LAB 5: K-Nearest Neighbors (K-NN)

Importing the libraries

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

Importing the dataset

In [2]:
```
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.c

# Assign column names to the dataset
names = ['class', 'alcohol', 'malicacid', 'ash', 'alcalinity', 'magnesium',

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

Data Analysis EDA

In [3]:
```
dataset.shape
```

Out[3]:
```
(178, 14)
```

In [4]:
```
dataset.head()
```

Out[4]:

	class	alcohol	malicacid	ash	alcalinity	magnesium	phenols	flavanoids
0	1	14.23	1.71	2.43	15.6	127	2.80	3.06
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24
3	1	14.37	1.95	2.50	16.8	113	3.85	3.49
4	1	13.24	2.59	2.87	21.0	118	2.80	2.69

In [5]:
```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 178 entries, 0 to 177
Data columns (total 14 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   class                 178 non-null    int64  
 1   alcohol               178 non-null    float64
 2   malicacid             178 non-null    float64
 3   ash                   178 non-null    float64
 4   alcalinity            178 non-null    float64
 5   magnesium             178 non-null    int64  
 6   phenols               178 non-null    float64
 7   flavanoids            178 non-null    float64
 8   nonflavanoid_phenols  178 non-null    float64
 9   proanthocyanins       178 non-null    float64
 10  Color_intensity       178 non-null    float64
 11  hue                   178 non-null    float64
 12  diluted_wines         178 non-null    float64
 13  proline               178 non-null    int64  
dtypes: float64(11), int64(3)
memory usage: 19.6 KB
```

In [6]:
```python
dataset.describe()
```

Out[6]:

	class	alcohol	malicacid	ash	alcalinity	magnesiu
count	178.000000	178.000000	178.000000	178.000000	178.000000	178.00000
mean	1.938202	13.000618	2.336348	2.366517	19.494944	99.74157
std	0.775035	0.811827	1.117146	0.274344	3.339564	14.28248
min	1.000000	11.030000	0.740000	1.360000	10.600000	70.00000
25%	1.000000	12.362500	1.602500	2.210000	17.200000	88.00000
50%	2.000000	13.050000	1.865000	2.360000	19.500000	98.00000
75%	3.000000	13.677500	3.082500	2.557500	21.500000	107.00000
max	3.000000	14.830000	5.800000	3.230000	30.000000	162.00000

In [7]:
```python
dataset.groupby('class').size()
```

Out[7]:
```
class
1    59
2    71
3    48
dtype: int64
```

Data Preprocessing

In [16]:
```python
a = [0]
b = [1,2,3,4]
```

```
X = dataset.iloc[:, b]
y = dataset.iloc[:, a].values.ravel()
```

Splitting the dataset into the Training set and Test set

In [17]:

```
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,rar
```

Feature Scaling

In [18]:

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
sc.fit(X_train)

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

In [21]:

