

# Pandas

track 2 - Pandas bases - 16th March

# Outline

---

1

Introduction

2

Data structures

3

Interactions I/O

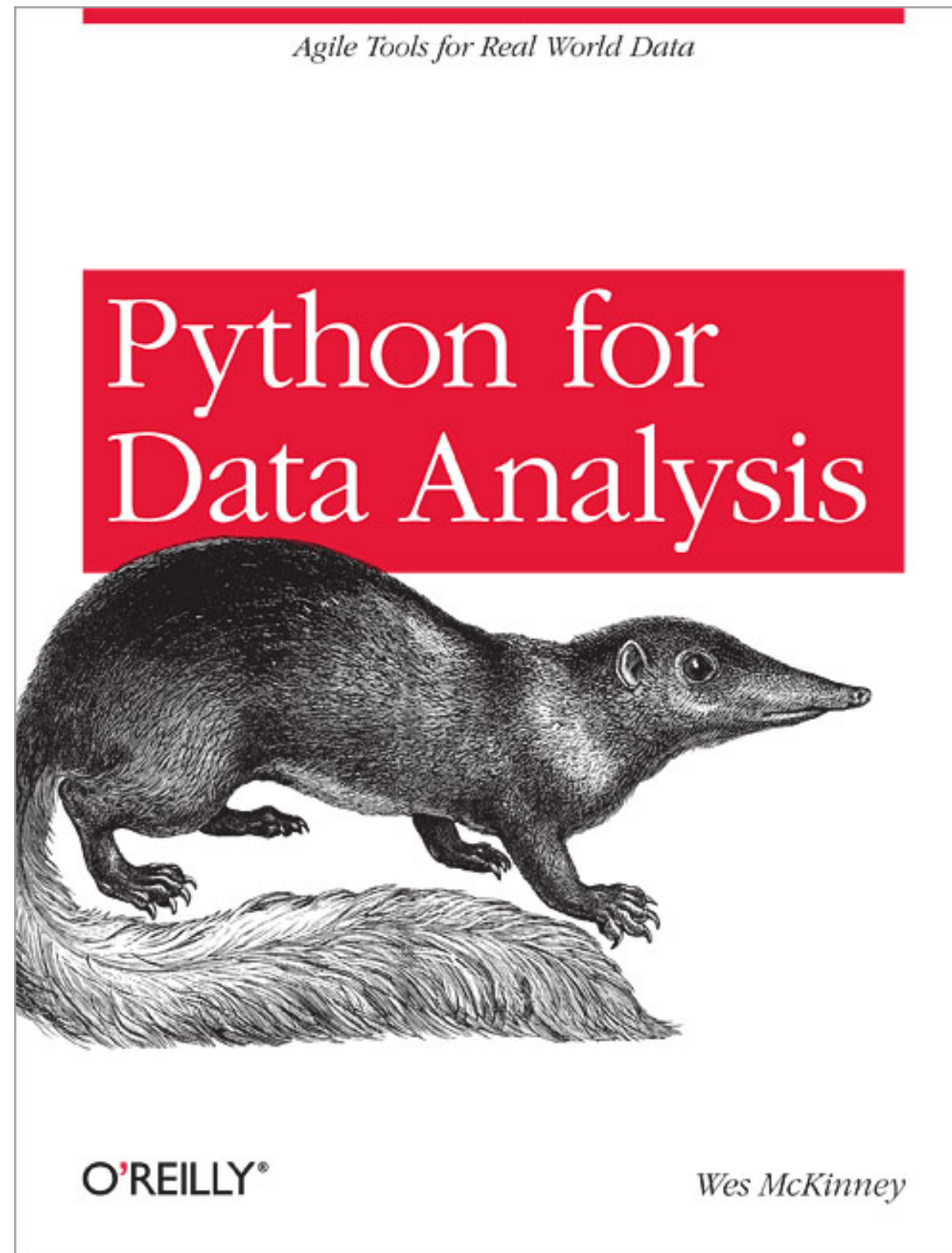
4

DataFrame endless possibilites

5

Exercices

# 1 Introduction



# 1 Introduction

---

- Pandas is a **toolkit** to manipulate data
- It based on **numpy** for computations (faster)
- **DataFrame** is the key element (like in R)
- It became quickly the data manipulation and analysis **standard** over data ecosystem in Python

