K Nearest Neighbors Project

Welcome to the KNN Project! This will be a simple project very similar to the lecture, except you'll be given another data set. Go ahead and just follow the directions below.

Import Libraries

Import pandas,seaborn, and the usual libraries.

```
In [29]: import pandas as pd
         import numpy as np
         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	33
1	1013.402760	577.587332	2644.141273	280.428203	1161.873391	2084.107872	853.404981	44
2	1300.035501	820.518697	2025.854469	525.562292	922.206261	2552.355407	818.676686	84
3	1059.347542	1066.866418	612.000041	480.827789	419.467495	685.666983	852.867810	34
4	1018.340526	1313.679056	950.622661	724.742174	843.065903	1370.554164	905.469453	65

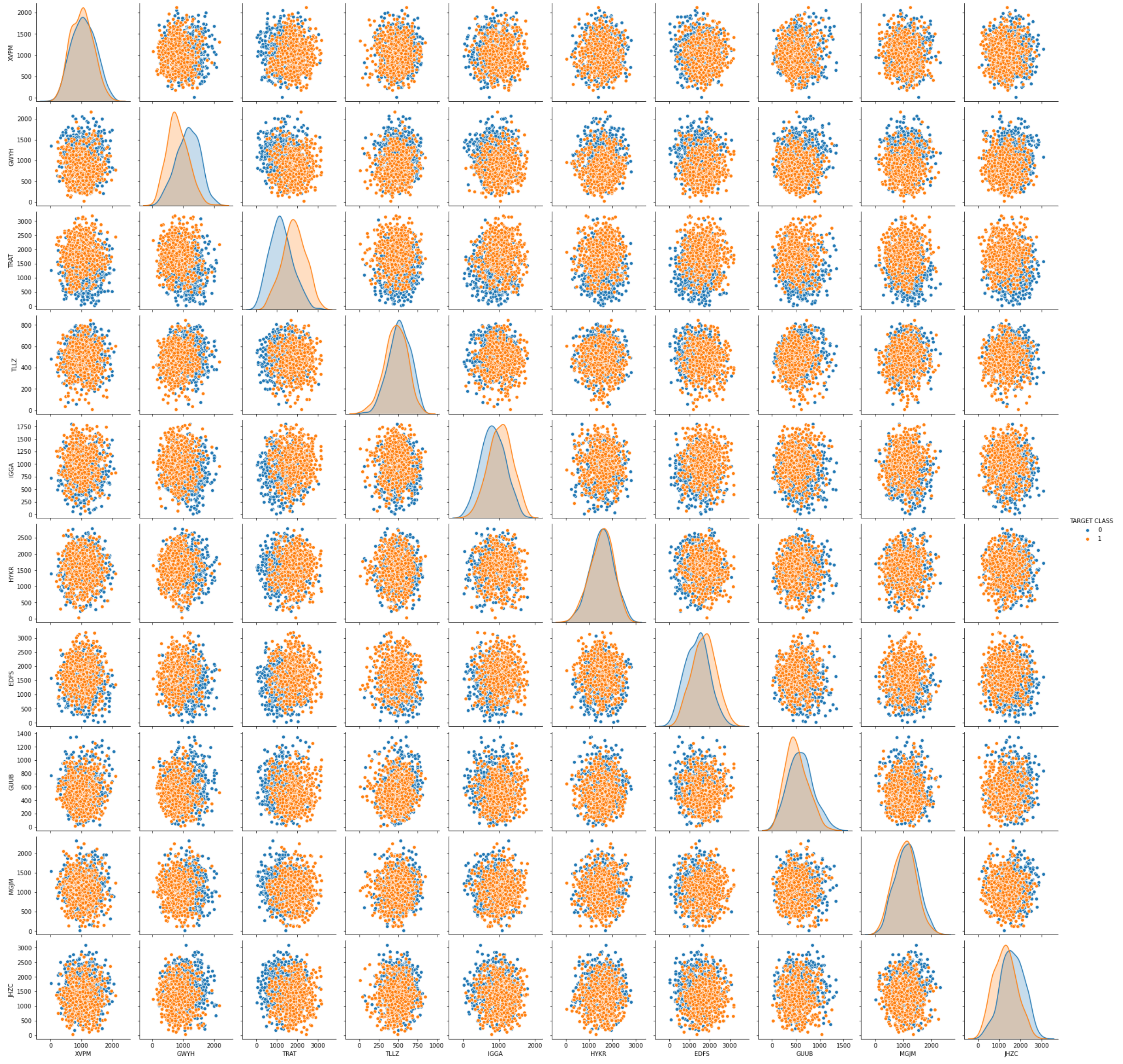
EDA

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 [6]:
```
import seaborn as sns
sns.pairplot(df,hue='TARGET CLASS')
```

Out[6]:
```
<seaborn.axisgrid.PairGrid at 0x206d2503dc0>
```



Standardize the Variables

Time to standardize the variables.

Import StandardScaler from Scikit learn.

In [7]:
```
from sklearn.preprocessing import StandardScaler
```

Create a StandardScaler() object called scaler.

In [8]:
```
scaler=StandardScaler()
```

Fit scaler to the features.

In [11]:
```
scaler.fit(df.drop('TARGET CLASS',axis=1))
```

Out[11]:
```
StandardScaler()
```

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

In [12]:
```
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.

In [14]:
```
df_feat=pd.DataFrame(scaled_features,columns=df.columns[:-1])
df_feat.head()
```

Out[14]:

	XVPM	GWYH	TRAT	TLLZ	IGGA	HYKR	EDFS	GUUB	MGJI
0	1.568522	-0.443435	1.619808	-0.958255	-1.128481	0.138336	0.980493	-0.932794	1.00831
1	-0.112376	-1.056574	1.741918	-1.504220	0.640009	1.081552	-1.182663	-0.461864	0.25832
2	0.660647	-0.436981	0.775793	0.213394	-0.053171	2.030872	-1.240707	1.149298	2.18478
3	0.011533	0.191324	-1.433473	-0.100053	-1.507223	-1.753632	-1.183561	-0.888557	0.16231
4	-0.099059	0.820815	-0.904346	1.609015	-0.282065	-0.365099	-1.095644	0.391419	-1.36560

Train Test Split

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

```
In [16]: from sklearn.model_selection import train_test_split
         X=df_feat
         y=df['TARGET CLASS']
```

```
In [17]: 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 [18]: from sklearn.neighbors import KNeighborsClassifier
```

Create a KNN model instance with n_neighbors=1

```
In [19]: knn=KNeighborsClassifier(n_neighbors=1)
```

Fit this KNN model to the training data.

```
In [20]: knn.fit(X_train,y_train)
Out[20]: KNeighborsClassifier(n_neighbors=1)
```

Predictions and Evaluations

Let's evaluate our KNN model!

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

In [21]:
```
pred=knn.predict(X_test)
```

Create a confusion matrix and classification report.

In [24]:
```
from sklearn.metrics import classification_report,confusion_matrix
print(confusion_matrix(y_test,pred))
```

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

In [25]:
```
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

    accuracy                           0.72       300
   macro avg       0.72      0.72      0.72       300
weighted avg       0.72      0.72      0.72       300
```

In [18]:
```

```

```
             precision    recall  f1-score   support

          0       0.77      0.74      0.75       152
          1       0.74      0.77      0.75       148

avg / total       0.75      0.75      0.75       300
```

Choosing a K Value

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

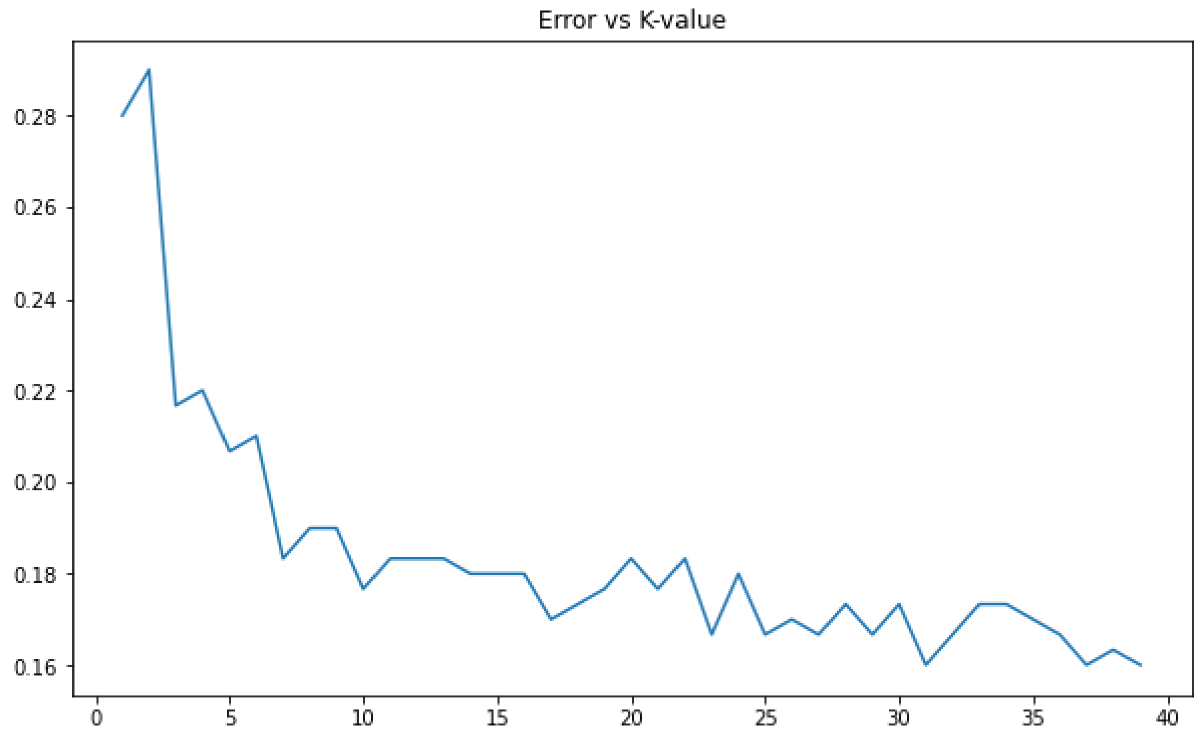
Create a for loop that trains various KNN models with different k values, then keep track of the error_rate for each of these models with a list. Refer to the lecture if you are confused on this step.

In [30]:
```
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))
```

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

In [31]:
```
plt.figure(figsize=(10,6))
plt.plot(range(1,40),error_rate)
plt.title('Error vs K-value')
```

Out[31]:
```
Text(0.5, 1.0, 'Error vs K-value')
```



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 [33]:
```
knn=KNeighborsClassifier(n_neighbors=30)
knn.fit(X_train,y_train)
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
```

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

```
              precision    recall  f1-score   support

           0       0.73      0.72      0.72       152
           1       0.71      0.72      0.72       148

    accuracy                           0.72       300
   macro avg       0.72      0.72      0.72       300
weighted avg       0.72      0.72      0.72       300
```

Great Job!