

K-means clustering: using "sklearn" and sythetic data

Cared Nicolle Castaños Manjarrez - A01742620

The notebook aims to study and implement a k-means clustering using "sklearn". A synthetic dataset will be used to identify clusters automatically using the K-means method.

Acknowledgments

- Inquiries: mauricio.antelis@tec.mx

✓ Importing libraries

```
# Define where you are running the code: colab or local
RunInColab          = True      # (False: no | True: yes)

# If running in colab:
if RunInColab:
    # Mount your google drive in google colab
    from google.colab import drive
    drive.mount('/content/drive')

    # Find location
    #!pwd
    #!ls
    #!ls "/content/drive/My Drive/Colab Notebooks/MachineLearningWithPython/"

    # Define path del proyecto
    Ruta          = "/content/drive/My Drive/TC1002S/NotebooksStudents/A01742620"

else:
    # Define path del proyecto
    Ruta          = ""

    Mounted at /content/drive

# Import the packages that we will be using
import numpy as np          # For array
import pandas as pd         # For data handling
import seaborn as sns       # For advanced plotting
import matplotlib.pyplot as plt # For showing plots

# Note: specific functions of the "sklearn" package will be imported when needed to show
```

✓ Importing data

```
# Dataset url
url = "/content/drive/My Drive/TC1002S/NotebooksProfessor/datasets/SyntheticData4Cluster:

# Load the dataset
df = pd.read_csv(url)
```

✓ Understanding and preprocessing the data

1. Get a general 'feel' of the data

```
# Print the dataframe
df
```

	x1	x2	x3	x4	x5	x6
0	1.914825	-1.380503	-3.609674	4.236011	-5.158681	5.712978
1	1.356415	9.767893	7.263659	8.750819	5.568930	-6.039122
2	1.185186	11.528344	9.999419	7.890027	7.308210	-8.899397
3	-1.739155	12.648965	7.965588	7.850296	10.235743	-10.175542
4	7.890985	-3.210880	-7.672016	2.438106	3.310904	-3.308334
...
1019	3.685106	-1.715503	-5.674443	6.510551	-0.121862	-6.166649
1020	-7.014173	-9.697874	4.093272	-0.590262	-9.882245	2.339336
1021	-2.993762	7.528182	7.877165	8.895835	9.318544	-7.445100
1022	4.576644	-1.720788	-6.581909	4.745839	1.497980	-4.828975
1023	2.616634	0.274593	-5.521864	9.582110	0.878266	-8.274990

1024 rows × 6 columns

```
# get the number of observations and variables
df.shape
```

(1024, 6)

2. Drop rows with any missing values

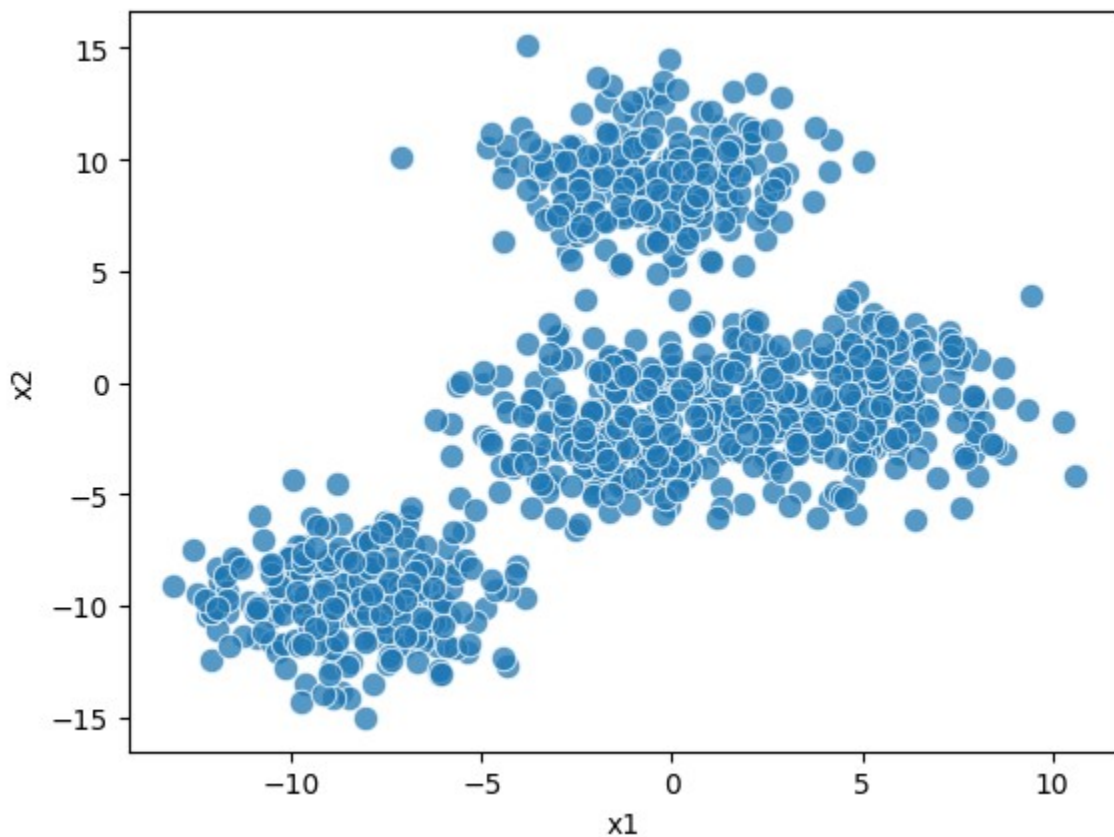
```
# Drop rows with NaN values if existing
df.dropna()

# Print the new shape
print(df.shape)

(1024, 6)
```

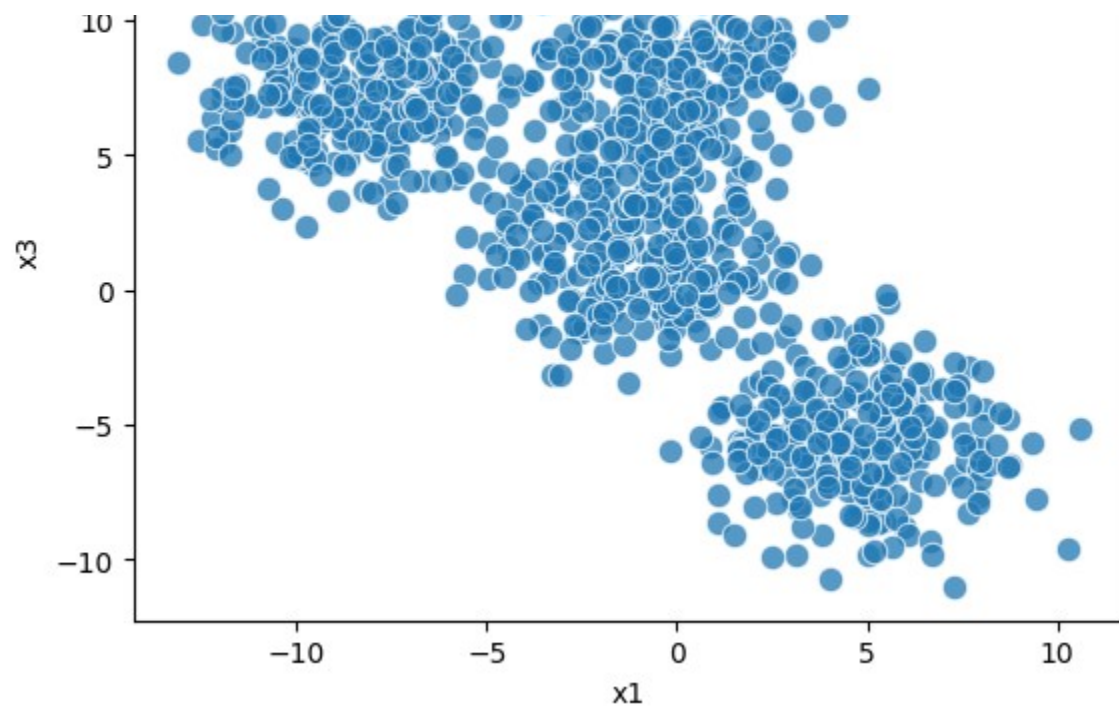
3. Scatterplot

```
# Scatterplot of x1 and x2
sns.scatterplot(data=df, x="x1", y="x2", s=80, alpha=0.75)
plt.show()
```



