K Nearest Neighbors Project

Welcome to the KNN Project!

Import Libraries

Import pandas,seaborn, and the usual libraries.

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

Get the Data

** Read the 'KNN_Project_Data csv file into a dataframe **

In [2]:
```
df = pd.read_csv('KNN_Project_Data')
```

Check the head of the dataframe.

In [3]:
```
df.head()
```

Out[3]:

	XVPM	GWYH	TRAT	TLLZ	IGGA	HYKR	EDFS	
0	1636.670614	817.988525	2565.995189	358.347163	550.417491	1618.870897	2147.641254	330
1	1013.402760	577.587332	2644.141273	280.428203	1161.873391	2084.107872	853.404981	447
2	1300.035501	820.518697	2025.854469	525.562292	922.206261	2552.355407	818.676686	845
3	1059.347542	1066.866418	612.000041	480.827789	419.467495	685.666983	852.867810	341
4	1018.340526	1313.679056	950.622661	724.742174	843.065903	1370.554164	905.469453	658

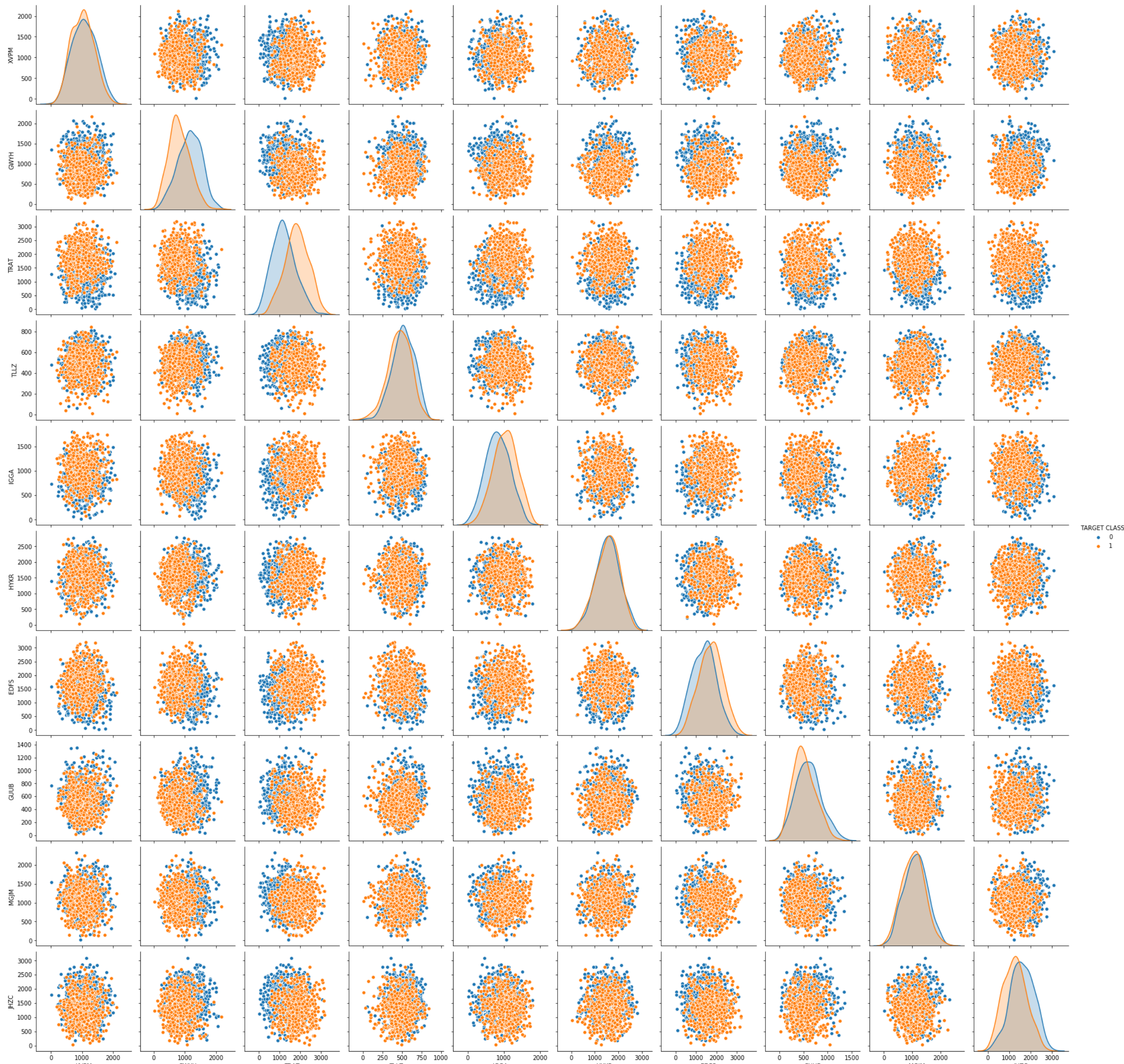
EDA

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 [4]:
```
sns.pairplot(df,hue='TARGET CLASS')
```

Out[4]:
```
<seaborn.axisgrid.PairGrid at 0x22e2e595ec8>
```



Standardize the Variables

Time to standardize the variables.

** Import StandardScaler from Scikit learn.**

In [5]:
```
from sklearn.preprocessing import StandardScaler
```

** Create a StandardScaler() object called scaler.**

In [6]:
```
scalar = StandardScaler()
```

** Fit scaler to the features.**

In [7]:
```
scalar.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 = scalar.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.

In [9]:
```
new_df = pd.DataFrame(scaled_features,columns=df.columns[:-1])
```

In [10]:
```
new_df.head()
```

Out[10]:

	XVPM	GWYH	TRAT	TLLZ	IGGA	HYKR	EDFS	GUUB	MGJM
0	1.568522	-0.443435	1.619808	-0.958255	-1.128481	0.138336	0.980493	-0.932794	1.008313
1	-0.112376	-1.056574	1.741918	-1.504220	0.640009	1.081552	-1.182663	-0.461864	0.258321
2	0.660647	-0.436981	0.775793	0.213394	-0.053171	2.030872	-1.240707	1.149298	2.184784
3	0.011533	0.191324	-1.433473	-0.100053	-1.507223	-1.753632	-1.183561	-0.888557	0.162310
4	-0.099059	0.820815	-0.904346	1.609015	-0.282065	-0.365099	-1.095644	0.391419	-1.365603

Train Test Split

Use train_test_split to split your data into a training set and a testing set.

In [11]:
```
from sklearn.model_selection import train_test_split
```

In [22]:
```
X = new_df
y=df['TARGET CLASS']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_
```

Using KNN

Import KNeighborsClassifier from scikit learn.

In [23]:
```
from sklearn.neighbors import KNeighborsClassifier
```

Create a KNN model instance with n_neighbors=1

In [24]:
```
knn = KNeighborsClassifier(n_neighbors=1)
```

Fit this KNN model to the training data.

In [25]:
```
knn.fit(X_train,y_train)
```

Out[25]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=1, p=2,
                     weights='uniform')
```

Predictions and Evaluations

Let's evaluate our KNN model!

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

In [50]:
```
pred = knn.predict(X_test)
```

** Create a confusion matrix and classification report.**

In [29]:
```
from sklearn.metrics import classification_report,confusion_matrix
```

In [32]:
```
print(confusion_matrix(y_test,pred))
```

```
[[119  44]
 [ 50 117]]
```

In [34]:
```
print(classification_report(y_test,pred))
```

```
              precision    recall  f1-score   support

           0       0.70      0.73      0.72       163
           1       0.73      0.70      0.71       167

    accuracy                           0.72       330
   macro avg       0.72      0.72      0.72       330
weighted avg       0.72      0.72      0.72       330
```

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. Refer to the lecture if you are confused on this step.**

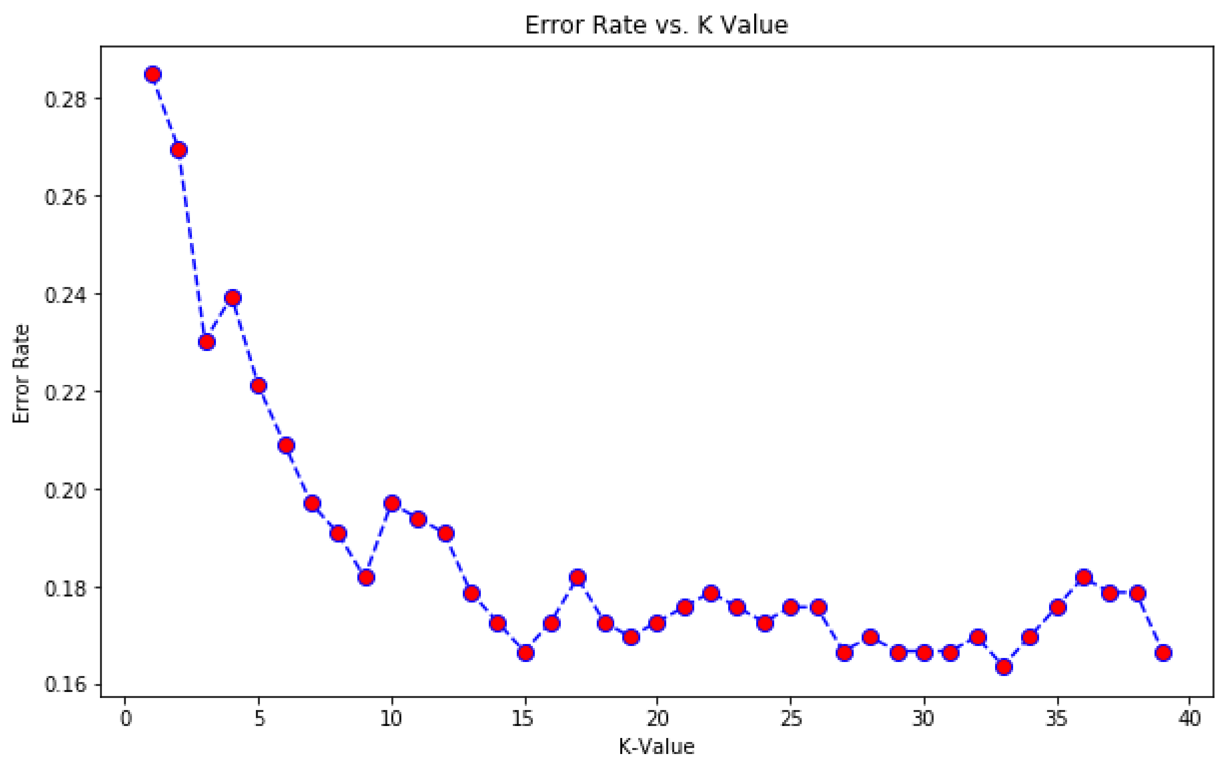
In [39]:
```
error_rate = []

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(pred_i != y_test))
#error_rate
```

Now create the following plot using the information from your for loop.

In [48]:
```
plt.figure(figsize=(10,6))
plt.plot(range(1,40),error_rate,color='b',ls='--',marker='o', markerfacecolor='r'
plt.title('Error Rate vs. K Value')
plt.xlabel('K-Value')
plt.ylabel('Error Rate')
```

Out[48]:
```
Text(0, 0.5, 'Error Rate')
```



Retrain with new K Value

Retrain your model with the best K value (up to you to decide what you want) and re-do the classification report and the confusion matrix.

In [49]:
```
knn = KNeighborsClassifier(n_neighbors=30)

knn.fit(X_train,y_train)
pred = knn.predict(X_test)

print('WITH K=30')
print('\n')
print(confusion_matrix(y_test,pred))
print('\n')
print(classification_report(y_test,pred))
```

```
WITH K=30


[[141  22]
 [ 33 134]]


              precision    recall  f1-score   support

           0       0.81      0.87      0.84       163
           1       0.86      0.80      0.83       167

    accuracy                           0.83       330
   macro avg       0.83      0.83      0.83       330
weighted avg       0.83      0.83      0.83       330
```

Great Job!