K-Nearest Neighbors (K-NN)

Importing the libraries

In [36]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

2. Dataset Description

The Glass Identification dataset (UCI ML Repository) contains 214 samples of glass shards.

Each instance has 9 chemical attributes and a refractive index, collected by the Forensic Science Service,

United Kingdom, for forensic identification of glass origin (building windows, vehicle windows, containers, tableware, etc.).

Features:

- RI: Refractive Index

- Na: Sodium (weight %)

- Mg: Magnesium

- Al: Aluminum

- Si: Silicon

- K: Potassium

- Ca: Calcium

- Ba: Barium

- Fe: Iron

Target Classes:

1: Building windows (float processed)

2: Building windows (non-float processed)

3: Vehicle windows (float processed)

4: Vehicle windows (non-float processed)

5: Containers

6: Tableware

7: Headlamps (rare)

Importing the dataset

In [58]:
```
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/glass/glass.data"

column_names = ['Id', 'RI', 'Na', 'Mg', 'Al', 'Si', 'K', 'Ca', 'Ba', 'Fe', 'Class']

# Read dataset to pandas dataframe
dataset = pd.read_csv(url, names=column_names)
```

Data Analysis EDA

In [59]:
```
dataset.shape
```

Out[59]:
```
(214, 11)
```

In [60]:
```
dataset.head()
```

Out[60]:

	Id	RI	Na	Mg	Al	Si	K	Ca	Ba	Fe	Class
0	1	1.52101	13.64	4.49	1.10	71.78	0.06	8.75	0.0	0.0	1
1	2	1.51761	13.89	3.60	1.36	72.73	0.48	7.83	0.0	0.0	1
2	3	1.51618	13.53	3.55	1.54	72.99	0.39	7.78	0.0	0.0	1
3	4	1.51766	13.21	3.69	1.29	72.61	0.57	8.22	0.0	0.0	1
4	5	1.51742	13.27	3.62	1.24	73.08	0.55	8.07	0.0	0.0	1

In [61]:
```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 214 entries, 0 to 213
Data columns (total 11 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Id      214 non-null    int64  
 1   RI      214 non-null    float64
 2   Na      214 non-null    float64
 3   Mg      214 non-null    float64
 4   Al      214 non-null    float64
 5   Si      214 non-null    float64
 6   K       214 non-null    float64
 7   Ca      214 non-null    float64
 8   Ba      214 non-null    float64
 9   Fe      214 non-null    float64
 10  Class   214 non-null    int64  
dtypes: float64(9), int64(2)
memory usage: 18.5 KB
```

In [62]:
```
dataset.describe()
```

Out[62]:

	Id	RI	Na	Mg	Al	Si	K	Ca	Ba	Fe
count	214.000000	214.000000	214.000000	214.000000	214.000000	214.000000	214.000000	214.000000	214.000000	214.000000
mean	107.500000	1.518365	13.407850	2.684533	1.444907	72.650935	0.497056	8.956963	0.175047	0.057009
std	61.920648	0.003037	0.816604	1.442408	0.499270	0.774546	0.652192	1.423153	0.497219	0.097439
min	1.000000	1.511150	10.730000	0.000000	0.290000	69.810000	0.000000	5.430000	0.000000	0.000000
25%	54.250000	1.516522	12.907500	2.115000	1.190000	72.280000	0.122500	8.240000	0.000000	0.000000
50%	107.500000	1.517680	13.300000	3.480000	1.360000	72.790000	0.555000	8.600000	0.000000	0.000000
75%	160.750000	1.519157	13.825000	3.600000	1.630000	73.087500	0.610000	9.172500	0.000000	0.100000
max	214.000000	1.533930	17.380000	4.490000	3.500000	75.410000	6.210000	16.190000	3.150000	0.510000

In [43]:
```
dataset.groupby('glass_type').size()
```

Out[43]:
```
glass_type
1    70
2    76
3    17
5    13
6     9
7    29
dtype: int64
```

