

Lab 4: Pandas

The objective of this notebook is to learn about the **Pandas** library ([official documentation](#)). You can find a good guide at this [link](#).

Outline

- [1. Pandas Series](#)
- [2. Pandas DataFrames](#)
- [3. Computation with Pandas](#)

First, run the following cell to import some useful libraries to complete this Lab. If not already done, you must install them in your virtual environment

```
In [1]: import pandas as pd
import numpy as np
import random
```

If the previous cell outputs one the following error: `ModuleNotFoundError: No module named 'pandas'`, then, you have to install the Pandas package. If you don't remember how to install a Python package, please retrieve the guide on Anaconda-Navigator.

To install **pandas** you can use one of the following commands from the terminal of your virtual environment:

```
conda install pandas
pip install pandas
```

1. Pandas Series

Exercise 1.1

Create a **Pandas Series** from the following list `[1.0, 5.0, 0.5, 4.1]` and **index** `["person 1", "person 2", "person 3", "person 4"]`. Then, print the **Series**.

```
In [2]: ##### START CODE HERE (~2 lines) #####
my_series = pd.Series([1.0, 5.0, 0.5, 4.1], index=["person 1", "person 2", "person 3", "person 4"])
print(my_series)
##### END CODE HERE #####
```

```
person 1    1.0
person 2    5.0
person 3    0.5
person 4    4.1
dtype: float64
```

Expected output

```
person 1    1.0
person 2    5.0
person 3    0.5
person 4    4.1
dtype: float64
```

Exercise 1.2

Create a **Pandas Series** from the following dictionary `{"person 1": 1.0, "person 2": 5.0, "person 3": 0.5, "person 4": 4.1}`. Then, print the **Series**.

```
In [3]: ##### START CODE HERE (~2 lines) #####
my_series = pd.Series({"person 1": 1.0, "person 2": 5.0, "person 3": 0.5, "person 4": 4.1})
print(my_series)
##### END CODE HERE #####
```

```
person 1    1.0
person 2    5.0
person 3    0.5
person 4    4.1
dtype: float64
```

Expected output

```
person 1    1.0
person 2    5.0
person 3    0.5
person 4    4.1
dtype: float64
```

You can see that the two methods of creating the Series are **equivalent** and produce the same output.

Exercise 1.3

Put **all the elements** of the Series `s1` into a variable `s1_values`. Then print the **value** of `s1_values` and its **type**.

```
In [4]: s1 = pd.Series([1.0, 5.0, 0.5, 4.1])
```

```
#### START CODE HERE (~3 lines) ####
s1_values = s1.values
print(s1_values)
print(type(s1_values))
#### END CODE HERE ####
```

```
[1.  5.  0.5 4.1]
<class 'numpy.ndarray'>
```

Expected output

```
[1.  5.  0.5 4.1]
<class 'numpy.ndarray'>
```

You can see that the **values** attribute of a Series returns a **Numpy array**.

Exercise 1.4

Access the value of the Series `s1` with the **index in position 1** (i.e., the second element) and print its **value**.

► Hints

```
In [5]: s1 = pd.Series({"person 1": 1.0, "person 2": 5.0, "person 3": 0.5, "person 4": 4.1})
```

```
#### START CODE HERE (~1 line) ####
print(s1.iloc[1])
#### END CODE HERE ####
```

```
5.0
```

Expected output

```
5.0
```

Exercise 1.5

Access the value of the Series `s1` with the **explicit index** equal to `"person 3"` and print its **value**.

► Hints

```
In [6]: s1 = pd.Series({"person 1": 1.0, "person 2": 5.0, "person 3": 0.5, "person 4": 4.1})
```

```
#### START CODE HERE (~1 line) ####
print(s1.loc["person 3"])
#### END CODE HERE ####
```

```
0.5
```

Expected output

```
0.5
```

Exercise 1.6

Access and **print** the **slice** of the Series `s1` from the **second** element until the **fourth**, both included.

► Hints

```
In [7]: s1 = pd.Series({"person 1": 1.0, "person 2": 5.0, "person 3": 0.5, "person 4": 4.1})
```

```
#### START CODE HERE (~1 line) ####
print(s1.iloc[1:4])
#### END CODE HERE ####
```

```
person 2    5.0
person 3    0.5
person 4    4.1
dtype: float64
```

Expected output

```
person 2    5.0
person 3    0.5
```

```
person 4    4.1
dtype: float64
```

Exercise 1.7

Access and print the slice of the Series `s1` from index "person 2" to "person 4", both included.

