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

Importing the dataset

In [2]:
```
column_name = ['fLength', 'fWidth', 'fSize', 'fConc', 'fConc1', 'fAsym', 'fM
               'class']
dataset = pd.read_csv('magic04.data', names=column_name)
dataset
```

Out[2]:

	fLength	fWidth	fSize	fConc	fConc1	fAsym	fM3Long	fM3Trans	
0	28.7967	16.0021	2.6449	0.3918	0.1982	27.7004	22.0110	-8.2027	4
1	31.6036	11.7235	2.5185	0.5303	0.3773	26.2722	23.8238	-9.9574	
2	162.0520	136.0310	4.0612	0.0374	0.0187	116.7410	-64.8580	-45.2160	7
3	23.8172	9.5728	2.3385	0.6147	0.3922	27.2107	-6.4633	-7.1513	1
4	75.1362	30.9205	3.1611	0.3168	0.1832	-5.5277	28.5525	21.8393	
...	...	...	...	...	...	...	...	...	
19015	21.3846	10.9170	2.6161	0.5857	0.3934	15.2618	11.5245	2.8766	
19016	28.9452	6.7020	2.2672	0.5351	0.2784	37.0816	13.1853	-2.9632	8
19017	75.4455	47.5305	3.4483	0.1417	0.0549	-9.3561	41.0562	-9.4662	3
19018	120.5135	76.9018	3.9939	0.0944	0.0683	5.8043	-93.5224	-63.8389	8
19019	187.1814	53.0014	3.2093	0.2876	0.1539	-167.3125	-168.4558	31.4755	5

19020 rows × 11 columns

Exploratory Data Analysis

In [3]:
```
dataset['class'].value_counts()
```

Out[3]:
```
class
g    12332
h     6688
Name: count, dtype: int64
```

balancing the dataset

In [4]:
```
g = dataset.groupby('class')
g = g.apply(lambda x: x.sample(g.size().min()).reset_index(drop=True))
g
```

Out[4]:

class		fLength	fWidth	fSize	fConc	fConc1	fAsym	fM3Long	fM3Tra
g	0	27.4926	8.2153	2.1931	0.5962	0.3365	-13.9212	-18.9347	-4.52
	1	51.5363	14.1394	2.8072	0.3507	0.2159	-52.3117	42.3068	-9.32
	2	19.4525	12.8362	2.3589	0.5252	0.3129	14.7674	13.7077	-3.08
	3	30.2518	22.7652	3.0443	0.2790	0.1621	6.8535	-9.0837	13.45
	4	13.5940	9.9525	2.2730	0.8213	0.4133	-14.9401	-2.4751	-11.82
...	...	...	...	...	...	...	...	...	
h	6683	149.5970	39.7260	3.1188	0.2784	0.1639	-123.4090	-109.5100	32.54
	6684	29.6989	10.8918	2.2122	0.5565	0.2925	-0.5062	14.8697	-5.80
	6685	44.5526	10.1376	2.7368	0.3593	0.1897	-0.8568	32.8751	4.00
	6686	34.4045	8.3875	2.7819	0.4116	0.2221	13.9775	50.6389	-7.90
	6687	150.7730	14.7692	2.6547	0.4142	0.2514	-67.1040	-157.1010	10.37

13376 rows × 11 columns

splitting the dataset into y and x

In [5]:
```
X = g.iloc[:, :-1].values
y = g.iloc[:, -1].values
```

In [6]:
```
X
```

Out[6]:
```
array([[ 27.4926,   8.2153,   2.1931, ...,  -4.5283,  58.4118, 143.198 ],
       [ 51.5363,  14.1394,   2.8072, ...,  -9.3238,  11.399 , 277.005 ],
       [ 19.4525,  12.8362,   2.3589, ...,  -3.085 ,   4.6487, 182.365 ],
       ...,
       [ 44.5526,  10.1376,   2.7368, ...,   4.003 ,   3.077 ,  78.6061],
       [ 34.4045,   8.3875,   2.7819, ...,  -7.9092,   7.1369, 271.5166],
       [150.773 ,  14.7692,   2.6547, ...,  10.3759,  25.6371, 162.925 ]])
```

