

 No description has been provided for this image

# K Nearest Neighbors Project - Solution

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 [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')
```

**Check the head of the dataframe.**

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

```
Out[3]:
```

	XVPM	GWYH	TRAT	TLLZ	IGGA	HYKR	
0	1636.670614	817.988525	2565.995189	358.347163	550.417491	1618.870897	2147.64
1	1013.402760	577.587332	2644.141273	280.428203	1161.873391	2084.107872	853.40
2	1300.035501	820.518697	2025.854469	525.562292	922.206261	2552.355407	818.67
3	1059.347542	1066.866418	612.000041	480.827789	419.467495	685.666983	852.86
4	1018.340526	1313.679056	950.622661	724.742174	843.065903	1370.554164	905.46



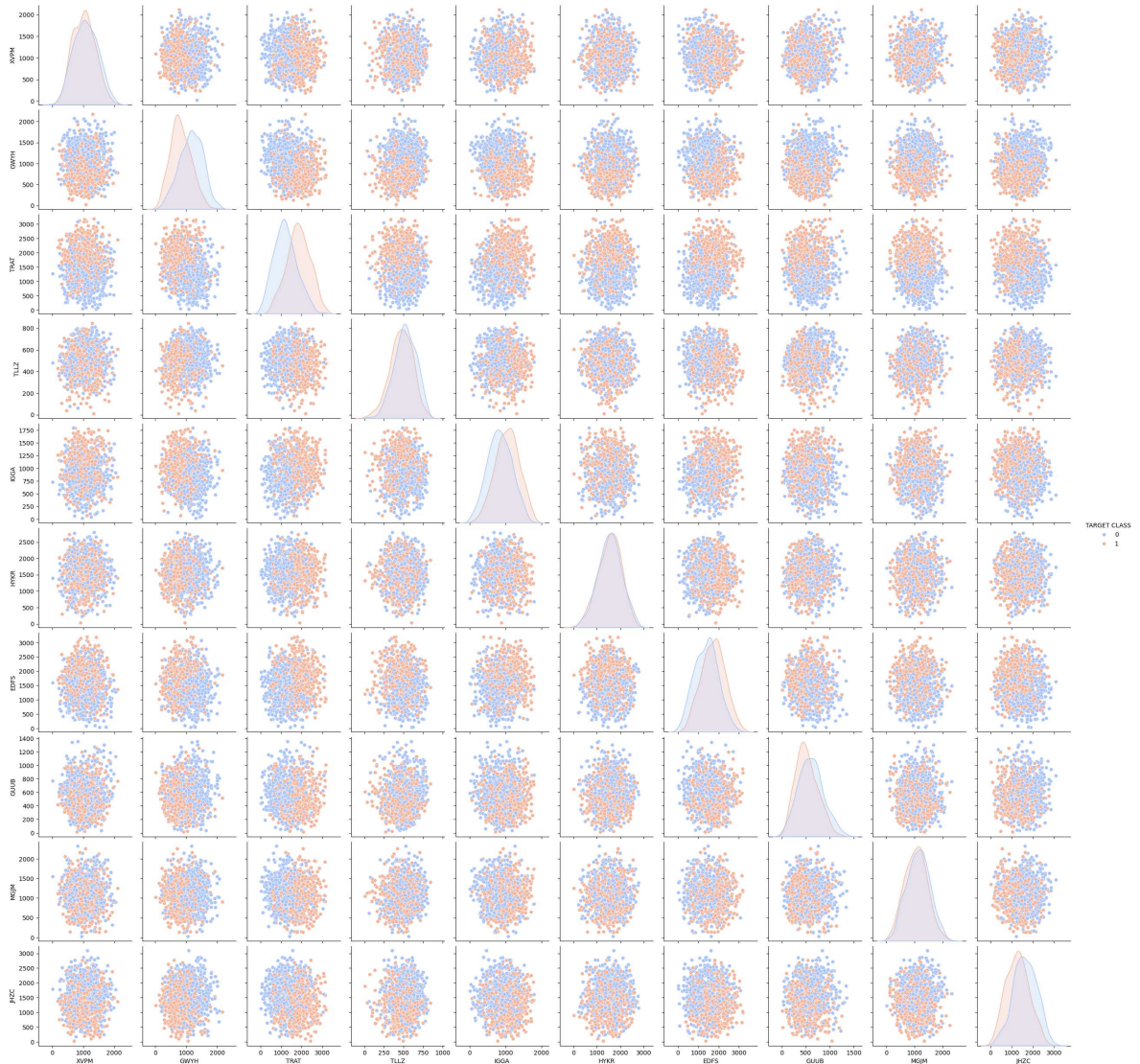
## EDA

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]: # THIS IS GOING TO BE A VERY LARGE PLOT
sns.pairplot(df, hue='TARGET CLASS', palette='coolwarm')
```

```
Out[4]: <seaborn.axisgrid.PairGrid at 0x7fb3d408f470>
```



## 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]: scaler = StandardScaler()
```

**\*\* Fit scaler to the features.\*\***

```
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.**

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

```
Out[9]:
```

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

## Train Test Split

**Use train\_test\_split to split your data into a training set and a testing set.**

```
In [10]: from sklearn.model_selection import train_test_split
```

```
In [11]: X_train, X_test, y_train, y_test = train_test_split(scaled_features,df['TARGET CLASS'],
                                                            test_size=0.30)
```

## Using KNN

**Import KNeighborsClassifier from scikit learn.**

```
In [12]: from sklearn.neighbors import KNeighborsClassifier
```

**Create a KNN model instance with n\_neighbors=1**

```
In [13]: knn = KNeighborsClassifier(n_neighbors=1)
```

**Fit this KNN model to the training data.**

```
In [14]: knn.fit(X_train,y_train)
```

```
Out[14]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                             metric_params=None, n_jobs=1, 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 [24]: pred = knn.predict(X_test)
```

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

```
In [16]: from sklearn.metrics import classification_report, confusion_matrix
```

```
In [17]: print(confusion_matrix(y_test, pred))
```

```
[[112  40]
 [ 34 114]]
```

```
In [18]: print(classification_report(y_test, pred))
```

	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 [25]: error_rate = []
```

```
# Will take some time
```

```
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.**

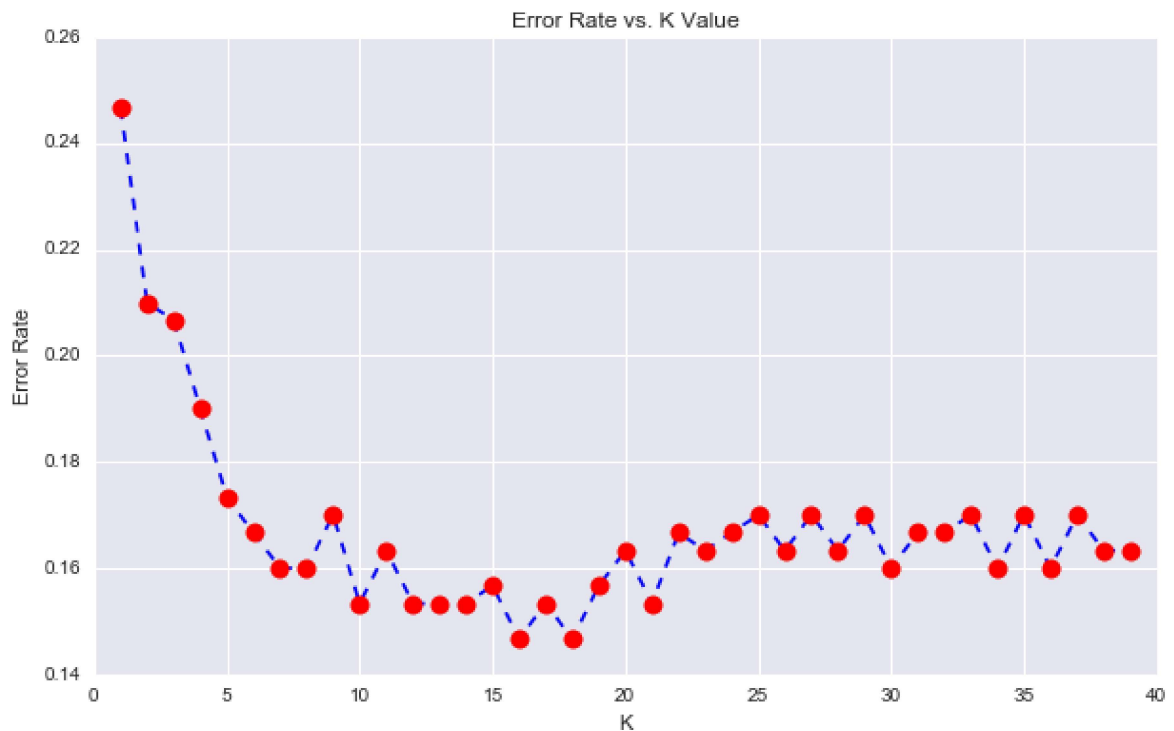
```
In [20]: plt.figure(figsize=(10,6))
plt.plot(range(1,40), error_rate, color='blue', linestyle='dashed', marker='o',
```

```

markerfacecolor='red', markersize=10)
plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')

```

Out[20]: <matplotlib.text.Text at 0x11cddb710>



## 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 [21]: # NOW WITH K=30
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

```
[[127 25]
 [ 23 125]]
```

	precision	recall	f1-score	support
0	0.85	0.84	0.84	152
1	0.83	0.84	0.84	148
avg / total	0.84	0.84	0.84	300

Great Job!