K-means clustering

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

In [ ]:

```
# 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 nee
```

Importing data

In [ ]:

```
# Dataset url
path = "/home/alex/TC1002S/NotebooksStudents/A01639643/iris/iris.csv"
header = ["sepal_length", "sepal_width", "petal_length", "petal_width", "Clas

# Load the dataset
df  = pd.read_csv(path, names = header)
ds = pd.read_csv(path, names = header)
dp = pd.read_csv(path, names = header)
```

In [ ]:

```
ds
```

Out[ ]:

	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

	sepal_length	sepal_width	petal_length	petal_width	Class
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

Undertanding and preprocessing the data

1. Get a general 'feel' of the data

In [ ]:

```
# Print the dataframe
df
```

Out[ ]:

	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

In [ ]:

```
# get the number of observations and variables
print("The number of observations are: ", df.shape[0])
print("The number of variables are: ", df.shape[1])
```

```
The number of observations are:  150
The number of variables are:  5
```

1. Drop rows with any missing values

In [ ]:

```
# Drop rows with NaN values if existing
df.dropna().describe()
print("The amount of NaN values in the dataset is: \n", df.isnull().sum())
df.notnull().sum()

# Print the new shape
df
```

```
The amount of NaN values in the dataset is: 
 sepal_length    0
sepal_width     0
petal_length    0
petal_width     0
Class           0
dtype: int64
```

Out[ ]:

	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

1. Scatterplot

In [ ]:

```
df.drop(columns=["Class"], inplace = True)
df
```

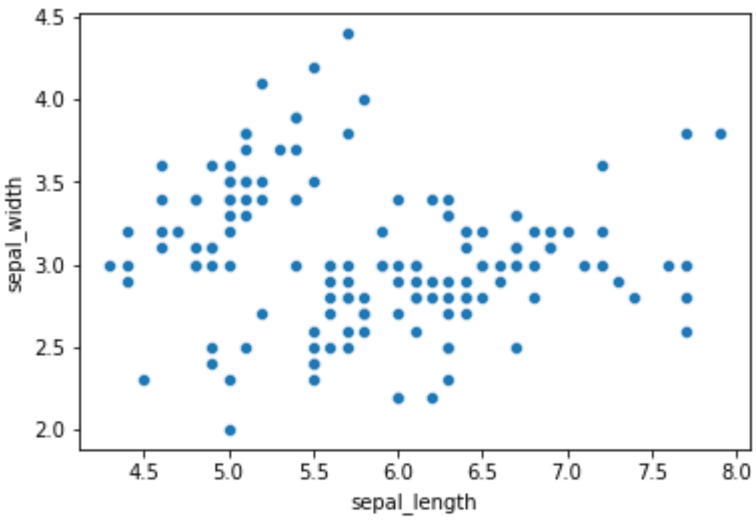
Out[ ]:

	sepal_length	sepal_width	petal_length	petal_width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
...	...	...	...	...
145	6.7	3.0	5.2	2.3

	sepal_length	sepal_width	petal_length	petal_width
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

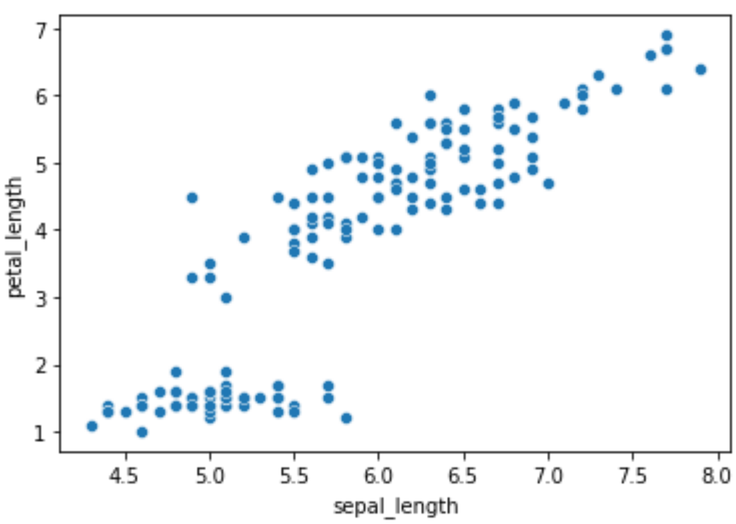
In [ ]:

```
# Scatterplot of x1 and x2
sns.scatterplot(data = df, x = "sepal_length", y = "sepal_width")
plt.show()
```



