Introduction to Python programming

Machine Learning in Python: Unsupervised Learning

- Armando Camerlingo

IR0000032 – ITINERIS, Italian Integrated Environmental Research Infrastructures System
(D.D. n. 130/2022 - CUP B53C22002150006) Funded by EU - Next Generation EU PNRR-Mission 4 “Education and Research” - Component 2: “From research to business” - Investment 3.1: “Fund for the realisation of an integrated system of research and innovation infrastructures”

Finanziato dall'Unione europea NextGenerationEU
Ministero dell'Università e della Ricerca
Italiadomani

Introduzione alla Programmazione in Python

Armando Camerlingo

09/06/2025 - Lezione 3

Pandas Dataframe: principali operazioni

- attributo **`shape`**: dimensioni del dataframe;
- metodo **`describe()`**: statistica descrittiva sui dati;
- metodo **`info()`**: ottieni informazioni sul dataframe;

```
iris.shape
iris.describe()
iris.info()
```

- attributo **`columns`**: attributo columns: nomi delle colonne in un oggetto di tipo index;
- attributo **`index`**: restituisce nomi delle righe in un array in un oggetto di tipo index;

```
iris.columns()
iris.index()
```

Pandas Dataframe: principali operazioni

- metodo `head()`: restituisce le prime n righe (default 5);
- metodo `tail()`: restituisce le ultime n righe (default 5);

```
iris.head()
iris.tail()
```

- `iloc`: seleziona i dati utilizzando gli indici;

```
iris_2 = iris.iloc[:, [1,2,3]]
```

- `loc`: seleziona i dati utilizzando i nomi delle righe e delle colonne;

```
iris_3 = iris.loc[[0,1,100], ['sepal_length', 'petal_length']]
```

- Si possono usare condizioni e operatori booleani per filtrare i dati (& per and, | per or). Usare le parentesi in modo corretto!!;

```
iris[(iris['species']=='setosa') & (iris['petal_length']>1.5)]
```

Seaborn

- Useremo la libreria seaborn, utile non solo per scaricare dataset di esempio ma anche per disegnare grafici

```
import seaborn as sns
import pandas as pd
```

- Si possono settare dei valori di default per i parametri delle figure (tema);

```
sns.set_theme(palette='pastel', rc={'figure.figsize':(15,8)})
```

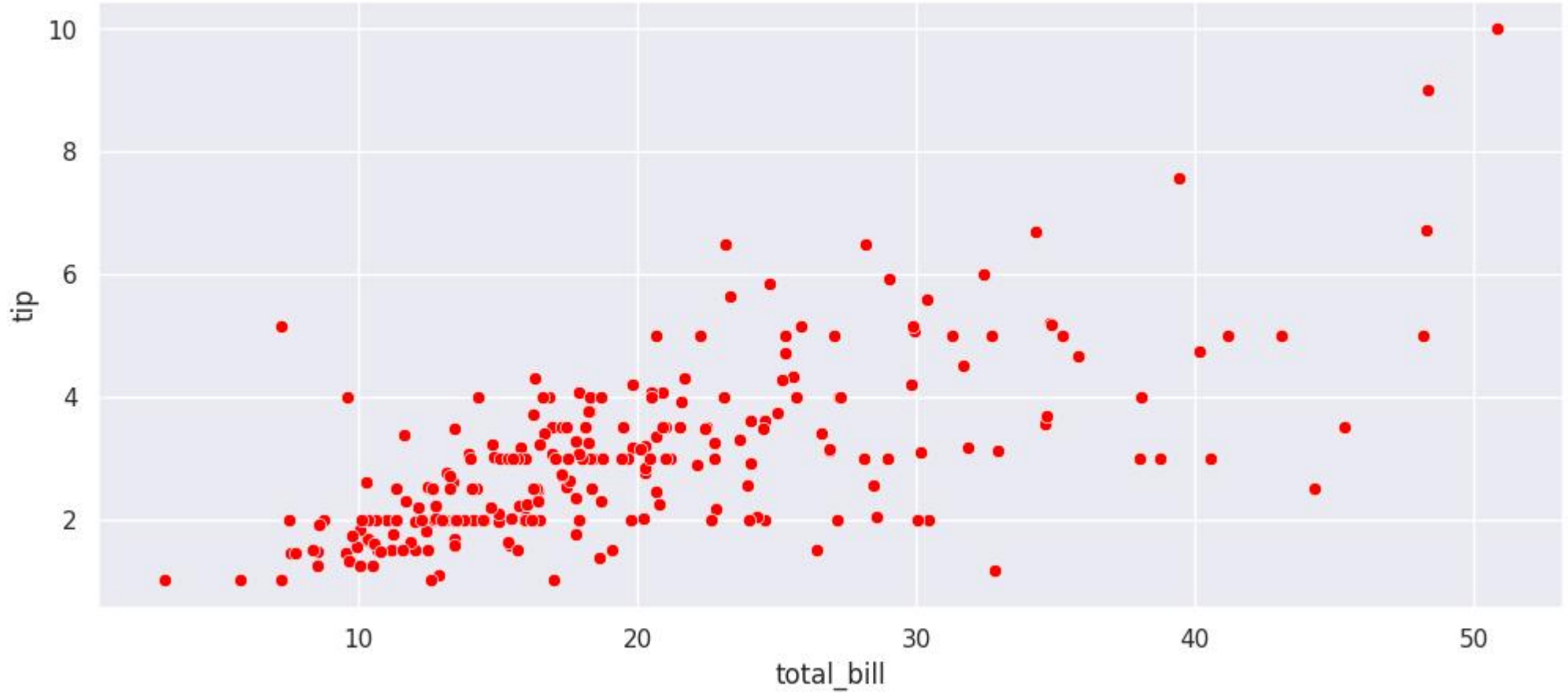
- Carichiamo un dataset di esempio e salviamo in un dataframe

```
mance = sns.load_dataset('tips')
```

Seaborn

- scatterplot()

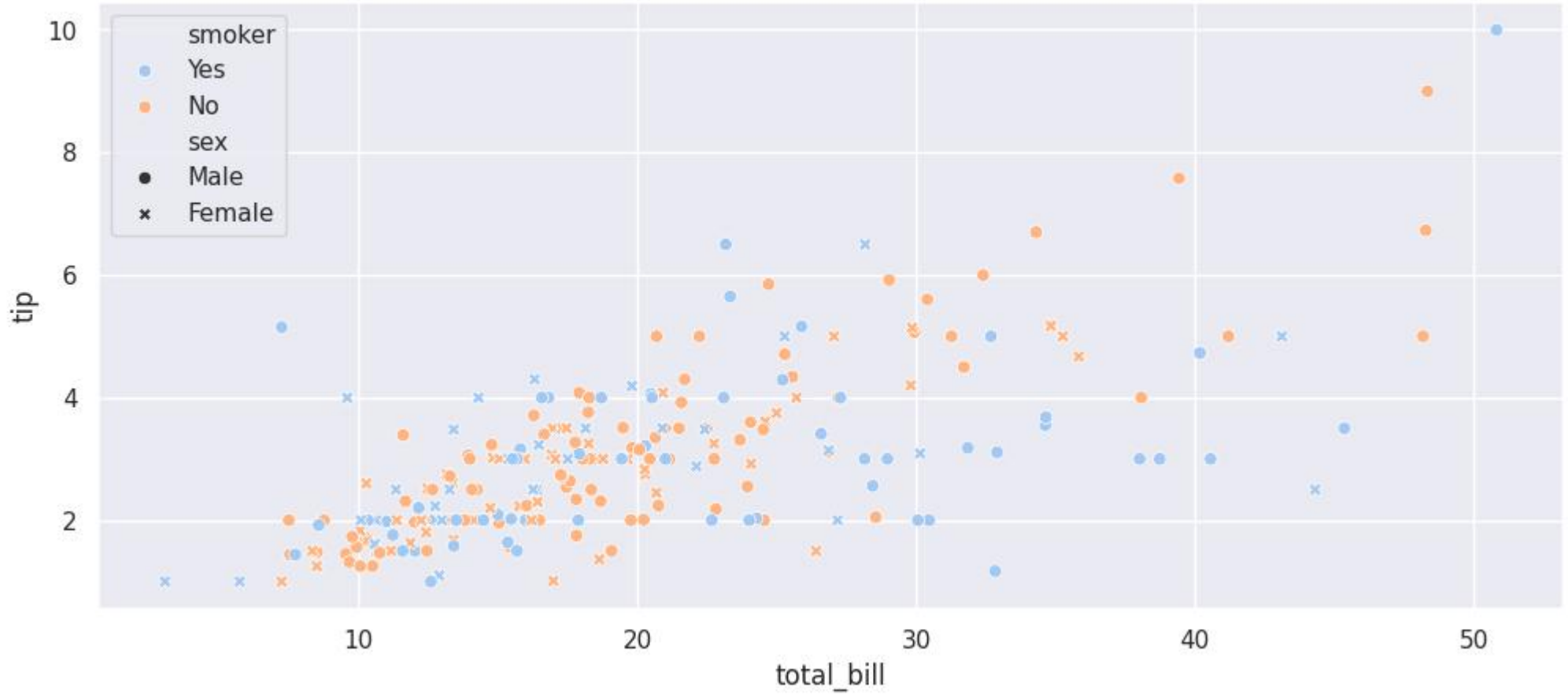
```
p = sns.scatterplot(data=mance, x='total_bill', y='tip', color='red')
```



Seaborn

- scatterplot()

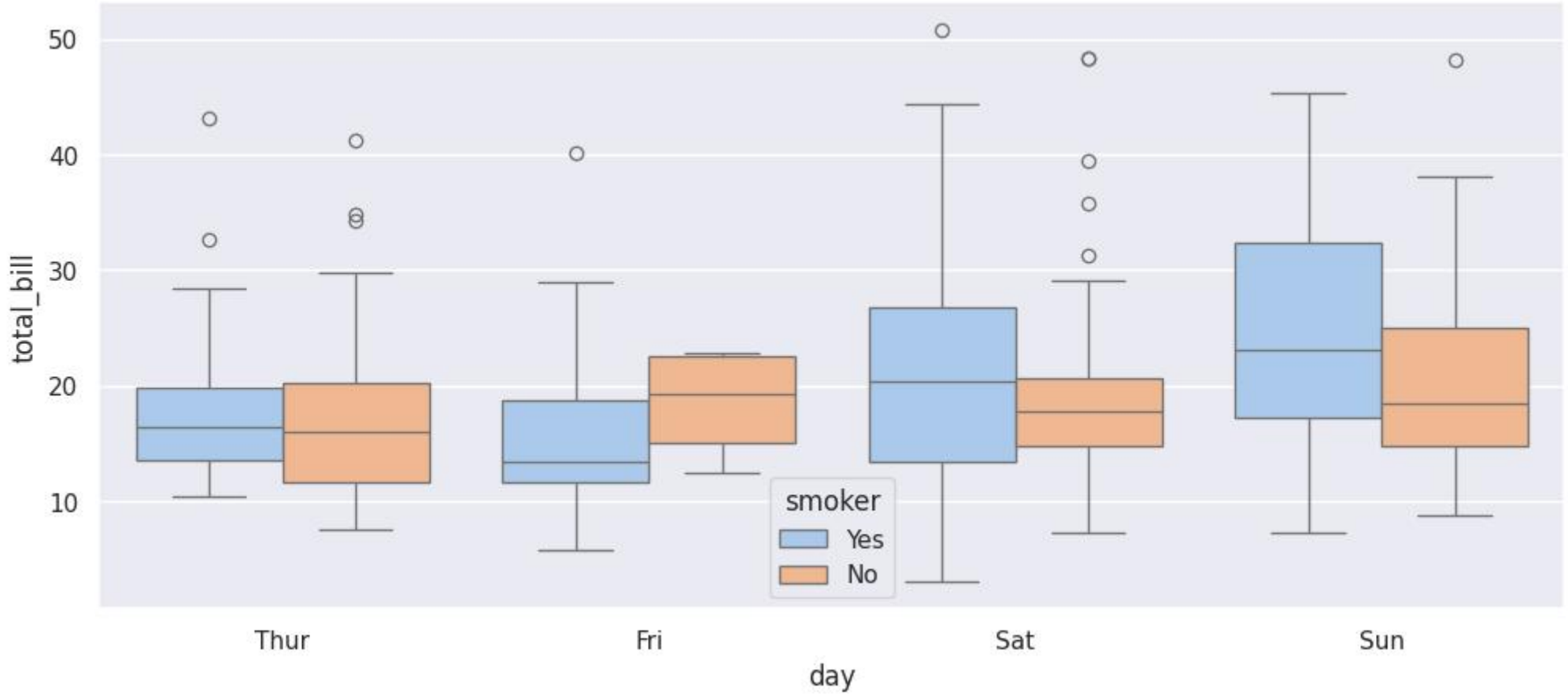
```
p = sns.scatterplot(data=mance, x='total_bill', y='tip', hue='smoker', style='sex')
```