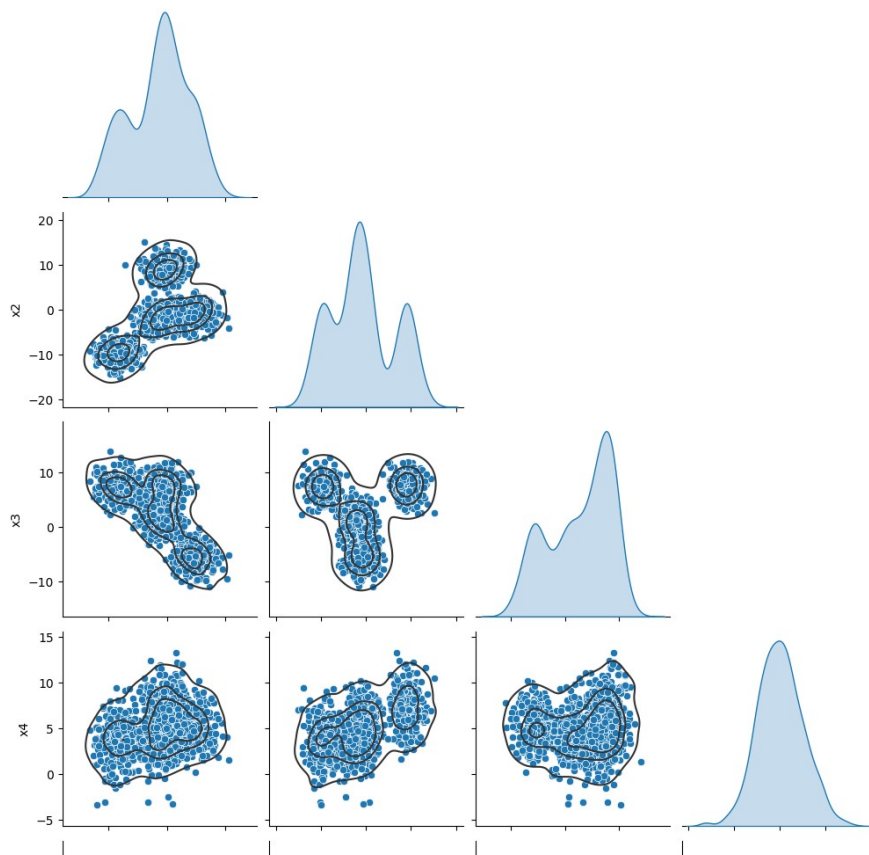
```
# Scatterplot of x1 and x3
sns.scatterplot(data=df, x="x1", y="x3", s=80, alpha=0.75)
plt.show()
```

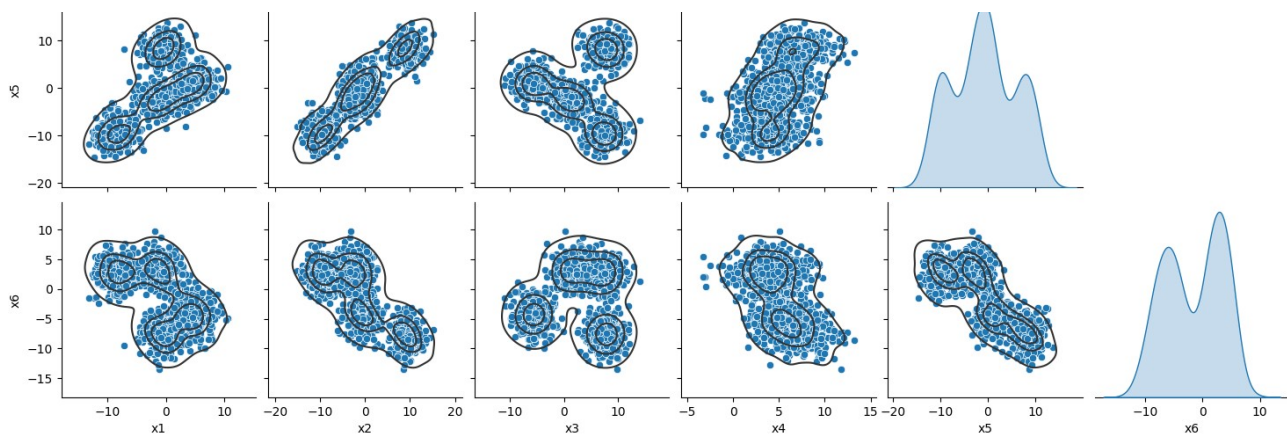




Difficult to plot independetly all combinations, let's use pairplot

```
# Pairplot: Scatterplot of all variables
g = sns.pairplot(df, corner =True, diag_kind="kde")
g.map_lower(sns.kdeplot, levels=4, color=".2")
plt.show()
```





It looks like there are 3 or 4 clusters/groups

Note that we do not know in advance the class/cluster/group to which each point belongs to:
we need to apply unsupervised learning ;

✓ Kmeans clustering

Kmeans clustering

```
# Import sklearn KMeans
from sklearn.cluster import KMeans

# Define number of clusters
K = 3 # Let's assume there are 2,3,4,5...? clusters/groups

# Create/Initialize the Kmeans box/object
km = KMeans(n_clusters=K, n_init="auto")
```

```
# Do K-means clustering (assing each point in the dataset to a cluster)
yestimated = km.fit_predict(df)
#yestimated = km.fit_predict(df[['x1','x2']] )
#yestimated = km.fit_predict(df[['x1','x3']] )
```

```
# Print estimated cluster of each point in the dataset
```

```
yestimated
```

```
array([2, 0, 0, ..., 0, 2, 2], dtype=int32)
```

```
# Add a new column to the dataset with the cluster information
```

```
df['yestimated'] = yestimated
```

```
df
```

	x1	x2	x3	x4	x5	x6	yestimated
0	1.914825	-1.380503	-3.609674	4.236011	-5.158681	5.712978	2
1	1.356415	9.767893	7.263659	8.750819	5.568930	-6.039122	0
2	1.185186	11.528344	9.999419	7.890027	7.308210	-8.899397	0
3	-1.739155	12.648965	7.965588	7.850296	10.235743	-10.175542	0
4	7.890985	-3.210880	-7.672016	2.438106	3.310904	-3.308334	2
...
1019	3.685106	-1.715503	-5.674443	6.510551	-0.121862	-6.166649	2
1020	-7.014173	-9.697874	4.093272	-0.590262	-9.882245	2.339336	1
1021	-2.993762	7.528182	7.877165	8.895835	9.318544	-7.445100	0
1022	4.576644	-1.720788	-6.581909	4.745839	1.497980	-4.828975	2
1023	2.616634	0.274593	-5.521864	9.582110	0.878266	-8.274990	2

```
1024 rows × 7 columns
```

```
# Print the Labels/Names of the existing clusters
```

```
df.yestimated.unique()
```

```
array([2, 0, 1], dtype=int32)
```

```
# Cluster centroides
```

```
km.cluster_centers_
```

```
array([[ -0.44229417,  9.13121533,  7.61409814,  7.22984721,  8.13001382,
        -7.62642221 ],
       [-8.3650671 , -9.59550917,  7.40711607,  3.77249056, -9.44226128,
         2.67666451],
       [ 1.85043266, -1.34592151, -2.11883656,  4.5718429 , -0.79519547,
```

```
-0.55114018]])

# Sum of squared error (sse) of the final model
km.inertia_

44295.1263266536

# The number of iterations required to converge
km.n_iter_

4
```

Important remarks

- The number of each cluster is randomly assigned
- The order of the number in each cluster is random

✓ Compute the number of points in each cluster

```
# Get a dataframe with the data of each cluster
df0 = df[df.yestimated==0]
df1 = df[df.yestimated==1]
df2 = df[df.yestimated==2]

# Number of points in cluster 0
NpointsCluster0 = df0.shape[0]
NpointsCluster1 = df1.shape[0]
NpointsCluster2 = df2.shape[0]

# Print
print("Number of points in clusters 0: " + str(NpointsCluster0) )
print("Number of points in clusters 1: " + str(NpointsCluster1) )
print("Number of points in clusters 02: " + str(NpointsCluster2) )

# Print total number of points
print("Total Number of points in the clusters: " + str(NpointsCluster0+NpointsCluster1+NpointsCluster2))

Number of points in clusters 0: 256
Number of points in clusters 1: 259
Number of points in clusters 02: 509
Total Number of points in the clusters: 1024
```