In [ ]:

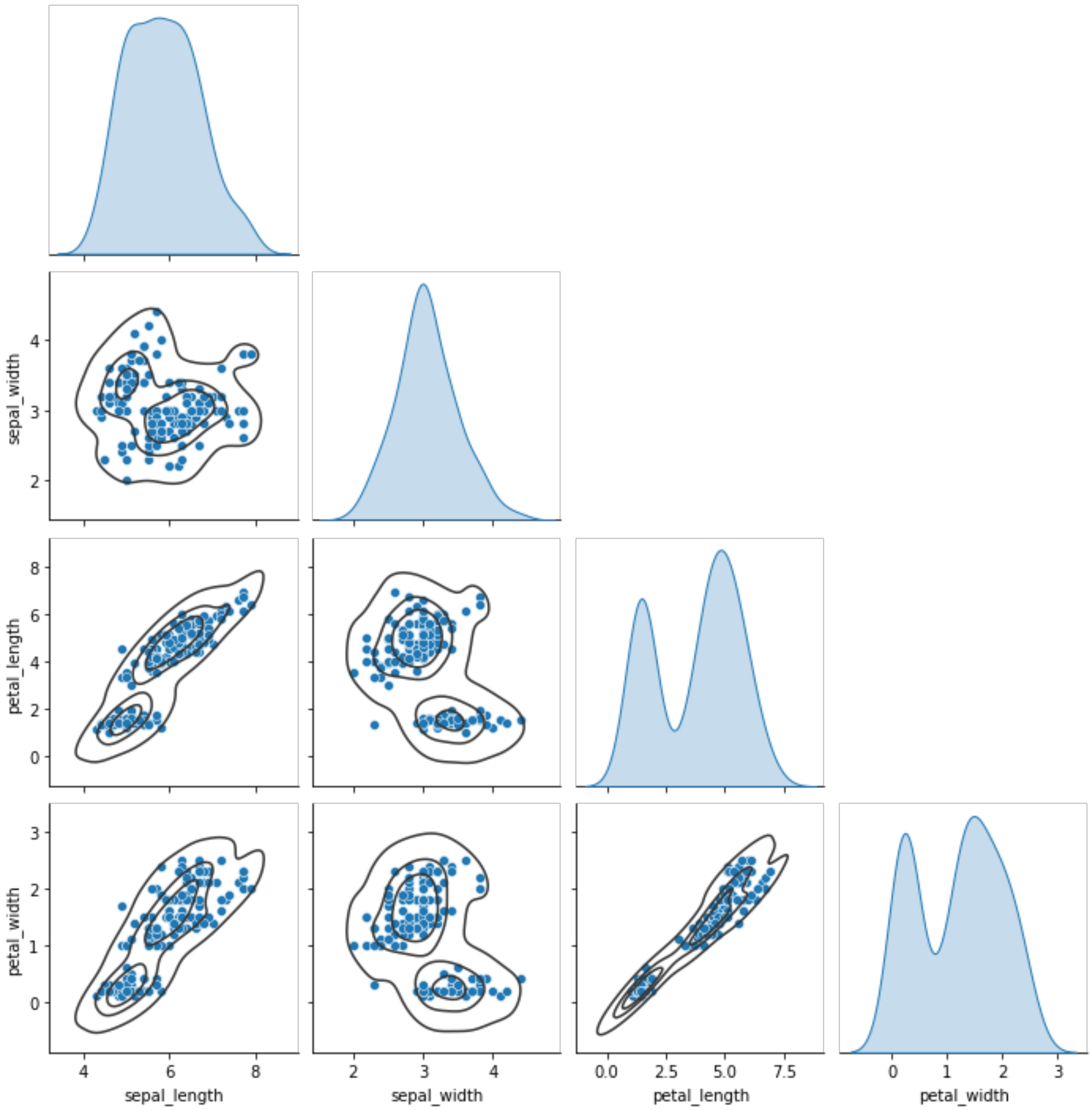
```
# Scatterplot of x1 and x3
sns.scatterplot(data = df, x = "sepal_length", y = "petal_length")
plt.show()
```