```
print(X_train)
```

```
[[ 0.87668336  0.79842885  0.64412971  0.12974277]
 [-0.36659076 -0.7581304  -0.39779858  0.33380024]
 [-1.69689407 -0.34424759 -0.32337513 -0.45327855]
 [ 0.51613387  1.38326325  0.42085937  1.00427477]
 [ 0.64046128 -0.50620174  0.90461179  0.12974277]
 [ 0.92641433 -0.78512276  1.23951731  0.85851943]
 [-0.8639004   0.4115384  -0.54664548 -0.45327855]
 [-0.47848543 -0.92908199 -1.73742067 -0.30752322]
 [-1.95798163 -1.46892913  0.49528281  0.42125344]
 [ 0.81451966  0.65446961  0.71855316 -1.26950841]
 [-0.47848543  0.07863266 -0.62106893 -0.30752322]
 [-1.27418086 -1.1540183  -0.24895168  0.42125344]
 [-0.91363137  1.35627089 -0.62106893 -0.30752322]
 [ 1.63508058 -0.40722976  1.31394076  0.12974277]
 [-0.13036867  0.55549764  0.12316557  0.12974277]
 [ 0.62802854  1.09534477 -0.65828065 -0.01601256]
 [ 0.71505773 -0.59617626 -0.21173995 -0.97799775]
 [ 1.68481154 -0.62316862  1.23951731  1.58729609]
 [ 0.90154885 -0.46121447 -0.02568133 -0.86139348]
 [-0.95092959 -0.97406926 -1.58857377 -0.16176789]
 [ 0.35450823  1.71616898 -0.39779858  0.12974277]
 [ 0.21774808  1.07734987 -0.80712755  0.42125344]
 [ 1.08803996 -0.7761253   1.12788213  1.58729609]
 [-0.1676669   0.8074263   0.79297661  0.7127641 ]
 [ 1.06317448  1.57220975  0.04874212 -0.01601256]
 [ 0.39180646 -0.58717881 -0.84433927 -0.74478922]
 [ 0.08098793  1.33827599 -0.17452823  0.85851943]
 [-0.5903801  -1.01905652 -0.4350103  -0.59903389]
 [-0.8639004  -0.86609983 -1.43972687 -1.03629988]
 [-0.26712883  0.95138554 -1.43972687 -1.03629988]
 [-0.03090674  0.96938044 -0.06289306 -0.30752322]
 [ 0.85181788 -1.01005906 -1.66299722 -0.45327855]
 [ 0.21774808 -0.01134186  1.12788213 -0.27837216]
 [-0.76443848 -1.28898009 -3.7468538  -2.61045747]
 [-0.64011106 -0.76712785 -0.62106893 -0.16176789]
 [-1.46067198 -0.22728071  1.38836421  0.56700877]
 [ 0.31721001  1.44624541 -0.2861634  -0.59903389]
 [ 1.38642575 -0.1912909  -0.24895168 -0.45327855]
 [-1.70932681 -0.92008454  1.23951731  0.12974277]
 [ 0.00639148 -0.63216607  0.86740006  3.04484941]
 [-0.26712883  0.0156505  -0.32337513 -0.01601256]
 [ 0.6155958  -0.63216607  1.01624696  0.85851943]
 [-1.89581792  1.23030656 -2.03511447 -0.01601256]
 [-1.13742071 -0.1912909  -0.7327041   0.42125344]
 [-1.44823924 -0.81211511 -1.40251515  0.36295131]
 [ 0.57829758  1.19431675  0.86740006  1.00427477]
 [-0.97579508  0.59148745 -0.17452823 -0.16176789]
 [ 0.08098793 -0.57818135 -1.21645652 -2.08573827]
 [ 0.23018082  2.53493714 -0.17452823  0.7127641 ]
 [-1.12498797 -0.93807945 -0.24895168  1.17918116]
 [-1.0130933  -0.83011002  0.60691799 -0.16176789]
 [-0.70227477  1.85113077  1.35115248  2.02456209]
 [-1.38607554  1.74316134  0.08595384  0.42125344]
 [-0.96336233 -1.06404378 -2.29559654 -0.80309135]
 [ 1.18750189 -0.57818135 -0.36058685 -0.62818495]
 [ 1.38642575 -0.31725524  0.12316557 -0.22007002]
```

```
[-0.39145624 -1.25299028 -0.47222203 -0.45327855]
[ 2.19455393 -0.57818135  0.08595384 -2.37724893]
[-0.91363137  2.11205689  0.64412971  0.42125344]
[-0.1676669  -0.69514823  0.56970626 -0.51158069]
[ 0.92641433  1.7881486  -0.39779858  0.85851943]
[ 1.33669479 -0.66815588 -0.32337513 -1.03629988]
[ 0.80208692 -0.50620174  1.23951731 -0.68648708]
[-0.81416944 -1.24399282 -1.55136204 -1.38611268]
[ 1.038309   -0.56018645  0.19758902 -1.61932121]
[-1.12498797 -0.26327052 -2.48165516 -0.59903389]
[ 0.05612244 -1.3249699  -2.44444344 -1.03629988]
[ 1.72210977 -0.3802374   0.49528281 -0.80309135]
[ 0.23018082  0.19559954  0.01153039  0.12974277]
[-0.32929253 -0.56018645 -0.32337513  0.85851943]
[ 1.74697525 -0.45221702  0.30922419 -1.44441481]
[ 0.44153742 -1.28898009 -0.02568133 -0.74478922]
[-1.6471631  -0.44321957 -1.66299722 -1.03629988]
[-1.42337376 -1.33396735  0.79297661 -0.45327855]
[-0.15523416 -0.92008454 -0.17452823 -0.45327855]
[ 0.95127981 -0.57818135  0.16037729 -1.03629988]
[-0.56551462  2.82285562  1.01624696  1.58729609]
[ 0.15558437 -0.42522466  1.42557593  1.73305143]
[ 1.73454251 -0.45221702  0.04874212 -2.20234254]
[ 1.54805139 -0.59617626  0.23480074 -1.15290415]
[-1.2244499  -0.7761253   0.19758902  0.7127641 ]
[ 0.30477727  0.83441866 -0.32337513 -0.30752322]
[ 1.73454251  1.09534477 -0.32337513 -1.03629988]
[ 1.41129124 -0.80311766 -0.17452823 -0.80309135]
[-1.2244499   0.95138554 -1.36530342 -0.16176789]
[-0.8017367   0.06963521  0.34643592  0.42125344]
[ 0.6155958  -0.50620174  0.16037729  0.27549811]
[ 0.37937372 -0.3622425   1.16509386 -0.80309135]
[ 0.52856661  1.32028108 -0.91876272 -0.22007002]
[ 0.90154885  2.94881995  0.30922419  0.27549811]
[-0.76443848 -1.04604887  0.71855316 -0.42412749]
[ 0.26747904 -0.53319409 -0.84433927 -2.43555107]
[-1.90825067  0.02464795  0.19758902  0.12974277]
[ 0.78965418  2.31899829 -0.06289306  0.12974277]
[ 1.11290545 -0.43422212  0.83018834 -1.32781054]
[-0.70227477  0.15960973 -0.36058685  0.7127641 ]
[-0.21739786 -0.05632912  0.12316557  1.29578543]
[-0.50335091 -0.97406926 -0.99318617  0.12974277]
[ 1.05074174 -0.72214059  0.94182351  0.12974277]
[ 0.10585341 -0.78512276 -0.99318617 -1.18205521]
[-0.35415802  1.05935497 -0.02568133  0.56700877]
[ 0.97614529 -0.42522466  1.16509386 -0.71563815]
[-0.18009964  0.52850528  0.90461179  1.29578543]
[-0.76443848 -0.66815588 -0.24895168  1.44154076]
[ 0.65289402 -0.51519919  1.05345869 -0.16176789]
[ 1.10047271 -0.42522466  1.61163456 -0.04516362]
[-1.12498797 -0.88409473  0.49528281  0.85851943]
[ 0.73992321 -0.64116352 -0.02568133 -0.13261682]
[ 0.11828615  1.3742658  -0.02568133  0.56700877]
[-1.48553746 -0.21828326  1.53721111  2.60758342]
[-0.66497655  0.59148745  1.01624696  2.17031742]
[ 0.86425062 -0.48820683 -0.02568133 -0.68648708]
```

```
 [ 0.66532676  0.71745178  1.31394076  1.1500301 ]
 [ 0.64046128  0.67246452  0.94182351  1.29578543]
 [-0.14280141  2.01308491  0.42085937  0.56700877]
 [ 1.28696382 -0.62316862 -0.5838572  -1.03629988]
 [-0.8639004   0.71745178 -0.5838572  -0.45327855]
 [-1.12498797 -1.11802849  0.53249454  1.29578543]
 [-0.35415802  1.34727344  0.12316557  1.00427477]
 [-2.4304258  -0.7761253  -0.62106893  0.56700877]
 [ 0.41667194  0.78043394  0.04874212  0.56700877]
 [ 1.42372398  1.55421484  1.38836421  1.44154076]
 [ 0.62802854 -0.64116352 -0.47222203  1.29578543]
 [-1.42337376  0.46552311 -0.50943375 -0.45327855]
 [ 0.08098793 -0.29026288  3.17452699  1.58729609]
 [-0.32929253 -0.50620174 -0.62106893 -0.22007002]
 [-0.20496512  0.89740082 -0.24895168 -0.01601256]
 [ 0.51613387  2.00408746  1.8349049   1.58729609]
 [ 1.53561865  1.45524287  0.53249454 -1.85252974]
 [ 0.91398159 -0.84810492  0.49528281 -0.83224242]
 [-0.06820497  0.39354349  1.23951731  0.42125344]
 [-1.67202858 -0.28126543  0.34643592  0.59615984]
 [-0.8639004  -0.68615078 -0.5838572   0.24634704]
 [-1.65959584 -0.63216607  0.94182351  1.87880676]
 [ 0.36694097 -0.65915842  1.76048145 -1.18205521]
 [-0.06820497  1.28429127  1.05345869 -0.27837216]
 [ 1.08803996 -0.92008454 -0.36058685 -1.03629988]
 [-1.44823924 -0.58717881 -1.81184412 -0.01601256]
 [-0.76443848 -1.08203868 -1.66299722  0.01313851]
 [-0.70227477 -0.68615078 -0.65828065  0.85851943]
 [ 1.13777093 -0.62316862 -0.91876272 -1.03629988]
 [ 1.4610222   0.12361993  0.42085937  0.12974277]]
```

In [22]:

```
print(X_test)
```

```
[[ 0.93884707 -0.63216607 -0.4350103  -0.91969562]
 [-0.24226334  0.26757916  0.42085937  0.7127641 ]
 [-0.76443848 -1.11802849 -0.76991583 -0.16176789]
 [ 0.71505773 -0.57818135  0.34643592  0.27549811]
 [ 0.08098793  3.08378173 -0.881551    0.56700877]
 [-1.77149051 -0.29026288  3.21173872  2.60758342]
 [ 1.33669479 -0.20028836  0.90461179 -0.56988282]
 [-0.77687122  1.31128363  0.04874212  0.42125344]
 [-0.91363137 -0.57818135 -0.91876272 -0.16176789]
 [-1.12498797 -0.48820683 -0.17452823 -0.30752322]
 [ 0.4664029   0.16860719 -0.06289306  0.12974277]
 [ 1.11290545  2.39997536 -0.50943375  0.12974277]
 [ 1.51075317 -0.551189    0.30922419 -1.26950841]
 [-0.76443848 -1.17201321 -0.99318617 -0.30752322]
 [ 0.73992321  0.18660209  1.20230558  1.44154076]
 [-1.2244499  -1.30697499 -1.36530342 -0.16176789]
 [ 0.96371255 -0.7581304   1.23951731 -0.01601256]
 [ 1.49832042 -0.70414569  0.42085937 -0.89054455]
 [-0.42875446 -0.91108709 -1.29087997 -0.80309135]
 [ 1.05074174 -0.65016097  0.86740006 -0.68648708]
 [ 0.15558437 -1.22599792 -2.48165516 -1.32781054]
 [ 0.08098793 -0.65016097  0.68134144 -0.45327855]
 [ 0.31721001  0.19559954  1.87211663  0.42125344]
 [-0.5903801  -0.57818135 -1.43972687  0.27549811]
 [-0.81416944 -1.14502085 -0.32337513 -1.03629988]
 [-0.57794736  0.05164031 -0.7327041   0.42125344]
 [-0.68984203 -0.7581304  -0.2861634   0.56700877]
 [-1.02552604 -0.68615078 -0.21173995  0.94597263]
 [-1.52283569  0.27657661  2.05817525  0.12974277]
 [ 0.98857803  0.34855623 -0.24895168  0.7127641 ]
 [ 0.50370113 -0.54219154  0.94182351 -1.00714881]
 [ 1.53561865 -0.60517371 -0.24895168 -0.94884668]
 [-1.17471893  1.73416389  0.04874212  0.7127641 ]
 [ 1.52318591  1.50023013  0.27201247 -0.19091896]
 [ 2.29401586 -0.65915842 -0.7327041  -1.61932121]
 [ 0.08098793 -0.54219154 -0.99318617 -0.74478922]]
```

Training the K-NN model on the Training set

In [27]:
```python
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=7,p=2,metric='cityblock')
classifier.fit(X_train,y_train)
```

Out[27]:
```
KNeighborsClassifier
KNeighborsClassifier(metric='cityblock', n_neighbors=7)
```

Getting nearest neighbours for each point in training data

In [28]:
```python
classifier.kneighbors(X=X_train, n_neighbors=5, return_distance=False)
```

```
Out[28]:  array([[  0, 141, 120,   9,  24],
                 [  1,  69, 132,  34, 125],
                 [  2, 123,  92, 131,   6],
                 [  3,  45, 118,  26, 108],
                 [  4, 104,  98,  86,  41],
                 [  5,  41,  22,  98,   4],
                 [  6, 116, 123,  10,  46],
                 [  7,  19, 138,  97,  28],
                 [  8,  11, 131,  38,  73],
                 [  9,  94,   0,  75, 101],
                 [ 10, 125,  40,   6,  34],
                 [ 11,  49,  80, 132,   1],
                 [ 12, 116,   6,  46,  84],
                 [ 13, 105,  98,   4, 141],
                 [ 14,  68, 126,  30,  40],
                 [ 15,  88,  21,  81,  20],
                 [ 16,  18,  75, 111,  61],
                 [ 17,  22,   5,  13,  41],
                 [ 18, 111,  75,  16, 129],
                 [ 19, 138,   7,  28, 137],
                 [ 20,  36,  15,  60,  93],
                 [ 21,  15,  88, 108,  26],
                 [ 22,  17,   5,  41,  77],
                 [ 23, 102, 130, 100,  45],
                 [ 24,  93,  20,  15,  60],
                 [ 25,  99, 140,  16,  54],
                 [ 26, 108, 118, 100,   3],
                 [ 27,  56,  34,  74, 125],
                 [ 28,  63,  19,  53,  72],
                 [ 29,  84, 116, 126,  30],
                 [ 30, 126,  81,  14, 100],
                 [ 31,   7,  25, 140, 138],
                 [ 32,  87, 104, 130,   4],
                 [ 33,  63,  53,  66,  28],
                 [ 34, 125, 132,  27,   1],
                 [ 35, 131,  38,  80,  85],
                 [ 36,  81,  20,  30,  88],
                 [ 37,  55,  54,  83,  61],
                 [ 38,  35,  50,  73,   8],
                 [ 39,  77, 133, 110,  22],
                 [ 40,  10, 126, 125,  68],
                 [ 41,   5,   4, 104,  98],
                 [ 42,  84, 137,  12, 123],
                 [ 43, 132, 139,  95,  11],
                 [ 44, 137,  19, 138, 132],
                 [ 45,   3, 113, 112,  23],
                 [ 46, 116,   6, 123, 126],
                 [ 47,  91,  99,  25, 140],
                 [ 48,  93,  26, 108, 114],
                 [ 49, 103, 117, 106,  11],
                 [ 50,  90, 106,  59,  73],
                 [ 51, 110,  76, 127,  58],
                 [ 52,  58, 100, 114, 118],
                 [ 53,  65,  28,  19,  63],
                 [ 54,  61, 115, 111,  83],
                 [ 55,  37, 141, 107, 111],
```

```
[ 56,  27,  74,  34, 125],
[ 57,  78,  70,  79,  64],
[ 58, 114,  52, 100, 118],
[ 59,  74,  90,  50, 129],
[ 60,  20,  26,  24,  93],
[ 61, 115, 136,  83,  54],
[ 62, 101,  87, 104,   4],
[ 63,  28,  53,  19, 138],
[ 64,  75,  79,  70,  94],
[ 65,  53, 137,  72,  19],
[ 66,  53,  99,  63,   7],
[ 67,  70,  79, 129,  55],
[ 68,  14,  40, 120,  86],
[ 69, 139,   1, 103,  95],
[ 70,  79,  67,  78,  64],
[ 71, 111,  16,  18,  74],
[ 72,  28, 137,  65,   2],
[ 73,  90,  50,  38,   8],
[ 74,  56,  27,  59, 125],
[ 75,  18,  16, 111,  64],
[ 76,  51,  58, 127, 114],
[ 77,  22,  41,  17, 124],
[ 78,  57,  70,  64,  79],
[ 79,  70,  75,  61,  67],
[ 80, 106,  11,  49, 131],
[ 81,  30, 126,  36,  15],
[ 82,  61, 128,  36,  37],
[ 83,  61,  54, 136, 115],
[ 84,  12, 116,  42,  46],
[ 85,  95, 131,  92,  14],
[ 86, 107,   4, 104,  68],
[ 87, 101,  62,  32, 104],
[ 88,  15,  21,  36,  81],
[ 89,  93,  48,  24, 114],
[ 90,  50,  73,  59,  27],
[ 91,  47,  99,  25, 140],
[ 92, 131,  85,   2,  80],
[ 93,  89,  24,  20,  48],
[ 94, 101,  64,  75,  70],
[ 95,  85,  69, 139,  40],
[ 96, 102,  69, 103,  95],
[ 97,  34,   1, 138, 132],
[ 98,   4, 104, 105,   5],
[ 99,  25,  47, 140,  28],
[100, 108, 118, 126, 120],
[101,  62,  87, 104,  94],
[102,  23, 113,  96, 130],
[103,  49, 139,  69, 132],
[104,   4,  62,  98, 101],
[105,  13,  98, 104, 101],
[106,  80, 117,  49,  50],
[107, 111,  86,  18,  16],
[108,  26, 100, 120, 118],
[109, 133,  35, 110,  77],
[110, 102,  51, 113,  23],
[111,  18,  75,  16,  54],
```

```
       [112, 113,  45, 102,   3],
       [113, 112,  45, 102,   3],
       [114,  58, 108, 100, 118],
       [115,  61, 140, 136,  54],
       [116,   6,  12, 123,  46],
       [117, 106,  49,  80, 103],
       [118, 100,  26, 108,   3],
       [119,  44, 132,  11,  43],
       [120, 108, 100,  68,  14],
       [121, 127, 112,  45, 113],
       [122,  69, 103,  86,  41],
       [123,   6, 116,  46,   2],
       [124,  77,  22, 102,  41],
       [125,  34,  10,   1,  40],
       [126,  30,  14,  81,  40],
       [127, 121,  51, 112,  45],
       [128,  82,   9,  70,  24],
       [129,  75,  18, 111,  83],
       [130,  23, 102,  32,  14],
       [131,  92,  80,  35,  85],
       [132,  34,   1, 139,  43],
       [133, 109, 117, 131, 106],
       [134,  87,  62, 101,  94],
       [135,  30,  32, 130,  23],
       [136,  61, 115,  54,  83],
       [137,  44,  19, 138,  72],
       [138,  19,   7,  97, 137],
       [139,  69, 132, 103,   1],
       [140, 115,  61, 136,  54],
       [141,  55,   0,  13,  37]], dtype=int64)
```

In [43]:
```
dataset.iloc[[0, 16, 73, 55, 54, 60, 29], a]
```

Out[43]:

	class
0	1
16	1
73	2
55	1
54	1
60	2
29	1

In [44]:
```
classifier.predict(X_train[[1]])
```

Out[44]:
```
array([2], dtype=int64)
```

Predicting the Test set results

In [49]:
```
y_pred = classifier.predict(X_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1),y_test.reshape(len(y_tes
```

```
[[1 1]
 [3 3]
 [2 2]
 [1 1]
 [3 2]
 [2 2]
 [1 1]
 [2 3]
 [2 2]
 [2 2]
 [3 3]
 [3 3]
 [1 1]
 [2 2]
 [3 3]
 [2 2]
 [1 1]
 [1 1]
 [2 2]
 [1 1]
 [2 2]
 [1 1]
 [3 1]
 [2 2]
 [2 2]
 [2 2]
 [2 2]
 [2 2]
 [2 2]
 [3 3]
 [1 1]
 [1 1]
 [3 2]
 [3 1]
 [1 1]
 [1 1]]
```

Evaluating the Algorithm

Making the Confusion Matrix & Predicting Accuracy Score

In [50]:
```
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)) + ' %.')
```

```
[[12  0  2]
 [ 0 14  2]
 [ 0  1  5]]
Accuracy of our model is equal 86.11 %.
```

Making Classification Report

In [51]:
```
from sklearn.metrics import classification_report
# here f1 score is goodness of fit .
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           1       1.00      0.86      0.92        14
           2       0.93      0.88      0.90        16
           3       0.56      0.83      0.67         6

    accuracy                           0.86        36
   macro avg       0.83      0.86      0.83        36
weighted avg       0.90      0.86      0.87        36
```

Comparing Error Rate with the K Value

Parameter Tuning Using cross-validation for parameter tuning:

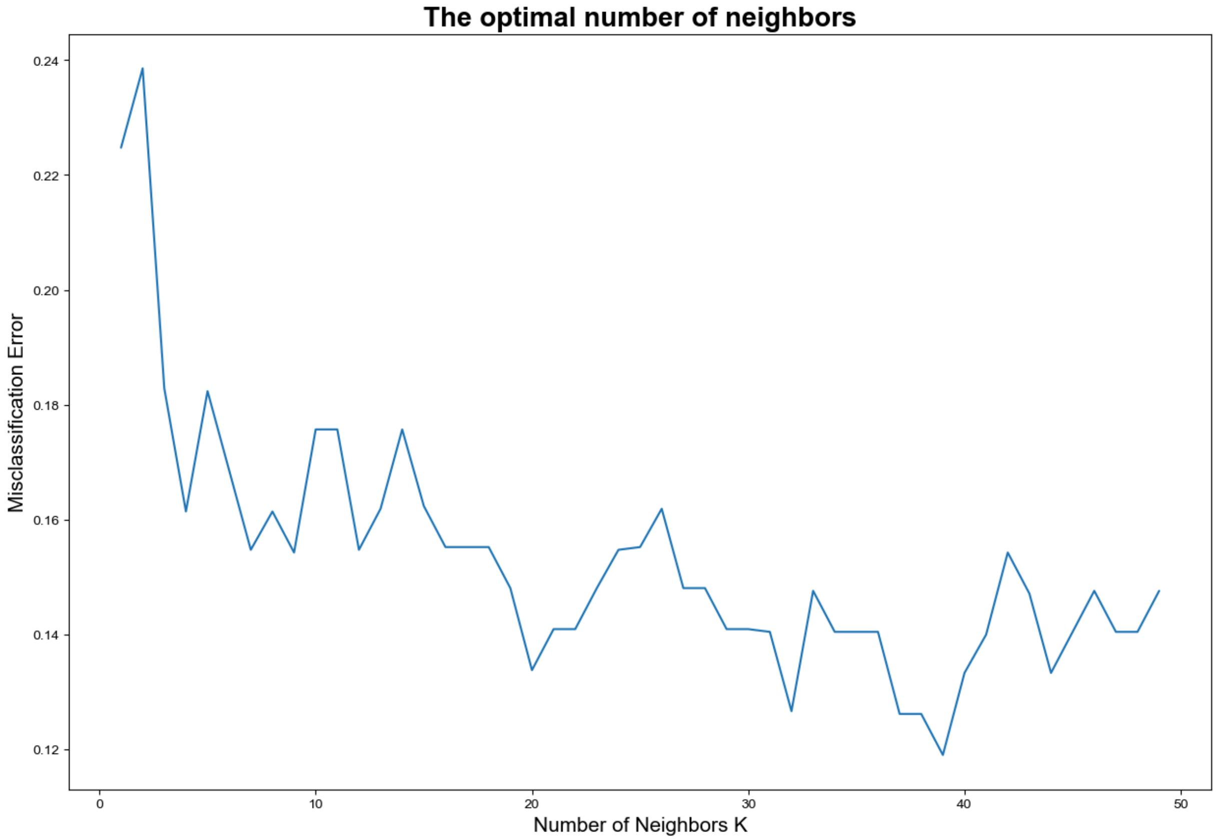
In [52]:
```
from sklearn.model_selection import cross_val_score

# creating list of K for KNN
k_list = list(range(1,50))

# 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())
```

Plot the error values against K values

In [53]:
```
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 [54]:
```
best_k = k_list[MSE.index(min(MSE))]
print("The optimal number of neighbors is %d." % best_k)
```

```
The optimal number of neighbors is 39.
```

In [55]:
```
#Class for Testing Example: {alcohol=20.1, malicacid=2.55, ash=1.77, alcalir
classifier.predict([[20.1,2.55,1.77,15.66]])
```

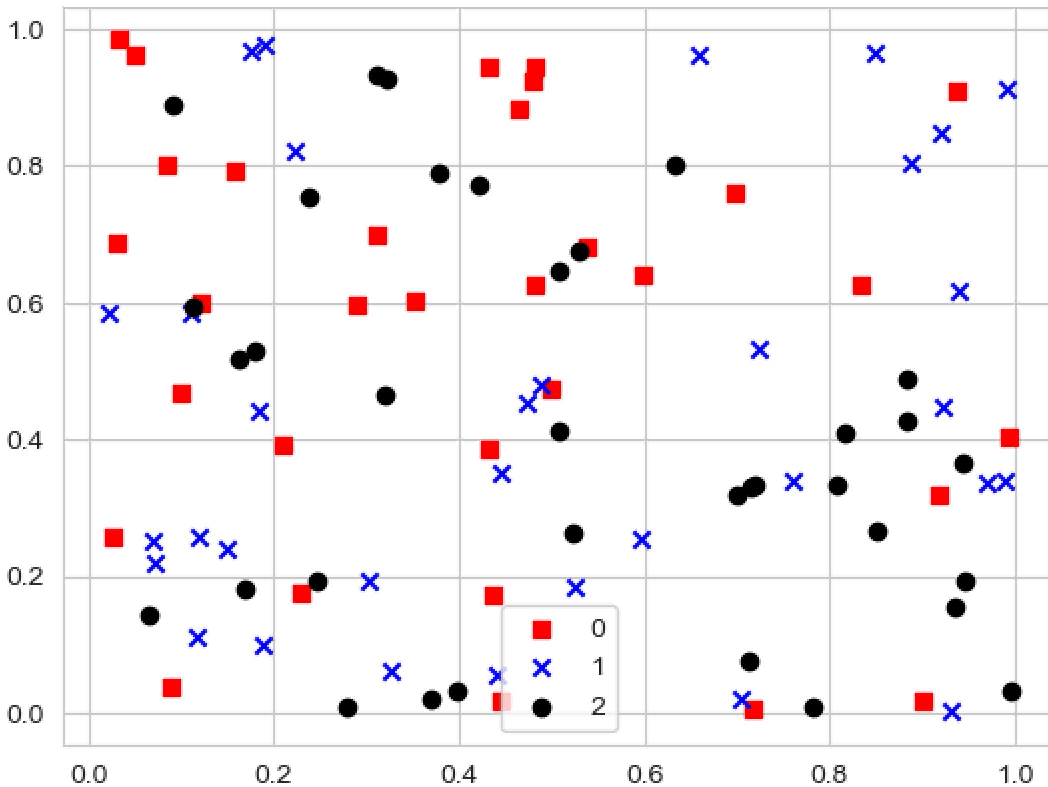
Out[55]:
```
array([3], dtype=int64)
```

Visualize Test Result of KNN

In [56]:
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
X = np.random.rand(100, 2)  # Example random data
```

```
y = np.random.randint(0, 3, size=100)  # Example random labels
markers = ('s', 'x', 'o')
colors = ('red', 'blue', 'black')
cmap = ListedColormap(colors[:len(np.unique(y))])
for idx, cl in enumerate(np.unique(y)):
    plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1], c=cmap(idx), marker=marker
plt.legend()
plt.grid(True)
plt.show()
```

```
C:\Users\Sanjana\AppData\Local\Temp\ipykernel_22928\3554044303.py:10: UserWa
rning: *c* argument looks like a single numeric RGB or RGBA sequence, which
should be avoided as value-mapping will have precedence in case its length m
atches with *x* & *y*.  Please use the *color* keyword-argument or provide a
2D array with a single row if you intend to specify the same RGB or RGBA val
ue for all points.
  plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1], c=cmap(idx), marker=markers
[idx], label=cl)
```




```
In [ ]:
```