Lesson 4: Data analysis in Python: Pandas

Alessio Cardillo

Department of computer science and mathematics (DΣIM),
Universitat Rovira i Virgili, Tarragona, Spain

Python Days
Dept. of physics and astronomy – University of Catania, Catania, Italy

UNIVERSITAT ROVIRA i VIRGILI

Foreword

Outline of the course/lesson

Course

1. ~~Introduction to the basics of the Python language.~~
2. ~~More "*basics*" of Python & basics of NumPy.~~
3. ~~IPython notebook & Visualization of data (Matplotlib).~~

Outline of the course/lesson

Course

1. ~~Introduction to the basics of the Python language.~~
2. ~~More "*basics*" of Python & basics of NumPy.~~
3. ~~IPython notebook & Visualization of data (Matplotlib).~~
4. Data analysis (Pandas).

Lesson 4

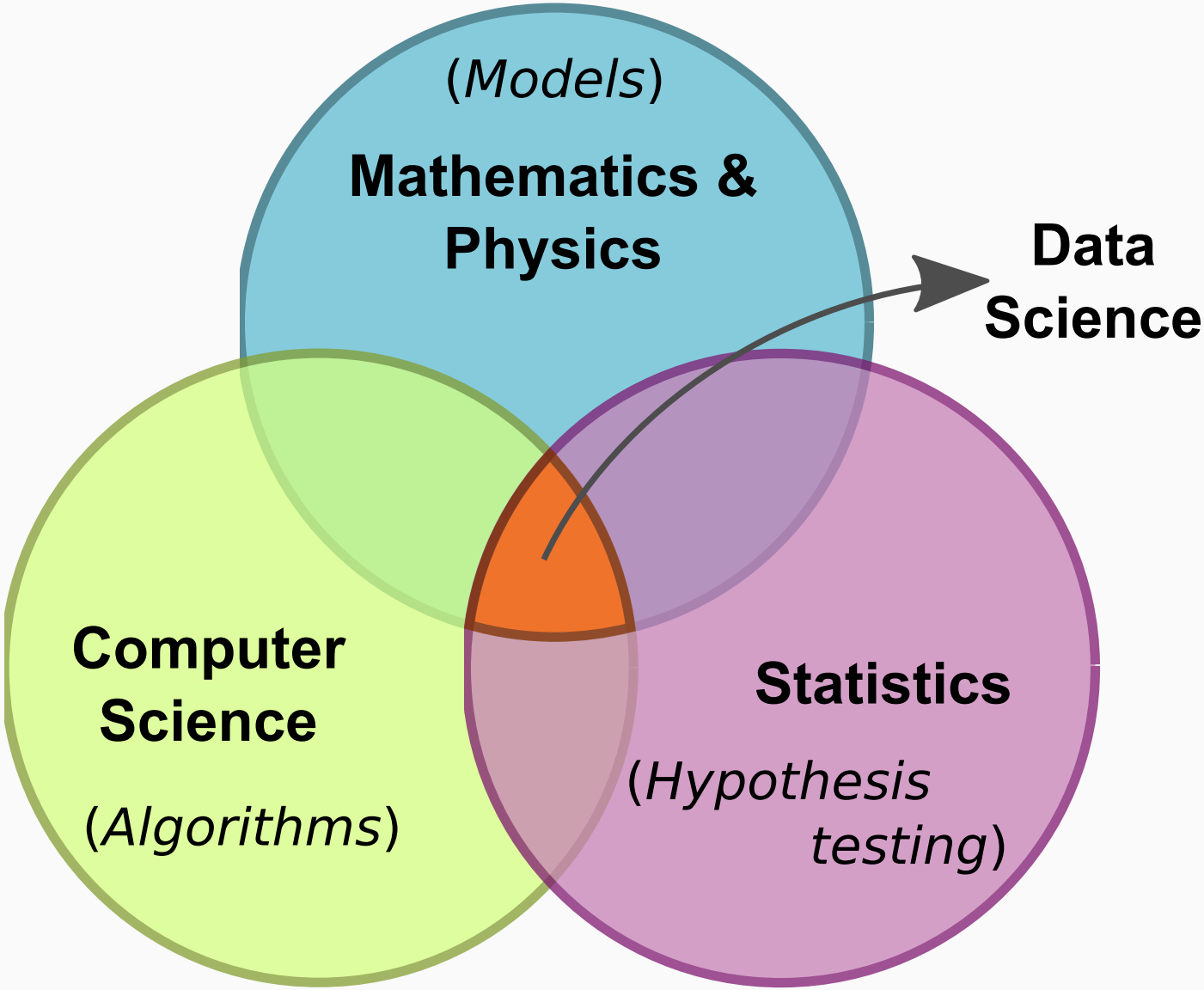
- Data science
- Data science with Python (Pandas)
- Hands-on session

What is “Data science”?

Data science

Data science is a cross-disciplinary field using scientific methods, processes, algorithms and systems to **extract knowledge and insights** from structural and unstructured **data**.

Data science



Data science

Data science is

Due to its intrinsic nature, data science is strongly associated with these concepts:

Data science

Data science is

Due to its intrinsic nature, data science is strongly associated with these concepts:

Data mining (**Knowledge mining from data**) Discover patterns in data, prepare data that are functional/useful to address the questions we have, and extract/pre-process data from sources (*e.g.*, using APIs).

Data science

Data science is

Due to its intrinsic nature, data science is strongly associated with these concepts:

Data mining (**Knowledge mining from data**) Discover patterns in data, prepare data that are functional/useful to address the questions we have, and extract/pre-process data from sources (*e.g.*, using APIs).

Machine learning A branch of **Artificial intelligence** studying computer algorithms that improve automatically through *experience* (*i.e.*, comparison between the outcome of a model on some training data, and the same thing based on "real" data). Applications include, among other things, classification, grouping (clustering), reduction, and filtering.

Data science

Data science is

Due to its intrinsic nature, data science is strongly associated with these concepts:

Data mining (**Knowledge mining from data**) Discover patterns in data, prepare data that are functional/useful to address the questions we have, and extract/pre-process data from sources (*e.g.*, using APIs).

Machine learning A branch of **Artificial intelligence** studying computer algorithms that improve automatically through *experience* (*i.e.*, comparison between the outcome of a model on some training data, and the same thing based on "real" data). Applications include, among other things, classification, grouping (clustering), reduction, and filtering.

Big data *Considerable* amount of data (usually obtained from digital records). Sometimes the prefix *big-* does not refer exclusively to the **size** of the data but, rather, to their dimensionality (~~big~~ data $\rightarrow$ better data).

Data science

Common mistake

"... the key word in **data science** is not '*data*'; it is '*science*.' **Data science is only useful when the data are used to answer a question**. That is the science part of the equation. The problem with this view of data science is that it is much harder than the view that focuses on data size or tools. It is much, much easier to calculate the size of a data set and say '*My data are bigger than yours*' or to say, '*I can code in Hadoop, can you?*' than to say, '*I have this really hard question, can I answer it with my data?*'."

(*Jeff Leek*)

- https://simplystatistics.org/2013/12/12/the-key-word-in-data-science-is-not-data-it-is-science/

Data science

What do we need to do data science?

To address our "questions" we need to **store**, **manipulate**, and **analyze** (*e.g.*, visualize) the data in an **efficient** and **scalable** way.

Data analysis with Python (Pandas)

Pandas

- Development started in 2008 (latest ver. 1.1.4).
  Wes McKinney is the Benevolent Dictator for Life.

- `https://pandas.pydata.org/pandas-docs/stable/user_guide/index.html`

Pandas

- Development started in 2008 (latest ver. 1.1.4). *Wes McKinney* is the Benevolent Dictator for Life.
- Goal: to be the fundamental high-level building block for doing practical, real world data analysis in Python. Additionally, it has the broader goal of becoming the most powerful and flexible open source data analysis/manipulation tool available in *any language*.

- `https://pandas.pydata.org/pandas-docs/stable/user_guide/index.html`

Pandas

- Development started in 2008 (latest ver. 1.1.4). *Wes McKinney* is the Benevolent Dictator for Life.
- Goal: to be the fundamental high-level building block for doing practical, real world data analysis in Python. Additionally, it has the broader goal of becoming the most powerful and flexible open source data analysis/manipulation tool available in *any language*.
- Highly optimized for performance, with critical code paths written in `Cython` and/or `C`.