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

In [ ]:
```
# Pairplot: Scatterplot of all variables
# sns.set(style = "ticks", color_codes = True)
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 ¡

1.- Kmeans clustering (4 features)

Kmeans clustering

In [ ]:

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

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

# Create the Kmeans box/object
km_df = KMeans(n_clusters = K)

# Do K-means clustering (assing each point in the dataset to a cluster)
yestimated = km_df.fit_predict(df)


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

Out[ ]:

```
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, 1, 1, 3, 3, 3, 0, 3, 0, 3, 0, 3, 0, 0, 0, 0, 3, 0, 3,
       0, 0, 3, 0, 3, 0, 3, 3, 3, 3, 3, 3, 3, 0, 0, 0, 0, 3, 0, 3, 3, 3,
       0, 0, 0, 3, 0, 0, 0, 0, 0, 3, 0, 0, 2, 3, 2, 2, 2, 2, 0, 2, 2, 2,
       3, 3, 2, 3, 3, 2, 2, 2, 2, 3, 2, 3, 2, 3, 2, 2, 3, 3, 2, 2, 2, 2,
       2, 3, 3, 2, 2, 2, 3, 2, 2, 2, 3, 2, 2, 2, 3, 3, 2, 3], dtype=int32)
```

In [ ]:

```python
# Add a new column to the dataset with the cluster information
df['yestimated'] = yestimated
df
```

Out[ ]:

	sepal_length	sepal_width	petal_length	petal_width	yestimated
0	5.1	3.5	1.4	0.2	1
1	4.9	3.0	1.4	0.2	1
2	4.7	3.2	1.3	0.2	1
3	4.6	3.1	1.5	0.2	1
4	5.0	3.6	1.4	0.2	1
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	2
146	6.3	2.5	5.0	1.9	3
147	6.5	3.0	5.2	2.0	3
148	6.2	3.4	5.4	2.3	2
149	5.9	3.0	5.1	1.8	3

150 rows × 5 columns

In [ ]:

```python
# Print the labels/Names of the existing clusters
df.yestimated.unique()
```

Out[ ]:

```
array([1, 3, 0, 2], dtype=int32)
```

In [ ]:
```
df.drop(columns=['yestimated'], inplace = True)
df
```

Out[ ]:

	sepal_length	sepal_width	petal_length	petal_width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
...	...	...	...	...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

150 rows × 4 columns

In [ ]:
```
# Cluster centroides
km_df.cluster_centers_
```

Out[ ]:
```
array([[5.53214286, 2.63571429, 3.96071429, 1.22857143],
       [5.006     , 3.428     , 1.462     , 0.246     ],
       [6.9125    , 3.1       , 5.846875  , 2.13125   ],
       [6.2525    , 2.855     , 4.815     , 1.625     ]])
```

In [ ]:
```
# Sum of squared error (sse) of the final model
km_df.inertia_
```

Out[ ]:
```
57.22847321428572
```

In [ ]:
```
# The number of iterations required to converge
km_df.n_iter_
```

Out[ ]:
```
5
```

Important remarks

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

Plot estimated clusters

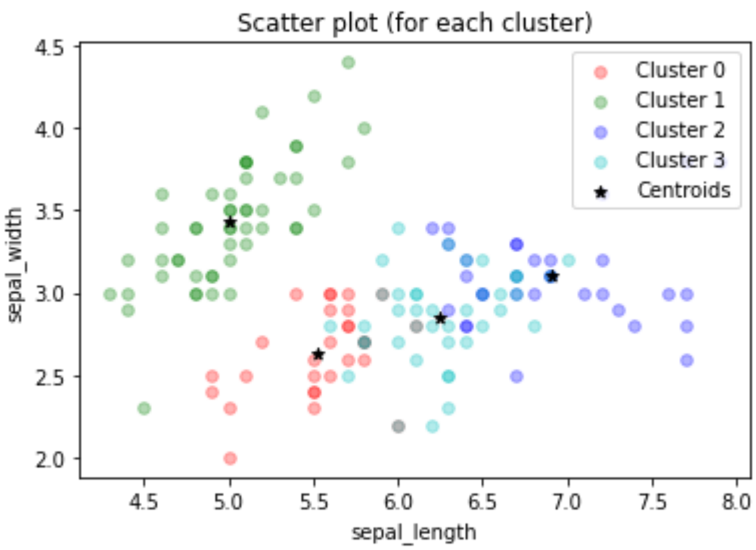
Plot estimated clusters

In [ ]:
```
# Get a dataframe with the data of each cluster
df1 = df[yestimated==0]
df2 = df[yestimated==1]
df3 = df[yestimated==2]
df4 = df[yestimated==3]

