Notes:

1. Please run the first two cells as is to load libraries
2. The code guides and output will help you develop your code
3. in case your results are not matching the previous output (left for you) this could be due to randomization and you should not worry about them
4. Most of the code is ready, all you need to do in fill in some parts of the codes to complete

loading Libraries

```
# run this cell as is

#  Essential Libraries
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

# run this cell as is
# For data
from sklearn import datasets
from sklearn.datasets import make_blobs

# For data processing
from sklearn.preprocessing import StandardScaler

# For modeling
from sklearn.cluster  import KMeans

# For evaluation
from sklearn.metrics import jaccard_score, adjusted_rand_score,
silhouette_score


# Other
import sklearn.metrics.pairwise as pw
# from math import factorial
# from itertools import combinations
```

Exercise 1

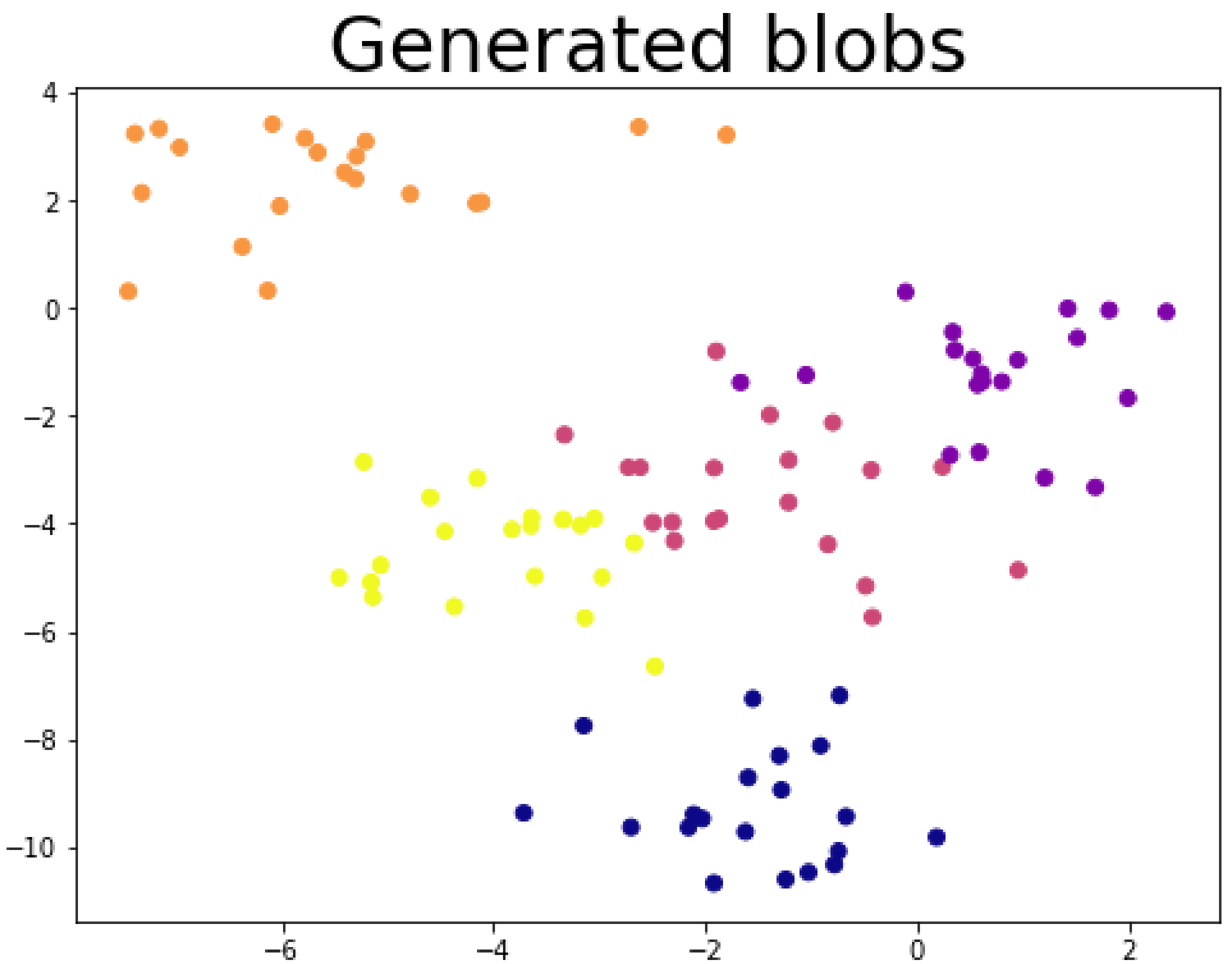
1. **Generate Data using Scikit learn Library**

Scikit-learn has a **datasets** package that helps us generate and fetch data. Such created or fetched data can be used to evaluate our ML models. For generated data, we can control the size of the dataset (n_sample s and n_features) as well as the statistical properties of the data. In part of this exercise, we want to practice generating blobs, circles, and moons datasets.

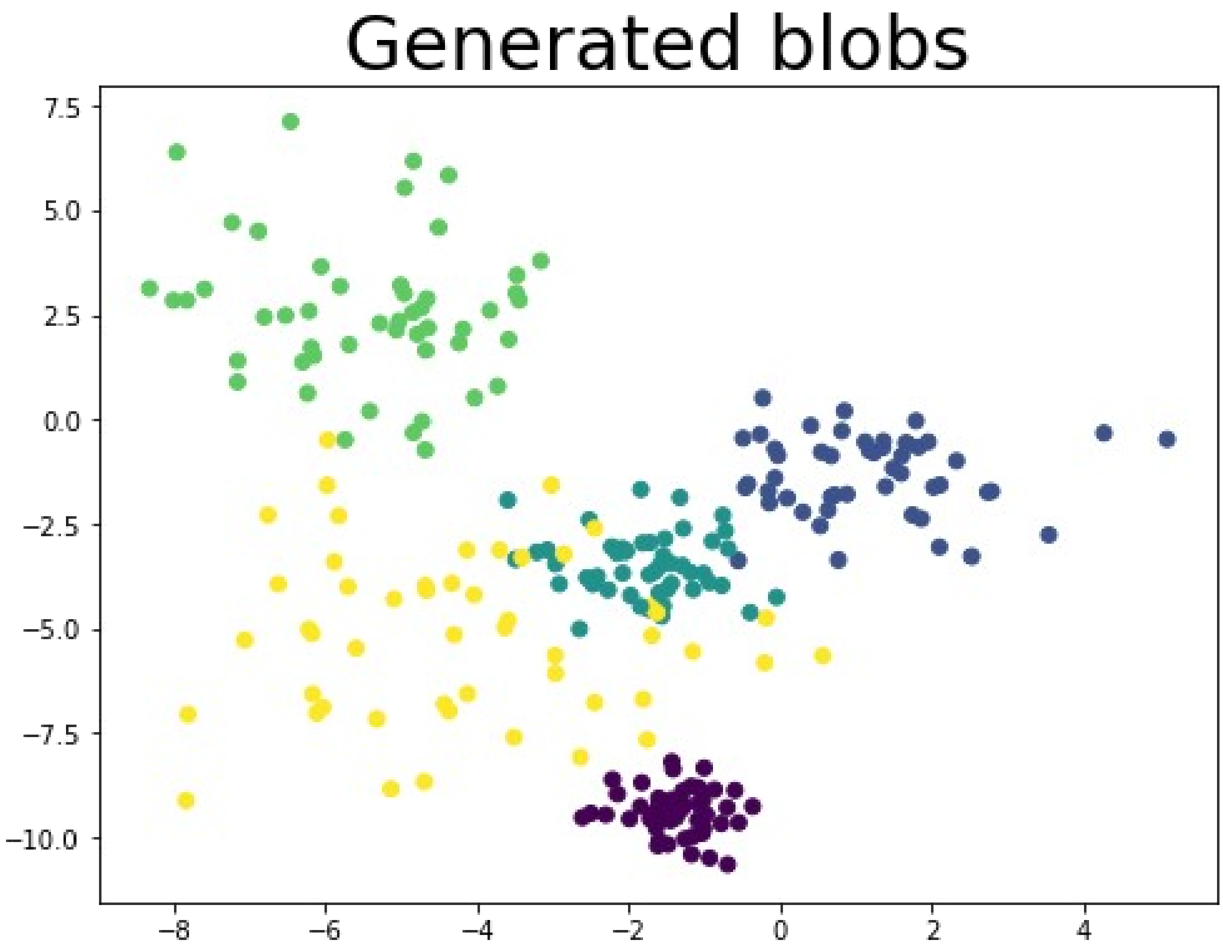
1. Generate a dataset with 5 groups, each group has 20 samples(dataset1).
2. Generate a dataset with 5 groups, each with 50 samples and each group has its own spread.
3. Generate a dataset with two inscribed circles of data points, each data circle has 20 data points.
4. Repeat task 3 and adjust the parameters that force the inner circle to move towards the outer circle.
5. Generate a moon dataset, with 30 samples per group. Moreover, study the noise parameter.
6. Finally, for each generated dataset visualize the results using scatter plot

top

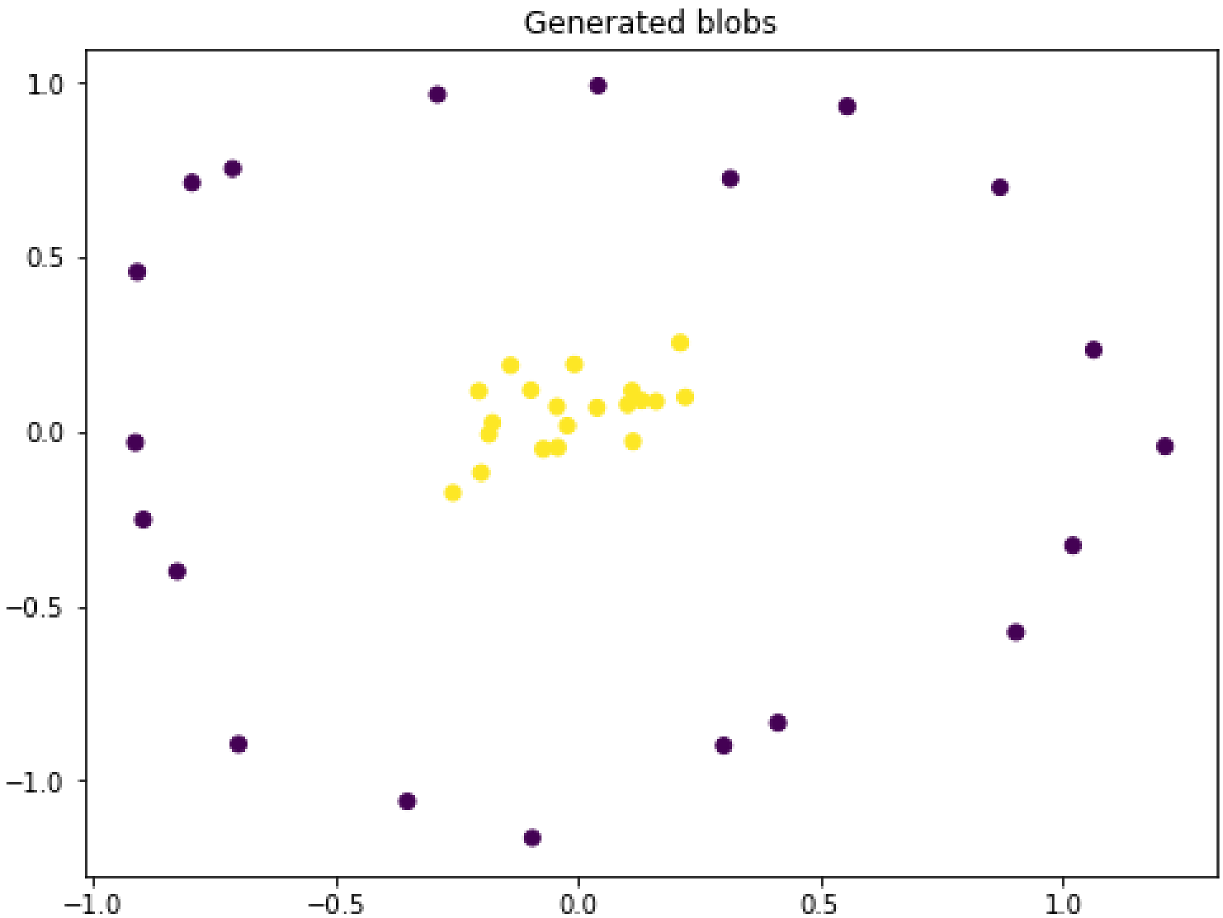
```
# First load the needed packages. (Note: it's already load it in the
top cell)
from sklearn import datasets

# Task 1: Done for you
X, y = datasets.make_blobs(n_samples=100,        # 5 groups with 20
samples each that make the total points =100
                               centers=5,        # determine groups
to be generated
                               n_features=2,     # determines how
many features per sample
                               cluster_std = 1.0, # determines the
spread per each generated group
                               shuffle = True,   # the data retuned
will be shuffled or not
                               random_state=2    # for
reproducibility use the same number
                               )

# scatter plot the data and use labels to color each group's samples
# this line sets the size of the plot
plt.figure(figsize=(8,6))
# this line scatter plot the data, c=y is the color paramter, and y is
the labels
plt.scatter(X[:,0],X[:,1], c = y, cmap='plasma')
#sets the title
plt.title('Generated blobs', fontsize=30)
#you may set the x coordinate and y coordinate labels to X and y

# finally, to show the plot
plt.show()
```