Data Preprocessing

In [63]:
```
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
```

Splitting the dataset into the Training set and Test set

In [64]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.20,
                                                    random_state = 0)
```

Feature Scaling

In [65]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
sc.fit(X_train)

X_train = sc.transform(X_train)
X_test = sc.transform(X_test)
```

In [66]:
```
print(X_train)
```

```
[[-1.41577636 -0.1936807  -0.94094292 ... -0.18247525 -0.35547602
  -0.55580588]
 [ 0.49746797 -0.17682261 -0.30269425 ... -0.3627608  -0.35547602
   2.53861104]
 [ 0.28845809 -0.52072759 -0.25263553 ... -0.59851883 -0.35547602
  -0.55580588]
 ...
 [ 0.20806967 -0.44992363  0.36058379 ... -0.75106814 -0.35547602
  -0.55580588]
 [-0.91736818  2.78345757  0.69848015 ...  0.59413942 -0.35547602
   0.54934302]
 [ 1.09234226 -1.75473961 -0.54047316 ... -1.40980379 -0.35547602
  -0.55580588]]
```

In [48]:
```
print(X_test)
```

```
[[ 1.49428435e+00 -3.85862894e-01  1.58702242e+00 -1.86873548e+00
   1.76307822e+00  8.67158232e-01 -7.10550375e-01 -9.12376258e-03
   1.04876001e+00 -5.55805878e-01]
 [-1.07814501e+00 -1.49849668e-01 -8.65854842e-01  5.54130679e-01
  -1.67776430e-01  4.40018990e-01  1.84610590e-01 -1.96343371e-01
  -3.55476017e-01 -5.55805878e-01]
 [-2.42105468e-01 -6.79193618e-01 -1.10363376e+00  5.54130679e-01
   8.46409850e-01  8.00417725e-01  1.70623700e-01 -6.12386944e-01
  -3.55476017e-01  4.38828130e-01]
 [ 1.15665300e+00  2.14285025e-01  7.10994826e-01 -2.04755561e-01
   2.41798798e-01 -3.47518989e-01 -7.10550375e-01  4.20787930e-01
  -3.55476017e-01 -5.55805878e-01]
 [ 1.06018690e+00  1.77871555e+00  1.01727515e-02 -1.86873548e+00
   2.80805963e-01 -5.47740509e-01 -2.62969892e-01  2.27218183e+00
  -3.55476017e-01 -5.55805878e-01]
 [-4.67193037e-01 -8.47774494e-01 -5.15443804e-01  6.23753269e-01
   1.44280887e-01  6.53588611e-01  2.54545041e-01 -6.95595659e-01
  -3.55476017e-01 -5.55805878e-01]
 [-1.29561683e-01 -1.49849668e-03 -5.15443804e-01  6.51602306e-01
  -7.33380317e-01 -3.87563293e-01  1.84610590e-01  1.15689310e-01
  -3.55476017e-01  1.10191747e+00]
 [ 5.29623340e-01 -4.39808774e-01 -6.78134643e-01  6.51602306e-01
   2.61302380e-01  4.40018990e-01  1.42649920e-01 -5.91584766e-01
  -3.55476017e-01 -5.55805878e-01]
 [-1.59263088e+00 -8.27544789e-01 -8.03281442e-01  6.44640047e-01
   3.58820292e-01  4.53367091e-01  1.84610590e-01 -6.19321004e-01
  -3.55476017e-01  2.31758126e+00]
 [-3.38571569e-01 -8.34288024e-01 -4.27841045e-01  5.81979715e-01
   2.22295216e-01  3.19886078e-01  2.40558150e-01 -6.33189123e-01
  -3.55476017e-01  4.38828130e-01]
 [-7.88746707e-01 -2.44254958e-01 -1.22878056e+00  1.80367297e-02
  -2.84797924e-01  1.42777849e+00  7.27154695e-02  6.71508926e-02
  -3.55476017e-01  2.09655148e+00]
 [ 6.58244808e-01 -8.24173171e-03 -7.28193363e-01  6.86413601e-01
  -3.82315836e-01 -8.05569624e-02  1.56636810e-01 -1.96343371e-01
  -3.55476017e-01  3.31221526e+00]
 [ 8.99410061e-01 -3.18430543e-02 -1.27488727e-01  4.56659052e-01
   2.02791634e-01 -6.54525320e-01  7.27154695e-02  1.86124757e-02
  -3.55476017e-01 -5.55805878e-01]
 [ 1.27681253e-01  3.15433550e+00 -9.91001641e-01 -1.86873548e+00
  -1.49402003e+00 -8.14702536e-01 -6.26629035e-01  3.76993870e+00
  -3.55476017e-01 -5.55805878e-01]
 [-4.83270721e-01 -8.27544789e-01 -5.15443804e-01  6.09828751e-01
   2.02791634e-01  6.40240510e-01  2.96505711e-01 -7.37200016e-01
  -3.55476017e-01 -5.55805878e-01]
 [ 1.57467277e+00 -1.10401743e+00  1.77474262e+00 -1.86873548e+00
   1.91910688e+00  1.45447469e+00 -7.10550375e-01 -3.97431098e-01
   8.35996975e-01 -5.55805878e-01]
 [ 3.52768822e-01 -5.88159945e-01 -6.15561244e-01  6.44640047e-01
  -3.43308671e-01  1.59708862e-01  7.27154695e-02 -2.51815847e-01
  -3.55476017e-01 -5.55805878e-01]
 [-1.48008710e+00 -8.51146111e-01 -6.90649323e-01  5.19319383e-01
  -7.02585187e-02  8.67158232e-01  2.54545041e-01 -6.33189123e-01
  -3.55476017e-01  2.09655148e+00]
 [ 7.86866276e-01 -7.80342144e-01 -1.48566083e-02  4.98432606e-01
  -4.21323000e-01  7.96202536e-02  1.14676140e-01 -4.45969515e-01
  -3.55476017e-01 -5.55805878e-01]
 [ 8.67254694e-01 -2.20653635e-01  1.22804871e-01  5.05394865e-01
   1.63784469e-01 -7.88006333e-01  1.00689250e-01 -1.20068716e-01
```

```
 -3.55476017e-01 -5.55805878e-01]
[ 1.02803153e+00 -5.91531562e-01 -7.15678683e-01 -1.86873548e+00
  7.68395520e-01  1.66804431e+00  6.46177963e-01  8.36831504e-01
 -3.55476017e-01 -5.55805878e-01]
[ 5.93934074e-01  3.22176785e-02 -2.90179566e-01  6.58564565e-01
 -7.13876735e-01  2.66493672e-01  8.67023596e-02 -3.83562979e-01
 -1.63989286e-01  1.32294725e+00]
[ 1.44605130e+00 -5.34214065e-01  1.41181690e+00 -1.86873548e+00
  1.06094925e+00  8.80506333e-01 -7.10550375e-01 -3.07288324e-01
  2.98490362e+00  2.17798350e-01]
[ 1.36566288e+00  1.76522908e+00  2.95112253e+00 -5.94642069e-01
 -2.45790759e-01 -2.93705065e+00 -2.76956783e-01 -2.44881788e-01
  3.21894296e+00 -5.55805878e-01]
[ 6.42167125e-01 -6.11761268e-01 -5.52987844e-01  3.45262906e-01
 -4.01819418e-01  4.53367091e-01  1.00689250e-01 -1.06200596e-01
 -3.55476017e-01  2.09655148e+00]
[-1.38362100e+00  2.34514730e-01  5.85848027e-01  7.28187156e-01
 -4.99337329e-01 -6.81221522e-01 -6.26629035e-01 -5.07281199e-02
 -3.55476017e-01 -5.55805878e-01]
[ 1.67113887e+00 -6.79193618e-01  1.17403798e+00 -1.86873548e+00
  2.54322151e+00  2.93189875e-01 -7.10550375e-01  3.37579215e-01
  7.93444368e-01 -5.55805878e-01]
[-1.43185405e+00 -2.71227898e-01 -7.78252082e-01  5.95904233e-01
 -4.01819418e-01  8.13765827e-01  1.00689250e-01 -3.97431098e-01
 -3.55476017e-01 -5.55805878e-01]
[-2.90338519e-01 -9.18578462e-01 -2.40120847e-01  5.61092938e-01
  6.62665574e-02  8.27113928e-01 -1.79048552e-01 -6.47057242e-01
 -3.55476017e-01 -5.55805878e-01]
[-5.31503771e-01  2.21028260e-02  2.60466350e-01  8.25658783e-01
 -3.23805089e-01 -8.94791144e-01  4.47416893e-02 -4.45969515e-01
 -3.55476017e-01  2.98067059e+00]
[-1.56047552e+00 -2.88085986e-01 -3.52752966e-01  6.44640047e-01
 -7.52883899e-01  8.13765827e-01  8.67023596e-02 -5.01441991e-01
 -3.55476017e-01 -5.55805878e-01]
[-6.60125239e-01  1.29994587e+00  9.23744384e-01  7.83885228e-01
 -1.27948062e+00 -1.70902532e+00 -7.10550375e-01  5.03996645e-01
 -3.55476017e-01 -5.55805878e-01]
[ 6.26089441e-01 -4.46552009e-01 -5.40473164e-01  5.47168420e-01
  6.90381191e-01  1.19664558e-01  2.12584370e-01 -5.43046349e-01
 -3.55476017e-01 -5.55805878e-01]
[-1.13484000e-01 -3.31917014e-01 -1.54164756e+00  3.93998720e-01
 -5.38344494e-01  1.22755697e+00  1.56636810e-01 -4.37940604e-02
 -3.55476017e-01  2.09655148e+00]
[ 5.13545657e-01 -1.19505110e-01 -5.40473164e-01  7.76922969e-01
 -6.94373152e-01  5.86848104e-01  7.27154695e-02 -4.04365158e-01
 -3.55476017e-01  7.70372799e-01]
[ 9.31565428e-01  1.24599999e+00  9.48773744e-01  7.62998451e-01
 -1.02593405e+00 -1.69567722e+00 -3.88851903e-01  1.22623369e-01
 -3.55476017e-01  3.53324504e+00]
[-1.14245574e+00 -2.98200838e-01 -1.07860440e+00  5.47168420e-01
 -1.09265683e-01  1.01398735e+00  1.28663030e-01 -2.86486145e-01
 -3.55476017e-01  1.07283460e-01]
[ 8.02943960e-01 -4.97126272e-01 -7.15678683e-01  6.23753269e-01
 -2.45790759e-01 -2.71645571e-02  1.42649920e-01 -1.20068716e-01
 -3.55476017e-01 -5.55805878e-01]
[-2.26027784e-01 -1.49849668e-03 -6.28075923e-01  7.35149415e-01
 -6.35862406e-01 -4.67651901e-01  1.84610590e-01 -2.18970302e-03
 -3.55476017e-01  1.87552170e+00]
[ 1.39781825e+00 -8.07315084e-01  1.77474262e+00 -1.86873548e+00
  1.84109255e+00  8.67158232e-01 -7.10550375e-01 -1.40870894e-01
  1.00620740e+00  4.38828130e-01]
[ 3.12151516e-02  4.32765840e+00 -3.38130550e+00 -1.86873548e+00
  1.29499224e+00 -3.76463293e+00  1.00689250e-01  3.00719215e+00
  6.34655956e+00  2.53861104e+00]
[ 1.22096373e+00 -1.82891519e+00  1.21158202e+00 -6.57302401e-01
  2.02791634e-01  2.56236710e+00 -7.10550375e-01 -9.52155863e-01
 -3.55476017e-01 -5.55805878e-01]
[ 5.61778707e-01 -5.64558622e-01 -7.03164003e-01  6.09828751e-01
  3.97827457e-01  6.80284814e-01  1.98597480e-01 -6.74793480e-01
 -3.55476017e-01 -5.55805878e-01]]
```

Training the K-NN model on the Training set

In [67]:
```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=11,p=2,metric='cityblock')
classifier.fit(X_train,y_train)
```

Out[67]:
```
KNeighborsClassifier
KNeighborsClassifier(metric='cityblock', n_neighbors=11)
```

Getting nearest neighbours for each point in training data

In [81]:
```
classifier.kneighbors(X=X_train, n_neighbors=11, return_distance=False)
```

Out[81]:
```
array([[  0, 104,  19, ..., 128,   3, 124],
       [  1,  61,  46, ...,  77,  96, 111],
       [  2, 158, 157, ...,  70,  62,  69],
       ...,
       [168,  53, 105, ...,  62,  78, 115],
       [169,  73,  59, ...,  84, 117,  94],
       [170, 112,  95, ..., 157, 142,   9]])
```

In [82]:
```
dataset.iloc[[  0,  16,  73,  55,  54,  60,  29],-1]
```

Out[82]:
```
0     1
16    1
73    2
55    1
54    1
60    1
29    1
Name: Class, dtype: int64
```

In [83]:
```
classifier.predict(X_train[[1]])
```

Out[83]:
```
array([2])
```

Predicting the Test set results

In [84]:
```
y_pred = classifier.predict(X_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1),y_test.reshape(len(y_test),1)),1))
```

```
[[7 7]
 [1 1]
 [2 2]
 [6 6]
 [5 5]
 [2 2]
 [1 2]
 [2 2]
 [1 1]
 [2 2]
 [1 1]
 [2 2]
 [2 3]
 [2 2]
 [2 2]
 [7 7]
 [2 2]
 [1 1]
 [2 3]
 [2 3]
 [2 5]
 [2 2]
 [7 7]
 [7 7]
 [2 2]
 [1 1]
 [7 7]
 [1 1]
 [2 2]
 [1 2]
 [1 1]
 [1 1]
 [2 2]
 [1 2]
 [2 2]
 [1 3]
 [1 1]
 [2 3]
 [1 2]
 [7 7]
 [2 2]
 [6 6]
 [2 2]]
```

Evaluating the Algorithm

Making the Confusion Matrix & Predicting Accuracy Score

In [85]:
```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test,y_pred)
print(cm)
accuracy = accuracy_score(y_test, y_pred)*100
print('Accuracy of our model is equal ' + str(round(accuracy, 2)) + ' %.')
```

```
[[ 9  0  0  0  0  0]
 [ 4 15  0  0  0  0]
 [ 1  4  0  0  0  0]
 [ 0  1  0  1  0  0]
 [ 0  0  0  0  2  0]
 [ 0  0  0  0  0  6]]
Accuracy of our model is equal 76.74 %.
```

Making Classification Report

In [88]:
```
from sklearn.metrics import classification_report
# here f1 score is goodness of fit .
print(classification_report(y_test, y_pred, zero_division=0))
```

```
              precision    recall  f1-score   support

           1       0.64      1.00      0.78         9
           2       0.75      0.79      0.77        19
           3       0.00      0.00      0.00         5
           5       1.00      0.50      0.67         2
           6       1.00      1.00      1.00         2
           7       1.00      1.00      1.00         6

    accuracy                           0.77        43
   macro avg       0.73      0.71      0.70        43
weighted avg       0.70      0.77      0.72        43
```

Comparing Error Rate with the K Value

Parameter Tuning Using

cross-validation for parameter tuning:

In [91]:
```
from sklearn.model_selection import cross_val_score

# creating list of K for KNN
k_list = list(range(1,214))

# creating list of cv scores
cv_scores = []

# perform 10-fold cross validation
for k in k_list:
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, X_train, y_train, cv=10, scoring='accuracy')
    cv_scores.append(scores.mean())
```

```
C:\Users\Mohammed Meraj\AppData\Roaming\Python\Python312\site-packages\sklearn\model_selection\_split.py:805: Us
erWarning: The least populated class in y has only 7 members, which is less than n_splits=10.
  warnings.warn(
C:\Users\Mohammed Meraj\AppData\Roaming\Python\Python312\site-packages\sklearn\model_selection\_split.py:805: Us
erWarning: The least populated class in y has only 7 members, which is less than n_splits=10.
  warnings.warn(
C:\Users\Mohammed Meraj\AppData\Roaming\Python\Python312\site-packages\sklearn\model_selection\_split.py:805: Us
erWarning: The least populated class in y has only 7 members, which is less than n_splits=10.
  warnings.warn(
C:\Users\Mohammed Meraj\AppData\Roaming\Python\Python312\site-packages\sklearn\model_selection\_split.py:805: Us
erWarning: The least populated class in y has only 7 members, which is less than n_splits=10.
  warnings.warn(
C:\Users\Mohammed Meraj\AppData\Roaming\Python\Python312\site-packages\sklearn\model_selection\_split.py:805: Us
erWarning: The least populated class in y has only 7 members, which is less than n_splits=10.
  warnings.warn(
C:\Users\Mohammed Meraj\AppData\Roaming\Python\Python312\site-packages\sklearn\model_selection\_split.py:805: Us
erWarning: The least populated class in y has only 7 members, which is less than n_splits=10.
  warnings.warn(
C:\Users\Mohammed Meraj\AppData\Roaming\Python\Python312\site-packages\sklearn\model_selection\_split.py:805: Us
erWarning: The least populated class in y has only 7 members, which is less than n_splits=10.
  warnings.warn(
C:\Users\Mohammed Meraj\AppData\Roaming\Python\Python312\site-packages\sklearn\model_selection\_split.py:805: Us
erWarning: The least populated class in y has only 7 members, which is less than n_splits=10.
  warnings.warn(
```

```
    neigh_ind = self.kneighbors(X, return_distance=False)
                ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\Mohammed Meraj\AppData\Roaming\Python\Python312\site-packages\sklearn\neighbors\_base.py", line
854, in kneighbors
    raise ValueError(
ValueError: Expected n_neighbors <= n_samples_fit, but n_neighbors = 213, n_samples_fit = 154, n_samples = 17

  warnings.warn(
C:\Users\Mohammed Meraj\AppData\Roaming\Python\Python312\site-packages\sklearn\model_selection\_validation.py:97
8: UserWarning: Scoring failed. The score on this train-test partition for these parameters will be set to nan.
Details:
Traceback (most recent call last):
  File "C:\Users\Mohammed Meraj\AppData\Roaming\Python\Python312\site-packages\sklearn\metrics\_scorer.py", line
140, in __call__
    score = scorer._score(
            ^^^^^^^^^^^^^^
  File "C:\Users\Mohammed Meraj\AppData\Roaming\Python\Python312\site-packages\sklearn\metrics\_scorer.py", line
380, in _score
    y_pred = method_caller(
             ^^^^^^^^^^^^^^
  File "C:\Users\Mohammed Meraj\AppData\Roaming\Python\Python312\site-packages\sklearn\metrics\_scorer.py", line
90, in _cached_call
    result, _ = _get_response_values(
                ^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\Mohammed Meraj\AppData\Roaming\Python\Python312\site-packages\sklearn\utils\_response.py", line
214, in _get_response_values
    y_pred = prediction_method(X)
             ^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\Mohammed Meraj\AppData\Roaming\Python\Python312\site-packages\sklearn\neighbors\_classification
.py", line 262, in predict
    probabilities = self.predict_proba(X)
                    ^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\Mohammed Meraj\AppData\Roaming\Python\Python312\site-packages\sklearn\neighbors\_classification
.py", line 371, in predict_proba
    neigh_ind = self.kneighbors(X, return_distance=False)
                ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\Mohammed Meraj\AppData\Roaming\Python\Python312\site-packages\sklearn\neighbors\_base.py", line
854, in kneighbors
    raise ValueError(
ValueError: Expected n_neighbors <= n_samples_fit, but n_neighbors = 213, n_samples_fit = 154, n_samples = 17

  warnings.warn(
```

plot the error values against K values

In [95]:

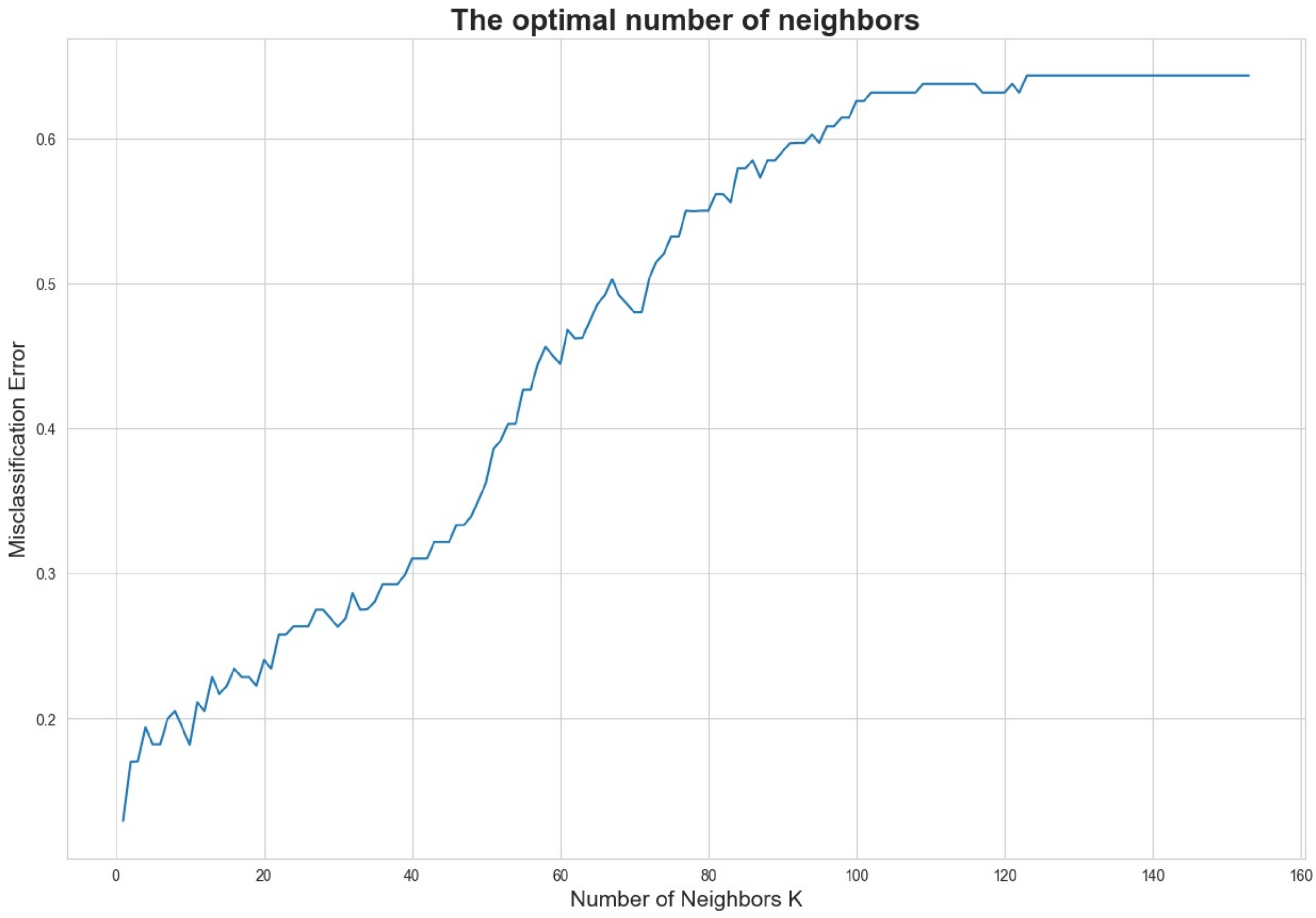
```python
import seaborn as sns

# changing to misclassification error
MSE = [1-x for x in cv_scores]

plt.figure()
plt.figure(figsize=(15,10))
plt.title('The optimal number of neighbors', fontsize=20, fontweight='bold')
plt.xlabel('Number of Neighbors K', fontsize=15)
plt.ylabel('Misclassification Error', fontsize=15)
sns.set_style("whitegrid")
plt.plot(k_list, MSE)

plt.show()
```

```
<Figure size 640x480 with 0 Axes>
```



finding best k

In [96]:
```
best_k = k_list[MSE.index(min(MSE))]
print("The optimal number of neighbors is %d." % best_k)
```

```
The optimal number of neighbors is 1.
```

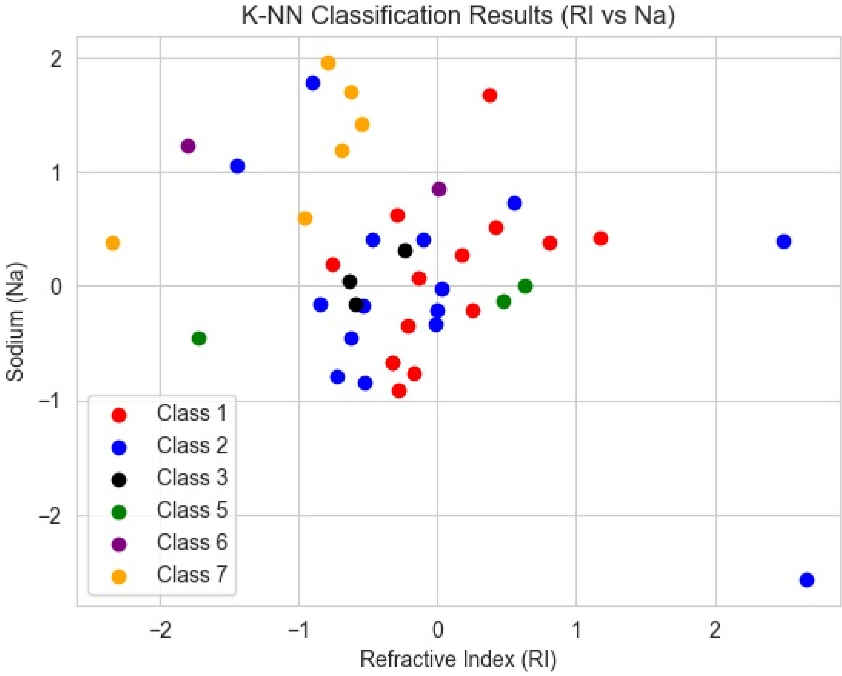
Visualize Test Result of KNN

In [101…
```
from matplotlib.colors import ListedColormap

markers = ('s', 'x', 'o')
colors = ('red', 'blue', 'black', 'green', 'purple', 'orange', 'cyan')
cmap = ListedColormap(colors[:len(np.unique(y_test))])

for idx, cl in enumerate(np.unique(y_test)):
    plt.scatter(
        X_test[y_test == cl][:, 0],
        X_test[y_test == cl][:, 1],
        c=[cmap(idx)],
        marker='o',
        label=f'Class {cl}'
```

```
    )
plt.title('K-NN Classification Results (RI vs Na)')
plt.xlabel('Refractive Index (RI)')
plt.ylabel('Sodium (Na)')
plt.legend()
plt.show()
```



In [ ]:

In [ ]: