PYTHON for Data Science

Part 2

M. Hamdi Özçelik

18.10.2021

https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-0001-introduction-to-computer-science-and-programming-in-python-fall-2016/

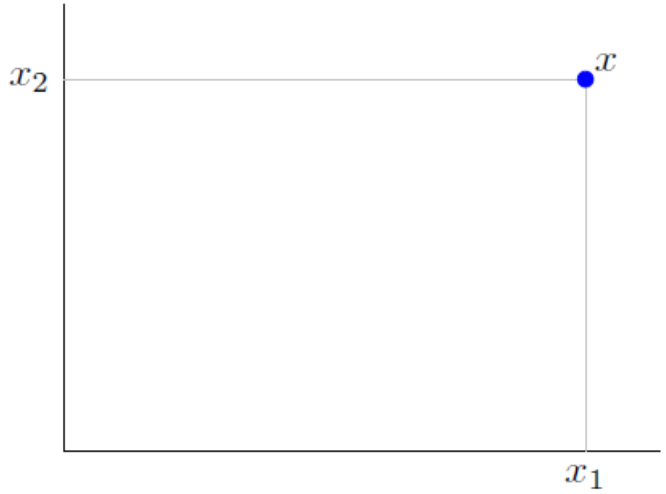
"A clever person solves a problem. A wise person avoids it."
— Albert Einstein

https://www.youtube.com/watch?v=pvMrPBRCApQ

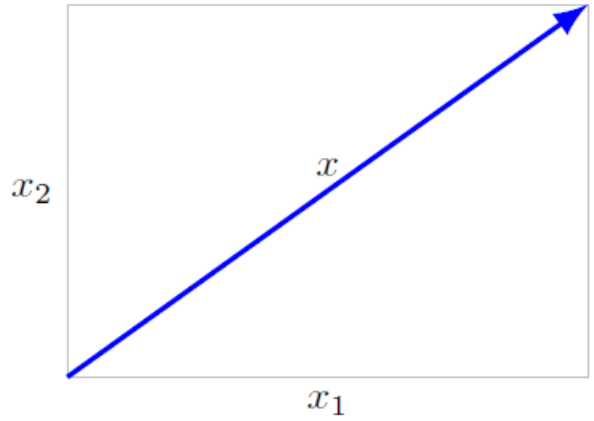
VECTORS

VECTORS

$$X = (x_1, x_2)$$



The 2-vector x species the position (shown as a dot) with coordinates x1 and x2 in a plane.



The 2-vector x represents a displacement in the plane (shown as an arrow) by x1 in the first axis and x2 in the second.

(Standard) Unit Vector

$$(e_i)_j = \begin{cases} 1 & j = i \\ 0 & j \neq i \end{cases} \qquad for\ j = 1, \dots, n$$

$$e_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \quad e_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \quad e_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

Zero Vector

$$0_n = (0, .., 0)$$

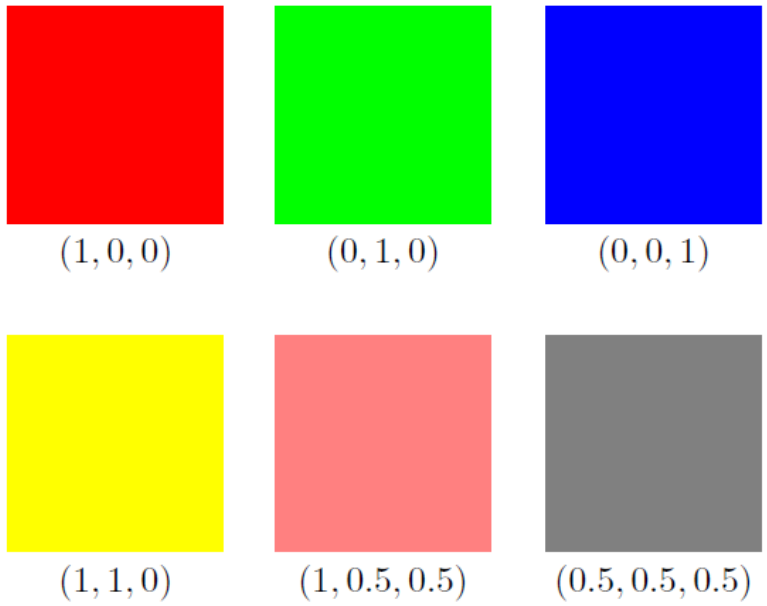
Ones Vectors

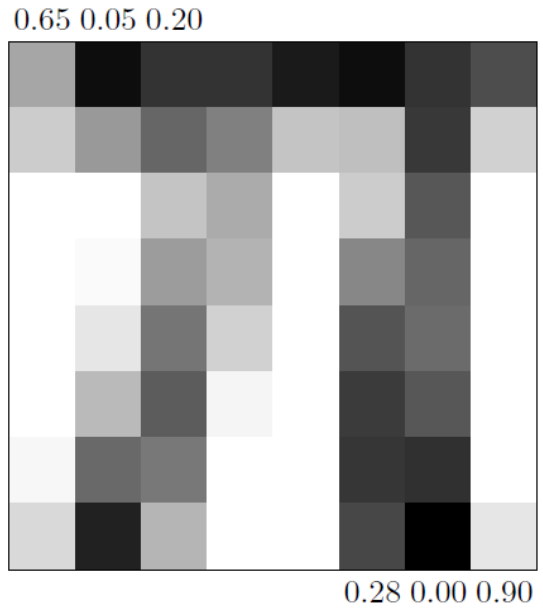
$$1_n = (1, .., 1)$$

Sparse Vector

A vector is said to be sparse if many of its entries are zero

VECTORS AS DATA STRUCTURES

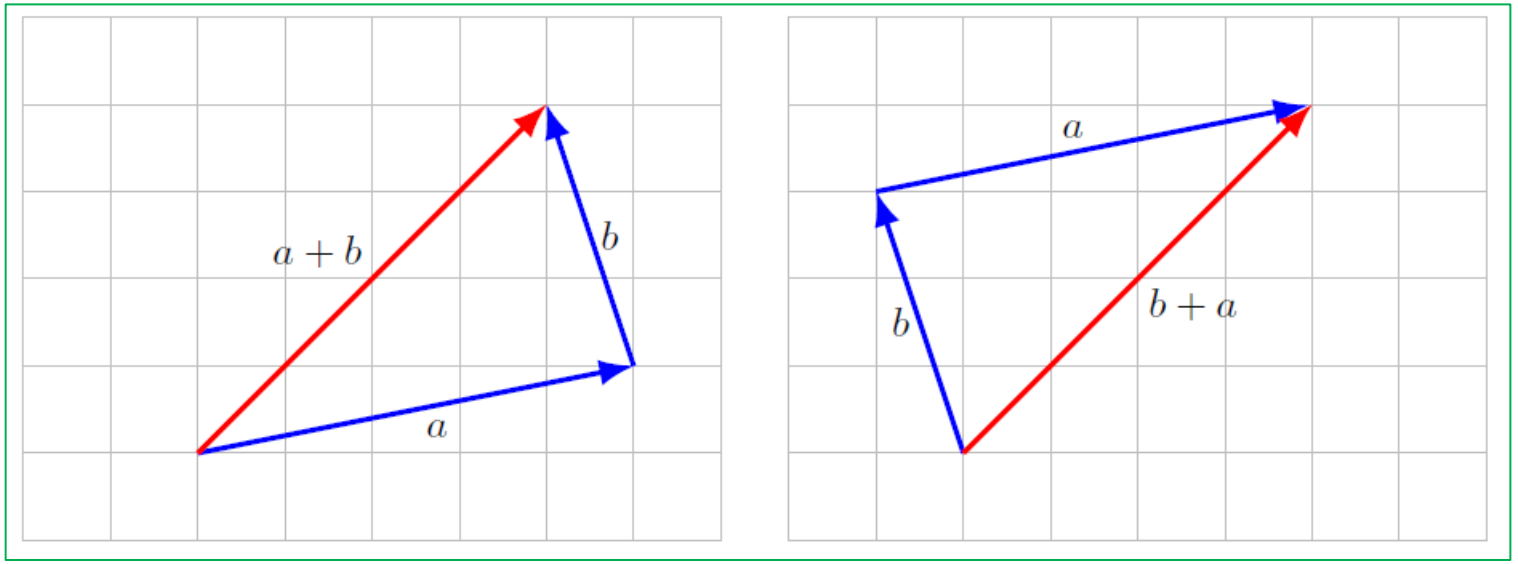


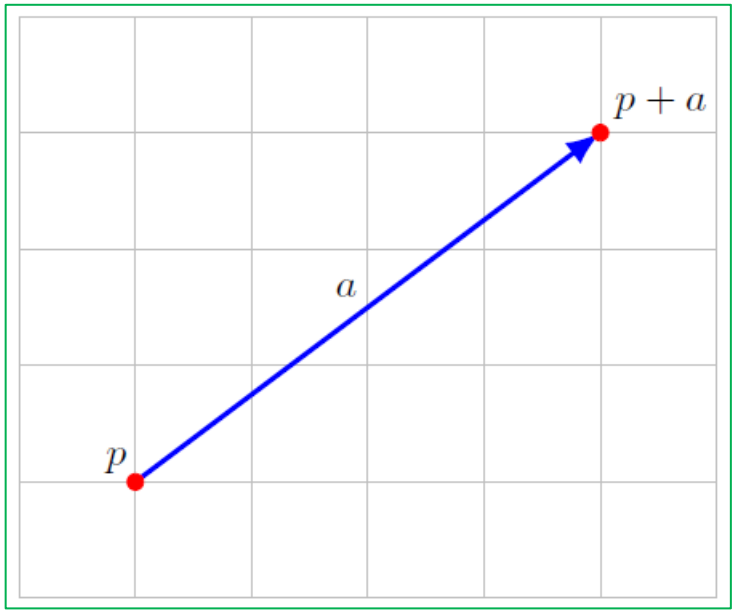


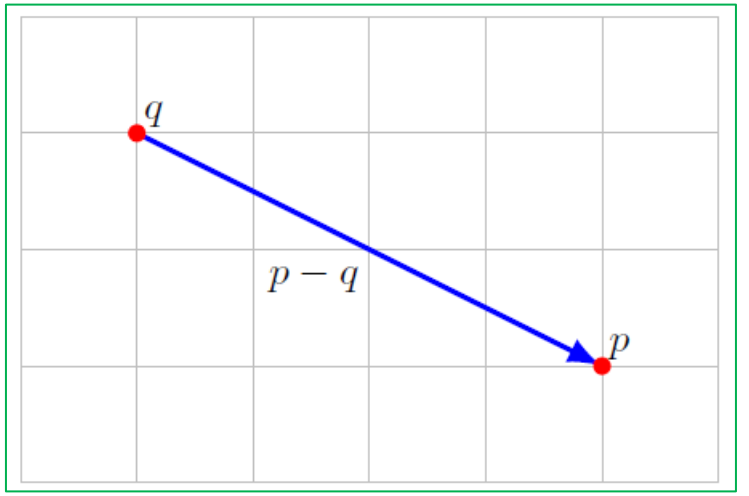
Word count vectors are used in computer based document analysis. Each entry of the word count vector is the number of times the associated dictionary word appears in the document.

$$\begin{matrix} \text{word} \\ \text{in} \\ \text{number} \\ \text{horse} \\ \text{the} \\ \text{document} \end{matrix} \begin{bmatrix} 3 \\ 2 \\ 1 \\ 0 \\ 4 \\ 2 \end{bmatrix}$$