In [7]:
```
y
```

Out[7]:
```
array(['g', 'g', 'g', ..., 'h', 'h', 'h'], dtype=object)
```

Encoding categorical data

In [8]:
```
from sklearn.preprocessing import LabelEncoder
e = LabelEncoder()
y = e.fit_transform(y)
y
```

Out[8]:
```
array([0, 0, 0, ..., 1, 1, 1])
```

normalizing the dataset

In [9]:
```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
X=sc.fit_transform(X)
X
```

Out[9]:
```
array([[-0.63095642, -0.73940571, -1.3721836 , ..., -0.20804723,
         1.0061989 , -0.67066734],
       [-0.11917557, -0.45565345, -0.07322263, ..., -0.41451202,
        -0.74156361,  1.07846765],
       [-0.80209353, -0.51807406, -1.02147894, ..., -0.14590759,
        -0.99251487, -0.15867342],
       ...,
       [-0.26782673, -0.64733147, -0.22213462, ...,  0.15925817,
        -1.05094488, -1.51501743],
       [-0.48383355, -0.73115768, -0.12673787, ..., -0.35360802,
        -0.90001277,  1.00672288],
       [ 1.99312174, -0.42548732, -0.39579476, ...,  0.43363612,
        -0.21224357, -0.41279453]])
```

splitting the dataset into training, validation and testing

In [10]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, tr
X_validation, X_test, y_validation, y_test = train_test_split(X_test, y_test
                                                              random_state=T
```

fitting the model

In [11]:
```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=1)
classifier.fit(X_train, y_train)
ypred = classifier.predict(X_test)
accuracy = classifier.score(X_test, y_test)
print(accuracy)
```

```
0.7892376681614349
```

In [12]:
```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=1)
classifier.fit(X_train, y_train)
accuracy_validation= classifier.score(X_validation, y_validation)
print(accuracy_validation)
```

```
0.7851445663010967
```

apply different values of k and choose the best one using cross validation

In [ ]:
```
from sklearn.model_selection import GridSearchCV
KN = KNeighborsClassifier()
k_range = list(range(1, 26,2))
param_grid = dict(n_neighbors=k_range)
grid=GridSearchCV(KN,param_grid,cv=10,scoring='accuracy',return_train_score=
g=grid.fit(X_train,y_train)
```

get the best value of k

In [ ]:
```
print(grid.best_params_)
accuracy=grid.best_score_*100
print("Accuracy for our training dataset with tuning is : {:.2f}%".format(ac
```

apply the best value of k on the test dataset

In [ ]:
```
knn=KNeighborsClassifier(n_neighbors=grid.best_params_['n_neighbors'])
knn.fit(X_train,y_train)
y_test_pred=knn.predict(X_test)
test_accuracy=knn.score(X_test,y_test)
print(test_accuracy)
```

apply the best value of k on the validation dataset

In [ ]:
```
knn=KNeighborsClassifier(n_neighbors=grid.best_params_['n_neighbors'])
knn.fit(X_train,y_train)
y_validation_pred=knn.predict(X_validation)
test_accuracy=knn.score(X_validation,y_validation)
print(test_accuracy)
```

plotting the confusion matrix and classification report

In [ ]:
```
from sklearn.metrics import confusion_matrix,classification_report
from sklearn.metrics import ConfusionMatrixDisplay
cm=confusion_matrix(y_test,y_test_pred)
print(cm)
print(classification_report(y_test,y_test_pred))
```

```
disp = ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=['g','h'])
disp = disp.plot(cmap=plt.cm.Blues)
```

comparing the results of the model with and without tuning

In [ ]:
```
results = pd.DataFrame(g.cv_results_)
needed_results=results[['param_n_neighbors','mean_train_score','mean_test_sc
needed_results
```