- BUILDING THE CLASSIFICATION MODEL USING THE BEST K-VALUE FROM THE QUESTION CM6 :

- PERFOMING THE DATA NORMALIZATION AND ANALYSIS :

- DEVELOPING CLASSIFICATION MODEL FOR BEST K-VALUE USING THE CONCEPT OF WEIGHTED KNN:

- REPORTING THE METRICS ACCURACY, AUC AND F-SCORE :

Importing the Libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

Loading the dataset

In [2]:
```
data = pd.read_csv('cleaned_iris_dataset.csv')
```

Displaying properties of dataset for further processing

In [3]:
```
data.info()
data.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 93 entries, 0 to 92
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   sepal_length  93 non-null     float64
 1   sepal_width   93 non-null     float64
 2   petal_length  93 non-null     float64
 3   petal_width   93 non-null     float64
 4   species       93 non-null     object 
dtypes: float64(4), object(1)
memory usage: 3.8+ KB
```

Out[3]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.045070	2.508203	3.018024	1.164924	Iris-versicolor
1	6.325517	2.498496	4.542052	1.413651	Iris-versicolor
2	5.257497	3.673501	1.470660	0.395348	Iris-setosa
3	6.675168	3.201700	5.785461	2.362764	Iris-virginica
4	5.595237	2.678166	4.077750	1.369266	Iris-versicolor

In [4]:
```
data.tail()
```

Out[4]:

	sepal_length	sepal_width	petal_length	petal_width	species
88	4.874848	3.217348	1.592887	0.123588	Iris-setosa
89	5.564197	2.771731	3.483588	1.074754	Iris-versicolor
90	5.548047	3.673501	1.453466	0.214527	Iris-setosa
91	5.510482	2.652867	4.276817	1.298032	Iris-versicolor
92	4.538713	3.056142	1.545136	0.241424	Iris-setosa

In [5]:
```
data.describe()
```

Out[5]:

	sepal_length	sepal_width	petal_length	petal_width
count	93.000000	93.000000	93.000000	93.000000
mean	5.867894	3.054063	3.808118	1.236858
std	0.892271	0.358692	1.811399	0.770872
min	4.344007	2.498496	1.033031	0.020731
25%	5.152435	2.794790	1.541564	0.343669
50%	5.636744	3.049459	4.192791	1.369266
75%	6.478961	3.239682	5.098860	1.837925
max	7.795561	3.673501	6.768611	2.603123

In [6]:
```
data.shape
```
Out[6]:
```
(93, 5)
```

Fitting the classification model :

In [7]:
```
features_columns = ['sepal_length','sepal_width','petal_length','petal_width']
X = data[features_columns].values
Y = data['species'].values
```

In [8]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2, randon
```

In [9]:
```
from sklearn.neighbors import KNeighborsClassifier

classifier = KNeighborsClassifier(n_neighbors=5)

# Fitting the model
classifier.fit(X_train, Y_train)

# Predicting the Test set results
Y_pred = classifier.predict(X_test)
```

Here, we have fitted our data classification model using the training data for the best K-value of **5.**

We have also tested this trained model on our test data and determined the various metrics of this KNN classifier model as shown.

CALCULATING ACCURACY, AUC AND F-SCORE OF THE TRAINED MODEL :

Testing for accuracy of the model:

In [10]:
```
from sklearn.metrics import accuracy_score
accuracy_rate = accuracy_score(Y_test, Y_pred)*100
print('The accuracy rate of model with the obtained k value of 5 is ' + str(round
```

```
The accuracy rate of model with the obtained k value of 5 is 100 %.
```

Calculating AUC for the model :

In [11]:
```
from sklearn.preprocessing import OneHotEncoder
from sklearn.metrics import roc_auc_score

enc = OneHotEncoder()
Y_train_enc = enc.fit_transform(Y_train.reshape(-1, 1)).todense()
Y_test_enc = enc.transform(Y_test.reshape(-1, 1)).todense()
Y_pred_enc = enc.transform(Y_pred.reshape(-1, 1)).todense()

auc_score = roc_auc_score(Y_test_enc, Y_pred_enc)
print('The AUC score of the classifier model is ', auc_score)
```

```
The AUC score of the classifier model is  1.0
```

In order to calculate the AUC score, initially we have used the one hot encoding to encode the **"species"** feature in the given dataset and then calculated the AUC score.

Calculating the f-score:

In [12]:
```
from sklearn.metrics import f1_score
fsc=f1_score(Y_test, Y_pred,average='micro')
print('The f-score of the best fit model is ' , fsc)
```

```
The f-score of the best fit model is  1.0
```

PERFORMING NORMALISATION AND ANALYSING THE EFFECT ON OUR TRAINED MODEL:

In [13]:
```
#Data Normalisation
from sklearn.preprocessing import MinMaxScaler

#fit scaler on training data
norm = MinMaxScaler().fit(X_train)

#transform training data
X_train_norm = norm.transform(X_train)

#transform testing data
X_test_norm = norm.transform(X_test)
```

Developing the model with normalised data with the best fit K-value and performing validation:

In [14]:
```
from sklearn.model_selection import cross_val_score,KFold
kf_norm=KFold(n_splits=5,random_state=275,shuffle=True)

classifier1 = KNeighborsClassifier(n_neighbors=5)

score_norm = cross_val_score(classifier1, X_train_norm, Y_train, cv=kf_norm, scor

accuracy_rate_norm = (score_norm.mean())*100
print('Validation Accuracy of normalised KNN model at k=5 is equal to '+ str(rour
```

```
Validation Accuracy of normalised KNN model at k=5 is equal to 90 %.
```

Fitting the model with normalised training set and testing with test set:

In [15]:
```
classifier1.fit(X_train_norm, Y_train)

# Predicting the Test set results
Y2_pred = classifier1.predict(X_test_norm)

#Calculating for accuracy of the normalised model with the test data
accuracy_rate1 = accuracy_score(Y_test, Y2_pred)*100
print('The accuracy rate of the best fit normalised model is ' + str(round(accura
```

```
The accuracy rate of the best fit normalised model is 100 %.
```

Here, we have normalised the classifier model by using **Min Max Normalization**. We can see that the accuracy of the normalized data model is **100%**

WEIGHTED KNN:

In [16]:
```
from sklearn.model_selection import cross_val_score,KFold
kf=KFold(n_splits=5,random_state=275,shuffle=True)
```

Performing validation on the model with best fit K-value and implementing euclidean weigthed KNN :

In [17]:
```
classifier_euclidean = KNeighborsClassifier(n_neighbors=5 , metric='euclidean', w
score_euc = cross_val_score(classifier_euclidean, X_train_norm, Y_train, cv=kf, s
accuracy_rate_euc = (score_euc.mean())*100
print('Validation Accuracy of euclidean weigthed and normalised KNN model at k=5
```

```
Validation Accuracy of euclidean weigthed and normalised KNN model at k=5 is eq
ual to 89 %.
```

Performing validation on the model with best fit K-value and implementing manhattan weigthed KNN :

In [18]:
```
# Testing the model with validation set for best fit K-value to the model and imp
classifier_manhattan  = KNeighborsClassifier(n_neighbors=5 , metric='manhattan',
score_man = cross_val_score(classifier_manhattan, X_train_norm, Y_train, cv=kf, s
accuracy_rate_man = (score_man.mean())*100
print('Validation Accuracy of manhattan weigthed and normalised KNN model at k=5
```

```
Validation Accuracy of manhattan weigthed and normalised KNN model at k=5 is eq
ual to 90 %.
```

Performing validation on the model with best fit K-value and implementing default(minkowski) weigthed KNN:

In [19]:
```
# Testing the model with validation set for best fit K-value to the model and imp
classifier_minkowski  = KNeighborsClassifier(n_neighbors=5 , metric='minkowski',
score_min = cross_val_score(classifier_minkowski, X_train_norm, Y_train, cv=kf, s
accuracy_rate_min = (score_min.mean())*100
print('Validation Accuracy of default (minkowski) weigthed and normalised KNN moc
```

```
Validation Accuracy of default (minkowski) weigthed and normalised KNN model at
k=5 is equal to 89 %.
```

Looking after the accuracy of each weighted metrics, we can conclude that **manhattan** distance could result in a good and improved weigthed KNN model.

Fitting the model on to the test dataset taking Manhattan as weighing parameter metric in our Weighted KNN:

In [20]:
```
#Fitting the weighted and normalised KNN model with the training set
classifier_manhattan.fit(X_train_norm, Y_train)

# Predicting the Test set results
Y2_pred_wgtnorm = classifier_manhattan.predict(X_test_norm)

#Testing for accuracy of this model with test data
accuracy_rate_wgtnorm = accuracy_score(Y_test, Y2_pred_wgtnorm)*100
print('The accuracy rate of the best fit normalised and weighted model with manha
```

```
The accuracy rate of the best fit normalised and weighted model with manhattan
distance is 100 %.
```

In [21]:
```
#Calculating f-score for weigthed and normalised model
fsc1=f1_score(Y_test, Y2_pred_wgtnorm,average='micro')
print('The f-score of the best fit normalised model is ' , fsc1)
```

```
The f-score of the best fit normalised model is  1.0
```

In [22]:
```
#Calculating AUC for weighted and normalised model

enc1 = OneHotEncoder()
Y_train_enc_wgtnorm = enc1.fit_transform(Y_train.reshape(-1, 1)).todense()
Y_test_enc_wgtnorm = enc1.transform(Y_test.reshape(-1, 1)).todense()
Y_pred_enc_wgtnorm = enc1.transform(Y2_pred_wgtnorm.reshape(-1, 1)).todense()

auc_score1 = roc_auc_score(Y_test_enc_wgtnorm, Y_pred_enc_wgtnorm)
print('The AUC of the normalised and weighted KNN classifier model is ', auc_scor
```

```
The AUC of the normalised and weighted KNN classifier model is  1.0
```