Seaborn

- boxplot()

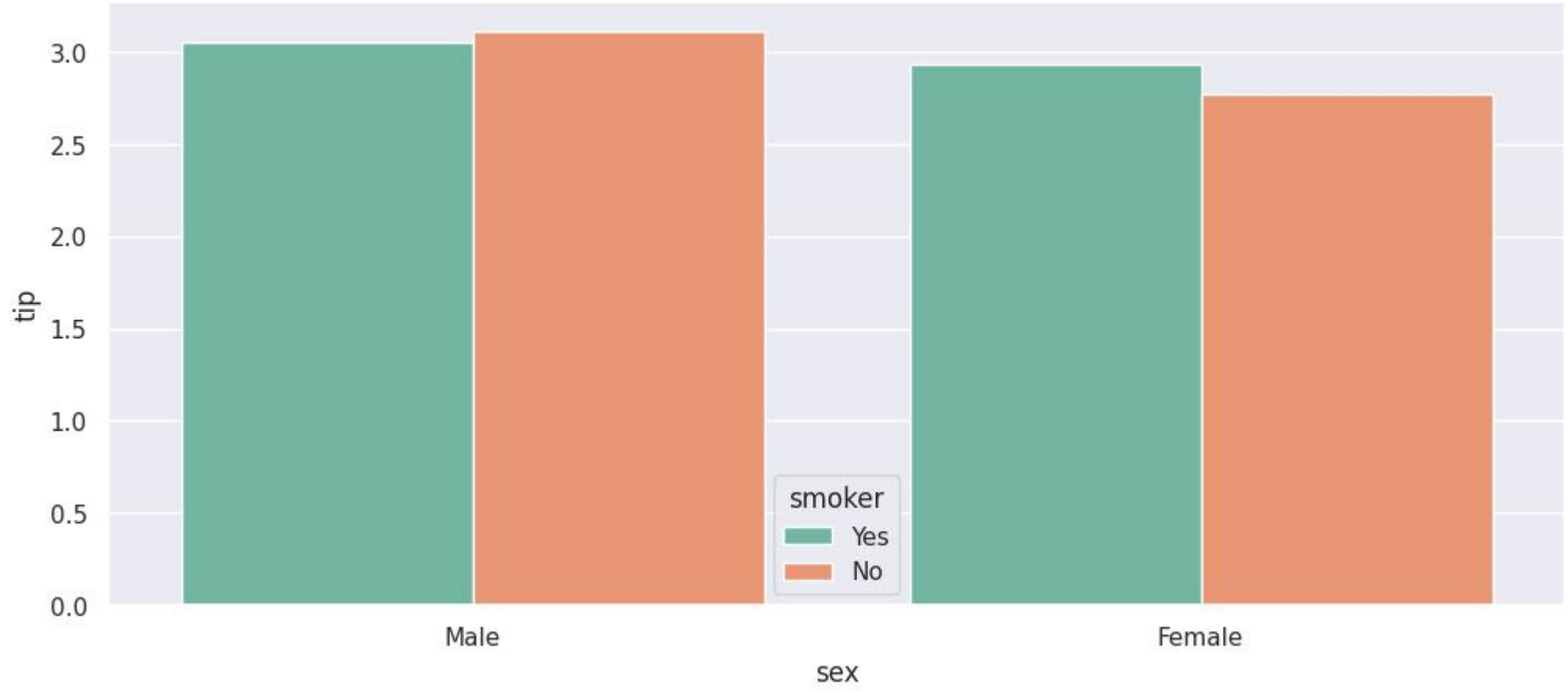
```
p = sns.boxplot(data=mance, x='day', y='total_bill', hue='smoker')
```



Seaborn

- barplot()

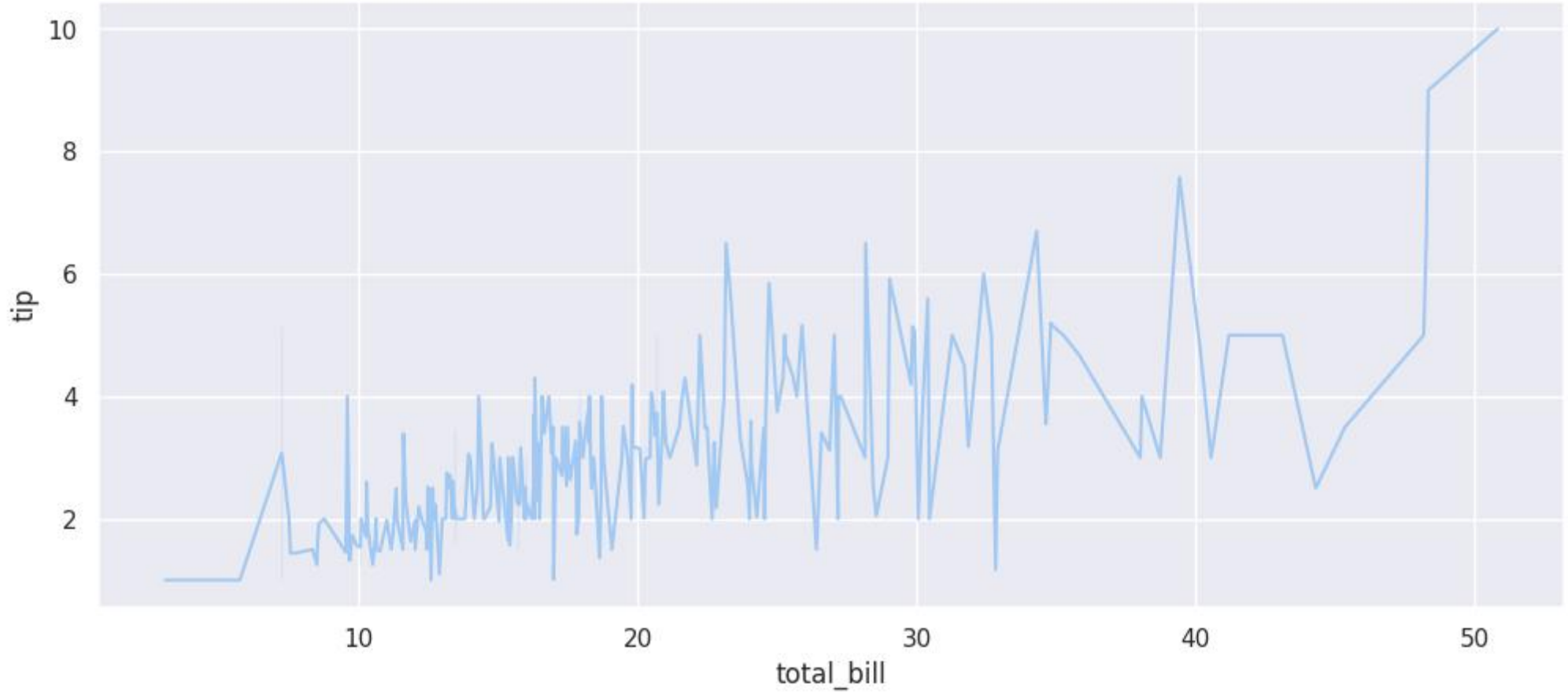
```
p = sns.barplot(data=mance, x='sex', y='tip', hue='smoker',
palette='Set2', errorbar=None)
```



Seaborn

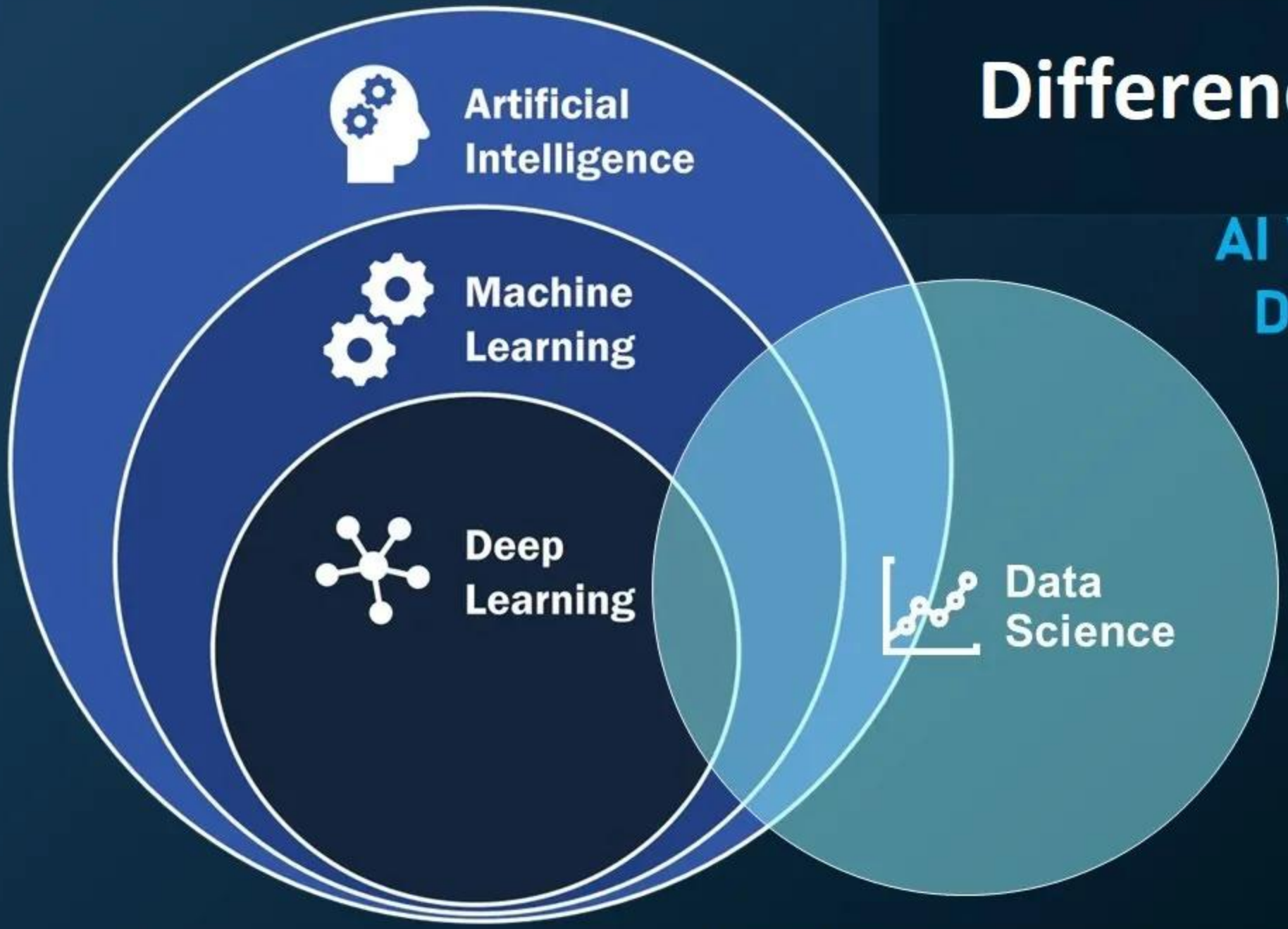
- lineplot()

```
p = sns.lineplot(data=mance, x='total_bill', y='tip')
```



Difference Between

AI VS ML VS DL VS DS



https://medium.com/@shahinaperveen9444/differences-between-ai-ml-dl-ds-cc899d6921bf

Machine Learning

Il machine learning (ML) studia metodi che possiamo applicare per construire applicazioni di IA che possono aiutare ad automatizzare diversi compiti, di solito di regressione o di predizione.

Si possono categorizzare in due classi in base al loro apprendimento:

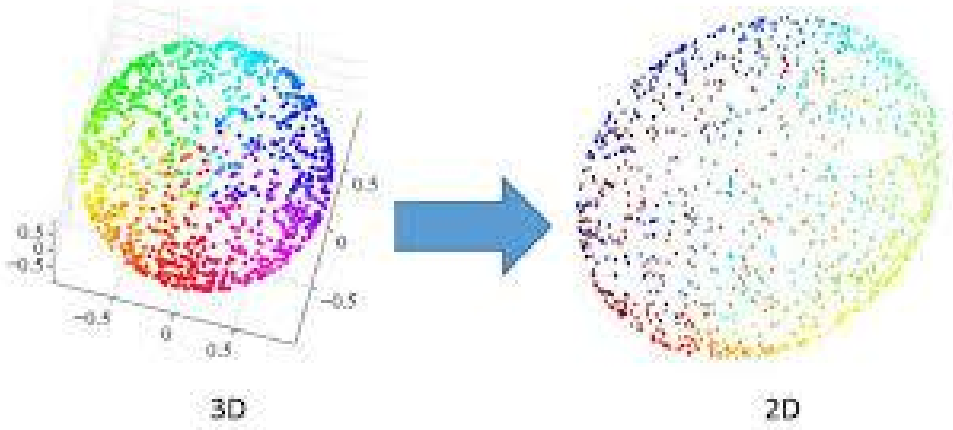
- Supervisionato (supervised): i dati sono forniti con delle “etichette” e l’algoritmo cerca di trovare una relazione tra esse e le variabili di input;

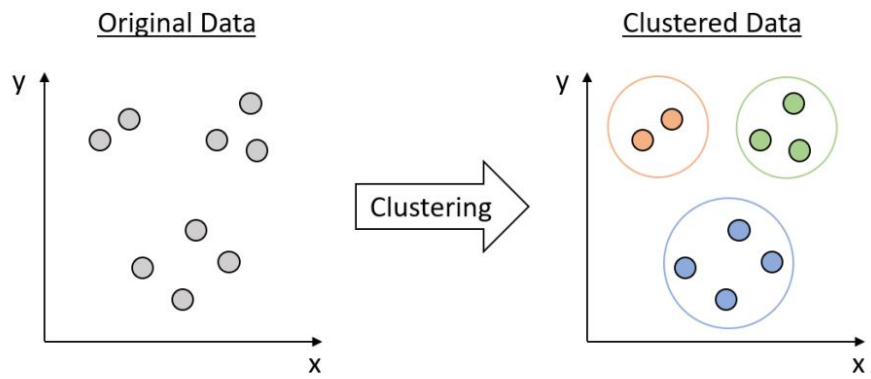
- Non supervisionato (unsupervised): i dati non sono etichettati e l’algoritmo cerca delle proprietà comuni ai campioni non conosciute in precedenza.

Unsupervised Machine Learning

Si lavora con dati non etichettati e di struttura non conosciuta, non sappiamo se ci sono relazioni tra di essi o una parte di essi.
Gli algoritmi di ML non supervisionato processano questi dati per estrarre informazioni da essi.
Ci sono due tecniche principali:

Clustering





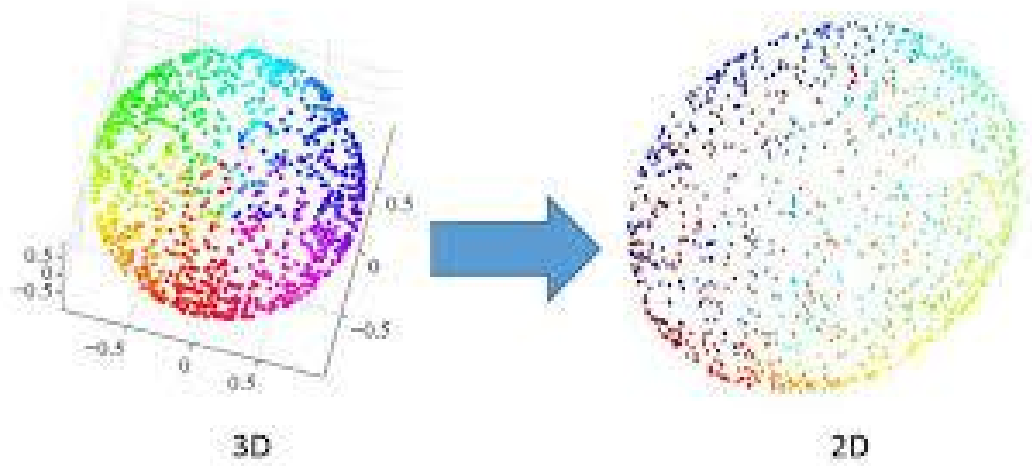
Riduzione di dimensionalità

Riduzione della dimensionalità

Utile quando si lavora con dati ad alta “dimensionalità”, ovvero composto da molte variabili o features. In particolare, viene adoperata durante la fase di pre-processing per:

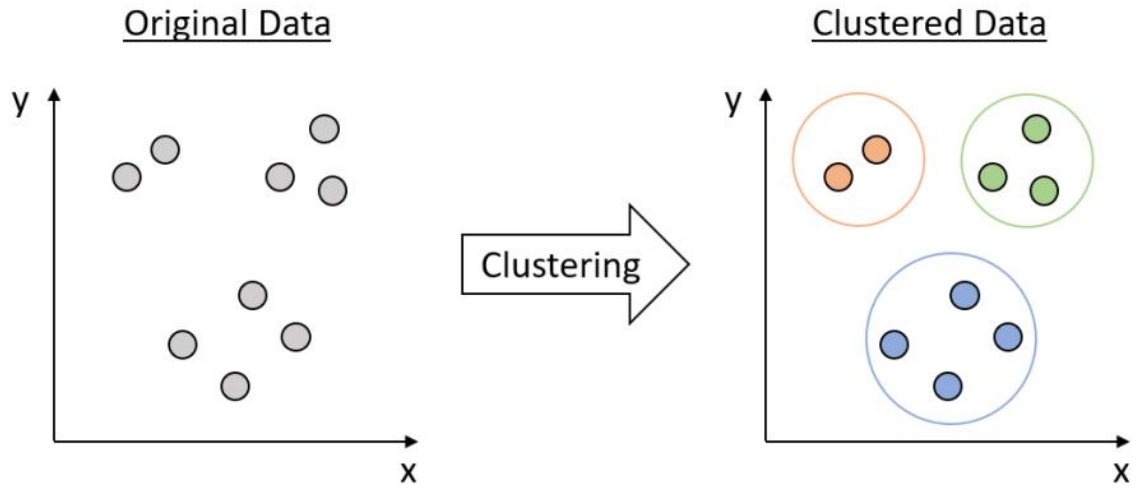
- Rimuovere rumore dai dati, che può influire negativamente sulla qualità delle predizioni;
- Ridurre le dimensioni del dato conservando la maggior parte delle informazioni rilevanti al caso;

Inoltre è utile anche per visualizzare i dati, proiettandoli in uno spazio 3D o 2D



Clustering

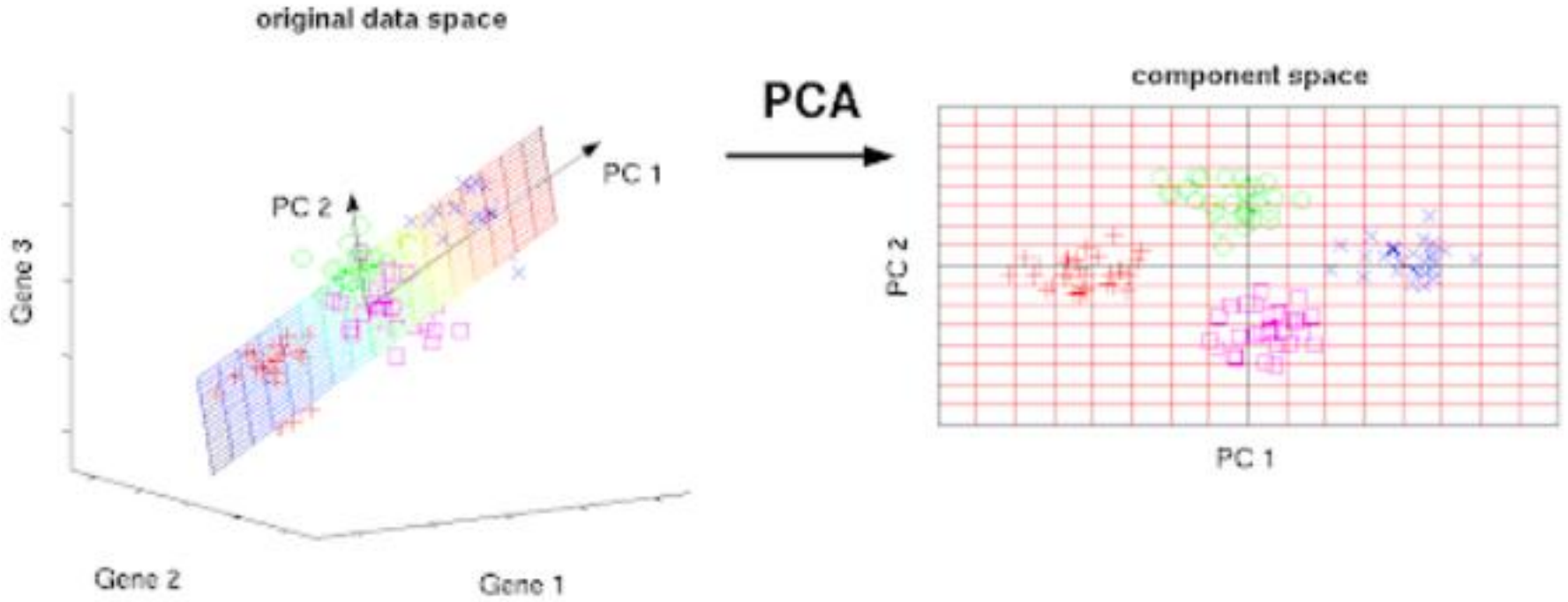
Organizza i dati in sottogruppi in cui gli elementi condividono un certo grado di similarità, però abbastanza dissimili da non appartenere ad un altro sottogruppo.
Utile per trovare relazioni significative tra i dati non conosciute precedentemente durante la raccolta dei campioni.



Principal Component Analysis

Usata per riduzione di features (features extraction) e riduzione di dimensionalità.

La PCA trova la combinazioni di variabili che descrivono meglio la varianza tra i dati e proietta i dati in uno spazio con meno dimensioni. Queste combinazioni sono chiamate componenti principali.



PCA

Ipotizziamo di avere un dataset di *n* campioni con *k* features, quindi lo possiamo scrivere come una matrice *(n,k)*. Si vuole ridurre a *d* features.

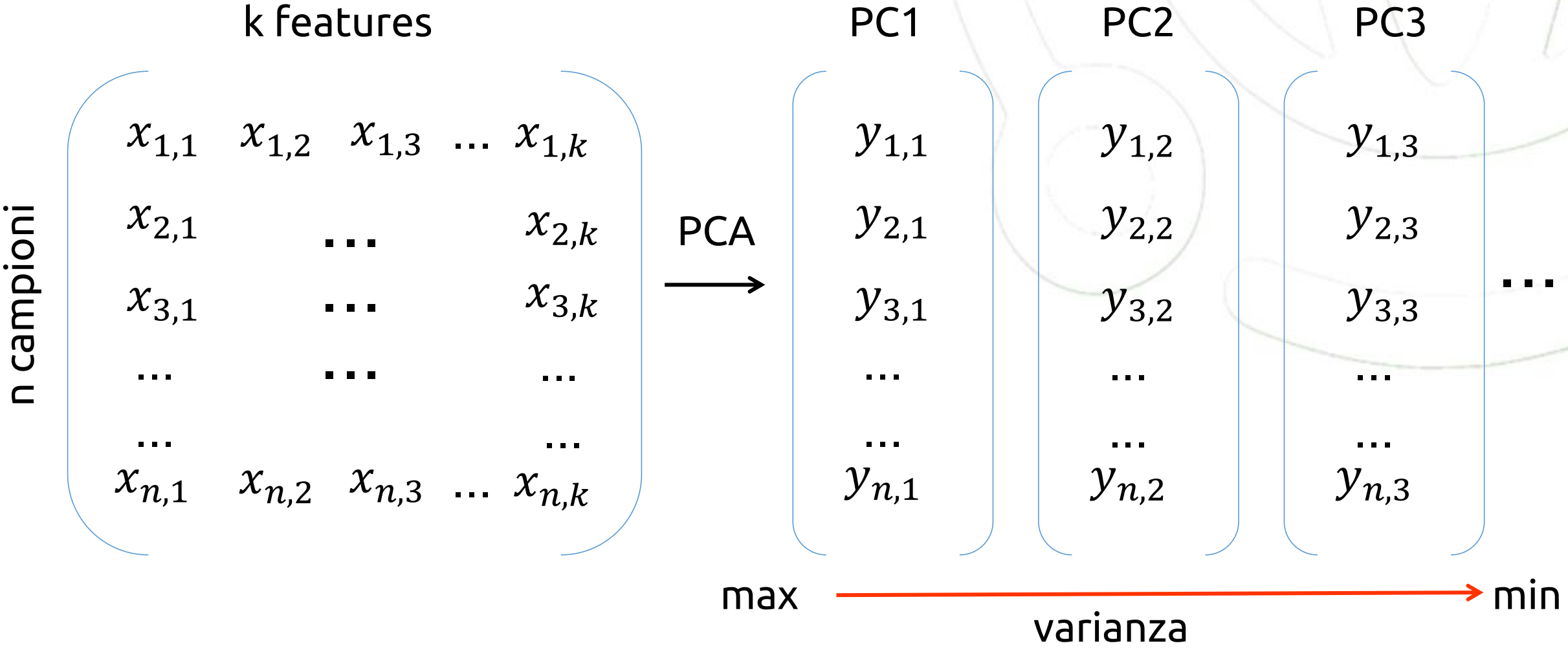
k features

n campioni

$$\begin{pmatrix} x_{1,1} & x_{1,2} & x_{1,3} & \dots & x_{1,k} \\ x_{2,1} & & \dots & & x_{2,k} \\ x_{3,1} & & \dots & & x_{3,k} \\ \dots & & \dots & & \dots \\ \dots & & & & \dots \\ x_{n,1} & x_{n,2} & x_{n,3} & \dots & x_{n,k} \end{pmatrix}$$

PCA

Ipotizziamo di avere un dataset di *n* campioni con *k* features, quindi lo possiamo scrivere come una matrice *(n,k)*. Si vuole ridurre a *d* features.



PCA: note

- Le componenti principali sono calcolate a partire dagli autovettori della matrice di covarianza e i rispettivi autovalori sono un indice della varianza legata alle componenti;

- Si seleziona un numero adeguato delle prime PC che contengono la maggior parte dell'informazione (varianza). Le PC vengono ordinate dall'autovalore più alto al più basso;

- PCA è molto suscettibile a dati misurati con scale diverse, conviene standardizzare le features, usando per esempio un autoscaler;

PCA in Python

Utilizzeremo il pacchetto Scikit-learn (**`sklearn`**), che contiene molte funzioni utili sia in ambito statistico sia in ambito di Machine Learning più specificatamente, oltre ad altri oggetti utili.

```
from sklearn import datasets
dataset = datasets.load_iris()
```

Salviamo le features e le variabili da predirre (target) separatamente

```
X = Dataset['data']
Y = Dataset['target']
```

PCA in Python

Utilizzeremo il pacchetto Scikit-learn (**`sklearn`**), che contiene molte funzioni utili sia in ambito statistico sia in ambito di Machine Learning più specificatamente, oltre ad altri oggetti utili.

```
from sklearn import datasets
dataset = datasets.load_iris()
```

Salviamo le features e le variabili da predirre (target) separatamente

```
X = dataset['data']
Y = dataset['target']
```

Scaliamo i dati

```
import sklearn.preprocessing

X_s = sklearn.preprocessing.scale(X)
```

PCA in Python

Possiamo effettuare la PCA utilizzando la funzione già disponibile in sklearn.decomposition

```
from sklearn.decomposition import PCA
```

Chiamiamo la funzione PCA e scegliamo quante componenti principali scegliere

```
pca = PCA(n_components=4)
```

Applichiamo la PCA sui nostri dati con il metodo fit associato alla funzione PCA

```
pca.fit(X_s)
```

PCA in Python

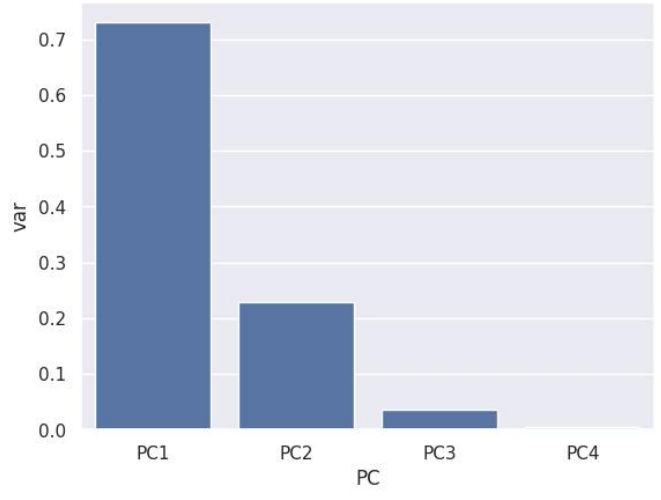
ITINERIS

Possiamo ottenere i risultati della PCA tramite metodi e attributi

```
perc_var_list = list(pca.explained_variance_ratio_)

pc_count = 1
for var in perc_var_list:
  print("Varianza spiegata da PC# ", pc_count, ": ", var)
  pc_count += 1
```

Possiamo visualizzarli con seaborn

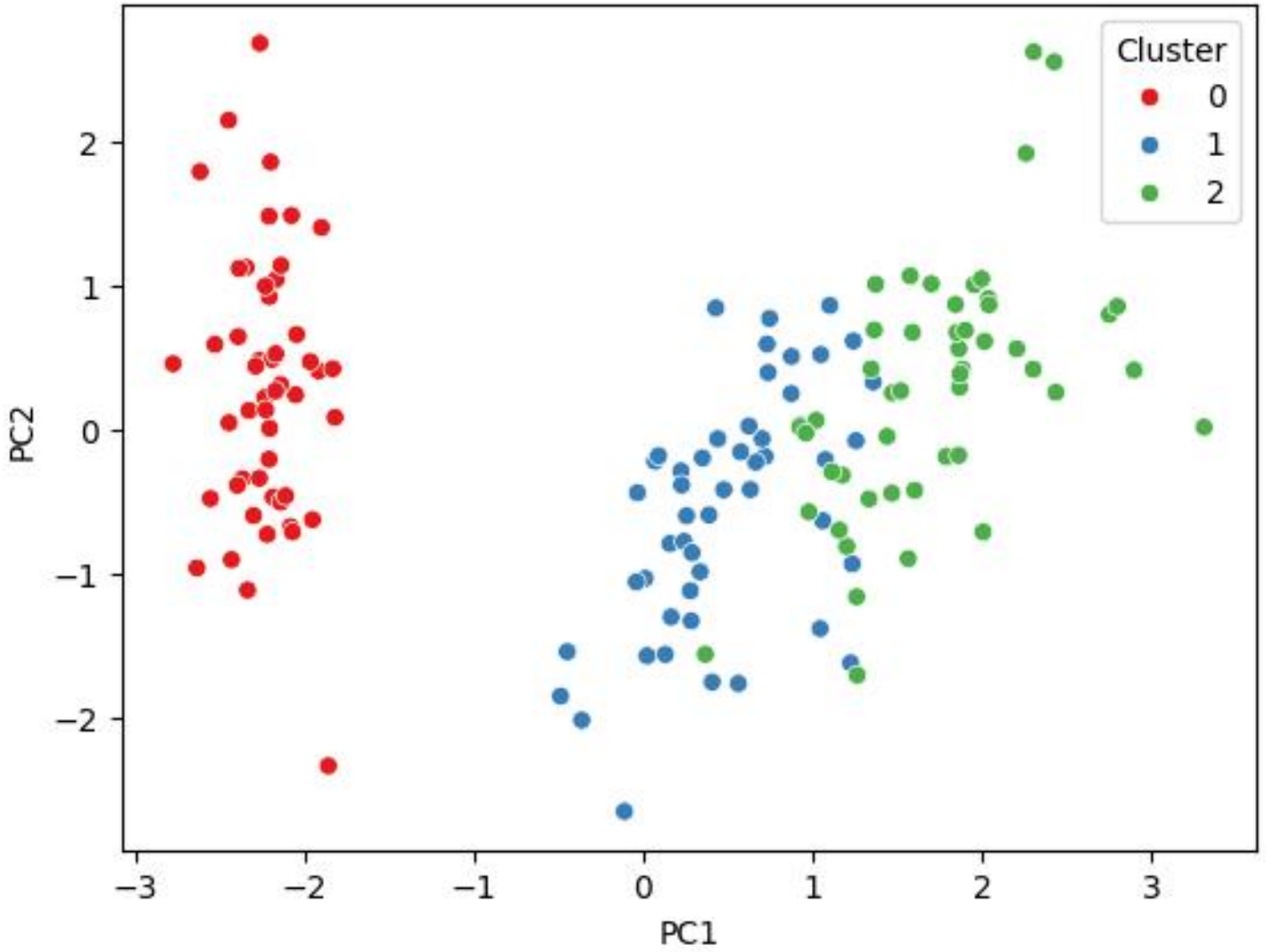
```
import seaborn as sns
sns.set()

df = pd.DataFrame({'var':pca.explained_varia
                   'PC': ['PC1', 'PC2', 'PC3
sns.barplot(data=df, x='PC', y='var');
```



PCA in Python

Visualizziamo le prime due componenti principali in un grafico

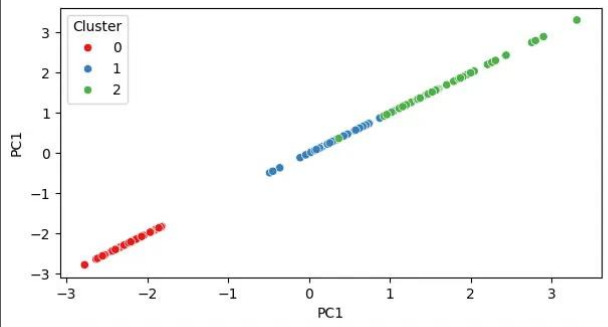
```
comp_princ = pca.fit_transform(X_s)
comp_princ_df = pd.DataFrame(data=comp_princ,
columns=['PC1', 'PC2', 'PC3', 'PC4'])
comp_princ_df['Cluster'] = Y
sns.scatterplot(data = comp_princ_df, x = 'PC1', y = 'PC2',
                hue = 'Cluster', palette='Set1');
```

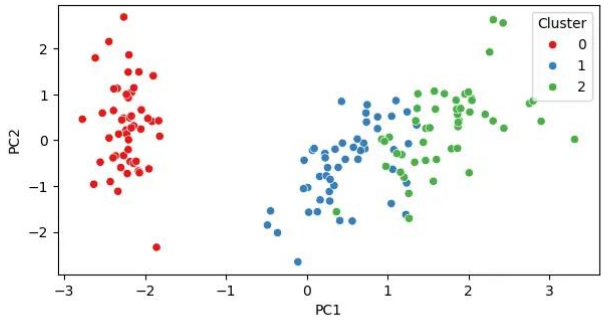

Cluster
0
1
2
PC2
PC1
2
1
0
−1
−2
−3
−2
−1
0
1
2
3

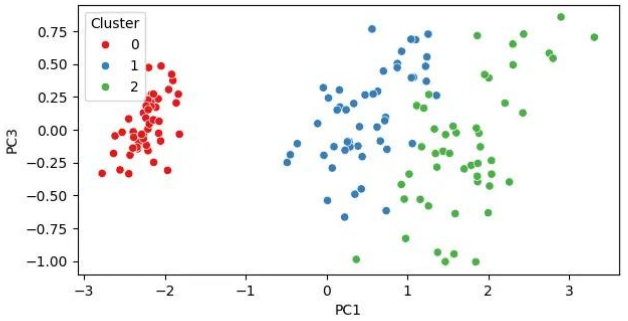
PCA in Python

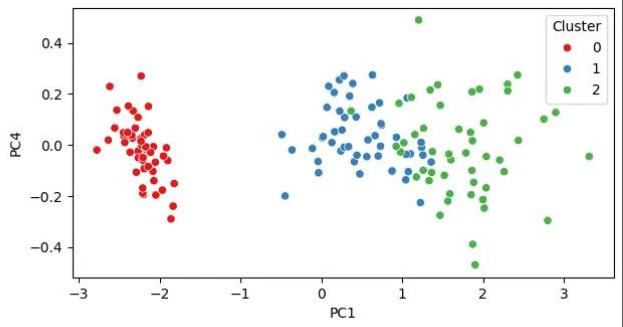
Visualizziamo tutte le coppie di PC possibili:

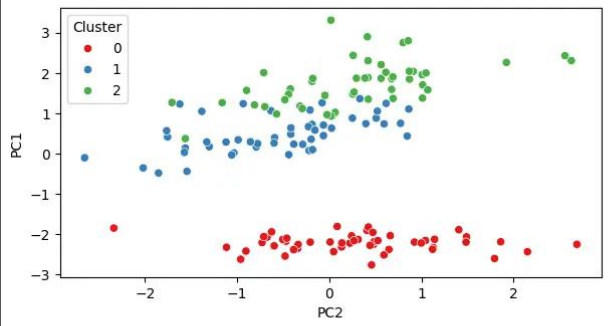
```
import matplotlib.pyplot as plt
num_comp = 4
fig, axes = plt.subplots(num_comp, num_comp,
figsize=(32,16))

for i in range(num_comp):
  for j in range(num_comp):

    pc_nome1 = "PC" + str(i+1)
    pc_nome2 = "PC" + str(j+1)

    sns.scatterplot(data=comp_princ_df, x=pc_nome1,
y=pc_nome2, hue='Cluster', palette='Set1', ax=axes[i][j])
```