# 1 Introduction

---

- You need pandas when you work with tabular or structured data
- To:
  - *import* and *clean* data
  - *explore* data
  - *process* and *prepare* data
  - and *analyse*

# 1 Introduction: key features

---

- Fast and easy API to I/O in different formats
- Working with missing records
- Merging / joining (`concat`, `join`)
- Grouping
- Reshaping
- Powerful time series manipulation

## 2 Data structures: import

---

```
import pandas as pd  
import numpy as np
```

- The pandas import is always simple as this
- We import np to have access to all numpy methods

## 2 Data structures: Series

```
s = pd.Series([0.2, 0.4, 0.8, 1.6])
```

```
s.index
```

```
# returns: Int64Index([0, 1, 2, 3], dtype="int64")
```

```
s.values
```

```
# returns: array([ 0.2, 0.4, 0.8, 1.6 ])
```

```
s[0] # to access data
```

```
s2 = pd.Series([0.2, 0.4, 0.8, 1.6], index=['a', 'c', 'b', 'd'])
```

- A Series is a **one-dimensional array-like** object containing an array of data
- We can access to values and indexes
- And so the data is indexed inside and can be accessible with: `s[0]`
- You can also create your own index => Series can be created from dictionaries



## 2 Data structures: Series

```
agedata = {"francois": 51, "angela": 51, "barack": 55}
```

```
s3 = pd.Series(agedata) # create from dict
```

```
s3[s3 > 52] # filtering
```

```
s3 * 2 # scalar multiplication  
s3.mean()
```

```
np.exp(s3) # exponential
```

```
"angela" in s3 # boolean to find if a key is in
```

```
s3[["angela", "barack"]] # get several values
```

- Series can be created from dictionaries (keys will be sorted)
- On Series we can do **filtering**, **scalar multiplication** or math functions applying
- Determine if an index is in the series with **in**

## 2 Data structures: Series

```
agedata = {"francois": 51, "angela": 51, "barack": 55}
presidents = ["barack", "francois", "angela", "georges"]
```

```
s4 = pd.Series(agedata, index=presidents)
```

```
pd.isnull(s4)
s4.isnull()
pd.notnull(s4)
```

```
s3 + s4
```

In [25]:

s3 + s4

Out[25]:

```
angela      102
barack       110
francois     102
georges      NaN
dtype: float64
```

In [20]:

s4

Out[20]:

```
barack      55
francois     51
angela       51
georges      NaN
dtype: float64
```

- NaN (Not a Number) is the null element in a Series object
- isnull and notnull returns if elements are nulls
- We can add 2 Series

## 2 Data structures: Series

```
s4.name = "presidents_ages"  
s4.index.name = "name"  
  
s4.index = ["Lula", "Cameron", "Renzi", "Putin"]
```

- An index can have a name
- Like a Series
- We can change the index afterwards

## 2

# Data structures: DataFrame

```
data = {  
    "city": ["Paris", "London", "Berlin"],  
    "density": [3550, 5100, 3750],  
    "area": [2723, 1623, 984],  
    "population": [9645000, 8278000, 3675000],  
}  
  
df = pd.DataFrame(data)
```

In [55]:

df

Out[55]:

	area	city	density	population
0	2723	Paris	3550	9645000
1	1623	London	5100	8278000
2	984	Berlin	3750	3675000

- A dataframe is a tabular data structure, containing an ordered collection of columns, each of which can be a different value type (numeric, boolean, string, etc.)
- DataFrame has both rows and columns index

# 2

## Data structures: DataFrame

```
columns = ["city", "area", "population", "density"]  
df = pd.DataFrame(data, columns=columns)
```

```
df["area"] or df.area  
# returns a Series object of the areas in the df
```

```
df.dtypes # to get the types of columns
```

```
df.info()  
df.describe() # give stats on the df  
df.values  
df.index
```

```
df = df.set_index("city")
```

- Columns can be specified
- We get a Series if we index a DataFrame

# 2

## Data structures: DataFrame

```
df["population"] / df["area"]  
  
df["real_density"] = df["population"] / df["area"]  
  
df.ix["Paris"] # to get the row with index "Paris"  
  
df[df["real_density"] < 5000] # to filter by density  
  
df.sort_index(by="real_density", ascending=True) # to sort
```

- Operations between columns are possible
- We can add new columns easily
- If we set index with `ix[]` rows are accessible

# 2

## Data structures: DataFrame

### Selecting the data

```
df["area"] # get the column
df.ix["Paris"] # get the row

# multiple columns
df[["area", "population"]]

# loc examples
df.loc["Paris", "area"] # will return the exact value
df.loc[df["density"] < 5000, ["population", "area"]]

# iloc example
df.iloc[1, 2]
```

- Be careful when getting column or row
- For advanced indexing we have:
  - loc: selecting by label
  - iloc: selecting by position

# 2

## Data structures: DataFrame

### Assigning the data

```
df.iloc[1, 2] = 10
```

```
df.iloc[1, :] = 10
```

```
df[df["density"] == 10] = 6000
```

- After selecting (with all different ways) the data we can assign them



## 3

# Interactions I/O

## Read the data: text file

```
df = pd.read_csv("population.csv")
```

```
# with parameters
df1 = pd.read_csv(filename,
    sep=",",
    header=None,
    names=[ ],
    index_col=[ ],
    na_values=[ ],
)
```

read_csv	load delimited data from a file, URL, of file-like object (comma as default delimiter)
read_table	load delimited data from a file, URL, of file-like object ('\t' as default delimiter)
read_fwf	Read data in fixed-width column
read_clipboard	Read data from the clipboard

- **read\_csv** is the most useful function to read the data (because the data is often used as csv format)
- we can specified a lot of parameters to read\_\*

# 3

## Interactions I/O

### Read the data: database

---

```
import sqlite3

connexion = sqlite3.connect(':memory:')

df = pd.io.sql.read_frame("SELECT * FROM table", connexion)
```

- We can do the same with a MySQL python connector

# 3

## Interactions I/O

Write the data

---

```
df = pd.to_csv("population_out.csv")
```

- **to\_csv** is like read\_csv and has the same parameters

## 4

## DataFrame endless possibilities

```
In [71]: country = pd.read_csv("country_n_sample.csv")
```

```
In [72]: country.head()
```

Out[72]:

	isocode	countryname	rbucode	currency	order	G6	IsocodeAggregation	IsocodeAggregationName
0	AT	Austria	NCE	€	11	NO	AT	Austria
1	BE	BELUX	NWE	€	6	NO	BE	BELUX
2	CH	Switzerland	NCE	CHF	12	NO	CH	Switzerland
3	CZ	Czech Republic	NSCEE	€	15	NO	CZ	Czech Republic
4	DE	Germany	NCE	€	10	YES	DE	Germany

```
In [73]: data = pd.read_csv("daily_n_sample.csv")
```

```
In [74]: data.head()
```

Out[74]:

	Date	Isocode	ModelCode	Source	QualifiedAudience	Visits	PageViews	UniqueVisitors
0	2015-04-01 00:00:00	AT	C11A	Mobile	29.93	NaN	NaN	NaN
1	2015-04-01 00:00:00	AT	D40	Mobile	8.57	NaN	NaN	NaN
2	2015-04-01 00:00:00	AT	E28B	Mobile	21.62	NaN	NaN	NaN
3	2015-04-01 00:00:00	AT	EM20	Mobile	5.55	NaN	NaN	NaN
4	2015-04-01 00:00:00	AT	EVA	Mobile	6.00	NaN	NaN	NaN