► Hints

```
In [8]: s1 = pd.Series({"person 1": 1.0, "person 2": 5.0, "person 3": 0.5, "person 4": 4.1})
```

```
#### START CODE HERE (~1 line) ####
print(s1.loc["person 2":"person 4"])
#### END CODE HERE ####
```

```
person 2    5.0
person 3    0.5
person 4    4.1
dtype: float64
```

Expected output

```
person 2    5.0
person 3    0.5
person 4    4.1
dtype: float64
```

Exercise 1.8

Define a **mask** into a variable `my_mask` starting from the Series `s1` with `True` for the elements that are **strictly greater than 1**, `False` otherwise. Then, **print** the **mask** and the **elements of the Series where the mask is `True`**.

► Hints

```
In [9]: s1 = pd.Series({"person 1": 1.0, "person 2": 5.0, "person 3": 0.5, "person 4": 4.1})
```

```
#### START CODE HERE (~3 lines) ####
my_mask = s1 > 1
print(my_mask)
print(s1[my_mask])
#### END CODE HERE ####
```

```
person 1    False
person 2     True
person 3    False
person 4     True
dtype: bool
person 2    5.0
person 4    4.1
dtype: float64
```

Expected output

```
person 1    False
person 2     True
person 3    False
person 4     True
dtype: float64
person 2    5.0
person 4    4.1
dtype: float64
```

Exercise 1.9

Use **Fancy Indexing** with **Explicit Indexing** to access and print the values of the Series `s1` with Index "person 2" and "person 4"

► Hints

```
In [10]: s1 = pd.Series({"person 1": 1.0, "person 2": 5.0, "person 3": 0.5, "person 4": 4.1})
```

```
#### START CODE HERE (~1 line) ####
print(s1.loc[["person 2", "person 4"]])
#### END CODE HERE ####
```

```
person 2    5.0
person 4    4.1
dtype: float64
```

Expected output

```

person 2    5.0
person 4    4.1
dtype: float64

```

2. Pandas DataFrames

Exercise 2.1

Create a **Pandas DataFrame** into a variable `df` from the series `age_s`, `gender_s`, and `country_s`, with column names `"age"`, `"gender"`, and `"country"`, respectively. Then **print** the DataFrame `df`.

```

In [11]: age_s = pd.Series({"person 1": 21, "person 2": 25, "person 3": 34, "person 4": 27})
gender_s = pd.Series({"person 1": "M", "person 2": "F", "person 3": "M", "person 4": "F"})
country_s = pd.Series({"person 1": "Italy", "person 2": "United Kingdom", "person 3": "France", "person 4": "Spain"})

#### START CODE HERE (~2 lines) ####
df = pd.DataFrame({"age": age_s, "gender": gender_s, "country": country_s})
print(df)
#### END CODE HERE ####

```

```

      age gender      country
person 1   21     M         Italy
person 2   25     F  United Kingdom
person 3   34     M         France
person 4   27     F         Spain

```

Expected output

```

age gender      country
person 1   21     M         Italy
person 2   25     F  United Kingdom
person 3   34     M         France
person 4   27     F         Spain

```

Exercise 2.2

Create a **Pandas DataFrame** into a variable `df` from the lists `age_list`, `gender_list`, and `country_list`, with column names `"age"`, `"gender"`, and `"country"`, respectively. The Index is specified in `index_list`. Then **print** the DataFrame `df`.

```

In [12]: age_list = [21, 25, 34, 27]
gender_list = ["M", "F", "M", "F"]
country_list = ["Italy", "United Kingdom", "France", "Spain"]
index_list = ["person 1", "person 2", "person 3", "person 4"]

#### START CODE HERE (~2 lines) ####
df = pd.DataFrame({"age": age_list, "gender": gender_list, "country": country_list}, index=index_list)
print(df)
#### END CODE HERE ####

```

```

      age gender      country
person 1   21     M         Italy
person 2   25     F  United Kingdom
person 3   34     M         France
person 4   27     F         Spain

```

Expected output

```

age gender      country
person 1   21     M         Italy
person 2   25     F  United Kingdom
person 3   34     M         France
person 4   27     F         Spain

