	7429600	3.14
6077	1984-09-12	26.87	27.00	26.12	26.12	4773600	2.98
6078	1984-09-11	26.62	27.37	26.62	26.87	5444000	3.07
6079	1984-09-10	26.50	26.62	25.87	26.37	2346400	3.01
6080	1984-09-07	26.50	26.87	26.25	26.50	2981600	3.02

- Https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.tail.html
  - Return the last n rows, default 5

DataFrame

In [61]:
```
df[ ['Open', 'High',  'Close'] ] # select mulitple columns using a list of column names
```

Out[61]:

	Open	High	Close
0	116.26	116.40	104.08
1	104.55	110.53	110.26
2	85.70	100.00	96.80
3	93.35	95.80	88.74
4	85.91	96.33	89.79
...	...	...	...
6076	27.50	27.62	27.50
6077	26.87	27.00	26.12
6078	26.62	27.37	26.87
6079	26.50	26.62	26.37
6080	26.50	26.87	26.50

6081 rows × 3 columns

DataFrame

```
In [62]: df[ 'Open' ] # each column in a DataFrame is a Series

Out[62]: 0       116.26
         1       104.55
         2        85.70
         3        93.35
         4        85.91
                  ...
         6076     27.50
         6077     26.87
         6078     26.62
         6079     26.50
         6080     26.50
         Name: Open, Length: 6081, dtype: float64

In [63]: type(df[ 'Open' ])

Out[63]: pandas.core.series.Series
```

Set the index to the "Date" column

In [64]:
```
# via a method
df = df.set_index("Date")
```

In [65]:
```
df.head()
```

Out[65]:

Date	Open	High	Low	Close	Volume	Adj Close
2008-10-14	116.26	116.40	103.14	104.08	70749800	104.08
2008-10-13	104.55	110.53	101.02	110.26	54967000	110.26
2008-10-10	85.70	100.00	85.00	96.80	79260700	96.80
2008-10-09	93.35	95.80	86.60	88.74	57763700	88.74
2008-10-08	85.91	96.33	85.68	89.79	78847900	89.79

Generate the Descriptive Statistics of a DataFrame

In [66]:
```
df.describe()
```

Out[66]:

	Open	High	Low	Close	Volume	Adj Close
count	6081.000000	6081.000000	6081.000000	6081.000000	6.081000e+03	6081.000000
mean	46.823511	47.681506	45.913595	46.798619	1.363986e+07	23.529794
std	33.993517	34.578077	33.273106	33.947235	1.352107e+07	37.375601
min	12.880000	13.190000	12.720000	12.940000	8.880000e+04	1.650000
25%	24.730000	25.010000	24.200000	24.690000	5.530000e+06	7.380000
50%	38.250000	38.880000	37.460000	38.130000	8.976400e+06	9.910000
75%	53.500000	54.550000	52.500000	53.610000	1.631920e+07	14.360000
max	200.590000	202.960000	197.800000	199.830000	2.650690e+08	199.830000

Info about the DataFrame

Can we get some info about the dataframe?
(i.e. a schema - a summary of columns)

In [67]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 6081 entries, 2008-10-14 to 1984-09-07
Data columns (total 6 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Open       6081 non-null   float64
 1   High       6081 non-null   float64
 2   Low        6081 non-null   float64
 3   Close      6081 non-null   float64
 4   Volume     6081 non-null   int64
 5   Adj Close  6081 non-null   float64
dtypes: float64(5), int64(1)
memory usage: 332.6+ KB
```

Sort values by “Date”

```
df = df.sort_values(by = "Date")
```

```
df.head(3)
```

Date	Open	High	Low	Close	Volume	Adj Close
1984-09-07	26.50	26.87	26.25	26.50	2981600	3.02
1984-09-10	26.50	26.62	25.87	26.37	2346400	3.01
1984-09-11	26.62	27.37	26.62	26.87	5444000	3.07

```
df.tail(3)
```

Date	Open	High	Low	Close	Volume	Adj Close
2008-10-10	85.70	100.00	85.00	96.80	79260700	96.80
2008-10-13	104.55	110.53	101.02	110.26	54967000	110.26
2008-10-14	116.26	116.40	103.14	104.08	70749800	104.08

Select one column: adjusted close

i.e. closing price after adjustments for all applicable splits and dividend distributions

```
In [68]: df_adj_close = df[ ["Adj Close"] ]
```

```
In [69]: df_adj_close
```

Out[69]:

Date	Adj Close
1984-09-07	3.02
1984-09-10	3.01
1984-09-11	3.07
1984-09-12	2.98
1984-09-13	3.14
...	...
2008-10-08	89.79
2008-10-09	88.74
2008-10-10	96.80
2008-10-13	110.26
2008-10-14	104.08

6081 rows × 1 columns

Calculate the returns from close

```
In [70]: df_R = df_adj_close.pct_change()
```

```
In [71]: df_R
```

Out[71]:

Date	Adj Close
1984-09-07	NaN
1984-09-10	-0.003311
1984-09-11	0.019934
1984-09-12	-0.029316
1984-09-13	0.053691
...	...
2008-10-08	0.007066
2008-10-09	-0.011694
2008-10-10	0.090827
2008-10-13	0.139050
2008-10-14	-0.056049

6081 rows × 1 columns

.pct_change() method

https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.pct_change.html

- Percentage change between the current and a prior element

In [72]:
```
# similar calculations

df_R = np.log(df_adj_close / df_adj_close.shift())

df_R = np.log(df_adj_close) - np.log(df_adj_close.shift(1))
```

Update the index "dtype" to "datetime"

```
In [73]: df_adj_close.index

Out[73]: Index(['1984-09-07', '1984-09-10', '1984-09-11', '1984-09-12', '1984-09-13',
                '1984-09-14', '1984-09-17', '1984-09-18', '1984-09-19', '1984-09-20',
                ...
                '2008-10-01', '2008-10-02', '2008-10-03', '2008-10-06', '2008-10-07',
                '2008-10-08', '2008-10-09', '2008-10-10', '2008-10-13', '2008-10-14'],
               dtype='object', name='Date', length=6081)

In [74]: df_adj_close.index = pd.to_datetime(df_adj_close.index)

In [75]: df_adj_close.index

Out[75]: DatetimeIndex(['1984-09-07', '1984-09-10', '1984-09-11', '1984-09-12',
                        '1984-09-13', '1984-09-14', '1984-09-17', '1984-09-18',
                        '1984-09-19', '1984-09-20',
                        ...
                        '2008-10-01', '2008-10-02', '2008-10-03', '2008-10-06',
                        '2008-10-07', '2008-10-08', '2008-10-09', '2008-10-10',
                        '2008-10-13', '2008-10-14'],
                       dtype='datetime64[ns]', name='Date', length=6081, freq=None)
```

https://pandas.pydata.org/docs/user_guide/timeseries.html

Calculate the rolling average

```
In [76]: df_one_year = df_adj_close[1:254]
         df_one_year.head()
```

Out[76]:

Date	Adj Close
1984-09-10	3.01
1984-09-11	3.07
1984-09-12	2.98
1984-09-13	3.14
1984-09-14	3.18

https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.rolling.html

```
In [77]: rolling = df_one_year.rolling(window=20).mean()
         rolling.tail()
```

Out[77]:

Date	Adj Close
1985-09-03	1.711
1985-09-04	1.709
1985-09-05	1.709
1985-09-06	1.708
1985-09-09	1.708

Visualize the rolling average

