Data Science Lecture

By Raymond Houé Ngouna

(raymond.houe-ngouna@enit.fr)

Analysis and visualization: Pandas basics

Lecture objectives

To understand and analyse data through visualization

Main concepts covered:

- Learn Pandas basics
- Data import/export
- Data cleansing
- Data plotting
- Navigating
- Data replacing

Basic features

Importing/exporting data

API reference: https://pandas.pydata.org/docs/reference/index.html

```python
import pandas as pd
```

Importing data from a file

- Import an excel file: `pd.read_excel()`
- Import a csv file: `pd.read_csv()`
- Import a json file: `pd.read_json()`
- Import an html file: `pd.read_html()`
- Import an sql file: `pd.read_sql()`

```python
# import raw data from excel file
titanic = pd.read_excel('../DATA/titanic3.xls')

# make 2 copies of the raw data for further purposes
df = titanic.copy()
df_copy = titanic.copy()
```

```python
# display the data shape
df.shape
```

```python
# display the first 5 samples
# - to change the default value i.e. 5, enter another value
# - to see the last 5 samples -> df.tail()
df.head()
```

Variables meaning

- **pclass:** a proxy for socio-economic status (SES)
  - 1st = Upper,
  - 2nd = Middle,
  - 3rd = Lower,
- **age:** age is fractional if less than 1. If the age is estimated, is it in the form of xx.5
- **sibsp:** the dataset defines family relations in this way...
  - Sibling = brother, sister, stepbrother, stepsister,
  - Spouse = husband, wife (mistresses and fiancés were ignored)
- **parch:** the dataset defines family relations in this way...
  - Parent = mother, father,
  - Child = daughter, son, stepdaughter, stepson,
  - Some children travelled only with a nanny, therefore parch=0 for them.
- **ticket:** Ticket number
- **fare:** Passenger fare
- **cabin:** Cabin number
- **embarked:** Port of Embarkation
  - C = Cherbourg,
  - Q = Queenstown,
  - S = Southampton

Exporting data in a file

- Export to an excel file: `df.to_excel()`
- Export to a csv file: `df.to_csv()`
- Export to a json file: `df.to_json()`
- Export to an html file: `df.to_html()`
- Export to an sql file: `df.to_sql()`
- Export to a laxtex file: `df.to_latex()`

```python
# export dataframe to a csv file
df.to_csv('../DATA/titanic_export.csv')
```

Useful functions

```python
# useful info of the dataset
# number of non null values, type of each variable ...
df.info()
```

```python
# basic statistics : total, mean ... of each variable
df.describe()
```

Data cleansing

- Remove columns: `df.drop(['col1','col2',...])`, which may cause loss of information
- Delete underlying rows: `df.dropna(axis=0)`, which may cause loss of information
- Fill missing with a default value: `df.fillna(df['age'].mean())`, which may introduce biases

```python
# let's remove unnecessary columns
df = df.drop(['name', 'sibsp', 'parch', 'ticket', 'fare', 'cabin',
              'embarked', 'boat', 'body', 'home.dest'], axis=1)
```

```python
"""
Same result with the following instruction (without assignment):
df.drop(['name', 'sibsp', 'parch', 'ticket', 'fare', 'cabin',
        'embarked', 'boat', 'body', 'home.dest'], axis=1, inplace=True)
"""
```

```python
# let's check the result by viewing the first 5 samples
df.head()
```

```python
df.info()
```

```python
# remove of all samples with missing values
# and display the shape to see the change
df = df.dropna(axis=0)
df.shape
```

```python
# new infos on the modifed dataset
df.describe()
```

```python
# statistics on the second copy of the raw data
df_copy.describe()
```

```python
# let's remove few columns, keeping the 'name' column
df_copy.drop(['sibsp', 'parch', 'ticket', 'fare', 'cabin',
              'embarked', 'boat', 'body', 'home.dest'], axis=1)
```

```python
# display the data to see the changes
# nothing happened -> need to specify 'inplace' to True
df_copy
```

```python
# setting 'inplace' parameter to True
df_copy.drop(['sibsp', 'parch', 'ticket', 'fare', 'cabin',
              'embarked', 'boat', 'body', 'home.dest'], axis=1, inplace=True)
```

```python
# result as expected
df_copy
```