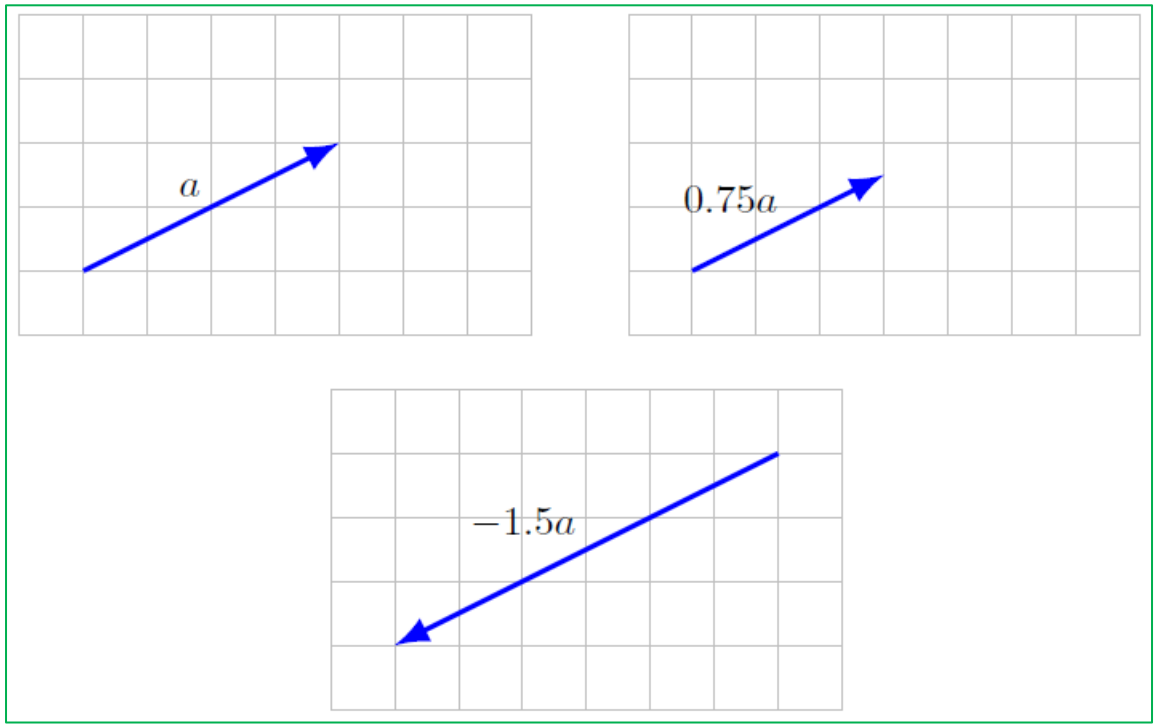
VECTOR ADDITION AND SUBTRACTION

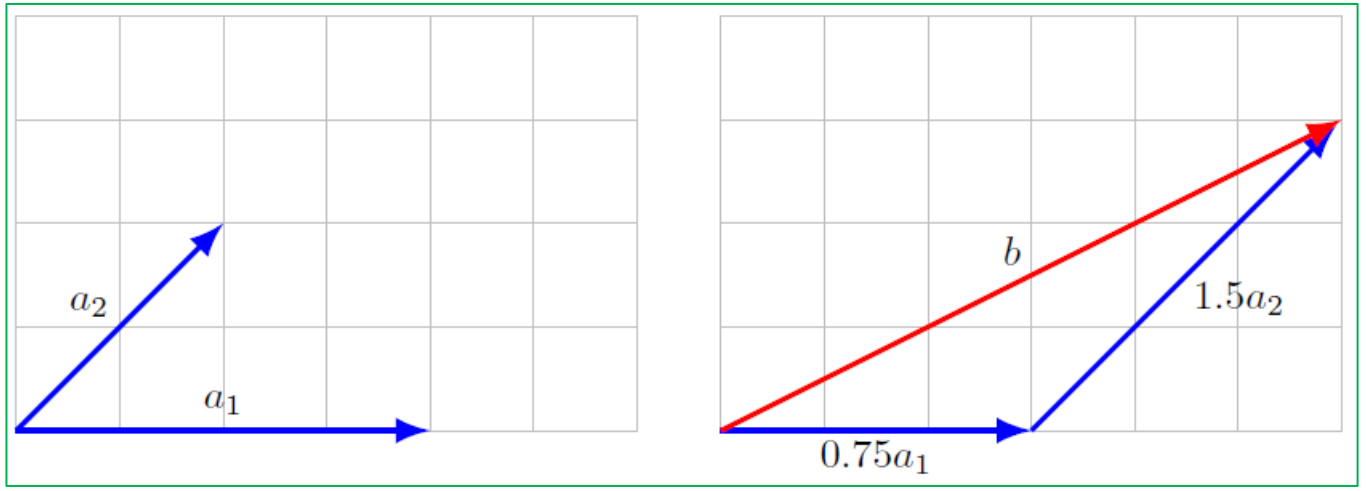






VECTOR AND SCALAR MULTIPLICATON





(STANDARD) INNER PRODUCT = DOT PRODUCT

applied analytics

Hadamard product: $\boldsymbol{a} \circ \boldsymbol{b} = [\, a_1 b_1, a_2 b_2, \ldots, a_n b_n]$

Dot product: $\boldsymbol{a} \cdot \boldsymbol{b} = a^T b = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + \ldots + a_n b_n$

$$\begin{bmatrix} -1 \\ 2 \\ 2 \end{bmatrix}^T \begin{bmatrix} 1 \\ 0 \\ -3 \end{bmatrix} = (-1)(1) + (2)(0) + (2)(-3) = -7$$

The inner product function: $\boldsymbol{f}(\boldsymbol{x}) = a^T x = a_1 x_1 + a_2 x_2 + \ldots + a_n x_n$

Regression function: $\hat{y} = \beta^T x + v$

$\hat{y}$: $prediction$

y: $true\ value, dependent\ variable, outcome, label$

x: $regressors, independent\ variables$

β: $weight\ vector, coefficient\ vector$

v: $a\ scalar$

SUMMARIZING THE VECTOR

The Euclidean **norm of an n-vector**, is the square root of the sum of the squares of its elements:

$$\|x\| = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}$$

$$\|x\| = \sqrt{x^T \cdot x}$$

The **mean-square** of an n-vector:

$$ms(x) = \frac{(x_1^2 + x_2^2 + \cdots + x_n^2)}{n} = \frac{\|x\|^2}{n}$$

The **root-mean-square** of an n-vector:

$$rms(x) = \sqrt{\frac{(x_1^2 + x_2^2 + \cdots + x_n^2)}{n}} = \frac{\|x\|}{\sqrt{n}}$$

NumPy

https://numpy.org/

LARGE SCALE SCIENTIFIC COMPUTING LIBRARY

AA applied analytics

POWERFUL N-DIMENSIONAL ARRAYS

Fast and versatile, the NumPy vectorization, indexing, and broadcasting concepts are the de-facto standards of array computing today.

NUMERICAL COMPUTING TOOLS

NumPy offers comprehensive mathematical functions, random number generators, linear algebra routines, Fourier transforms, and more.

INTEROPERABLE

NumPy supports a wide range of hardware and computing platforms, and plays well with distributed, GPU, and sparse array libraries.

PERFORMANT

The core of NumPy is well-optimized C code. Enjoy the flexibility of Python with the speed of compiled code.

EASY TO USE

NumPy's high level syntax makes it accessible and productive for programmers from any background or experience level.

OPEN SOURCE

Distributed under a liberal BSD license, NumPy is developed and maintained publicly on GitHub by a vibrant, responsive, and diverse community.

NumPy

ndarray.ndim

the number of axes (dimensions) of the array.

ndarray.shape

the dimensions of the array. This is a tuple of integers indicating the size of the array in each dimension. For a matrix with *n* rows and *m* columns, `shape` will be `(n,m)`. The length of the `shape` tuple is therefore the number of axes, `ndim`.

ndarray.size

the total number of elements of the array. This is equal to the product of the elements of `shape`.

ndarray.dtype

an object describing the type of the elements in the array. One can create or specify dtype's using standard Python types. Additionally NumPy provides types of its own. numpy.int32, numpy.int16, and numpy.float64 are some examples.

ndarray.itemsize

the size in bytes of each element of the array. For example, an array of elements of type `float64` has `itemsize` 8 (=64/8), while one of type `complex32` has `itemsize` 4 (=32/8). It is equivalent to `ndarray.dtype.itemsize`.

ndarray.data

the buffer containing the actual elements of the array. Normally, we won't need to use this attribute because we will access the elements in an array using indexing facilities.

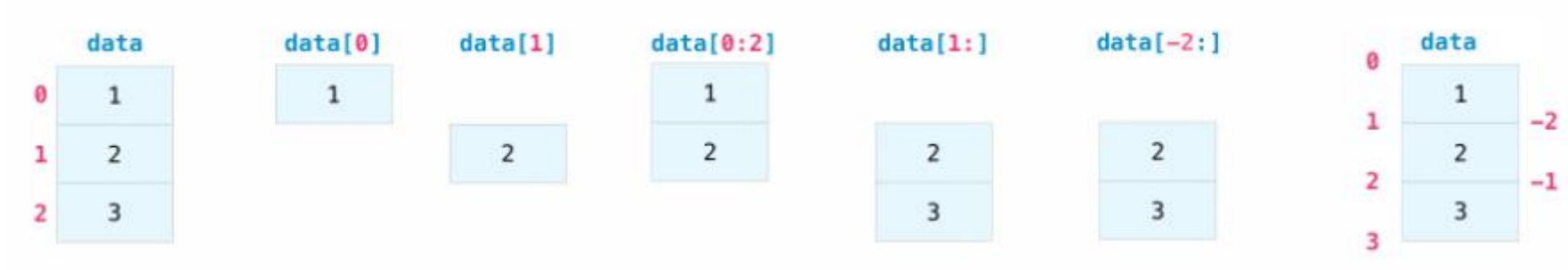
NUMPY DATA TYPES

AA applied analytics

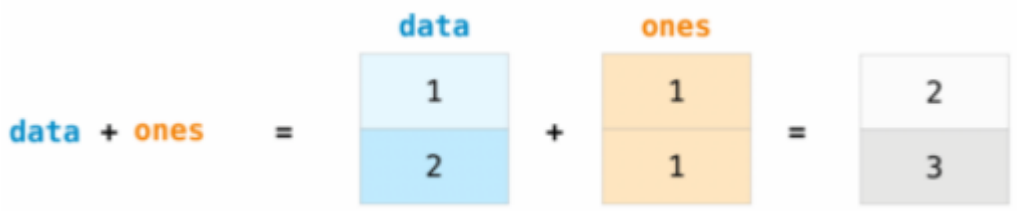
Data type	Description
bool_	Boolean (True or False) stored as a byte
int_	Default integer type (same as C long; normally either int64 or int32)
intc	Identical to C int (normally int32 or int64)
intp	Integer used for indexing (same as C ssize_t; normally either int32 or int64)
int8	Byte (-128 to 127)
int16	Integer (-32768 to 32767)
int32	Integer (-2147483648 to 2147483647)
int64	Integer (-9223372036854775808 to 9223372036854775807)
uint8	Unsigned integer (0 to 255)
uint16	Unsigned integer (0 to 65535)
uint32	Unsigned integer (0 to 4294967295)
uint64	Unsigned integer (0 to 18446744073709551615)
float_	Shorthand for float64.
float16	Half precision float: sign bit, 5 bits exponent, 10 bits mantissa
float32	Single precision float: sign bit, 8 bits exponent, 23 bits mantissa
float64	Double precision float: sign bit, 11 bits exponent, 52 bits mantissa
complex_	Shorthand for complex128.
complex64	Complex number, represented by two 32-bit floats
complex128	Complex number, represented by two 64-bit floats

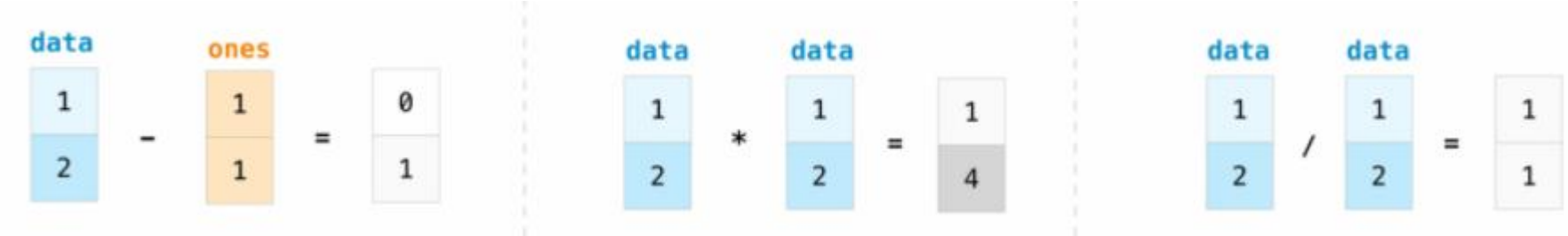
1D ARRAYS: VECTORS

Numpy.ndarray: A list with elements at a certain type



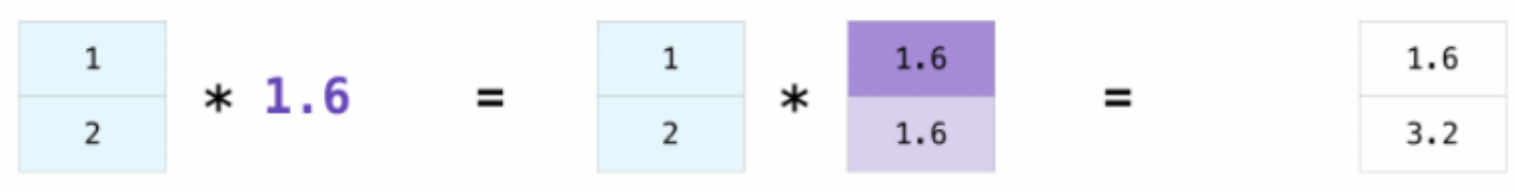
1D ARRAYS: VECTORS

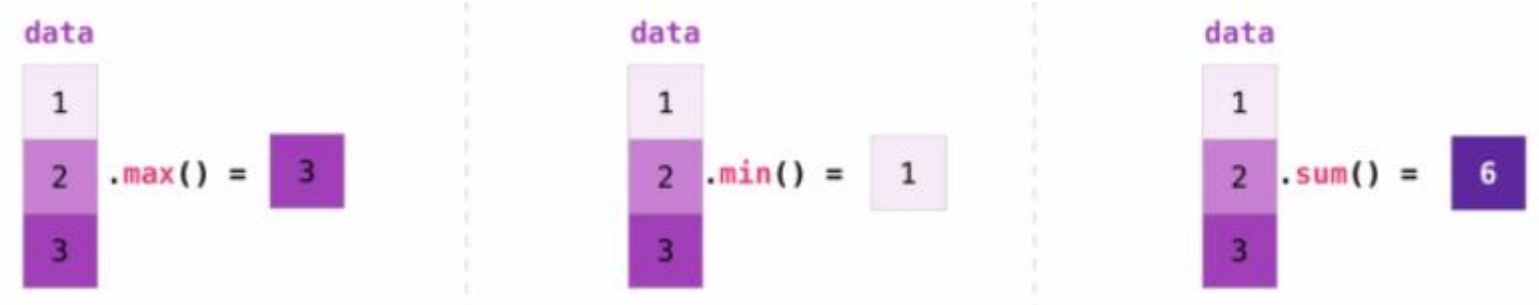




NumPy

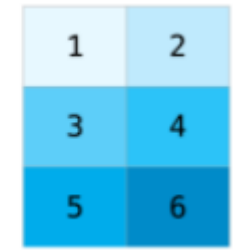
1D ARRAYS: VECTORS

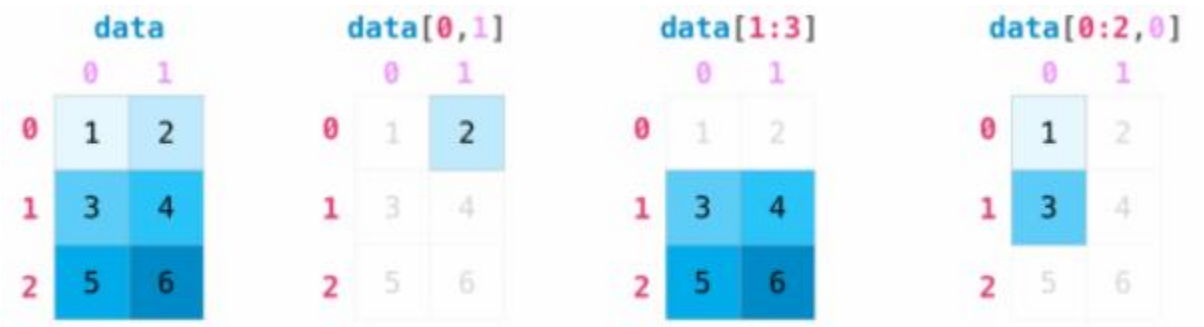




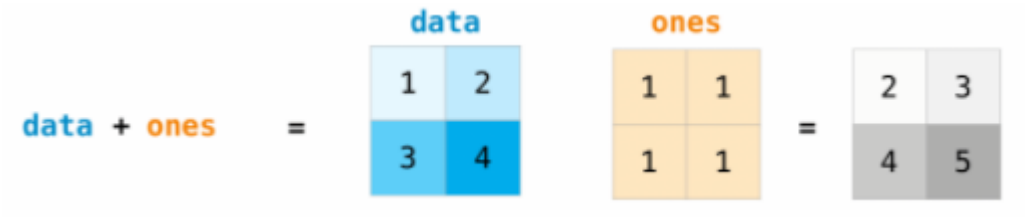
2D ARRAYS: MATRICES

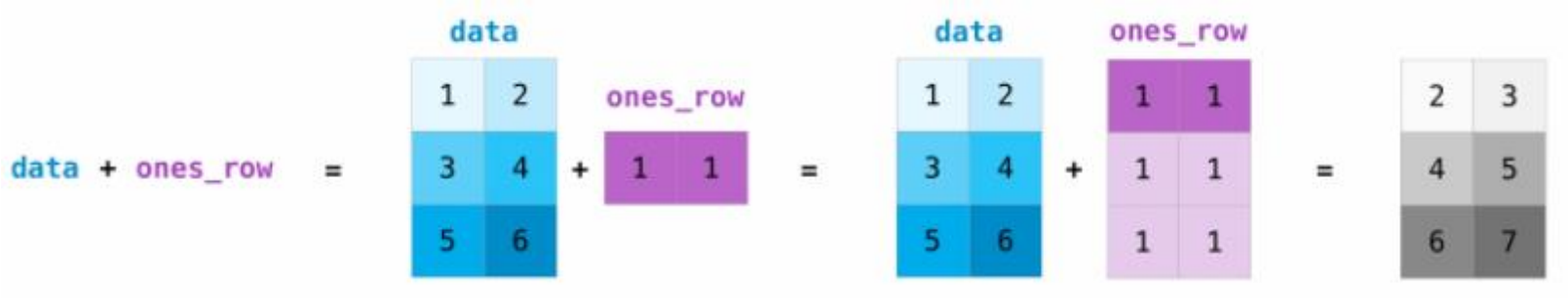
```
np.array([[1,2],[3,4],[5,6]])
```

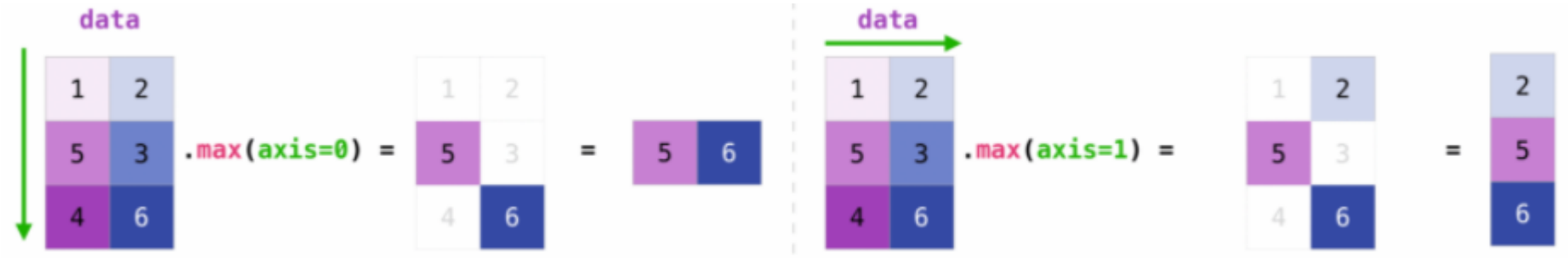




2D ARRAYS: MATRICES

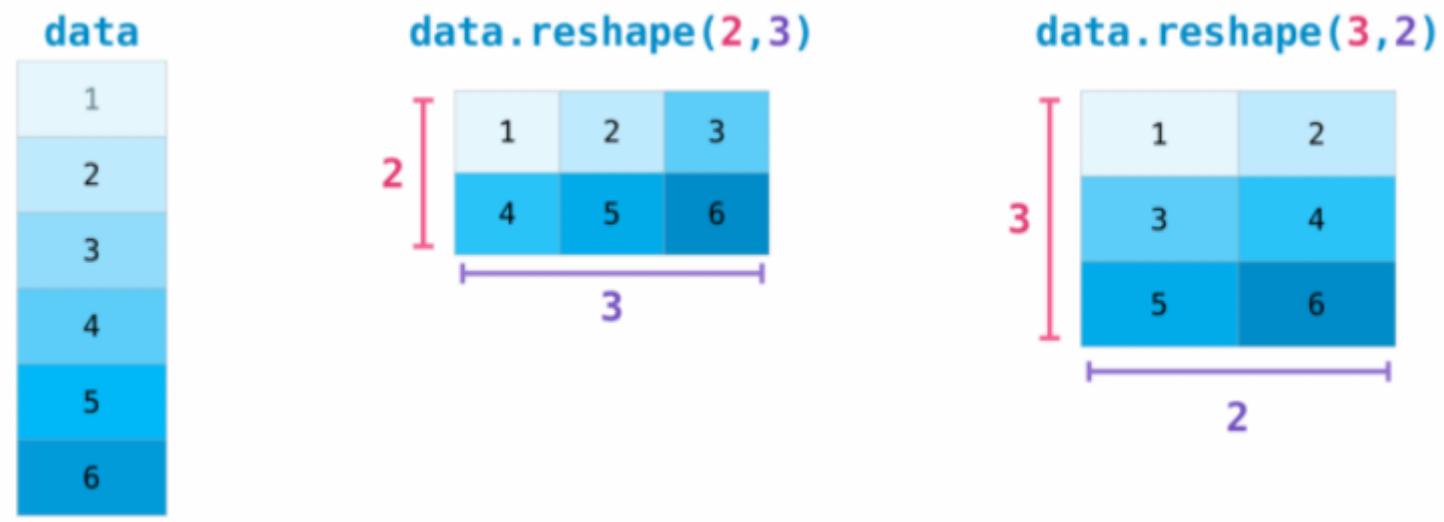




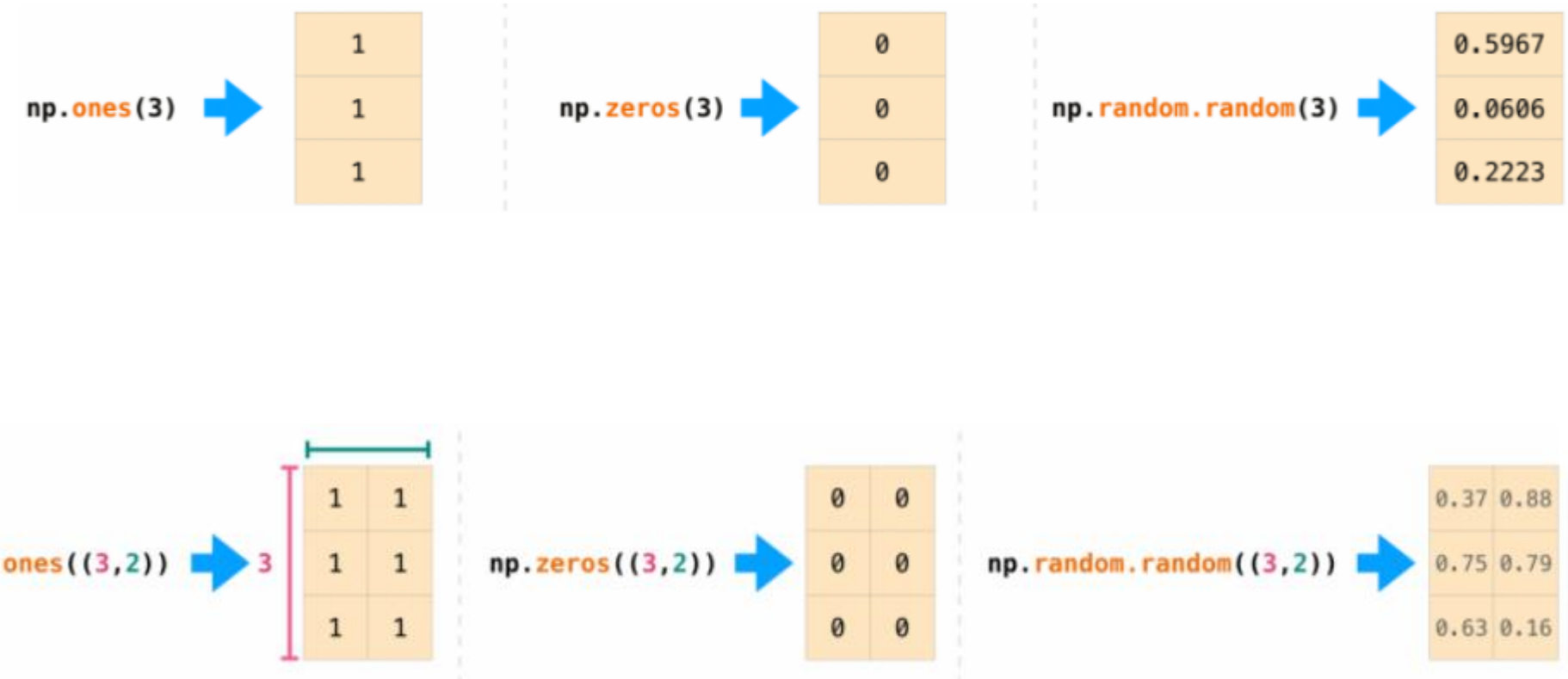


NumPy

2D ARRAYS: MATRICES



2D ARRAYS: MATRICES




pandas

PANDAS

applied analytics

Installation

Working with conda?

pandas is part of the Anaconda distribution and can be installed with Anaconda or Miniconda:

```
conda install pandas
```

Prefer pip?

pandas can be installed via pip from PyPI.

```
pip install pandas
```

In-depth instructions?

Installing a specific version? Installing from source? Check the advanced installation page.

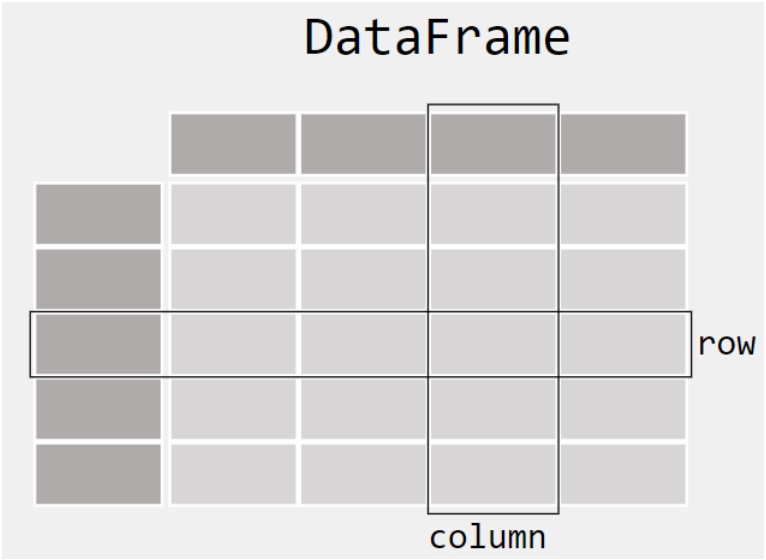
Learn more

pandas

https://pandas.pydata.org/pandas-docs/stable/getting_started/index.html#getting-started

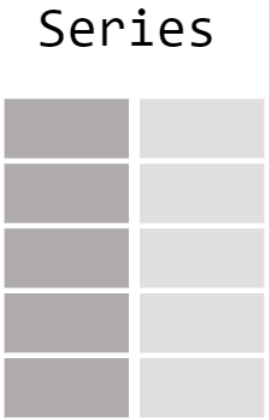
DATAFRAME and SERIES

Dimensions	Name	Description
1	Series	1D labeled homogeneously-typed array
2	DataFrame	General 2D labeled, size-mutable tabular structure with potentially heterogeneously-typed column



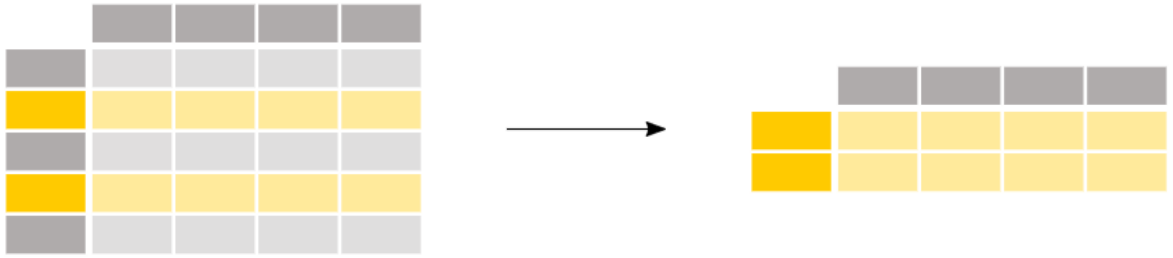
Each column in a DataFrame is a Series

- *rows are called as index*
- *columns are called as column*



SUBSETTING A DATAFRAME

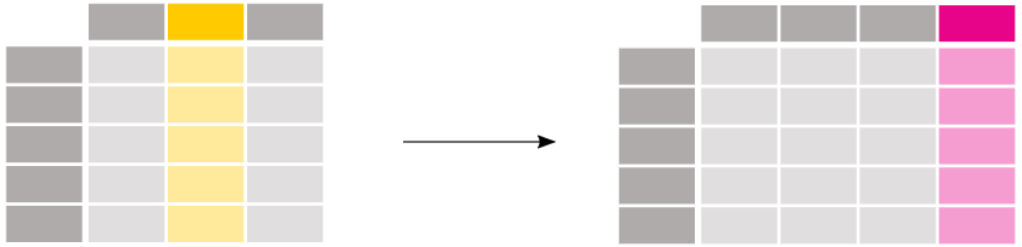
```
df[['City_ID','City_Name']]
```



```
df[df.City_ID>=2]
```

```
df.loc[df.City_ID>=2,'City_Type']

df.iloc[0:2,1:3]
```

NEW COLUMNS



```
temp3['Pop_Density']=temp3['Population']/temp3['City_Area']
```

AGGREGATIONS

```
df_city.iloc[:,1:].groupby(by="City_Type").mean()
```

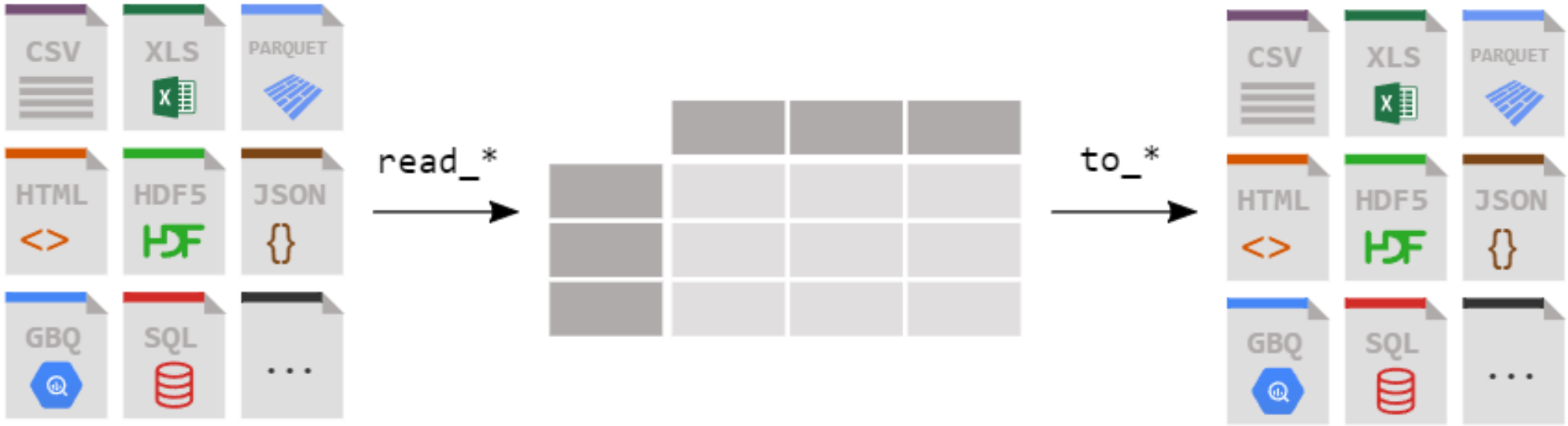
MERGE

```
temp1 = df_city[['City_ID','City_Name']]
temp  = pd.DataFrame([[101,'unknown1']],columns=['City_ID','City_Name'])
temp1 = temp1.append(temp)

temp2 = df_city[['City_ID','City_Area','Population']]
temp  = pd.DataFrame([[102,1000,2000]],columns=['City_ID','City_Area','Population'])
temp2 = temp2.append(temp)

pd.merge(temp1,temp2,on="City_ID").head()
```

JOINING DATAFRAMES BY KEYS


```
pd.merge(temp1,temp2,on="City_ID",how="inner").head()

pd.merge(temp1,temp2,on="City_ID",how="left").tail()

pd.merge(temp1,temp2,on="City_ID",how="right").tail()

pd.merge(temp1,temp2,on="City_ID",how="outer").tail()

pd.merge(temp1,temp2,how="left",left_on='City_ID',right_on='City_ID')
```

READING FROM and WRITING TO EXTERNAL SOURCES

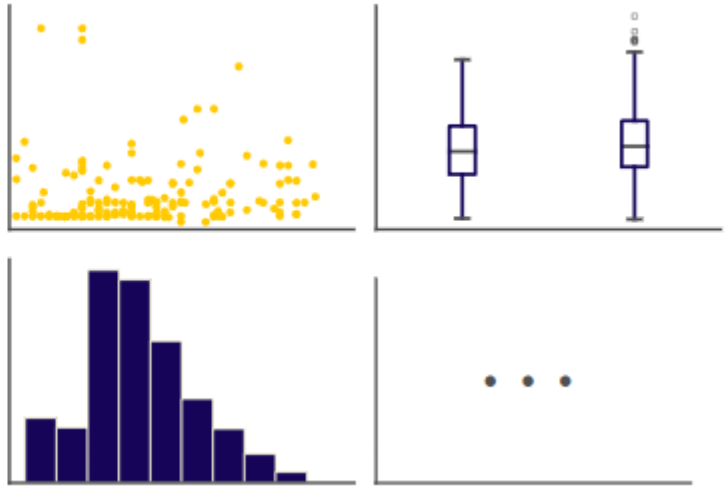


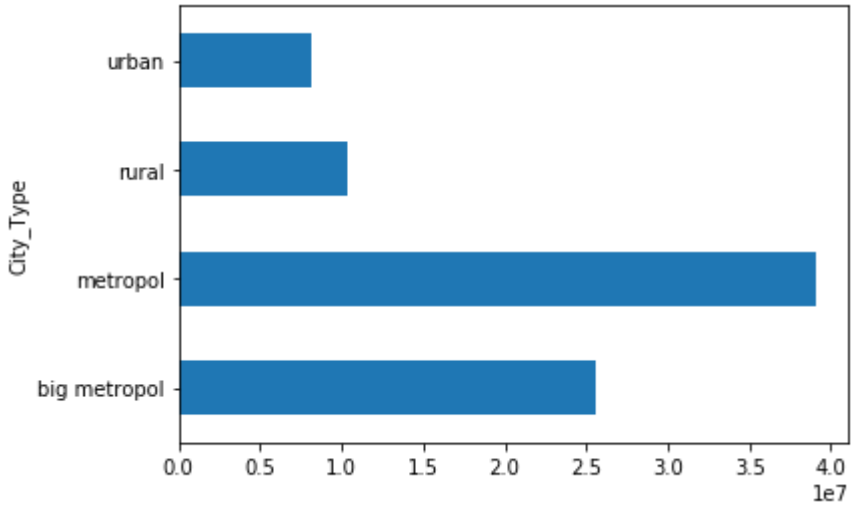

```
df = pd.read_excel("cities.xlsx", sheet_name="city list")
```

```
df.to_excel("cities.xlsx", sheet_name="city list")
```

PLOTTING WITH PANDAS

.plot.*





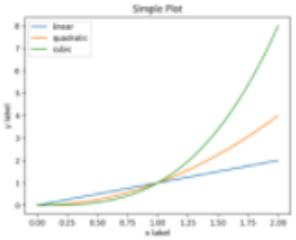

```
df_city.groupby(['City_Type'])['Population'].sum().plot(kind='barh')
```

pandas

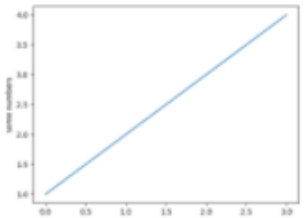

matplotlib
Version 3.4.3

Introductory

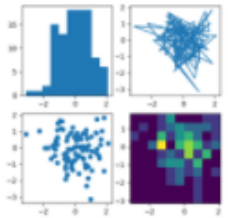
These tutorials cover the basics of creating visualizations with Matplotlib, as well as some best-practices in using the package effectively.



