K Nearest Neighbors Project

In [3]:
```
import pandas as pd
import numpy as np
import seaborn as sb
import matplotlib.pyplot as plt
%matplotlib inline
```

In [4]:
```
df= pd.read_csv('KNN_Project_Data')
```

Check the head of the dataframe.

In [5]:
```
df.head()
```

Out[5]:

	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.

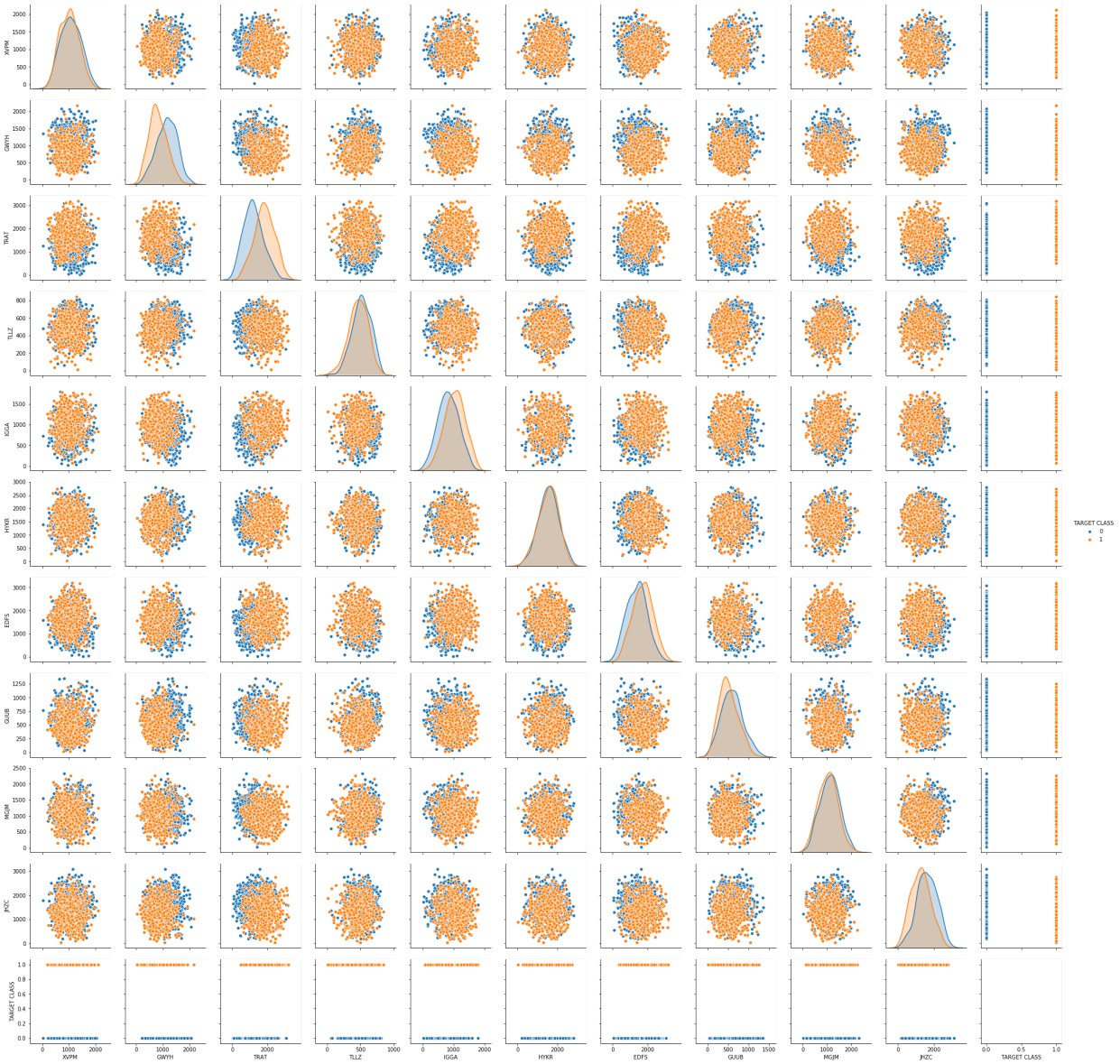
Exploratory Data Analysis

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

In [6]:
```
sb.pairplot(df,hue='TARGET CLASS')
```

```
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/statsmodels/nonparametric/kde.py:488: RuntimeWarning: invalid value encountered in true_divide
  binned = fast_linbin(X, a, b, gridsize) / (delta * nobs)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/statsmodels/nonparametric/kdetools.py:34: RuntimeWarning: invalid value encountered in double_scalars
  FAC1 = 2*(np.pi*bw/RANGE)**2
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/numpy/core/fromnumeric.py:83: RuntimeWarning: invalid value encountered in reduce
  return ufunc.reduce(obj, axis, dtype, out, **passkwargs)
```

Out[6]: <seaborn.axisgrid.PairGrid at 0x11afa04a8>



Standardize the Variables

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

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

** Fit scaler to the features.**

In [9]:
```
scaler.fit(df.drop(labels='TARGET CLASS',axis=1))
```

Out[9]:
```
StandardScaler(copy=True, with_mean=True, with_std=True)
```

transform the features to a scaled version.

In [11]:
```
sf=scaler.transform(df.drop(labels='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 [12]:
```
sfdf= pd.DataFrame(sf,columns=df.columns[:-1])
sfdf.head()
```

Out[12]:

	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

In [13]:
```
from sklearn.model_selection import train_test_split
```

In [14]:
```
X=sfdf
y=df['TARGET CLASS']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, r
```

Using KNN

In [15]:
```
from sklearn.neighbors import KNeighborsClassifier
```

Create a KNN model instance with n_neighbors=1

In [16]:
```
knn= KNeighborsClassifier(n_neighbors=1)
```

Fit this KNN model to the training data.

In [17]:
```
knn.fit(X_train,y_train)
```

Out[17]:
```
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!

In [18]:
```
predict=knn.predict(X_test)
```

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

In [19]:
```
from sklearn.metrics import classification_report,confusion_matrix
```

In [20]:
```
confusion_matrix(y_test,predict)
```

Out[20]:
```
array([[115,  47],
       [ 48, 120]])
```

In [22]:
```
print(classification_report(y_test,predict))
```

```
              precision    recall  f1-score   support

           0       0.71      0.71      0.71       162
           1       0.72      0.71      0.72       168

   micro avg       0.71      0.71      0.71       330
   macro avg       0.71      0.71      0.71       330
weighted avg       0.71      0.71      0.71       330
```

Choosing a K Value

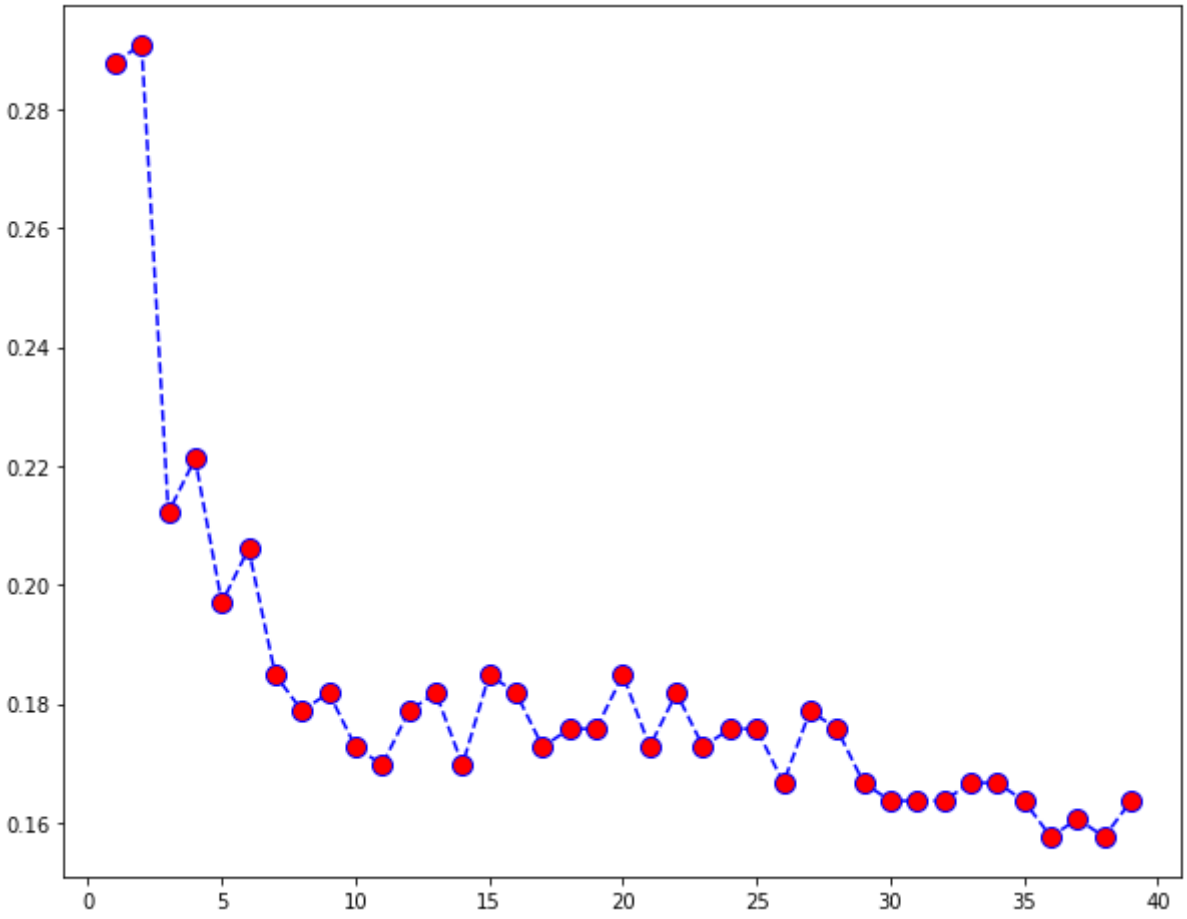
Let's go ahead and use the elbow method to pick a good K Value!

In [23]:
```
error=[]
for i in range(1,40):
    knn=KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train,y_train)
    predict=knn.predict(X_test)
    error.append(np.mean(predict != y_test))
```

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

In [26]:
```
plt.figure(figsize=(10,8))
plt.plot(range(1,40),error,color='blue', linestyle='dashed', marker='o',
         markerfacecolor='red', markersize=10)
```

Out[26]:
```
[<matplotlib.lines.Line2D at 0x12bc29390>]
```



Retrain with new K Value

In [27]:
```
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


[[135  27]
 [ 27 141]]


              precision    recall  f1-score   support

           0       0.83      0.83      0.83       162
           1       0.84      0.84      0.84       168

   micro avg       0.84      0.84      0.84       330
   macro avg       0.84      0.84      0.84       330
weighted avg       0.84      0.84      0.84       330
```