- `https://pandas.pydata.org/pandas-docs/stable/user_guide/index.html`

Pandas

- Development started in 2008 (latest ver. 1.1.4). *Wes McKinney* is the Benevolent Dictator for Life.
- Goal: to be the fundamental high-level building block for doing practical, real world data analysis in Python. Additionally, it has the broader goal of becoming the most powerful and flexible open source data analysis/manipulation tool available in *any language*.
- Highly optimized for performance, with critical code paths written in `Cython` and/or `C`.
- Based on two types of data structures: `Series` and `DataFrame`.

- `https://pandas.pydata.org/pandas-docs/stable/user_guide/index.html`

Pandas

- Flexible handling of data supporting operations like: *reshaping*, *pivoting*, *label-based slicing*, *fancy indexing*, *insertion*, *split*, *merge*, and *join*.

- `https://pandas.pydata.org/pandas-docs/stable/user_guide/index.html`

Pandas

- Flexible handling of data supporting operations like: *reshaping*, *pivoting*, *label-based slicing*, *fancy indexing*, *insertion*, *split*, *merge*, and *join*.
- Intelligent data alignment and integrated handling of "*messy data*" and missing entries.

- `https://pandas.pydata.org/pandas-docs/stable/user_guide/index.html`

Pandas

- Flexible handling of data supporting operations like: *reshaping*, *pivoting*, *label-based slicing*, *fancy indexing*, *insertion*, *split*, *merge*, and *join*.
- Intelligent data alignment and integrated handling of "*messy data*" and missing entries.
- Time series-functionality: date range generation and frequency conversion, moving window statistics, date shifting and lagging. Even create domain-specific time offsets and join time series without losing data.

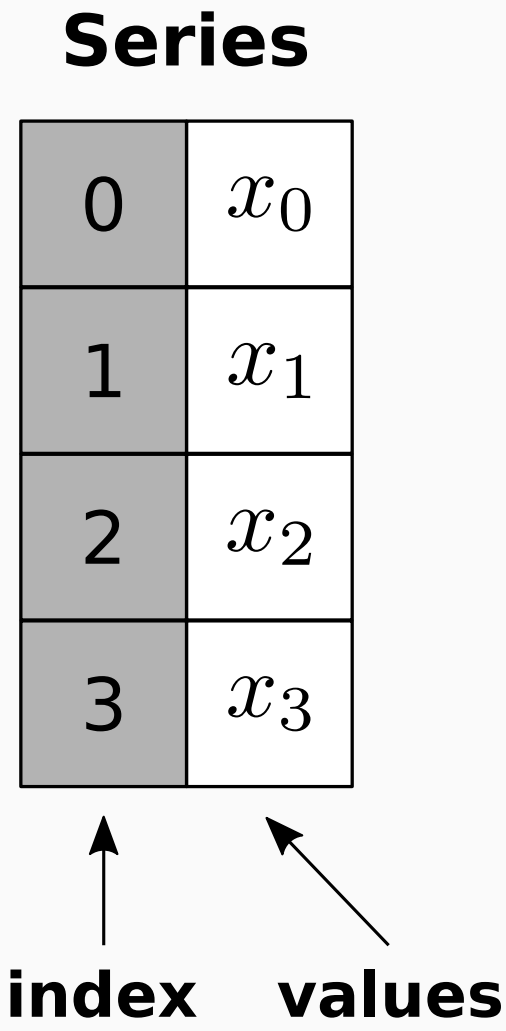
- `https://pandas.pydata.org/pandas-docs/stable/user_guide/index.html`

Pandas

- Flexible handling of data supporting operations like: *reshaping*, *pivoting*, *label-based slicing*, *fancy indexing*, *insertion*, *split*, *merge*, and *join*.
- Intelligent data alignment and integrated handling of "*messy data*" and missing entries.
- Time series-functionality: date range generation and frequency conversion, moving window statistics, date shifting and lagging. Even create domain-specific time offsets and join time series without losing data.
- Provides tools for reading and writing data in multiple formats (`CSV`, `TXT`, `Excel`, `SQL`, `JSON`, `HDF5`).

- `https://pandas.pydata.org/pandas-docs/stable/user_guide/index.html`

Data structures: Series



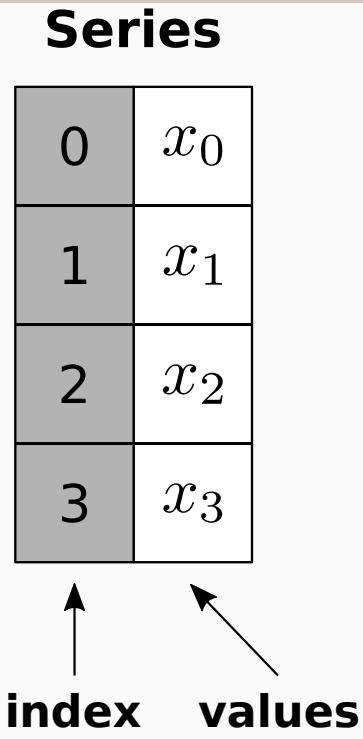
`Series` is a **one-dimensional labeled array** capable of holding any data type (int, str, float, Python objects, etc.). The axis labels are collectively referred to as the index. **Note:** The index is not necessarily numeric.

- https://pandas.pydata.org/pandas-docs/stable/user_guide/dsintro.html#series

Data structures: Series

Creating/initializing a Series

```
# series from single value
ser1 = pd.Series(5.0, index=np.arange(10))

# series with non-numerical index
ser2 = pd.Series(5.0, index=['a','b','c','d','e'])

# series from ndarray
ser1 = pd.Series(np.random.random(10), index=np.arange(10))

# series from dictionary
mydict = {'a': 10., 'b': True, 'c': 'alex', 'd': -3, 'e': -2.5}

ser1 = pd.Series(mydict)
```



Basic operations on Series include, among others, viewing, and description.

Hands-on

```
# defining a Series
ser = pd.Series(np.random.random(10), index=np.arange(10))

# visualizing a Series (eg. the first/last 3 rows)
ser.head(3) # alternatively ser.tail(3)

# obtaining basic information over a Series
ser.describe()
```

Playing with `Series`

Operations with `Series`

`Series` behave like NumPy arrays (`ndarray`), and support operations like sum, product (by a scalar or between `Series`), and so on. `Series` support also operations like average, maximum (minimum), and so on. However, a key difference between `Series` and `ndarray` is that operations between `Series` automatically align the data based on label (index). Thus, you can write computations without giving consideration to whether the `Series` involved have the same labels or not.

Playing with Series

Hands-on

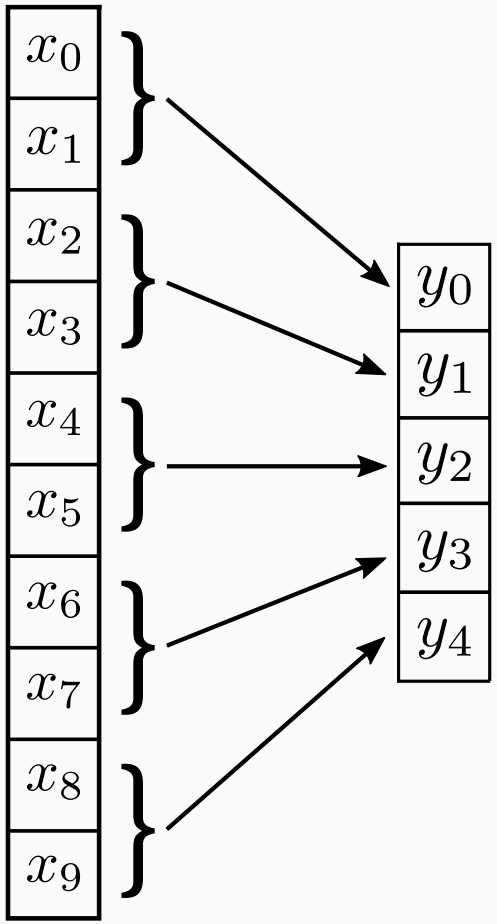
```
# Series with the same indexes
ser1 = pd.Series(2.5, index=np.arange(10))
ser2 = pd.Series(2.5, index=np.arange(10))

ser3 = ser1+ser2

# Series with the different indexes
ser1 = pd.Series(2.0, index=np.arange(5))
ser2 = pd.Series(2.0, index=np.arange(3,8))

ser3 = ser1+ser2
```

Playing with `Series`

Resampling and rolling on `Series`

pandas contains extensive capabilities and features for working with time series data. In particular, we will concentrate on:

- Performing resampling operations during frequency conversion (*e.g.*, converting secondly data into 5-minutely data).
- Performing rolling operations (*e.g.*, computing a moving average).



- `https://pandas.pydata.org/pandas-docs/stable/user_guide/timeseries.html`
- `https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.rolling.html`

Playing with Series

Hands-on

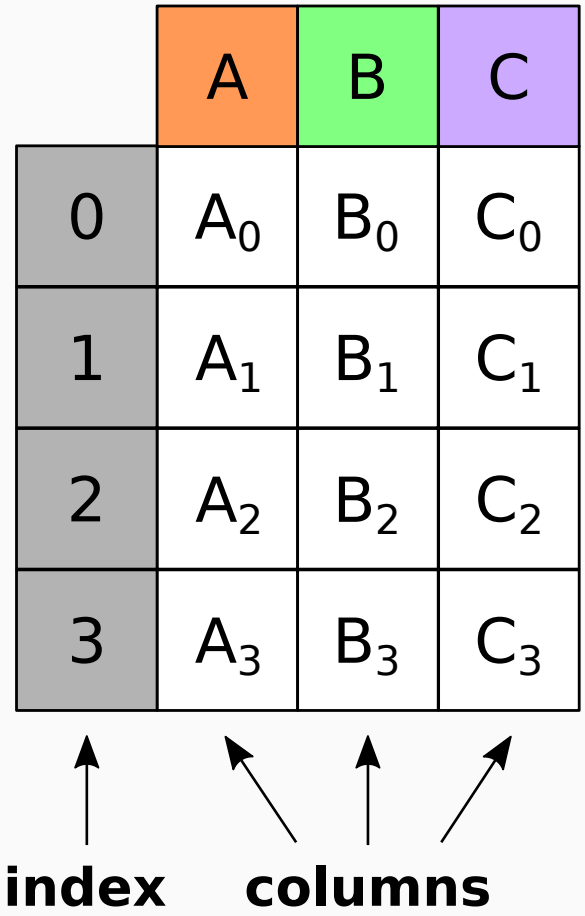
```
##### RESAMPLING #####
# creating the index of the Series
myindex = pd.date_range(start='1/1/2018', end='4/30/2018', freq='D')
# creating a Series made all by the same value
ser1 = pd.Series(5., index=myindex)
# resampling the original Series into another with monthly resolution
# whose values are the sum of the values of the original Series
ser2 = ser1.resample('M').sum()

##### ROLLING #####
# setting parameters
sersize = 500
serroll = 25

# creating a Series made of random integers in the range [0,100]
ser1 = pd.Series(np.random.randint(0, high=101, size=sersize), index=range(sersize))
# computing the rolling mean using a window of size "serroll"
ser2 = ser1.rolling(serroll).mean()
# computing the rolling mean using a window of size "serroll"
# evaluated at the center of the window
ser3 = ser1.rolling(serroll, center=True).mean()
```

Data structures: `DataFrame`

A `DataFrame` is a **2-dimensional labeled data structure** with columns of (potentially) different types. You can think of it like a *spreadsheet* or a *SQL table*, or a *dictionary* of `Series` objects.

Data structures: DataFrame

DataFrame



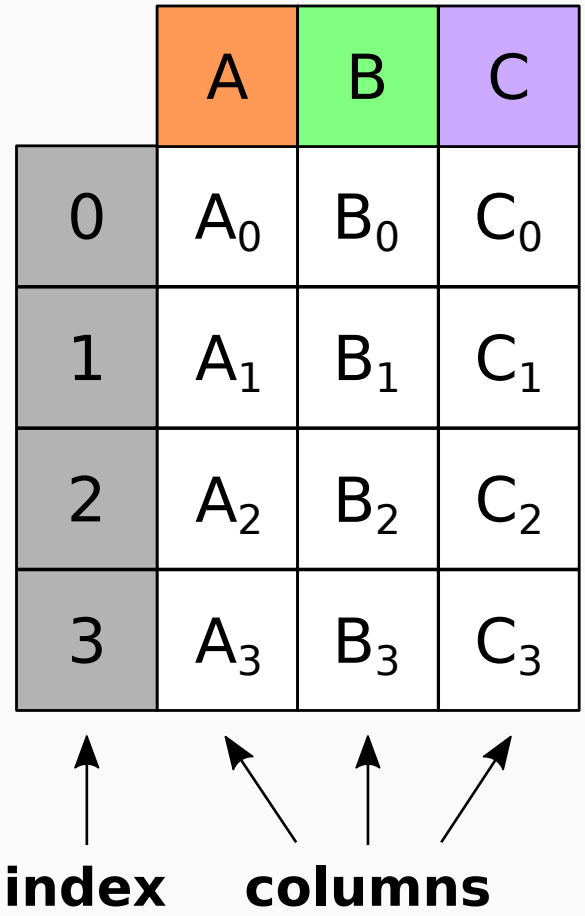
Along with the data, you can optionally pass index (row labels) and columns (column labels) arguments. Data alignment between DataFrame objects automatically align **on both** the columns and the index (discard all data not matching up to the passed index).

Note: If axis labels are not passed, they will be constructed from the input data based on common sense rules.

- https://pandas.pydata.org/pandas-docs/stable/user_guide/dsintro.html#dataframe

Data structures: DataFrame

DataFrame



You can treat a `DataFrame` semantically like a dictionary of `Series` objects using the same indexes. This means that getting, setting, and deleting columns works with the same syntax as the analogous dictionary operations.

- https://pandas.pydata.org/pandas-docs/stable/user_guide/dsintro.html#dataframe

Playing with DataFrame(s)

Creating a DataFrame

```
# importing pandas
import pandas as pd

# common paramters
nr_rows = 10
nr_cols = 5
alphabet = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'

# creating a DataFrame (setting only the values)
mydf = pd.DataFrame(10.*np.random.random((nr_rows,nr_cols)))

# creating a DataFrame (setting index and columns names)
mydf = pd.DataFrame(10.*np.random.random((nr_rows,nr_cols)),\
                    index=list(range(nr_rows)),\
                    columns=[x for x in alphabet[:nr_cols]])

# creating a DataFrame from a dictionary
mydict = {'A': 1.5,
          'B': pd.Timestamp('20130102'),
          'C': pd.Series(-5., index=list(range(4)), dtype='float32'),
          'D': np.array([3] * 4, dtype='int32'),
          'E': pd.Categorical(["test", "train", "test", "train"]),
          'F': 'foo'
         }

mydf = pd.DataFrame(mydict)
```

Playing with DataFrame(s)

Read/Write DataFrame

To read/write a DataFrame (or a Series) from a CSV file, we can use the methods `.read_csv()` and `.to_csv()`, respectively.

Note

Analogous methods exist also for the HDF5, Excel, JSON, HTML, Stata, pickle, and SQL formats.

Hands-on

```
# reading a CSV file
mydf = pd.read_csv('scorsese_filmography.txt',
                   sep=';', # specifying the separator
                   names=['Year', 'Title', 'Director',\
                          'Producer', 'Writer'], # cols names
                   skiprows=1 # skipping the first row
                  )

# writing the DataFrame on file
mydf.to_csv('df-csv-output_example.csv',
            sep = ':', # separator
            header = True, # writing the header (column names)
            index = True, # writing the row indexes
            mode = 'w', # writing mode
            compression = None # compression mode
           )
```

- https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html
- https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.to_csv.html

Playing with DataFrame(s)

Let us extract some **basic** information about a DataFrame

```
# common paramters
nr_rows = 20
nr_cols = 5
alphabet = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'

# creating a Dataframe
mydf = pd.DataFrame(10.*np.random.random((nr_rows,nr_cols)),\
                    index=list(range(nr_rows)),\
                    columns=[x for x in alphabet[:nr_cols]]
                   )
```

Getting information

```
# printing ALL the DataFrame
mydf

# printing DataFrame (only the first 3 rows)
mydf.head(3)

# printing DataFrame (only the last 3 rows)
mydf.tail(3)

# printing the types of each column
mydf.dtypes

# printing the indexes
mydf.index

# printing the columns
mydf.columns

# getting a summary description of the DataFrame
mydf.describe()
```

- https://pandas.pydata.org/pandas-docs/stable/user_guide/10min.html

Playing with DataFrame(s)

Case study: England climate data

Let's consider the records of the average monthly temperature (in °C) registered in central England from year 1659 to 2016.

```
# reading the file
mydf = pd.read_csv('data_climate-uk.dat',
                   skiprows=6, # skipping header
                   sep=r'\s+', # deleting multiple
                               # spaces with a regexp
                   # telling pandas how to deal with
                   # "incorrect entries"
                   na_values=[-99.9, -99.99]
                  )
```

Hands-on

```
# extracting a single column
mydf['MAY'] # equivalently we can use  mydf.MAY

# extracting rows (by their index values = label)
# extracting all the values of the '80
mydfeighties = mydf.loc[1980:1989]

# the same as above but extracting only two columns
mydfeighties = mydf.loc[1980:1989, ['MAY', 'NOV']]

# extracting the first three rows (by their position)
mydfeighties.iloc[0:3]

# extracting three rows (by their position)
# at steps of two
mydfeighties.iloc[0:6:2]

# same as above but selecting only one column
# (the last)
mydfeighties.iloc[0:6:2,1]

# sorting by column (eg. the month of May)
mydfsort = mydf.sort_values(by='MAY').copy()
```

Playing with DataFrame(s)

Hands-on: Logic-slicing

```
temp_thres = 10.5

# logic slicing
# finding for which years the average temperature in May is
# above a given threshold
mydf['MAY'] > temp_thres

# extracting only the years (indexes) with temperature > threshold
tmpdf = mydf['MAY'] > temp_thres
mylist = list(mydf[tmpdf == True].index)
# alternatively use:   mydf[ tmpdf == True].index.tolist()

# selecting the data only for the selected years
mydf2 = mydf.loc[mylist]

# printing the results
mydf2
```

Playing with DataFrame(s)

Hands-on: Add/deleting data

```
# extracting the years (and converting them into strings)
years = pd.Series(mydf.index, index=mydf.index).apply(str)

# creating lists containing the decade and the century
decade = [int(x[:3]+'0') for x in years]
century = [int(x[:2]+'00') for x in years]
millenium = [int(x[:1]+'000') for x in years]

# adding the new columns
mydf['decade'] = decade
mydf['century'] = century
mydf['millennium'] = millennium

# showing the updated DataFrame
mydf.head()

# deleting one column
mydf = mydf.drop(columns=['millennium'])

# deleting one row
mydf = mydf.drop(1659)

# showing the updated DataFrame
mydf.head()
```

Playing with DataFrame(s)

Hands-on: Operations with DataFrame

```
# showing the the average of the columns of the DataFrame
mydf.mean()

# summing two columns first and making the average then
mydf2 = 0.5*(mydf['JAN']+mydf['FEB'])

mydf2

# accounting (or not for the presence of NaN values)
mymax1 = mydf['AUG'].max(skipna=False)
mymax2 = mydf['AUG'].max()
```

Playing with DataFrame(s)

Merging & Grouping

There are three categories of merging/grouping operations:

- Splitting the data into groups based on some criteria.
- Applying a function to each group independently.
- Combining the results into a data structure.

