UNIVERSITÀ di VERONA

Linguaggio di Programmazione Python

9/05/2022

Giulio Mazzi

giulio.mazzi@univr.it

Pandas - Introduzione

Pandas (Python for Data Analysis) è un pacchetto Python che si basa su NumPy. Fornisce le funzionalità necessarie a dare una struttura ai nostri dati. È progettato per rendere facile e intuitiva la gestione dei dati.

Ha due strutture dati principali: **Series** (monodimensionali) e **DataFrame** (bidimensionali)

Un frame di dati pandas è, in pratica, un foglio di lavoro "intelligente": una tabella che riporta etichette per le colonne (le variabili), le righe (le osservazioni) e una ricca collezione di operazioni interne.


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

Pandas Series

Possiamo creare una pandas.Series a partire da una lista.

Al suo interno, usa un ndarray per strutturare I dati.

Nel caso di default, le Series usano come indice una sequenza di numeri (pensate ad un enumerate in python).

I dati contenuti nella lista devono essere di tipo omogeneo (in questo caso float64). Ma ci sono differenza importanti rispetto a numpy, specialmente nella gestione dei dati mancanti.

```
abitanti_veneto = pd.Series([
    840.153,
    198.725,
    931.290,
    229.376,
    877.405,
    927.682,
    853.460]
)
```

```
print(abitanti_veneto)
```

```
0    840.153
1    198.725
2    931.290
3    229.376
4    877.405
5    927.682
6    853.460
dtype: float64
```

Pandas - Nomi e Indici

Spesso non è facile e pratico lavorare con semplici sequenze di numeri. Per questo è possibile associare un nome alla Serie. Si possono anche usare indici non numerici (in genere stringhe).

Gli indici si comportano in modo simile a dizionari (o nello specifico, a ordered_dict).

```
abitanti_veneto.name = 'Numero abitanti Veneto per provincia'
abitanti_veneto.index = [
'Venezia',
'Belluno',
'Padova',
'Rovigo',
'Treviso',
'Verona',
'Vicenza',
]
print(abitanti_veneto)
```

```
Venezia    840.153
Belluno    198.725
Padova     931.290
Rovigo     229.376
Treviso    877.405
Verona     927.682
Vicenza    853.460
Name: Numero abitanti Veneto per provincia, dtype: float64
```

Pandas - dati mancanti

Pandas gestisce bene l'assenza di dati in una tabella. In particolare, "riempie I buchi" usando Nan, e adatta le sue operazioni a queste lacune.

```
pd.Series(abitanti_veneto, index=[
    'Verona', 'Vicenza', 'Padova', 'Milano', 'Venezia'])
```

```
Verona     927.682
Vicenza    853.460
Padova     931.290
Milano         NaN
Venezia    840.153
Name: Numero abitanti Veneto per provincia, dtype: float64
```

Pandas – slicing & fancy indexing

Possiamo usare tutte le funzionalità di indexing avanzate viste per NumPy.

Ricordiamoci però che gli indici possono anche non essere di tipo numerico!

```
# indexing (ritorna un singolo valore)
print(abitanti_veneto['Verona'])
```

Click to add text

```
927.682
```

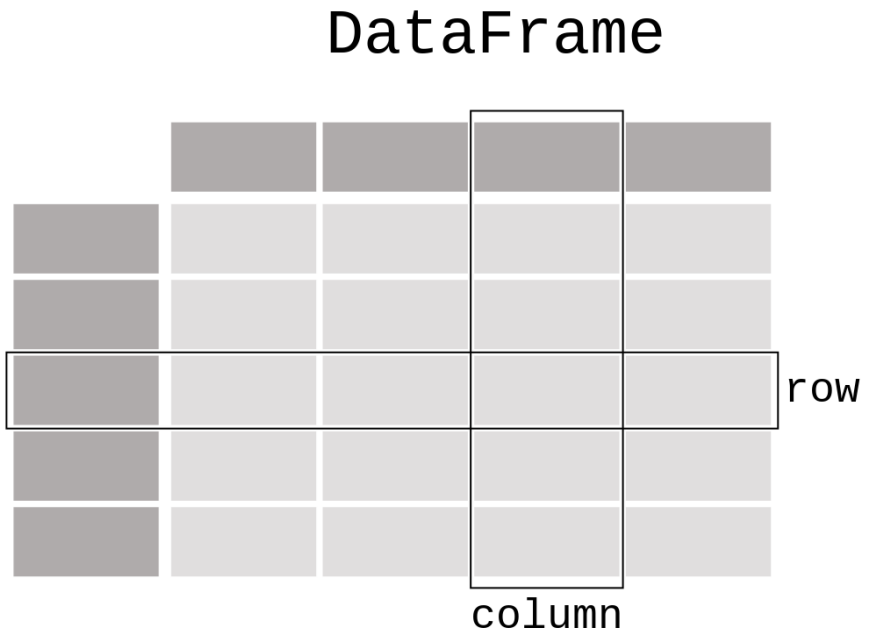
```
# multiplo (ritorna un'altra series)
print(abitanti_veneto[['Rovigo', 'Verona']])
```

```
Rovigo    229.376
Verona    927.682
Name: Numero abitanti Veneto per provincia, dtype: float64
```

Pandas DataFrame

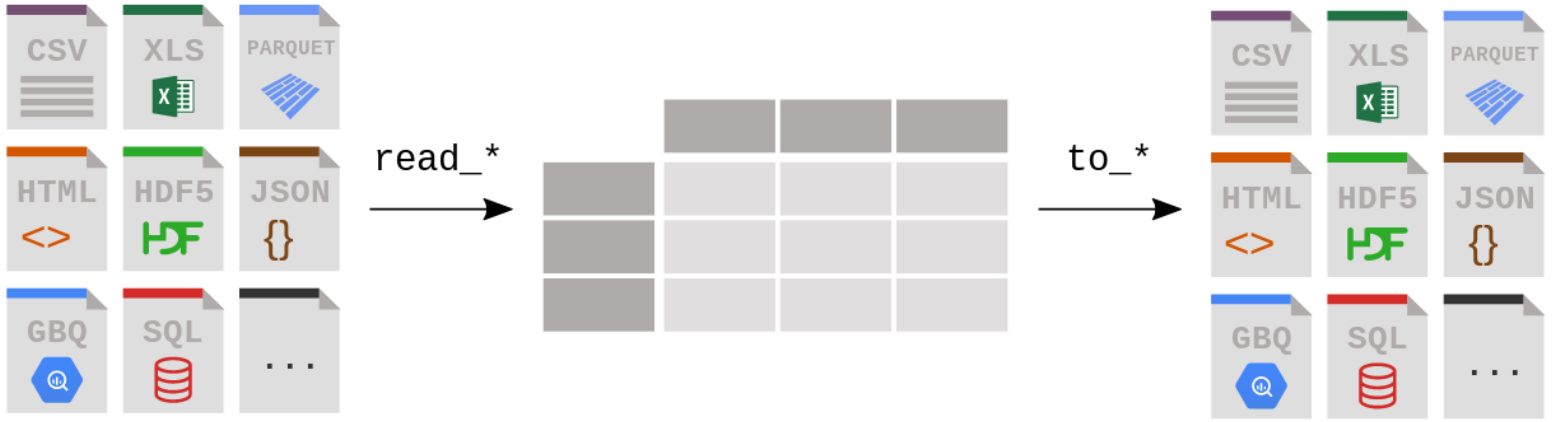
Un DataFrame è una struttura dati bidimensionale che può salvare dati (anche di tipo diverso!) in colonne. Possiamo pensarla come alla tabella di un database o ad un foglio di calcolo.

Una Series è un DataFrame con una colonna.



Lettura da file

Possiamo leggere DataFrame (e Series) da moltissime fonti e salvare I risultati in molti formati.



Matplotlib - Introduzione

Rappresentare dati in tabella non è sufficiente, vogliamo poter creare grafici a partire dai nostri dati.

La libreria python più nota è matplotlib. In particolare, consideriamo il modulo pyplot.

Documentazione:

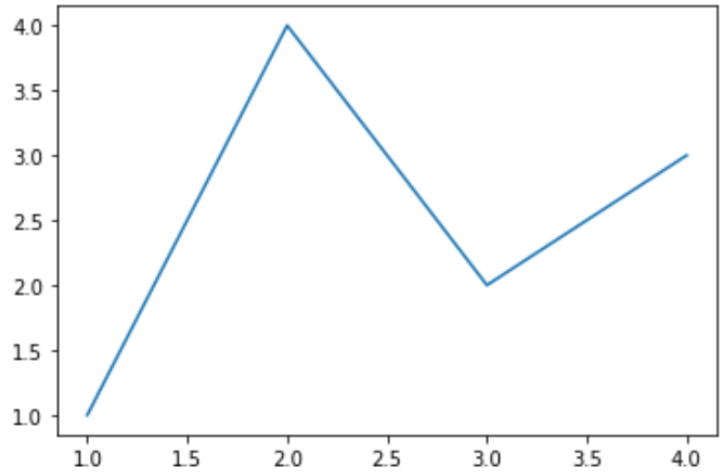
Click to add text

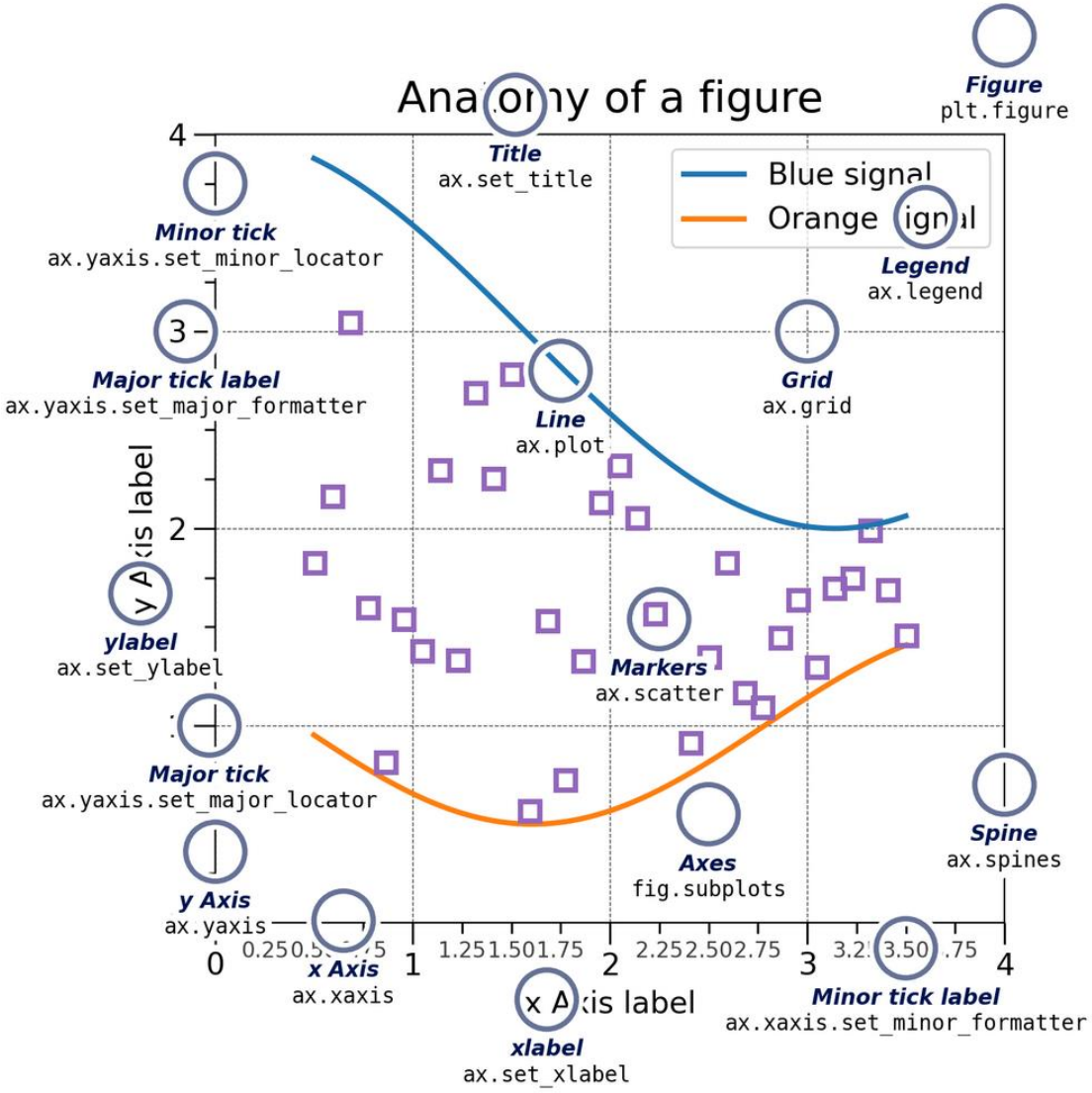
https://matplotlib.org/stable/tutorials/introductory/usage.html


```
import matplotlib.pyplot as plt
```

```
fig, ax = plt.subplots()
ax.plot([1, 2, 3, 4], [1, 4, 2, 3])
```

```
[<matplotlib.lines.Line2D at 0x7f111c539940>]
```





Matplotlib - Elementi

Possiamo accedere ai diversi elementi di un figura. I più importanti sono:

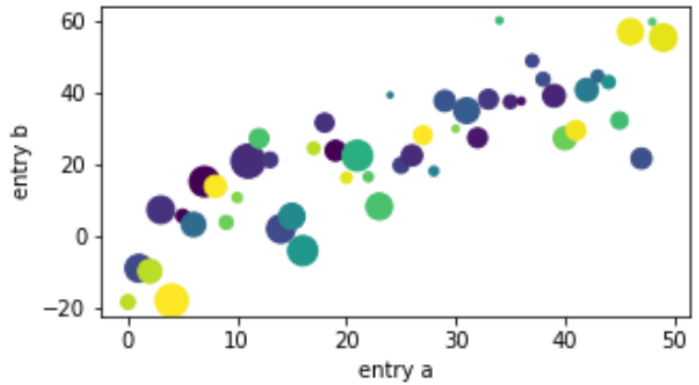
- Axis (x e y)
- Line
- Legend
- Title
- Grid

Matplotlib - Stampa

Matplotlib supporta molti tipi di grafici, tra cui scatterplot e istogrammi.Possiamo definire tutti i dettagli del grafico. In questo caso, specifichiamo il nome degli assi e i dati.
Possiamo specificare anche nome, legenda, griglia,...

```
# esempio di stampa
data = {'a': np.arange(50),
        'c': np.random.randint(0, 50, 50),
        'd': np.random.randn(50)}
data['b'] = data['a'] + 10 * np.random.randn(50)
data['d'] = np.abs(data['d']) * 100

fig, ax = plt.subplots(figsize=(5, 2.7))
ax.scatter('a', 'b', c='c', s='d', data=data)
ax.set_xlabel('entry a')
ax.set_ylabel('entry b');
```

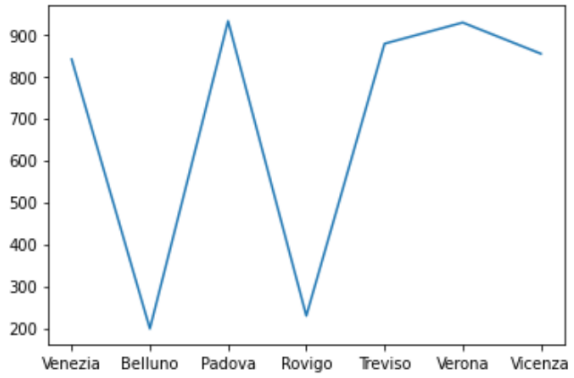


Integrazione con Pandas

Pandas integra direttamente matplotlib. Possiamo usare alcune funzioni di stampa sul dataframe.

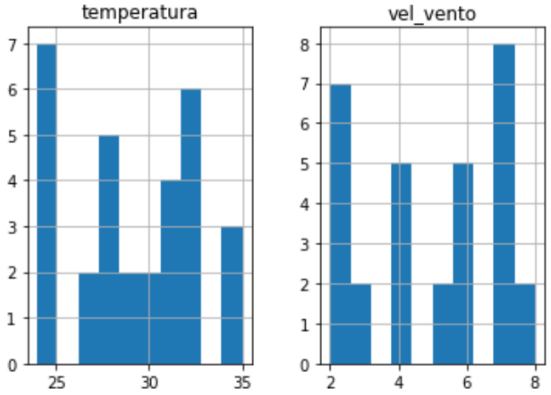
```
plt.figure()
abitanti_veneto.plot()
```

```
<AxesSubplot:>
```




```
plt.figure()
df.hist();
```

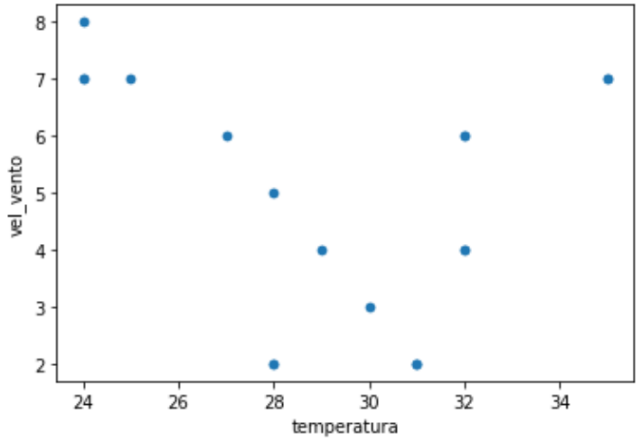
```
<Figure size 432x288 with 0 Axes>
```




```
plt.figure(); df.plot.scatter(x='temperatura', y='vel_vento')
```

```
<AxesSubplot:xlabel='temperatura', ylabel='vel_vento'>
```

```
<Figure size 432x288 with 0 Axes>
```



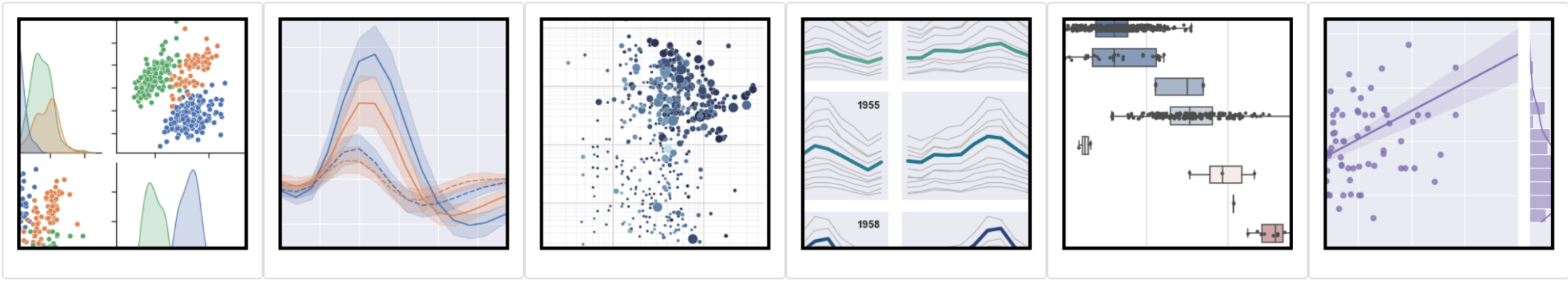
Seaborn

Matplotlib è una libreria di "basso" livello, ci richiede di specificare molti dettagli.

Per questo sono nate librerie più generali, basate su matplotlib, che facilitano il lavoro (e permettono di creare risultati visivamente più appaganti!).

Non ne vedremo nessuna al corso, ma la più nota è seaborn.

https://seaborn.pydata.org/



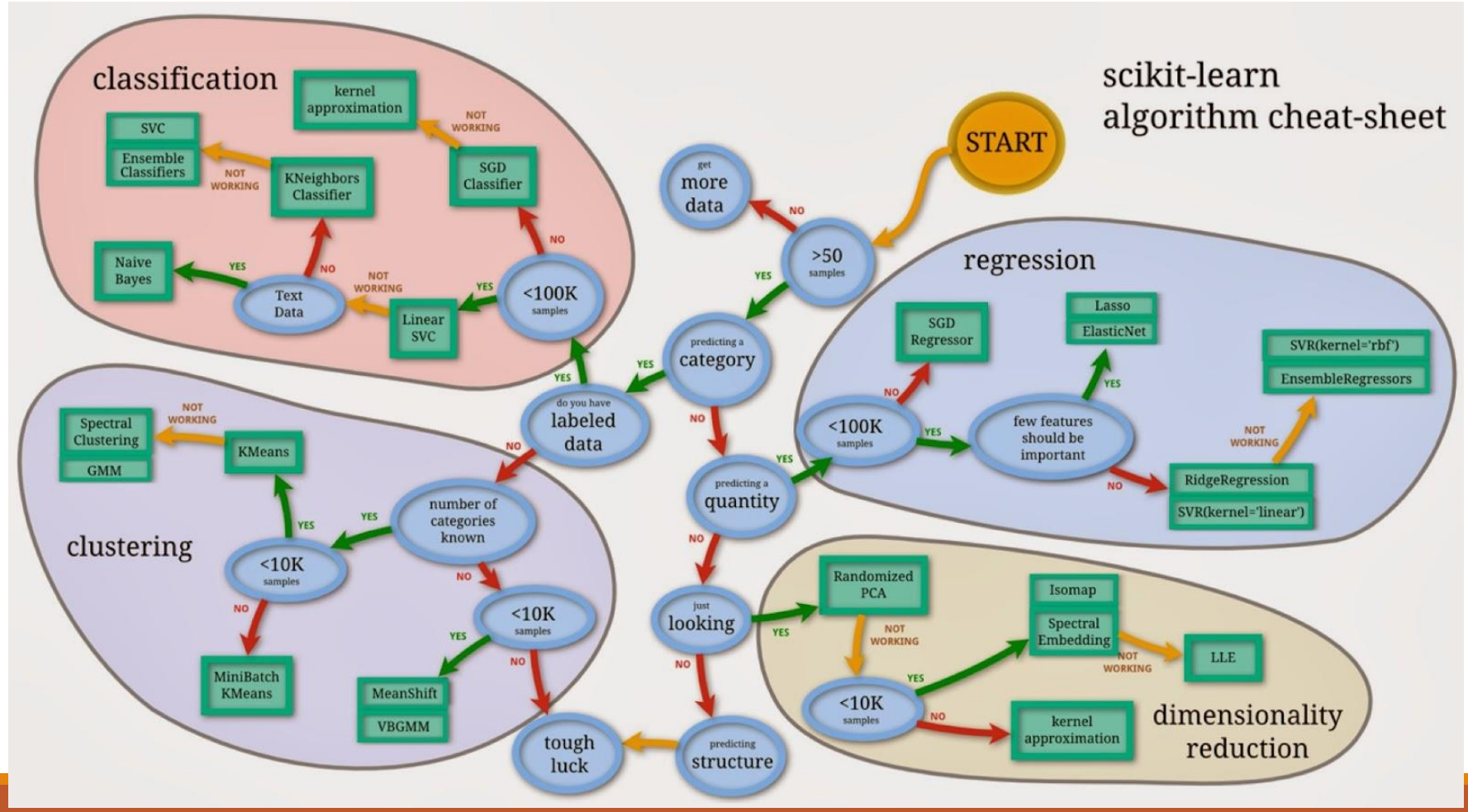
Scikit Learn - Introduzione

La branca dell'intelligenza artificiale che si occupa di apprendere dai dati si chiama machine learning (ML). In un problema di apprendimento, vogliamo predire il valore di un dato sconosciuto partendo da una collezione di dati già disponibili.

Python è uno degli strumenti più diffusi nel mondo dell'ML, e la libreria più nota per fare ML in python è scikit-learn (https://scikit-learn.org/stable/index.html).

Scikit implementa diverse tecniche di **supervised** e **unsupervised learning.** Offre inoltre algoritmi di **dimensionality reduction.**

Scikit learn - organizzazione



Supervised Learning

Nell'apprendimento supervisionato, partiamo da un dataset di dati già etichettati (ovvero, sappiamo già che valore aspettarci). Il problema di apprendimento consiste nel predire il valore di un nuovo dato.

Il problema della **classificazione** consiste nel identificare a quale classe appartiene un dato (per esempio, indovinare a quale cifra corrisponde la foto fatta ad una cifra scritta a mano)

Il problema della **regressione** consiste nel predire un valore numerico partendo dalle feature di un oggetto (per esempio, predire il costo di un appartamento considerando m^2, distanza dal centro, numero stanze ecc.)

Scelto l'algoritmo, usiamo sempre il metodo `fit` per addestrare l'algoritmo e `predict` per predire un nuovo valore.

Supervised learning - example

In [124]:
```
from sklearn import datasets

# scikit-learn offre dei dataset di default, utili per sperimentare
digits = datasets.load_digits()
```

In [104]:
```
# apprendimento supervisionato: i dati sono etichettati
digits.target
```

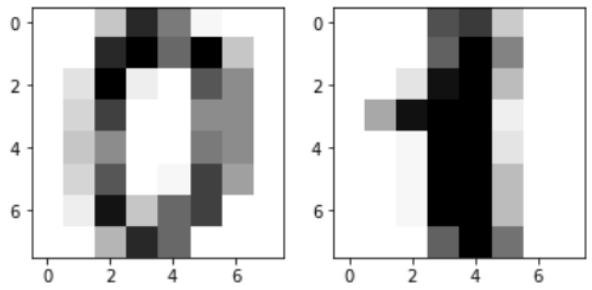
Out[104]:
```
array([0, 1, 2, ..., 8, 9, 8])
```

In [143]:
```
plt.figure()

plt.subplot(1, 2, 1) # o plt.subplot(121)
plt.imshow(digits.data[0].reshape(8, 8), cmap='Greys')
plt.subplot(1, 2, 2) # o plt.subplot(121)
plt.imshow(digits.data[1].reshape(8, 8), cmap='Greys')

print(digits.target[0])
```

```
0
```



In [108]:
```
from sklearn import svm
# clasificatore
clf = svm.SVC(gamma=0.001, C=100.)
```

In [111]:
```
# fit addestra il clasificatore
# addestriamo su tutti i dati tranne l'ultimo
clf.fit(digits.data[:-1], digits.target[:-1])
```

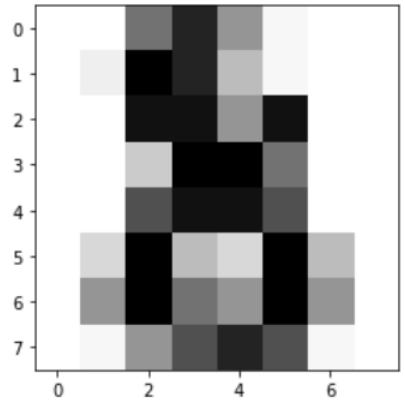
Out[111]:
```
SVC(C=100.0, gamma=0.001)
```

In [160]:
```
clf.predict(digits.data[-1:])
```

Out[160]:
```
array([8])
```

In [120]:
```
plt.imshow(digits.data[-1:].reshape(8, 8), cmap='Greys')
```

Out[120]:
```
<matplotlib.image.AxesImage at 0x7f10f8c82ee0>
```



Unsupervised Learning

Nel problema dell'apprendimento non supervisionato, I dati non sono preclassificati e vogliamo dividerli in "famiglie" in base alle loro features.

Utile per fare anomaly detection (i.e., identificare i comportamenti che si discostano significativamente dalla norma), sistemi di raccomandazione (netflix, amazon) e tanto altro.

Vediamo un esempio con uno degli algoritmi più semplici e popolari, k-means.

Unsupervised Learning - example

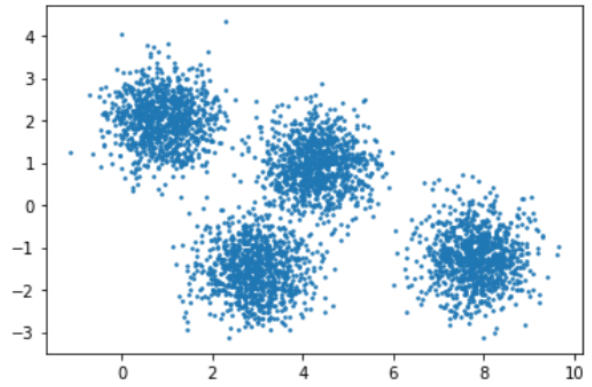
```
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs

# Generate sample data
n_samples = 4000
n_components = 4

X, y_true = make_blobs(
    n_samples=n_samples, centers=n_components, cluster_std=0.60, random_state=0
)
X = X[:, ::-1]

plt.scatter(X[:,0], X[:, 1], marker=".", s=10)
```

```
<matplotlib.collections.PathCollection at 0x7f111c1703a0>
```




```
# Calculate seeds from kmeans++
kmean = KMeans(n_clusters=4, init='k-means++', random_state=0).fit(X)

# Plot init seeds along side sample data
plt.figure(1)
colors = ["#4EACC5", "#FF9C34", "#4E9A06", "m"]

for k, col in enumerate(colors):
    cluster_data = kmean.predict(X) == k
    plt.scatter(X[cluster_data, 0], X[cluster_data, 1], c=col, marker=".", s=10)


plt.scatter(kmean.cluster_centers_[:, 0], kmean.cluster_centers_[:, 1], c="b", s=50)
plt.title("K-Means++ Initialization")
plt.xticks([])
plt.yticks([])
plt.show()
```