# Scatter plot of each cluster
plt.scatter(df1.sepal_length, df1.sepal_width, label="Cluster 0", c="r", mark
plt.scatter(df2.sepal_length, df2.sepal_width, label="Cluster 1", c="g", mark
plt.scatter(df3.sepal_length, df3.sepal_width, label="Cluster 2", c="b", mark
plt.scatter(df4.sepal_length, df4.sepal_width, label="Cluster 3", c="c", mark

#Plot centroids
plt.scatter(km_df.cluster_centers_[:,0], km_df.cluster_centers_[:,1], color='

plt.title("Scatter plot (for each cluster)")
plt.xlabel("sepal_length")
plt.ylabel("sepal_width")
plt.legend()
plt.show()
```



Selecting K: elbow plot

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

In [ ]:

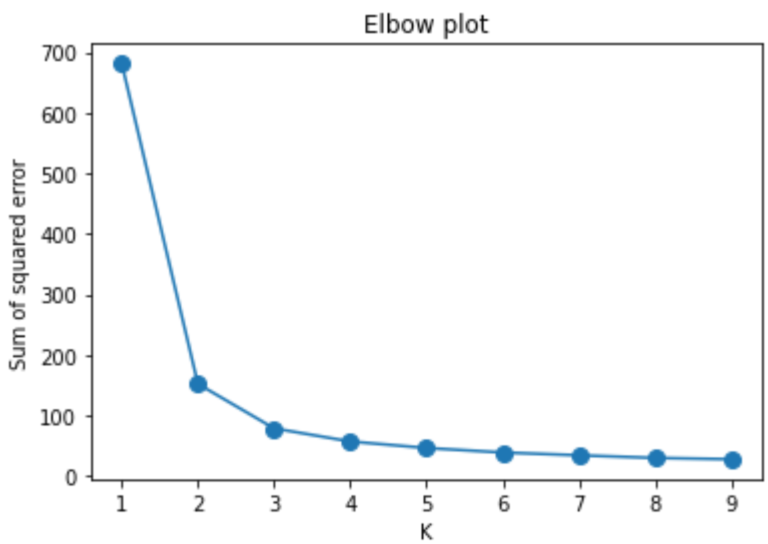
```
# Intialize a list to hold sum of squared error (sse)
sse = []

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

# For each k
for k in k_rng:
    #Create model
    km_df = KMeans(n_clusters=k)
    #Do K-means clustering
    km_df.fit_predict(df[["sepal_length", "sepal_width", "petal_length", "pet
    #Save sse for each k
    sse.append(km_df.inertia_)
```

In [ ]:

```
# 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

2.- Kmeans clustering (Two Petal measurements)

Kmeans clustering

In [ ]:

```
dp.drop(columns=["sepal_length"], inplace=True)
dp.drop(columns=["sepal_width"], inplace=True)
dp.drop(columns=["Class"], inplace=True)
dp
```

Out[ ]:

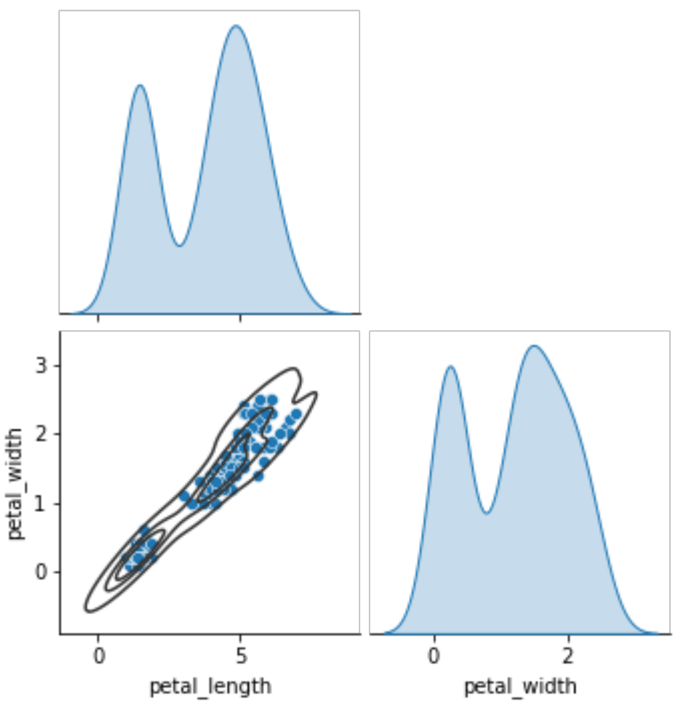
	petal_length	petal_width
0	1.4	0.2
1	1.4	0.2
2	1.3	0.2
3	1.5	0.2
4	1.4	0.2
...	...	...
145	5.2	2.3
146	5.0	1.9
147	5.2	2.0
148	5.4	2.3
149	5.1	1.8

150 rows × 2 columns

In [ ]:

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

plt.show()
```



In [ ]:
```
# 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 the Kmeans box/object
km_dp = KMeans(n_clusters = K)

# Do K-means clustering (assing each point in the dataset to a cluster)
yestimated = km_dp.fit_predict(dp)


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

Out[ ]:
```
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, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 2, 2, 2, 2, 2, 0, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], dtype=int32)
```

In [ ]:
```
# Add a new column to the dataset with the cluster information
dp['yestimated'] = yestimated
dp
```

Out[ ]:

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

150 rows × 3 columns

In [ ]:
```
# Print the labels/Names of the existing clusters
dp.yestimated.unique()
```

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

In [ ]:
```
dp.drop(columns=['yestimated'], inplace = True)
dp
```

Out[ ]:

	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

In [ ]:
```
# Cluster centroides
km_dp.cluster_centers_
```

Out[ ]:
```
array([[5.59583333, 2.0375    ],
       [1.462     , 0.246     ],
       [4.26923077, 1.34230769]])
```

In [ ]:
```
# Sum of squared error (sse) of the final model
km_dp.inertia_
```

Out[ ]:
```
31.371358974358966
```

In [ ]:
```
# The number of iterations required to converge
km_dp.n_iter_
```

Out[ ]:
```
7
```

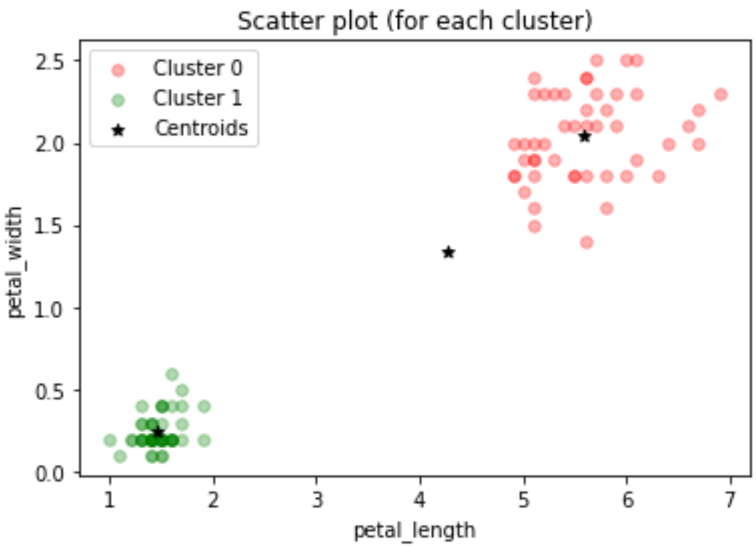
In [ ]:

```
# Get a dataframe with the data of each cluster
dp1 = dp[yestimated==0]
dp2 = dp[yestimated==1]

# Scatter plot of each cluster
plt.scatter(dp1.petal_length, dp1.petal_width, label="Cluster 0", c="r", mark
plt.scatter(dp2.petal_length, dp2.petal_width, label="Cluster 1", c="g", mark

#Plot centroids
plt.scatter(km_dp.cluster_centers_[:,0], km_dp.cluster_centers_[:,1], color='

plt.title("Scatter plot (for each cluster)")
plt.xlabel("petal_length")
plt.ylabel("petal_width")
plt.legend()
plt.show()
```