- we defined two dataframes to illustrate the DF manipulations

# 4

## DataFrame endless possibilities

```
data.sort_index(axis=1, ascending=False).head() # sort on column names

data.sort_index(by="QualifiedAudience", ascending=False)

data1 = data[["Date", "Isocode", "QualifiedAudience"]].set_index(["Date", "Isocode"])
data1.rank(ascending=False, method='max')
```

- we can sort by an index or do a ranking

## 4

## DataFrame endless possibilities

```
In [78]: pd.merge(data, country, left_on="Isocode", right_on="isocode").head()
```

```
Out[78]:
```

	Date	Isocode	ModelCode	Source	QualifiedAudience	Visits	PageViews	UniqueVisitors	isocode	countryname	rbucode	currency	order	G
0	2015-04-01 00:00:00	AT	C11A	Mobile	29.93	NaN	NaN	NaN	AT	Austria	NCE	€	11	N
1	2015-04-01 00:00:00	AT	D40	Mobile	8.57	NaN	NaN	NaN	AT	Austria	NCE	€	11	N
2	2015-04-01 00:00:00	AT	E28B	Mobile	21.62	NaN	NaN	NaN	AT	Austria	NCE	€	11	N
3	2015-04-01 00:00:00	AT	EM20	Mobile	5.55	NaN	NaN	NaN	AT	Austria	NCE	€	11	N
4	2015-04-01 00:00:00	AT	EVA	Mobile	6.00	NaN	NaN	NaN	AT	Austria	NCE	€	11	N

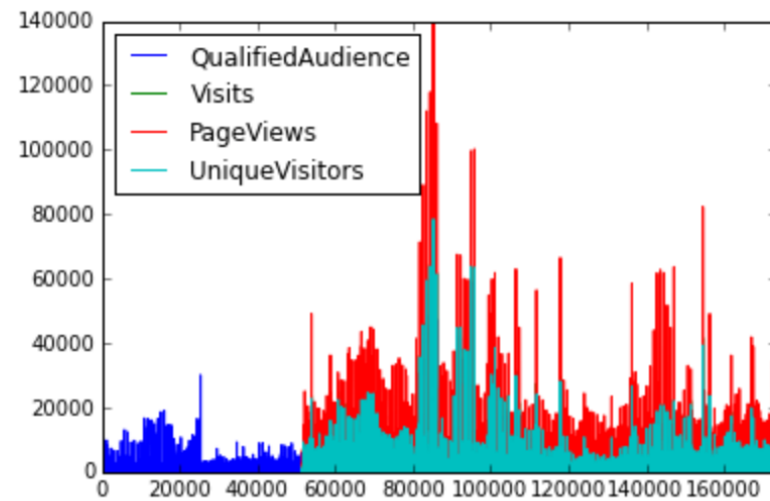
- merge is done between two dataframes on a **key** and **how**
- we can use: key, left\_key and right\_key
- how values are: inner (by default), left, right, outer
- after the merge you have a new dataframe to work with

## 4

# DataFrame endless possibilities

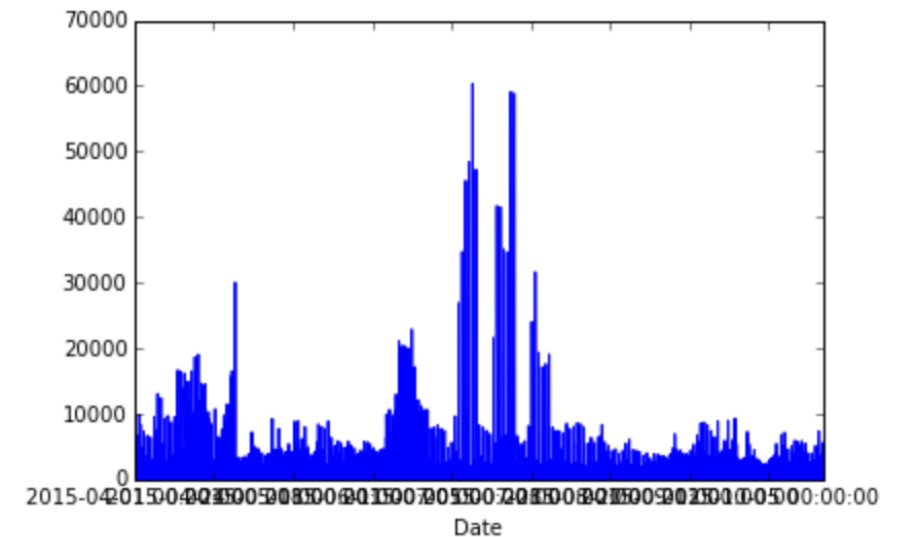
```
In [79]: data.plot()
```

```
Out[79]: <matplotlib.axes._subplots.AxesSubplot at 0x107cda5c0>
```



```
In [92]: data["QualifiedAudience"].plot(kind="line")
```

```
Out[92]: <matplotlib.axes._subplots.AxesSubplot at 0x1117df518>
```



- you can plot the data easily in a notebook

# 4 DataFrame endless possibilities

```
In [110]: data[["Date", "Isocode", "QualifiedAudience"]].groupby(["Date", "Isocode"]).mean()
```

Out[110]:

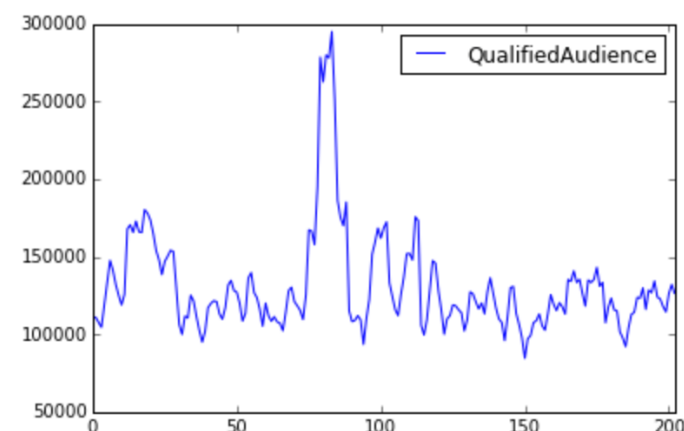
		QualifiedAudience
Date	Isocode	
2015-04-01 00:00:00	AT	30.031628
	BE	52.230698
	CH	31.318182
	CZ	35.504194
	DE	212.037500
	DK	48.300000
	EE	12.826800
	ES	125.191957
	FI	36.344000
	FR	486.217073
	HU	23.504412
	IT	201.562444
	LT	12.544000
	LU	6.422000

```
data[["Date", "Isocode", "QualifiedAudience"]].groupby(["Date", "Isocode"], as_index=False).mean()
```

	Date	Isocode	QualifiedAudience
0	2015-04-01 00:00:00	AT	30.031628
1	2015-04-01 00:00:00	BE	52.230698
2	2015-04-01 00:00:00	CH	31.318182
3	2015-04-01 00:00:00	CZ	35.504194
4	2015-04-01 00:00:00	DE	212.037500
5	2015-04-01 00:00:00	DK	48.300000

```
In [124]: data[["Date", "Isocode", "QualifiedAudience"]].groupby(["Date"], as_index=False).sum().plot(kind="line")
```

Out[124]: <matplotlib.axes.\_subplots.AxesSubplot at 0x11695e860>



- the group\_by function needs a apply (here the apply is mean)



# 4 DataFrame endless possibilities

- List of useful functions:
  - `df.sort_values(by, axis, ascending)`
  - `serie.value_counts()`
  - `pd.pivot_table(data, values, index, columns, aggfunc)`
  - `df.rename(columns)`
  - `df.shape`
  - `df.unique()`
  - `pd.crosstab(serie1, serie2)`
  - `df.dropna()` / `df.fillna()`
  - `pd.concat(dataframes)`
  - `df.tail(n)`