Python : Introduction to Pandas and Time Series Analysis

IYKRA

auzaneykbal@gmail.com

Introduction of Python

Common Python Libraries for Data Science

Overview of Pandas Library

Coding

What is Python?

Python is a dynamic, high level, free open source and **interpreted programming language.** It supports object-oriented programming as well as procedural oriented programming.

The latest programming language trends claim Python to be especially effective for such domains:

- Analytics
- Web development
- Desktop apps development
- Machine learning
- AI
- Automation
- Credit Scoring and more

Sample of apps that using Python

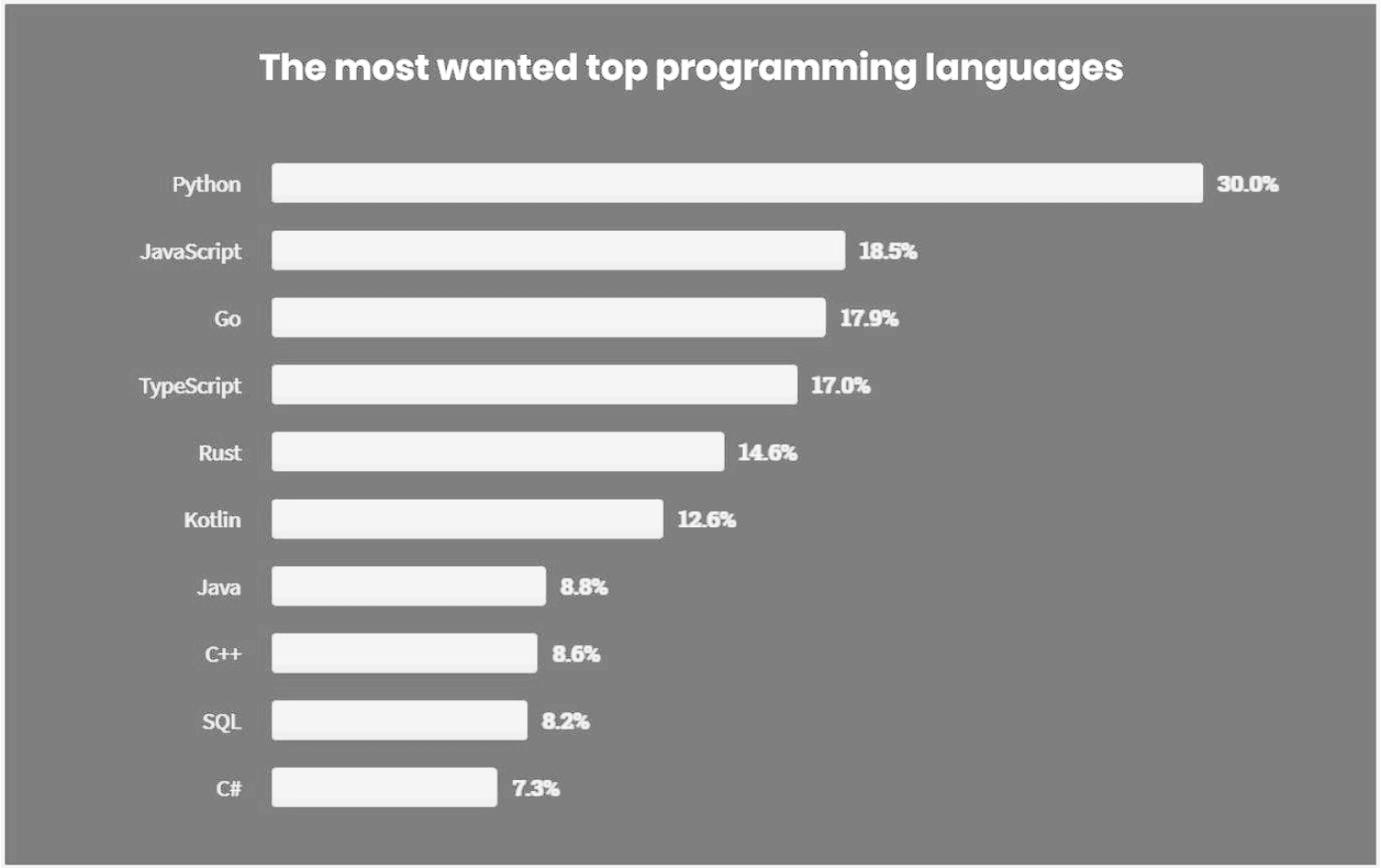
- Instagram
- Google
- Spotify
- Netflix and more

Trend

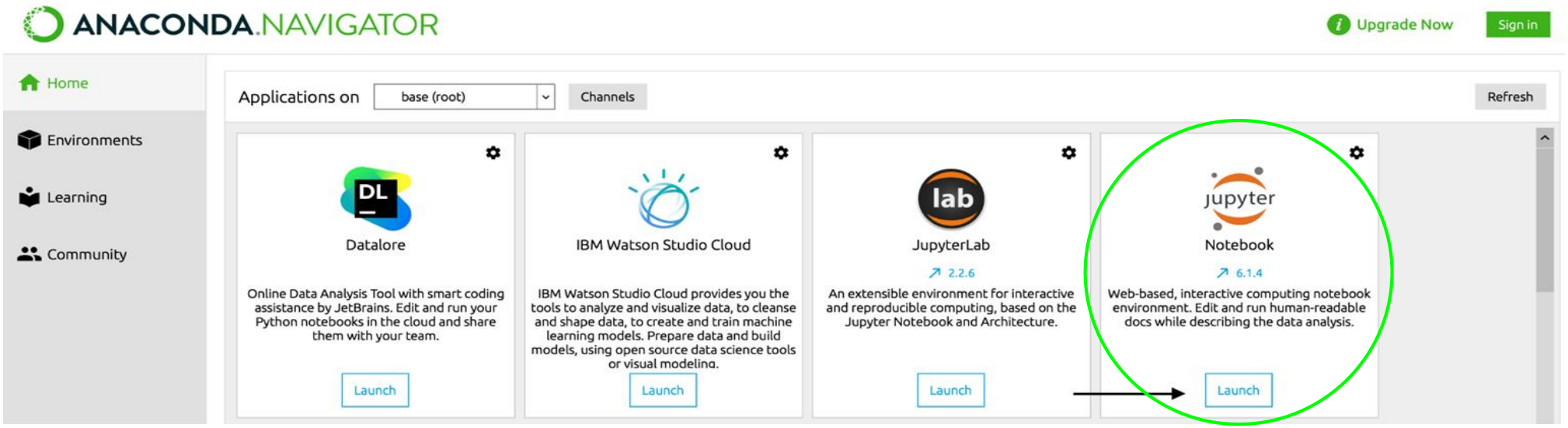
Based on survey from Stackoverflow.com

Survey population:

65,000 software developers from 186 countries around the world.



GUI (Graphical User Interface)



Anaconda Navigator is a desktop graphical user interface (GUI) that allows you to **launch applications** and easily manage conda packages, environments, and channels **without using command-line commands**

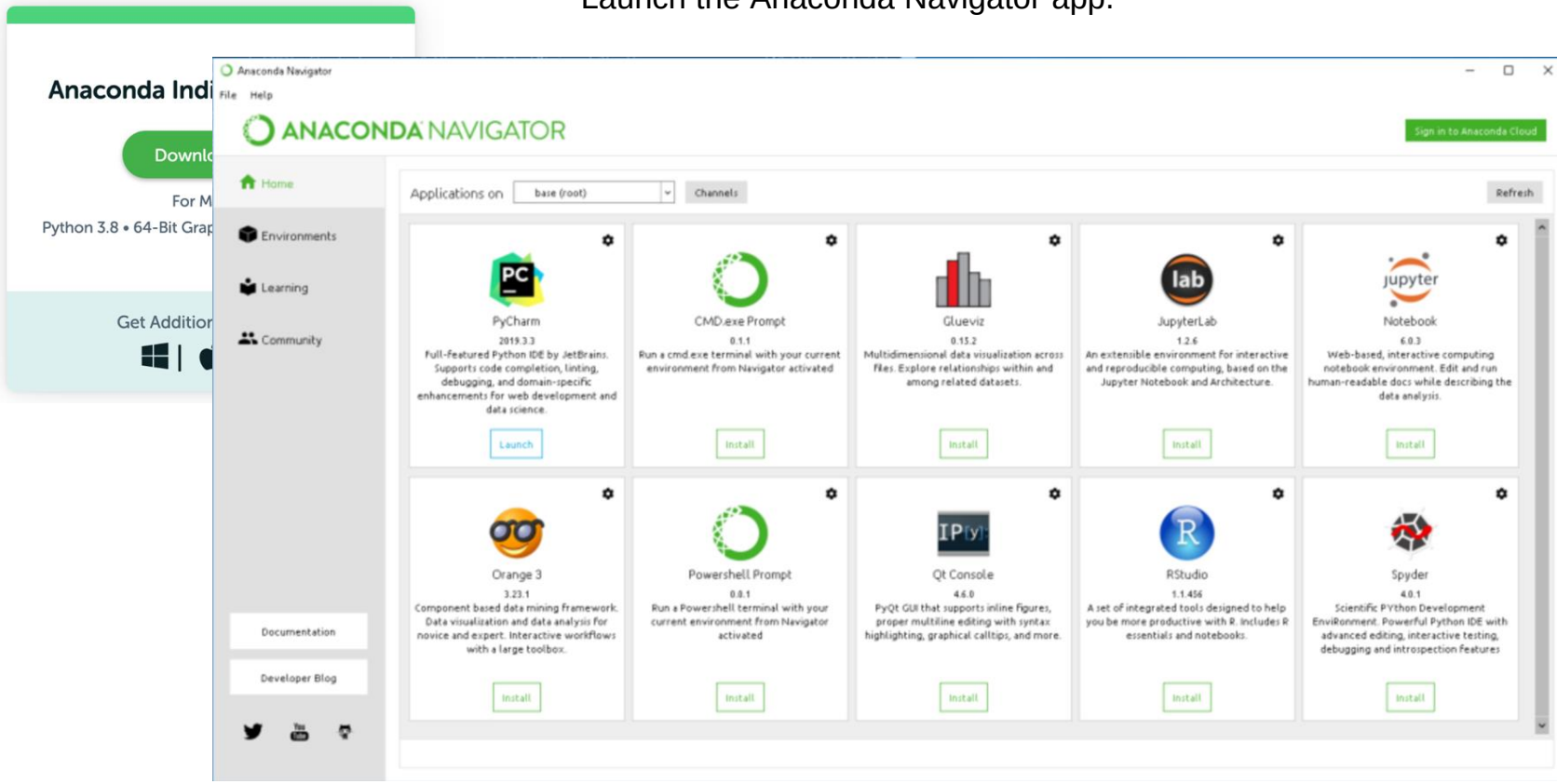
How to access it?

Individual Edition

Your data science toolkit

With over 25 million users worldwide, the open-source Individual Edition (Distribution) is the easiest way to perform Python/R data science and machine learning on a single machine. Developed for solo practitioners, it is the toolkit that equips you to work with thousands of open-source packages and libraries.

Launch the Anaconda Navigator app.



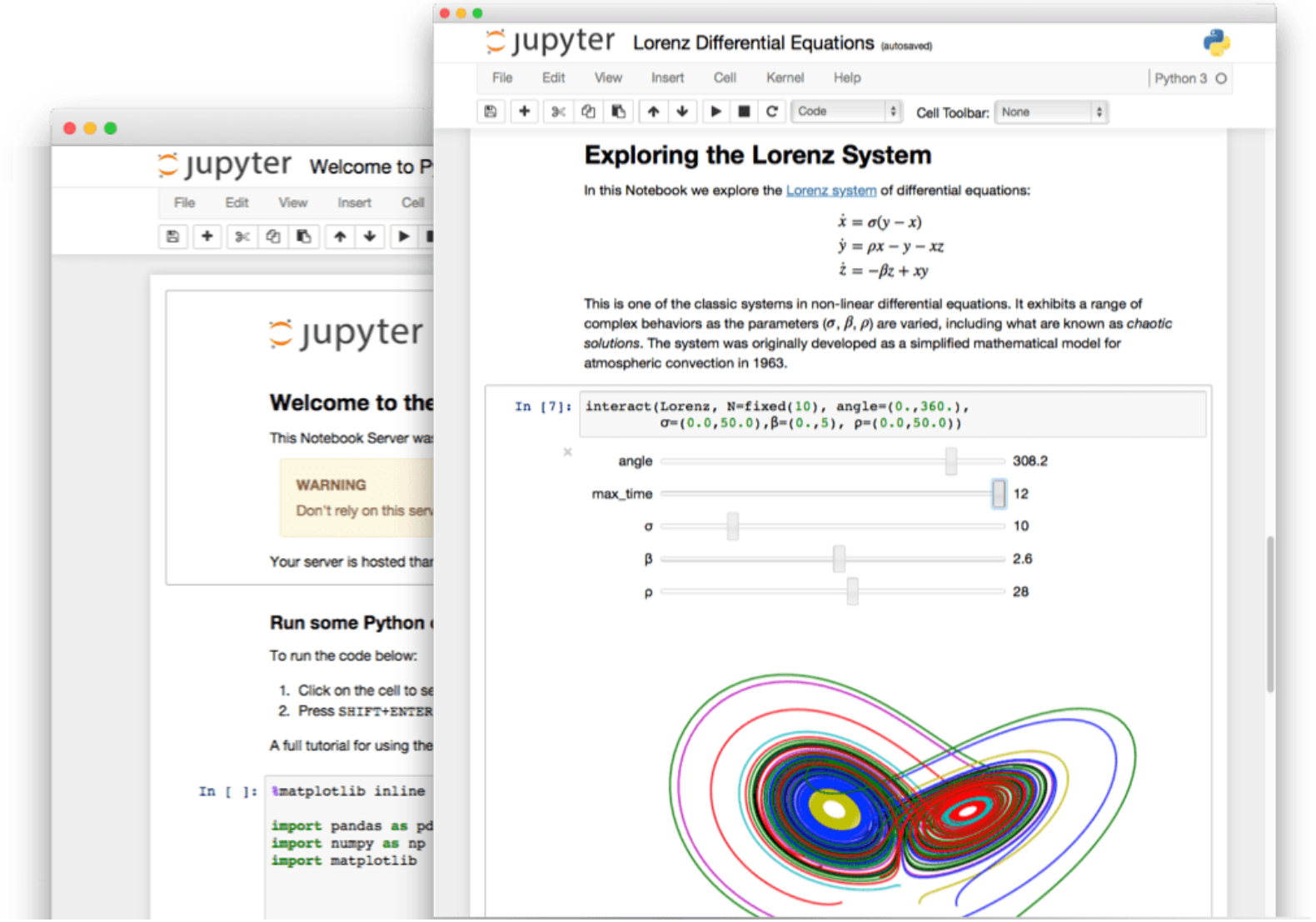
Install Anaconda Navigator on
https://www.anaconda.com/products/individual

Jupyter Notebook

Code Editor

The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text.

Uses include: data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more.



Python Libraries for Data Science

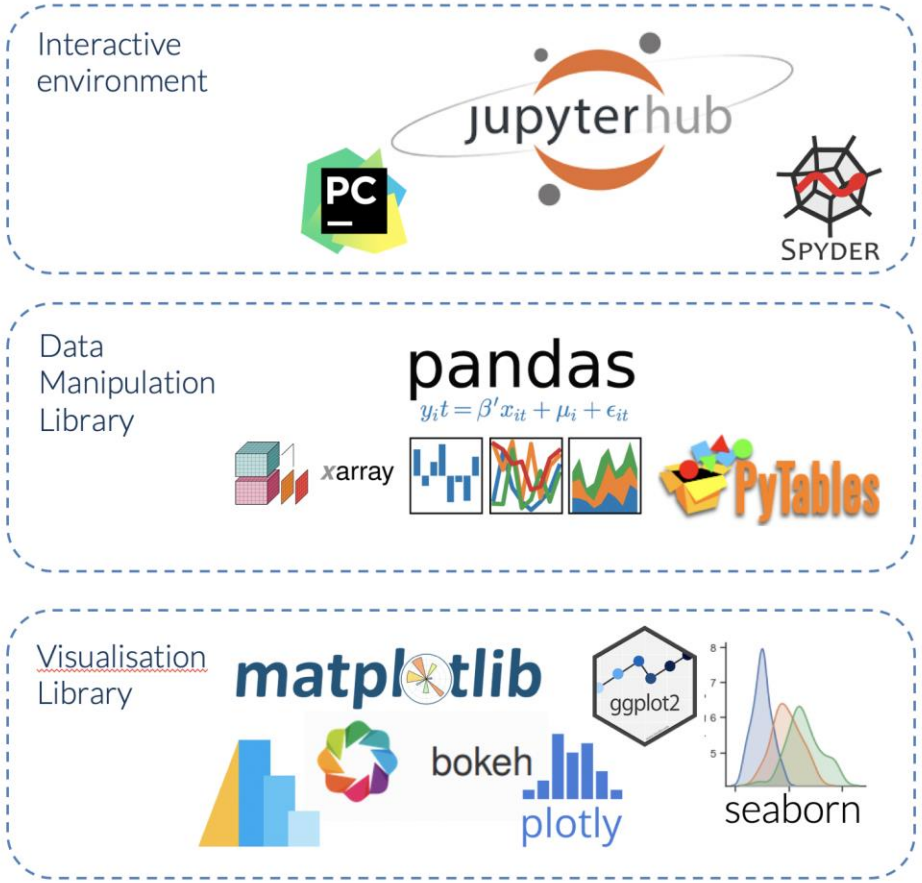
Many popular Python toolboxes/libraries:

- **Pandas**
- NumPy
- SciPy
- SciKit-Learn
- and many more …

Visualization libraries

- matplotlib
- Seaborn
- and many more …

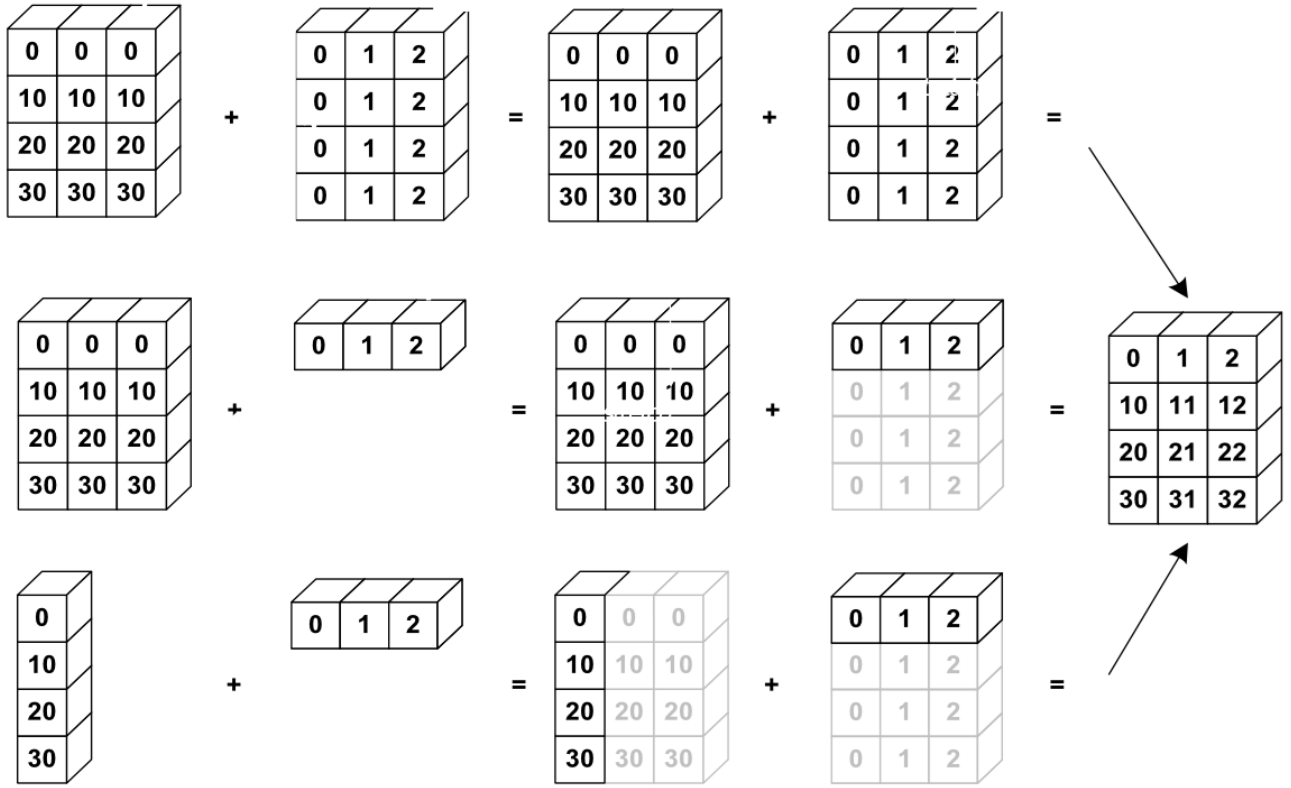
All these libraries are installed through Anaconda Navigator



NumPy

Introduces objects for multidimensional arrays and matrices, as well as functions that allow to easily perform advanced mathematical and statistical operations on those objects

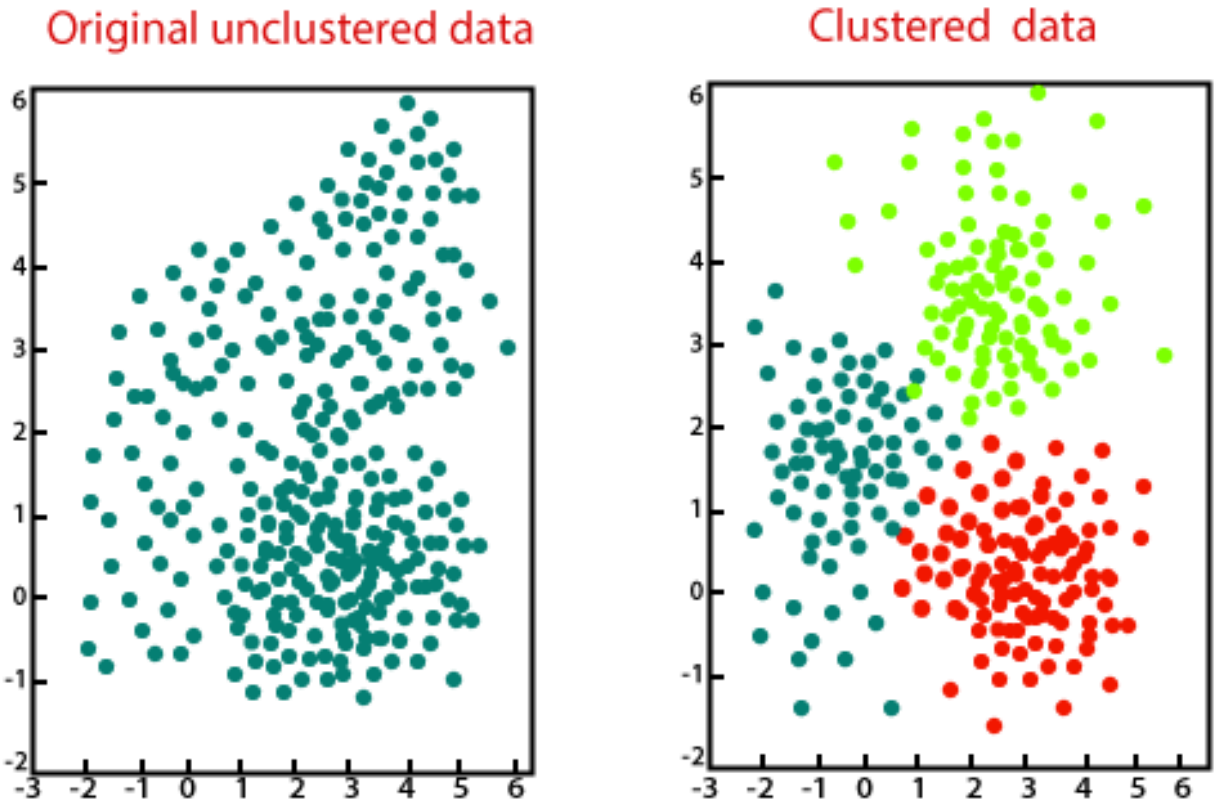
Provides vectorization of mathematical operations on arrays and matrices which significantly improves the performance



Link: http://www.numpy.org/

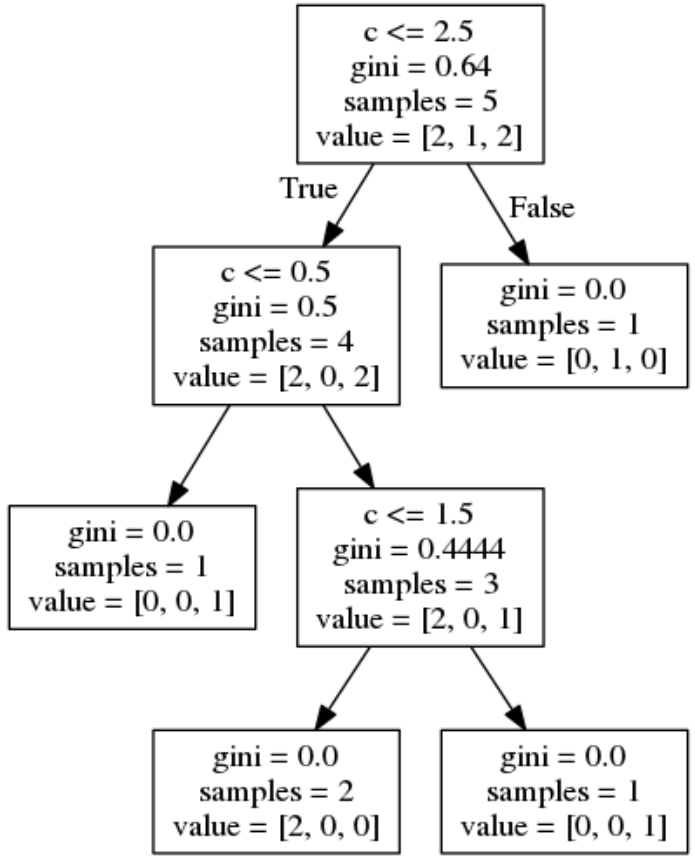
SciPy.org

- collection of algorithms for linear algebra, differential equations, numerical integration, optimization, statistics and more
- part of SciPy Stack
- built on NumPy



Link: https://www.scipy.org/scipylib/

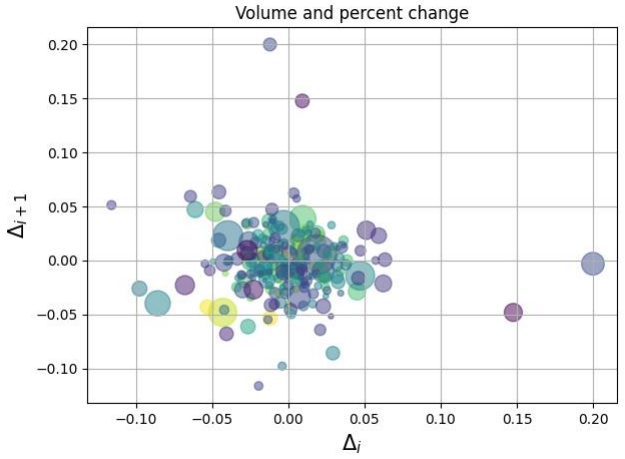
- provides machine learning algorithms: classification, regression, clustering, model validation etc.
- built on NumPy, SciPy and matplotlib



Link: http://scikit-learn.org/

matplotlib

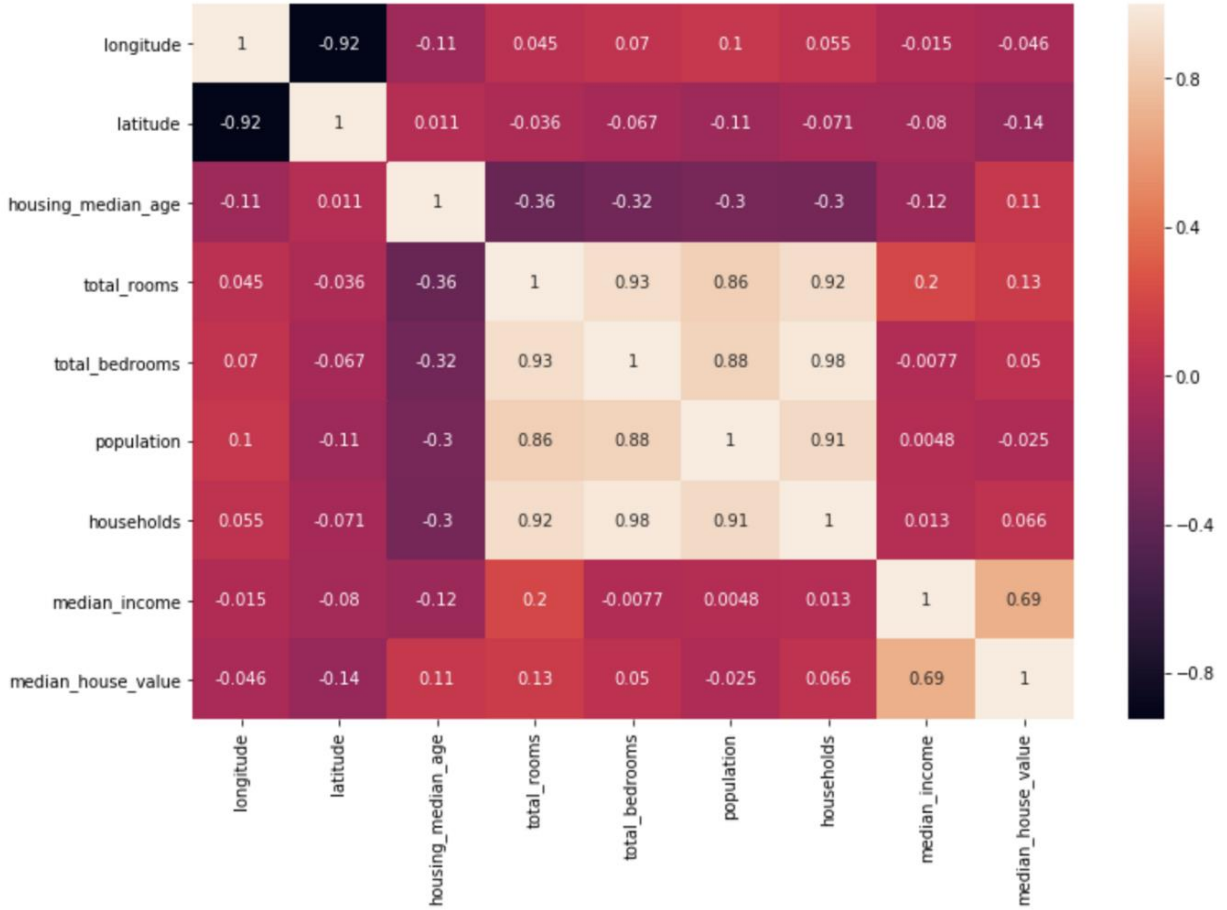
- python 2D plotting library which produces publication quality figures in a variety of hardcopy formats
- a set of functionalities similar to those of MATLAB
- line plots, scatter plots, barcharts, histograms, pie charts etc.
- relatively low-level; some effort needed to create advanced visualization



Link: https://matplotlib.org/

Seaborn

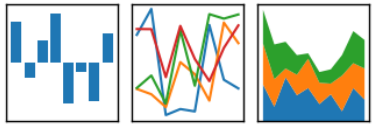
- based on matplotlib
- provides high level interface for drawing attractive statistical graphics
- Similar (in style) to the popular ggplot2 library in R



Link: https://seaborn.pydata.org/

pandas

$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$



Adds data structures and tools designed to work with table-like data

Export

Import

XLSX, XLS, CSV, JSON, XML, TXT

Provides tools for data manipulation: reshaping, merging, sorting, slicing, aggregation etc.

From

2021-03-28 14:15:00

2021-03-28 14:17:28

2021-03-28 14:50:50

To

14:15:00

14:17:28

14:50:50

Allows handling missing data

From

A | 50 | 3.2

B | | 3.5

C | 75 | 8.5

To

A | 50 | 3.2

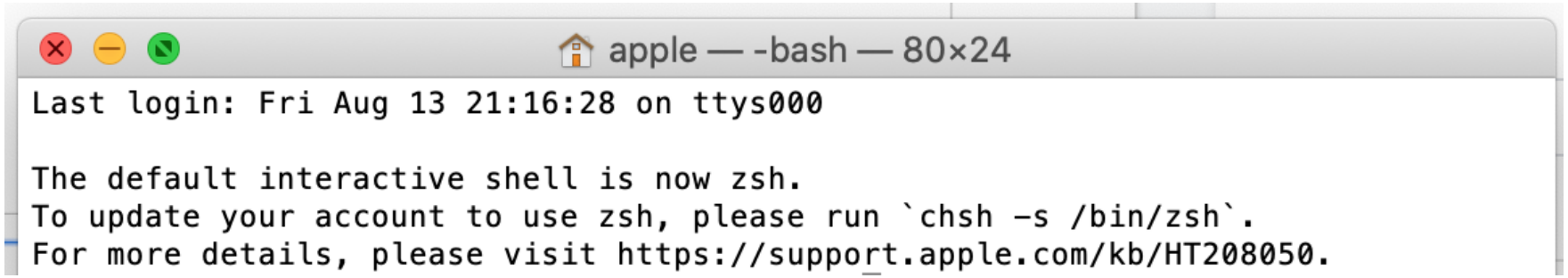
B | NaN | 3.5

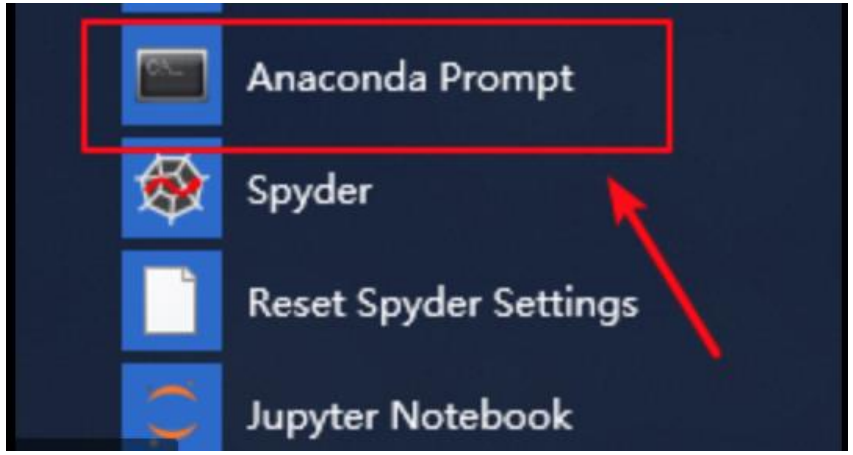
C | 75 | 8.5

Link: http://pandas.pydata.org/

Need more libraries?

Using “terminal” on Mac OR “Anaconda Prompt” on Windows after GUI installation completed

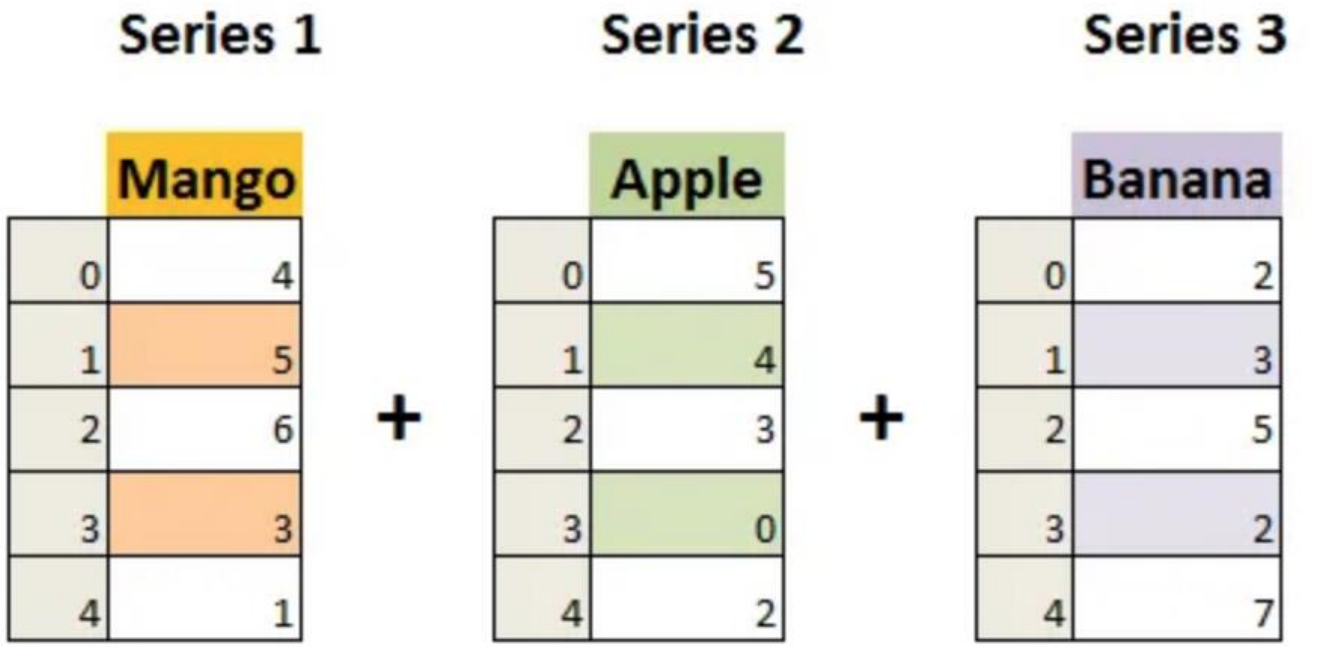




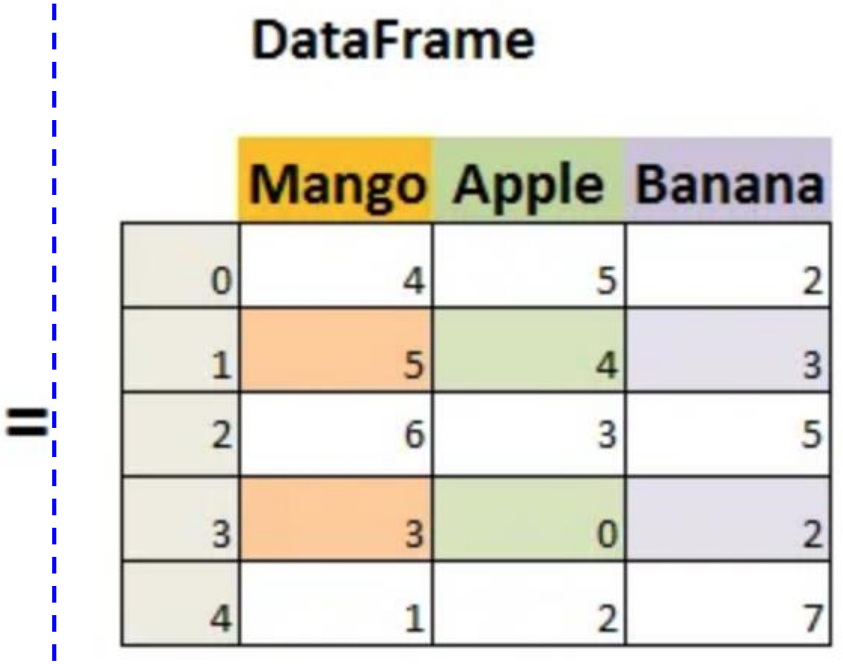
To install, please visit https://anaconda.org/ and search any libraries for your model

