U18ISI6204 – Machine Learning Techniques

LAB- EXPERIMENT 6

NAME: Aaron Mathew

ROLL_NO: 20BIS001

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:

- **Step-1:** Select the number K of the neighbors
- **Step-2:** Calculate the Euclidean distance of **K number of neighbors**
- **Step-3:** Take the K nearest neighbors as per the calculated Euclidean distance.
- **Step-4:** Among these k neighbors, count the number of the data points in each category.
- **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()
```

```
In [1]: from sklearn.datasets import load_iris

In [18]: from sklearn.model_selection import train_test_split, cross_val_score

In [19]: import matplotlib.pyplot as plt

In [5]: import seaborn as sns

In [6]: import pandas as pd

In [7]: import numpy as np

In [8]: 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)
```

In [10]:
```python
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)
```

```
(150, 4)
     sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0                  5.1               3.5                1.4               0.2
1                  4.9               3.0                1.4               0.2
2                  4.7               3.2                1.3               0.2
3                  4.6               3.1                1.5               0.2
4                  5.0               3.6                1.4               0.2
..                 ...               ...                ...               ...
145                6.7               3.0                5.2               2.3
146                6.3               2.5                5.0               1.9
147                6.5               3.0                5.2               2.0
148                6.2               3.4                5.4               2.3
149                5.9               3.0                5.1               1.8
```

```
     target
0       0.0
1       0.0
2       0.0
3       0.0
4       0.0
..      ...
145     2.0
146     2.0
147     2.0
148     2.0
149     2.0

[150 rows x 5 columns]
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
```

df[iris.feature_names].describe()

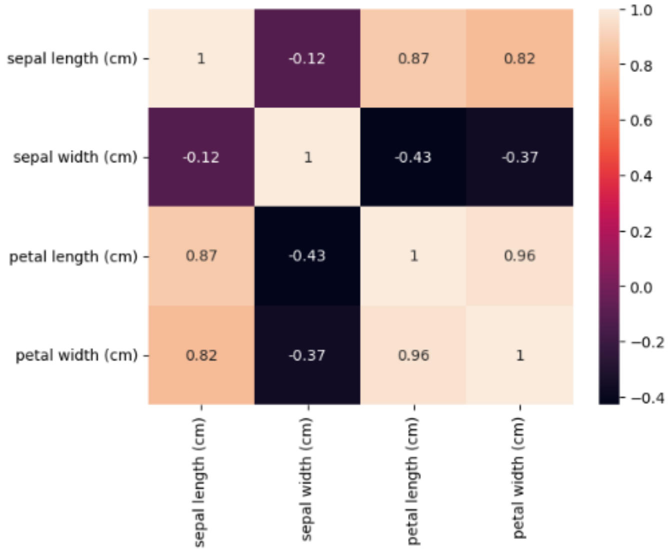
In [11]:
```python
df[iris.feature_names].describe()
```

Out[11]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

sns.heatmap(df[iris.feature_names].corr(),annot=True)
plt.plot()

```
In [12]: sns.heatmap(df[iris.feature_names].corr(),annot=True)
         plt.plot()
Out[12]: []
```



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)

```
In [14]: 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)

         (120, 4)
         (30, 4)
```

from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics

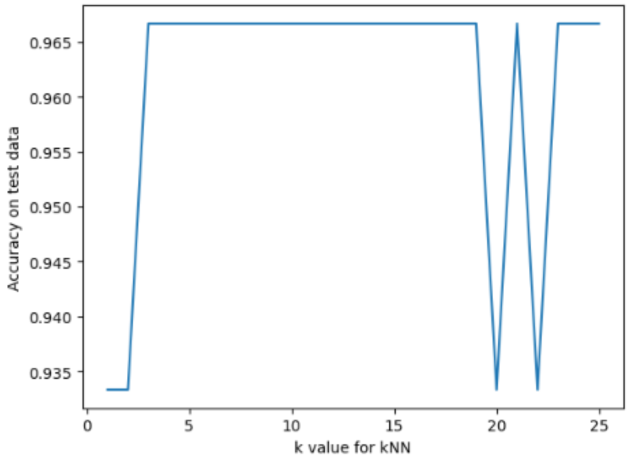
```
test_k= range(1,26)
scores=[]
for k in test_k:
    knn= KNeighborsClassifier(n_neighbors=k)
    knn.fit(x_train, y_train)
    y_pred= knn.predict(x_test)
    scores.append(metrics.accuracy_score(y_test,y_pred))
```

```
In [20]: 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))
```

```
1.0, this behavior will change: the default value of `keepdims` will become False, the `axis` over which the statistic is ta
ken will be eliminated, and the value None will no longer be accepted. Set `keepdims` to True or False to avoid this warnin
g.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
C:\Users\MADL22\anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:228: FutureWarning: Unlike other reduction
functions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts along. In SciPy 1.1
1.0, this behavior will change: the default value of `keepdims` will become False, the `axis` over which the statistic is ta
ken will be eliminated, and the value None will no longer be accepted. Set `keepdims` to True or False to avoid this warnin
g.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
C:\Users\MADL22\anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:228: FutureWarning: Unlike other reduction
functions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts along. In SciPy 1.1
1.0, this behavior will change: the default value of `keepdims` will become False, the `axis` over which the statistic is ta
ken will be eliminated, and the value None will no longer be accepted. Set `keepdims` to True or False to avoid this warnin
g.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
C:\Users\MADL22\anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:228: FutureWarning: Unlike other reduction
functions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts along. In SciPy 1.1
1.0, this behavior will change: the default value of `keepdims` will become False, the `axis` over which the statistic is ta
ken will be eliminated, and the value None will no longer be accepted. Set `keepdims` to True or False to avoid this warnin
```

plt.plot(test_k,scores)
plt.xlabel('k value for kNN')
plt.ylabel('Accuracy on test data')

```
In [26]: plt.plot(test_k,scores)
         plt.xlabel('k value for kNN')
         plt.ylabel('Accuracy on test data')

Out[26]: Text(0, 0.5, '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())
```

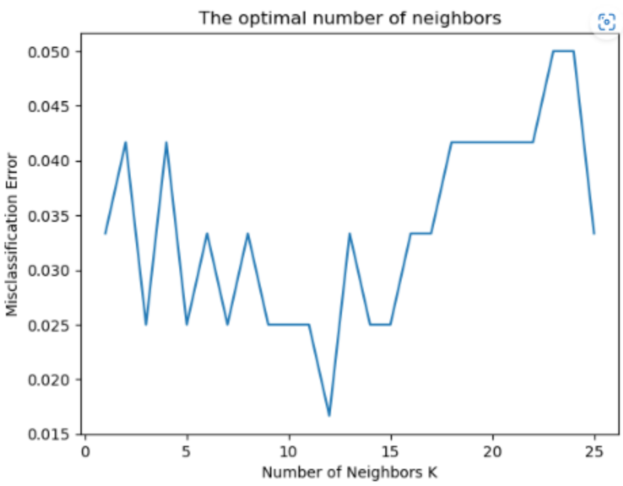
MSE= [1-x for x in cv_scores]

```
In [28]: 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]
```

```
1.0, this behavior will change: the default value of `keepdims` will become False, the `axis` over which the statistic is ta
ken will be eliminated, and the value None will no longer be accepted. Set `keepdims` to True or False to avoid this warnin
g.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
C:\Users\MADL22\anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:228: FutureWarning: Unlike other reduction
functions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts along. In SciPy 1.1
1.0, this behavior will change: the default value of `keepdims` will become False, the `axis` over which the statistic is ta
ken will be eliminated, and the value None will no longer be accepted. Set `keepdims` to True or False to avoid this warnin
g.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
C:\Users\MADL22\anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:228: FutureWarning: Unlike other reduction
functions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts along. In SciPy 1.1
1.0, this behavior will change: the default value of `keepdims` will become False, the `axis` over which the statistic is ta
ken will be eliminated, and the value None will no longer be accepted. Set `keepdims` to True or False to avoid this warnin
g.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
C:\Users\MADL22\anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:228: FutureWarning: Unlike other reduction
functions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts along. In SciPy 1.1
1.0, this behavior will change: the default value of `keepdims` will become False, the `axis` over which the statistic is ta
ken will be eliminated, and the value None will no longer be accepted. Set `keepdims` to True or False to avoid this warnin
```

plt.title('The optimal number of neighbors')
plt.xlabel('Number of Neighbors K')
plt.ylabel('Misclassification Error')
plt.plot(test_k, MSE)
plt.show()

```
In [30]: 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)

```
In [32]: 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)

C:\Users\MADL22\anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:228: FutureWarning: Unlike other reduction fun
ctions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts along. In SciPy 1.11.0, t
his behavior will change: the default value of `keepdims` will become False, the `axis` over which the statistic is taken will
be eliminated, and the value None will no longer be accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)

Out[32]: 0.9666666666666667
```

WITHOUT USING PYTHON LIBRARY

KNN implementation on Iris Dataset

Python · 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
```


```
In [34]: 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()
```

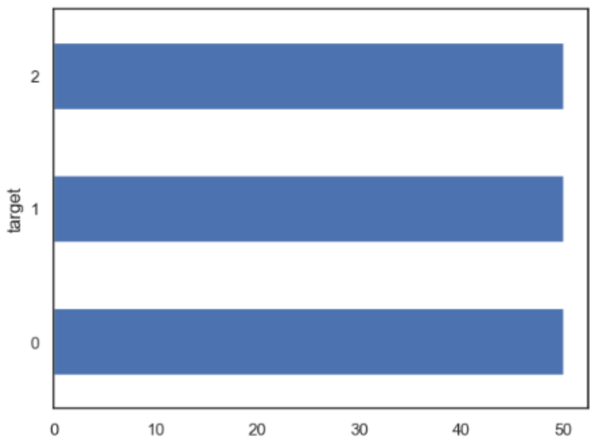
```
In [35]: iris = datasets.load_iris()
         df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
         df['target'] = iris.target
         df.head()
Out[35]:
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

Checking if the dataset is balanced or not

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

```
In [36]: df.groupby('target').size().plot(kind='barh')
Out[36]: <AxesSubplot:ylabel='target'>
```



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
```

```
In [38]: 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()
```

```
In [39]: 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()
```

Out[39]:

	dist
0	2.7
1	2.0
2	2.3
3	2.1
4	2.7

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.¶

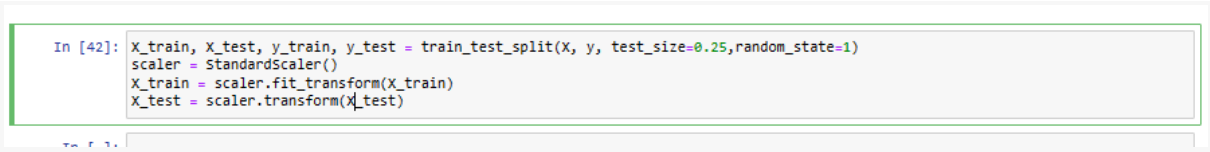
```
sorted_dists = knn_sort(5, dists)
print(sorted_dists)

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.



```
In [42]: 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)
```

```
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))
```

```
In [43]: 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))

In [ ]:
```

Calling the function

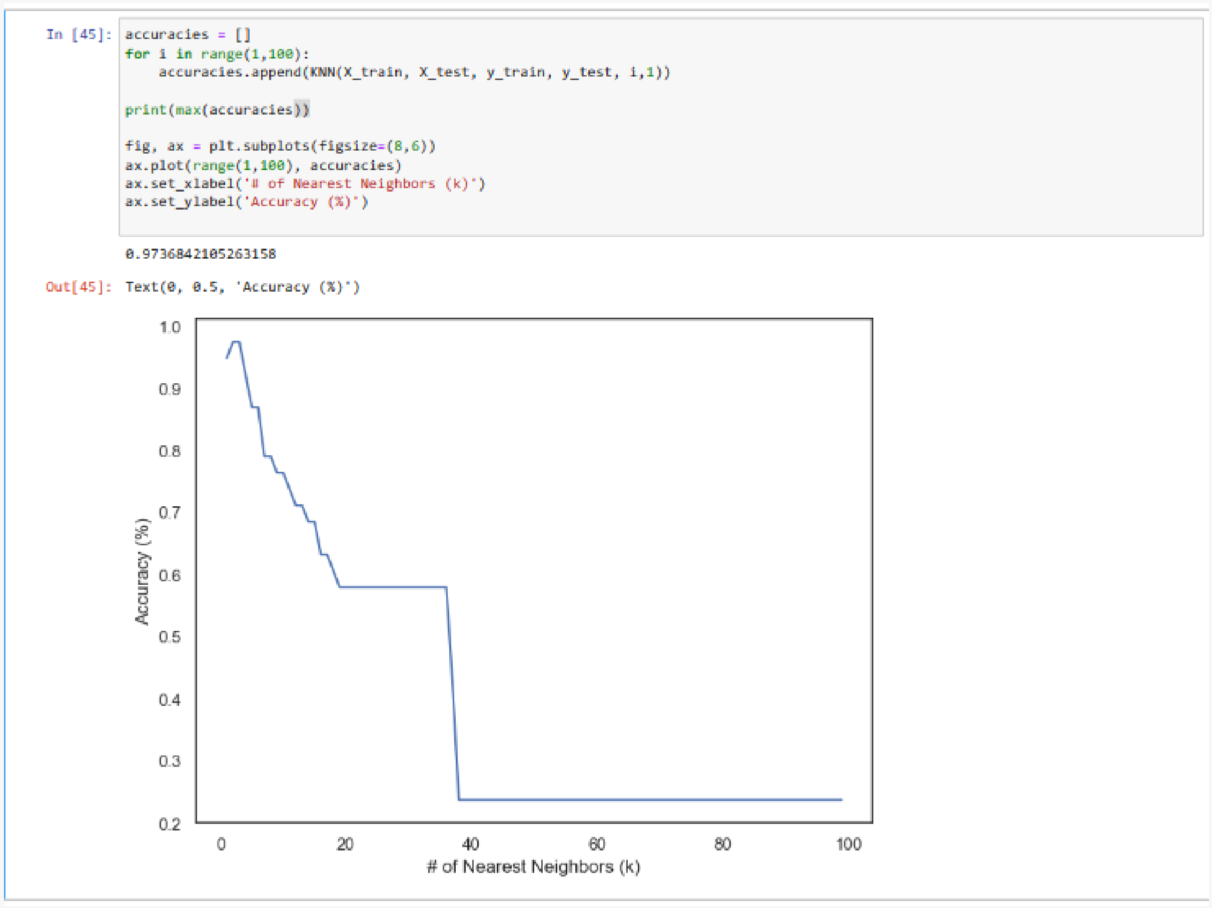
```
KNN(X_train, X_test, y_train, y_test, 5,1)
```

```
In [44]: KNN(X_train, X_test, y_train, y_test, 5,1)

Out[44]: 0.868421052631579

In [ ]:
```

ACCURACY:

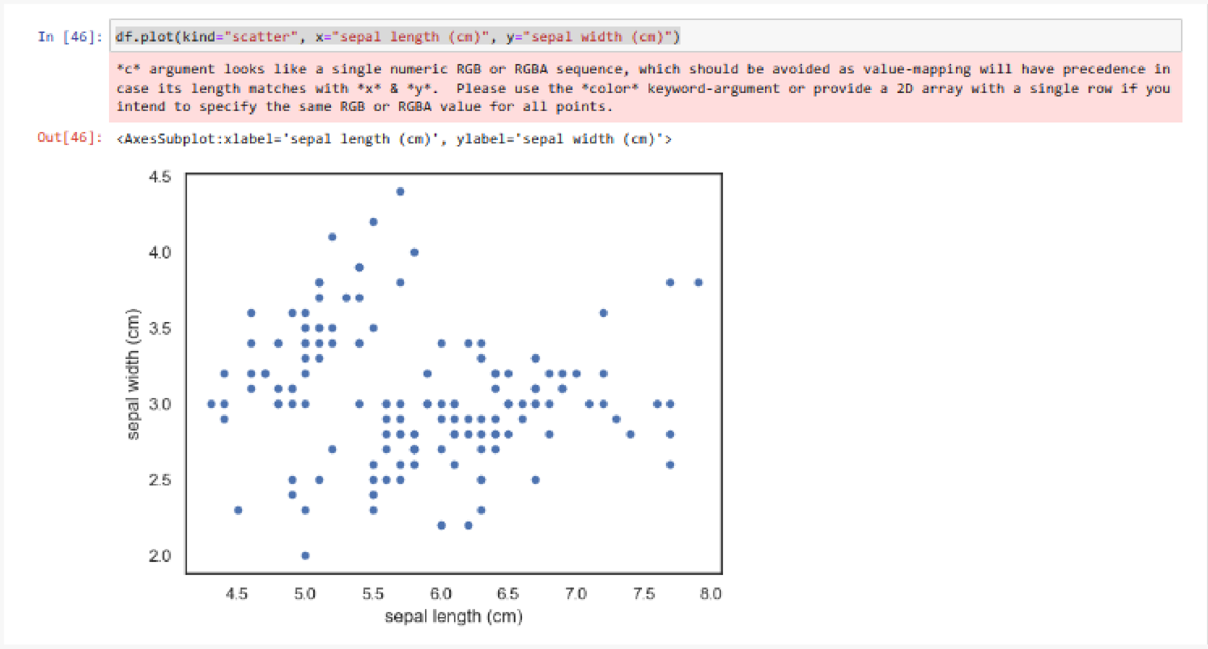
```
accuracies = []
for i in range(1,100):
    accuracies.append(KNN(X_train, X_test, y_train, y_test, i,1))
```

```
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 (%)')
```



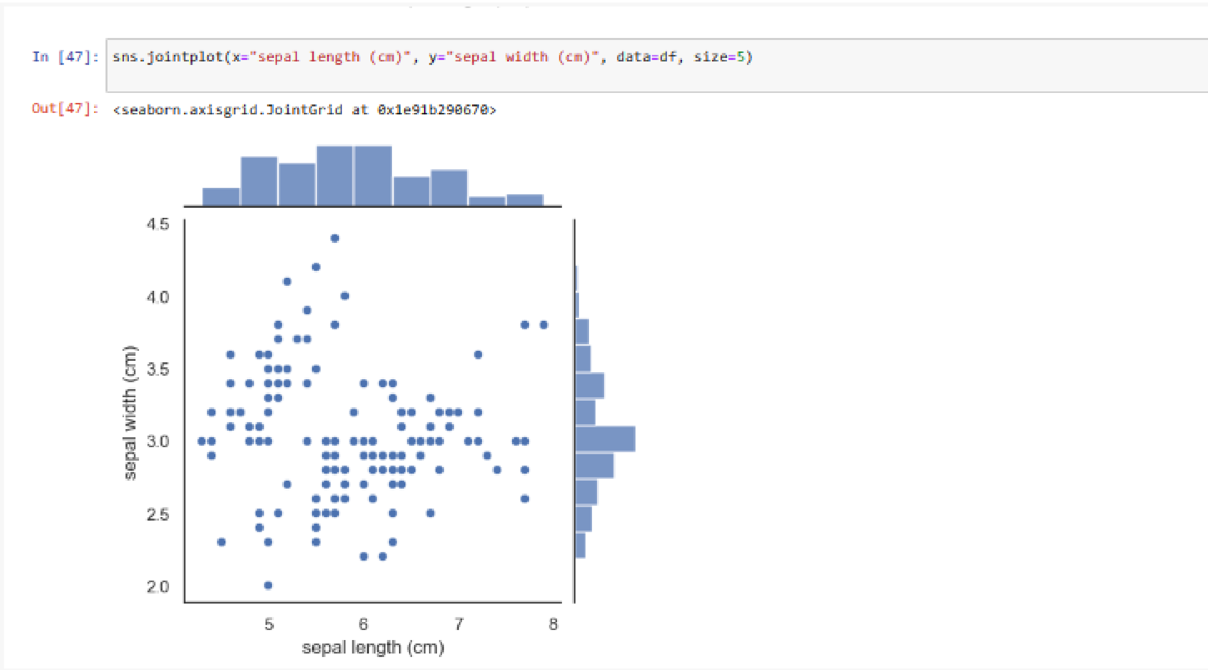
Data Visualization

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



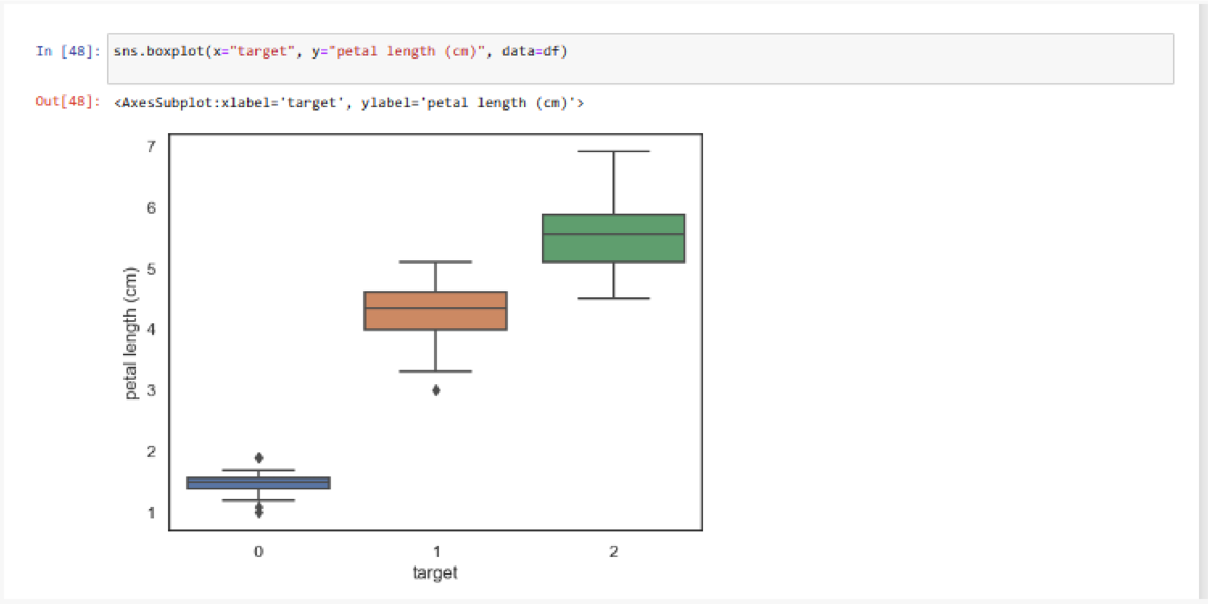

```
sns.jointplot(x="sepal length (cm)", y="sepal width (cm)", data=df,
size=5)
```

Out[13]:

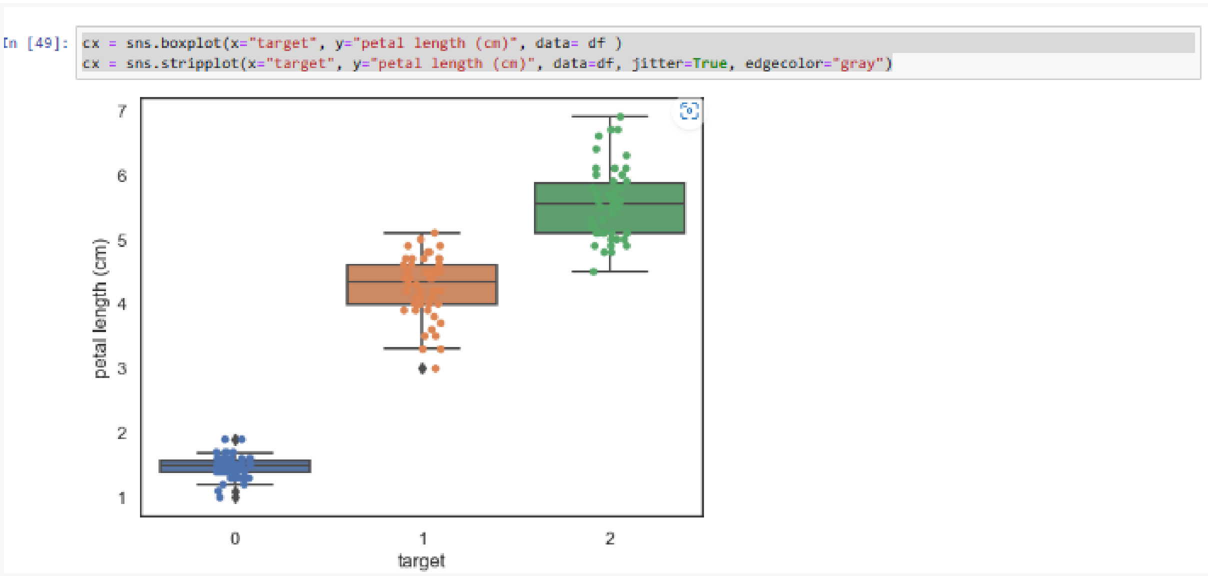



```
sns.boxplot(x="target", y="petal length (cm)", data=df)
```

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")
```

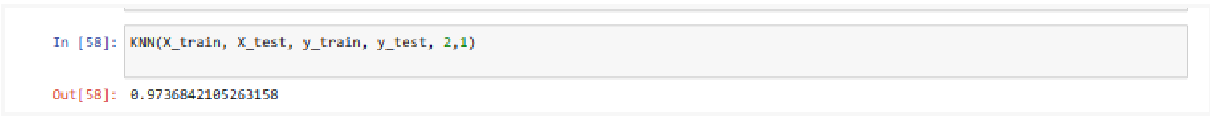


K=2

```
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))
```

```
In [40]: def knn_sort(k,dists): return dists.sort_values(by = 'dist')[:k]

In [50]: 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))

             dist
         98   1.4
         57   1.5
         1
```

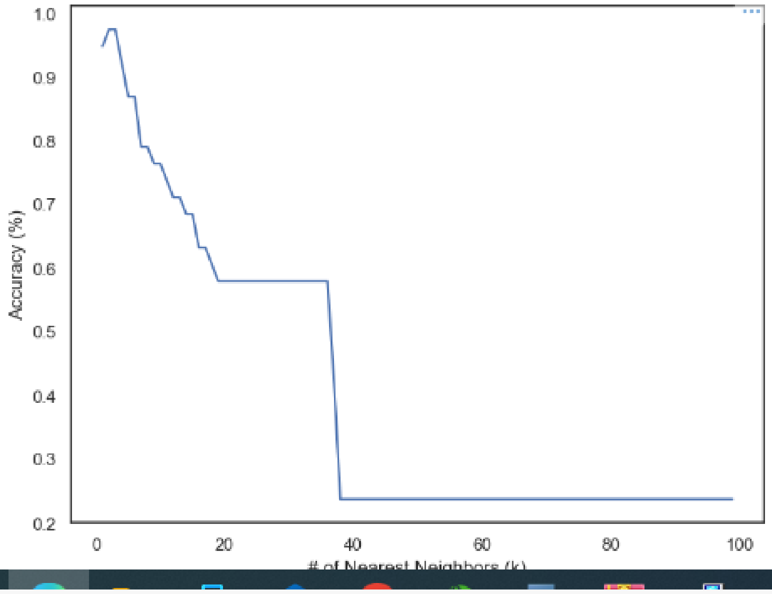


```
In [58]: KNN(X_train, X_test, y_train, y_test, 2,1)

Out[58]: 0.9736842105263158
```

In [60]:
```
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 (%)')
```

```
0.9736842105263158
```

Out[60]: `Text(0, 0.5, 'Accuracy (%)')`



K=3

In [63]:
```
sorted_dists = knn_sort(3, 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))
```

```
    dist
98   1.4
57   1.5
93   1.7
1
```

In [64]:
```
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)
```

In [65]:
```
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))
```

In [66]:
```
KNN(X_train, X_test, y_train, y_test, 3,1)
```

Out[66]: 0.9736842105263158

In [67]:
```
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 (%)')
```

0.9736842105263158

Out[67]: Text(0, 0.5, 'Accuracy (%)')