**U18ISI6204 – Machine Learning Techniques**

**LAB- EXPERIMENT 6**

**NAME:** Navina G A

**ROLL\_NO:** 20BIS028

Implement KNN algorithm using the balanced iris data set for multiclass classification and predict the flower species.

**INTRODUCTION**

In this experiment, we have to perform k nearest neighbor on the iris dataset. The K-NN working can be explained on the basis of the below algorithm:

o **Step-1:** Select the number K of the neighbors

o **Step-2:** Calculate the Euclidean distance of **K number of neighbors**

o **Step-3:** Take the K nearest neighbors as per the calculated Euclidean distance.

o **Step-4:** Among these k neighbors, count the number of the data points in each category.

o **Step-5:** Assign the new data points to that category for which the number of the neighbor is maximum.

# **OBJECTIVE OF THE EXERCISE/EXPERIMENT**

To perform K- nearest neighbor on the given dataset, using scikit library

# **STEP 2: ACQUISITION PROCEDURE:**

**STEP-1:** Start the program.

**STEP-2:** import all the necessary libraries

iv) Numpy – array manipulation

v) Pandas – dataframe manipulation

vi) Matplotlib and seaborn – for data visualization

vii) Sklearn.model\_selection – train test data split and cross\_val\_score

viii) Sklearn.metrics – model evaluation.

ix) Sklearn.datasets – For iris dataset

x) Sklearn.neighbor – For KNeighborsClassifier

**STEP-3:** Loading the dataset using load\_iris method in sklearn.datasets module.

**STEP-4:** Analyze the dataset using info method, which gives its data types and number of non- null values in each columns.

**STEP-5:** Perform basic statistic operation using describe() method.

**STEP-6:** Use heatmaps, correlation matrix, regression plots and pairplots in seaborn to find the relationship between features.

**STEP-7:** Implement KNeighborClassifier with k value ranging from 1 to 25 and save the accuracy score of test dataset for each k value in a score list.

**STEP-8:** Plot the accuracy\_score in y axis and k value in x axis, find out the k value which gives high accuracy on test data.

**STEP-9:** Do the step 7 and 8 for 10-fold validation set.

**STEP-10:** Conclude the best k value which works good in both test and validation set. **STEP-11:** Use that K value to build the final KNN model and print the accuracy\_score. **STEP-12:** Stop the Program.

# **PROGRAM:**

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split, cross\_val\_score

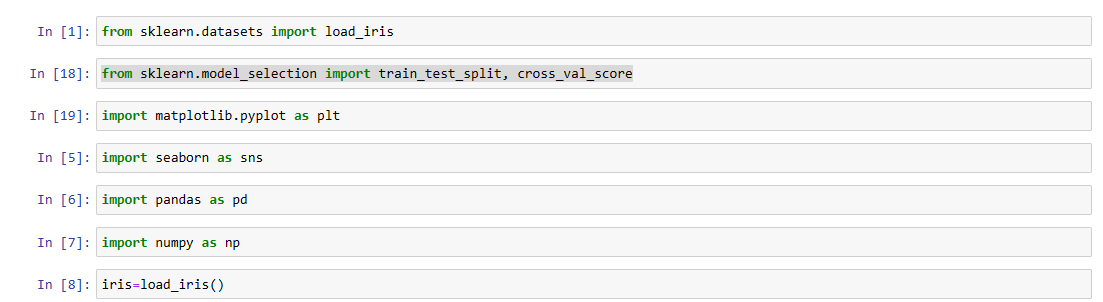
import matplotlib.pyplot as plt

import seaborn as sns

import pandas as pd

import numpy as np

iris=load\_iris()



x=iris.data

y=iris.target

print(x.shape)