Usage Guide

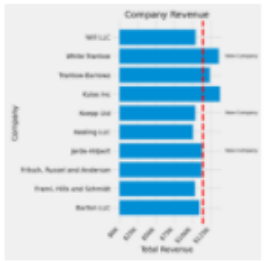


Pyplot tutorial

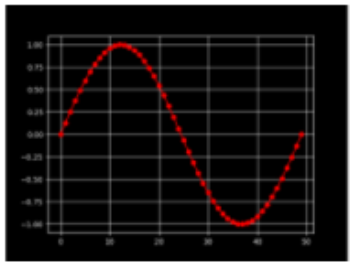


Sample plots in Matplotlib

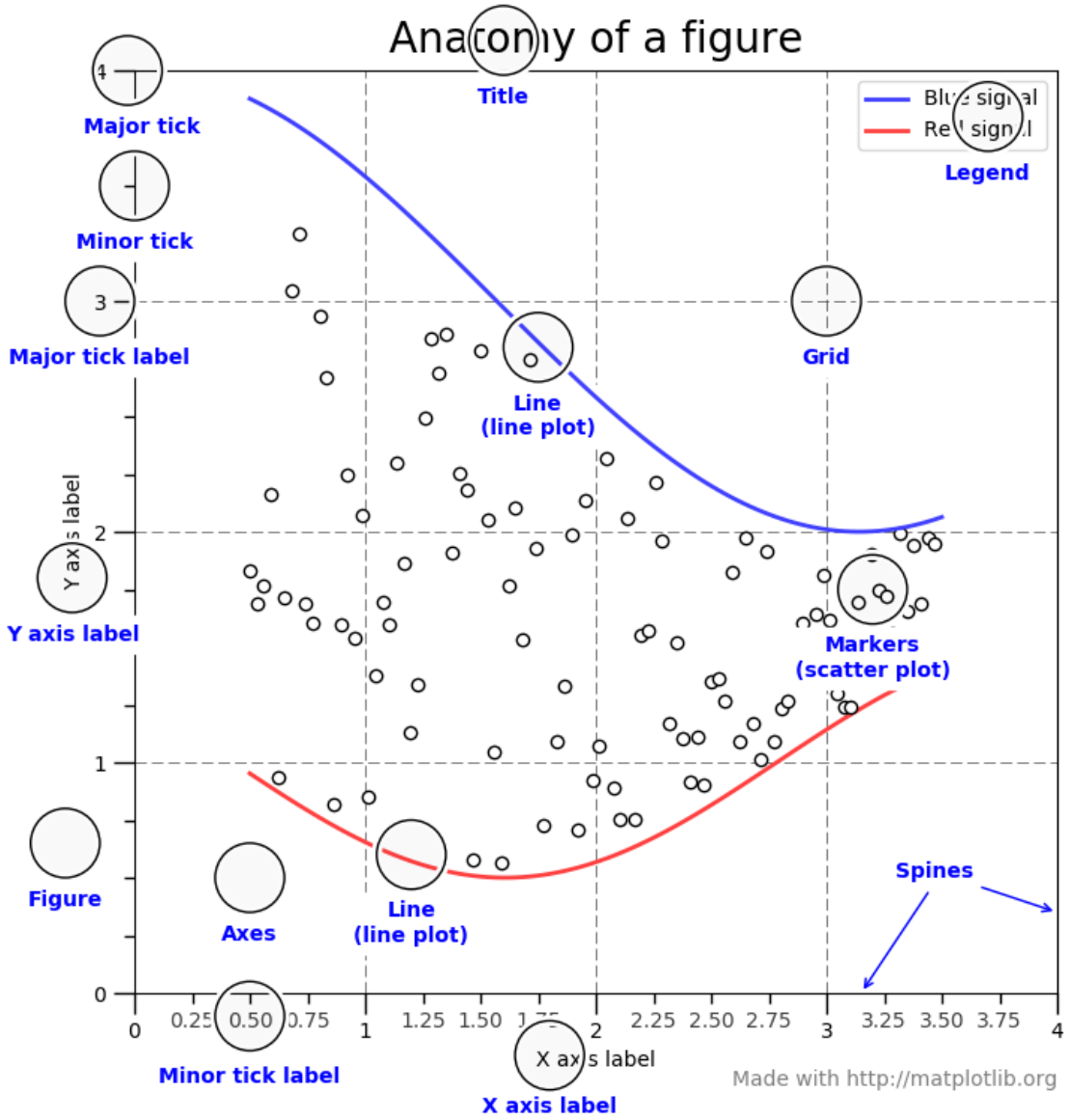
Image tutorial



The Lifecycle of a Plot



Customizing Matplotlib with style sheets and rcParams


Anatomy of a figure
Title
Major tick
Minor tick
Major tick label
Legend
Grid
Line
(line plot)
Markers
(scatter plot)
Y axis label
Figure
Axes
Line
(line plot)
Spines
Minor tick label
X axis label
Made with http://matplotlib.org

Matplotlib for beginners

Matplotlib is a library for making 2D plots in Python. It is designed with the philosophy that you should be able to create simple plots with just a few commands:

1 Initialize

```
import numpy as np
import matplotlib.pyplot as plt
```

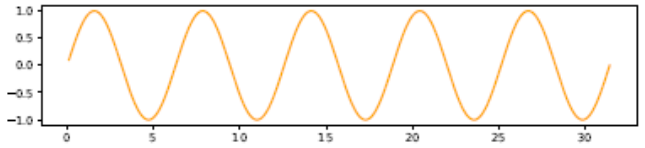
2 Prepare

```
X = np.linspace(0, 4*np.pi, 1000)
Y = np.sin(X)
```

3 Render

```
fig, ax = plt.subplots()
ax.plot(X, Y)
fig.show()
```

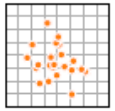
4 Observe



Choose

Matplotlib offers several kind of plots (see Gallery):

```
X = np.random.uniform(0, 1, 100)
Y = np.random.uniform(0, 1, 100)
ax.scatter(X, Y)
```



```
X = np.arange(10)
Y = np.random.uniform(1, 10, 10)
ax.bar(X, Y)
```

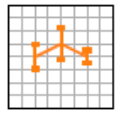
```
Z = np.random.uniform(0, 1, (8,8))

ax.imshow(Z)
```

```
Z = np.random.uniform(0, 1, (8,8))

ax.contourf(Z)
```

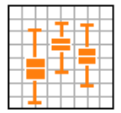
```
Z = np.random.uniform(0, 1, 4)

ax.pie(Z)
```

```
Z = np.random.normal(0, 1, 100)

ax.hist(Z)
```

```
X = np.arange(5)
Y = np.random.uniform(0, 1, 5)
ax.errorbar(X, Y, Y/4)
```



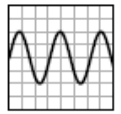
```
Z = np.random.normal(0, 1, (100,3))

ax.boxplot(Z)
```



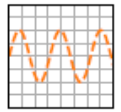
Tweak

You can modify pretty much anything in a plot, including limits, colors, markers, line width and styles, ticks and ticks labels, titles, etc.

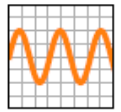
```
X = np.linspace(0, 10, 100)
Y = np.sin(X)
ax.plot(X, Y, color="black")
```



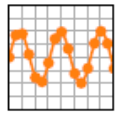
```
X = np.linspace(0, 10, 100)
Y = np.sin(X)
ax.plot(X, Y, linestyle="--")
```



```
X = np.linspace(0, 10, 100)
Y = np.sin(X)
ax.plot(X, Y, linewidth=5)
```



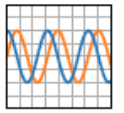
```
X = np.linspace(0, 10, 100)
Y = np.sin(X)
ax.plot(X, Y, marker="o")
```



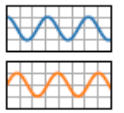
Organize

You can plot several data on the the same figure, but you can also split a figure in several subplots (named *Axes*):

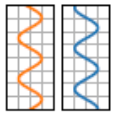
```
X = np.linspace(0, 10, 100)
Y1, Y2 = np.sin(X), np.cos(X)
ax.plot(X, Y1, X, Y2)
```



```
fig, (ax1, ax2) = plt.subplots((2,1))
ax1.plot(X, Y1, color="C1")
ax2.plot(X, Y2, color="C0")
```

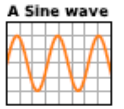


```
fig, (ax1, ax2) = plt.subplots((1,2))
ax1.plot(Y1, X, color="C1")
ax2.plot(Y2, X, color="C0")
```

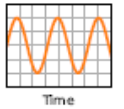


Label (everything)

```
ax.plot(X, Y)
fig.suptitle(None)
ax.set_title("A Sine wave")
```




```
ax.plot(X, Y)
ax.set_ylabel(None)
ax.set_xlabel("Time")
```



Explore

Figures are shown with a graphical user interface that allows to zoom and pan the figure, to navigate between the different views and to show the value under the mouse.

Save (bitmap or vector format)

```
fig.savefig("my-first-figure.png", dpi=300)
fig.savefig("my-first-figure.pdf")
```

WHAT WE HAVE LEARNED AT THIS LECTURE?

INSPIRATION

REASONING

INTUITION

PRINCIPLE

PRACTICE

CONCLUSION

MATH

THINKING

CODING SKILL

FORMULA

USE CASE

THEORY

IDEA