Imports

In [103…

```
import pandas as pd
import numpy as np
import knn
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import r2_score
from sklearn.model_selection import StratifiedShuffleSplit
from sklearn.neighbors import KNeighborsClassifier
```

KNN Classifier

In [136…

```
from scipy.stats import mode
from scipy.spatial import distance as dd
from math import sqrt

def l3distance(point1, point2):
    return dd.minkowski(point1, point2, 3) # p = 3

def manDistance(point1, point2):
    return np.abs(point1[:,None] - point2).sum(-1)

def eucledianDistance(point1, point2):
    dist = 0;
    for i in range(len(point1)):
        ai = point1[i];
        bi = point2[i]
        result = (ai-bi)**2
        dist += result
    return sqrt(dist)

def KNNPredict(x_train, y_train , x_test, k, majority = True, distFunc = 'ED'):
    classLabels = []


    for example in x_test:

        # Storing Euclidean distance from "example" point in test data to all ot
        distToAllPts = np.array([])
        for i in range(len(x_train)):
            distance = eucledianDistance(example, np.array(x_train[i, :]))
            distToAllPts = np.append(distToAllPts, distance)

        # Sorting np array distToAllPts (closest->furthest)
        dist = np.argsort(distToAllPts)

        # Getting the first k elements within array
        dist = dist[:k]

        # Getting associated label of each data point in dist and storing it in
        labels = y_train[dist]
```

```
        if (majority == True):
            # Sort labels by mode (most occurrences -> least occurrences)
            newLabel = mode(labels)

            # Get first element (majority voting, one with most occurrences)
            newLabel = newLabel.mode[0]
        else:
            newLabel = np.mean(labels)

        # Append label to final class labels
        classLabels.append(newLabel)

    return classLabels
```

Iris Dataset

Loading Iris Dataset into Data Frame

In [123…

```
irisData = pd.read_csv("iris.csv")
```

Encoding class labels

In [124…

```
irisData.replace({"Iris-versicolor": 0, "Iris-setosa": 1, "Iris-virginica": 2},
```

Assigning meaningful column names accordingly

In [124…

```
irisData = irisData.rename(columns={'5.1':'sepal length', '3.5':'sepal width', '
```

Current Dataset

In [124…

```
irisData
```

Out[124…

	sepal length	sepal width	petal length	petal width	class label
0	4.9	3.0	1.4	0.2	1
1	4.7	3.2	1.3	0.2	1
2	4.6	3.1	1.5	0.2	1
3	5.0	3.6	1.4	0.2	1
4	5.4	3.9	1.7	0.4	1
...	...	...	...	...	...
144	6.7	3.0	5.2	2.3	2
145	6.3	2.5	5.0	1.9	2
146	6.5	3.0	5.2	2.0	2
147	6.2	3.4	5.4	2.3	2

	sepal length	sepal width	petal length	petal width	class label
148	5.9	3.0	5.1	1.8	2

149 rows × 5 columns

Splitting data into training and testing data

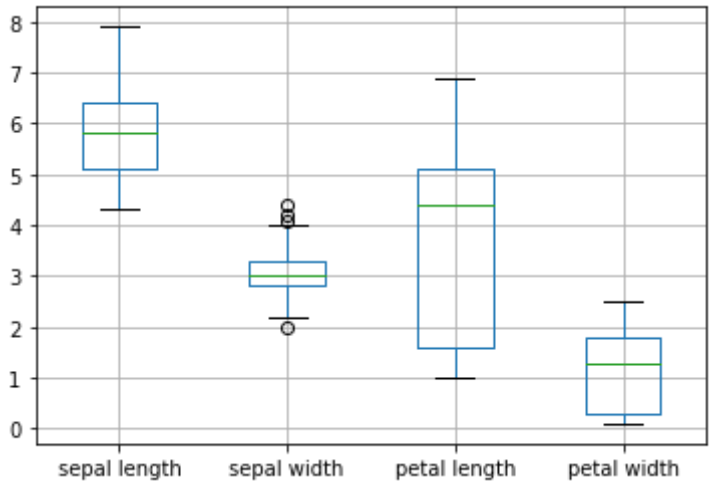
In [124…

```
def getSplit(train, test):
    # We convert all pandas sub-dataframes to numpy arrays so we can perform cla
    x_train = train.iloc[:,:-1].to_numpy()
    y_train = train.iloc[:,-1:].to_numpy()

    x_test = test.iloc[:,:-1].to_numpy() # iloc here gets all columns except the
    y_test = test.iloc[:,-1:].to_numpy() # iloc here gets the last column (class

    # When we convert "class label" column into a numpy array, we get a 2D array
    # So, since we want it to be a 1D array of class labels, we "squeeze" it fro
    y_train = np.squeeze(y_train)
    y_test = np.squeeze(y_test)

    return x_train,x_test,y_train,y_test
```

(A)

In [124…

```
boxplot = irisData.boxplot(column=['sepal length', 'sepal width', 'petal length'
```



(B)

In [129…

```
from sklearn.metrics import confusion_matrix
first2Features = irisData
train, test = train_test_split(first2Features, test_size=0.20, stratify=irisData
x_train, x_test, y_train, y_test = getSplit(train, test)
```

In [130…

```
y_pred = KNNPredict(x_train, y_train, x_test, 1)
print("Accuracy: " + str(r2_score(y_pred, y_test)))
```

```
print("Confusion Matrix: ")
print(confusion_matrix(y_test, y_pred, labels=[0,1,2]))
```

```
Accuracy: 0.7986577181208054
Confusion Matrix:
[[10  0  0]
 [ 0 10  0]
 [ 1  0  9]]
```

(C)

In [130…

```
y_pred = KNNPredict(x_train, y_train, x_test, 2)
print("Accuracy: " + str(r2_score(y_pred, y_test)))
print("Confusion Matrix: ")
print(confusion_matrix(y_test, y_pred, labels=[0,1,2]))
```

```
Accuracy: 0.7986577181208054
Confusion Matrix:
[[10  0  0]
 [ 0 10  0]
 [ 1  0  9]]
```

In [130…

```
y_pred = KNNPredict(x_train, y_train, x_test, 4)
print("Accuracy: " + str(r2_score(y_pred, y_test)))
print("Confusion Matrix: ")
print(confusion_matrix(y_test, y_pred, labels=[0,1,2]))
```

```
Accuracy: 0.7986577181208054
Confusion Matrix:
[[10  0  0]
 [ 0 10  0]
 [ 1  0  9]]
```

In [130…

```
y_pred = KNNPredict(x_train, y_train, x_test, 6)
print("Accuracy: " + str(r2_score(y_pred, y_test)))
print("Confusion Matrix: ")
print(confusion_matrix(y_test, y_pred, labels=[0,1,2]))
```

```
Accuracy: 0.7986577181208054
Confusion Matrix:
[[10  0  0]
 [ 0 10  0]
 [ 1  0  9]]
```

In [130…

```
y_pred = KNNPredict(x_train, y_train, x_test, 8)
print("Accuracy: " + str(r2_score(y_pred, y_test)))
print("Confusion Matrix: ")
print(confusion_matrix(y_test, y_pred, labels=[0,1,2]))
```

```
Accuracy: 1.0
Confusion Matrix:
[[10  0  0]
 [ 0 10  0]
 [ 0  0 10]]
```

In [130…

```
y_pred = KNNPredict(x_train, y_train, x_test, 10)
print("Accuracy: " + str(r2_score(y_pred, y_test)))
```

```
print("Confusion Matrix: ")
print(confusion_matrix(y_test, y_pred, labels=[0,1,2]))
```

```
Accuracy: 1.0
Confusion Matrix:
[[10  0  0]
 [ 0 10  0]
 [ 0  0 10]]
```

In [129…

```
# By increasing the number of neighbors, the prediction is much more
# accurate and, thus, the decision boundaries fit the data much better
```

(D)

In [123…

```
# Done
```

Bank Notes Dataset

In [138…

```
bankData = pd.read_csv("data_banknote_authentication.csv")
bankData = bankData.rename(columns={'3.6216':'Variance of Wavelet Transformed im
```

(A)

In [138…

```
features = bankData
train, test = train_test_split(features, test_size=0.20, stratify=bankData["clas
x_train, x_test, y_train, y_test = getSplit(train, test)
y_pred = KNNPredict(x_train, y_train, x_test, 2)
print("Accuracy: " + str(r2_score(y_pred, y_test)))
print("Confusion Matrix: ")
print(confusion_matrix(y_test, y_pred, labels=[0,1]))
```

```
Accuracy: 1.0
Confusion Matrix:
[[153   0]
 [  0 122]]
```

(B)

In [138…

```
y_pred = KNNPredict(x_train, y_train, x_test, 2, majority=False) # Majority = Fa
print("Accuracy: " + str(r2_score(y_pred, y_test)))
print("Confusion Matrix: ")
print(confusion_matrix(y_test, y_pred, labels=[0,1]))
```

```
Accuracy: 1.0
Confusion Matrix:
[[153   0]
 [  0 122]]
```

In [ ]:

```
# It did not affect the error rate
```

(C)

In [139…
```
y_pred = KNNPredict(x_train, y_train, x_test, 2, majority=False, distFunc = 'MD'
print("Accuracy: " + str(r2_score(y_pred, y_test)))
print("Confusion Matrix: ")
print(confusion_matrix(y_test, y_pred, labels=[0,1]))
```

```
Accuracy: 1.0
Confusion Matrix:
[[153   0]
 [  0 122]]
```

In [139…
```
y_pred = KNNPredict(x_train, y_train, x_test, 2, majority=False, distFunc = 'LD'
print("Accuracy: " + str(r2_score(y_pred, y_test)))
print("Confusion Matrix: ")
print(confusion_matrix(y_test, y_pred, labels=[0,1]))
```

```
Accuracy: 1.0
Confusion Matrix:
[[153   0]
 [  0 122]]
```

In [ ]:
```
# Did not change the error rate
```

MNIST dataset

In [138…
```
train = pd.read_csv("mnist_train.csv")
test = pd.read_csv("mnist_test.csv")[:1000]
x_train, x_test, y_train, y_test = getSplit(train, test)
```

(A)

In [138…
```
train500 = x_train[:500]
train1000 = x_train[:1000]
train2500 = x_train[:2500]
train5000 = x_train[:5000]
train10000 = x_train[:10000]
classError = []
labels = ['500','1000','2500','5000','10000']
```

In [ ]:
```
y_pred = KNNPredict(train500, y_train[:500], x_test, 2)
print("Accuracy: " + str(r2_score(y_pred, y_test)))
print("Confusion Matrix: ")
print(confusion_matrix(y_test, y_pred, labels=[0,1,2,3,4,5,6,7,8,9]))
```

In [ ]:
```
y_pred = KNNPredict(train1000, y_train[:1000], x_test, 2)
print("Accuracy: " + str(r2_score(y_pred, y_test)))
print("Confusion Matrix: ")
print(confusion_matrix(y_test, y_pred, labels=[0,1,2,3,4,5,6,7,8,9]))
```

In [ ]:
```
y_pred = KNNPredict(train2500, y_train[:2500], x_test, 2)
print("Accuracy: " + str(r2_score(y_pred, y_test)))
print("Confusion Matrix: ")
print(confusion_matrix(y_test, y_pred, labels=[0,1,2,3,4,5,6,7,8,9]))
```

In [ ]:
```
y_pred = KNNPredict(train5000, y_train[:5000], x_test, 2)
print("Accuracy: " + str(r2_score(y_pred, y_test)))
print("Confusion Matrix: ")
print(confusion_matrix(y_test, y_pred, labels=[0,1,2,3,4,5,6,7,8,9]))
```

In [ ]:
```
y_pred = KNNPredict(train10000, y_train[:10000], x_test, 2)
print("Accuracy: " + str(r2_score(y_pred, y_test)))
print("Confusion Matrix: ")
print(confusion_matrix(y_test, y_pred, labels=[0,1,2,3,4,5,6,7,8,9]))
```

Best Model

In [ ]:
```
y_pred = KNNPredict(train10000, y_train[:10000], x_test, 2)
print("Accuracy: " + str(r2_score(y_pred, y_test)))
print("Confusion Matrix: ")
print(confusion_matrix(y_test, y_pred, labels=[0,1,2,3,4,5,6,7,8,9]))
```