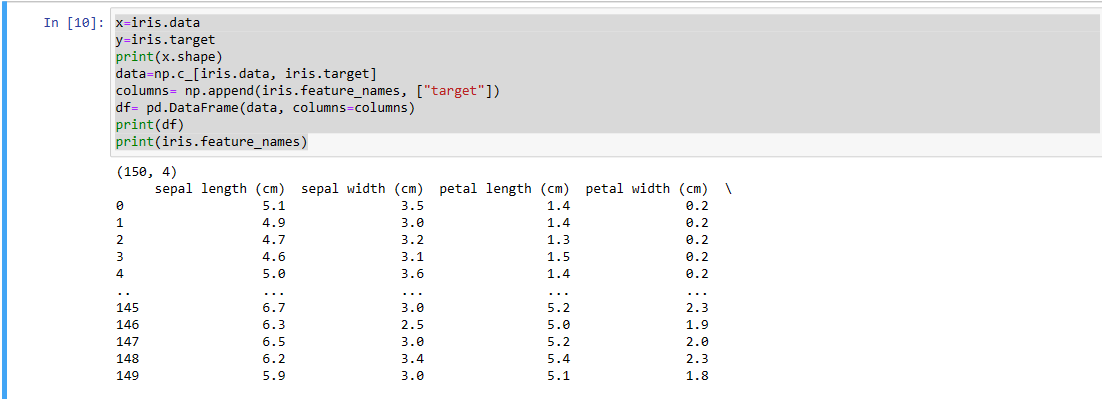
data=np.c\_[iris.data, iris.target]

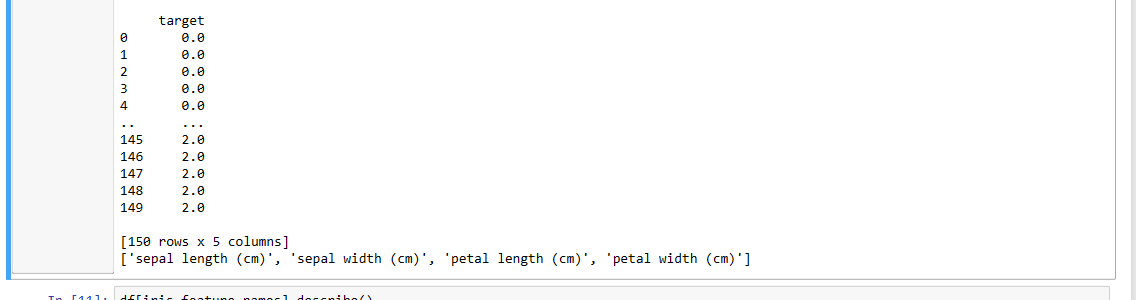
columns= np.append(iris.feature\_names, ["target"])

df= pd.DataFrame(data, columns=columns)

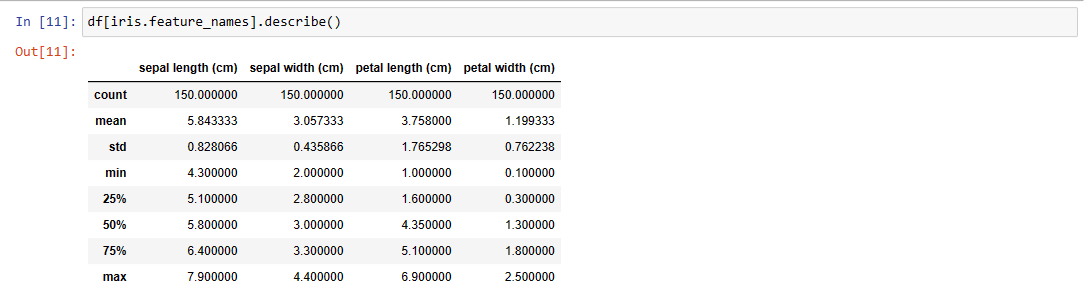
print(df)

print(iris.feature\_names)



