**Dictionnaires**

Run the following cell

Entrée [19]:

a **=** {

'key1': 1,

'key2': 'hello',

'name': 'john'

}

​

print(a)

print(a['name'])

print(a.keys())

print(a.values())

{'key1': 1, 'key2': 'hello', 'name': 'john'}

john

dict\_keys(['key1', 'key2', 'name'])

dict\_values([1, 'hello', 'john'])

What does it ? What is the goal of the keys and values methods ?

Réponse : It prints all values that are in "a". The goal is to sort values and get them with a special value that is the "key".

With the synthax a['key\_name'] add the "last\_name" key to the dictionnary with an arbitrary value. Check that the new key is indeed in the dictionnary.

Entrée [9]:

a['last\_name']**=**"arbitrary value"

a

Out[9]:

{'key1': 1, 'key2': 'hello', 'name': 'john', 'last\_name': 'arbitrary value'}

Update the key1 key with the value 42 and check it has been updated

Entrée [14]:

a['key1'] **=** 42

a

Out[14]:

{'key1': 42, 'key2': 'hello', 'name': 'john'}

Check on internet how to delete a key of a dictionnary and delete the name entry of the dictionnary

Entrée [20]:

**del** a['name']

*# a.pop('name')*

a

Out[20]:

{'key1': 1, 'key2': 'hello'}

Create a second dictionnary b with any values. Merge the dictionnaries a and b (Look on the internet on how to do it)

Entrée [23]:

b **=** {'a': 1, 'b': 2, 'c': 3}

a.update(b)

a

Out[23]:

{'key1': 1, 'key2': 'hello', 'a': 1, 'b': 2, 'c': 3}

Create a list k = ['a', 'b', 'c', 'd'] and v = [22, 11, 44, 33] From the list k and v create a dictionnary where the keys will be the value of *k\* and the values will be the values of \*v*

Entrée [25]:

k **=** ['a', 'b', 'c', 'd']

v **=** [22, 11, 44, 33]

dictionary **=** dict(zip(k, v))

dictionary

Out[25]:

{'a': 22, 'b': 11, 'c': 44, 'd': 33}

**Introduction to pandas**

Entrée [2]:

**import** pandas **as** pd

Load with pandas the file houses.csv with the method "read\_csv" into varaible **df**

Entrée [3]:

df **=** pd.read\_csv("./house.csv")

Print the type of **df**

Entrée [9]:

print(type(df))

<class 'pandas.core.frame.DataFrame'>

Use the describe method on the df varaible. What does it tell you ?

Entrée [35]:

pd.DataFrame.describe(df)

df.describe()

Out[35]:

|  | **size** | **nb\_rooms** | **garden** | **price** |
| --- | --- | --- | --- | --- |
| **count** | 950.000000 | 950.000000 | 950.000000 | 9.500000e+02 |
| **mean** | 151.679733 | 1.972632 | 0.502105 | 1.517604e+06 |
| **std** | 49.322384 | 0.816898 | 0.500259 | 4.932795e+05 |
| **min** | -4.079018 | 1.000000 | 0.000000 | -3.770230e+04 |
| **25%** | 116.864214 | 1.000000 | 0.000000 | 1.169006e+06 |
| **50%** | 152.112430 | 2.000000 | 1.000000 | 1.522411e+06 |
| **75%** | 185.684296 | 3.000000 | 1.000000 | 1.857463e+06 |
| **max** | 295.630840 | 3.000000 | 1.000000 | 2.953889e+06 |

The the method head on the dataframe

Entrée [36]:

pd.DataFrame.head(df)

df.head()

Out[36]:

Display the columns of the dataframe

Entrée [8]:

df.columns

Out[8]:

Index(['size', 'nb\_rooms', 'garden', 'orientation', 'price'], dtype='object')

With a mask display all the houses whose size is above 80

Entrée [14]:

above\_80 **=** (df["size"].values **>** 80)

df[above\_80]

Out[14]:

With a mask display all the houses whose size are above 100 and have more the 2 rooms

Entrée [28]:

above\_100\_more\_2 **=** (df["size"] **>** 100) **&** (df["nb\_rooms"] **>** 2)

df[above\_100\_more\_2]

Out[28]:

259 rows × 5 columns

Search how to do a histogram plot on a given column with pandas and plot the histogram of the size and then the prices

Entrée [31]:

**import** matplotlib.pyplot **as** plt

Entrée [37]:

plt.hist(df["size"])

df.hist(column**=**"size")

Out[37]:

array([[<matplotlib.axes.\_subplots.AxesSubplot object at 0x11fe17748>]],

dtype=object)

Entrée [38]:

plt.hist(df["price"])

df.hist(column**=**"price")

Out[38]:

array([[<matplotlib.axes.\_subplots.AxesSubplot object at 0x11f8e0ac8>]],

dtype=object)

# Numpy

Numpy (numerical py) is a library to do scientific computing : it allows to manipulate vector, matrices and so on in an efficient way. In this notebook we are going to study it a bit.

But before let's see again how basic python list works.  
Create two list v1 and v2 with the values 𝑣1=[1,2,3]v1=[1,2,3] and 𝑣2=[2,1,1]v2=[2,1,1]  
display v1 + v2

Entrée [1]:

v1 **=** [1,2,3]

v2 **=** [2,1,1]

print(v1**+**v2)

[1, 2, 3, 2, 1, 1]

With a for loop and the zip function compute the sum of v1 and v2 element by element and store it in v3

Entrée [2]:

v3 **=** [sum(i) **for** i **in** zip(v1, v2)]

print(v3)

[3, 3, 4]

Pretty complicated, don't you think ? Having to do a for loop each time you want to add vector is pretty cumbersome. Numpy will ease that

Entrée [3]:

**import** numpy **as** np

with the np.array function create a vector with values 𝑣1=(1,2,3,4)v1=(1,2,3,4) from a list

Entrée [5]:

v1 **=** np.array((1,2,3,4))

In numpy, arithmetic operations (+, \*, etc.) are done element by element. Create a second vector v2 with the same size as v1 (any value). Then add v1 and v2

Entrée [8]:

v2 **=** np.array((4,3,2,1))

v3 **=** v1 **+** v2

print(v3)

[5 5 5 5]

## Les masks

Masks allow to filter data withing numpy arrays.

Let's look at the following cell : execute it.

Entrée [9]:

v **=** np.array([x **for** x **in** range(20)])

​

even\_mask **=** (v **%** 2 **==** 0)

print(even\_mask)

print(v[even\_mask])

[ True False True False True False True False True False True False

True False True False True False True False]

[ 0 2 4 6 8 10 12 14 16 18]

What did it do ?

Ca affiche les indices de tout les element pairs

By using a mask, get all values above 10 of the vector v and put it in the variable above\_10

Entrée [13]:

above\_ten\_mask **=** (v **>** 10)

print(v[above\_ten\_mask])

[11 12 13 14 15 16 17 18 19]

With the function mean of numpy compute the average of all even numbers of v. Then compute the average of all odd numbers of v.

Entrée [15]:

print(np.mean(v[even\_mask]))

odd\_mask **=** (v **%** 2 **!=** 0)

print(np.mean(v[odd\_mask]))

9.0

10.0

**A small linear regression with scikit learn**

Entrée [1]:

**import** pandas **as** pd

**import** matplotlib.pyplot **as** plt

**import** seaborn **as** sns

**import** numpy **as** np

**import** math

​

**from** sklearn.linear\_model **import** LinearRegression

**from** sklearn **import** metrics

Load the dataset house.csv with pandas into the variable **df**

Entrée [2]:

df **=** pd.read\_csv("./house.csv")

Check on the internet how to plot the correlations between variables and plot them

Entrée [5]:

*# pd.scatter\_matrix(df, figsize=(6, 6))*

sns.pairplot(df)

plt.show()

In the linear model module of scikit learn import the LinearRegression module and instanciate a LinearRegression object in the **model** variable

Entrée [4]:

model **=** LinearRegression()

Select the column containing the number the size of the house and put it into a variable **X\*\* (use double bracket to select the column).  
Select the column price and put it in the variable \*\*y**

Entrée [7]:

X **=** df[['size']]

y **=** df[['price']]

Use the fit method on X and y of the **model** variable. It will automatically find the best parameters of the linear regression

Entrée [8]:

model.fit(X, y)

Out[8]:

LinearRegression(copy\_X=True, fit\_intercept=True, n\_jobs=None,

normalize=False)

Create a numpy array with values from 0 to 300 evenly spaced with the numpy function arange

Entrée [12]:

sizes **=** np.arange(300).reshape(300,1)

Compute the price associated to each of the values of the previous array.

Entrée [10]:

prediction\_price **=** model.predict(sizes)

Plot both the scatter plot of the houses (price versus size) and the predicted prices versus the sizes

Entrée [11]:

plt.scatter(sizes, prediction\_price, c**=**'red', marker**=**'+')

plt.plot(X,y)

plt.show()

Display the coefficients of the linear regression (look at the linear\_regression attributes ) in the doc

Entrée [78]:

model.coef\_, model.intercept\_

Out[78]:

(array([[10001.00319699]]), array([654.99477315]))

Display the MSE. Display the R2 coefficient (what is it) ?

Entrée [77]:

y\_test **=** model.predict(X)

mse **=** metrics.mean\_squared\_error(y, y\_test)

math.sqrt(mse)

Out[77]:

2475.519190271693

Entrée [74]:

metrics.r2\_score(y,y\_test)

Out[74]:

0.9999747882044046

## Numpy come back

Entrée [1]:

**import** numpy **as** np

## Masks revisions

With the randn method of numpy create a vector with 100 values

Entrée [2]:

v **=** np.random.randn(100)

With a mask, select all values above 0.1 and store it in a vector v2

Entrée [3]:

v2 **=** v[ v **>** 0.1 ]

with a mask select all values above 0.1 or below -0.1 and store it in a vector v3

Entrée [4]:

v3 **=** v[(v **>** 0.1) **|** (v **<** **-**0.1) ]

## Vector shapes

Reshape the vector in a 10x10 matrix

Entrée [5]:

v.reshape(10, 10)

Out[5]:

reshape the matrix into a 50x2 matrix

Entrée [6]:

v.reshape(50,2)

*# v.reshape(50,-1)*

Out[6]:

Reshape the matrix into a 2x5x10 tensor (3D array)

Entrée [7]:

v **=** v.reshape(2,5,10)

Compute the mean along the third axis of the 3D tensor

Entrée [15]:

np.mean(v, axis**=**2)

Out[15]:

array([[ 0.35541799, -0.33819551, 0.13635708, -0.03407223, -0.07807166],

[-0.08467603, 0.23631526, 0.49757725, -0.05938357, -0.08764337]])

## Pandas come back

load the house dataset in a dataframe

Entrée [9]:

**import** pandas **as** pd

Entrée [10]:

df **=** pd.read\_csv("../TP1/house.csv")

df

Out[10]:

With the groupby function, compute the average price of houses per number of bed rooms

Entrée [18]:

df.groupby('nb\_rooms').agg(np.mean)[['price']]

Out[18]:

Entrée [19]:

df.groupby('garden').agg(np.mean)[['price']]

Out[19]:

Compute the median price by garden

Entrée [20]:

df.groupby('garden').agg(np.median)[['price']]

Out[20]:

Select the price column and put it in a variable s. Select the columns "bed rooms" and "price" in one way and put it a variable v. Print the type and shape of v and s

Entrée [21]:

v **=** df[['nb\_rooms', 'price']]

s **=** df['price']

v.shape, s.shape

Out[21]:

((950, 2), (950,))