In [ ]:

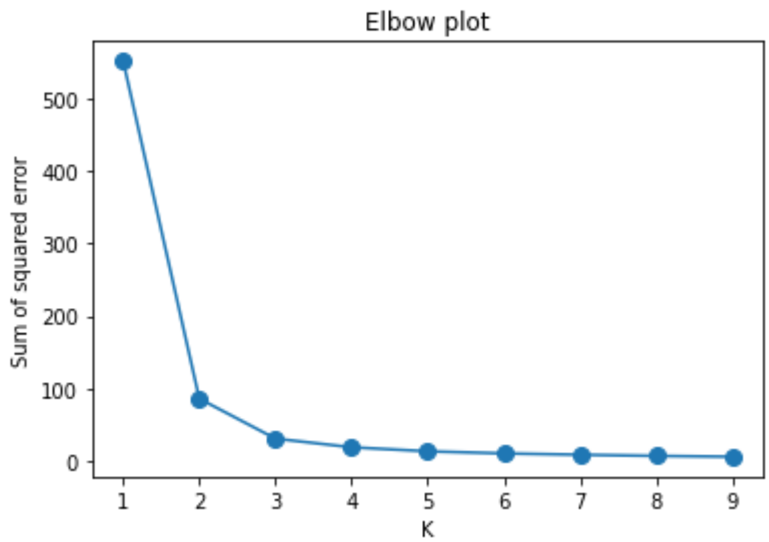
```
# Intialize a list to hold sum of squared error (sse)
sse = []

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

# For each k
for k in k_rng:
    #Create model
    km_dp = KMeans(n_clusters=k)
    #Do K-means clustering
    km_dp.fit_predict(df[["petal_length", "petal_width"]])
    #Save sse for each k
    sse.append(km_dp.inertia_)
```

In [ ]:

```
# 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()
```



3.- Kmeans clustering (Two Sepal measurements)

Kmeans clustering

In [ ]:

```
ds.drop(columns=["petal_length"], inplace=True)
ds.drop(columns=["petal_width"], inplace=True)
ds.drop(columns=["Class"], inplace=True)
ds
```

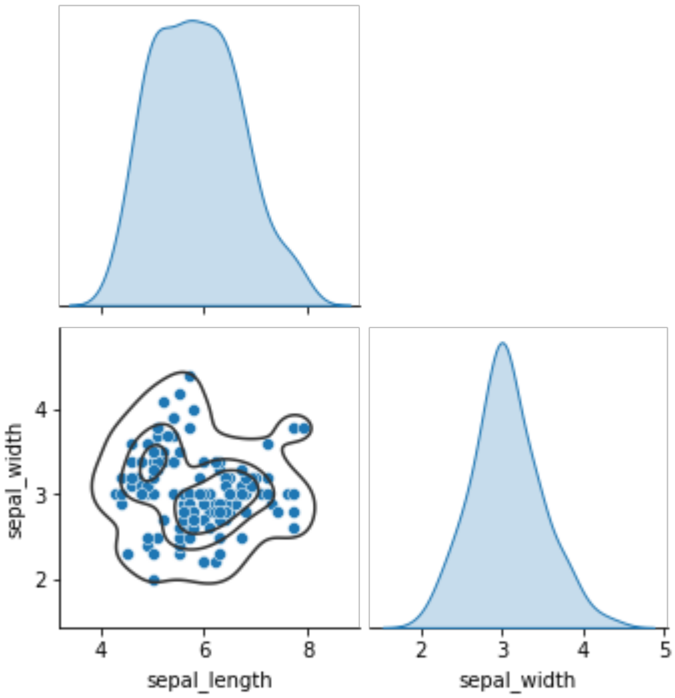
Out[ ]:

	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

	sepal_length	sepal_width
149	5.9	3.0

In [ ]:
```
# Pairplot: Scatterplot of all variables
g = sns.pairplot(ds, corner = True, diag_kind="kde")
g.map_lower(sns.kdeplot, levels=4, color=".2")

plt.show()
```



In [ ]:
```
# 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 the Kmeans box/object
km_ds = KMeans(n_clusters = K)

# Do K-means clustering (assing each point in the dataset to a cluster)
yestimated = km_ds.fit_predict(ds)


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

Out[ ]:
```
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, 2, 1, 2, 1, 2, 1, 2, 2, 2, 2, 2, 2, 1,
       2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 1, 1, 1, 1, 2, 1, 1, 1,
       1, 1, 1, 2, 2, 1, 1, 1, 1, 2, 1, 2, 1, 2, 1, 1, 2, 2, 1, 1, 1, 1,
       1, 2, 2, 1, 1, 1, 2, 1, 1, 1, 2, 1, 1, 1, 2, 1, 1, 2], dtype=int32)
```

In [ ]:

```
ds['yestimated'] = yestimated
ds
```

Out[ ]:

	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	2
147	6.5	3.0	1
148	6.2	3.4	1
149	5.9	3.0	2

150 rows × 3 columns

In [ ]:

```
# Print the labels/Names of the existing clusters
ds.yestimated.unique()
```

Out[ ]:

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

In [ ]:

```
ds.drop(columns=['yestimated'], inplace = True)
ds
```

Out[ ]:

	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

In [ ]:
```
# Cluster centroides
km_ds.cluster_centers_
```

Out[ ]:
```
array([[5.006     , 3.428     ],
       [6.81276596, 3.07446809],
       [5.77358491, 2.69245283]])
```

In [ ]:
```
# Sum of squared error (sse) of the final model
km_ds.inertia_
```

Out[ ]:
```
37.0507021276596
```

In [ ]:
```
# The number of iterations required to converge
km_ds.n_iter_
```

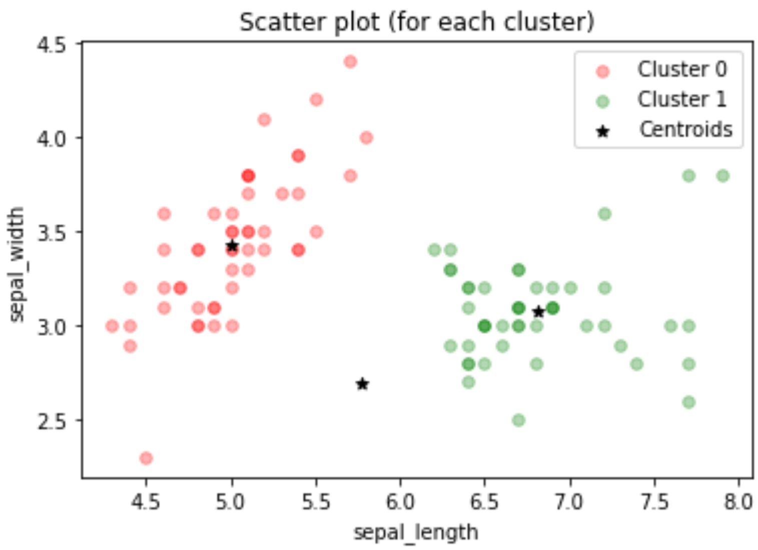
Out[ ]:
```
14
```

In [ ]:
```
# Get a dataframe with the data of each cluster
ds1 = ds[yestimated==0]
ds2 = ds[yestimated==1]

# Scatter plot of each cluster
plt.scatter(ds1.sepal_length, ds1.sepal_width, label="Cluster 0", c="r", mark
plt.scatter(ds2.sepal_length, ds2.sepal_width, label="Cluster 1", c="g", mark

#Plot centroids
plt.scatter(km_ds.cluster_centers_[:,0], km_ds.cluster_centers_[:,1], color='

plt.title("Scatter plot (for each cluster)")
plt.xlabel("sepal_length")
plt.ylabel("sepal_width")
plt.legend()
plt.show()
```



In [ ]:

```
# 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_ds = KMeans(n_clusters=k)
    #Do K-means clustering
    km_ds.fit_predict(df[["sepal_length", "sepal_width"]])
    #Save sse for each k
    sse.append(km_ds.inertia_)
```

In [ ]:

```
# 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()
```