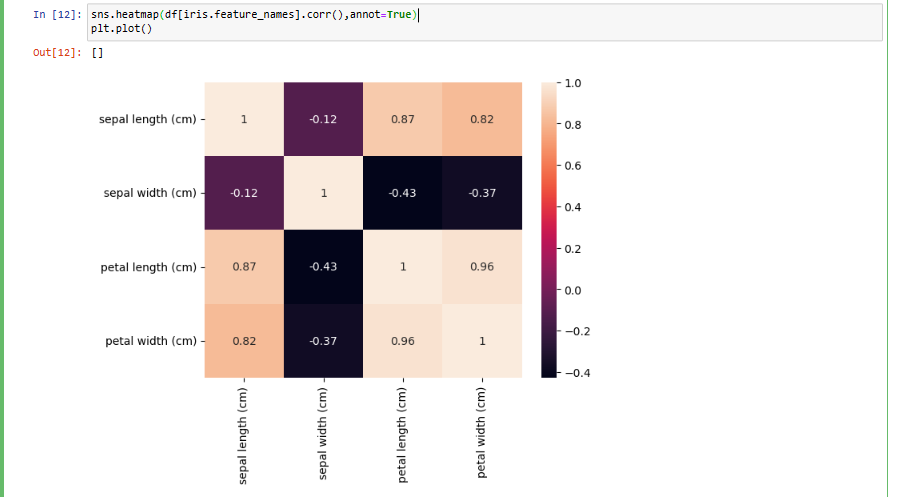


df[iris.feature\_names].describe()



sns.heatmap(df[iris.feature\_names].corr(),annot=True)

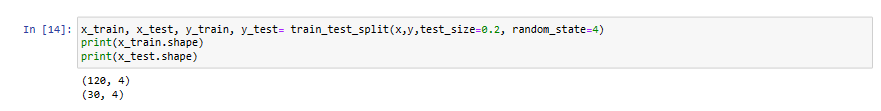
plt.plot()



x\_train, x\_test, y\_train, y\_test= train\_test\_split(x,y,test\_size=0.2, random\_state=4)

print(x\_train.shape)

print(x\_test.shape)



from sklearn.neighbors import KNeighborsClassifier

from sklearn import metrics

test\_k= range(1,26)

scores=[]

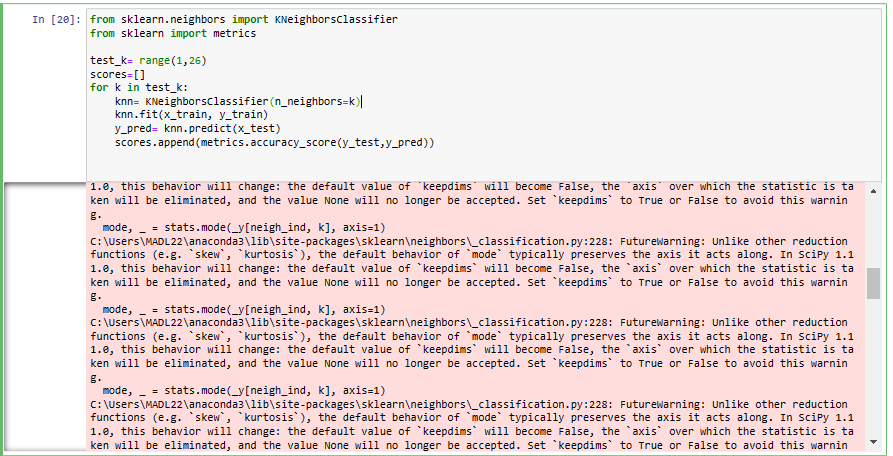
for k in test\_k:

knn= KNeighborsClassifier(n\_neighbors=k)

knn.fit(x\_train, y\_train)

y\_pred= knn.predict(x\_test)

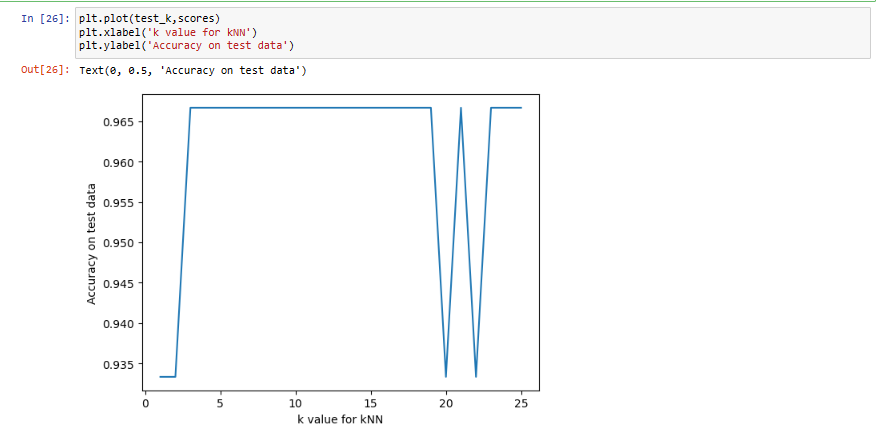
scores.append(metrics.accuracy\_score(y\_test,y\_pred))



plt.plot(test\_k,scores)

plt.xlabel('k value for kNN')

plt.ylabel('Accuracy on test data')



cv\_scores=[]

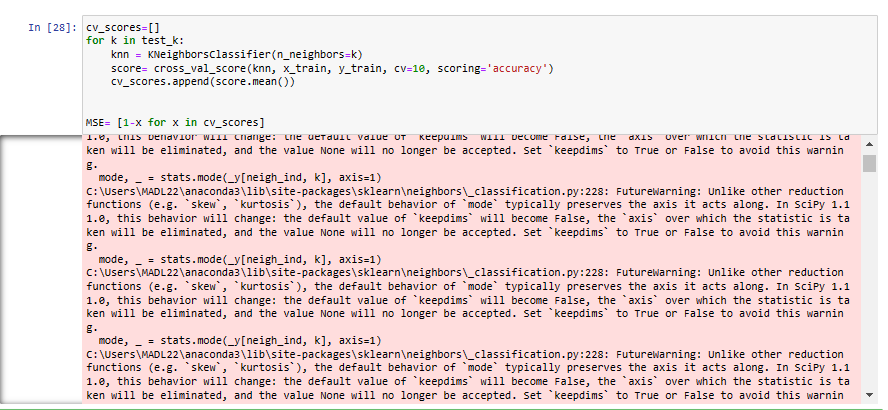
for k in test\_k:

knn = KNeighborsClassifier(n\_neighbors=k)

score= cross\_val\_score(knn, x\_train, y\_train, cv=10, scoring='accuracy')

cv\_scores.append(score.mean())

MSE= [1-x for x in cv\_scores]



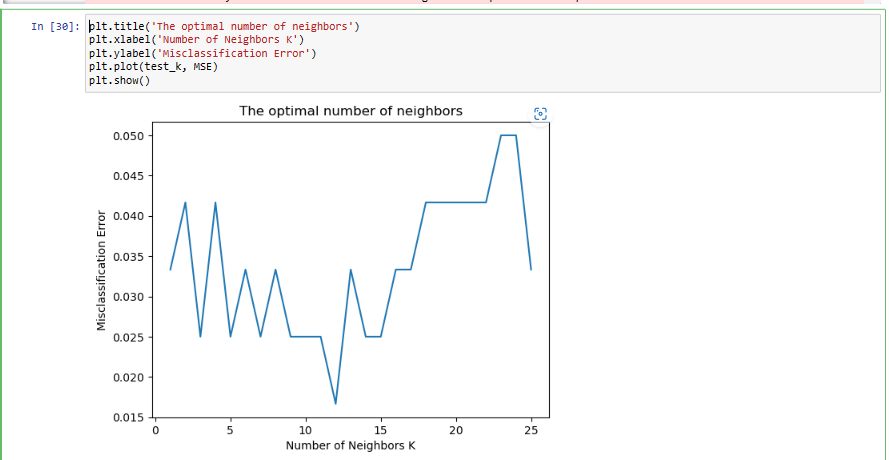
plt.title('The optimal number of neighbors')

plt.xlabel('Number of Neighbors K')

plt.ylabel('Misclassification Error')

plt.plot(test\_k, MSE)

plt.show()