Hands-on

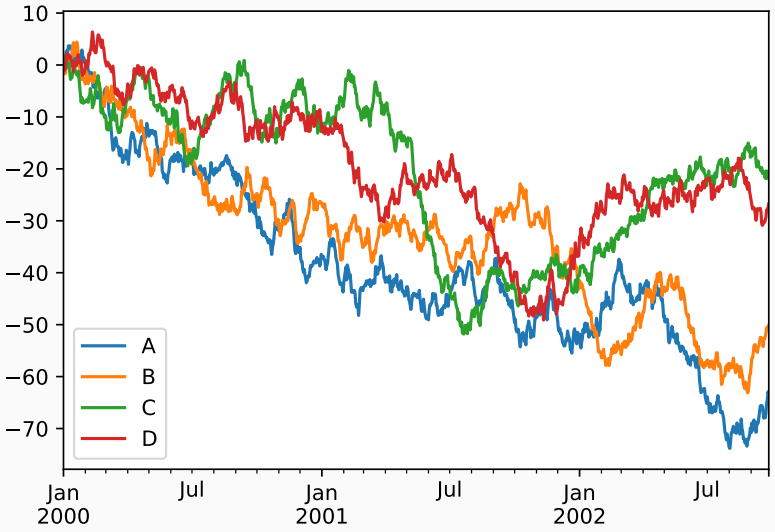
```
# grouping temperatures according to the decade
gp_by_decade = mydf.groupby('decade')
mydf_agg_decade = gp_by_decade.aggregate(np.mean)

# grouping temperatures according to the century
gp_by_century = mydf.groupby('century')
mydf_agg_century = gp_by_century.aggregate(np.mean)

#### EXAMPLE OF JOIN ####
# creating two DataFrames with one column (key)
# present in both DataFrames

mydf_left = pd.DataFrame({'key': ['a', 'b', 'c', 'd'],
                          'lval': np.arange(4)})
mydf_right = pd.DataFrame({'key': ['a', 'b', 'c', 'e'],
                           'rval': np.arange(4,8)})

# merging the two DataFrames using the column "key" as
# index to perform the matching
mydf_merge = pd.merge(mydf_left, mydf_right, on='key')

mydf_merge
```

- https://pandas.pydata.org/pandas-docs/stable/user_guide/merging.html

Playing with DataFrame(s)

Hands-on: Plotting a DataFrame

```
# creating a DataFrame
mydf = pd.DataFrame(np.random.randn(1000, 4),
                    index=pd.date_range('1/1/2000',
                    periods=1000),
                    columns=['A', 'B', 'C', 'D'])

mydf = mydf.cumsum()

plt.figure()
mydf.plot()
```



Playing with `DataFrame`(s)

Check the Cheatsheet provided with the materials!

Hands-on Session

Excercise 1

Task

After loading the data of births and deaths occurred in France from January 1946 to December 2019, do:

1. Load the two datasets as two distinct `DataFrames`.
2. Merge the two `DataFrames` onto a new one.
3. On the new `DataFrame`, create a new column accounting for the "decade".
4. Find the year/month with the highest number of births and deaths.
5. Compute the so-called *natural population change*, N_{pc} (*i.e.*, the difference between births and deaths).
6. Find the year/month of negative values of N_{pc}.
7. Group the results (applying the sum method) per year and decade.
8. Plot the value of the N_{pc} per year together with its (3 points) moving average.

Excercise 1

Data

The data are freely available from the website of the French *National Institute of Statistics and Economic Studies* (INSEE). Specifically:

births `www.insee.fr/fr/statistiques/serie/000436391`

deaths `www.insee.fr/fr/statistiques/serie/000436394`

The data can be downloaded as a CSV file (you can select the range of month/years).

Excercise 1

Tip

The data are provided as `.zip` archives, and the file containing the data has a header and columns which are not useful. Use the power of the `.read_csv()` function to get rid of the unnecessary information.

Caution

Be careful with the handling of values as `strings`.

Bibliography i

- J. Leek. *The key word in "Data Science" is not Data, it is Science*. Available at:https://simplystatistics.org/2013/12/12/the-key-word-in-data-science-is-not-data-it-is-science/
- Homepage of the pandas Python package. Available at: https://pandas.pydata.org/
- Pandas User Guide. Available at: https://pandas.pydata.org/pandas-docs/stable/user_guide/index.html
- Pandas documentation – Series. Available at: https://pandas.pydata.org/pandas-docs/stable/user_guide/dsintro.html#series

Bibliography ii

- Pandas documentation – Time series. Available at: https://pandas.pydata.org/pandas-docs/stable/user_guide/timeseries.html
- Pandas documentation – Rolling operations on Series. Available at: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.rolling.html
- Pandas documentation – 10 minutes to Pandas. Available at: https://pandas.pydata.org/pandas-docs/stable/user_guide/10min.html
- Pandas documentation – Read CSV. Available at: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html

Bibliography iii

- Pandas documentation – Write CSV. Available at: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.to_csv.html
- Pandas documentation – Merging of DataFrame. Available at: https://pandas.pydata.org/pandas-docs/stable/user_guide/merging.html