UCI Machine Learning Repository - Glass Classification Data set

For this project we will be working with the UCI Repository Glass Classification Data Set. This is a very famous data set and quite many papers have been published on the same!

We'll be trying to predict a classification- Diagnosis is Type '1' or '2'. Let's begin our understanding of implementing K-Nearest Neighbour in Python for classification.

We'll use the raw version of the data set (the one provided), and perform some categorical variables encoding (dummy variables) in python for quite many features if required.

Import Libraries

Let's import some libraries to get started!

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

The Data

Let's start by reading in the adult.csv file into a pandas dataframe.

In [16]:
```
df = pd.read_csv('glass.data', names=['id','ri','Na','Mg','Al','Si','K','Ca',
'Ba','Fe','Type'])
```

In [17]:
```
df.head()
```

Out[17]:

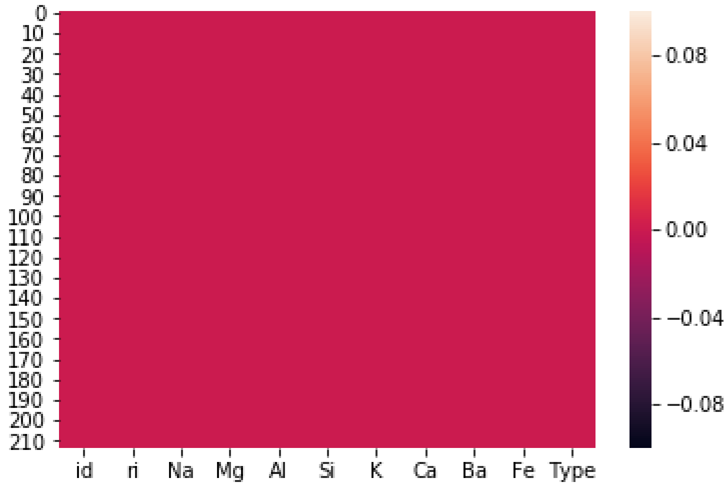
	id	ri	Na	Mg	Al	Si	K	Ca	Ba	Fe	Type
0	1	1.52101	13.64	4.49	1.10	71.78	0.06	8.75	0.0	0.0	1
1	2	1.51761	13.89	3.60	1.36	72.73	0.48	7.83	0.0	0.0	1
2	3	1.51618	13.53	3.55	1.54	72.99	0.39	7.78	0.0	0.0	1
3	4	1.51766	13.21	3.69	1.29	72.61	0.57	8.22	0.0	0.0	1
4	5	1.51742	13.27	3.62	1.24	73.08	0.55	8.07	0.0	0.0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

Exploratory Data Analysis

Performing some exploratory data analysis for better knowing the data set

In [18]:
```
sns.heatmap(df.isnull())
```

Out[18]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x84dbd68>
```



In [19]:
```
sns.pairplot(df, hue='Type')
```

Out[19]: <seaborn.axisgrid.PairGrid at 0xc8c5e80>

Data Cleaning

Cleaning the data, if any required and creating categorical variables, if any required

The 'id' column is note required, it can be dropped as it is just an id

In [24]:
```
df.drop('id', axis=1, inplace=True)
```

In [28]:
```
df.head()
```

Out[28]:

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

In [30]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 214 entries, 0 to 213
Data columns (total 10 columns):
ri      214 non-null float64
Na      214 non-null float64
Mg      214 non-null float64
Al      214 non-null float64
Si      214 non-null float64
K       214 non-null float64
Ca      214 non-null float64
Ba      214 non-null float64
Fe      214 non-null float64
Type    214 non-null int64
dtypes: float64(9), int64(1)
memory usage: 16.8 KB
```

In [33]:
```
df.columns
```

Out[33]:
```
Index(['ri', 'Na', 'Mg', 'Al', 'Si', 'K', 'Ca', 'Ba', 'Fe', 'Type'], dtype='o
bject')
```

Feature Scaling

Normalising the features using Scikit Learn Standard Scaler

In [34]:
```
from sklearn.preprocessing import StandardScaler
```

In [35]:
```
scaler = StandardScaler()
```

In [36]:
```
scaler.fit(df.drop('Type', axis=1))
```

Out[36]:
```
StandardScaler(copy=True, with_mean=True, with_std=True)
```

In [37]:
```
scaled_feature = scaler.transform(df.drop('Type', axis=1))
```

In [38]:
```
df_glass = pd.DataFrame(scaled_feature, columns=df.columns[:-1])
```

In [39]:
```
df_glass.head()
```

Out[39]:

	ri	Na	Mg	Al	Si	K	Ca	Ba	
0	0.872868	0.284953	1.254639	-0.692442	-1.127082	-0.671705	-0.145766	-0.352877	-
1	-0.249333	0.591817	0.636168	-0.170460	0.102319	-0.026213	-0.793734	-0.352877	-
2	-0.721318	0.149933	0.601422	0.190912	0.438787	-0.164533	-0.828949	-0.352877	-
3	-0.232831	-0.242853	0.698710	-0.310994	-0.052974	0.112107	-0.519052	-0.352877	-
4	-0.312045	-0.169205	0.650066	-0.411375	0.555256	0.081369	-0.624699	-0.352877	-

Training & Test Data

Creating the Training & Test Data from the Data Set

In [40]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(df_glass, df['Type'], test
_size=0.33, random_state=101)
```

Initializing, Training & Testing Model

Now inititaling the model, then traning with the training data set and testing with the test data set

In [41]:
```
from sklearn.neighbors import KNeighborsClassifier
```

In [65]:
```
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)
```

Out[65]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=4, p=2,
           weights='uniform')
```

In [66]:
```
pred = knn.predict(X_test)
```

Evaluating the Model

Now once the model has been trained and tested, we will evaluate the model using the metrics

In [67]:
```
from sklearn.metrics import classification_report, confusion_matrix
```

In [68]:
```
print(classification_report(y_test,pred))
print(confusion_matrix(y_test, pred))
```

```
             precision    recall  f1-score   support

          1       0.71      0.83      0.77        30
          2       0.58      0.82      0.68        17
          3       0.00      0.00      0.00         9
          5       1.00      0.83      0.91         6
          6       0.67      0.50      0.57         4
          7       0.75      0.60      0.67         5

avg / total       0.62      0.69      0.64        71

[[25  5  0  0  0  0]
 [ 2 14  0  0  1  0]
 [ 6  3  0  0  0  0]
 [ 0  1  0  5  0  0]
 [ 1  0  0  0  2  1]
 [ 1  1  0  0  0  3]]
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:
1135: UndefinedMetricWarning: Precision and F-score are ill-defined and being
set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
```

Finding the best probable value of 'K' - Elbow Method

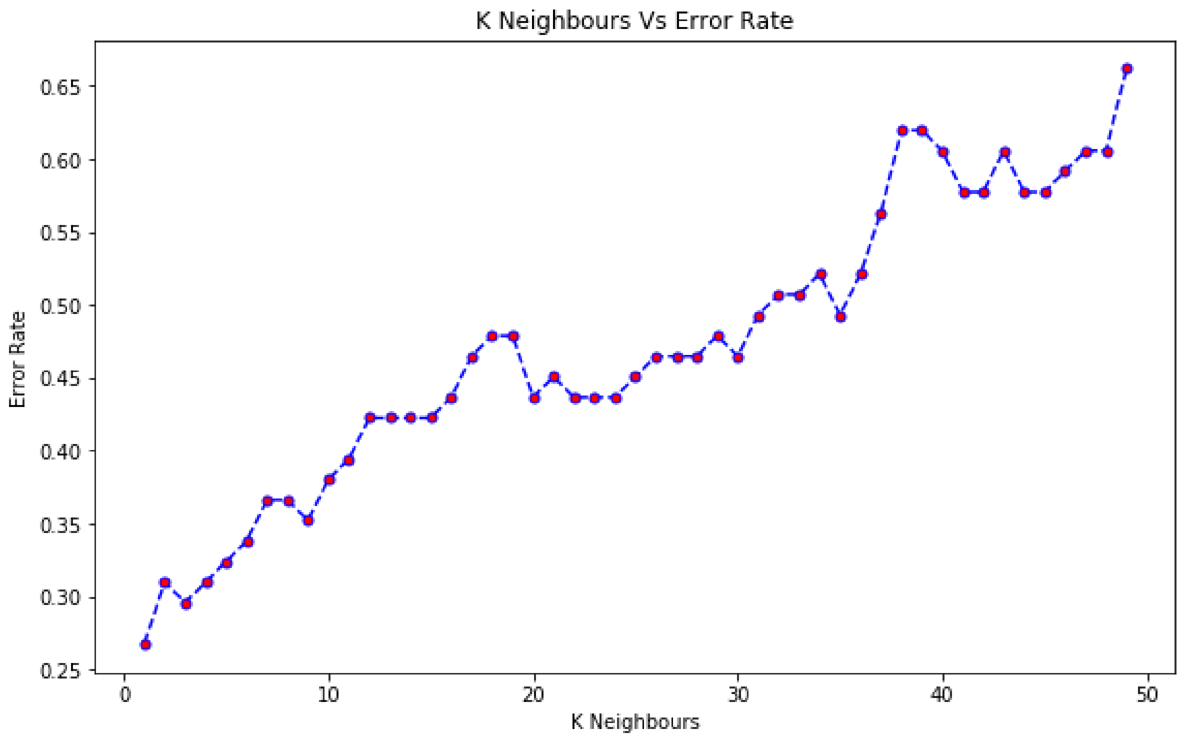
Using the elbow method to find the best probable value of 'K' for which the error is least

In [51]:
```
error_rate = []
for i in range(1,50):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train,y_train)
    pred_i = knn.predict(X_test)
    error_rate.append(np.mean(pred_i != y_test))
```

Now we have got an array 'error_rate' that contains the mean of errors for every iteration of 'K' from 1 to 50. We will use this to get the value of 'K' for which the error rate is lowest. Let's plot this information on graph so that it's easy to interpret the same.

In [52]:
```
plt.figure(figsize=(10,6))
plt.plot(range(1,50), error_rate, linestyle='--', marker='o', color='blue', ma
rkersize=5, markerfacecolor='red')
plt.title('K Neighbours Vs Error Rate')
plt.xlabel('K Neighbours')
plt.ylabel('Error Rate')
```

Out[52]: Text(0,0.5,'Error Rate')



So from the above graph, it looks k=3 is actually the good parameter. Increasing the value of 'K' increases the error rate. Hence we won't go and train the model further for higher values of 'K'