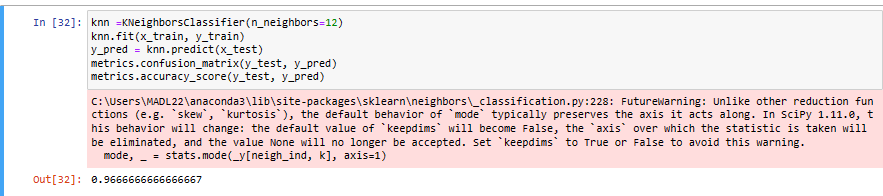
knn =KNeighborsClassifier(n\_neighbors=12)

knn.fit(x\_train, y\_train)

y\_pred = knn.predict(x\_test)

metrics.confusion\_matrix(y\_test, y\_pred)

metrics.accuracy\_score(y\_test, y\_pred)



WITHOUT USING PYTHON LIBRARY

# **KNN implementation on Iris Dataset**

Python · [Iris Flower Dataset](https://www.kaggle.com/datasets/arshid/iris-flower-dataset)

import pandas as pd

import warnings

warnings.filterwarnings("ignore")

import seaborn as sns

import numpy as np

import matplotlib.pyplot as plt

sns.set(style="white", color\_codes=True)

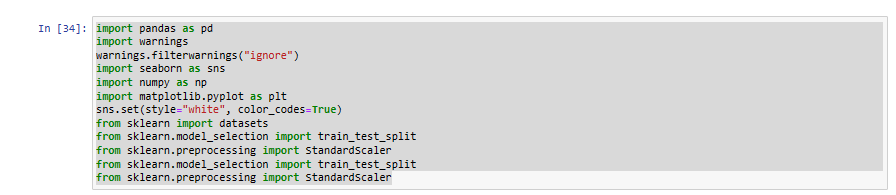
from sklearn import datasets

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler



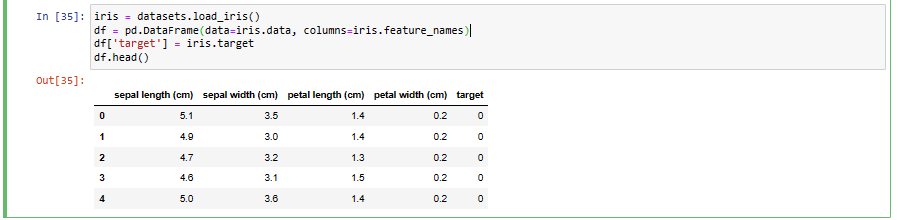
LOADING DATASET

iris = datasets.load\_iris()

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

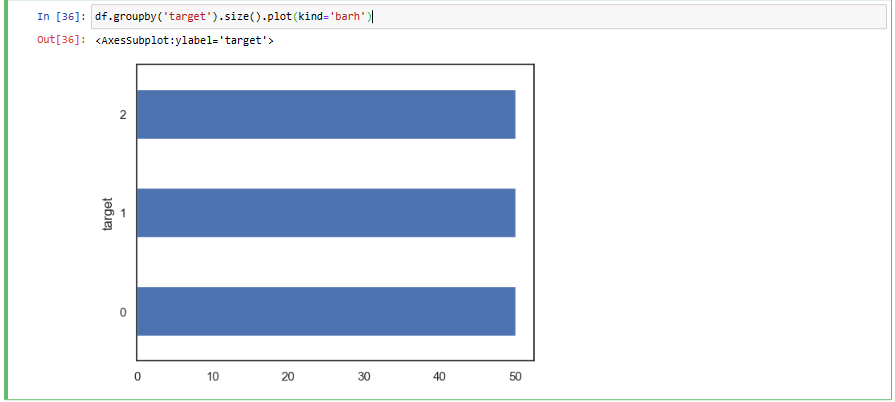
df['target'] = iris.target

df.head()



# Checking if the dataset is balanced or not

df.groupby('target').size().plot(kind='barh')



# Euclidean distance function

def dis(a, b, p=1):

l = len(a)

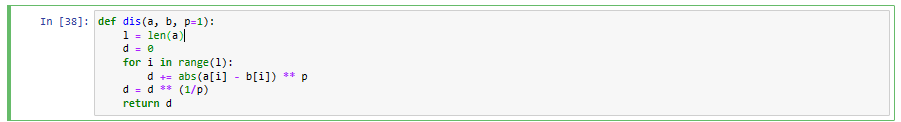
d = 0

for i **in** range(l):

d += abs(a[i] - b[i]) \*\* p

d = d \*\* (1/p)

return d



X = df.drop('target', axis=1)

y = df.target

test\_pt = [4.8, 2.7, 2.5, 0.7]

distances = []

for i **in** X.index:

a = dis(test\_pt, X.iloc[i])

distances.append(a)

dists = pd.DataFrame(data=distances, index=X.index, columns=['dist'])

dists.head()



# Distance DataFrame is sorted to measure which class the nearest

def knn\_sort(k,dists): return dists.sort\_values(by = 'dist')[:k]

# 

# 

# Value of k is determined.[¶](https://www.kaggle.com/code/abirhasan1703100/knn-implementation-on-iris-dataset#Value-of-k-is-determined.)

sorted\_dists = knn\_sort(5, dists)

print(sorted\_dists)

count\_set = {}

for i in sorted\_dists.index:

if y[i] not in count\_set:

count\_set[y[i]] = 1

else:

count\_set[y[i]] += 1

print(max(count\_set))



# Split the data - 75% train, 25% test

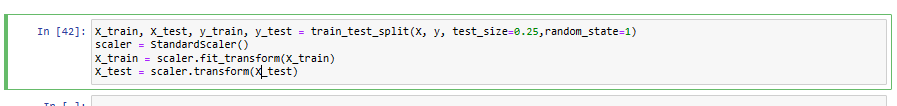
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25,random\_state=1)

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

# Training and predicting the test set and checking accuracy.



def KNN(X\_train, X\_test, y\_train, y\_test, k, p):

y\_predict = []

for test\_pt **in** X\_test:

distances = []

for i **in** X\_train:

a = dis(test\_pt, i, p)

distances.append(a)

dists = pd.DataFrame(data=distances, index=y\_train.index, columns=['dist'])

sorted\_dists = knn\_sort(k, dists)

*#print(sorted\_dists)*

count\_set = {}

for i **in** sorted\_dists.index:

if y\_train[i] **not** **in** count\_set:

count\_set[y\_train[i]] = 1

else:

count\_set[y\_train[i]] += 1

y\_predict.append(max(count\_set))

y = y\_test.tolist()

accr = 0

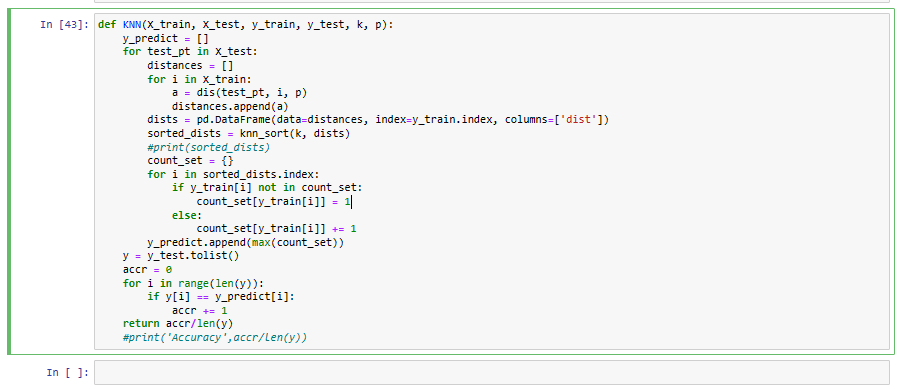
for i **in** range(len(y)):

if y[i] == y\_predict[i]:

accr += 1

return accr/len(y)

*#print('Accuracy',accr/len(y))*



# Calling the function

KNN(X\_train, X\_test, y\_train, y\_test, 5,1)



ACCURACY:

accuracies = []

for i **in** range(1,100):

accuracies.append(KNN(X\_train, X\_test, y\_train, y\_test, i,1))

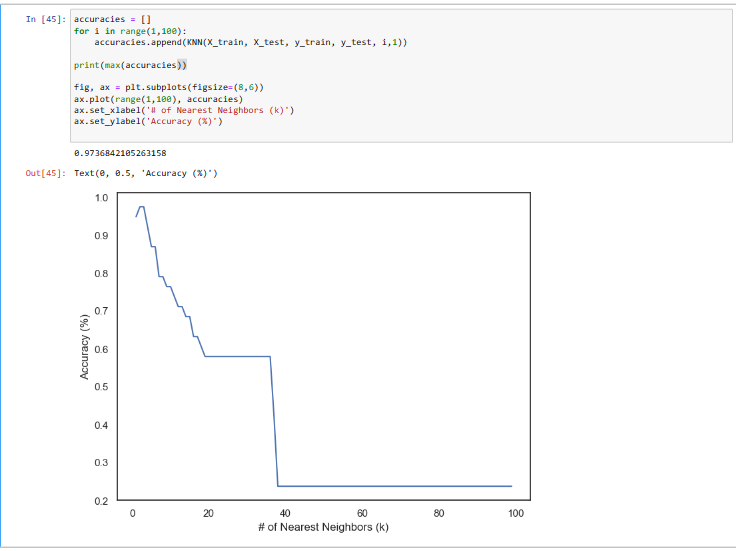
print(max(accuracies))

fig, ax = plt.subplots(figsize=(8,6))

ax.plot(range(1,100), accuracies)

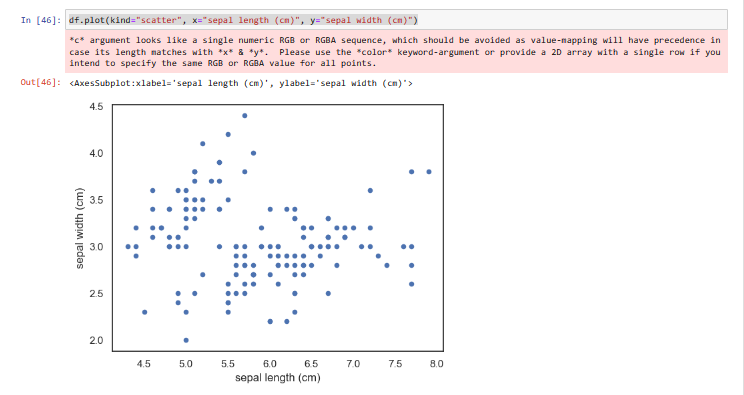
ax.set\_xlabel('# of Nearest Neighbors (k)')

ax.set\_ylabel('Accuracy (%)')



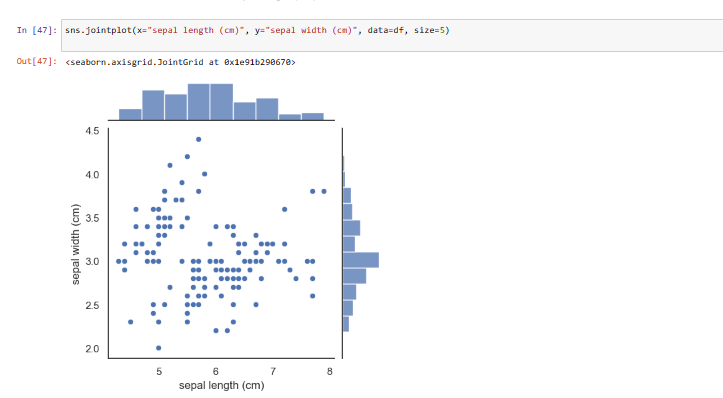
# Data Visualization

df.plot(kind="scatter", x="sepal length (cm)", y="sepal width (cm)")



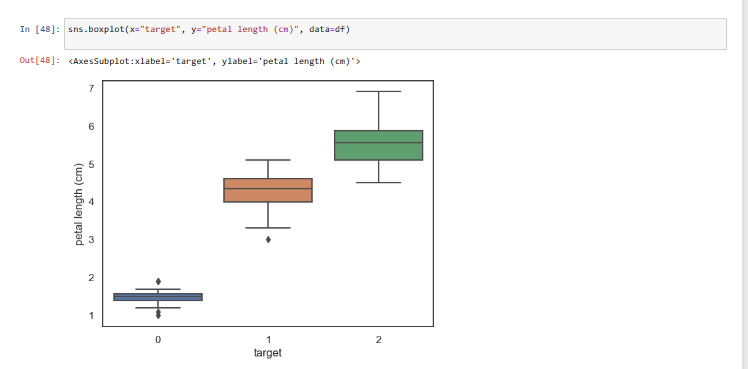
sns.jointplot(x="sepal length (cm)", y="sepal width (cm)", data=df, size=5)

Out[13]:



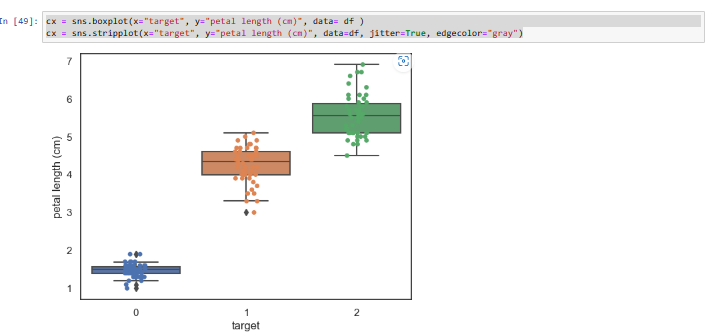
sns.boxplot(x="target", y="petal length (cm)", data=df)

Out[14]:



cx = sns.boxplot(x="target", y="petal length (cm)", data= df )

cx = sns.stripplot(x="target", y="petal length (cm)", data=df, jitter=True, edgecolor="gray")



K=2

sorted\_dists = knn\_sort(2, dists)

print(sorted\_dists)

count\_set = {}

for i in sorted\_dists.index:

if y[i] not in count\_set:

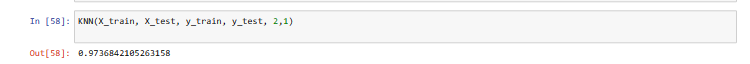
count\_set[y[i]] = 1

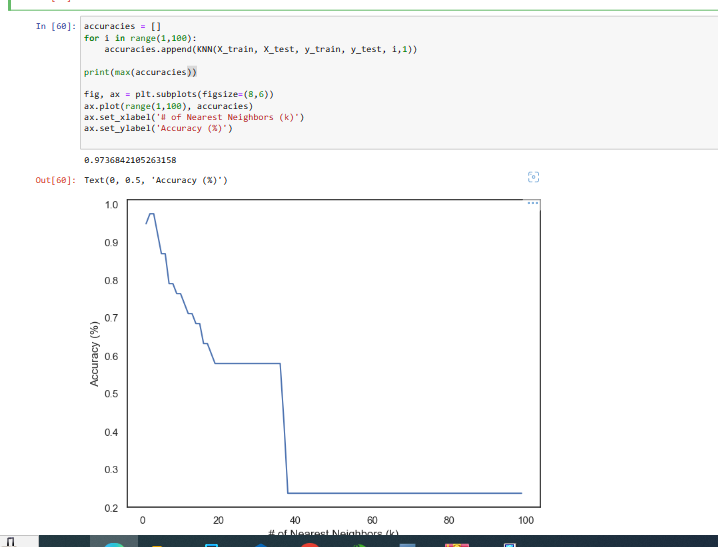
else:

count\_set[y[i]] += 1

print(max(count\_set))







K=3