✓ Plot estimated clusters

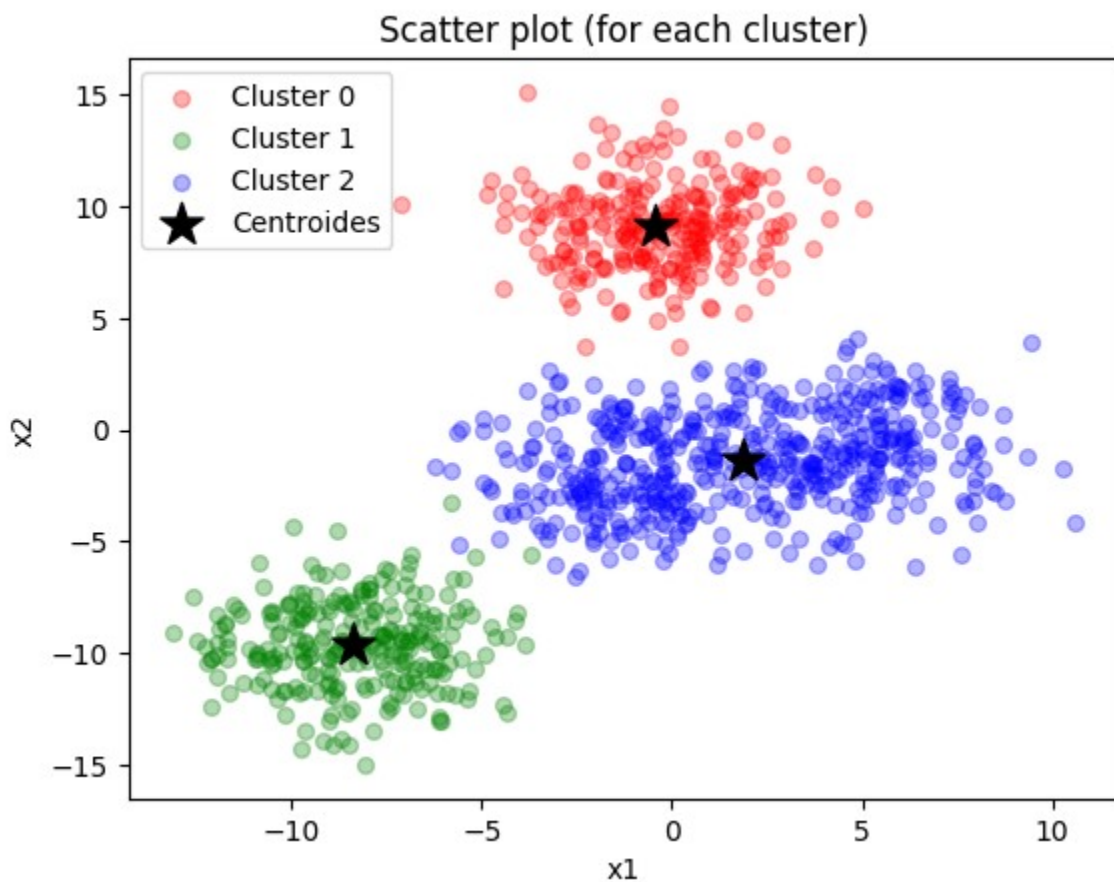
Plot estimated clusters

```
# Get a dataframe with the data of each cluster
df1 = df[df.yestimated==0]
df2 = df[df.yestimated==1]
df3 = df[df.yestimated==2]

# Scatter plot of each cluster
plt.scatter(df1.x1, df1.x2, label='Cluster 0', c='r', marker='o', s=32, alpha=0.3)
plt.scatter(df2.x1, df2.x2, label='Cluster 1', c='g', marker='o', s=32, alpha=0.3)
plt.scatter(df3.x1, df3.x2, label='Cluster 2', c='b', marker='o', s=32, alpha=0.3)

# Plot centroids
plt.scatter(km.cluster_centers[:,0], km.cluster_centers[:,1], color='black', marker='*')

plt.title('Scatter plot (for each cluster)')
plt.xlabel('x1')
plt.ylabel('x2')
plt.legend()
plt.show()
```



✓ Selecting K: elbow plot

Check the accuracy of the model using k-fold cross-validation

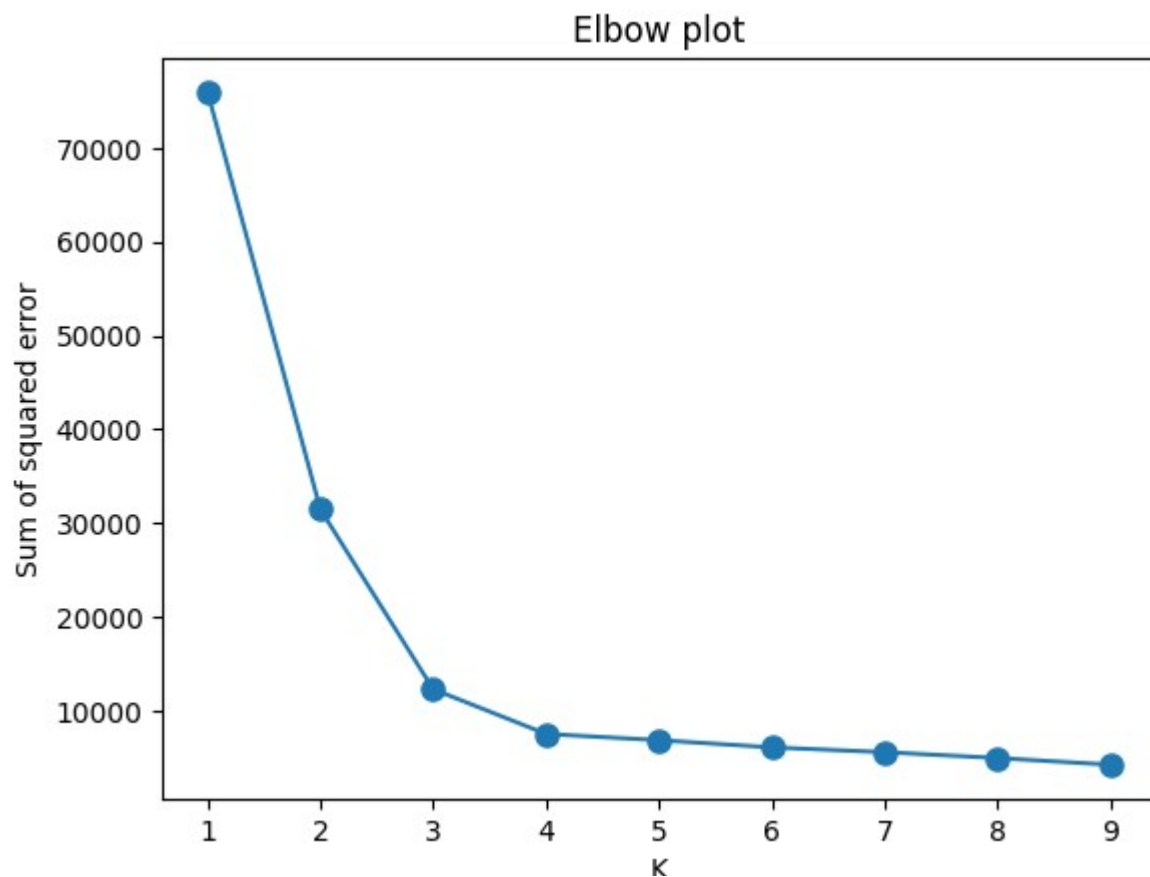
```
# Intialize a list to hold sum of squared error (sse)
sse = []

# Define values of k
k_rng = range(1,10)

# For each k
for k in k_rng:
    # Create model
    km = KMeans(n_clusters=k, n_init="auto")
    # Do K-means clustering
    km.fit_predict(df[['x1','x2']])
    # Save sse for each k
    sse.append(km.inertia_)

# Plot sse versus k
plt.plot(k_rng,sse, 'o-', markersize=8)

plt.title('Elbow plot')
plt.xlabel('K')
plt.ylabel('Sum of squared error')
plt.show()
```



..

Choose the k after which the sse is minimally reduced

Important remarks

- Observations?

Final remarks

- K-Means clustering algorithm is perhaps the simplest and most popular unsupervised learning algorithm
- The number of clusters have to be defined by the user (i.e., by you ;))
- The number assigned to each cluster is randomly assigned from set 0, 1, 2
- If there is no information about the number of clusters k, then use the elbow plot method to choose the best number of clusters k
- The order of the number in each cluster is random
- The **sklearn** package provides the tools for data processing suchs as k-means

Activity:

1. Repeat this analysis using other pair of features, e.g., x3 and x6
2. Repeat this analysis using all six features, e.g., x1, x2,..., x6
3. Provide conclusions

Activity: work with the iris dataset

1. Do clustering with the iris flower dataset to form clusters using as features the four features
2. Do clustering with the iris flower dataset to form clusters using as features the two petal measurements: Drop out the other two features
3. Do clustering with the iris flower dataset to form clusters using as features the two sepal measurements: Drop out the other two features

measurements. Drop out the other two features

- Which one provides the better grouping? Solve this using programming skills, e.g., compute performance metrics

1. Do clustering with the iris flower dataset to form clusters using as features the four features

```
# Dataset url
url = "/content/drive/My Drive/TC1002S/NotebooksProfessor/datasets/iris/iris.csv"

# Load the dataset
newHeader=["Sepal_length", "Sepal_width", "Petal_length", "Petal_width", "Class"]
dataset2 = pd.read_csv(url, header=None, names=newHeader )

#dataset = dataset.rename(columns={"Sepal_length": 0, "Sepal_width": 1, "Petal_length": 2, "Petal_width": 3, "Class": 4})

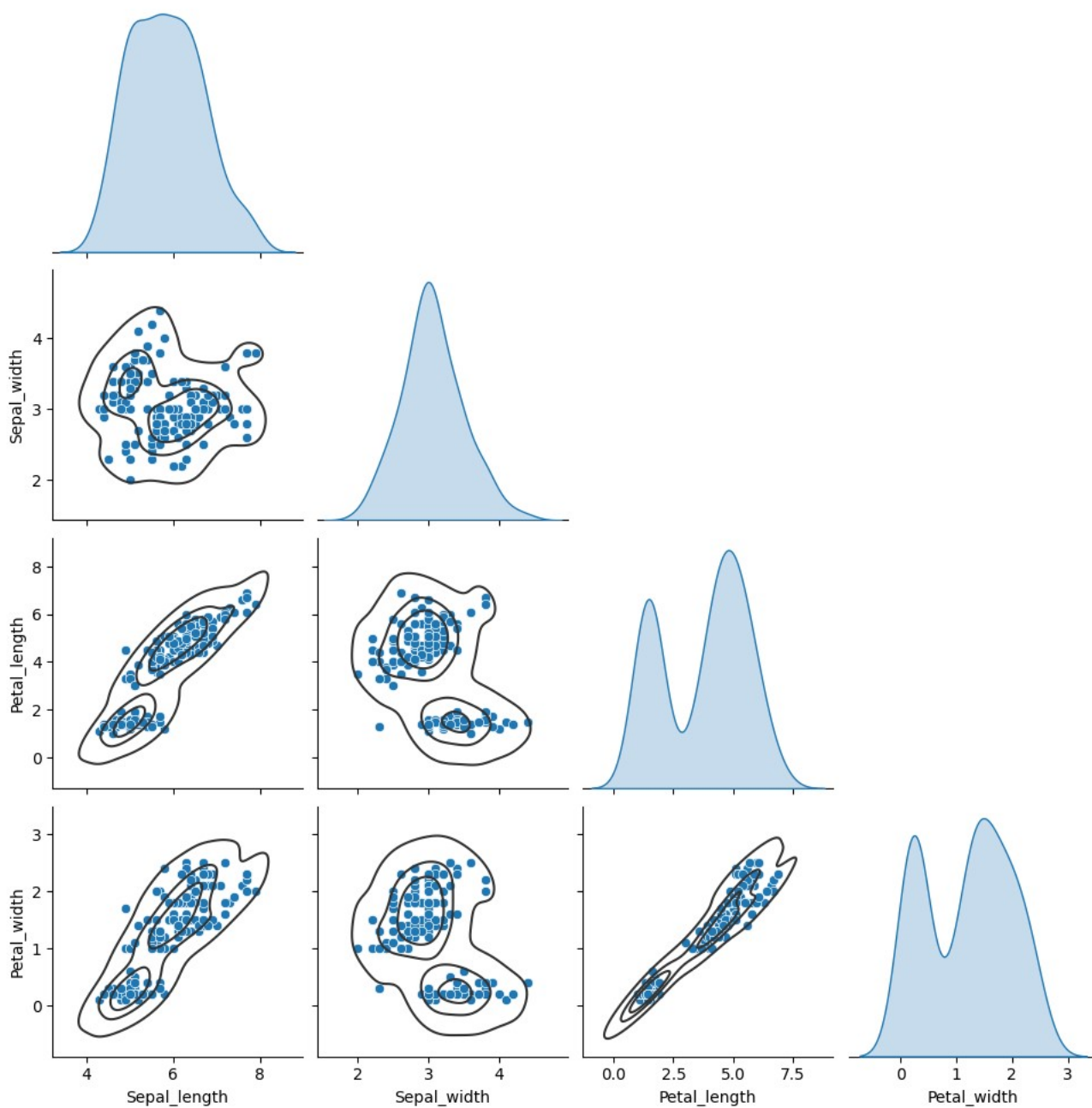
# Print the dataset
dataset2
```

	Sepal_length	Sepal_width	Petal_length	Petal_width	Class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
...
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 5 columns

```
# Pairplot: Scatterplot of all variables
g = sns.pairplot(dataset2, corner =True, diag_kind="kde")
g.map_lower(sns.kdeplot, levels=4, color=".2")
plt.show()
```

```
plt.show()
```



```

# Define number of clusters
from sklearn.cluster import KMeans
K = 2 # Let's assume there are 2,3,4,5...? clusters/groups

# Create/Initialize the Kmeans box/object
km = KMeans(n_clusters=K, n_init="auto")

# Do K-means clustering (assing each point in the dataset to a cluster)
yestimated = km.fit_predict(dataset2[['Sepal_length', 'Sepal_width', 'Petal_length',
#yestimated = km.fit_predict(df[['x1','x2']] )
#yestimated = km.fit_predict(df[['x1','x3']] )

# Print estimated cluster of each point in the dataset
yestimated

array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
      1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
      1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], dtype=int32)

dataset2['yestimated'] = yestimated

dataset2

```

	Sepal_length	Sepal_width	Petal_length	Petal_width	Class	yestimated
0	5.1	3.5	1.4	0.2	Iris-setosa	1
1	4.9	3.0	1.4	0.2	Iris-setosa	1
2	4.7	3.2	1.3	0.2	Iris-setosa	1
3	4.6	3.1	1.5	0.2	Iris-setosa	1
4	5.0	3.6	1.4	0.2	Iris-setosa	1
...
145	6.7	3.0	5.2	2.3	Iris-virginica	0
146	6.3	2.5	5.0	1.9	Iris-virginica	0
147	6.5	3.0	5.2	2.0	Iris-virginica	0
148	6.2	3.4	5.4	2.3	Iris-virginica	0
149	5.9	3.0	5.1	1.8	Iris-virginica	0

150 rows × 6 columns

```
# Print the Labels/Names of the existing clusters
dataset2.yestimated.unique()

array([1, 0], dtype=int32)

# Cluster centroides
km.cluster_centers_

array([[6.30103093, 2.88659794, 4.95876289, 1.69587629],
       [5.00566038, 3.36981132, 1.56037736, 0.29056604]])

# Sum of squared error (sse) of the final model
km.inertia_

152.3479517603579

# The number of iterations required to converge
km.n_iter_

4

# Get a dataframe with the data of each cluster
dt0 = dataset2[dataset2.yestimated==0]
dt1 = dataset2[dataset2.yestimated==1]

# Number of points in cluster 0
NpointsCluster0 = dt0.shape[0]
NpointsCluster1 = dt1.shape[0]

# Print
print("Number of points in clusters 0: " + str(NpointsCluster0) )
print("Number of points in clusters 1: " + str(NpointsCluster1) )

# Print total number of points
print("Total Number of points in the clusters: " + str(NpointsCluster0+NpointsCluster1) )

    Number of points in clusters 0: 97
    Number of points in clusters 1: 53
    Total Number of points in the clusters: 150

# Initialize a list to hold sum of squared error (sse)
sse = []

# Define values of k
k_rng = range(1,10)

# For each k
for k in k_rng:
```

```
# Create model
km = KMeans(n_clusters=k, n_init="auto")
# Do K-means clustering
km.fit_predict(dataset2[['Sepal_length', 'Sepal_width', 'Petal_length', 'Petal_width']])
# Save sse for each k
sse.append(km.inertia_)

# Get a dataframe with the data of each cluster
dt1 = dataset2[dataset2.yestimated==0]
dt2 = dataset2[dataset2.yestimated==1]

# Scatter plot of each cluster
plt.scatter(dt1.Sepal_length, dt1.Sepal_width, label='Cluster 0', c='r', marker='o', s=3)
plt.scatter(dt2.Sepal_length, dt2.Sepal_width, label='Cluster 1', c='g', marker='o', s=3)

plt.scatter(5.00566038, 6.30103093, color='black', marker='*', label='Centroides', s=256)
plt.scatter(3.36981132, 2.88659794, color='black', marker='*', label='Centroides', s=256)
plt.scatter(1.56037736, 4.95876289, color='black', marker='*', label='Centroides', s=256)
plt.scatter(0.29056604, 1.69587629, color='black', marker='*', label='Centroides', s=256)
plt.scatter(km.cluster_centers_[0], km.cluster_centers_[1], color='black', marker='*')

plt.title('Scatter plot (for each cluster)')
plt.xlabel('x1')
plt.ylabel('x2')
plt.legend()
plt.show()

plt.scatter(dt1.Sepal_length, dt1.Petal_width, label='Cluster 0', c='r', marker='o', s=3)
plt.scatter(dt2.Sepal_length, dt2.Petal_width, label='Cluster 1', c='g', marker='o', s=3)

plt.scatter(5.00566038, 6.30103093, color='black', marker='*', label='Centroides', s=256)
plt.scatter(3.36981132, 2.88659794, color='black', marker='*', label='Centroides', s=256)
plt.scatter(1.56037736, 4.95876289, color='black', marker='*', label='Centroides', s=256)
plt.scatter(0.29056604, 1.69587629, color='black', marker='*', label='Centroides', s=256)
plt.scatter(km.cluster_centers_[0], km.cluster_centers_[1], color='black', marker='*')

plt.title('Scatter plot (for each cluster)')
plt.xlabel('x1')
plt.ylabel('x2')
plt.legend()
plt.show()

plt.scatter(dt1.Petal_length, dt1.Sepal_width, label='Cluster 0', c='r', marker='o', s=3)
plt.scatter(dt2.Petal_length, dt2.Sepal_width, label='Cluster 1', c='g', marker='o', s=3)

plt.scatter(5.00566038, 6.30103093, color='black', marker='*', label='Centroides', s=256)
plt.scatter(3.36981132, 2.88659794, color='black', marker='*', label='Centroides', s=256)
plt.scatter(1.56037736, 4.95876289, color='black', marker='*', label='Centroides', s=256)
```

```

plt.scatter(4.30037738, 4.33070293, color='black', marker='*', label='Centroides', s=256)
plt.scatter(0.29056604, 1.69587629, color='black', marker='*', label='Centroides', s=256)
plt.scatter(km.cluster_centers_[0], km.cluster_centers_[1], color='black', marker='*')

plt.title('Scatter plot (for each cluster)')
plt.xlabel('x1')
plt.ylabel('x2')
plt.legend()
plt.show()

plt.scatter(dt1.Petal_length, dt1.Petal_width, label='Cluster 0', c='r', marker='o', s=3)
plt.scatter(dt2.Petal_length, dt2.Petal_width, label='Cluster 1', c='g', marker='o', s=3)

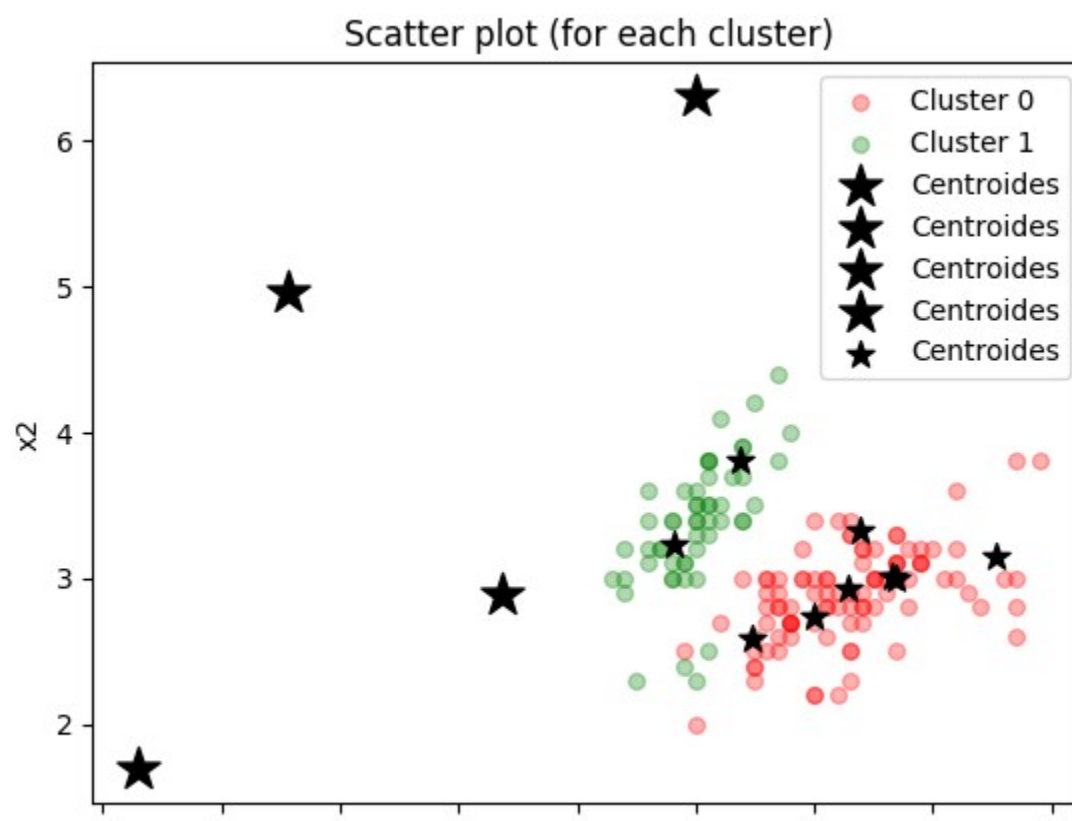
plt.scatter(5.00566038, 6.30103093, color='black', marker='*', label='Centroides', s=256)
plt.scatter(3.36981132, 2.88659794, color='black', marker='*', label='Centroides', s=256)
plt.scatter(1.56037736, 4.95876289, color='black', marker='*', label='Centroides', s=256)
plt.scatter(0.29056604, 1.69587629, color='black', marker='*', label='Centroides', s=256)
plt.scatter(km.cluster_centers_[0], km.cluster_centers_[1], color='black', marker='*')

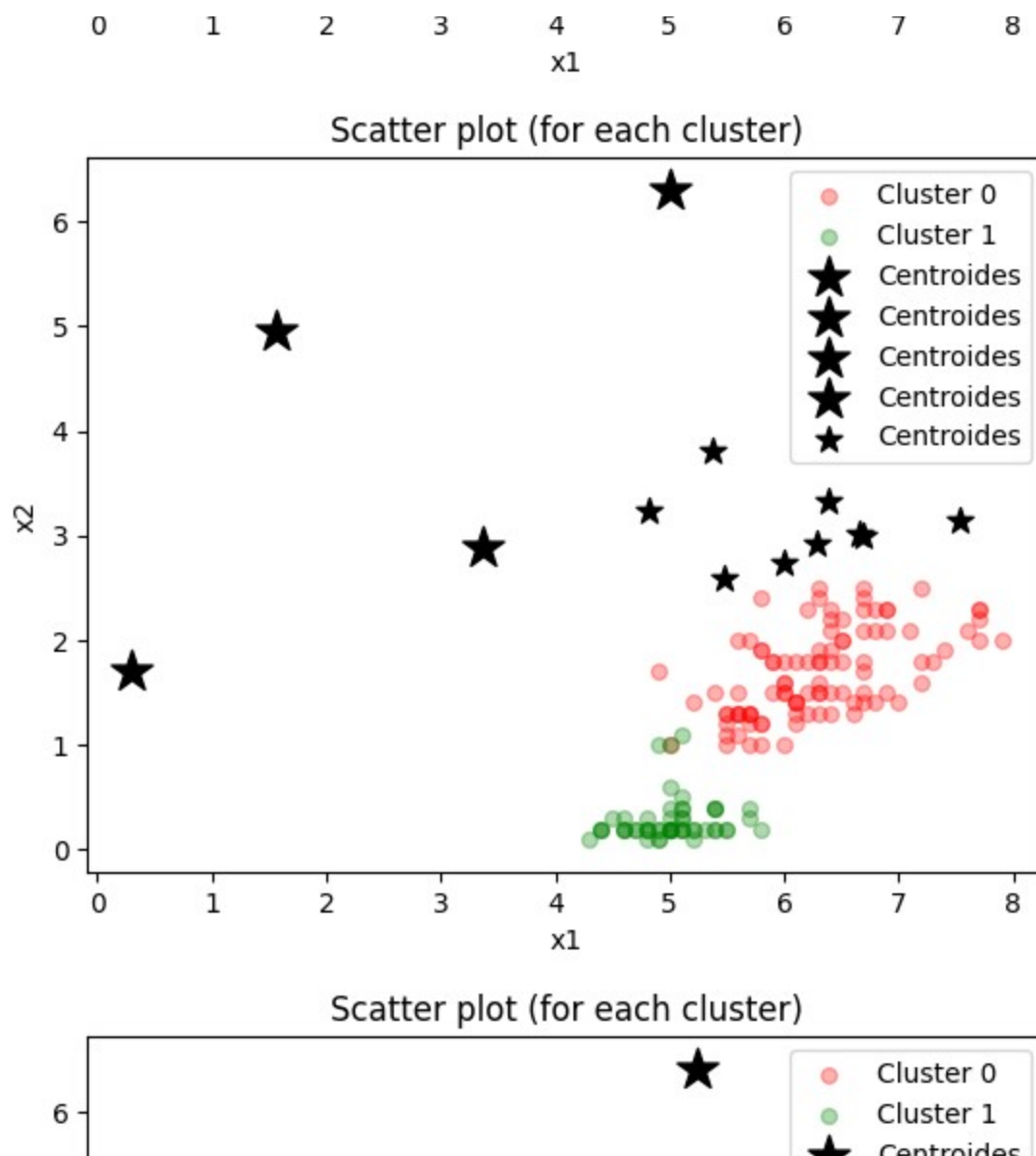
#g = sns.pairplot(dataset2, corner =True, diag_kind="kde")
#g.map_lower(sns.kdeplot, levels=4, color=".2")

# Plot centroides

plt.title('Scatter plot (for each cluster)')
plt.xlabel('x1')
plt.ylabel('x2')
plt.legend()
plt.show()

```





2. Do clustering with the iris flower dataset to form

- clusters using as features the two petal measurements:
Drop out the other two features

```
# Dataset url
url = "/content/drive/My Drive/TC1002S/NotebooksProfessor/datasets/iris/iris.csv"

# Load the dataset
newHeader=["Sepal_length", "Sepal_width", "Petal_length", "Petal_width", "Class"]
dataset2 = pd.read_csv(url, header=None, names=newHeader )
dataset21= dataset2.loc[:,["Petal_length", "Petal_width"]]

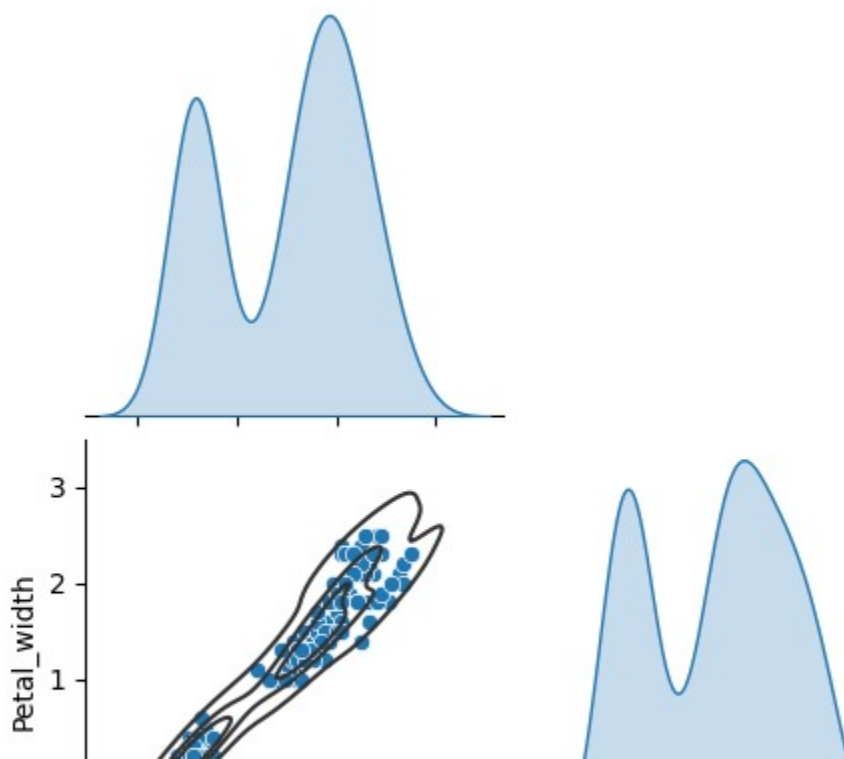
#dataset = dataset.rename(columns={"Sepal_length": 0, "Sepal_width": 1, "Petal_length": :
```

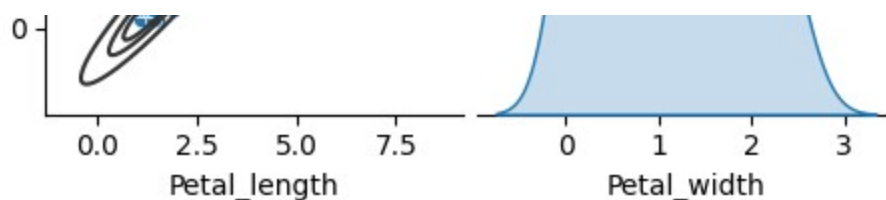
```
# Print the dataset  
dataset21
```

	Petal_length	Petal_width
0	1.4	0.2
1	1.4	0.2
2	1.3	0.2
3	1.5	0.2
4	1.4	0.2
...
145	5.2	2.3
146	5.0	1.9
147	5.2	2.0
148	5.4	2.3
149	5.1	1.8

150 rows × 2 columns

```
# Pairplot: Scatterplot of all variables  
g = sns.pairplot(dataset21, corner=True, diag_kind="kde")  
g.map_lower(sns.kdeplot, levels=4, color=".2")  
plt.show()
```





```
# Define number of clusters
from sklearn.cluster import KMeans
K = 2 # Let's assume there are 2,3,4,5...? clusters/groups

# Create/Initialize the Kmeans box/object
km = KMeans(n_clusters=K, n_init="auto")

# Do K-means clustering (assing each point in the dataset to a cluster)
yestimated = km.fit_predict(dataset21[['Petal_length', 'Petal_width' ]])
#yestimated = km.fit_predict(df[['x1','x2']] )
#yestimated = km.fit_predict(df[['x1','x3']] )

# Print estimated cluster of each point in the dataset
yestimated
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], dtype=int32)
```

```
dataset21['yestimated'] = yestimated
```

```
dataset21
```

	Petal_length	Petal_width	yestimated
0	1.4	0.2	0
1	1.4	0.2	0
2	1.3	0.2	0
3	1.5	0.2	0
4	1.4	0.2	0
...
145	5.2	2.3	1
146	5.0	1.9	1
147	5.2	2.0	1
148	5.4	2.3	1

1.0	0.0	2.0	1
149	5.1	1.8	1

150 rows × 3 columns

```
# Print the Labels/Names of the existing clusters
```

```
dataset21.yestimated.unique()
```

```
array([0, 1], dtype=int32)
```

```
# Cluster centroides
```

```
km.cluster_centers_
```

```
array([[1.49215686, 0.2627451 ],
       [4.92525253, 1.68181818]])
```

```
# Sum of squared error (sse) of the final model
```

```
km.inertia_
```

```
86.39021984551397
```

```
# The number of iterations required to converge
```

```
km.n_iter_
```

```
3
```

```
# Get a dataframe with the data of each cluster
```

```
dt0 = dataset21[dataset21.yestimated==0]
```

```
dt1 = dataset21[dataset21.yestimated==1]
```

```
# Number of points in cluster 0
```

```
NpointsCluster0 = dt0.shape[0]
```

```
NpointsCluster1 = dt1.shape[0]
```

```
# Print
```

```
print("Number of points in clusters 0: " + str(NpointsCluster0) )
```

```
print("Number of points in clusters 1: " + str(NpointsCluster1) )
```

```
# Print total number of points
```

```
print("Total Number of points in the clusters: " + str(NpointsCluster0+NpointsCluster1) )
```

```
Number of points in clusters 0: 51
```

```
Number of points in clusters 1: 99
```

```
Total Number of points in the clusters: 150
```

```
# Initialize a list to hold sum of squared error (sse)
```

```
sse = []
```

```
# Define values of k
```

```
k_rng = range(1,10)
```

```

# For each k
for k in k_rng:
    # Create model
    km = KMeans(n_clusters=k, n_init="auto")
    # Do K-means clustering
    km.fit_predict(dataset2[['Petal_length', 'Petal_width']])
    # Save sse for each k
    sse.append(km.inertia_)

# Get a dataframe with the data of each cluster
dt1 = dataset21[dataset21.yestimated==0]
dt2 = dataset21[dataset21.yestimated==1]

# Scatter plot of each cluster
plt.scatter(dt1.Petal_length, dt1.Petal_width, label='Cluster 0', c='r', marker='o', s=3)
plt.scatter(dt2.Petal_length, dt2.Petal_width, label='Cluster 1', c='g', marker='o', s=3)

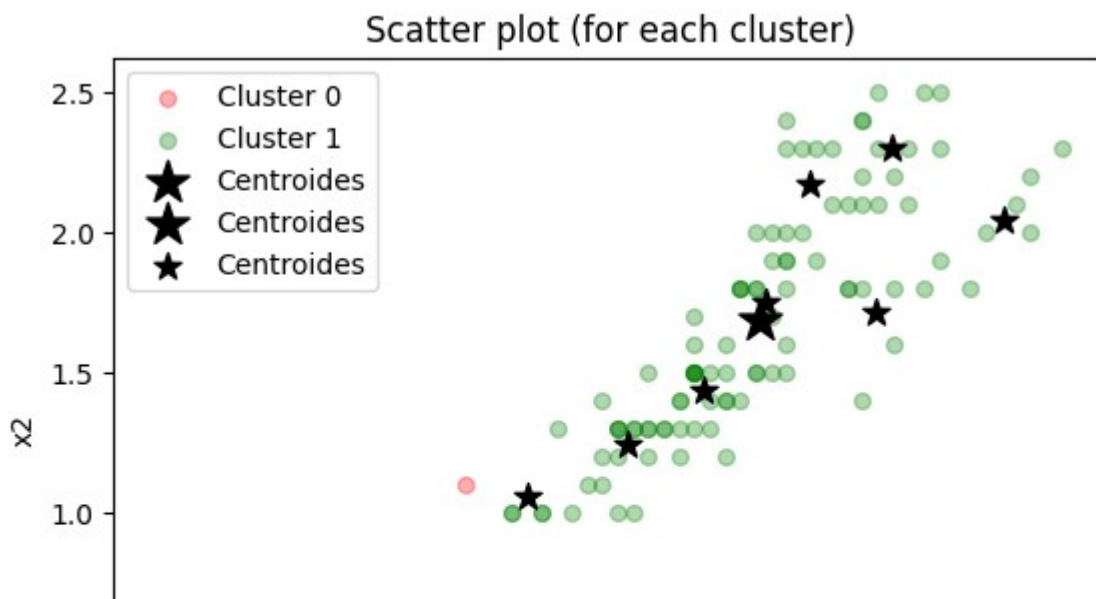
plt.scatter(4.92525253, 1.68181818, color='black', marker='*', label='Centroides', s=256)
plt.scatter(1.49215686, 0.2627451, color='black', marker='*', label='Centroides', s=256)
plt.scatter(km.cluster_centers_[0], km.cluster_centers_[1], color='black', marker='*')

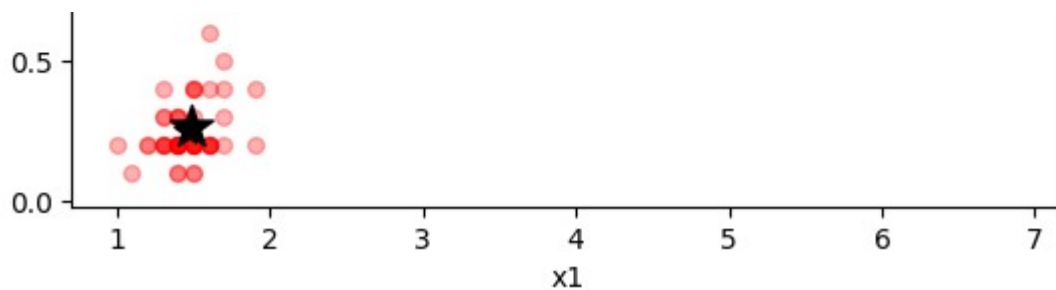
#g = sns.pairplot(dataset2, corner =True, diag_kind="kde")
#g.map_lower(sns.kdeplot, levels=4, color=".2")

# Plot centroides

plt.title('Scatter plot (for each cluster)')
plt.xlabel('x1')
plt.ylabel('x2')
plt.legend()
plt.show()

```





3. Do clustering with the iris flower dataset to form

- ✓ clusters using as features the two sepal measurements:
Drop out the other two features

```
# Dataset url
url = "/content/drive/My Drive/TC1002S/NotebooksProfessor/datasets/iris/iris.csv"

# Load the dataset
newHeader=["Sepal_length", "Sepal_width", "Petal_length", "Petal_width", "Class"]
dataset2 = pd.read_csv(url, header=None, names=newHeader )
dataset21= dataset2.loc[:,["Sepal_length", "Sepal_width"]]

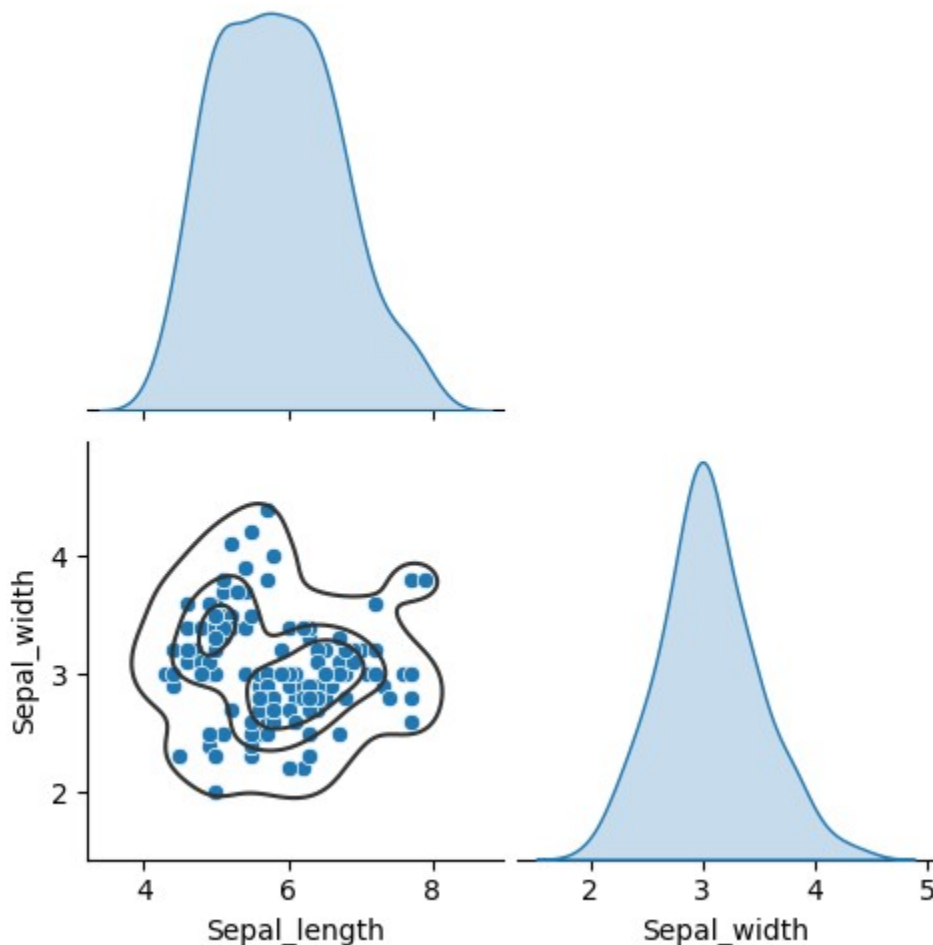
#dataset = dataset.rename(columns={"Sepal_length": 0, "Sepal_width": 1, "Petal_length": 2, "Petal_width": 3, "Class": 4})

# Print the dataset
dataset21
```

	Sepal_length	Sepal_width
0	5.1	3.5
1	4.9	3.0
2	4.7	3.2
3	4.6	3.1
4	5.0	3.6
...
145	6.7	3.0
146	6.3	2.5
147	6.5	3.0
148	6.2	3.4
149	5.9	3.0

150 rows × 2 columns

```
# Pairplot: Scatterplot of all variables
g = sns.pairplot(dataset21, corner =True, diag_kind="kde")
g.map_lower(sns.kdeplot, levels=4, color=".2")
plt.show()
```



```
# Define number of clusters
from sklearn.cluster import KMeans
K = 2 # Let's assume there are 2,3,4,5...? clusters/groups

# Create/Initialize the Kmeans box/object
km = KMeans(n_clusters=K, n_init="auto")

# Do K-means clustering (assing each point in the dataset to a cluster)
yestimated = km.fit_predict(dataset21[['Sepal_length', 'Sepal_width' ]])
#yestimated = km.fit_predict(df[['x1','x2']] )
#yestimated = km.fit_predict(df[['x1','x3']] )

# Print estimated cluster of each point in the dataset
yestimated
array([0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0])
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1,
       0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1,
       0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1,
       1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], dtype=int32)
```

```
dataset21['yestimated'] = yestimated
```

```
dataset21
```

	Sepal_length	Sepal_width	yestimated
0	5.1	3.5	0
1	4.9	3.0	0
2	4.7	3.2	0
3	4.6	3.1	0
4	5.0	3.6	0
...
145	6.7	3.0	1
146	6.3	2.5	1
147	6.5	3.0	1
148	6.2	3.4	1
149	5.9	3.0	1

```
150 rows × 3 columns
```

```
# Print the Labels/Names of the existing clusters
```

```
dataset21.yestimated.unique()
```

```
array([0, 1], dtype=int32)
```

```
# Cluster centroides
```

```
km.cluster_centers_
```

```
array([[5.15      , 3.16891892],
       [6.51842105, 2.94868421]])
```

```
# Sum of squared error (sse) of the final model
```

```
km.inertia_
```

```
58.44759246088195
```



```
# The number of iterations required to converge
km.n_iter_

8

# Get a dataframe with the data of each cluster
dt0 = dataset21[dataset21.yestimated==0]
dt1 = dataset21[dataset21.yestimated==1]

# Number of points in cluster 0
NpointsCluster0 = dt0.shape[0]
NpointsCluster1 = dt1.shape[0]

# Print
print("Number of points in clusters 0: " + str(NpointsCluster0) )
print("Number of points in clusters 1: " + str(NpointsCluster1) )

# Print total number of points
print("Total Number of points in the clusters: " + str(NpointsCluster0+NpointsCluster1) )

    Number of points in clusters 0: 74
    Number of points in clusters 1: 76
    Total Number of points in the clusters: 150

# Initialize a list to hold sum of squared error (sse)
sse = []

# Define values of k
k_rng = range(1,10)

# For each k
for k in k_rng:
    # Create model
    km = KMeans(n_clusters=k, n_init="auto")
    # Do K-means clustering
    km.fit_predict(dataset21[['Sepal_length', 'Sepal_width']])
    # Save sse for each k
    sse.append(km.inertia_)

# Get a dataframe with the data of each cluster
dt1 = dataset21[dataset21.yestimated==0]
dt2 = dataset21[dataset21.yestimated==1]

# Scatter plot of each cluster
plt.scatter(dt1.Sepal_length, dt1.Sepal_width, label='Cluster 0', c='r', marker='o', s=3)
plt.scatter(dt2.Sepal_length, dt2.Sepal_width, label='Cluster 1', c='g', marker='o', s=3)

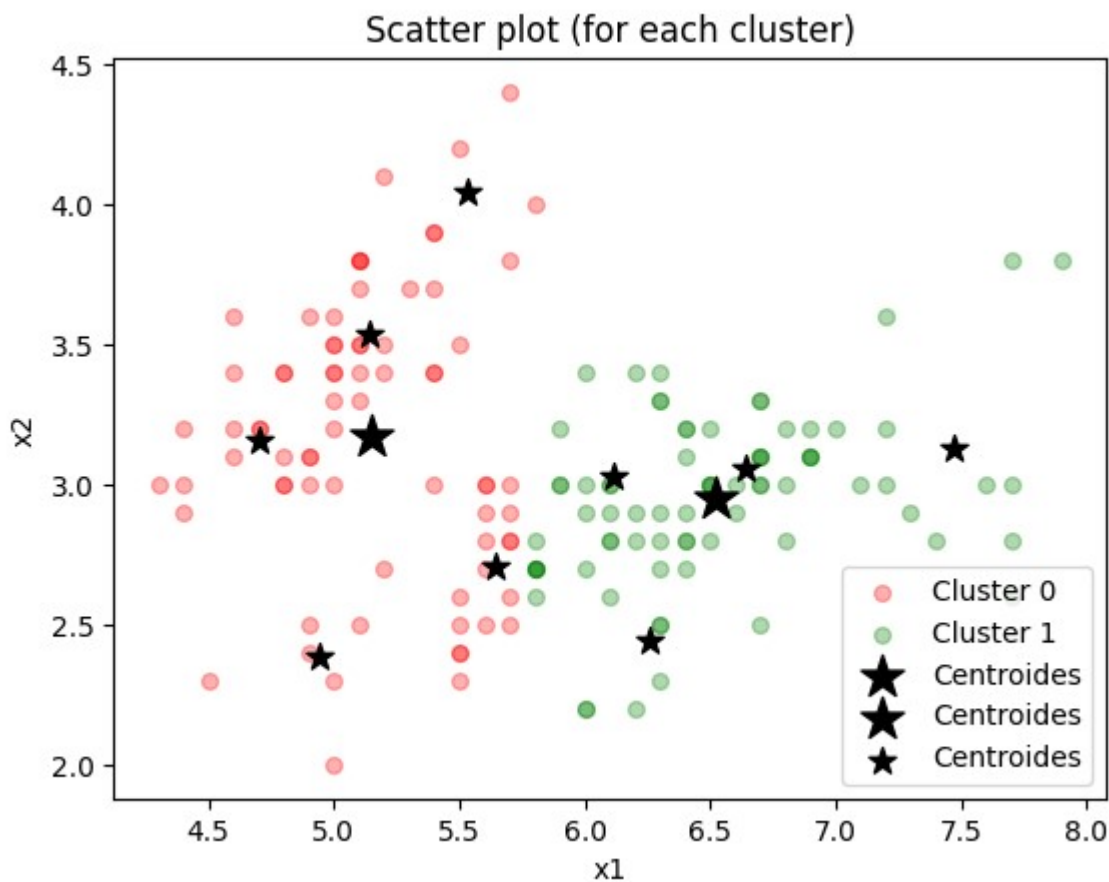
plt.scatter(5.15, 3.16891892, color='black', marker='*', label='Centroides', s=256)
plt.scatter(6.51842105, 2.94868421, color='black', marker='*', label='Centroides', s=256)
plt.scatter(km.cluster_centers_[0][0], km.cluster_centers_[0][1], color='black', marker='*')
```

```
plt.scatter(km.cluster_centers_[0], km.cluster_centers_[1], color='black', marker='^')

#g = sns.pairplot(dataset2, corner=True, diag_kind="kde")
#g.map_lower(sns.kdeplot, levels=4, color=".2")

# Plot centroides

plt.title('Scatter plot (for each cluster)')
plt.xlabel('x1')
plt.ylabel('x2')
plt.legend()
plt.show()
```



```
print(km.cluster_centers_)

[[4.7      3.155    ]
 [6.64285714 3.05357143]
 [6.115     3.025    ]
 [5.64583333 2.70416667]
 [7.475     3.125    ]
 [5.14090909 3.53181818]
 [4.94285714 2.38571429]
 [5.52857143 4.04285714]
 [6.26      2.44     ]]
```

4 Which one provides the better grouping? Solve this

- .. which one provides the better grouping. Come and
- ✓ using programming skills, e.g., compute performance metrics

Firstly, we have to take into consideration that, while in reality there are 3 groups of flowers in the dataset, our number of clusters was instead considered to be 2, because all of the plots showed two groups of observations.

Even though, by principal, the best option to get better grouping would be all four of the features, due to the very different positions of the clusters in each plot of each combination of two features, the centroids that we get in the end are not as defined as the ones where only two features are used.

Out of the two where only two features line up, the centroids that seems to match up the best are the ones where the petal measurements were considered