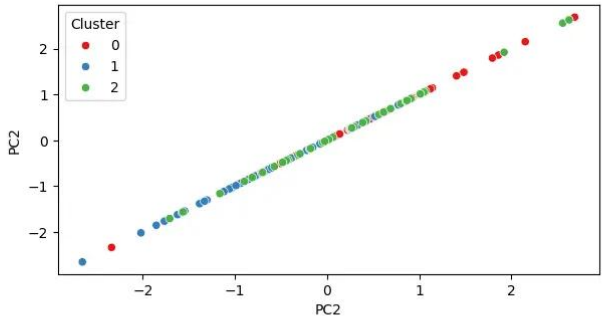

Cluster
0
1
2
PC1
PC1

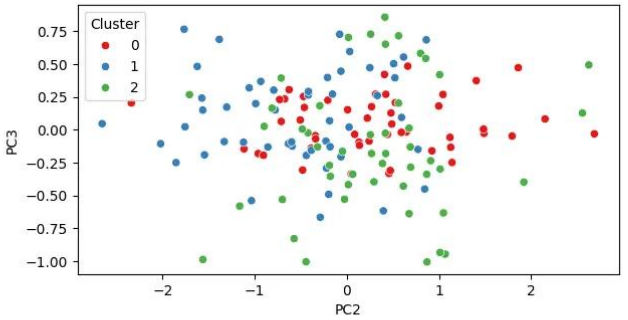

Cluster
0
1
2
PC2
PC1

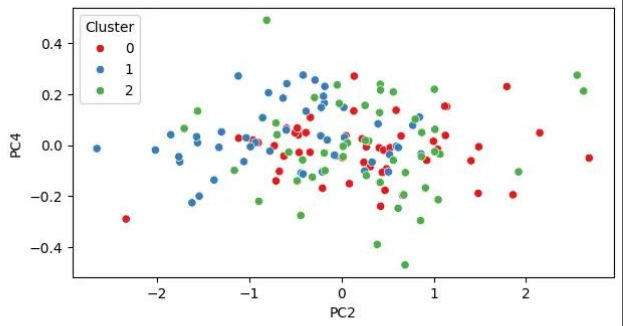

Cluster
0
1
2
PC3
PC1

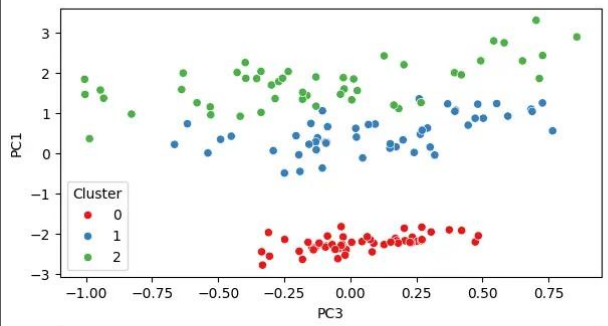

Cluster
0
1
2
PC4
PC1

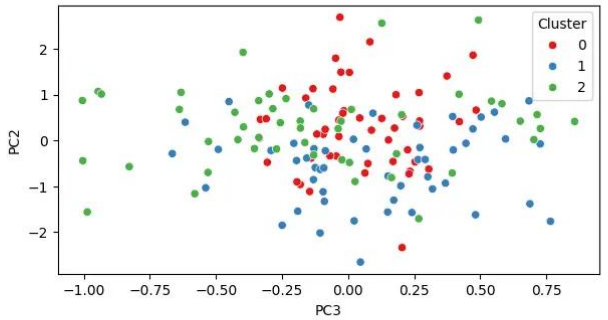

Cluster
0
1
2
PC1
PC2

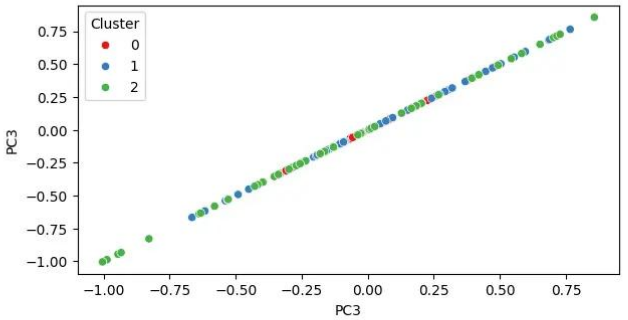

Cluster
0
1
2
PC2
PC2

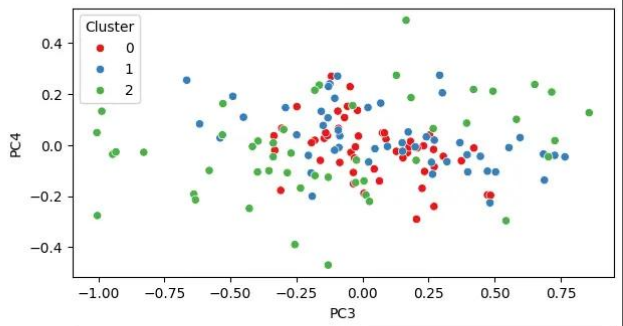

Cluster
0
1
2
PC3
PC2

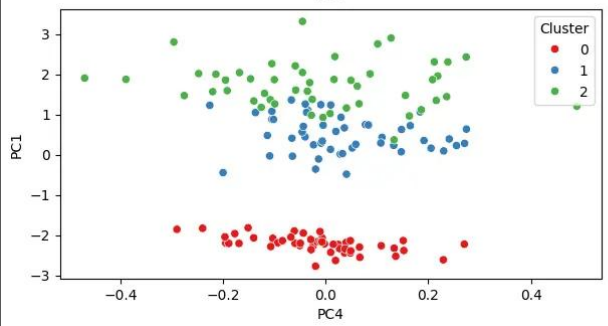

Cluster
0
1
2
PC4
PC2

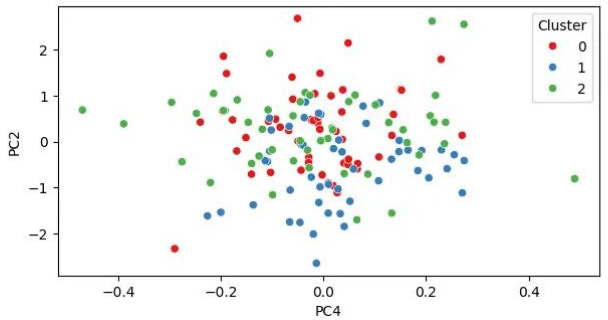

Cluster
0
1
2
PC1
PC3

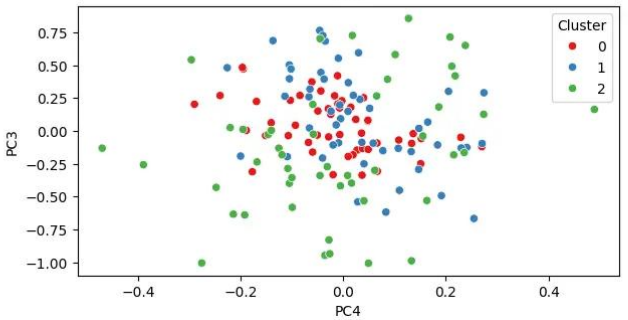

Cluster
0
1
2
PC2
PC3

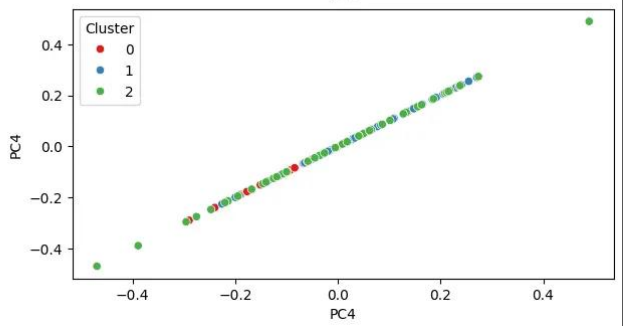

Cluster
0
1
2
PC3
PC3


Cluster
0
1
2
PC4
PC3


Cluster
0
1
2
PC1
PC4


Cluster
0
1
2
PC2
PC4


Cluster
0
1
2
PC3
PC4


Cluster
0
1
2
PC4
PC4

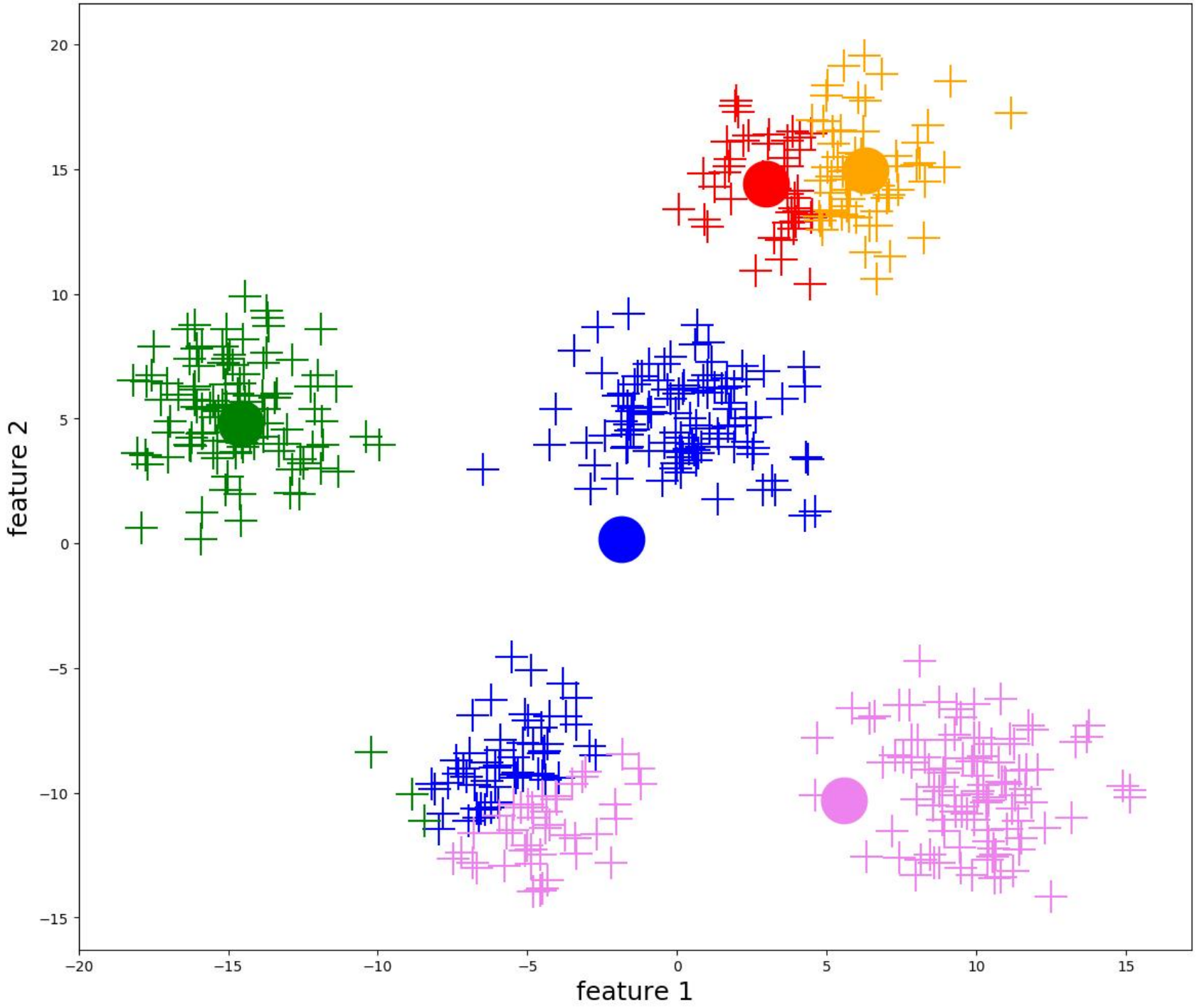
K-means Clustering

ITINERIS

- Uno degli algoritmi di clustering più usati, basato sull'uso di **prototipi**;
- Ogni cluster è rappresentato da un prototipo che può essere
  - un **centroide** per variabili numeriche: p.e. media per quelle continue, mediana per le intere;
  - un **medioide** per variabili categoriche: il punto più rappresentativo o che minimizza una "distanza" da tutti gli altri punti che appartengono al cluster;
- Svantaggio: bisogna pre-selezionare il numero di cluster da cercare. Influenza molto le performance;

K-means Clustering: passi

1. Si scelgono k campioni casuali come centroidi iniziali per k cluster;
2. SI assegna ogni campione del dataset al centroide più vicino secondo una distanza stabilita;
3. Individuare i nuovi centroidi a partire dalla nuova clusterizzazione;
4. Ripetere gli step 2 e 3 fino a che non ci siano più nuovi assegnazioni ai cluster o raggiunta una determinata condizione;


feature 2
feature 1
20
15
10
5
0
−5
−10
−15
−20
−15
−10
−5
0
5
10
15

- Come misuriamo la similarità tra i punti?

- Come valutiamo la qualità del clustering?

- Come misuriamo la similarità tra i punti?

Dobbiamo scegliere una distanza adatta per il caso in esame. Molto spesso si usa la classica distanza euclidea che per due punti x e y in uno spazio a m dimensioni è data da:

$$d(x,y)^2 = \sum_{j=1}^{m} (x_j - y_j)^2$$

- Come valutiamo la qualità del clustering?

- Come misuriamo la similarità tra i punti?

Dobbiamo scegliere una distanza adatta per il caso in esame. Molto spesso si usa la classica distanza euclidea che per due punti x e y in uno spazio a m dimensioni è data da:

$$d(x,y)^2 = \sum_{j=1}^{m} (x_j - y_j)^2$$

- Come valutiamo la qualità del clustering?

Dipende dalla distanza scelta. Nel caso della distanza euclidea, si utilizza la somma degli errori quadratici (chiamata anche cluster inertia) :

n campioni e k cluster
$\mu^{(j)}$ centroidi
$w^{(i,j)}$ 1 se campione i appartiene a cluster j, 0 altrimenti

$$SSE = \sum_{i=1}^{n} \sum_{j=1}^{k} w^{(i,j)} \left\| x^{(i)} - \mu^{(j)} \right\|_2^2$$

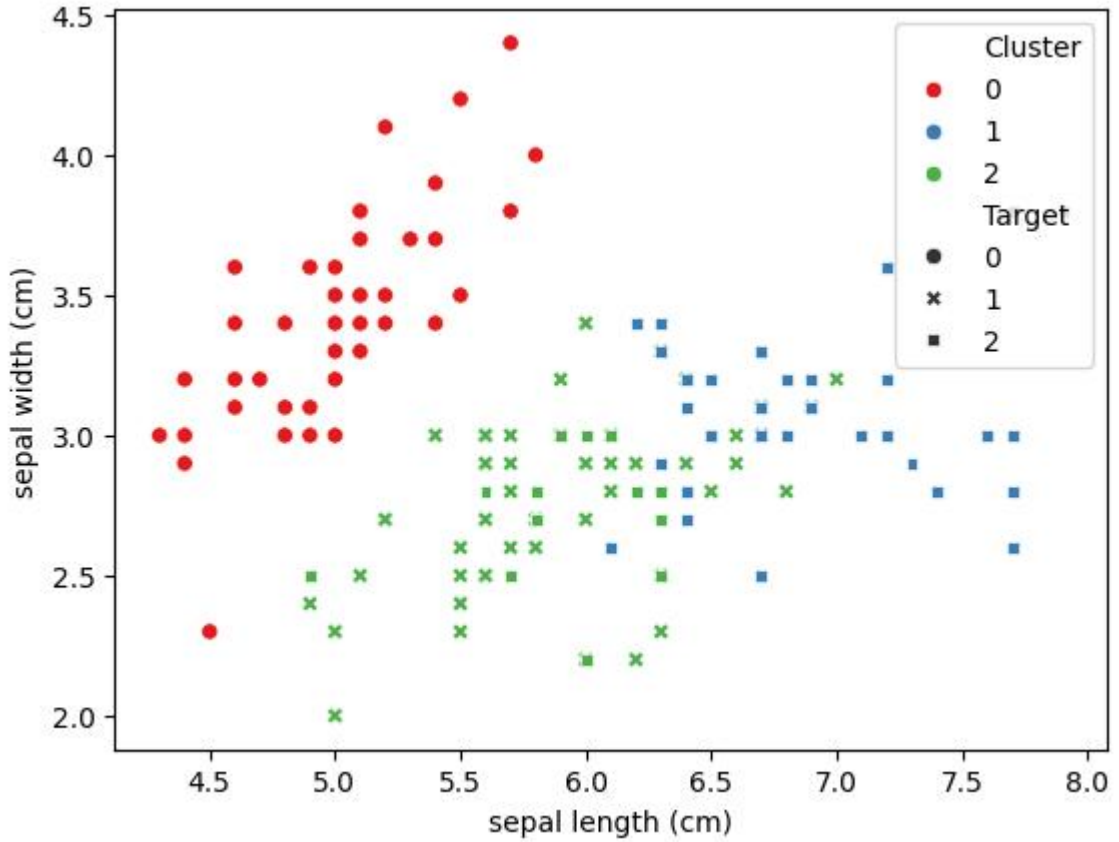
K-means Clustering in Python

```
data = datasets.load_iris()
X = data['data']
Y = data['target']
nomi_var_input = data['feature_names']
nomi_target = data['target_names']
```

```
km = KMeans(n_clusters = 3, #numero di clusters
            init='random', #metodo di selezione centroidi
            n_init=10, #numero di volte che viene eseguito
l'algoritmo
            max_iter=300, #numero massimo di iterazioni,
            tol=1e-4, #se l'errore raggiunge questo valore si ferma
            random_state=0 #seed per la casualità
            )

clusters = km.fit_predict(X)
```

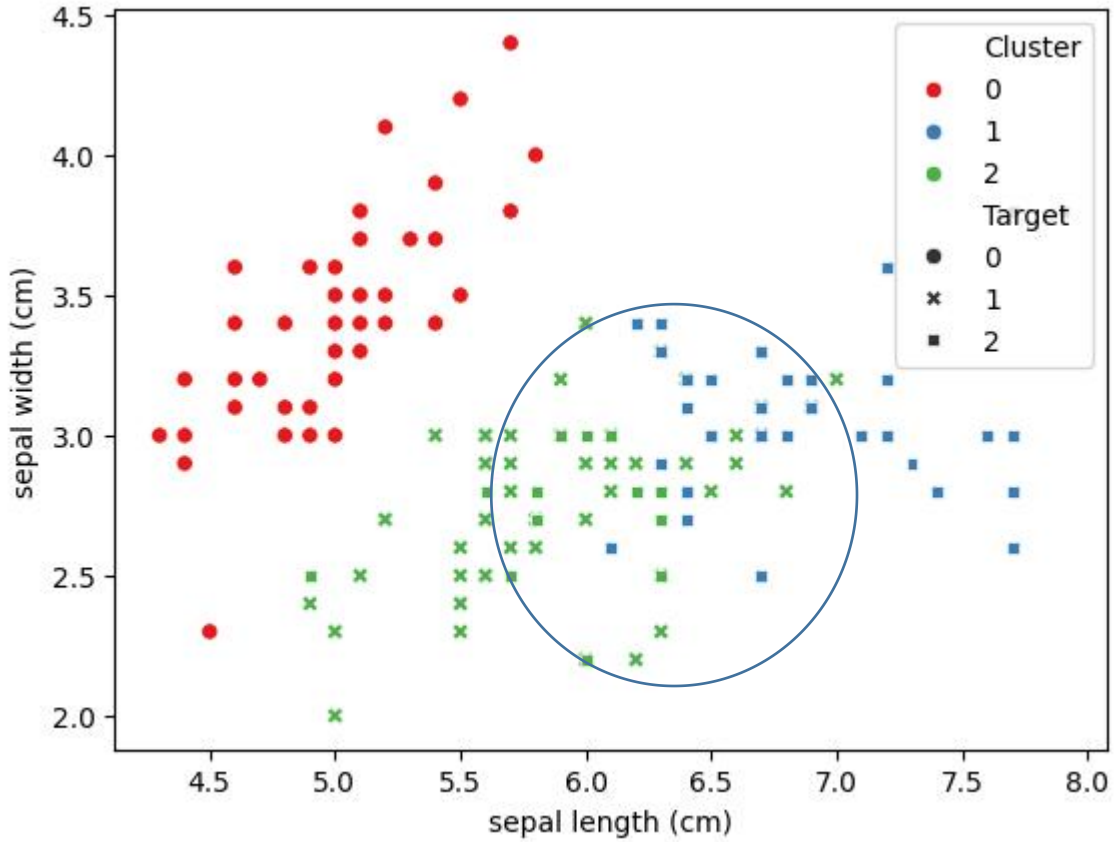
K-means Clustering in Python

```
data_df = pd.DataFrame(data=X, columns=nomi_var_input)
data_df['Cluster'] = clusters
data_df['Target'] = Y

sns.scatterplot(data=data_df, x='sepal length (cm)', y='sepal width
(cm)', hue='Cluster', palette='Set1', style='Target')
```



K-means Clustering in Python

```
data_df = pd.DataFrame(data=X, columns=nomi_var_input)
data_df['Cluster'] = clusters
data_df['Target'] = Y

sns.scatterplot(data=data_df, x='sepal length (cm)', y='sepal width
(cm)', hue='Cluster', palette='Set1', style='Target')
```



K-means Clustering in Python: k-means++

- Abbiamo ancora confusione tra due cluster, scegliere casualmente i centroidi può portare a risultati non corretti;
- Usiamo l'algoritmo **k-means++**, versione migliorata del k-means classico;
- Centroidi non scelti casualmente ma secondo un approccio matematico che li pone distanti dagli altri già scelti.
  1. Si sceglie il primo centroide tra i punti del dataset;
  2. Si calcolano le distanze dei punti dal centroide;
  3. Viene scelto il punto avente la distanza massima come secondo centroide;
  4. Si ripetono gli step 2 e 3 fino ad arrivare a k centroidi;
  5. Si applica l'algoritmo k-means;

K-means Clustering in Python: k-means++

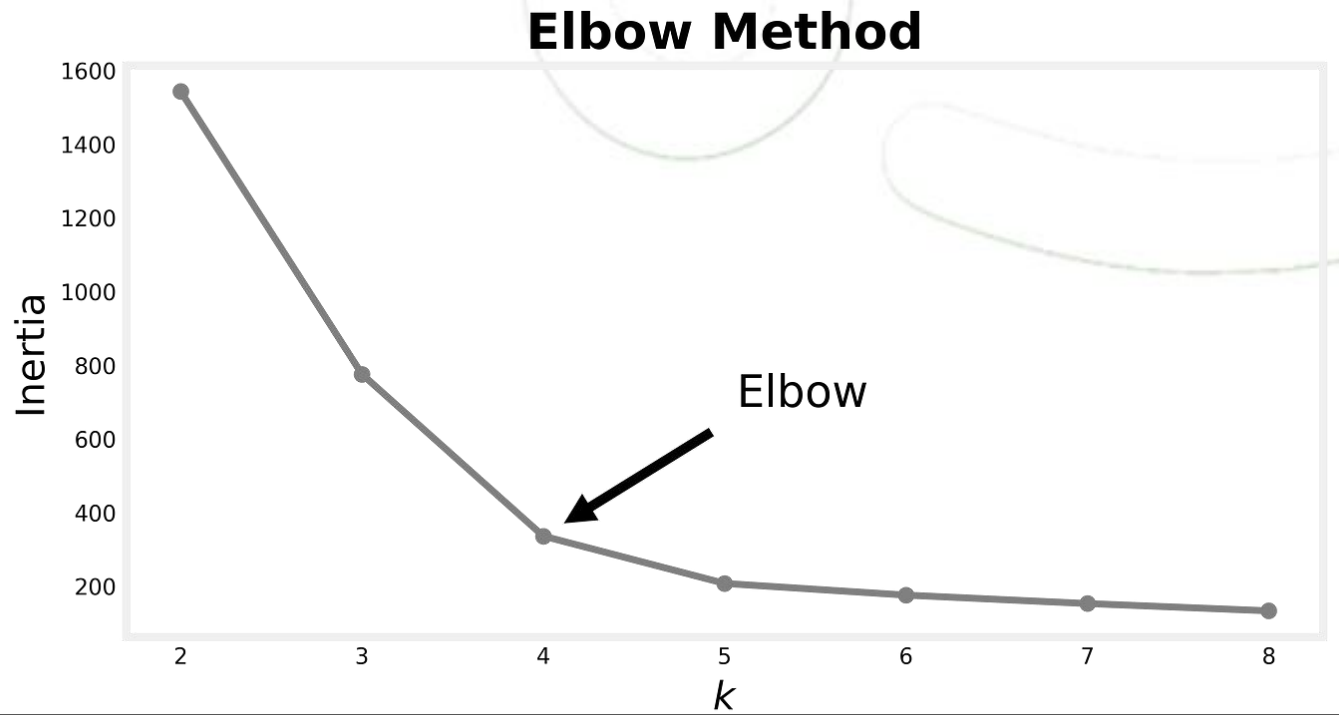
ITINERIS

```
km_2  = KMeans(n_clusters = 3,
               init='k-means++', #metodo di
selezione centroidi
               n_init=10,
               max_iter=300,
               tol=1e-4,
               random_state=0
               )

clusters_2 = km.fit_predict(X)
```

K-means Clustering in Python

Come scegliere k?

Possiamo usare l'**elbow method**, un metodo grafico basato sul SSE. Si visualizza il suo andamento rispetto a k e generalmente il valore di k in cui si nota una curvatura maggiore è ottimale per il k-means (il "gomito" della curva).



K-means Clustering in Python

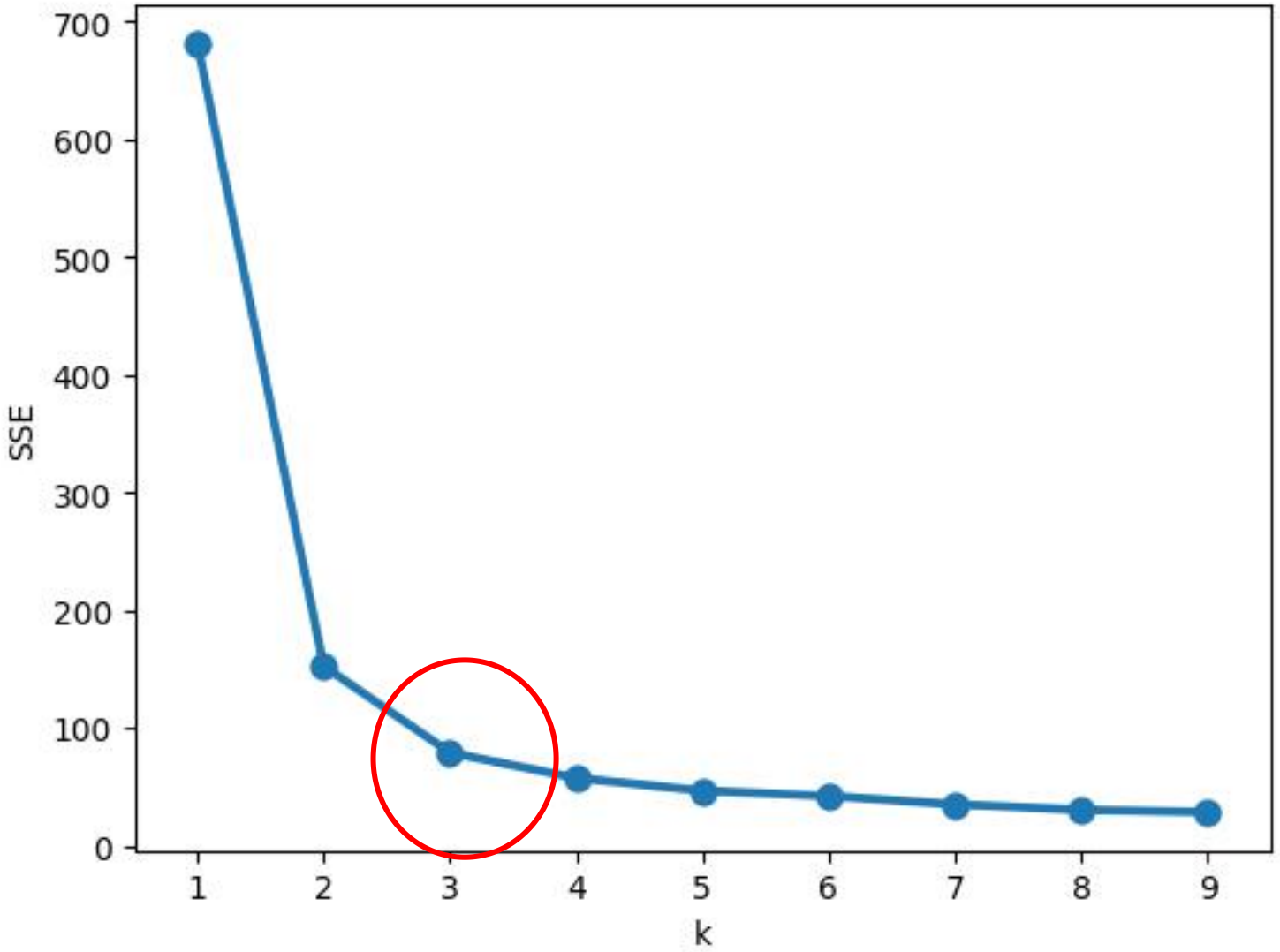
ITINERIS

```
valori_k = range(1,10)

distorsione = []

for k in valori_k:
  km = KMeans(n_clusters = k,
              init='random',
              n_init=10,
              max_iter=300,
              tol=1e-4,
              random_state=0
              )
  km.fit_predict(X)
  distorsione.append(km.inertia_)

risultati = pd.DataFrame({'k':valori_k, 'SSE':distorsione})
```

K-means Clustering in Python

ITINERIS

```
sns.pointplot(data=risultati, x='k', y='SSE')
```

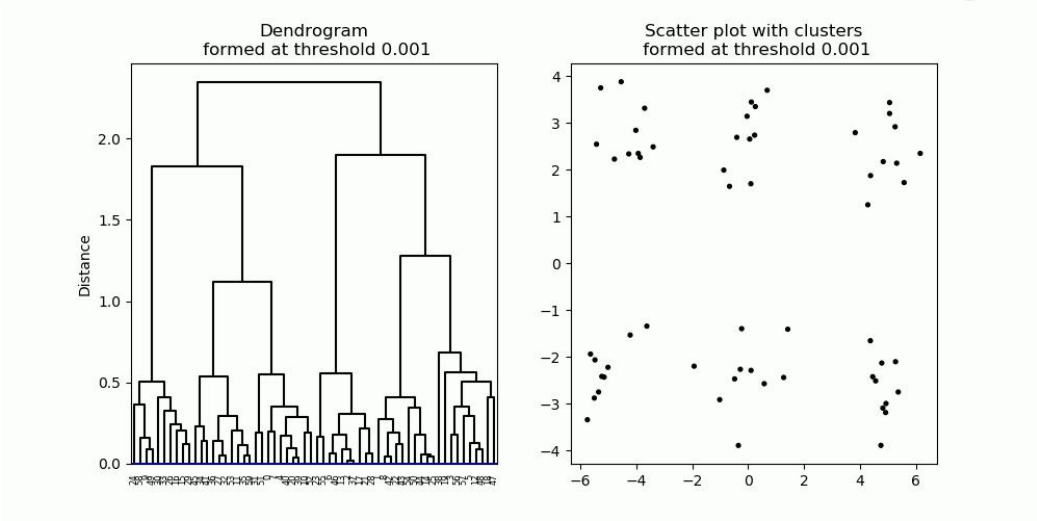


Clustering gerarchico

Ogni punto è trattato inizialmente come un cluster individuale e la coppia di cluster più “vicini” viene unita ad ogni step, fino ad arrivare ad un unico cluster.

1. All’inizio, sono presenti k cluster, dove k sono il numero di punti nel dataset;
2. Si forma un nuovo cluster unendo la coppia di cluster più vicini secondo una distanza definita, ottenendo così k-1 cluster;
3. Si ripete lo step 2 fino ad ottenere un singolo cluster;

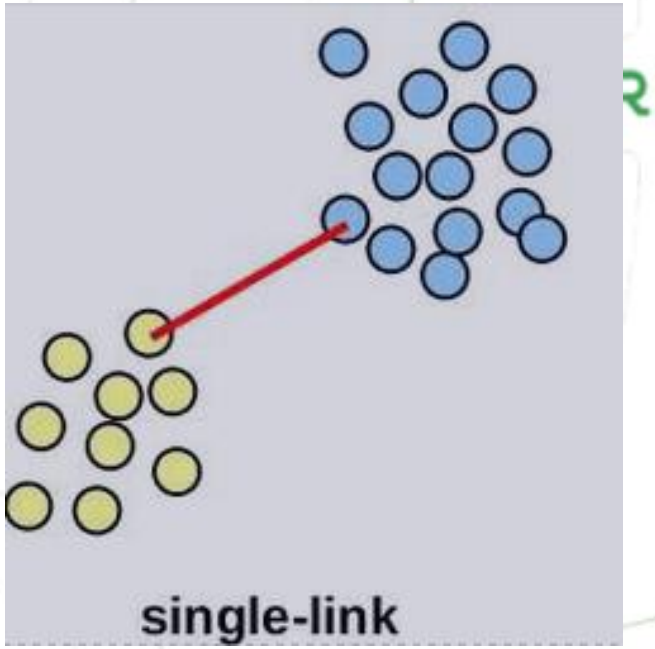
Questa procedura può essere schematizzata utilizzando un dendrogramma, rendendo anche più semplice decidere in quanti cluster vogliamo dividere il dataset. Quindi a differenza di k-means, il numero di cluster **NON** deve essere stabilito in precedenza.

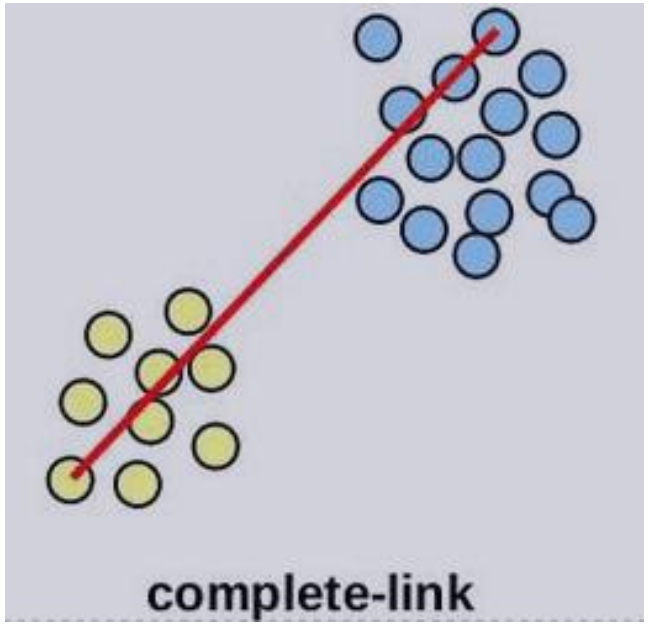


Clustering gerarchico

Due principali strategie per scegliere quali cluster unire:

- Single linking: vengono uniti i cluster aventi gli elementi più simili e più vicini tra loro, ovvero se ci sono un elemento nel cluster A e un elemento nel cluster B che distano meno di ogni altro punto nei restanti cluster, A e B sono uniti;





- Complete linking: vengono uniti i cluster che hanno gli elementi più dissimili, ma più vicini, ovvero se l'elemento del cluster A più distante da un elemento del cluster B ha una distanza maggiore rispetto agli elementi di tutti gli altri gruppi, A e B vengono uniti;

Clustering gerarchico in python

ITINERIS

```
from sklearn import datasets
import pandas as pd
from scipy.spatial.distance import pdist, squareform
import matplotlib.pyplot as plt
```

```
iris = datasets.load_iris()
data = pd.DataFrame(iris.data)
data.columns = iris.feature_names
data['Tipo'] = iris.target
X = data[['sepal length (cm)','sepal width (cm)', 'petal length
(cm)', 'petal width (cm)']]
```

```
distanze = pdist(X, metric='euclidean') #scegliamo la metrica
euclidea come distanza
print(distanze.shape)
print("\n Distanze: ")
print(squareform(distanze))
print("\n")
```

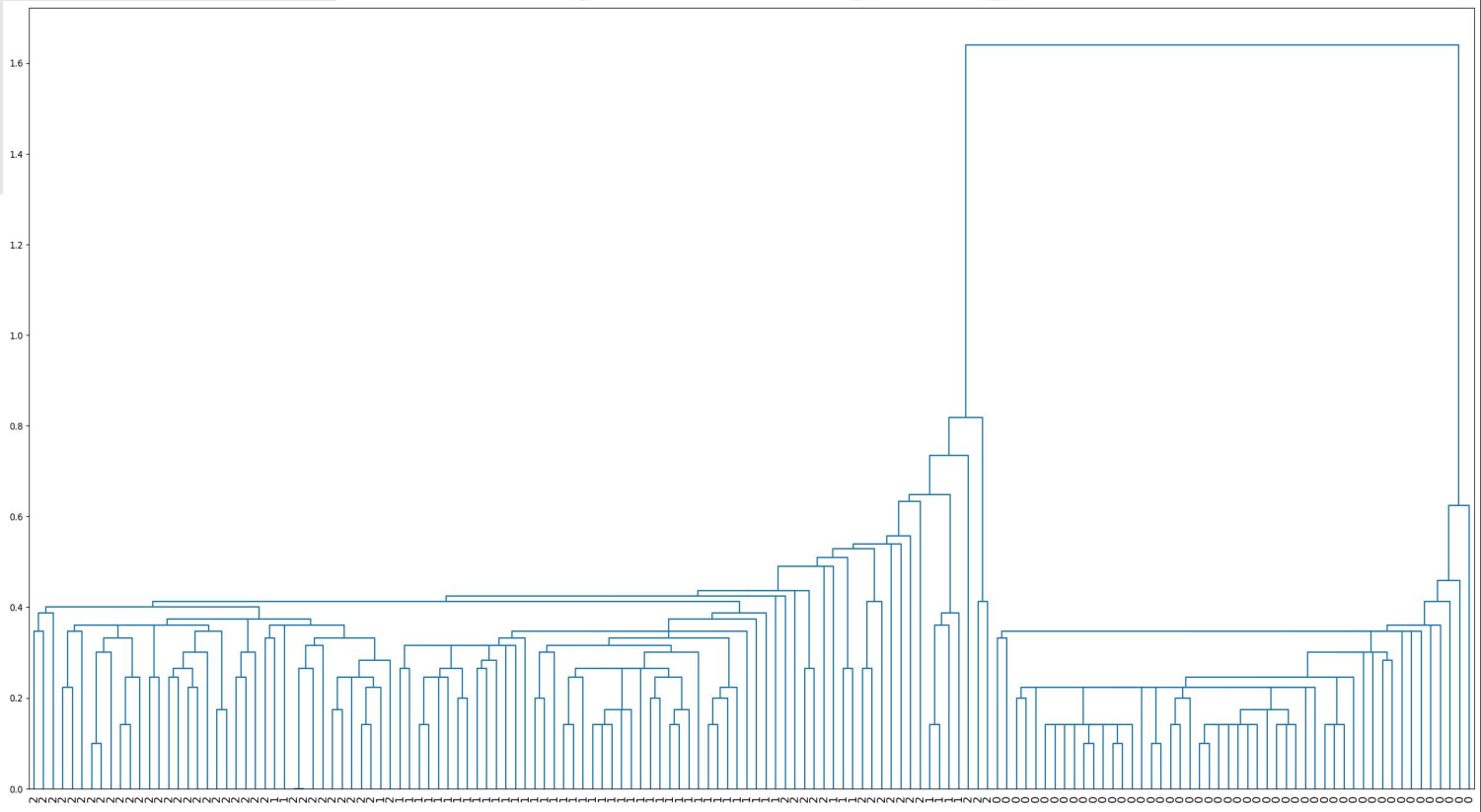
Clustering gerarchico in python

ITINERIS

```
from scipy.cluster.hierarchy import linkage,
dendrogram

hc = linkage(distanze, method="single")
hc_2 = linkage(distanze, method="complete")

listaTipi = iris.target
```

Clustering gerarchico in python

```
plt.figure(figsize=(30,15))
dendrogram(hc,
           orientation='top',
           labels=listaTipi,
           distance_sort='descending',
           show_leaf_counts=True,
           leaf_font_size=12
           )
plt.show()
```



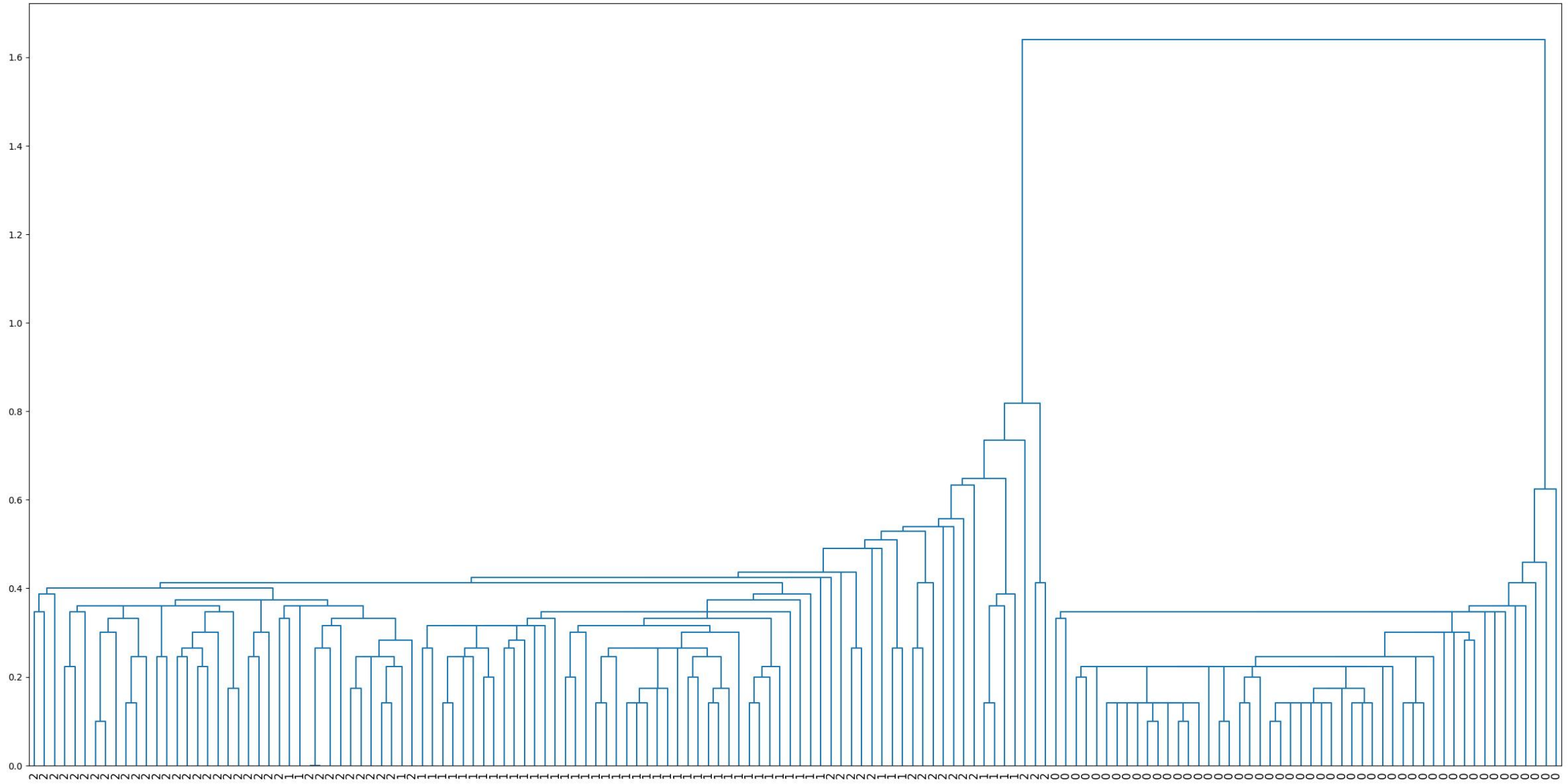
Clustering gerarchico in python

ITINERIS

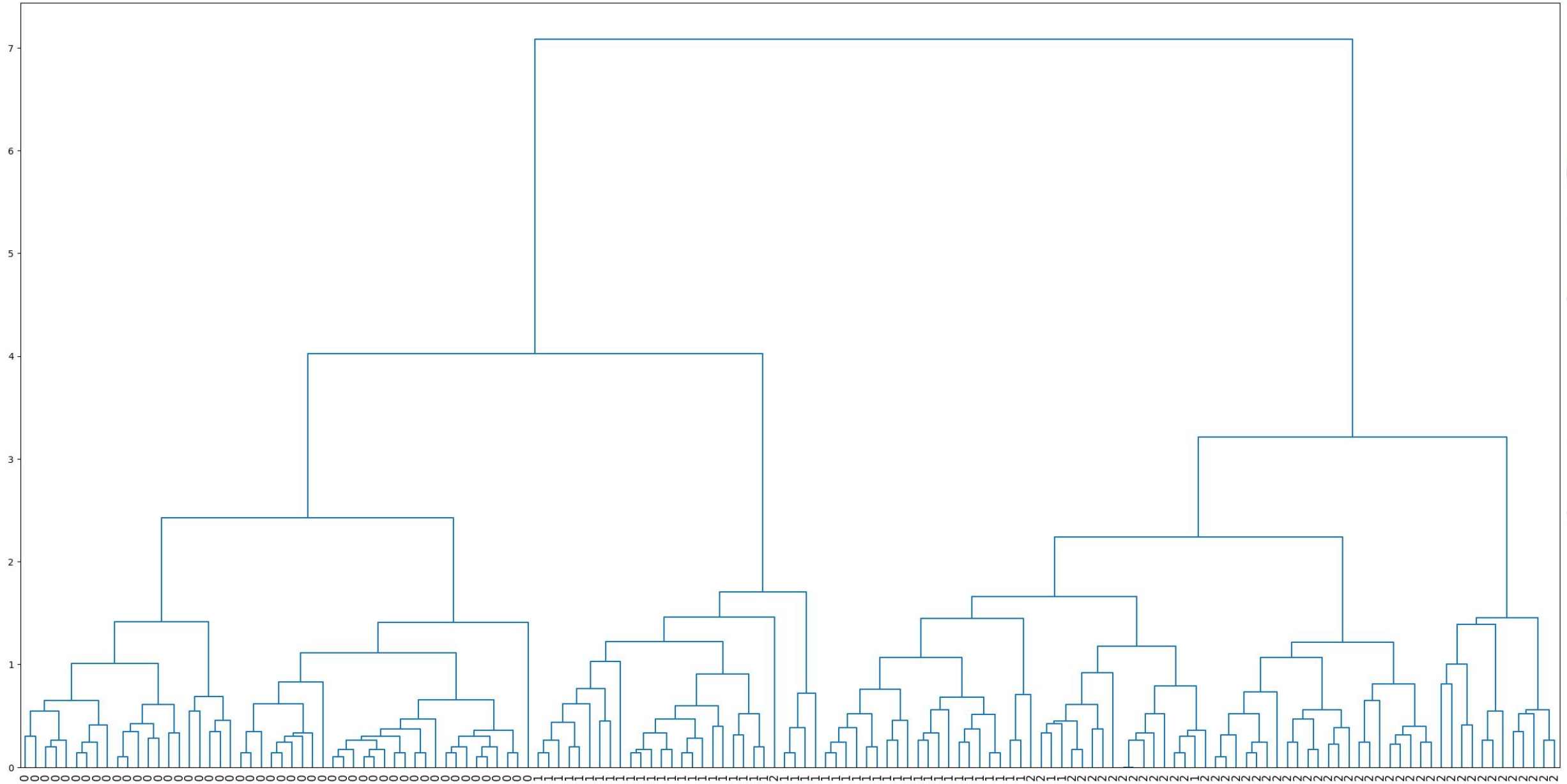
```
plt.figure(figsize=(30,15))
dendrogram(hc, #clustering
           orientation='top', #orientazione
           labels=listaTipi, #classi sull'asse x
           distance_sort='descending', #ordine
           show_leaf_counts=True, #ripetizioni nei campioni
           leaf_font_size=12, #font asse x
           color_threshold=0 #colore in base alla distanza
           )
plt.show()
```

Clustering gerarchico in python: single linking

ITINERIS



Clustering gerarchico in python: complete linking



THANKS!

IR0000032 – ITINERIS, Italian Integrated Environmental Research Infrastructures System
(D.D. n. 130/2022 - CUP B53C22002150006) Funded by EU - Next Generation EU PNRR-Mission 4 “Education and Research” - Component 2: “From research to business” - Investment 3.1: “Fund for the realisation of an integrated system of research and innovation infrastructures”

Finanziato dall'Unione europea NextGenerationEU
Ministero dell'Università e della Ricerca
Italiadomani