Data Structures

Each column in a **DataFrame** is a **Series**



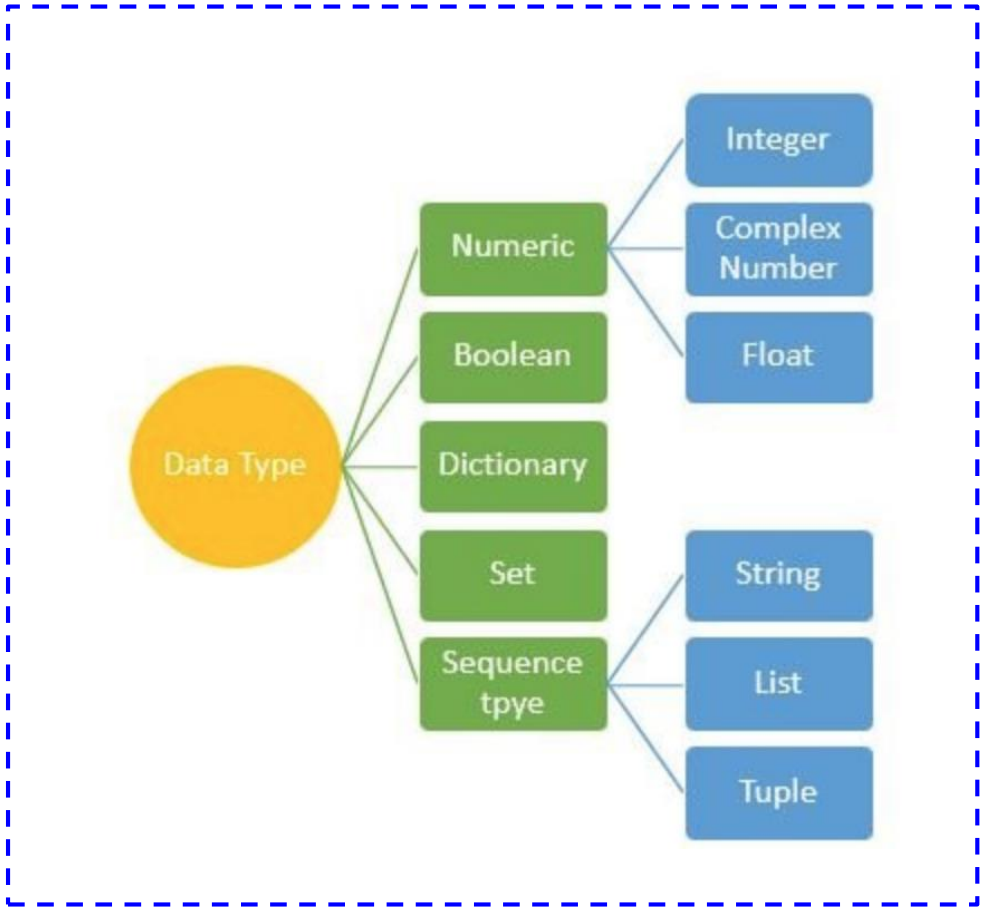
A **DataFrame** is a two-dimensional data structure



You can do things by applying a method to a DataFrame or Series

```
df = pd.DataFrame( {"a" : [4 ,5, 6], "b" : [7, 8, 9], "c" : [10, 11, 12]}, index = [1, 2, 3])
```

Common Data Types

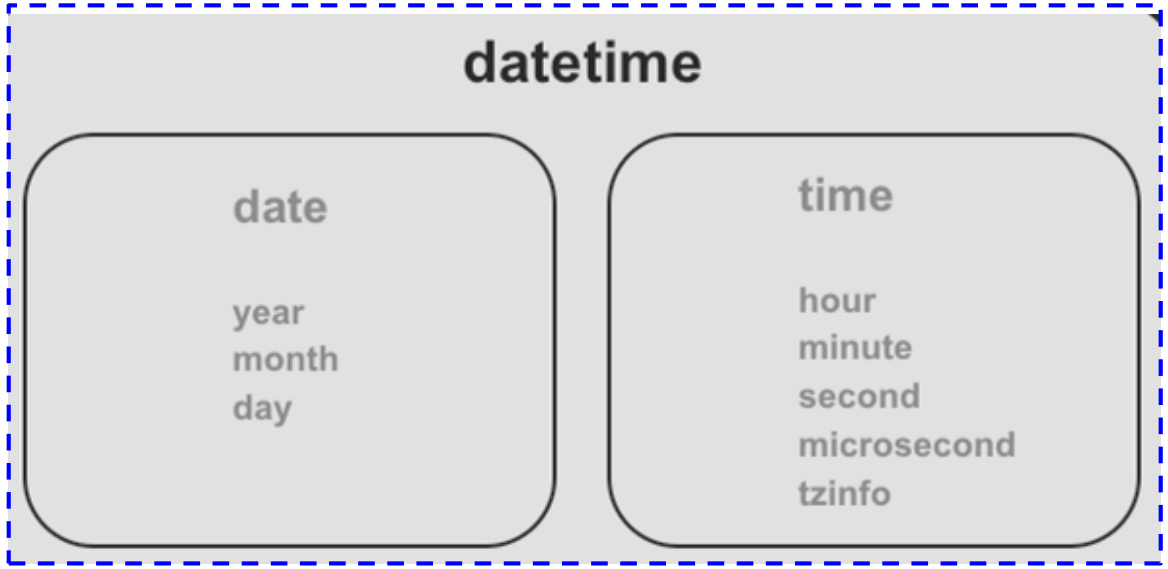



```
x1 = str("Hello World") #str#
x2 = int(20) #int#
x3 = float(20.5) #float#
x4 = list(("apple", "banana", "cherry")) #list#
x5 = tuple(("apple", "banana", "cherry")) #tuple#
x6 = dict(name="Mr X", age=36) #dict#
x7 = set(("apple", "banana", "cherry")) #set#
x8 = frozenset(("apple", "banana", "cherry")) #frozenset#
x9 = bool(5) #bool#
```

```
display(x1,x2,x3,x4,x5,x6,x7,x8,x9)
```

```
'Hello World'

20

20.5

['apple', 'banana', 'cherry']

('apple', 'banana', 'cherry')

{'name': 'Mr X', 'age': 36}

{'apple', 'banana', 'cherry'}

frozenset({'apple', 'banana', 'cherry'})

True
```

Common Date/Time Types




```
from datetime import *

x12 = date.today()
x13 = date.today().isoformat()
x14 = date.today().ctime()
x15 = date.today().strftime("%d/%m/%y")
x16 = date.today().strftime("%A %d. %B %Y")
x17 = datetime.today().strftime("%d/%m/%y %H:%M:%S")
x18 = date.today().year
x19 = date.today().month
x20 = date.today().day
x21 = datetime.today().hour
x22 = datetime.today().minute
x23 = datetime.today().second
```

```
display(x12,x13,x14,x15,x16,x17,x18,x19,x20,x21,x22,x23)
```

```
datetime.date(2021, 8, 14)

'2021-08-14'

'Sat Aug 14 00:00:00 2021'

'14/08/21'

'Saturday 14. August 2021'

'14/08/21 02:17:49'

2021

8

14

2

17

49
```

Read Data

```
data_mac = pd.read_csv('/Users/apple/Documents/Python/state-population.csv')

data_windows = pd.read_csv('C:\Users\folder_x\state-population.csv')
```

Export Data

```
data_mac.to_excel("output.xlsx",
                  sheet_name='data_test')
```

Info & Describe Data

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3 entries, 1 to 3
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   a       3 non-null      int64
 1   b       3 non-null      int64
 2   c       3 non-null      int64
dtypes: int64(3)
memory usage: 96.0 bytes
```

```
df.describe()
```

	a	b	c
count	3.0	3.0	3.0
mean	5.0	8.0	11.0
std	1.0	1.0	1.0
min	4.0	7.0	10.0
25%	4.5	7.5	10.5
50%	5.0	8.0	11.0
75%	5.5	8.5	11.5
max	6.0	9.0	12.0

Create Data

```
import pandas as pd
df = pd.DataFrame({"a" : [4 ,5, 6],
                   "b" : [7, 8, 9],
                   "c" : [10, 11, 12]},
                  index = [1, 2, 3])
df
```

	a	b	c
1	4	7	10
2	5	8	11
3	6	9	12

Gather columns into rows

```
df2 = pd.melt(df)
```

```
df2
```

	variable	value
0	a	4
1	a	5
2	a	6
3	b	7
4	b	8
5	b	9
6	c	10
7	c	11
8	c	12

Gather columns into rows

```
df1 = pd.DataFrame({"a" : [4 ,5, 6],
                    "b" : [7, 8, 9],
                    "c" : [10, 11, 12]},
                   index = [1, 2, 3])
```

```
df2 = pd.DataFrame({"a" : [9 ,10, 12],
                    "b" : [1, 1, 1],
                    "c" : [2, 2, 2]},
                   index = [1, 2, 3])
```

```
pd.concat([df1,df2])
```

	a	b	c
1	4	7	10
2	5	8	11
3	6	9	12
1	9	1	2
2	10	1	2
3	12	1	2

Concatenate Column No Append

```
df5 = pd.concat([df1,df2], axis=1)
df5
```

	a	b	c	a	b	c
1	4	7	10	9	1	2
2	5	8	11	10	1	2
3	6	9	12	12	1	2

Rename Columns

```
df7= df2.copy()
df7.rename(columns = {'a':'a2',
                      'b':'b2',
                      'c':'c2'}, inplace = True)
df7
```

	a2	b2	c2
1	9	1	2
2	10	1	2
3	12	1	2

Sort Column by Values

```
df7.sort_values('a2',ascending=False)
```

	a2	b2	c2
3	12	1	2
2	10	1	2
1	9	1	2

Sort Column by Index

```
df8 = df7.sort_index()
df8
```

	a2	b2	c2
1	9	1	2
2	10	1	2
3	12	1	2

Reset Index

```
df9 = df8.reset_index()
df9
```

	index	a2	b2	c2
0	1	9	1	2
1	2	10	1	2
2	3	12	1	2

Drop Column

```
df10 = df9.drop(columns=['index'])
df10
```

	a2	b2	c2
0	9	1	2
1	10	1	2
2	12	1	2

Operators and if Condition

```
import numpy as np
df10['times']= df10.a2 * df10.c2
df10['square']= df10.a2 ** df10.c2
df10['divide']= df10.a2 / df10.c2
df10['floor']= df10.a2 // df10.c2
df10['modulus']= df10.a2 % df10.c2
df10['addition']= df10.a2 + df10.c2
df10['subtraction']= df10.a2 - df10.c2
df10['equal']= df10.a2 == df10.c2
df10['not_equal']= df10.a2 != df10.c2
df10['greater']= df10.a2 > df10.c2
df10['lower']= df10.a2 < df10.c2
df10['if'] = df10['times'].apply(lambda x: 'True' if x <= 20 else 'False')

conditions = [
    (df10['a2'] > 2) & (df10['addition'] < 12) & df10['greater']==True,
    (df10['a2'] > 9) & (df10['addition'] <= 12),
    (df10['a2'] > 12)]

values = ['tag1','tag2','tag3']
df10['tag'] = np.select(conditions, values,default='no_tag')

df10
```

	a2	b2	c2	times	divide	square	modulus	addition	subtraction	equal	not_equal	greater	lower	floor	if	tag
0	9	1	2	18	4.5	81	1	11	7	False	True	True	False	4	True	tag1
1	10	1	2	20	5.0	100	0	12	8	False	True	True	False	5	True	tag2
2	12	1	2	24	6.0	144	0	14	10	False	True	True	False	6	False	no_tag

Filter Dataframe

```
df11 = df10.query('a2 <= 9 & tag=="tag1"')

df11
```

	a2	b2	c2	times	divide	square	modulus	addition	subtraction	equal	not_equal	greater	lower	floor	if	tag
0	9	1	2	18	4.5	81	1	11	7	False	True	True	False	4	True	tag1

```
df12 = df10.query('tag=="tag1" | tag=="tag2"')

df12
```

	a2	b2	c2	times	divide	square	modulus	addition	subtraction	equal	not_equal	greater	lower	floor	if	tag
0	9	1	2	18	4.5	81	1	11	7	False	True	True	False	4	True	tag1
1	10	1	2	20	5.0	100	0	12	8	False	True	True	False	5	True	tag2

Convert Datetime to Multiple date/time format

```
dt['datetime'] = pd.DataFrame({"date" : [datetime.today()]})
dt['just_date'] = dt['datetime'].dt.date
dt['date_format1'] = dt['datetime'].dt.strftime("%d/%m/%y")
dt['date_format2'] = dt['datetime'].dt.strftime("%A %d. %B %Y")
dt['date_format3'] = dt['datetime'].dt.strftime("%d/%m/%y %H:%M:%S")
dt['year'] = dt['datetime'].dt.year
dt['month'] = dt['datetime'].dt.month
dt['day'] = dt['datetime'].dt.day
dt['hour'] = dt['datetime'].dt.hour
dt['min'] = dt['datetime'].dt.minute
dt['sec'] = dt['datetime'].dt.second
dt['time_only'] = dt['datetime'].dt.strftime("%H:%M:%S")
dt
```

	datetime	just_date	year	month	day	hour	min	sec	date_format1	date_format2	date_format3	time_only
0	2021-08-14 02:34:45.887236	2021-08-14	2021	8	14	2	34	45	14/08/21	Saturday 14. August 2021	14/08/21 02:34:45	02:34:45

Convert Float to String

```
data = pd.DataFrame({"float_" : [65.325]})
data['string'] = data['float_'].astype(str)
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1 entries, 0 to 0
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   float_  1 non-null      float64
 1   string  1 non-null      object
dtypes: float64(1), object(1)
memory usage: 144.0+ bytes
```

Code sample