```
In [78]: import matplotlib.pyplot as plt
```

Subpackage:

https://matplotlib.org/stable/api/pyplot_summary.html

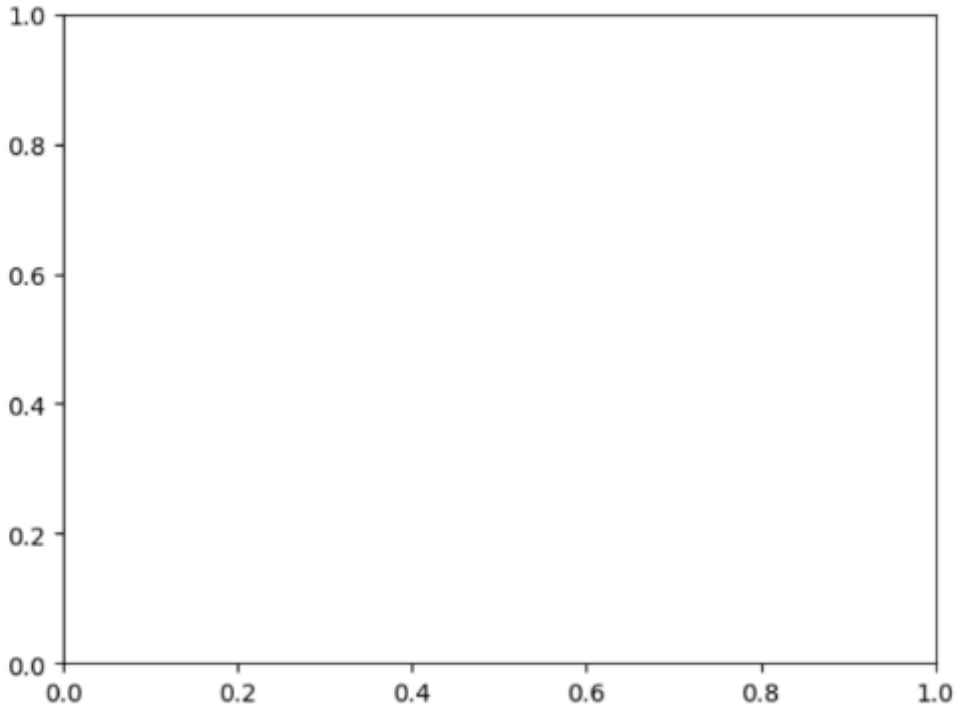
- i.e. a way to plot

Visualize the rolling average

In [79]:
```
plt.subplots() # https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.subplots.html

# create a figure
```

Out[79]:
```
(<Figure size 640x480 with 1 Axes>, <Axes: >)
```

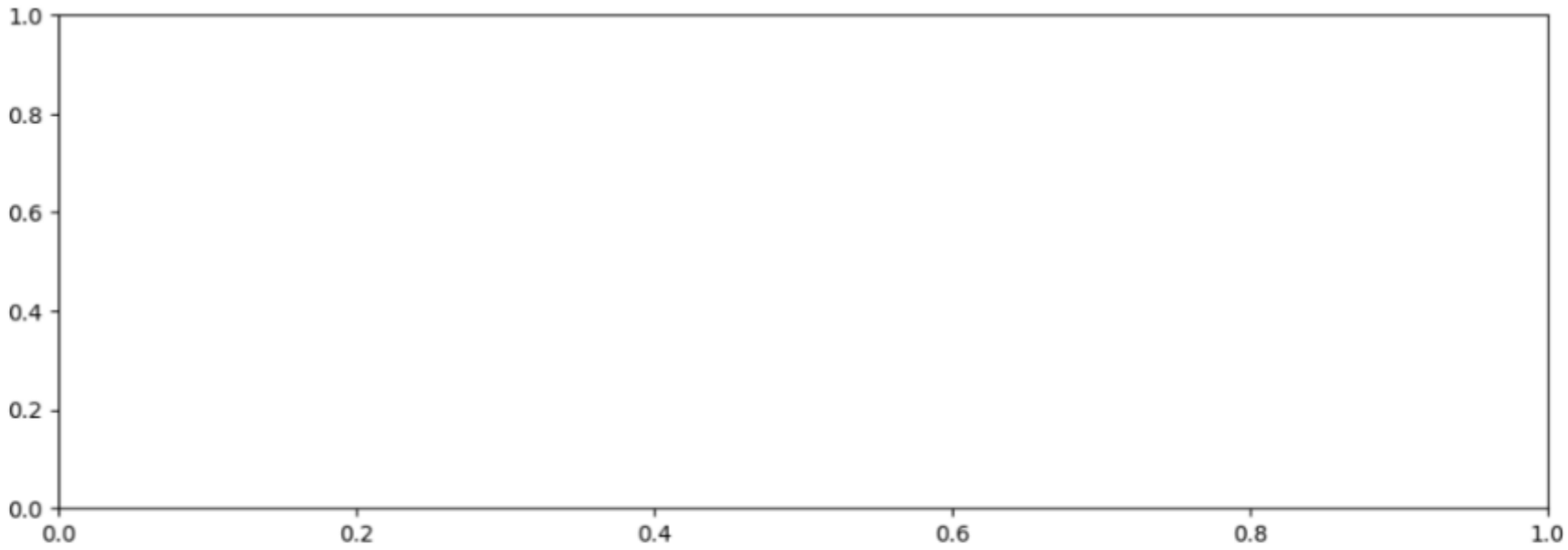


Visualize the rolling average

In [80]:
```
# adjust the figsize for my screen

plt.subplots(figsize=(12,4))
```

Out[80]:
```
(<Figure size 1200x400 with 1 Axes>, <Axes: >)
```

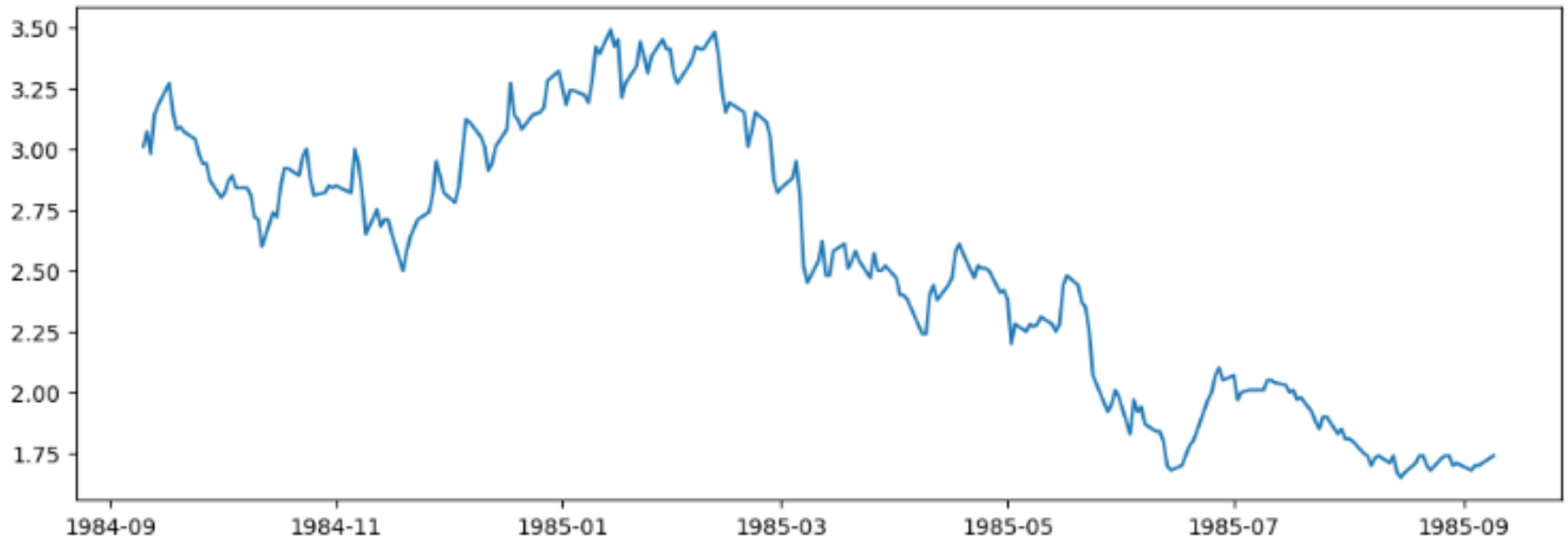


Visualize the rolling average

In [81]:
```
plt.subplots(figsize=(12,4))

plt.plot(df_one_year) # default is blue
```

Out[81]:
```
[<matplotlib.lines.Line2D at 0x21575d6a3d0>]
```



Visualize the rolling average

In [82]:
```
import matplotlib.pyplot as plt
plt.subplots(figsize=(12,4))

plt.plot(df_one_year) # default is blue
plt.plot(rolling)     # default is orange
```

Out[82]:
```
[<matplotlib.lines.Line2D at 0x21575dff790>]
```