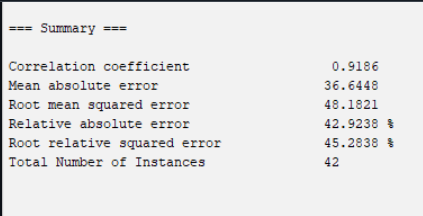
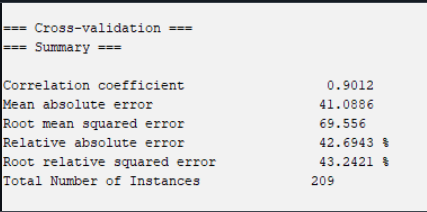
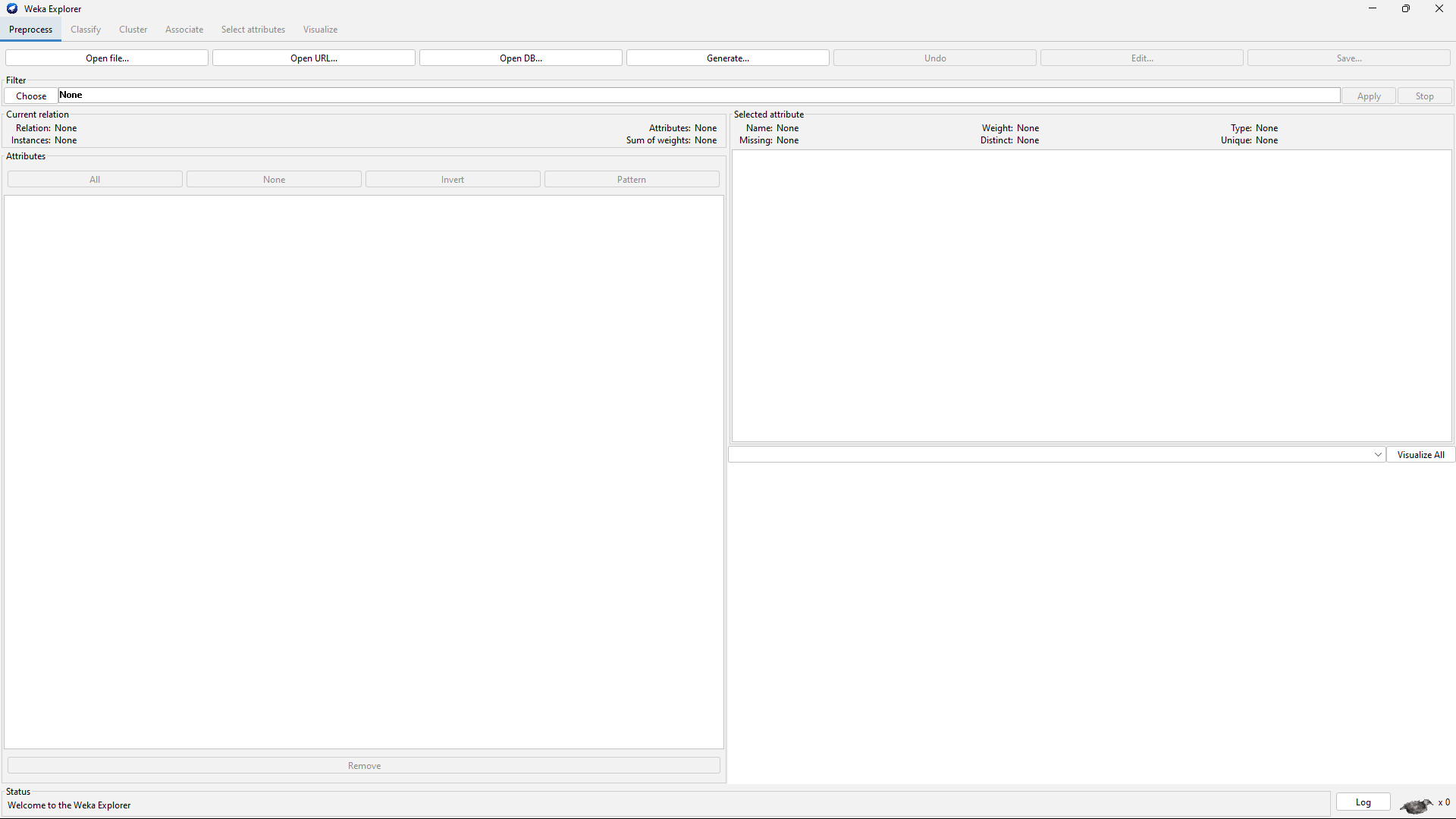
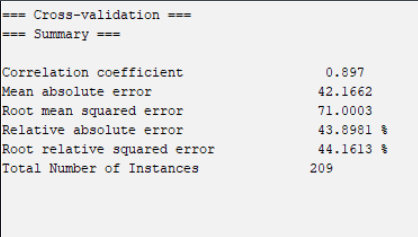
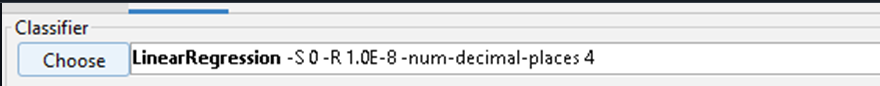
***Aditya Parashar***

