IT402 Assignment-1 :- KNN Classifier

Name: Suyash Chintawar(191IT109)

In [ ]:

```
import random
import operator
import numpy as np
import pandas as pd
import seaborn as sn
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import KFold
from sklearn.metrics import confusion_matrix
```

In [ ]:

```
#Load Iris dataset
iris = load_iris()
df = pd.DataFrame(data= np.c_[iris['data'], iris['target']], columns= ['sepal_lengt
df = df.sample(frac = 1,random_state=12).reset_index(drop=True)
df
```

Out[25]:

	sepal_length	sepal_width	petal_length	petal_width	class
0	5.0	3.5	1.3	0.3	0.0
1	6.3	2.5	5.0	1.9	2.0
2	4.4	3.0	1.3	0.2	0.0
3	5.7	2.8	4.1	1.3	1.0
4	6.8	3.2	5.9	2.3	2.0
...	...	...	...	...	...
145	6.8	2.8	4.8	1.4	1.0
146	4.6	3.1	1.5	0.2	0.0
147	7.4	2.8	6.1	1.9	2.0
148	6.1	2.6	5.6	1.4	2.0
149	6.6	3.0	4.4	1.4	1.0

150 rows × 5 columns

In [ ]:

```
def euclidean_distance(row1, row2):
    distance = 0.0
    for i in range(len(row1)-1):
        distance += (row1[i] - row2[i])**2
    return np.sqrt(distance)
```

In [ ]:

```
def get_neighbors(trainingSet, testInstance, k):
    distances = []
    length = len(testInstance)-1
    for x in range(len(trainingSet)):
        dist = euclidean_distance(testInstance, trainingSet[x])
        distances.append((trainingSet[x], dist))
    distances.sort(key=operator.itemgetter(1))
    neighbors = []
    for x in range(k):
        neighbors.append(distances[x][0])
    return neighbors
```

In [ ]:

```
def get_class(neighbors):
    classVotes = {}
    for x in range(len(neighbors)):
        response = neighbors[x][-1]
        if response in classVotes:
            classVotes[response] += 1
        else:
            classVotes[response] = 1
    sortedVotes = sorted(classVotes.items(), key=operator.itemgetter(1), reverse=Tr
    return sortedVotes[0][0]
```

In [ ]:

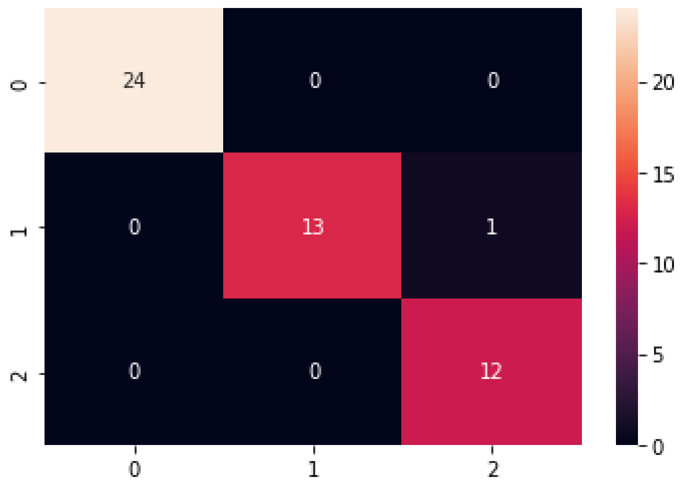
```
def get_metrics(actual, pred):
  conf_mat = confusion_matrix(actual, pred)
  accuracy = np.sum(np.diag(conf_mat))/np.sum(conf_mat)
  precision = np.mean(np.diag(conf_mat)/np.sum(conf_mat, axis=0))
  recall = np.mean(np.diag(conf_mat)/np.sum(conf_mat, axis=1))
  f1_score = 2*precision*recall/(precision+recall)
  return accuracy, precision, recall, f1_score
```

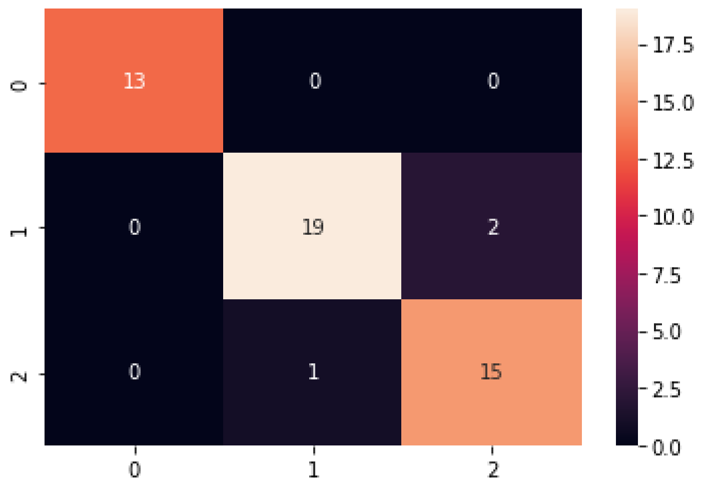
In [ ]:

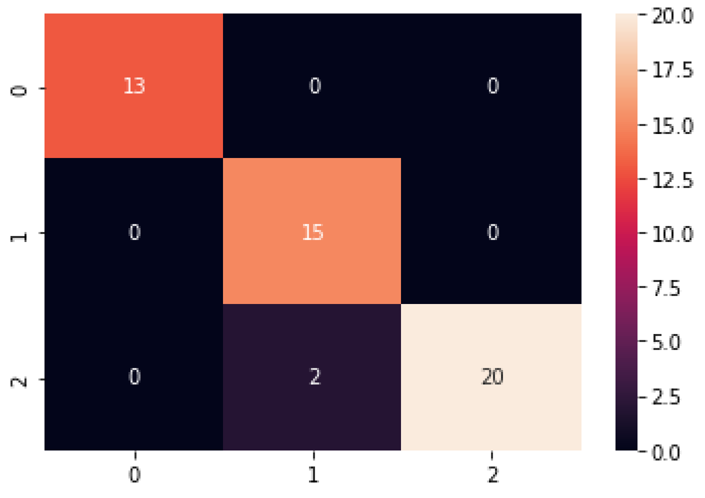
```
# KNN Classifier
# Parameters:
# 1) 'k' value in KNN: 3
# 2) 'k' value in cross validation: 3

k = 3
df_numpy = df.values
kf = KFold(n_splits=k, random_state=None, shuffle=False)

accuracies = []
precisions = []
recalls = []
f1_scores = []
pred=[]
actual=[]
fold=0

for train_index, test_index in kf.split(df_numpy):
  fold+=1
  pred=[]
  df_train, df_test = df_numpy[train_index], df_numpy[test_index]
  df_train = df_train.tolist()
  df_test = df_test.tolist()

  for x in range(len(df_test)):
    neighbors = get_neighbors(df_train, df_test[x], k)
    result = get_class(neighbors)
    pred.append(result)

  actual = [row[-1] for row in df_test]

  acc, pre, rec, f1 = get_metrics(pred, actual)
  accuracies.append(acc)
  precisions.append(pre)
  recalls.append(rec)
  f1_scores.append(f1)

  print('****Metrics for fold no:',fold,'****')
  print('Accuracy:',acc)
  print('Precision:',pre)
  print('Recall:',rec)
  print('F1 score:',f1,end='\n\n')

  plt.figure()
  cm = confusion_matrix(actual, pred)
  sn.heatmap(cm, annot=True)

print('****FINAL METRICS ACROSS ALL FOLDS****')
print('Accuracy:',np.mean(accuracies))
print('Precision:',np.mean(precisions))
print('Recall:',np.mean(recalls))
print('F1 score:',np.mean(f1_scores))
```

```
****Metrics for fold no: 1 ****
Accuracy: 0.98
Precision: 0.9761904761904763
Recall: 0.9743589743589745
F1 score: 0.9752738654147106
```

```
****Metrics for fold no: 2 ****
Accuracy: 0.94
Precision: 0.9474206349206349
Recall: 0.9441176470588234
F1 score: 0.9457662571652464

****Metrics for fold no: 3 ****
Accuracy: 0.96
Precision: 0.9696969696969697
Recall: 0.9607843137254902
F1 score: 0.9652200677131424

****FINAL METRICS ACROSS ALL FOLDS****
Accuracy: 0.96
Precision: 0.9644360269360269
Recall: 0.9597536450477627
F1 score: 0.9620867300976998
```







```
In [ ]:
```