K Nearest Neighbors Project

The data for this project is artificial

Import Libraries

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

Get the Data

Read the 'KNN_Project_Data csv file into a dataframe

In [2]:

```
df = pd.read_csv('KNN_Project_Data')
df.head()
```

Out[2]:

	XVPM	GWYH	TRAT	TLLZ	IGGA	HYKR	EDFS	
0	1636.670614	817.988525	2565.995189	358.347163	550.417491	1618.870897	2147.641254	3
1	1013.402760	577.587332	2644.141273	280.428203	1161.873391	2084.107872	853.404981	4
2	1300.035501	820.518697	2025.854469	525.562292	922.206261	2552.355407	818.676686	8
3	1059.347542	1066.866418	612.000041	480.827789	419.467495	685.666983	852.867810	3
4	1018.340526	1313.679056	950.622661	724.742174	843.065903	1370.554164	905.469453	6

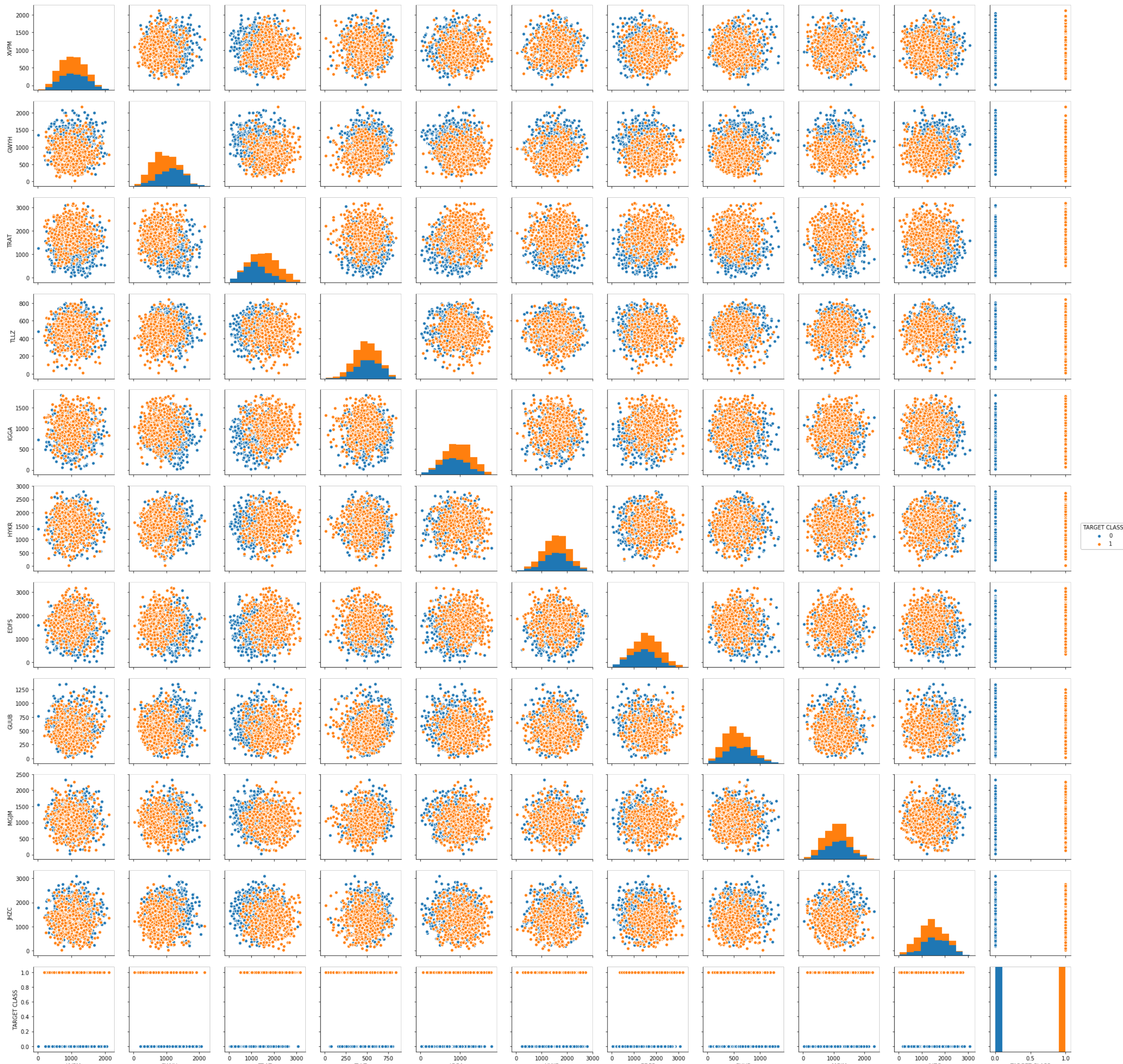
Exploratory Data Analysis

Since this data is artificial, we'll just do a large pairplot with seaborn.

Use seaborn on the dataframe to create a pairplot with the hue indicated by the TARGET CLASS column.

In [5]:
```
sns.pairplot(df, hue = 'TARGET CLASS')
```

Out[5]: <seaborn.axisgrid.PairGrid at 0x10a864cc0>



Observation : There is a clear separation between categories in the TARGET CLASS column.

Standardize the Variables

Since we don't know what the columns represent, we will give the column equal weight. For this, we will standardize the variable.

Import StandardScaler from Scikit learn.

In [4]:
```
from sklearn.preprocessing import StandardScaler
```

Create a StandardScaler() object called scaler.

In [5]:
```
scaler = StandardScaler()
```

Fit scaler to the features. For this we will consider all columns except the TARGET CLASS column.

In [7]:
```
scaler.fit(df.drop('TARGET CLASS', axis = 1))
```

Out[7]:
```
StandardScaler(copy=True, with_mean=True, with_std=True)
```

Use the .transform() method to transform the features to a scaled version.

In [8]:
```
scaled_features = scaler.transform(df.drop('TARGET CLASS', axis = 1))
```

Convert the scaled features to a dataframe and check the head of this dataframe to make sure the scaling worked. For this, we will create a dataframe with scaled features with all the columns of the df dataframe except the last column i.e. 'TARGET CLASS'.

In [10]:
```
df_feat = pd.DataFrame(scaled_features, columns = df.columns[ : -1])
df_feat.head()
```

Out[10]:

	XVPM	GWYH	TRAT	TLLZ	IGGA	HYKR	EDFS	GUUB	MGJ
0	1.568522	-0.443435	1.619808	-0.958255	-1.128481	0.138336	0.980493	-0.932794	1.0083
1	-0.112376	-1.056574	1.741918	-1.504220	0.640009	1.081552	-1.182663	-0.461864	0.2583
2	0.660647	-0.436981	0.775793	0.213394	-0.053171	2.030872	-1.240707	1.149298	2.1847
3	0.011533	0.191324	-1.433473	-0.100053	-1.507223	-1.753632	-1.183561	-0.888557	0.1623
4	-0.099059	0.820815	-0.904346	1.609015	-0.282065	-0.365099	-1.095644	0.391419	-1.3656

Train Test Split

Use train_test_split to split your data into a training set and a testing set with test_size = 30%.

In [14]:
```
X = df_feat
y = df['TARGET CLASS']
```

In [15]:
```
from sklearn.model_selection import train_test_split

```

In [16]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.3,
                                                    random_state=101)
```

Using KNN

Import KNeighborsClassifier from scikit learn.

In [17]:
```
from sklearn.neighbors import KNeighborsClassifier
```

Create a KNN model instance with n_neighbors=1

In [18]:
```
knn = KNeighborsClassifier(n_neighbors = 1)
```

Fit this KNN model to the training data.

In [19]:
```
knn.fit(X_train, y_train)
```

Out[19]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski
',
           metric_params=None, n_jobs=1, n_neighbors=1, p=2,
           weights='uniform')
```

Predictions and Evaluations

We will now evaluate the KNN model.

Use the predict method to predict values using the KNN model and X_test.

In [20]:
```
pred = knn.predict(X_test)
```

Create a confusion matrix and classification report.

In [21]:
```
from sklearn.metrics import confusion_matrix, classification_report
```

In [22]:
```
print(confusion_matrix(y_test, pred))
```

```
[[109  43]
 [ 41 107]]
```

In [23]:
```
print(classification_report(y_test, pred))
```

```
             precision    recall  f1-score   support

          0       0.73      0.72      0.72       152
          1       0.71      0.72      0.72       148

avg / total       0.72      0.72      0.72       300
```

Choosing a K Value

Let's go ahead and use the elbow method to pick a good K Value.

Create a for loop that trains various KNN models with different k values, then keep track of the error_rate for each of these models with a list.

In [24]:
```
error_rate = []

for i in range(1, 40):
    knn = KNeighborsClassifier(n_neighbors = i)
    knn.fit(X_train, y_train)
    pred_i = knn.predict(X_test)
    error_rate.append(np.mean(y_test != pred_i))
```

In [25]:
```
print(error_rate)
```

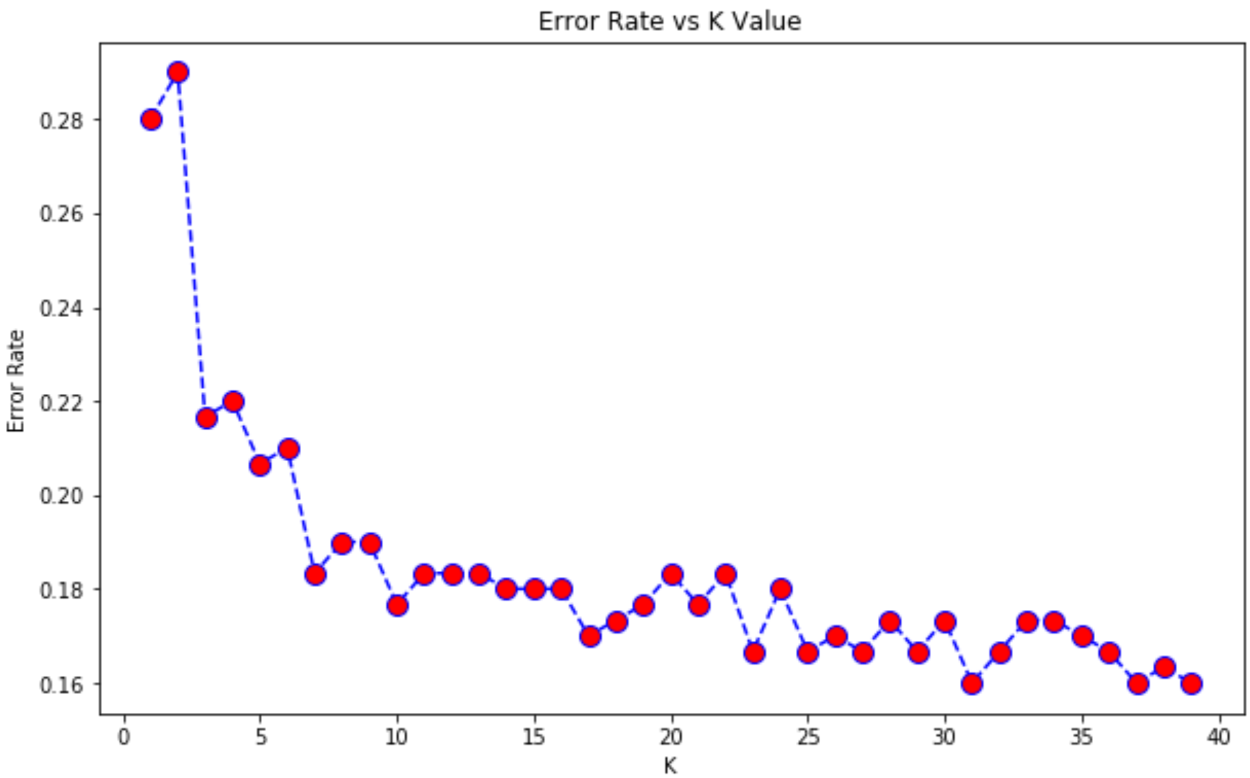
```
[0.28, 0.29, 0.21666666666666667, 0.22, 0.20666666666666667, 0.21, 0.1
8333333333333332, 0.19, 0.19, 0.17666666666666667, 0.18333333333333332
, 0.18333333333333332, 0.18333333333333332, 0.18, 0.18, 0.18, 0.17, 0.
17333333333333334, 0.17666666666666667, 0.18333333333333332, 0.1766666
6666666667, 0.18333333333333332, 0.16666666666666666, 0.18, 0.16666666
666666666, 0.17, 0.16666666666666666, 0.17333333333333334, 0.166666666
66666666, 0.17333333333333334, 0.16, 0.16666666666666666, 0.1733333333
3333334, 0.17333333333333334, 0.17, 0.16666666666666666, 0.16, 0.16333
333333333333, 0.16]
```

Now we will create the following plot using the information from the for loop.

In [26]:

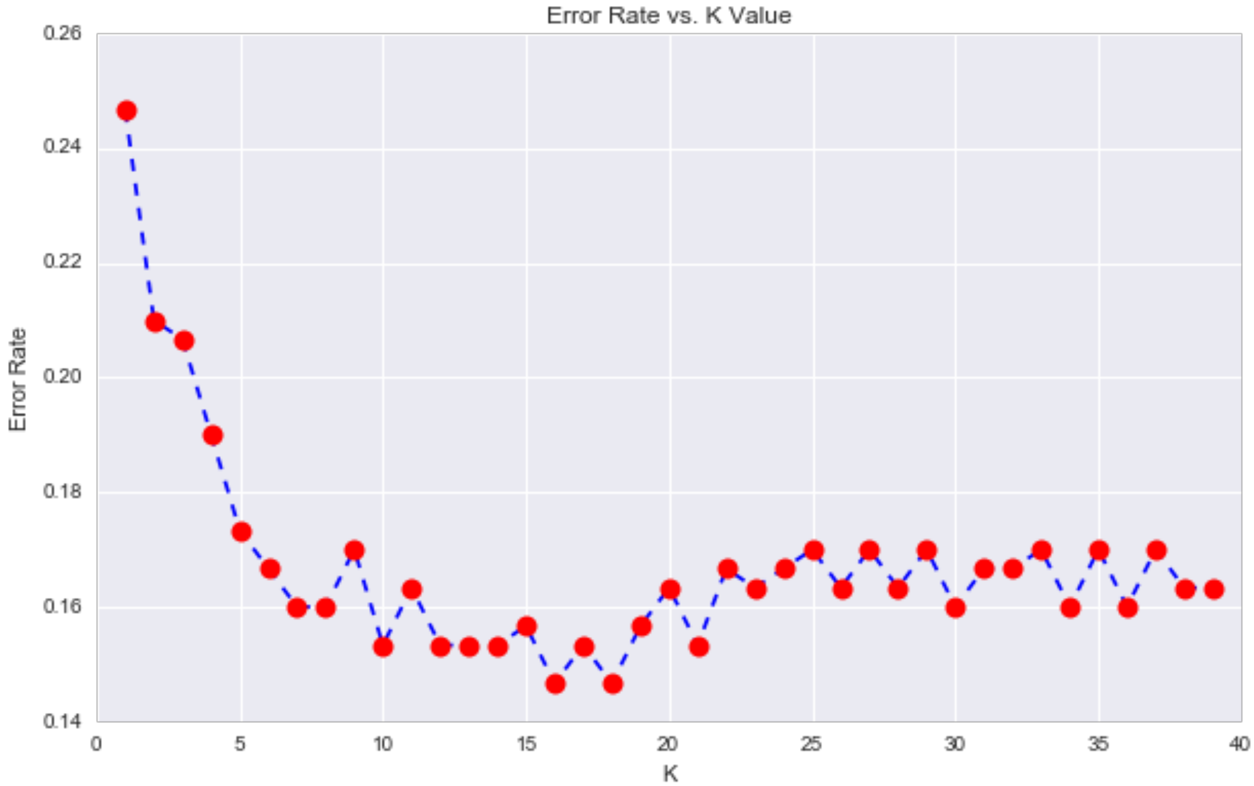
```
plt.figure(figsize = (10, 6))
plt.plot( range(1, 40),  error_rate, linestyle = 'dashed',
        marker = 'o', markerfacecolor = 'red', color = 'blue',
        markersize = 10)
plt.title('Error Rate vs K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
```

Out[26]: <matplotlib.text.Text at 0x1a1795ff98>



In [20]:
```
1
```

Out[20]: <matplotlib.text.Text at 0x11cbdb710>



Observation: After K = 30, the Error Rate seems to be steady. So we will pick K = 30.

Retrain with new K Value

Retrain the model with the best K value (i.e. K = 30) and re-do the classification report and the confusion matrix.

In [36]:

```
knn = KNeighborsClassifier(n_neighbors= 30)
knn.fit(X_train, y_train)
pred = knn.predict(X_test)
print('Confusion Matrix')
print(confusion_matrix(y_test, pred))
print('\n-----------------------------------')
print('Classification Report')
print(classification_report(y_test, pred))
```

```
Confusion Matrix
[[124  28]
 [ 24 124]]

-----------------------------------
Classification Report
             precision    recall  f1-score   support

          0       0.84      0.82      0.83       152
          1       0.82      0.84      0.83       148

avg / total       0.83      0.83      0.83       300
```

Observation: By increasing the K value from 1 to 30, we can see a lot of improvement in the model.