```

You can see that the two methods of creating the DataFrames are **equivalent** and produce the same output. Another equivalent method is the following:

```

In [13]: data_list = [{"age": 21, "gender": "M", "country": "Italy"},
                    {"age": 25, "gender": "F", "country": "United Kingdom"},
                    {"age": 34, "gender": "M", "country": "France"},
                    {"age": 27, "gender": "F", "country": "Spain"}]

index_list = ["person 1", "person 2", "person 3", "person 4"]

df = pd.DataFrame(data_list, index=index_list)
print(df)

```

	age	gender	country
person 1	21	M	Italy
person 2	25	F	United Kingdom
person 3	34	M	France
person 4	27	F	Spain

Exercise 2.3

Access and print the column "country" of the DataFrame `df`.

```
In [14]: data_list = [{"age": 21, "gender": "M", "country": "Italy"},
                    {"age": 25, "gender": "F", "country": "United Kingdom"},
                    {"age": 34, "gender": "M", "country": "France"},
                    {"age": 27, "gender": "F", "country": "Spain"}]

index_list = ["person 1", "person 2", "person 3", "person 4"]

df = pd.DataFrame(data_list, index=index_list)

#### START CODE HERE (~1 line) ####
print(df["country"])
#### END CODE HERE ####
```

```
person 1      Italy
person 2  United Kingdom
person 3      France
person 4      Spain
Name: country, dtype: object
```

Expected output

```
person 1      Italy
person 2  United Kingdom
person 3      France
person 4      Spain
Name: country, dtype: object
```

When you access a single column, you can see that a Series is returned.

Exercise 2.4

Access and print the row corresponding to the index "person 2" of the DataFrame `df` (with explicit indexing).

► Hints

```
In [15]: data_list = [{"age": 21, "gender": "M", "country": "Italy"},
                    {"age": 25, "gender": "F", "country": "United Kingdom"},
                    {"age": 34, "gender": "M", "country": "France"},
                    {"age": 27, "gender": "F", "country": "Spain"}]

index_list = ["person 1", "person 2", "person 3", "person 4"]

df = pd.DataFrame(data_list, index=index_list)

#### START CODE HERE (~1 line) ####
print(df.loc["person 2"])
#### END CODE HERE ####
```

```
age          25
gender       F
country  United Kingdom
Name: person 2, dtype: object
```

Expected output

```
age          25
gender       F
country  United Kingdom
Name: person 2, dtype: object
```

You can see that this time, the is returned a Series where the index is replaced with the name of the columns.

Exercise 2.5

Access the slice of rows from index 1 until the end of the DataFrame `df` (both included) and print those rows (with implicit indexing).

► Hints

```
In [16]: data_list = [{"age": 21, "gender": "M", "country": "Italy"},
                    {"age": 25, "gender": "F", "country": "United Kingdom"},
                    {"age": 34, "gender": "M", "country": "France"},
                    {"age": 27, "gender": "F", "country": "Spain"}]
```

```

index_list = ["person 1", "person 2", "person 3", "person 4"]

df = pd.DataFrame(data_list, index=index_list)

#### START CODE HERE (~1 line) ####
print(df.iloc[1:])
#### END CODE HERE ####

```

```

      age gender      country
person 2  25     F  United Kingdom
person 3  34     M           France
person 4  27     F           Spain

```

Expected output

```

age gender      country
person 2  25     F  United Kingdom
person 3  34     M           France
person 4  27     F           Spain

```

You can see that this time, it is returned a DataFrame.

Exercise 2.6

Access the **slice** of the DataFrame `df` (with **explicit indexing**) containing rows from `person 2` to `person 4` (both included), and columns from `gender` to `country` (both included), and **print the slice**.

► Hints

```

In [17]: data_list = [{"age": 21, "gender": "M", "country": "Italy"},
                    {"age": 25, "gender": "F", "country": "United Kingdom"},
                    {"age": 34, "gender": "M", "country": "France"},
                    {"age": 27, "gender": "F", "country": "Spain"}]

index_list = ["person 1", "person 2", "person 3", "person 4"]

df = pd.DataFrame(data_list, index=index_list)

#### START CODE HERE (~1 line) ####
print(df.loc["person 2": "person 4", "gender":"country"])
#### END CODE HERE ####

```

```

      gender      country
person 2     F  United Kingdom
person 3     M           France
person 4     F           Spain

```

Expected output

```

gender      country
person 2     F  United Kingdom
person 3     M           France
person 4     F           Spain

```

Exercise 2.7

Access and **print** the **columns** `gender` and `country` of the DataFrame `df` for **all the rows** with **gender equal to** `F`.

► Hints

```

In [18]: data_list = [{"age": 21, "gender": "M", "country": "Italy"},
                    {"age": 25, "gender": "F", "country": "United Kingdom"},
                    {"age": 34, "gender": "M", "country": "France"},
                    {"age": 27, "gender": "F", "country": "Spain"}]

index_list = ["person 1", "person 2", "person 3", "person 4"]

df = pd.DataFrame(data_list, index=index_list)

#### START CODE HERE (~1 line) ####
print(df.loc[df['gender'] == 'F', ['gender', 'country']])
#### END CODE HERE ####

```

```

      gender      country
person 2     F  United Kingdom
person 4     F           Spain

```

Expected output

```

gender      country
person 2     F  United Kingdom
person 4     F           Spain

```

Exercise 2.8

Add a new column `working hours` with the values in the following list `['part time', 'full time', 'full time', 'full time']`.

```
In [19]: data_list = [{"age": 21, "gender": "M", "country": "Italy"},
                    {"age": 25, "gender": "F", "country": "United Kingdom"},
                    {"age": 34, "gender": "M", "country": "France"},
                    {"age": 27, "gender": "F", "country": "Spain"}]

index_list = ["person 1", "person 2", "person 3", "person 4"]

df = pd.DataFrame(data_list, index=index_list)

#### START CODE HERE (~1 line) ####
df['working hours'] = ['part time', 'full time', 'full time', 'full time']
#### END CODE HERE ####

print(df)
```

```
   age gender  country working hours
person 1   21     M      Italy    part time
person 2   25     F  United Kingdom    full time
person 3   34     M      France    full time
person 4   27     F      Spain    full time
```

Expected output

	age	gender	country	working hours
person 1	21	M	Italy	part time
person 2	25	F	United Kingdom	full time
person 3	34	M	France	full time
person 4	27	F	Spain	full time

3. Computation with Pandas

Exercise 3.1

Compute the **mean** in a variable `mean_series` and the **standard deviation** in a variable `std_series` of the columns of the dataframe `df`. The mean and the standard deviation should be computed for each column **separately**. The dataframe `df` contains a column `size` containing the size of the houses, and another column `n_rooms` containing the number of rooms of the houses. Therefore, computing the mean and the standard deviation for each column separately, you will compute the mean and the standard deviation of the size of the houses and of the number of rooms (separately).

```
In [22]: n_houses = 100

size_houses = np.random.normal(500, 50, (n_houses,))

n_rooms = np.random.normal(3, 2, (n_houses,)) # Generate n_samples samples for the number of rooms with the specified
n_rooms = n_rooms.astype(int) # convert the number of rooms to an array containing only integers numbers
n_rooms = n_rooms+np.min(n_rooms)*-1+1 # move the samples with a mininum number of rooms of 1 (this will change the me

index_list = [f"House {i}" for i in range(n_houses)]

df = pd.DataFrame({"size":size_houses, "n_rooms":n_rooms}, index=index_list)

#### START CODE HERE (~2 lines) ####
mean_series = df.mean()
std_series = df.std()
#### END CODE HERE ####
```

```
In [23]: print("Mean of the columns")
print(mean_series)
print("\nStandard deviation of the columns")
print(std_series)
```

```
Mean of the columns
size      507.728951
n_rooms    5.560000
dtype: float64
```

```
Standard deviation of the columns
size      51.189169
n_rooms    1.871207
dtype: float64
```

Expected output

Mean of the columns	
size	497.440337

```
n_rooms      5.210000
dtype: float64
```

Standard deviation of the columns

```
size      49.829333
n_rooms   1.945183
dtype: float64
```

Exercise 3.2

Implement the **Min-Max normalization** with Pandas of the DataFrame `df` for **each column separately**. After the normalization, **all the columns must be in the range** `[0, 1]`.

Remember that the formula for the **Min-Max normalization** is the following:

$$X_{norm} = \frac{(x - x_{min})}{(x_{max} - x_{min})}$$

Firstly, run the next cell to create the DataFrame.

```
In [24]: n_houses = 100

size_houses = np.random.normal(500, 50, (n_houses,))

n_rooms = np.random.normal(3, 2, (n_houses,)) # Generate n_samples samples for the number of rooms with the specified
n_rooms = n_rooms.astype(int) # convert the number of rooms to an array containing only integers numbers
n_rooms = n_rooms+np.min(n_rooms)*-1+1 # move the samples with a mininum number of rooms of 1 (this will change the me

index_list = [f"House {i}" for i in range(n_houses)]

df = pd.DataFrame({"size":size_houses, "n_rooms":n_rooms}, index=index_list)

print("First 10 lines of the dataframe")
print(df.head(10))
```

First 10 lines of the dataframe

```
      size  n_rooms
House 0  545.227143      8
House 1  434.241300      4
House 2  569.713310      6
House 3  496.629444      7
House 4  414.249542      5
House 5  533.151643      6
House 6  407.772477      7
House 7  567.906347      7
House 8  455.328272      4
House 9  451.650865      7
```

Now, perform the **Min-Max normalization** of the DataFrame `df` and put the result into a new variable `df_norm`.

► Hints

```
In [25]: ##### START CODE HERE (~1 line) #####
df_norm = (df - df.min()) / (df.max() - df.min())
##### END CODE HERE #####
```

```
In [26]: print("Minimum values after normalization")
print(df_norm.min())
print("\nMaximum values after normalization")
print(df_norm.max())
```

```
Minimum values after normalization
size      0.0
n_rooms   0.0
dtype: float64
```

```
Maximum values after normalization
size      1.0
n_rooms   1.0
dtype: float64
```

Expected output

```
Minimum values after normalization
size      0.0
n_rooms   0.0
dtype: float64
```

```
Maximum values after normalization
size      1.0
```



```
n_rooms 1.0
dtype: float64
```

Exercise 3.3

Compute the **mean** of **all the columns** for the groups obtained by aggregating the column `size`.

Firstly, run the next cell to create the DataFrame.

```
In [31]: city_zones = ["crocetta", "santa rita", "centro", "vanchiglia", "san paolo"]
n_houses = 100
size_houses = np.random.normal(500, 50, (n_houses,))
n_rooms = np.random.normal(3, 2, (n_houses,)) # Generate n_samples samples for the number of rooms with the specified
n_rooms = n_rooms.astype(int) # convert the number of rooms to an array containing only integers numbers
n_rooms = n_rooms+np.min(n_rooms)*-1+1 # move the samples with a minimum number of rooms of 1 (this will change the me
zones = [city_zones[random.randint(0, len(city_zones)-1)] for i in range(n_houses)]
index_list = [f"House {i}" for i in range(n_houses)]
df = pd.DataFrame({"size":size_houses, "n_rooms":n_rooms, "zone":zones}, index=index_list)
print("first 10 rows:")
df.head(10)
```

first 10 rows:

```
Out[31]:
```

	size	n_rooms	zone
House 0	478.058446	6	centro
House 1	552.907640	3	crocetta
House 2	448.522096	5	san paolo
House 3	482.226842	8	santa rita
House 4	393.396507	4	centro
House 5	420.439241	1	vanchiglia
House 6	528.669641	4	santa rita
House 7	540.806055	3	san paolo
House 8	471.743012	1	vanchiglia
House 9	436.005860	4	vanchiglia

Now, compute and print the mean of the `size` and `n_rooms` columns of the groups obtained by aggregating the `zone` column. After the group operation, you should put `.reset_index()` to **reset the multi-level index**.

```
In [36]: ##### START CODE HERE (~1 line) #####
df.groupby(["zone"]).mean().reset_index()
##### END CODE HERE #####
```

```
Out[36]:
```

	zone	size	n_rooms
0	centro	514.149771	3.888889
1	crocetta	505.889859	3.857143
2	san paolo	506.128830	3.521739
3	santa rita	504.819210	3.684211
4	vanchiglia	492.087236	3.117647

Expected output

	zone	size	n_rooms
0	centro	511.350284	5.411765
1	crocetta	499.389683	5.583333
2	san paolo	490.504449	5.214286
3	santa rita	501.955531	5.300000
4	vanchiglia	505.539748	5.960000

Exercise 3.4

Compute the **maximum value** of the column `size` for the **groups obtained by aggregating** the column `zone` (i.e., in each row you should obtain the maximum size for each zone). After the group operation, you should put `.reset_index()` to **reset the multi-level index**.

```
In [37]: ##### START CODE HERE (~1 line) #####
df.groupby(["zone"])['size'].max().reset_index()
```

```
#### END CODE HERE ####
```

Out[37]:

	zone	size
0	centro	608.947503
1	crocetta	565.018991
2	san paolo	596.072535
3	santa rita	569.898028
4	vanchiglia	579.317779

Expected output

—	zone	size
0	centro	608.947503
1	crocetta	565.018991
2	san paolo	596.072535
3	santa rita	569.898028
4	vanchiglia	579.317779

In []: