Introduzione alla *Data Science* in Python

Advanced Computer Programming

Prof. Luigi De Simone

Sommario

- pandas: *Series* e *DataFrame*
- Gestione dei file *csv*
- Data visualization in Python: *matplotlib*

Riferimenti

- Tony Gaddis. **Introduzione a Python**. 5° ed. – Pearson, 2021
- Paul J. Deitel, Harvey M. Deitel. **Introduzione a Python. Per l'informatica e la data science.** Pearson, 2021
- **Python: How to Think Like a Computer Scientist interactive edition** https://runestone.academy/runestone/books/published/thinkcspy/index.html
- Allen Downey. **Think Python** - https://greenteapress.com/thinkpython2/thinkpython2.pdf

pandas: Series e DataFrames

- **pandas** (panel data) è la libreria più popolare per gestire ***Big Data***
  - Supporto per tipi di dati misti, indicizzazione personalizzata, dati mancanti, dati non strutturati in modo coerente e dati che devono essere manipolati in moduli appropriati per i database e i pacchetti di analisi dei dati, etc.
- Fornisce le **`Series`** per *collection* (tuple, liste, dictionary) unidimensionali e **`DataFrames`** per collection bidimensionali
- È possibile utilizzare **`MultiIndex`** per manipolare dati multidimensionali nel contesto di Series e DataFrames.
- *Series* e *DataFrames* **utilizzano gli `array` NumPy** e sono tipi di dato validi per molte operazioni consentite in NumPy e viceversa
- Installazione
  - `pip install pandas`

pandas **Series**

- Una `Series` è un `array` unidimensionale migliorato
- Mentre gli `array` utilizzano solo indici interi partendo da zero, le Series **supportano l'indicizzazione personalizzata** (indici non interi come le stringhe)
- Le `Series` offrono anche funzionalità aggiuntive che le rendono più **convenienti per molte attività orientate alla *data science***
    - Ad esempio, `Series` potrebbe avere dati mancanti e molte operazioni `Series` ignorano i dati mancanti per impostazione predefinita.

pandas **Series**

- **Creazione di una `Series`, indice di default (0)**

```
import pandas as pd
In [2]: grades = pd.Series([87, 100, 94])

In [3]: grades
Out[3]:
0     87
1    100
2     94
dtype: int64
```

- **Creazione di una `Series`, con tutti gli elementi uguali ad un valore**

```
In [4]: pd.Series(98.6, range(3))
Out[4]:
0 98.6
1 98.6
2 98.6
dtype: float64
```

- **Accesso agli elementi di una `Series`**

```
In [5]: grades[0]
Out[5]: 87
```

pandas **Series**

- **Statistica descrittiva di una `Series`.** Oltre ad utilizzare le classiche funzioni `mean()`, `min()`, `max()`, `std()`, etc. è possibile utilizzare `describe()`

```
In [11]: grades.describe()
Out[11]:
count      3.000000
mean      93.666667
std        6.506407
min       87.000000
25%       90.500000
50%       94.000000
75%       97.000000
max      100.000000
dtype: float64
```

Il 25%, 50% e 75% sono quartili:
• 50% rappresenta la mediana dei valori ordinati.
• 25% rappresenta la mediana della prima metà dei valori ordinati.
• 75% rappresenta la mediana della seconda metà dei valori ordinati.

pandas **Series**

- **Creazione di una `Series`, indice custom**

```
grades = pd.Series([87, 100, 94], index=['Wally', 'Eva', 'Sam’])
In [13]: grades
Out[13]:
Wally     87
Eva      100
Sam       94
dtype: int64
```

- **Creazione di una `Series`, inizializzata con un dictionary**

```
In [14]: grades = pd.Series({'Wally': 87, 'Eva': 100, 'Sam': 94})
In [15]: grades
Out[15]:
Wally     87
Eva      100
Sam       94
dtype: int64
```

- **Accesso agli elementi di una `Series` con indice custom**

```
In [16]: grades['Eva']
Out[16]: 100
In [17]: grades.Wally
Out[17]: 87
In [19]: grades.values
Out[19]: array([ 87, 100,  94])
```

pandas **Series**

- **Creazione di una `Series` di stringhe e manipolazione**

```
In [20]: hardware = pd.Series(['Hammer', 'Saw', 'Wrench'])
In [21]: hardware
Out[21]:
0
1
2
dtype: object

In [22]: hardware.str.contains('a')
Out[22]:
0     True
1     True
2    False
dtype: bool

In [23]: hardware.str.upper()
Out[23]:
0    HAMMER
1       SAW
2    WRENCH
dtype: object
```

pandas **DataFrame**

- Un `DataFrame` è un **array bidimensionale avanzato**
  - Indici di riga e colonna personalizzati
  - Operazioni e funzionalità aggiuntive convenienti per molte attività orientate alla ***data science***
  - Supportano anche i dati mancanti

- **Ogni colonna in un `DataFrame` è una `Series`**
  - Le serie che rappresentano ogni colonna possono contenere diversi tipi di elementi

pandas **DataFrame**

- **Creazione di un DataFrame da dictionary**

```
In [1]: import pandas as pd
In [2]: grades_dict = {'Wally': [87, 96, 70], 'Eva': [100, 87, 90],
   ...:                'Sam': [94, 77, 90], 'Katie': [100, 81, 82],
   ...:                'Bob': [83, 65, 85]}
   ...:
In [3]: grades = pd.DataFrame(grades_dict)
   Wally  Eva  Sam  Katie  Bob
0     87  100   94    100   83
1     96   87   77     81   65
2     70   90   90     82   85
```

- pandas visualizza i DataFrames in **formato tabellare**
  - Le **chiavi** del dizionario diventano i **nomi delle colonne**
  - I **valori** associati a ciascuna chiave diventano i valori degli elementi nella colonna corrispondente
- Per impostazione predefinita, gli indici di riga sono interi generati automaticamente a partire da 0

pandas **DataFrame**

- **Modificare l'indice di default di un `DataFrame` esistente**

```
In [5]: grades.index = ['Test1', 'Test2', 'Test3']
In [6]: grades
Out[6]:
       Wally  Eva  Sam  Katie  Bob
Test1     87  100   94    100   83
Test2     96   87   77     81   65
Test3     70   90   90     82   85
```

- **Accesso ad una colonna del `DataFrame`**

```
In [7]: grades['Eva']
Out[7]:
Test1    100
Test2     87
Test3     90
Name: Eva, dtype: int64

In [8]: grades.Sam
Out[8]:
Test1    94
Test2    77
Test3    90
Name: Sam, dtype: int64
```

pandas **DataFrame**

- **Selezionare le righe attraverso `loc` (specificare custom index) e `iloc` (zero-based index)**

```
In [9]: grades.loc['Test1']
Out[9]:
Wally     87
Eva      100
Sam       94
Katie    100
Bob       83
Name: Test1, dtype: int64

In [10]: grades.iloc[1]
Out[10]:
Wally    96
Eva      87
Sam      77
Katie    81
Bob      65
Name: Test2, dtype: int64
```

pandas **DataFrame**

- **Selezionare le righe attraverso *slices* e *liste* con `loc` (specificare custom index) e `iloc` (zero-based index)**

```
In [11]: grades.loc['Test1':'Test3']
Out[11]:
       Wally  Eva  Sam  Katie  Bob
Test1     87  100   94    100   83
Test2     96   87   77     81   65
Test3     70   90   90     82   85

In [12]: grades.iloc[0:2]
Out[12]:
       Wally  Eva  Sam  Katie  Bob
Test1     87  100   94    100   83
Test2     96   87   77     81   65

In [13]: grades.loc[['Test1', 'Test3']] # grades.iloc[[0, 2]]
Out[13]:
            Wally        Eva          Sam          Katie        Bob
Test1       87           100          94           100          83
Test3       70           90           90           82           85
```

pandas **DataFrame**

- **Selezionare sottoinsiemi di righe e colonne**

```
In [15]: grades.loc['Test1':'Test2', ['Eva', 'Katie']]
Out[15]:
       Eva  Katie
Test1  100    100
Test2   87     81

In [16]: grades.iloc[[0, 2], 0:3]
Out[16]:
       Wally  Eva  Sam
Test1     87  100   94
Test3     70   90   90
```