Group_by() and value_counts()

```python
# group of data (1st copy the raw data) by 'sex'
# the result is a dataFrame (we will see the meaning later)
df.groupby(['sex'])
```

```python
# with the grouped data (by sex), compute the mean()
df.groupby(['sex']).mean()
```

```python
# same with nested group -> by 'sex' -> for each 'sex' -> by 'pclass'
df.groupby(['sex','pclass']).mean()
```

```python
# for a given parameter (column)
# count the number of values per category
df['pclass'].value_counts()
```

```python
# count the number of values per category of 'pcalss'
# only for passengers aging less than 18 (boolean indexing)
df[df['age'] < 18]['pclass'].value_counts()
```

Data plotting

Pandas is built on top of Matplotlib !

- `df.plot()`
- `df.plot.bar()`
- `df.hist(bins=...)`
- `df.scatter(x=..., y=...)`
- `pd.plotting.scatter_matrix(df)`

```python
# display a barplot of numbers of passengers per 'pclass'
# 1- count the number per category
# 2- call the plot you need
df['pclass'].value_counts().plot.bar()
```

```python
# same for the 'survived' column
# NB: add the ';' add the end of the call to avoid
# uggly message above the figure
df['survived'].value_counts().plot.bar();
```

```python
# display the histogram of the passengers ages
df['age'].hist();
```

```python
# customize the number of 'bins'
df['age'].hist(bins=20);
```

```python
# scatter matrix plot
# - the diagonal > histograms
# - elsewhere -> scatter plots
pd.plotting.scatter_matrix(df);
```

Exercises

```python
""" dont'remove this cell -> create your own to check the result """
# use the original (raw) dataframe -> 'titanic'
# plot on the same figure
# (1) 1st plot -> the histogram passengers ages who survived
# (2) 2nd plot -> the histogram passengers ages who did not
# is there any correlation between 'age' and 'survival'?
```

```python
# complete code here
```

```python
""" dont'remove this cell -> create your own to check the result """
# same exercise with passengers 'fare'
# is there any correlation between passengers 'fare' and their 'survival'?
```

```python
# complete code here
```

Series and DataFrames

- A Pandas **serie** is a NumPy 1D array which has an index axis (which is independent from Numpy).
- A Pandas **dataframe** is a dictionary of series.

Note: Pandas is composed of **series** and **dataframes**.
`df['column'] = serie`

```python
# let's recall the data for the illustration
# just the first 3 samples
df_copy.head(3)
```

```python
# let's change the default index -> use the name
df_copy.set_index('name')
```

```python
# let's retrieve only the passengers ages
# each column is a serie -> 1D array
df['age']
```

```python
# let's check the type
type(df['age'])
```

```python
ser1 = pd.Series([1,2,3],index = ['USA', 'Germany','Japan'])
ser1
```

```python
ser2 = pd.Series([10,20,30],index = ['USA', 'Germany','Japan'])
ser2
```

```python
ser3 = ser1 + ser2
ser3
```

```python
df_new = pd.DataFrame(data=[[200,'cat1'],[250,'cat2'],[300,'cat1']],
                      columns=['COL1','COL2'])
```

```python
df_new
```

Navigating through the data

- `df['age'] = serie` : *ndarray*
- `df['age'][0:18]` : *indexing*
- `df['age'] < 18` : *mask*
- `df[df['age'] < 18] = dataFrame` : *boolean indexing*

```python
# indexing
df['age'][0:18]
```

```python
# masking
df['age'] < 18
```

```python
# boolean indexing
df[df['age'] < 18]
```

```python
# group function after boolean indexing
df[df['age'] < 18].groupby(['sex','pclass']).mean()
```

Localization methods fo navigating through data

- **Localisation by index**
  `df.iloc[5:10,0:4]`
- **Localisation (by column)**
  `df.loc[5:10,'column']`

```python
# direct nagivation on indexes not allowed
df[5:10,0:4]
```

Navigation by index

```python
# iloc method allow it
df.iloc[5:10,0:4]
```

```python
# iloc navigation on a specific axis
# with all data on the other axis
df.iloc[5:10]
```

```python
# let's make make it more explicit -> recommended
df.iloc[5:10,:]
```

Navigation by column

```python
# explicit indexing on one axis
# implicitly -> all data on the other axis
df.loc[5:10]
```

```python
# more explicit navigation
df.loc[5:10,:]
```

```python
# noy allowed -> only for index localisation
df.loc[5:10,0:3]
```

```python
# navigating through many columns at once
df.loc[5:10,['pclass', 'survived', 'sex', 'age']]
```

Exercise

```python
"""
For the following exercises, don't delete the cells with the expected results
Create your own in order the chech the matching with your result
use df_copy (2nd copy of the raw data)
"""
```

```python
# create 4 categories of age using boolean indexing an loc() function:
# age <= 20 -> 0
# 20 < age <= 30 -> 1
# 30 < age <= 40 -> 2
# age > 40 -> 3

# display the result as a dataframe
# complete code below


# display the resulting dataframe
df_copy
```

```python
# let Pandas count the values of each of the created categories of ages
df_copy['age'].value_counts()
```

Replacing data with new values

- `df['sex'].map({'male':0, 'female':1})`
- `df['sex'].replace(['male','female'],[0,1])`
- `df['sex'].astype('category').cat.codes`

```python
df_new
```

```python
df_new['COL2'] = df_new['COL2'].astype('category').cat.codes
```

```python
df_new
```

```python
df_new['COL2'] = df_new['COL2'].replace([0,1],['cat1','cat2'])
```

```python
df_new
```

```python
df_new['COL2'] = df_new['COL2'].map({'cat1':0, 'cat2':1})
```

```python
df_new
```

```python
df_new['COL2'] = df_new['COL2'].map(lambda x:x+1)
```

```python
df_new
```

Congratulations !

```python
from IPython.core.display import HTML
def css_styling():
    styles = open("../styles/custom.css", "r").read()
    return HTML(styles)
css_styling()
```

```python
df_copy.loc[(df_copy['age'] > 20) & (df_copy['age'] <= 30), 'age'] = 1
```

```python
df_copy.loc[df_copy['age'] <= 20, 'age'] = 0
df_copy.loc[(df_copy['age'] > 20) & (df_copy['age'] <= 30), 'age'] = 1
df_copy.loc[(df_copy['age'] > 30) & (df_copy['age'] <= 40), 'age'] = 2
df_copy.loc[df_copy['age'] > 40, 'age'] = 3

df_copy
```

```python
df_copy['age'].value_counts()
```

```python
import matplotlib.pyplot as plt

plt.figure(figsize=(10,5))

plt.hist(titanic['fare'][titanic['survived'] == 0], bins=30,
         alpha=0.5, edgecolor='black', label='Did not survive', color='r')
plt.hist(titanic['fare'][titanic['survived'] == 1], bins=30,
         alpha=0.5, edgecolor='black', label='Survived', color='g')

plt.ylabel('Fare')
plt.legend();
```

```python
import matplotlib.pyplot as plt

plt.figure(figsize=(10,5))

plt.hist(df['age'][df['survived'] == 0], bins=30,
         alpha=0.5, edgecolor='black', label='Did not survive', color='r')
plt.hist(df['age'][df['survived'] == 1], bins=30,
         alpha=0.5, edgecolor='black', label='Survived', color='g')

plt.ylabel('Passengers ages')
plt.legend();
```

In [1]:
```python
from IPython.core.display import HTML
def css_styling():
    styles = open("../styles/custom.css", "r").read()
    return HTML(styles)
css_styling()
```

Out[1]: