Aim: Program to implement k-NN classification using any standard dataset available in the public domain and find the accuracy of the algorithm

Algorithm:

The class of an unknown instance is computed using the following steps:

1. The distance between the unknown instance and all other training instances is computed.
2. The k nearest neighbors are identified.
3. The class labels of the k nearest neighbors are used to determine the class label of the unknown instance by using techniques like majority voting.

```
from sklearn import neighbors, datasets, preprocessing
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
iris=datasets.load_iris()
x=iris.data[:,:]
y=iris.target
x_train,x_test,y_train,y_test=train_test_split(x,y,stratify=y,random_state=42)
scalar=preprocessing.StandardScaler().fit(x_train)
x_train=scalar.transform(x_train)
x_test=scalar.transform(x_test)
x_train



```

```
array([[ 1.79213839, -0.60238047,  1.31532306,  0.92095427],
       [ 2.14531053, -0.60238047,  1.65320421,  1.05135487],
       [-0.4446185 , -1.50797259, -0.03620155, -0.25265117],
       [ 0.26172578, -0.60238047,  0.13273902,  0.13855064],
       [-0.4446185 , -1.28157456,  0.13273902,  0.13855064],
       [ 0.49717388,  0.52960968,  1.25900953,  1.70335789],
       [-1.50413492,  0.75600771, -1.33141264, -1.1654554 ],
       [ 0.49717388, -0.8287785 ,  0.63956075,  0.79055366],
       [-1.26868682,  0.07681362, -1.21878559, -1.295856  ],
       [ 0.37944983, -0.60238047,  0.58324723,  0.79055366],
       [-0.91551468,  1.66159983, -1.04984501, -1.03505479],
       [ 0.61489792, -0.8287785 ,  0.86481486,  0.92095427],
       [-1.03323873, -2.41356471, -0.1488286 , -0.25265117],
       [-0.4446185 ,  2.56719194, -1.33141264, -1.295856  ],
       [-0.79779064,  2.34079391, -1.27509911, -1.42625661],
       [-0.09144636, -0.8287785 ,  0.0764255 ,  0.00815004],
       [ 1.55669029, -0.14958441,  1.14638248,  0.52975245],
       [-0.91551468,  0.98240574, -1.33141264, -1.1654554 ],
       [-0.2091704 ,  3.019988  , -1.27509911, -1.03505479],
       [-0.79779064, -0.8287785 ,  0.0764255 ,  0.26895125],
       [ 1.3212422 ,  0.30321165,  0.5269337 ,  0.26895125],
       [ 1.20351815,  0.07681362,  0.63956075,  0.39935185],
       [-0.68006659,  1.4352018 , -1.27509911, -1.295856  ],
       [-0.32689445, -0.37598244, -0.09251508,  0.13855064],
       [ 0.96807006, -0.14958441,  0.80850133,  1.44255668],
       [-1.15096278, -0.14958441, -1.33141264, -1.295856  ],
       [ 1.08579411,  0.30321165,  1.20269601,  1.44255668],
       [ 0.49717388, -1.73437062,  0.35799313,  0.13855064],
       [ 1.20351815,  0.30321165,  1.09006896,  1.44255668],
       [-1.03323873,  1.20880377, -1.33141264, -1.295856  ],
       [-1.15096278,  0.07681362, -1.27509911, -1.295856  ],
       [ 0.96807006,  0.07681362,  0.5269337 ,  0.39935185],
       [-1.15096278,  0.07681362, -1.27509911, -1.42625661],
       [ 0.49717388, -0.60238047,  0.7521878 ,  0.39935185],
       [-1.15096278,  1.20880377, -1.33141264, -1.42625661],
       [-1.26868682,  0.75600771, -1.04984501, -1.295856  ],
       [-0.32689445, -0.14958441,  0.18905255,  0.13855064],
       [-0.91551468,  0.75600771, -1.27509911, -1.295856  ],
       [ 0.14400174, -0.37598244,  0.41430665,  0.39935185],
       [-1.85730706, -0.14958441, -1.50035321, -1.42625661],
       [-1.26868682, -0.14958441, -1.33141264, -1.42625661],
       [-0.2091704 , -0.14958441,  0.24536608,  0.00815004],
       [-0.09144636, -0.60238047,  0.7521878 ,  1.57295728],
       [-0.09144636, -0.8287785 ,  0.7521878 ,  0.92095427],
       [-1.03323873, -0.14958441, -1.21878559, -1.295856  ],
       [ 0.73262197,  0.30321165,  0.7521878 ,  1.05135487],
       [ 0.61489792, -0.37598244,  0.3016796 ,  0.13855064],
       [-0.91551468,  1.66159983, -1.27509911, -1.1654554 ],
       [ 1.08579411, -0.14958441,  0.97744191,  1.18175547],
       [-1.38641087,  0.30321165, -1.21878559, -1.295856  ],
       [ 0.61489792, -0.60238047,  1.03375543,  1.18175547],
       [ 2.14531053,  1.66159983,  1.65320421,  1.31215608],
       [-0.56234254,  1.88799786, -1.38772616, -1.03505479],
       [ 0.73262197, -0.60238047,  0.47062018,  0.39935185],
       [ 0.49717388, -1.28157456,  0.69587428,  0.92095427],
       [ 2.38075862,  1.66159983,  1.48426364,  1.05135487],
       [ 0.96807006,  0.07681362,  0.35799313,  0.26895125],
       [ 0.49717388, -1.28157456,  0.63956075,  0.39935185],
```

```
# Identify the ideal value for k
scores=[]
for k in range(1,15):
  knn=neighbors.KNeighborsClassifier(n_neighbors=k)
  knn.fit(x_train,y_train)
  y_pred=knn.predict(x_test)
  scores.append(accuracy_score(y_test,y_pred))
  print('when k=%s,accuracy is %s'%(k,accuracy_score(y_test,y_pred)))



```

```
when k=1,accuracy is 0.9473684210526315
when k=2,accuracy is 0.9210526315789473
when k=3,accuracy is 0.9210526315789473
when k=4,accuracy is 0.9210526315789473
when k=5,accuracy is 0.9210526315789473
when k=6,accuracy is 0.9210526315789473
when k=7,accuracy is 0.9473684210526315
when k=8,accuracy is 0.9210526315789473
when k=9,accuracy is 0.9736842105263158
when k=10,accuracy is 0.9736842105263158
when k=11,accuracy is 0.9736842105263158
when k=12,accuracy is 0.9736842105263158
when k=13,accuracy is 0.9736842105263158
when k=14,accuracy is 0.9473684210526315
```

```
# S4.2: Train kNN regressor model for 'k = 6'.

knn6=neighbors.KNeighborsClassifier(n_neighbors=6)
knn6.fit(x_train,y_train)


# Perform prediction using 'predict()' function.
y_pred=knn6.predict(x_test)


# Call the 'score()' function to check the accuracy score of the train set and test set

print("Test set accuracy:",knn6.score(x_test,y_test))
print("confusion matrix")
print(confusion_matrix(y_test,y_pred))
print(classification_report(y_test,y_pred))

```

```
Test set accuracy: 0.9210526315789473
confusion matrix
[[12  0  0]
 [ 0 13  0]
 [ 0  3 10]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        12
           1       0.81      1.00      0.90        13
           2       1.00      0.77      0.87        13

    accuracy                           0.92        38
   macro avg       0.94      0.92      0.92        38
weighted avg       0.94      0.92      0.92        38
```