- **Boolean indexing**

```
In [17]: grades[grades >= 90]
Out[17]:
       Wally    Eva   Sam  Katie  Bob
Test1    NaN  100.0  94.0  100.0  NaN
Test2   96.0    NaN   NaN    NaN  NaN
Test3    NaN   90.0  90.0    NaN  NaN
```

I voti per i quali la condizione è **False** sono rappresentati come NaN (not a number) nel nuovo DataFrame.
NaN è la notazione di pandas per i valori mancanti.

pandas **DataFrame**

- **Manipolazione di una cella del `DataFrame` per riga e colonna tramite `at` e `iat`**

```
# Accesso
In [19]: grades.at['Test2', 'Eva']
Out[19]: 87
In [20]: grades.iat[2, 0]
Out[20]: 70

# Scrittura
In [21]: grades.at['Test2', 'Eva'] = 100
In [22]: grades.at['Test2', 'Eva']
Out[22]: 100
In [23]: grades.iat[1, 2] = 87
In [24]: grades.iat[1, 2]
Out[24]: 87.0
```

pandas **DataFrame**

- **Statistiche descrittive, metodo `describe()`**

```
grades.describe()
```

	Wally	Eva	Sam	Katie	Bob
count	3.000000	3.000000	3.000000	3.000000	3.000000
mean	84.333333	92.333333	87.000000	87.666667	77.666667
std	13.203535	6.806859	8.888194	10.692677	11.015141
min	70.000000	87.000000	77.000000	81.000000	65.000000
25%	78.500000	88.500000	83.500000	81.500000	74.000000
50%	87.000000	90.000000	90.000000	82.000000	83.000000
75%	91.500000	95.000000	92.000000	91.000000	84.000000
max	96.000000	100.000000	94.000000	100.000000	85.000000

- **Impostazione della precision attraverso `set_option()`**

```
pd.set_option('display.precision', 2) # or simply ‘precision’ depending on Python version
grades.describe()
```

	Wally	Eva	Sam	Katie	Bob
count	3.00	3.00	3.00	3.00	3.00
mean	84.33	92.33	87.00	87.67	77.67
std	13.20	6.81	8.89	10.69	11.02
min	70.00	87.00	77.00	81.00	65.00
25%	78.50	88.50	83.50	81.50	74.00
50%	87.00	90.00	90.00	82.00	83.00
75%	91.50	95.00	92.00	91.00	84.00
max	96.00	100.00	94.00	100.00	85.00

pandas **DataFrame**

- **Trasposizione del `DataFrame` con l'attributo T**

```
In [29]: grades.T
Out[29]:
       Test1  Test2  Test3
Wally     87     96     70
Eva      100     87     90
Sam       94     77     90
Katie    100     81     82
Bob       83     65     85
```

- **Ordinamento per indice delle righe**

```
In [32]: grades.sort_index(ascending=False) # in maniera discendente
Out[32]:
       Wally  Eva  Sam  Katie  Bob
Test3     70   90   90     82   85
Test2     96   87   77     81   65
Test1     87  100   94    100   83
```

pandas **DataFrame**

- **Ordinamento per indice delle colonne**

```
In [33]: grades.sort_index(axis=1)
Out[33]:
       Bob  Eva  Katie  Sam  Wally
Test1   83  100    100   94     87
Test2   65   87     81   77     96
Test3   85   90     82   90     70
```

- **Ordinamento per valore delle colonne**

```
In [34]: grades.sort_values(by='Test1', axis=1, ascending=False)
Out[34]:
       Eva  Katie  Sam  Wally  Bob
Test1  100    100   94     87   83
Test2   87     81   77     96   65
Test3   90     82   90     70   85
```

Il modulo **csv**

- Il modulo **csv** fornisce funzioni per lavorare con i **file CSV (Comma Separated Value)**
  - Altre librerie Python hanno il supporto CSV integrato
- Per **scrivere** un file csv
  - Aprire un file con estensione .csv
  - Usare la funzione **`writer`** del modulo csv per ottenere un oggetto *scrittore* di dati CSV nell'oggetto file specificato
  - Effettuare una chiamata al metodo **`writerow`** sull'oggetto writer che riceve in *ingresso un iterabile* (e.g., una lista) che specifica i dati da scrivere
  - Di default, **`writerow`** delimita i valori con virgole, ma si possono specificare delimitatori personalizzati
- Per **leggere** un file csv
  - La funzione **`reader`** del modulo csv restituisce un oggetto *lettore* che legge i dati in formato CSV dall'oggetto file specificato
  - Possiamo iterare sull'oggetto restituito da **`reader`** per ottenere le linee del file dei valori delimitati da virgole

Il modulo **csv**

- **Scrittura**

```
In [1]: import csv
In [2]: with open('accounts.csv', mode='w', newline='') as accounts:
   ...:     writer = csv.writer(accounts)
   ...:     writer.writerow([100, 'Jones', 24.98])
   ...:     writer.writerow([200, 'Doe', 345.67])
   ...:     writer.writerow([300, 'White', 0.00])
   ...:     writer.writerow([400, 'Stone', -42.16])
   ...:     writer.writerow([500, 'Rich', 224.62])
   ...:
```

- **Lettura**

```
In [3]: with open('accounts.csv', 'r', newline='') as accounts:
            print(f'{"Account":<10}{"Name":<10}{"Balance":>10}')
            reader = csv.reader(accounts)
            for record in reader:
                        account, name, balance = record
                        print(f'{account:<10}{name:<10}{balance:>10}')
```

```
Account   Name         Balance
100       Jones          24.98
200       Doe           345.67
300       White            0.0
400       Stone         -42.16
500       Rich          224.62
```

NOTA:
print(f'{expression}') è la notazione **formatted string literals**, che permette di includere i valori di espressioni Python all'interno di una stringa anteponendo la stringa **f** o **F** e scrivendo l'espressioni con la notazione **{expression}**

pandas e CSV

❖ Caricare CSV in DataFrame

- Per **caricare un dataset CSV in un *DataFrame*** pandas con la funzione **`read_csv`**
- **Esempio**

```
import pandas as pd
df = pd.read_csv('accounts.csv', names=['account', 'name', 'balance'])
```

- La keyword **`names`** specifica i nomi delle colonne del *DataFrame*
  - Se non si fornisce questo argomento, *read_csv* presuppone che **la prima riga del file CSV sia un elenco di nomi di colonne delimitato da virgole**
- Nota: la keyword **`skiprows`** permette di non considerare le prime N righe

❖ Salvare DataFrame in CSV

- Per **salvare un DataFrame in un file CSV** è possibile usare il metodo **`to_csv`**

```
df.to_csv('accounts_from_dataframe.csv', index=False)
```

- In to_csv, **`index=False`** indica che i nomi di riga non vengono scritti nel file.
  Il file risultante contiene i nomi delle colonne come prima riga.

pandas e CSV

- **Caricare un dataset CSV da url**

```
import pandas as pd
titanic =
    pd.read_csv('https://vincentarelbundock.github.io/Rdatasets/csv/carData/TitanicSurvival.csv')
```

	Unnamed: 0	survived	sex	age	passengerClass
0	Allen, Miss. Elisabeth Walton	yes	female	29.00	1st
1	Allison, Master. Hudson Trevor	yes	male	0.92	1st
2	Allison, Miss. Helen Loraine	no	female	2.00	1st
3	Allison, Mr. Hudson Joshua Crei	no	male	30.00	1st
4	Allison, Mrs. Hudson J C (Bessi	no	female	25.00	1st
...	...	...	...	...	...
1304	Zabour, Miss. Hileni	no	female	14.50	3rd
1305	Zabour, Miss. Thamine	no	female	NaN	3rd
1306	Zakarian, Mr. Mapriededer	no	male	26.50	3rd
1307	Zakarian, Mr. Ortin	no	male	27.00	3rd
1308	Zimmerman, Mr. Leo	no	male	29.00	3rd

1309 rows × 5 columns

pandas e CSV

- I Metodi **`head`** e **`tail`** sono utili **per non stampare a video tutto il dataset**
  - `titanic.head()` ritorna le **prime 5 righe del dataset di default**

	Unnamed: 0	survived	sex	age	passengerClass
0	Allen, Miss. Elisabeth Walton	yes	female	29.00	1st
1	Allison, Master. Hudson Trevor	yes	male	0.92	1st
2	Allison, Miss. Helen Loraine	no	female	2.00	1st
3	Allison, Mr. Hudson Joshua Crei	no	male	30.00	1st
4	Allison, Mrs. Hudson J C (Bessi	no	female	25.00	1st

  - `titanic.tail()` ritorna le **ultime 5 righe del dataset di default**

	Unnamed: 0	survived	sex	age	passengerClass
1304	Zabour, Miss. Hileni	no	female	14.5	3rd
1305	Zabour, Miss. Thamine	no	female	NaN	3rd
1306	Zakarian, Mr. Mapriededer	no	male	26.5	3rd
1307	Zakarian, Mr. Ortin	no	male	27.0	3rd
1308	Zimmerman, Mr. Leo	no	male	29.0	3rd

- **Cambiare i nomi di colonne**

```
titanic.columns = ['name', 'survived', 'sex', 'age', 'class']
```

Data visualization in Python

- La ***data visualization*** ci aiuta a "conoscere" i dati da analizzare al di là della semplice visualizzazione di dati grezzi.
- Le librerie più utilizzare in Python sono **`Matplotlib`** e **`Seaborn`**
  - Permettono di creare grafici, istogrammi, boxplot, etc.
  - *Seaborn* è costruita al di sopra di Matplotlib e semplifica molte operazioni di tale libreria

- Installazione
  - `pip install matplotlib`
  - `pip install seaborn`

Matplotlib

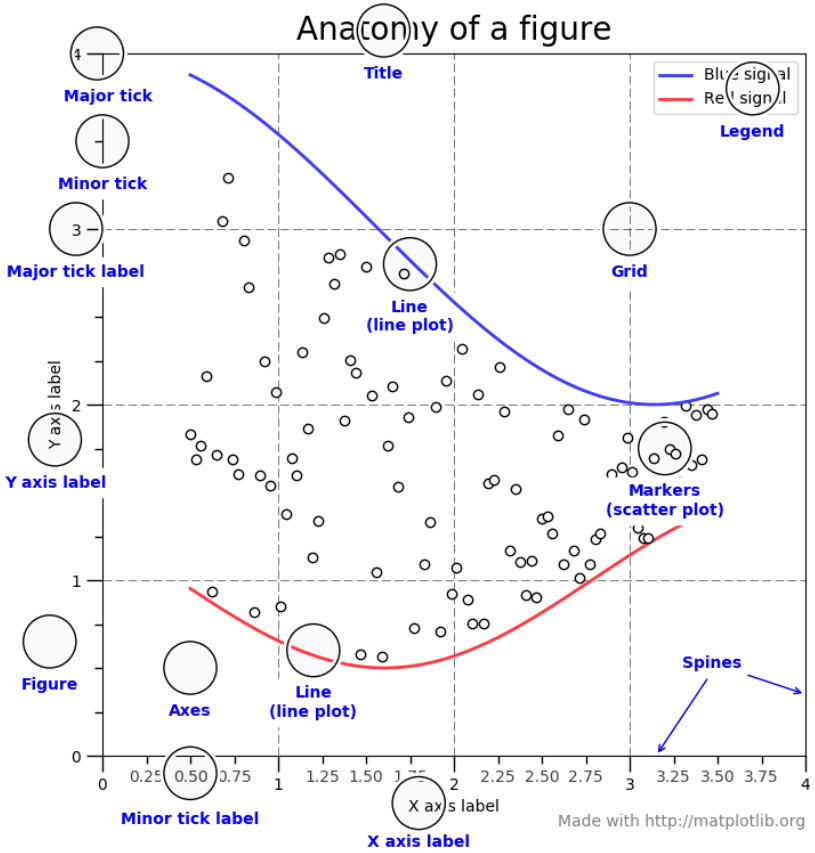
- Per tutti i grafici **`Matplotlib`**, possiamo iniziare a creare una figura e un asse

```
from matplotlib import pyplot as plt
import numpy as np

plt.style.use('seaborn-whitegrid')

fig = plt.figure()
ax = plt.axes()
```

- In Matplotlib, la figura (un'istanza della classe **`plt.figure`**) è un contenitore che contiene tutti gli oggetti che rappresentano assi, grafica, testo ed etichette

- Gli assi (un'istanza della classe **`plt.axes`**) è ciò che vediamo sopra: un riquadro di delimitazione con *ticks* ed etichette, che alla fine conterrà gli elementi del grafico che compongono la nostra visualizzazione.

Matplotlib


Anatomy of a figure
Title
Major tick
Minor tick
Major tick label
Legend
Grid
Line
(line plot)
Y axis label
Markers
(scatter plot)
Figure
Axes
Line
(line plot)
Spines
Minor tick label
X axis label
Made with http://matplotlib.org

Line plots

- Dopo aver creato un asse, possiamo usare la funzione **`ax.plot`** per tracciare alcuni dati e aggiungere info riguardo il titolo (**`set_title`**) e le etichette degli assi (**`set_xlabel`**, **`set_ylabel`**)

```
fig = plt.figure()
ax = plt.axes()

x = np.linspace(0, 10, 1000)
ax.plot(x, np.sin(x));

ax.set_title('Simple Plot')       # Add a title
ax.set_xlabel('x label')          # Add x label
ax.set_ylabel('y label');         # Add y label
```

Funzione del modulo **NumPy** che consente di produrre intervalli a virgola mobile equidistanti

- I primi due argomenti della funzione specificano i valori iniziali e finali nell'intervallo e il valore finale è incluso nell'array.
- Il terzo definisce il numero di valori a distanza uniforme da produrre (50 è il valore di default)

Line plots

- **Aggiungere la legenda**

```
ax.plot(x, np.sin(x), label = 'sin')
ax.plot(x, np.cos(x), label = 'cos’)
ax.legend()
```

- **Modificare il colore delle line dei plot**

```
ax.plot(x, np.sin(x), label = 'sin', color = 'red')   # specify color by name
ax.plot(x, np.cos(x), label = 'cos', color = 'g')     # short color code (rgbcmyk)
```

- **Modificare lo stile delle line dei plot**

```
ax.plot(x, np.sin(x), label = 'sin', linestyle = 'dashed')
ax.plot(x, np.cos(x), label = 'cos', linestyle = 'dotted')
ax.plot(x, np.sin(x+1), label = 'cos', linestyle = 'dashdot')
```

- **Limitare gli assi**

```
ax.set_xlim(-5, 15)
ax.set_ylim(-3, 3);
```

Line plots

- **Subplot**

```
fig, axs = plt.subplots(2, 2, figsize=(10,6)) # a figure with a 2x2 grid of Axes
x = np.linspace(0, 10, 1000)

axs[0,0].plot(x, np.sin(x))
axs[0,1].plot(x, np.cos(x), color = 'maroon')
axs[1,0].plot(x, np.sin(x+2), color = 'blue')
axs[1,1].plot(x, np.sin(x+4), color = 'green');
```