SAP ID:500119510

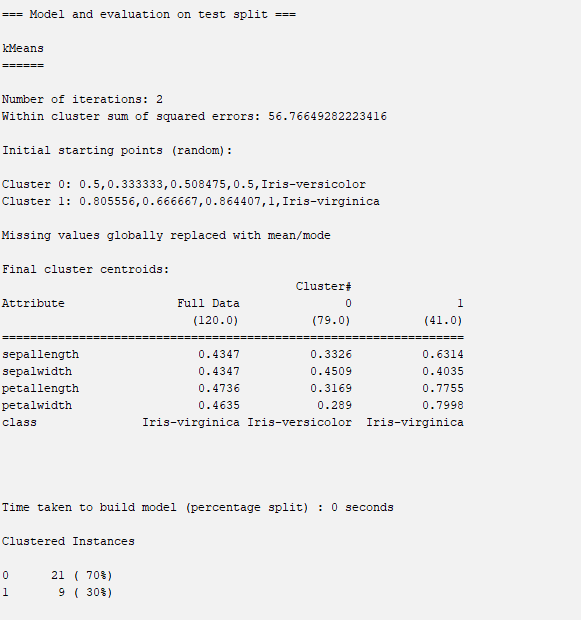
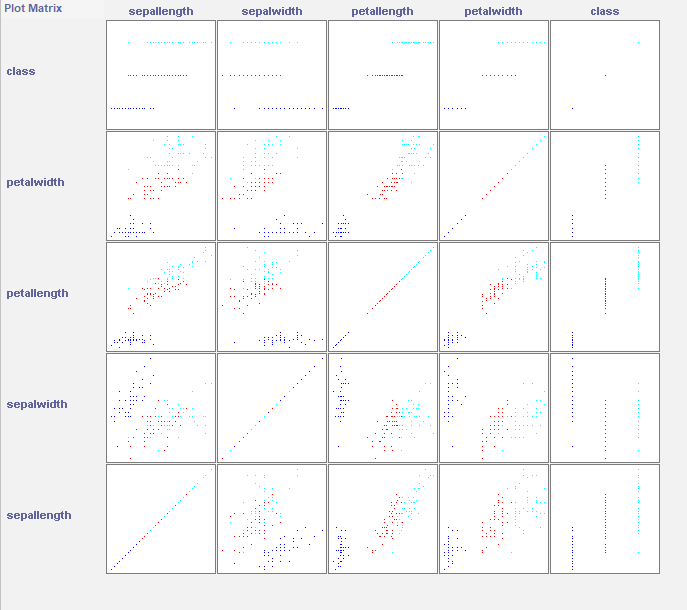
**AIML WORK**

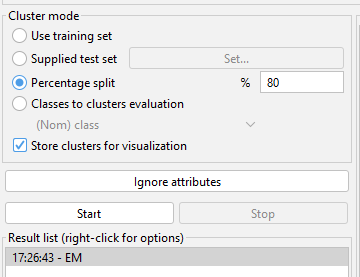
# **Lab - 1**



# **Lab – 2**

# **Lab – 3**





The task of grouping data points based on their similarity with each other is called Clustering or Cluster Analysis.

The various types of clustering are:

* Hierarchical clustering
* Partitioning clustering

Hierarchical clustering is further subdivided into:

* Agglomerative clustering
* Divisive clustering

Partitioning clustering is further subdivided into:

* K-Means clustering
* Fuzzy C-Means clustering

W There is no labeled data for this clustering, unlike in supervised learning. K-Means performs the division of objects into clusters that share similarities and are dissimilar to the objects belonging to another cluster.

## STEPS INVOLVED

dataset used :- *iris.arff*

* NORMALISE THE DATASET
* Go to cluster menu and use percentage split **80%**

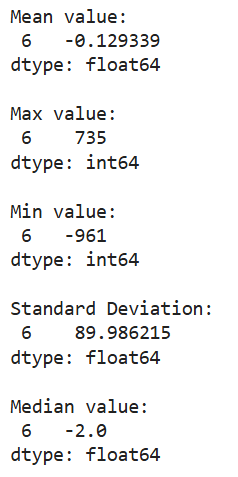
# **Lab – 4**

**import** pandas **as** pd

**import** pandas **as** pd  
df1 **=** pd**.**read\_csv('Normal-ECG.csv')  
df2 **=** pd**.**read\_csv('Dia-ECG.csv')

mean\_value **=** df1**.**mean()  
max\_value **=** df1**.**max()  
min\_value **=** df1**.**min()  
std\_dev **=** df1**.**std()  
median\_value **=** df1**.**median()

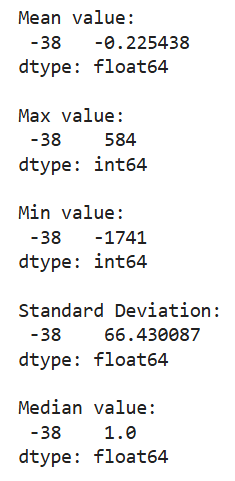
print("Mean value:\n", mean\_value)  
print("\nMax value:\n", max\_value)  
print("\nMin value:\n", min\_value)  
print("\nStandard Deviation:\n", std\_dev)  
print("\nMedian value:\n", median\_value)

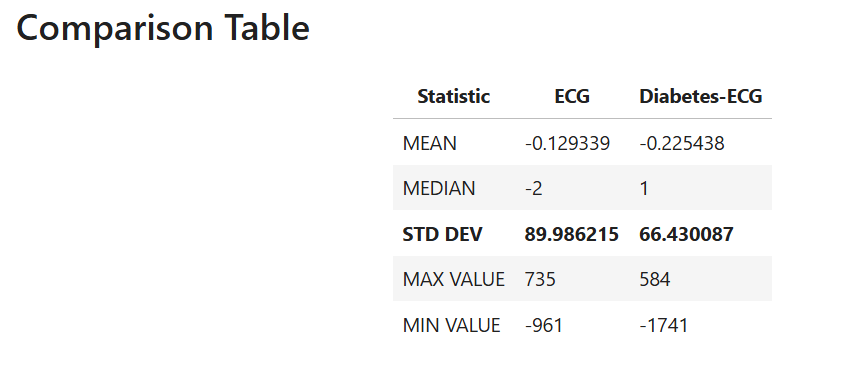


df2**.**replace(['ekg', 'uv'], 0, inplace**=True**)

mean\_value **=** df2**.**mean()  
max\_value **=** df2**.**max()  
min\_value **=** df2**.**min()  
std\_dev **=** df2**.**std()  
median\_value **=** df2**.**median()

print("Mean value:\n", mean\_value)  
print("\nMax value:\n", max\_value)  
print("\nMin value:\n", min\_value)  
print("\nStandard Deviation:\n", std\_dev)  
print("\nMedian value:\n", median\_value)





# **Lab – 5**

*# Load libraries*  
**import** pandas **as** pd  
**from** sklearn.tree **import** DecisionTreeClassifier *# Import Decision Tree Classifier*  
**from** sklearn.model\_selection **import** train\_test\_split *# Import train\_test\_split function*  
**from** sklearn **import** metrics *#Import scikit-learn metrics module for accuracy calculation*

emotions **=** pd**.**read\_csv("emotions.csv")

positive **=** emotions[emotions['label'] **==** 'POSITIVE']**.**drop(columns**=**['label'])  
negative **=** emotions[emotions['label'] **==** 'NEGATIVE']**.**drop(columns**=**['label'])  
neutral **=** emotions[emotions['label'] **==** 'NEUTRAL']**.**drop(columns**=**['label'])

*# MEANS*   
avg\_positive **=** positive**.**mean()  
print(avg\_positive)  
avg\_negative **=** negative**.**mean()  
print(avg\_negative)  
avg\_neutral **=** neutral**.**mean()  
print(avg\_neutral)

*# MEDIAN*  
median\_positive **=** positive**.**median()  
print(median\_positive)  
median\_negative **=** negative**.**median()  
print(median\_negative)  
median\_neutral **=** neutral**.**median()  
print(median\_neutral)

*# MODE*  
mode\_positive **=** positive**.**mode()  
print(mode\_positive)  
mode\_negative **=** negative**.**mode()  
print(mode\_negative)  
mode\_neutral **=** neutral**.**mode()  
print(mode\_neutral)

*# MAX*  
max\_positive **=** positive**.**max()  
print(max\_positive)  
max\_negative **=** negative**.**max()  
print(max\_negative)  
max\_neutral **=** neutral**.**mode()  
print(max\_neutral)

*# MIN*  
min\_positive **=** positive**.**min()  
print(min\_positive)  
min\_negative **=** negative**.**min()  
print(min\_negative)  
min\_neutral **=** neutral**.**min()  
print(min\_neutral)

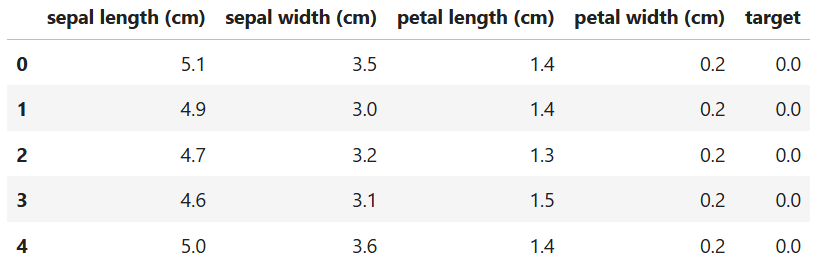
*# STANDARD DEVIATION*  
std\_positive **=** positive**.**std()  
print(std\_positive)  
std\_negative **=** negative**.**std()  
print(std\_negative)  
std\_neutral **=** neutral**.**std()  
print(std\_neutral)

count\_positive **=** positive**.**value\_counts()  
print(count\_positive)  
count\_negative **=** negative**.**value\_counts()  
print(count\_negative)  
count\_neutral **=** neutral**.**value\_counts()  
print(count\_neutral)

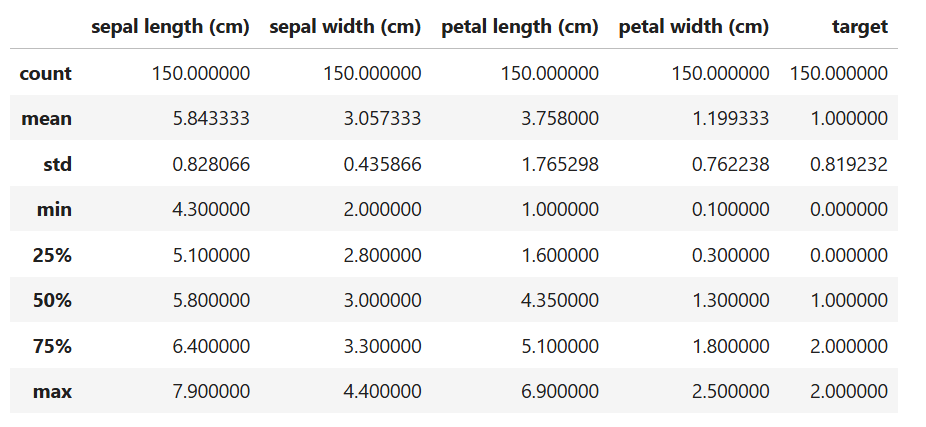
# **Lab – 6**

**import** matplotlib.pyplot **as** plt  
**import** numpy **as** np  
**import** pandas **as** pd  
**import** seaborn **as** sns  
  
**from** sklearn **import** datasets  
**from** sklearn.model\_selection **import** train\_test\_split , KFold  
**from** sklearn.preprocessing **import** Normalizer  
**from** sklearn.metrics **import** accuracy\_score  
**from** sklearn.neighbors **import** KNeighborsClassifier  
  
**from** collections **import** Counter

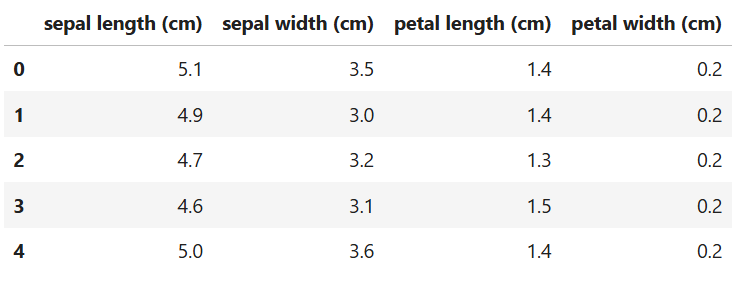
iris **=** datasets**.**load\_iris()  
*# np.c\_ is the numpy concatenate function*  
iris\_df **=** pd**.**DataFrame(data**=** np**.**c\_[iris['data'], iris['target']],  
 columns**=** iris['feature\_names'] **+** ['target'])  
iris\_df**.**head()

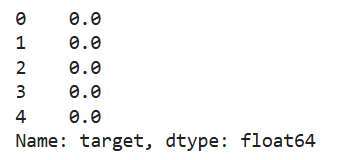
iris**.**feature\_names  
iris**.**target\_names

iris\_df**.**describe()

x**=** iris\_df**.**iloc[:, :**-**1]  
y**=** iris\_df**.**iloc[:, **-**1]

x**.**head()

y**.**head()



iris\_df[iris\_df**.**target **==**1]**.**head  
iris\_df[iris\_df**.**target **==**2]**.**head

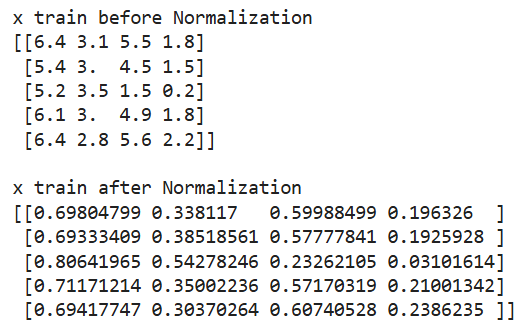
x\_train, x\_test, y\_train, y\_test**=** train\_test\_split(x, y,  
 test\_size**=** 0.2,  
 shuffle**=** **True**,  
 random\_state**=** 0)  
x\_train**=** np**.**asarray(x\_train)  
y\_train**=** np**.**asarray(y\_train)  
x\_test**=** np**.**asarray(x\_test)  
y\_test**=** np**.**asarray(y\_test)

print(f'training set size: {x\_train**.**shape[0]} samples \ntest set size: {x\_test**.**shape[0]} samples')



scaler**=** Normalizer()**.**fit(x\_train)  
normalized\_x\_train**=** scaler**.**transform(x\_train)  
normalized\_x\_test**=** scaler**.**transform(x\_test)

print('x train before Normalization')  
print(x\_train[0:5])  
print('\nx train after Normalization')  
print(normalized\_x\_train[0:5])



*## Before*  
*# View the relationships between variables; color code by species type*  
di**=** {0.0: 'Setosa', 1.0: 'Versicolor', 2.0:'Virginica'} *# dictionary*  
  
*before***=** sns**.**pairplot(iris\_df**.**replace({'target': di}), hue**=** 'target')  
before**.**fig**.**suptitle('Pair Plot of the dataset Before normalization', y**=**1.08)  
  
*## After*  
iris\_df\_2**=** pd**.**DataFrame(data**=** np**.**c\_[normalized\_x\_train, y\_train],  
 columns**=** iris['feature\_names'] **+** ['target'])  
di**=** {0.0: 'Setosa', 1.0: 'Versicolor', 2.0: 'Virginica'}  
after**=** sns**.**pairplot(iris\_df\_2**.**replace({'target':di}), hue**=** 'target')  
after**.**fig**.**suptitle('Pair Plot of the dataset After normalization', y**=**1.08)

**def** distance\_ecu(x\_train, x\_test\_point):  
 """  
 Input:  
 - x\_train: corresponding to the training data  
 - x\_test\_point: corresponding to the test point  
  
 Output:  
 -distances: The distances between the test point and each point in the training data.  
  
 """  
 distances**=** [] *## create empty list called distances*  
 **for** row **in** range(len(x\_train)): *## Loop over the rows of x\_train*  
 current\_train\_point**=** x\_train[row] *#Get them point by point*  
 current\_distance**=** 0 *## initialize the distance by zero*  
  
 **for** col **in** range(len(current\_train\_point)): *## Loop over the columns of the row*  
   
 current\_distance **+=** (current\_train\_point[col] **-** x\_test\_point[col]) **\*\***2  
 *## Or current\_distance = current\_distance + (x\_train[i] - x\_test\_point[i])\*\*2*  
 current\_distance**=** np**.**sqrt(current\_distance)  
  
 distances**.**append(current\_distance) *## Append the distances*  
  
 *# Store distances in a dataframe*  
 distances**=** pd**.**DataFrame(data**=**distances,columns**=**['dist'])  
 **return** distances

In [46]:

**def** nearest\_neighbors(distance\_point, K):  
 """  
 Input:  
 -distance\_point: the distances between the test point and each point in the training data.  
 -K : the number of neighbors  
  
 Output:  
 -df\_nearest: the nearest K neighbors between the test point and the training data.  
  
 """  
  
 *# Sort values using the sort\_values function*  
 df\_nearest**=** distance\_point**.**sort\_values(by**=**['dist'], axis**=**0)  
  
 *## Take only the first K neighbors*  
 df\_nearest**=** df\_nearest[:K]  
 **return** df\_nearest

In [47]:

**def** voting(df\_nearest, y\_train):  
 """  
 Input:  
 -df\_nearest: dataframe contains the nearest K neighbors between the full training dataset and the test point.  
 -y\_train: the labels of the training dataset.  
  
 Output:  
 -y\_pred: the prediction based on Majority Voting  
  
 """  
  
 *## Use the Counter Object to get the labels with K nearest neighbors.*  
 counter\_vote**=** Counter(y\_train[df\_nearest**.**index])  
  
 y\_pred**=** counter\_vote**.**most\_common()[0][0] *# Majority Voting*  
  
 **return** y\_pred

In [48]:

**def** KNN\_from\_scratch(x\_train, y\_train, x\_test, K):  
  
 """  
 Input:  
 -x\_train: the full training dataset  
 -y\_train: the labels of the training dataset  
 -x\_test: the full test dataset  
 -K: the number of neighbors  
  
 Output:  
 -y\_pred: the prediction for the whole test set based on Majority Voting.  
  
 """  
  
 y\_pred**=**[]  
  
 *## Loop over all the test set and perform the three steps*  
 **for** x\_test\_point **in** x\_test:  
 distance\_point **=** distance\_ecu(x\_train, x\_test\_point) *## Step 1*  
 df\_nearest\_point**=** nearest\_neighbors(distance\_point, K) *## Step 2*  
 y\_pred\_point **=** voting(df\_nearest\_point, y\_train) *## Step 3*  
 y\_pred**.**append(y\_pred\_point)  
  
 **return** y\_pred

In [49]:

K**=**3  
y\_pred\_scratch**=** KNN\_from\_scratch(normalized\_x\_train, y\_train, normalized\_x\_test, K)  
print(y\_pred\_scratch)

knn**=**KNeighborsClassifier(K)  
knn**.**fit(normalized\_x\_train, y\_train)  
y\_pred\_sklearn**=** knn**.**predict(normalized\_x\_test)  
print(y\_pred\_sklearn)

print(np**.**array\_equal(y\_pred\_sklearn, y\_pred\_scratch))

print(f'The accuracy of our implementation is {accuracy\_score(y\_test, y\_pred\_scratch)}')  
print(f'The accuracy of sklearn implementation is {accuracy\_score(y\_test, y\_pred\_sklearn)}')

n\_splits**=** 4 *## Choose the number of splits*  
kf**=** KFold(n\_splits**=** n\_splits) *## Call the K Fold function*  
  
*accuracy\_k***=** [] *## Keep track of the accuracy for each K*  
k\_values**=** list(range(1,30,2)) *## Search for the best value of K*  
  
**for** k **in** k\_values: *## Loop over the K values*  
 accuracy\_fold**=** 0  
 **for** normalized\_x\_train\_fold\_idx, normalized\_x\_valid\_fold\_idx **in** kf**.**split(normalized\_x\_train): *## Loop over the splits*  
 normalized\_x\_train\_fold**=** normalized\_x\_train[normalized\_x\_train\_fold\_idx] *## fetch the values*  
 y\_train\_fold**=** y\_train[normalized\_x\_train\_fold\_idx]  
  
 normalized\_x\_test\_fold**=** normalized\_x\_train[normalized\_x\_valid\_fold\_idx]  
 y\_valid\_fold**=** y\_train[normalized\_x\_valid\_fold\_idx]  
 y\_pred\_fold**=** KNN\_from\_scratch(normalized\_x\_train\_fold, y\_train\_fold, normalized\_x\_test\_fold, k)  
  
 accuracy\_fold**+=** accuracy\_score (y\_pred\_fold, y\_valid\_fold) *## Accumulate the accuracy*  
 accuracy\_fold**=** accuracy\_fold**/** n\_splits *## Divide by the number of splits*  
 accuracy\_k**.**append(accuracy\_fold)

print(f'The accuracy for each K value was {list ( zip (accuracy\_k, k\_values))}') *## creates a tuple with accuracy corresponding to k value*

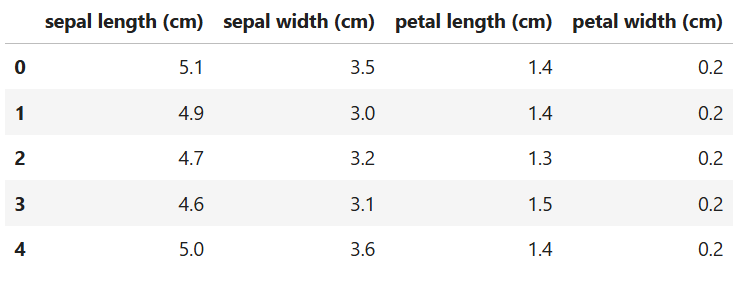
print(f'Best accuracy was {np**.**max(accuracy\_k)}, which corresponds to a value of K= {k\_values[np**.**argmax(accuracy\_k)]}')

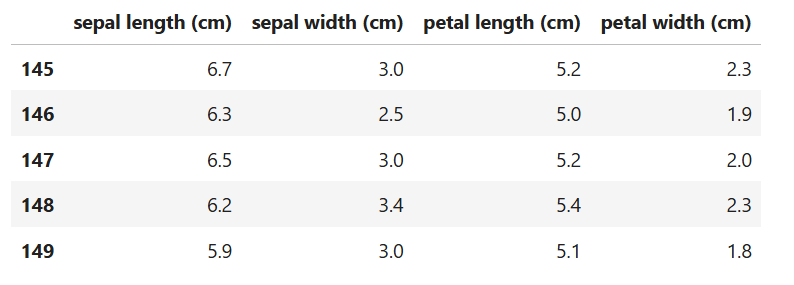
k\_values **=** [1, 3, 5, 7, 9]  
  
results **=** {}  
  
**for** k **in** k\_values:  
 knn **=** KNeighborsClassifier(n\_neighbors**=**k)  
   
 knn**.**fit(x\_train, y\_train)  
   
 y\_pred **=** knn**.**predict(x\_test)  
 accuracy **=** accuracy\_score(y\_test, y\_pred)  
   
 results[k] **=** accuracy  
 print(f'k={k}, Accuracy={accuracy:.4f}')

# **Lab – 7**

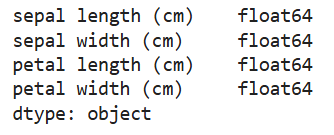
**import** numpy **as** np  
**import** pandas **as** pd  
**from** sklearn.preprocessing **import** OneHotEncoder  
**from** sklearn **import** datasets  
  
iris **=** datasets**.**load\_iris()

df **=** pd**.**DataFrame(iris**.**data, columns**=**iris**.**feature\_names)  
df**.**head()

df**.**tail()

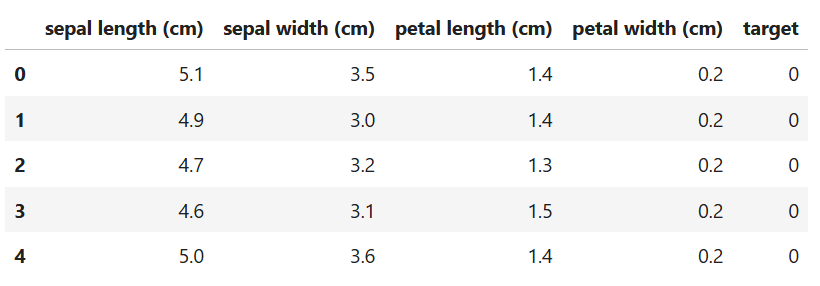


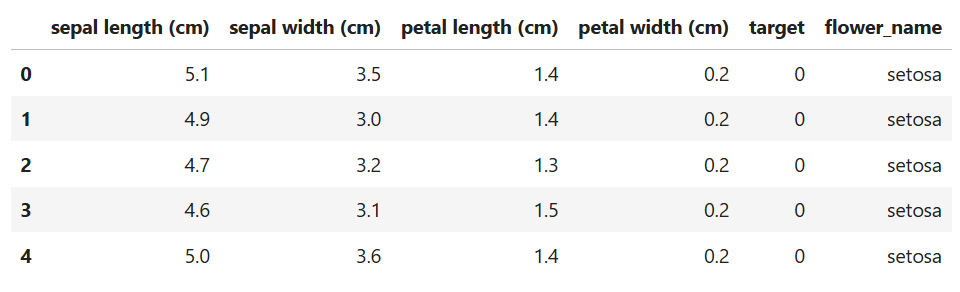
Df**.**dtypes

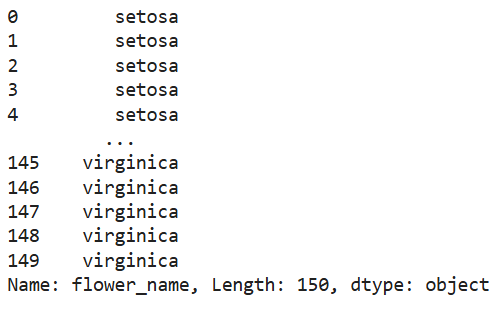


encoder **=** OneHotEncoder()

iris\_data **=** iris['data']  
iris\_target **=** iris['target']  
  
df['target'] **=** iris**.**target  
df**.**head()

df['flower\_name'] **=** df**.**target**.**apply(**lambda** x : iris**.**target\_names[x])  
df**.**head()

print(df["flower\_name"])



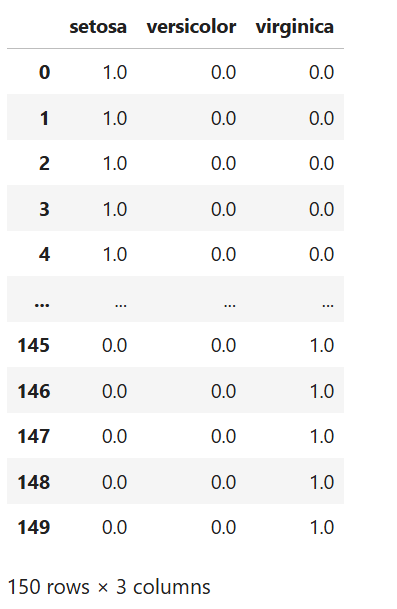
feature\_arr **=** encoder**.**fit\_transform(df[["flower\_name"]])**.**toarray()

feature\_arr

feature\_labels **=** encoder**.**categories\_

feature\_labels **=** np**.**array(feature\_labels)**.**ravel()

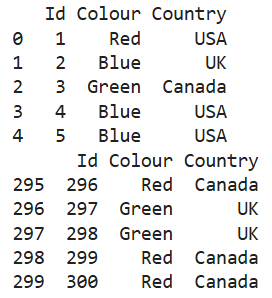
pd**.**DataFrame(feature\_arr, columns**=**feature\_labels)



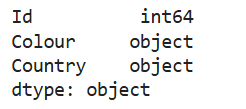
**import** pandas **as** pd  
**from** sklearn.preprocessing **import** OneHotEncoder  
**import** numpy **as** np

df **=** pd**.**read\_csv("data.csv")

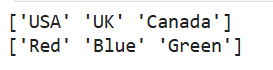
print(df**.**head())  
print(df**.**tail())



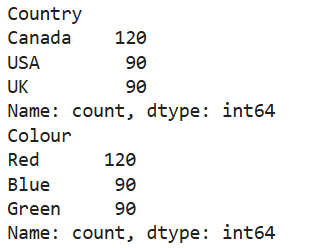
Df**.**dtypes



print(df['Country']**.**unique())  
print(df['Colour']**.**unique())

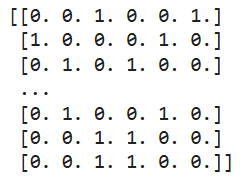


print(df['Country']**.**value\_counts())  
print(df['Colour']**.**value\_counts())



ohe **=** OneHotEncoder()

print(ohe**.**fit\_transform(df[["Colour" , "Country"]])**.**toarray)  
print(ohe**.**fit\_transform(df[["Colour" , "Country"]])**.**toarray())

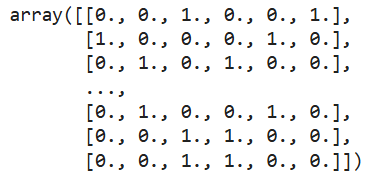


feature\_array **=** ohe**.**fit\_transform(df[["Colour" , "Country"]])**.**toarray()

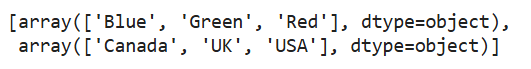
ohe**.**categories\_  
feature\_labels **=** ohe**.**categories\_

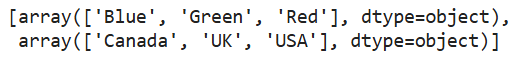
feature\_labels **=** np**.**array(feature\_labels)**.**ravel()

feature\_array

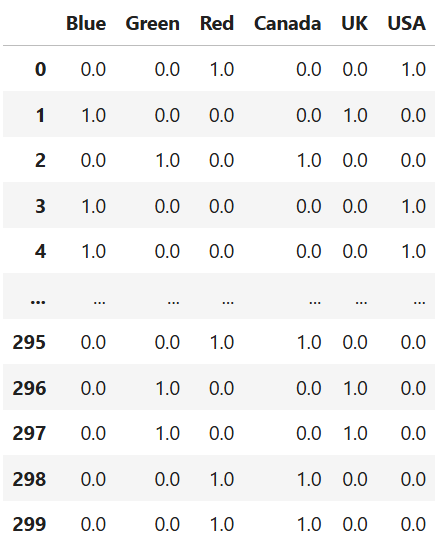


feature\_labels





pd**.**DataFrame(feature\_array, columns**=**feature\_labels)



# **Lab – 8**

**import** numpy **as** np  
**import** pandas **as** pd  
**from** sklearn.decomposition **import** PCA  
**from** sklearn.datasets **import** load\_iris  
**from** sklearn.preprocessing **import** StandardScaler  
**import** matplotlib.pyplot **as** plt

iris **=** load\_iris()

scaler **=** StandardScaler()  
iris\_scaled **=** scaler**.**fit\_transform(iris**.**data)

X **=** iris**.**data *# Features*  
y **=** iris**.**target *#species*  
feature\_names **=** iris**.**feature\_names

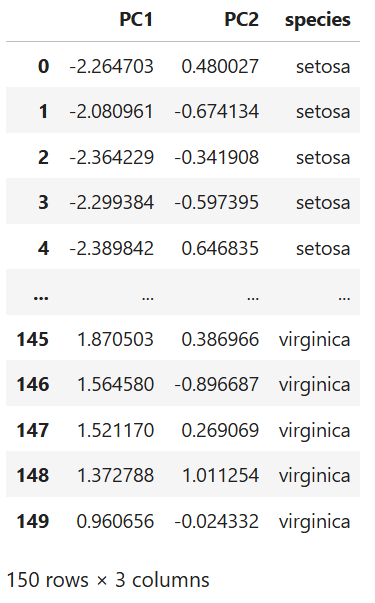
print("Mean of features after scaling:", np**.**mean(iris\_scaled, axis**=**0))  
print("Standard deviation of features after scaling:", np**.**std(iris\_scaled, axis**=**0))

pca **=** PCA(n\_components**=**2)  
iris\_pca **=** pca**.**fit\_transform(iris\_scaled)

pca **=** PCA(n\_components**=**2)  
iris\_pca **=** pca**.**fit\_transform(iris\_scaled)

pca\_df **=** pd**.**DataFrame(data**=**iris\_pca, columns**=**['PC1', 'PC2'])  
pca\_df['species'] **=** pd**.**Categorical**.**from\_codes(y, iris**.**target\_names)

pca\_df



plt**.**figure(figsize**=**(8, 6))  
**for** species **in** iris**.**target\_names:  
 species\_data **=** pca\_df[pca\_df['species'] **==** species]  
 plt**.**scatter(species\_data['PC1'], species\_data['PC2'], label**=**species)  
  
plt**.**xlabel('Principal Component 1')  
plt**.**ylabel('Principal Component 2')  
plt**.**title('PCA of Iris Dataset')  
plt**.**legend()  
plt**.**show()