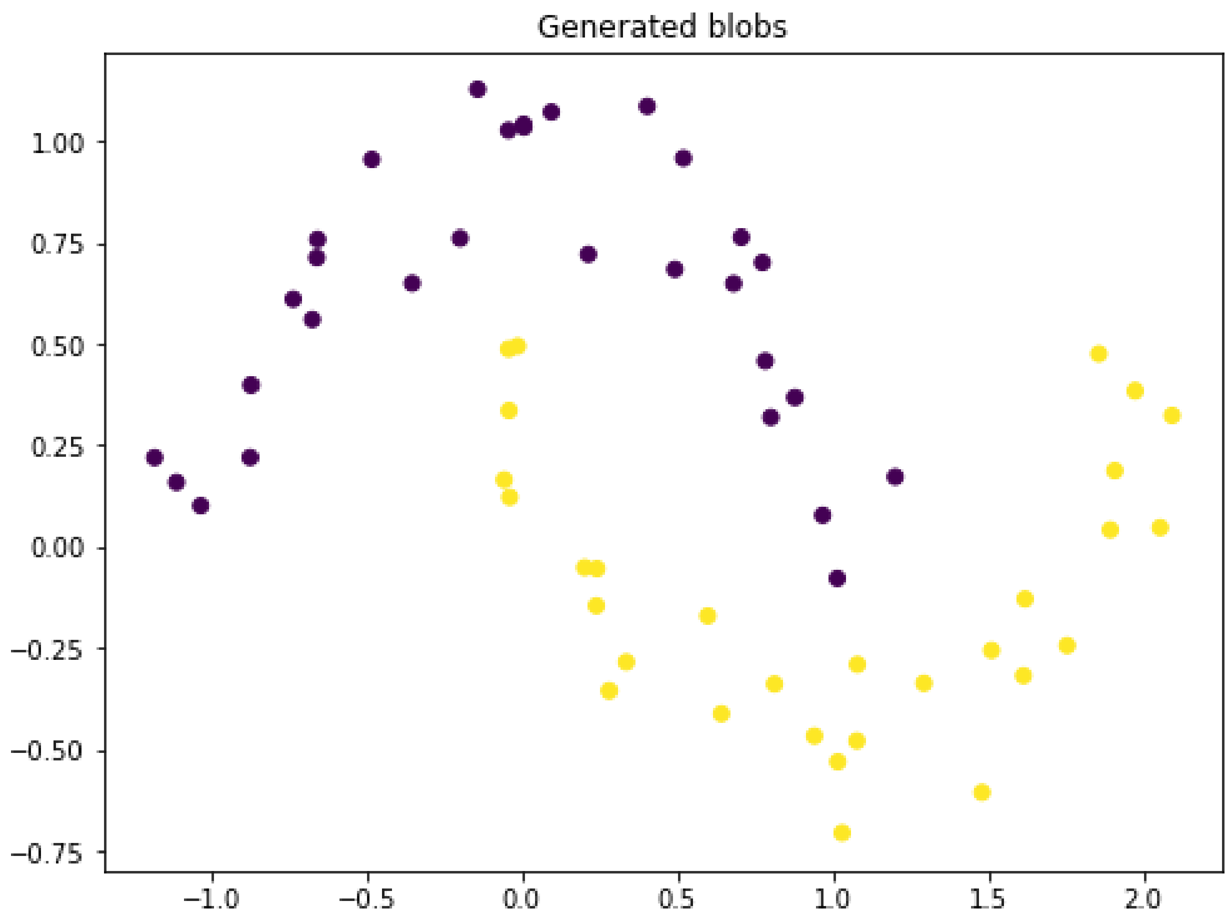

```
## Task2

Xb, yb = datasets.make_blobs(n_samples=   ,
                              centers=   ,
                              n_features= ,
                              cluster_std=[  ],  # to set different spread you need a list
                              shuffle = True,
                              random_state=2
                             )

# plot the the generated data

# write your code here
```




```
# Task 3
Xc, yc = datasets.make_circles(n_samples= ,
                                 noise=   ,   # a value (0 to 1): it
sets how data points disturbed
                                 factor=  ,    # a value btw (0 to 1):
how far the inner data from the outer group
                                 shuffle=True,
                                 random_state=2
                                 )

# plot the the generated data

# write your code here
```




```
# Task 4
Xmn, ymn = datasets.make_moons(n_samples= ,
                               noise=     ,
                               shuffle=True,
                               random_state=2)


# plot the the generated data


# write your code here
```



1. **Load Benchmark dataset from Scikit learn**

In this part we want to load sklearn datasets. Such data are useful to learn and test ML algorithms. Let us load and explore three different datasets.

- Load Iris dataset and find out the following:
  - Total number of samples
  - How many features the dataset has
  - what is the target variable
  - which ML methodology we can apply on this dataset (supervised, unsupervised)
- Load the Diabetes dataset and find out the following:
  - Total number of samples
  - How many features the dataset has
  - what is the target variable
  - which ML methodology we can apply on this dataset (supervised, unsupervised)
- Fetch the news group dataset from sklearn
  - Total number of samples
  - what is the target variable
  - which ML methodology we can apply on this dataset (supervised, unsupervised)

```
# 1. iris dataset:
# complete the code below if needed
# use datasets library to get the iris data
irisdata = datasets.load_iris() # done for you

# extract data only from irisdata
X_iris =

# extract targets values only irisdata
y_iris =

# set information required as dict
datainfo = {'total samples':          ,  # use .shape[0]
            'total features':         ,  # use .shape[1]
            'total target labels':     } # hint: you can use set() to
get unique values

# let us show the results using loop
for key in datainfo:
    print(key,':', datainfo[key])

# list features names from the dataset
i = 1
print('Feature names')
for feat in irisdata.  :  # complete this line with target names key
    print(i, '. ', feat)
    i += 1


# diabetes dataset

# complete the following code
diabetesdata =

# extract data only from diabetesdata
X_diabetes =

# extract targets values only from diabetesdata
y_diabetes =


# set information required as dict
datainfo = {'total samples':                    ,
            'total features':                   ,
            'total target labels':[min(y_diabetes), max(y_diabetes)]
} # hint: get the min and max of the targets

# let us show the results using loop
for key in datainfo:
    print(key,':', datainfo[key])
```

```
# list features names from the dataset
print('Feature names')
i=1
for feat in diabetesdata.   :  # complete this line with feature names
    print(i, '. ', feat)
    i+=1
```

```
total samples : 442
total features : 10
total target labels : [25.0, 346.0]
Feature names
1 .  age
2 .  sex
3 .  bmi
4 .  bp
5 .  s1
6 .  s2
7 .  s3
8 .  s4
9 .  s5
10 .  s6
```

Since the targets are continuous, the problem is **...**

```
# 20newsgroups dataset
# this dataset is text, you may not be able to use it directly, but
some preprocessing/feature extraction are required.
data20news =

# extract data only  from data20news
X_data20news =

# extract targets values only from data20news
y_data20news =


# set information required as dict
datainfo = {'total samples':                      , #the data in a
form of a list, so len will return the number of items
            'total target labels':                }# hint: use set
here

# let us show the results using loop
for key in datainfo:
    print(key,':', datainfo[key])

# list target names from the dataset (topics)
print('Target names')
i=1
for topic in data20news.   :            # complete this line with
```

```
target names
    print(i, '. ', topic)
    i+=1

total samples : 11314
total target labels : {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,
14, 15, 16, 17, 18, 19}
Feature names
1 .  alt.atheism
2 .  comp.graphics
3 .  comp.os.ms-windows.misc
4 .  comp.sys.ibm.pc.hardware
5 .  comp.sys.mac.hardware
6 .  comp.windows.x
7 .  misc.forsale
8 .  rec.autos
9 .  rec.motorcycles
10 .  rec.sport.baseball
11 .  rec.sport.hockey
12 .  sci.crypt
13 .  sci.electronics
14 .  sci.med
15 .  sci.space
16 .  soc.religion.christian
17 .  talk.politics.guns
18 .  talk.politics.mideast
19 .  talk.politics.misc
20 .  talk.religion.misc
```

This data is not ready for modeling, it may need some feature engineering before passing it to machine learning

---

Exercise 2 (Kmeans Clustering)

Let's perform clustering using Kmeans algorithm on the **blob** data generated in the previous exercise.

Steps:

1. Load the Kmeans model from the cluster package
2. Initiate the Kmeans model with required parameters such as K=2 (n_clusters)
3. Fit the model to get results
4. Use a scatter plot to show clustering results by coloring each point with its label from the clustering result (y_hat).

Note: You may also check out n_clusters to 3, 4, and 5 and study the validation results (in case you finished this ex fast).

Return to Table of Contents

```
# First we need to import the kmeans from cluster package
from sklearn.cluster  import KMeans

# initiate the KMeans model with k = 2 clusters,
k_means = KMeans(  #write paramters and value here   )  # note
n_clusters parameter is the K

# fit the configure model using the previous data X
k_means.fit(  ) # fill in between brackets

# Get the estimated label (y_hat) of each point
y_hat = k_means.predict(  ) # fill in between brackets

# 2.Use scatter plot to show clustering results labeled by y_pred.

# write your code here
```

Kmeans identified two clusters as configured. The first is the top yellow cluster and another at the bottom of the plot above

Return to Question List of Tasks

Exercise 3 (Evaluation External Index)

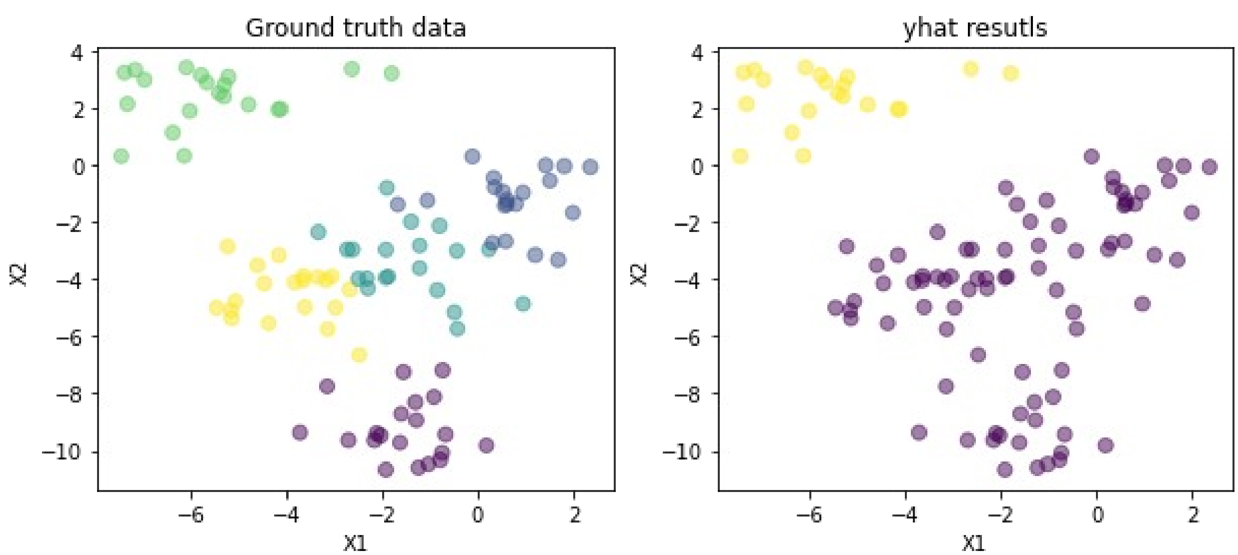
The blob dataset used in Exercise 2 was generated using Sklearn. This dataset has labels already which we can use as a ground truth. In this exercise, we want to validate our clustering results using an external index metric (homogeneity score). If the results are not satisfactory, we can repeat the clustering and improve.

Return to Table of Contents

```
# we need to load the metric first
from sklearn.metrics import homogeneity_score

# We want to find out how homogenious our results compared to ground-
truth

score =                   # compute homogeneity score here

print('Homogenity of the clustering results is (%0.2f)'% score)
```

```
Homogenity of the clustering results is (0.31)
```

```
# To be clear, we can plot data and color them using the new labels


# write your code here to show two subplots
```



The above results showed 31% homogeneity between the ground-truth and clustering results. How can we make it better?

```
# improve the results above 31% can be improved further using 3
clusters
```

```
# write your code here
# plot results

# improve the results above 31% can be improved further using 4
clusters
# write your code here
# plot results

# improve the results above 31% can be improved further using 5
clusters
# write your code here
# plot results
```

Return to Question List of Tasks

Exercise 4 (Evaluation: Internal Index)

We can look also at the internal index for the above clustering results as an exercise. To do that, we can compute the WSS and BSS and compare them. If BSS is greater than WSS, then that is a good indication of good clustering

1. Within the Sum of Squares (WSS) also called inertia, what do you think about the computed value?
2. Between Sum of Squares (BSS), What is the relation between WSS and BSS?
3. Silhouette index

Return to Table of Contents

```
# 1 and 2 wss and bss

# borrow the code from the previous exercise with K = 2
k_means = KMeans(     )  # set k = 2
k_means.fit(   ) # fit the model
y_hat = k_means.predict(  )  # predict the new labels


# to get wss, we need to reach to inertia_ attribute of the model
wss = k_means.



# The BSS is not provided and we can code the equation to compute it
# BSS = Sum(|Ci| * (M - mi)^2 )
#where Ci is the cordinality of cluster i and mi is the mean of
cluster i and M is the mean of the whole data

# let me walk you through
```

```
# stp1: get the mean of the data
M = 

# stp2: let's build a loop to compute bss
bss = 0

# for each label in the prediction
for lbl in set(y_hat):

    # get size of that group lbl and save it in c
    c = len(    )

    # compute mean of that group lbl of the data and save it in m
    m = np.mean(    )

    # apply in bss equation, the += will accumulate results of each 
group
    bss += c * ((M-m)**2 ).sum()


print('WSS=%0.2f'% wss)
print('BSS=%0.2f'% bss)

WSS=1122.18
BSS=1043.92
```

Notes:

1. If your results are not the same that may be due to your generated data is different, or the results produced by the Kmeans is different due to Kmeans initialization

2. The above performance indicate good clustering results?? however, it doesn't look good with homogeneity score!

```
# 3. Silhouette index
# Silhouette score using scikit learn
from sklearn.metrics import silhouette_score

sil =                          # compelte this line to compute silhouette
print ('silhouettee score =%0.2f '% sil)

silhouettee score =0.51
```

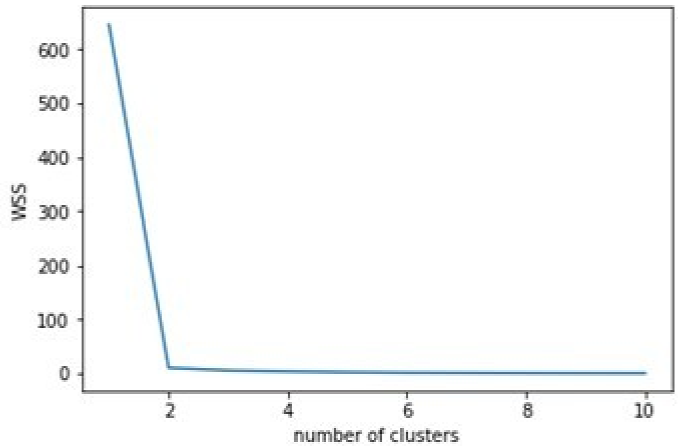
Conclusion

write your observations

Return to Question List of Tasks

Exercise 5 (Estimate number of clusters)

The Elbow method is used to estimate the right number of clusters for a given dataset. The turning point that forms an elbow shape is usually used as a good value for K in K-means Clustering. The following Figure shows a curve with an elbow shape.

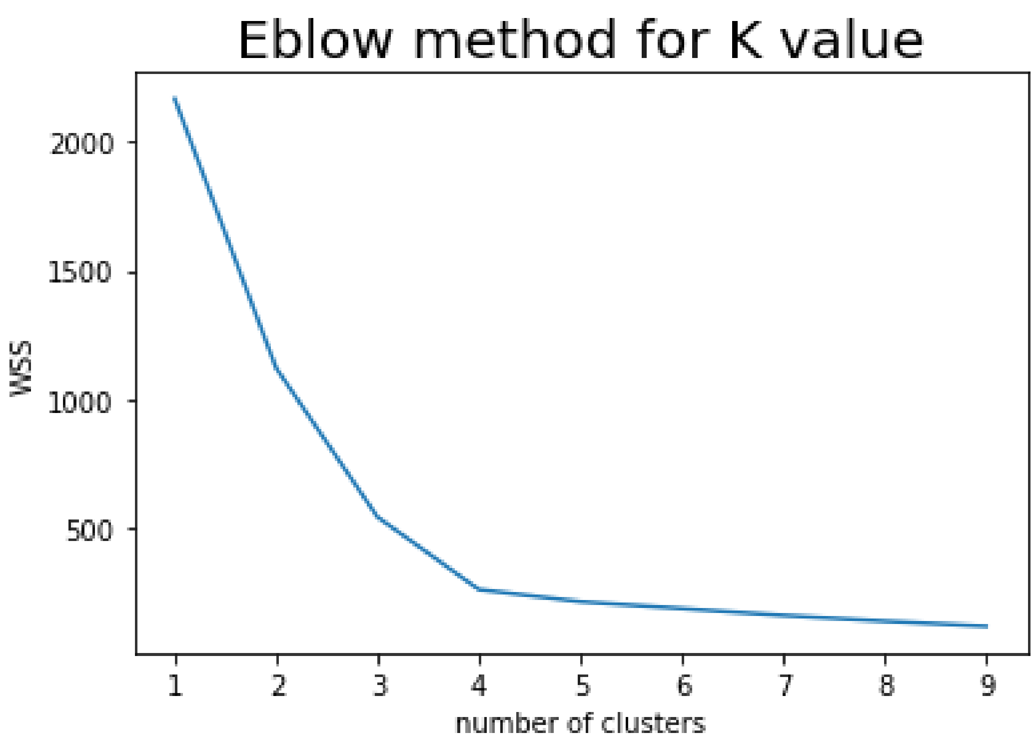


Given the previously used data, write a python code that performs K-means clustering repeatedly on the data using different K values each time. Then, plot K values against computed WSS (inertia) in each iteration.

Return to Table of Contents

```
# define a function to find best K
# input: data X
# output list of wss and different value for K
# hint: kmeans has an output attribute as inertia_
def Find_K_Kmeans(X, ks):

    # should go from 1 cluster to n number of clusters where WSS will
be zero
    WSS = []
    for k in range(1,  ): # complete code here

        # initiat K-means
        kmeans = KMeans(n_clusters=  ) # complete code here

        kmeans.fit( ) # complete code here

        d      =          # complete code here
        WSS.append( d )


    return WSS # this will return a list of wss
```

```
# Call the developed function

# how many ks you want to test? set klst
klst =     # complete code here

# call the function pass the data and klst
wss = Find_K_Kmeans(   ) # complete code here

# plot the wss in (y-axis) and klst(x=axis) and figureout what is the
optimal k?

# write your ploting code here
```

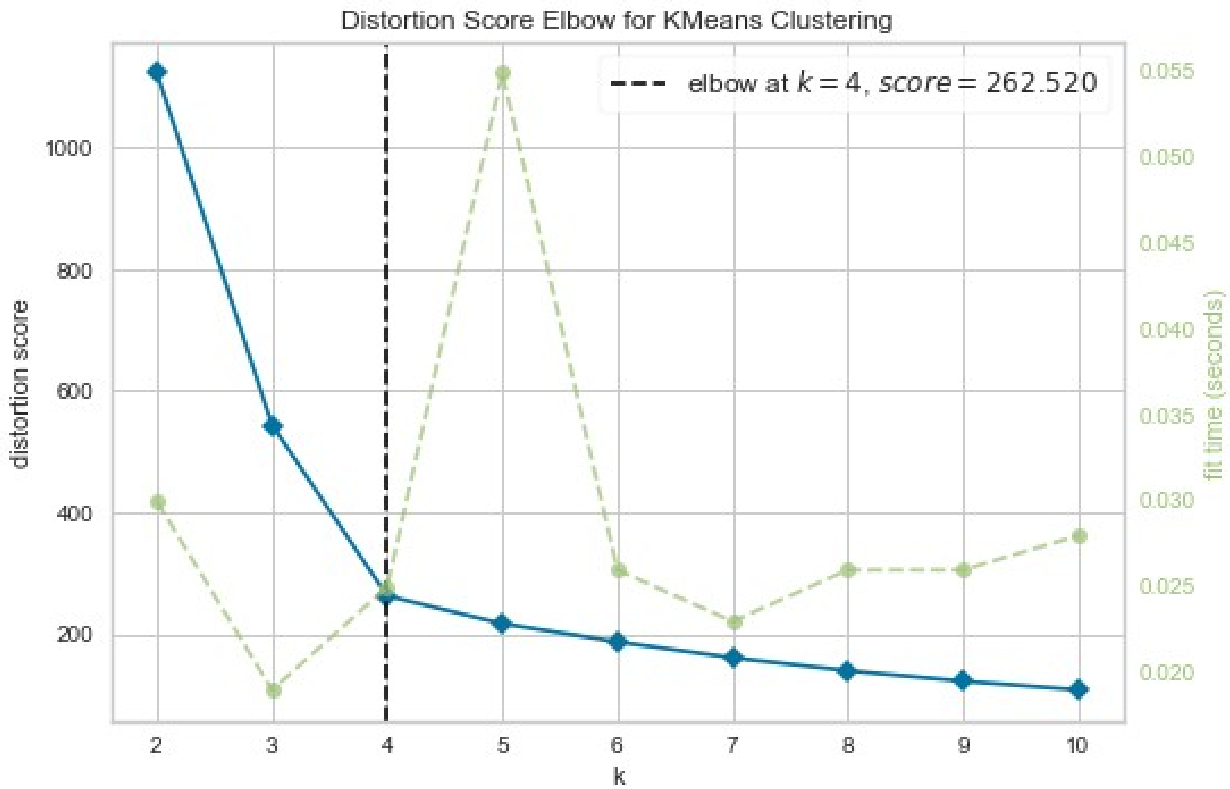


Conclusion: write your observation and conclusion

Note: we can use yellowbrick library to visualize the elbow method. This library is not installed by default in local machines, so you can install it as follows:

- conda install -c districtdatalabs yellowbrick
- for documentation: https://www.scikit-yb.org/en/latest/api/cluster/elbow.html
- Unfortunately, if you have the latest version of Sklearn, the yellowbrick library won't work properly due to a bug. In this case, you can install an older version 0.22 (conda install scikit-learn=0.22) **not recommended**

```
# import sklearn
# if sklearn.__version__[0] == '0':
#     print('yellowbrick may work correctly.. ')
#     print('if you are in Google colab, it most likely that
yellowbrick is install and ready')
#     print('first import it to check if it is exist in your system')
# else:
#     print('No need to install yellowbrick, use matplot to visualize
the elbow method')
#     print('version 1.0.2 was debuged and should work fine')

# In case you don't have yellowbrick installed, uncomment the below
line of code
#!pip install yellowbrick

# This  tool is to show the elbow method and determine the K value
# In case it doesn't work with you, it is maybe necessary to downgrade
sklearn to 0.22 (not recommended)

# load the visualizer from cluster package
from yellowbrick.cluster import KElbowVisualizer

# call the visulizaer with KMeans class and set the max K to select an
optimal value for k
Elbow_M = KElbowVisualizer(KMeans(), k=10)

# Fit Elbow_M model
Elbow_M.fit(X)

# call show method from Elbow_M
Elbow_M.show()

# end with plt.show
plt.show()
```



Exercise 6 (Determine value of K using Silhouette score)

As WSS is already considered in silhouette score and our target is to have the highest score, we can optimize silhouette metric and find K that maximum the score.

We can repeat our work above to find the best K using silhouette score

1. develop a function that builds several kmeans models and return silhouette scores of each model
2. each model is created using different K value
3. select the best K

Note Use the same dataset

Return to Table of Contents

```
# develop a  function
# you can copy the previous function and make necessary changes to get
silhouette scores
```

```
# call the function

# plot the results
# write your ploting code here
```

write your conclusions

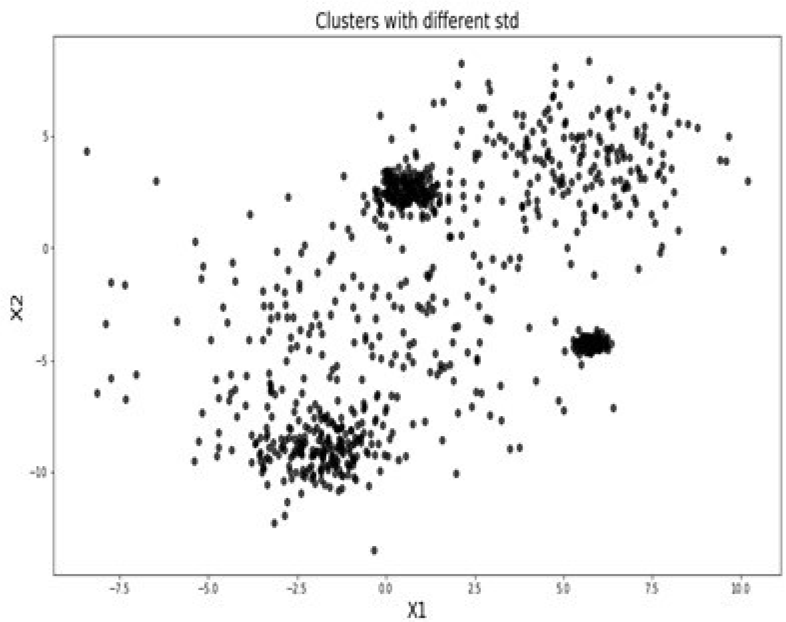
Exercise 7 (Kmeans Ability)

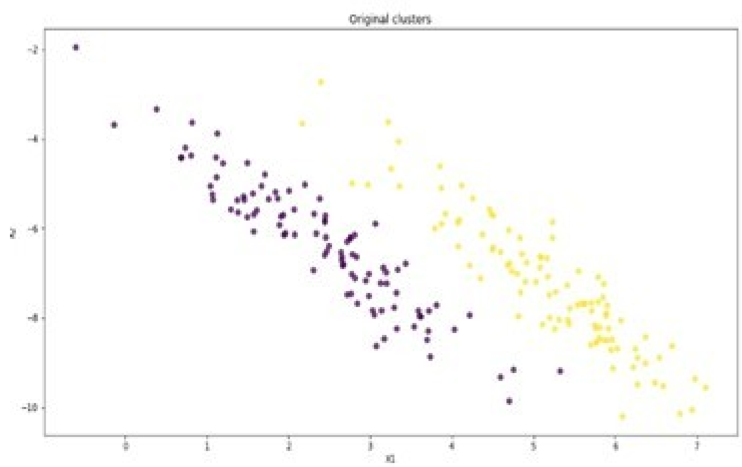
In the previous exercises, we performed kmeans clustering on blob dataset with almost similar densities. In this exercise, we want to figure out the ability of the Kmeans method in the following situations:

1. How good Kmeans is in clustering data with various densities or skewed data
2. How good Kmeans in clustering circles dataset
3. How good Kmeans in clustering moons datasets

Consider the following datasets of two variables X1, and X2:

1. **Challenge 1 (variations):** Create a dataset with 5 clusters, and each cluster has different standard deviation; std = {1, 0.2 ,3 ,0.5, 2}. Then, use scatter plot to show the data
2. **Challenge 2 (stretched datasets):** Generate another dataset with 2 features and 2 clusters. Use the anisotropic transformation (Note: copy the code from the slides) to make the two clusters stretched. Then, repeat clustering using Kmeans. Observe the results.
3. **Challenge 3 (circles and moons datasets):** Create circles and moons datasets. Then, use silhouette method to determine a value for K. After that, perform clustering using identified **K** to predict labels using Kmeans. Observe the results.

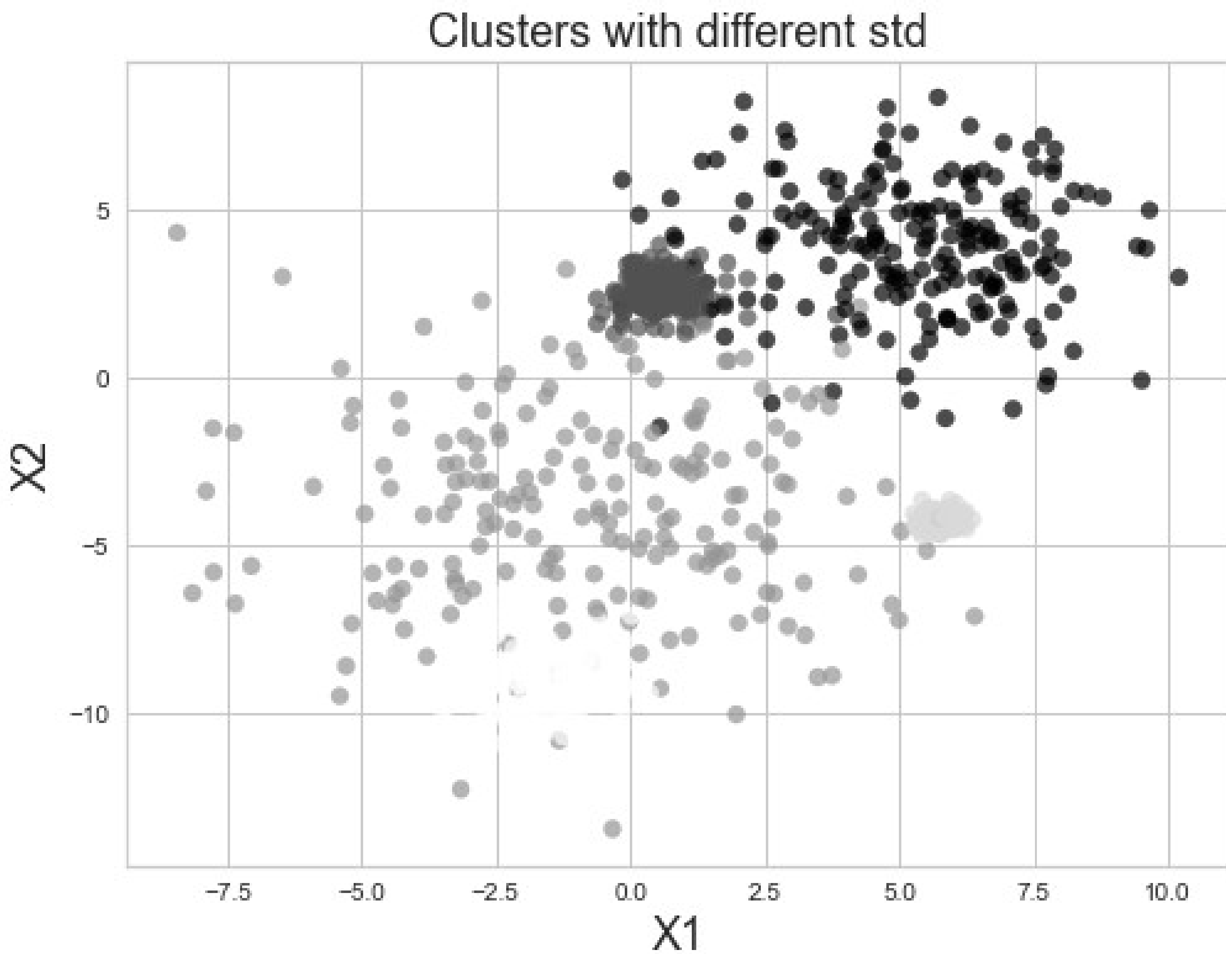




Return to Table of Contents

challenge 1(variations):

```
# let us generate the data with various densities and viz it

# using blobs 1000 samples with 5 groups different densities
Xbd, ybd = datasets.make_blobs(n_samples = 1000, n_features=2,
centers=5, cluster_std=[1,0.2,3,0.5,2] , random_state=40)

# show the data

# write your ploting code here
```




```
# Build kmeans model with an optimal K and compute its silhouette
score
# note: make use of the previous develop functions to find the best
model (best k)

# start coding here



# once you know the best model, follow these steps

# instantiate fresh KMeans with winning k value
kmeans =

# fit kmeans to the data
# write your code here

# label the data points
y_hat = kmeans.
```

```
# compute the silhouette score of the model
# write your code here

# scatter plot results with colors to indicate formed clusters

# write your ploting code here
```

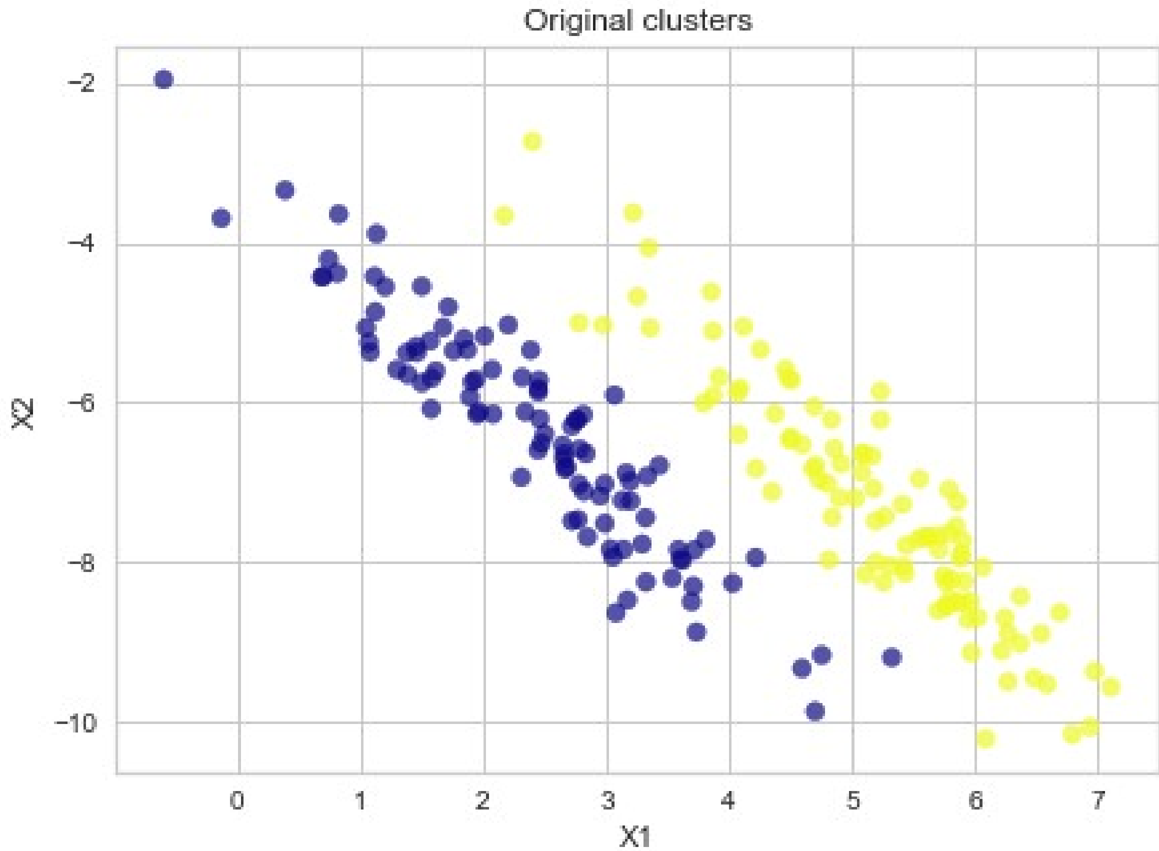
Concluions

write you comments and observation

Challenge 2 (stretched dataset)

Return to Question 8

```
# Let us generate streched data
# normal blob data
Xbs, ybs = datasets.make_blobs (n_samples=200, centers =2,
n_features=2, cluster_std=1.5, random_state=40)

# transform parameters
transformation = [[0.60, -0.63], [-0.40, 0.85]]

# Dot product the generated data points to have the new stretched
dataset
Xbs = np.dot(Xbs, transformation)

#plot results cmap=plasma and alpha =0.7

# write your ploting code here
```




```
# Build kmeans model with an optimal K and compute its silhouette
score
# note: make use of the previous develop functions to find the best
model (best k)

# start coding here



# once you know the best model, follow these steps

# instantiate fresh KMeans with winning k value
kmeans =

# fit kmeans to the data
# write your code here

# label the data points
y_hat = kmeans.

# compute the silhouette score of the model
# write your code here
```

```
# scatter plot results with colors to indicate formed clusters

# write your ploting code here
```

Concluions

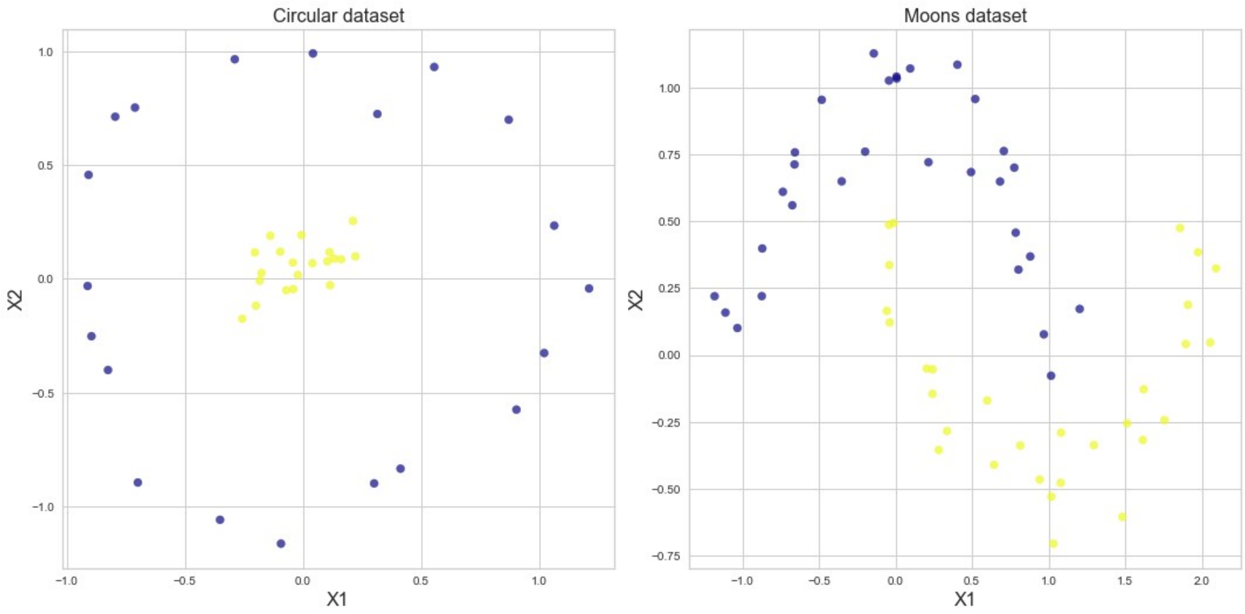
write you comments and observation

what do you think about the results??

Challenge 3 (Circles and Moons Dataset)

Return to Question 8

```
# Data from Exercise 1
#plot results
plt.figure(figsize=[10,5])

# left plot
plt.subplot(1,2,1)
plt.scatter (Xc[:,0],Xc[:,1], c= yc, alpha = 0.7, cmap='plasma' )
plt.xlabel('X1', fontsize=16)
plt.ylabel('X2', fontsize=16)
plt.title('Circular dataset', fontsize=16)

# right plot
plt.subplot(1,2,2)
plt.scatter (Xmn[:,0],Xmn[:,1], c= ymn, alpha = 0.7, cmap='plasma' )
plt.xlabel('X1', fontsize=16)
plt.ylabel('X2', fontsize=16)
plt.title('Moons dataset', fontsize=16)


plt.tight_layout(rect=(0,0,1.5, 1.5))
plt.show()
```




```python
# Build kmeans model with an optimal K and compute its silhouette score
# note: make use of the previous develop functions to find the best model (best k)

# start coding here



# once you know the best model, follow these steps

# instantiate fresh KMeans with winning k value
kmeans =

# fit kmeans to the data
# write your code here

# label the data points
y_hat = kmeans.

# compute the silhouette score of the model
# write your code here

# scatter plot results with colors to indicate formed clusters

# write your ploting code here
```

Concluions

write you comments and observation

what do you think about the results??

Exercise 8 (Dendrograms)

Using the following data of two features x1and x2, study the behavior of linkages methods (single, complete, average, ward) by plotting the dendrograms of each method's result.

Points	x1	x2
P1	0.5	0.5
P2	0.5	2
P3	5	5
P4	3	4
P5	4	4
P6	3	3.5

Tasks:

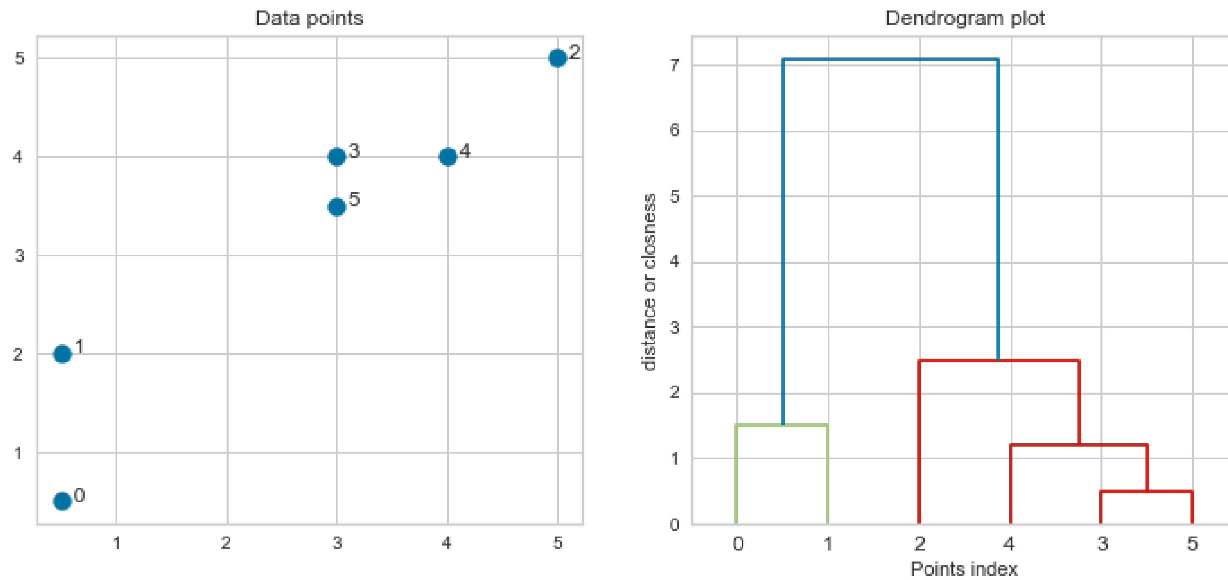
1. Order the points as they got grouped early? Use the linkage matrix |Points| Order| |:---|:---| |P1| | |P2| | |P3| | |P4| | |P5| | |P6| |

2. In case of three clusters, which points are grouped together? |Cluster| Points| |:---|:---| |Cluster 1| (.........) | |Cluster 2 | (.........) | |Cluster 3| (.........) |

Note: load both packages linkage and dendrogram from **scipy.cluster.hierarchy**

```
# load the linkage
from scipy.cluster.hierarchy import dendrogram, linkage

# make numpy array with the data from the question description
data    = np.array([[0.5,0.5], [0.5,2], [5,5], [3,4], [4,4], [3,
3.5]])

# Use linkage function and make sure 'ward' is used
linkage_matrix =     # complete the code here

# Plot the dedrogram using the linkage_matrix
plt.figure(figsize=(12,5))

# left part - data points
plt.subplot(1,2,1)
plt.scatter(data[:,0], data[:,1], s=100)
plt.title('Data points')
i = 0
for d in data:
    plt.annotate(' '*2 + str(i), (d[0], d[1]) )
    i+=1
```

```
# right plot - dendrogram
plt.subplot(1,2,2)
dendrogram(                    ) # complete the code here
plt.title('Dendrogram plot')
plt.xlabel('Points index')
plt.ylabel('distance or closness')
plt.show()
```




```
# either from the dendrogram plot or the linkage matrix we can study
clsuters

# convert them to dictionary using dict(enumerate () )
datadic = dict(enumerate(data.tolist()))

# List the coordinate and its serial number
for key, val in datadic.items():
    print(key, val)


print() # new line


# let us print the linkage matrix, you can find answers now
print ('linkage matrix:\n',    ) # complete the code here
```

```
0 [0.5, 0.5]
1 [0.5, 2.0]
2 [5.0, 5.0]
3 [3.0, 4.0]
4 [4.0, 4.0]
5 [3.0, 3.5]
```

```
linkage matrix:
[[5.         3.         0.5        2.        ]
 [6.         4.         1.19023807 3.        ]
 [0.         1.         1.5        2.        ]
 [7.         2.         2.49165273 4.        ]
 [8.         9.         7.08578389 6.        ]]
```

Answer:

1. Order the points as they got grouped early? Use the linkage matrix |Points| Order| |:---|:---| |P1| 3 | |P2| 3 | |P3| 4 | |P4| 1 | |P5| 2 | |P6| 1 |

2. In case of three clusters, which points are grouped together? From the plot below using **complete, centroid, average linkages**, we have the following 3 clusters | Cluster| Points| |:---|:---| |Cluster 1| (1, 2) | |Cluster 2 | (3) | |Cluster 3| (4, 5, 6) |

3. In case of three clusters, which points are grouped together? From the plot below using **single linkages**, we have no 3 clusters !!

```
# Repeat the above exercise using single link instead of ward!
# is there any differences

# write your code here




# write your plotting code here
```

Note you may study the effect of different linkages

Exercise 9 (Perform Hierarchical Clustering)

Let us perform Agglomerative Clustering on our previous blob dataset X,y.

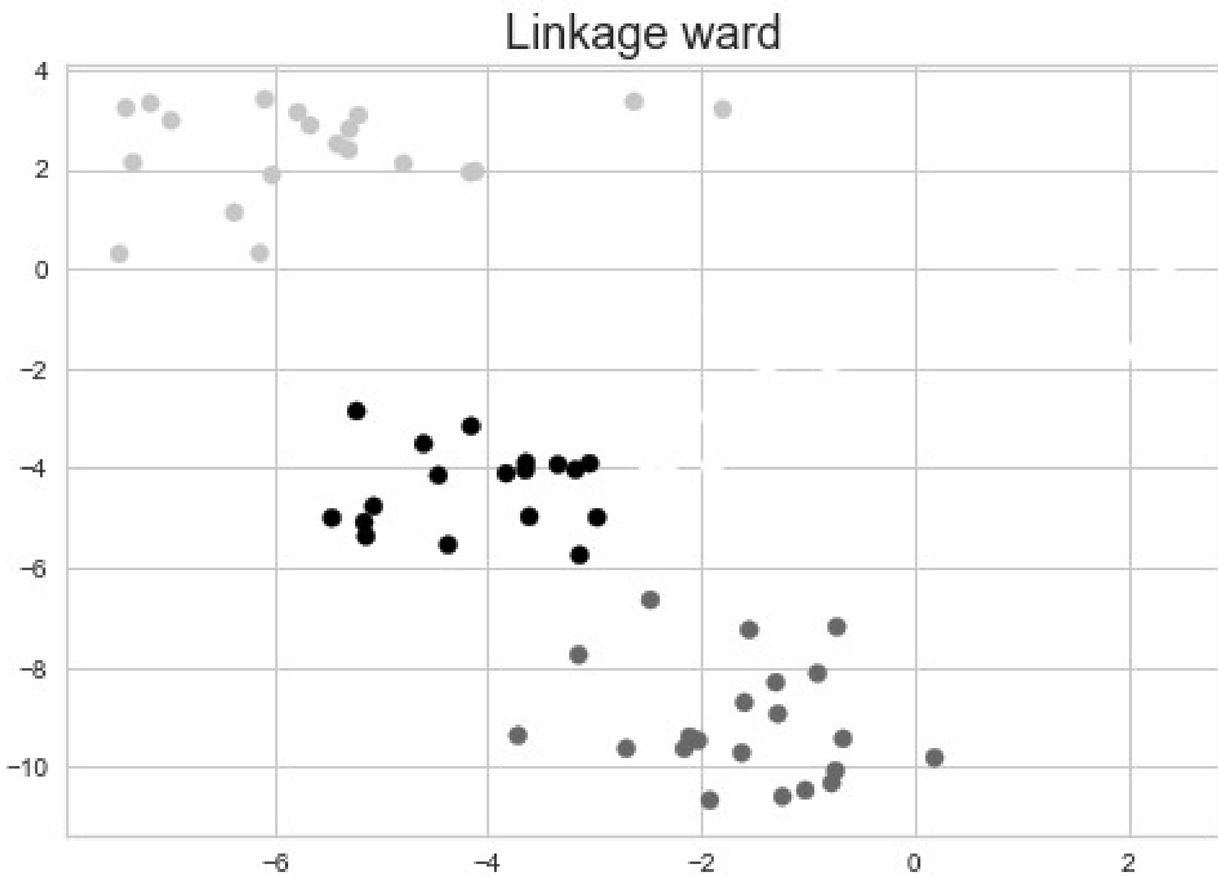
1. we can use denderogram to figure out how many clusters or what is K
2. Then, perform hierarchical clustering using different linkages (ward, complete, average)

```
# let us try AC with 100 point dataset X, y

# use 'ward'linkage to build the dendrogram
linkage_array =  linkage() # complete the code here
dendrogram(   )            # complete the code here

# you can use plt.axhline to determine your cut decision on the plot
```

```
#as this example
plt.axhline(20, color='k', linestyle='--') # you may change the value 20 to yours



# larger and tigher plot
plt.tight_layout(rect=(0,0,1.5,1.5))
plt.show()
```

From the plot, our cut can be defined on either **xxx** or **yyy**. In case of **xxx**, we end-up with **xxx1** clusters so K = **xxx1** In case of cut on **yyy**, we end-up with **yyy1** clusters so K = **yyy1**

```
# As we determine the K value for the AC clustering, let's perform clustering

# load the Agglomerative clustering
from sklearn.cluster import AgglomerativeClustering

# instantiate the AC clustering
ward = AgglomerativeClustering() # complete this code with K value and linkage to be used

ward.fit( ) # pass the data here


# label the data
y_hat = ward.fit_predict( ) # label the data

# plot the data with the predicted labels using Hierarchical Clustering
plt.scatter(X[:,0], X[:,1], c = y_hat, cmap='plasma')
plt.title("Linkage ward", fontsize=18)
plt.show()
```



Compare with KMeans results before !!

Exercise 10 (Hierarchical Clustering Ability)

In this exercise, we want to check the performance of AC clustering on the challenges set as before.
Consider the following datasets of two variables X1, and X2:

1. Challenge 1 (variations): use AC to cluster 5 groups using the previous dataset with 5 clusters and different standard different spreads. Then, evaluate the results using the silhouette score and visualization.
2. Challenge 2 (stretched datasets): use AC on the previous stretched dataset to group them into 2 groups. Then, evaluate the results using the silhouette score and visualization.
3. Challenge 3 (circles and moons datasets): use AC to cluster circles and moons datasets into two groups. Then, evaluate the results using the silhouette score and visualization.

```
# challenge 1

# start coding here
```

```
# challenge 2

# start coding here

# challenge 3

# start coding here
```

Exercise 11 (DBSCAN)

In this exercise, we want to check the performance of DBSCAN clustering on the challenges set before for AC and Kmeans.
Consider the following datasets of two variables X1, and X2:

1. Challenge 1 (variations): use DBSCAN to cluster 5 groups using the previous dataset with 5 clusters and different standard different spreads. Then, evaluate the results using the silhouette score and visualization.
2. Challenge 2 (stretched datasets): use DBSCAN on the previous stretched dataset to group them into 2 groups. Then, evaluate the results using the silhouette score and visualization.
3. Challenge 3 (circles and moons datasets): use DBSCAN to cluster circles and moons datasets into to groups. Then, evaluate the results using the silhouette score and visualization.

```
# challenge 1

# start coding here

# challenge 2

# start coding here

# challenge 3

# start coding here
```

Example using real data: study this as Exercise 12 (Optional real data)

Customer Personality Analysis including customer segmentation is an important practice a company wants to perform to improve future sales. This analysis helps in separating customers into groups that reflect customers with a common interest in each cluster. Moreover, it helps sales managers to modify products according to the distinct needs and behaviors of different types of customers. Kaggle maintains a marketing campaign dataset which we can use in this exercise to identify customer segmentation using clustering algorithms. The dataset may require further preprocessing and preparation to use all features available, however, for this exercise, we will select only those attributes that require nearly no or minimal preprocessing and can be used with Kmeans.

https://www.kaggle.com/code/karnikakapoor/customer-segmentation-clustering/data

A good notebook to this exercise can be accessed from:

https://www.kaggle.com/code/karnikakapoor/customer-segmentation-clustering/notebook

```
# You can download the data from Kaggle, the link given above:
# load the data to pandas Dataframe
marketdata  = pd.read_csv('marketing_campaign.csv',  sep="\t")

# let's do minimal preprocessing
#1- check the dataset information
#2- remove all features that are not numerical
#3- check missing values, if many then remove those features

# check info
marketdata.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2240 entries, 0 to 2239
Data columns (total 29 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   ID                   2240 non-null   int64
 1   Year_Birth           2240 non-null   int64
 2   Education            2240 non-null   object
 3   Marital_Status       2240 non-null   object
 4   Income               2216 non-null   float64
 5   Kidhome              2240 non-null   int64
 6   Teenhome             2240 non-null   int64
 7   Dt_Customer          2240 non-null   object
 8   Recency              2240 non-null   int64
 9   MntWines             2240 non-null   int64
 10  MntFruits            2240 non-null   int64
 11  MntMeatProducts      2240 non-null   int64
 12  MntFishProducts      2240 non-null   int64
 13  MntSweetProducts     2240 non-null   int64
 14  MntGoldProds         2240 non-null   int64
 15  NumDealsPurchases    2240 non-null   int64
 16  NumWebPurchases      2240 non-null   int64
 17  NumCatalogPurchases  2240 non-null   int64
 18  NumStorePurchases    2240 non-null   int64
 19  NumWebVisitsMonth    2240 non-null   int64
```

```
 20  AcceptedCmp3         2240 non-null   int64
 21  AcceptedCmp4         2240 non-null   int64
 22  AcceptedCmp5         2240 non-null   int64
 23  AcceptedCmp1         2240 non-null   int64
 24  AcceptedCmp2         2240 non-null   int64
 25  Complain             2240 non-null   int64
 26  Z_CostContact        2240 non-null   int64
 27  Z_Revenue            2240 non-null   int64
 28  Response             2240 non-null   int64
dtypes: float64(1), int64(25), object(3)
memory usage: 507.6+ KB

# data basic stats
marketdata.describe()

                 ID   Year_Birth         Income      Kidhome
Teenhome  \
count   2240.000000  2240.000000    2216.000000  2240.000000
2240.000000
mean    5592.159821  1968.805804   52247.251354     0.444196
0.506250
std     3246.662198    11.984069   25173.076661     0.538398
0.544538
min        0.000000  1893.000000    1730.000000     0.000000
0.000000
25%     2828.250000  1959.000000   35303.000000     0.000000
0.000000
50%     5458.500000  1970.000000   51381.500000     0.000000
0.000000
75%     8427.750000  1977.000000   68522.000000     1.000000
1.000000
max    11191.000000  1996.000000  666666.000000     2.000000
2.000000

           Recency     MntWines    MntFruits  MntMeatProducts  \
count  2240.000000  2240.000000  2240.000000      2240.000000
mean     49.109375   303.935714    26.302232       166.950000
std      28.962453   336.597393    39.773434       225.715373
min       0.000000     0.000000     0.000000         0.000000
25%      24.000000    23.750000     1.000000        16.000000
50%      49.000000   173.500000     8.000000        67.000000
75%      74.000000   504.250000    33.000000       232.000000
max      99.000000  1493.000000   199.000000      1725.000000

       MntFishProducts  ...  NumWebVisitsMonth  AcceptedCmp3
AcceptedCmp4  \
count      2240.000000  ...        2240.000000   2240.000000
2240.000000
mean         37.525446  ...           5.316518      0.072768
0.074554
```

```
std          54.628979  ...             2.426645      0.259813
0.262728
min           0.000000  ...             0.000000      0.000000
0.000000
25%           3.000000  ...             3.000000      0.000000
0.000000
50%          12.000000  ...             6.000000      0.000000
0.000000
75%          50.000000  ...             7.000000      0.000000
0.000000
max         259.000000  ...            20.000000      1.000000
1.000000

       AcceptedCmp5  AcceptedCmp1  AcceptedCmp2     Complain
Z_CostContact  \
count   2240.000000   2240.000000   2240.000000  2240.000000
2240.0
mean       0.072768      0.064286      0.013393     0.009375
3.0
std        0.259813      0.245316      0.114976     0.096391
0.0
min        0.000000      0.000000      0.000000     0.000000
3.0
25%        0.000000      0.000000      0.000000     0.000000
3.0
50%        0.000000      0.000000      0.000000     0.000000
3.0
75%        0.000000      0.000000      0.000000     0.000000
3.0
max        1.000000      1.000000      1.000000     1.000000
3.0

       Z_Revenue     Response
count     2240.0  2240.000000
mean        11.0     0.149107
std          0.0     0.356274
min         11.0     0.000000
25%         11.0     0.000000
50%         11.0     0.000000
75%         11.0     0.000000
max         11.0     1.000000

[8 rows x 26 columns]

# let us remove IDs Education, and Marital_Status. Also remove those
records without income
marketdata_smaller = marketdata.drop(['ID', 'Education',
'Marital_Status'], axis=1)
```

the data removed above are important to the problem, but for simplicity we removed them here

```
# drop na rows
marketdata_smaller.dropna(subset = ["Income"], inplace=True)

# let us check how many samples we have now
marketdata_smaller.describe()
```

```
        Year_Birth         Income      Kidhome     Teenhome
Recency  \
count  2216.000000    2216.000000  2216.000000  2216.000000
2216.000000
mean   1968.820397   52247.251354     0.441787     0.505415
49.012635
std      11.985554   25173.076661     0.536896     0.544181
28.948352
min    1893.000000    1730.000000     0.000000     0.000000
0.000000
25%    1959.000000   35303.000000     0.000000     0.000000
24.000000
50%    1970.000000   51381.500000     0.000000     0.000000
49.000000
75%    1977.000000   68522.000000     1.000000     1.000000
74.000000
max    1996.000000  666666.000000     2.000000     2.000000
99.000000

          MntWines    MntFruits  MntMeatProducts  MntFishProducts  \
count  2216.000000  2216.000000      2216.000000      2216.000000
mean    305.091606    26.356047       166.995939        37.637635
std     337.327920    39.793917       224.283273        54.752082
min       0.000000     0.000000         0.000000         0.000000
25%      24.000000     2.000000        16.000000         3.000000
50%     174.500000     8.000000        68.000000        12.000000
75%     505.000000    33.000000       232.250000        50.000000
max    1493.000000   199.000000      1725.000000       259.000000

       MntSweetProducts  ...  NumWebVisitsMonth  AcceptedCmp3
AcceptedCmp4  \
count       2216.000000  ...        2216.000000   2216.000000
2216.000000
mean          27.028881  ...           5.319043      0.073556
0.074007
std           41.072046  ...           2.425359      0.261106
0.261842
min            0.000000  ...           0.000000      0.000000
0.000000
25%            1.000000  ...           3.000000      0.000000
0.000000
```

```
50%              8.000000  ...          6.000000      0.000000
0.000000
75%             33.000000  ...          7.000000      0.000000
0.000000
max            262.000000  ...         20.000000      1.000000
1.000000

       AcceptedCmp5  AcceptedCmp1  AcceptedCmp2     Complain
Z_CostContact  \
count   2216.000000   2216.000000   2216.000000  2216.000000
2216.0
mean       0.073105      0.064079      0.013538     0.009477
3.0
std        0.260367      0.244950      0.115588     0.096907
0.0
min        0.000000      0.000000      0.000000     0.000000
3.0
25%        0.000000      0.000000      0.000000     0.000000
3.0
50%        0.000000      0.000000      0.000000     0.000000
3.0
75%        0.000000      0.000000      0.000000     0.000000
3.0
max        1.000000      1.000000      1.000000     1.000000
3.0

       Z_Revenue     Response
count     2216.0  2216.000000
mean        11.0     0.150271
std          0.0     0.357417
min         11.0     0.000000
25%         11.0     0.000000
50%         11.0     0.000000
75%         11.0     0.000000
max         11.0     1.000000

[8 rows x 25 columns]
```

```
# we can index the data using dates, let us format
marketdata_smaller["Dt_Customer"] =
pd.to_datetime(marketdata_smaller["Dt_Customer"])
marketdata_smaller.index = marketdata_smaller["Dt_Customer"]
```

```
 # These info can be summarized as one feature 'Spent', you can check
other extracted feature in the solution from kaggle
marketdata_smaller.iloc[:, 5:11]
```

```
             Recency  MntWines  MntFruits  MntMeatProducts
MntFishProducts  \
```

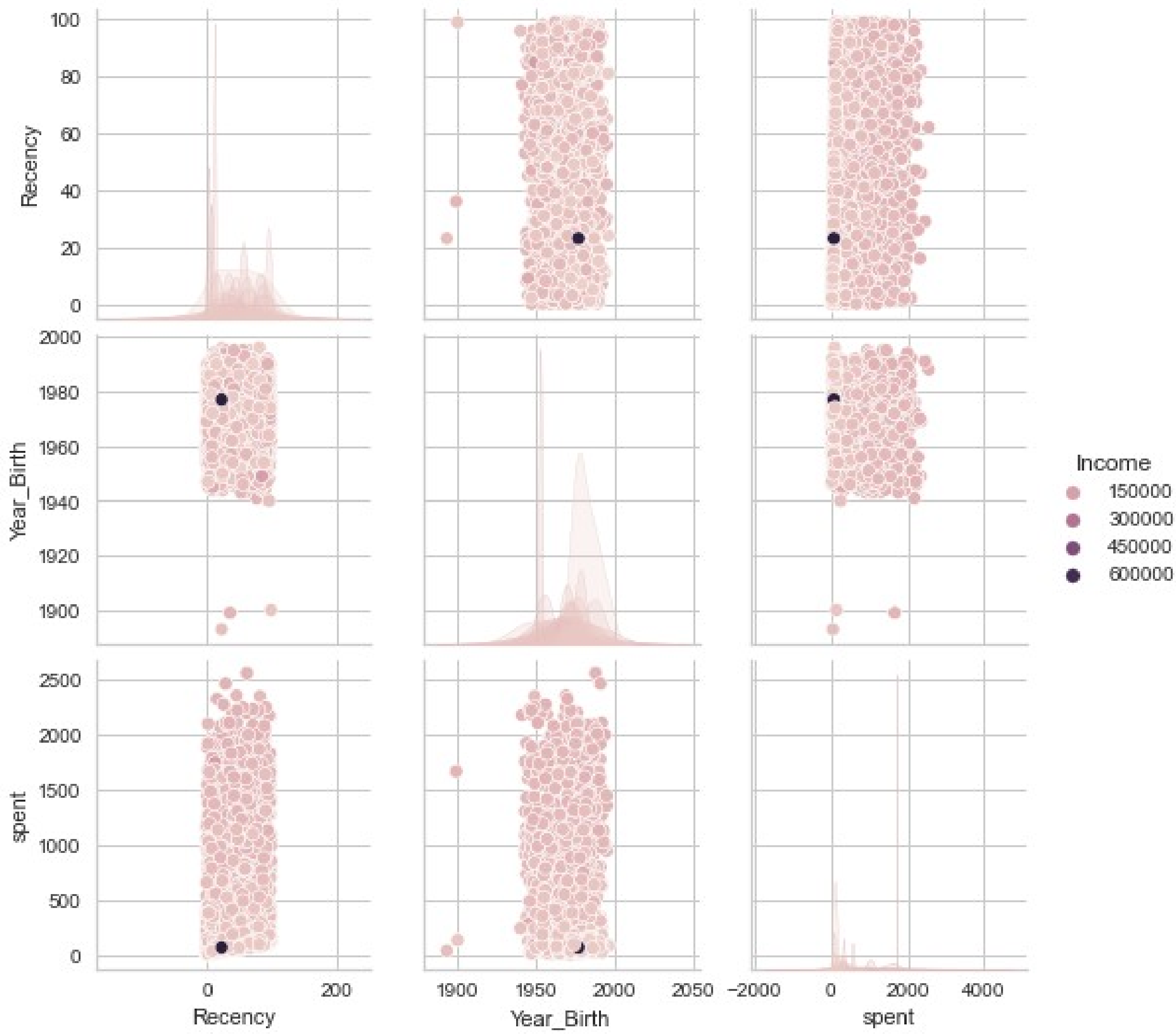
```
Dt_Customer

2012-04-09        58      635       88              546
172
2014-08-03        38       11        1                6
2
2013-08-21        26      426       49              127
111
2014-10-02        26       11        4               20
10
2014-01-19        94      173       43              118
46
...              ...      ...      ...              ...
...
2013-06-13        46      709       43              182
42
2014-10-06        56      406        0               30
0
2014-01-25        91      908       48              217
32
2014-01-24         8      428       30              214
80
2012-10-15        40       84        3               61
2

             MntSweetProducts
Dt_Customer
2012-04-09                 88
2014-08-03                  1
2013-08-21                 21
2014-10-02                  3
2014-01-19                 27
...                       ...
2013-06-13                118
2014-10-06                  0
2014-01-25                 12
2014-01-24                 30
2012-10-15                  1

[2216 rows x 6 columns]

# let use create spent feature
# this feature is the total spent on varius items
marketdata_smaller['spent'] = marketdata_smaller.iloc[:,
5:11].sum(axis=1)

# let us remove raw data that are summarized to spent above
marketdata_smaller= marketdata_smaller.drop(['Dt_Customer',
                                             'MntWines',
                                             'MntFruits',
```

```
                                                          'MntMeatProducts',
                                                          'MntFishProducts',
                                                          'MntSweetProducts',
                                                          'MntGoldProds'], axis=1)

# let us slice a year of customer behavior
marketdata_smaller['2013-01-01': '2013-12-30']
```

```
             Year_Birth    Income  Kidhome  Teenhome  Recency  \
Dt_Customer
2013-08-21         1965   71613.0        0         0       26
2013-09-09         1967   62513.0        0         1       16
2013-08-05         1985   33454.0        1         0       32
2013-06-06         1974   30351.0        1         0       19
2013-11-15         1959   63033.0        0         0       82
...                 ...       ...      ...       ...      ...
2013-03-03         1962   57967.0        0         1       39
2013-03-16         1984   11012.0        1         0       82
2013-02-06         1977  666666.0        1         0       23
2013-01-07         1974   34421.0        1         0       81
2013-06-13         1967   61223.0        0         1       46

             NumDealsPurchases  NumWebPurchases
NumCatalogPurchases  \
Dt_Customer

2013-08-21                   1                8                    2

2013-09-09                   2                6                    4

2013-08-05                   2                4                    0

2013-06-06                   1                3                    0

2013-11-15                   1                3                    4

...                        ...              ...                  ...

2013-03-03                   5                4                    2

2013-03-16                   3                3                    1

2013-02-06                   4                3                    1

2013-01-07                   1                1                    0

2013-06-13                   2                9                    3


             NumStorePurchases  NumWebVisitsMonth  AcceptedCmp3
AcceptedCmp4  \
```

```
Dt_Customer

2013-08-21                     10                    4             0
0
2013-09-09                     10                    6             0
0
2013-08-05                      4                    8             0
0
2013-06-06                      2                    9             0
0
2013-11-15                      8                    2             0
0
...                           ...                  ...           ...
...
2013-03-03                      8                    5             0
0
2013-03-16                      2                    9             1
0
2013-02-06                      3                    6             0
0
2013-01-07                      2                    7             0
0
2013-06-13                      4                    5             0
0

             AcceptedCmp5  AcceptedCmp1  AcceptedCmp2  Complain  \
Dt_Customer
2013-08-21              0             0             0         0
2013-09-09              0             0             0         0
2013-08-05              0             0             0         0
2013-06-06              0             0             0         0
2013-11-15              0             0             0         0
...                   ...           ...           ...       ...
2013-03-03              0             0             0         0
2013-03-16              0             0             0         0
2013-02-06              0             0             0         0
2013-01-07              0             0             0         0
2013-06-13              0             0             0         0

             Z_CostContact  Z_Revenue  Response  spent
Dt_Customer
2013-08-21               3         11         0    760
2013-09-09               3         11         0    718
2013-08-05               3         11         0    178
2013-06-06               3         11         1     63
2013-11-15               3         11         0   1154
...                    ...        ...       ...    ...
2013-03-03               3         11         0    416
2013-03-16               3         11         0    143
2013-02-06               3         11         0     73
```

```
2013-01-07                3         11          0    102
2013-06-13                3         11          0   1140

[1170 rows x 20 columns]

To_Plot = [ "Income", "Recency", "Year_Birth", "spent"]
marketdata_smaller[To_Plot]

              Income  Recency  Year_Birth  spent
Dt_Customer
2012-04-09   58138.0       58        1957   1587
2014-08-03   46344.0       38        1954     59
2013-08-21   71613.0       26        1965    760
2014-10-02   26646.0       26        1984     74
2014-01-19   58293.0       94        1981    501
...              ...      ...         ...    ...
2013-06-13   61223.0       46        1967   1140
2014-10-06   64014.0       56        1946    492
2014-01-25   56981.0       91        1981   1308
2014-01-24   69245.0        8        1956    790
2012-10-15   52869.0       40        1954    191

[2216 rows x 4 columns]

# There are many info in the dataset, let us pick a subset for this
exercise and perform clustering

# We can detect outliers using various methods, may pairplot using
seaborn library is a nice method
# box plot and other methods can be used to conduct thorough denoising

import seaborn as sns
To_Plot = [ "Income", "Recency", "Year_Birth", "spent"]
data= marketdata_smaller[To_Plot]

# the indexing to avoid sns error due to repeated date indexing
data.index = range(0,len(marketdata_smaller))
plt.figure()
sns.pairplot(data, hue='Income')
plt.show()

<Figure size 576x396 with 0 Axes>
```



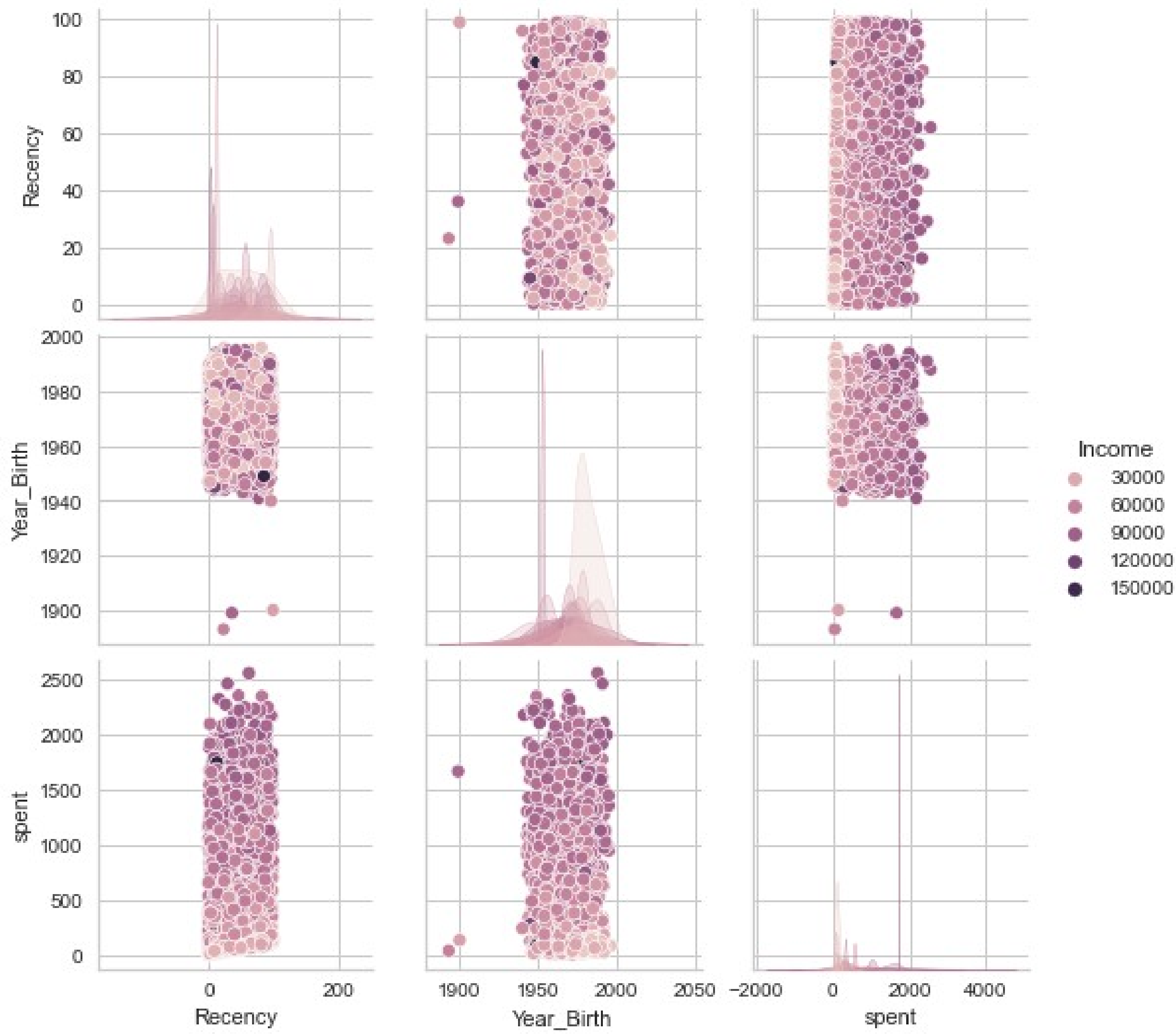
over 600000 income seams outlier where we can remove them. also we can check year_birth or age to remove records that don't make sense

```
# remove those records greater than 600K
data = data[(data['Income']<600000)]
len(data)
```

```
2215
```

```
plt.figure()
sns.pairplot(data, hue='Income')
plt.show()
```

```
<Figure size 576x396 with 0 Axes>
```

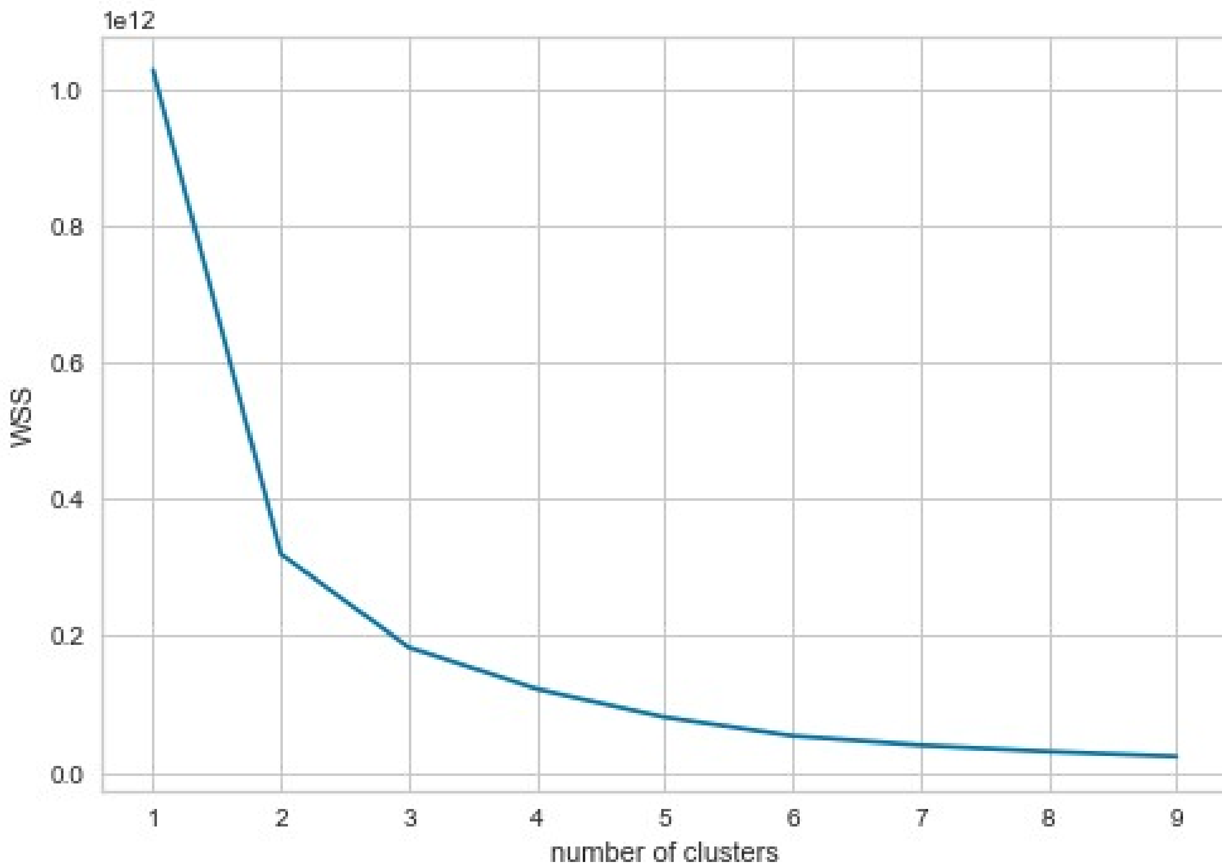


Clustering

we can perform clustering on the four features, but the big question how many clusters we should have?, then which clustering algorithm of the three methods we should use? Our exercise is to try the above three clustering algorithms on this dataset (**data**), and validating the results, to answer the questions in this exercise.

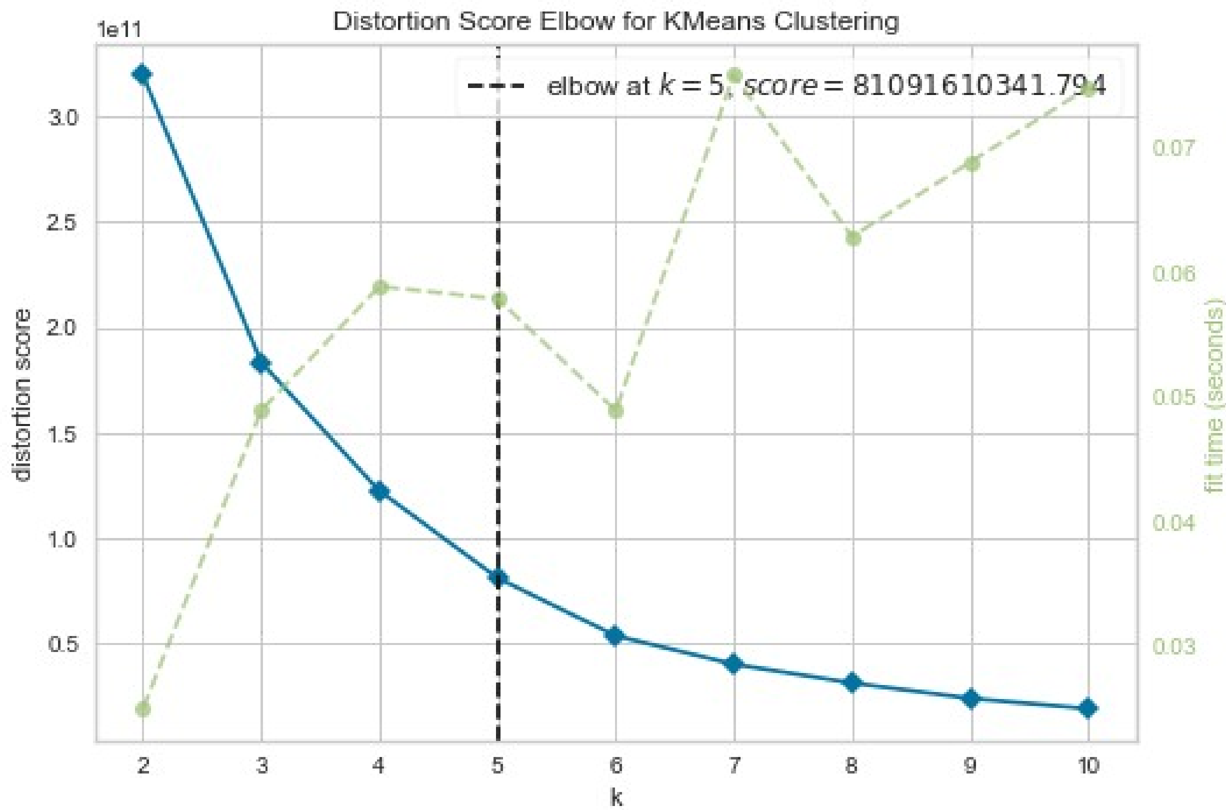
```
# Elbow method using Kmeans
# set a value for the list of k to be tested
klst = 10
# We already created a function to find wss using Kmeans
wss = Find_K_Kmeans(data, klst)
# plot the wss and ks and figureout what is the optimal k ?
plt.plot(range(1,klst), wss)
plt.xlabel('number of clusters')
```

```
plt.ylabel('WSS')
plt.show()
```



The elbow method tells us the data may have either 2 or 3 clusters. We can check both for sure and further analysis has to be conducted with data domain experts to validate findings!

```
# using a tool KElbowvisualizer
vis = KElbowVisualizer(KMeans(), k=10)
vis.fit(data)
vis.show()
plt.show()
```



The answer to how many clusters is different here.. 5 clusters could be the solution

```
# culstering results
kmeans = KMeans(n_clusters=2).fit(data)
y = kmeans.predict(data)
# clustering validation
print('Silhouette score %0.2f'%silhouette_score(data,y))

Silhouette score 0.61

# culstering results
kmeans = KMeans(n_clusters=3).fit(data)
y = kmeans.predict(data)
# clustering validation
print('Silhouette score %0.2f'%silhouette_score(data,y))

Silhouette score 0.54

# culstering results
kmeans = KMeans(n_clusters=5).fit(data)
y = kmeans.predict(data)
# clustering validation
print('Silhouette score %0.2f'%silhouette_score(data,y))

Silhouette score 0.54
```

In general, positive Silhouette and above 50% is good clustering result

```
# Elbow method using AC ( additional parameter - linkage)

# DBSCAN method
```

Conclusions