Project on K Nearest Neighbors:

Welcome to the KNN Project! This will be a beginner level project to understand the basic functionalities of KNN Classification.

KNN is a non-parametric supervised classification method which is used to make predictions about classifying a data point to a particular classe/group by proximity.

Import Libraries like pandas,seaborn, and other specific libraries needed for performing KNN analysis.

In [10]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

Get the Data

Read the 'KNN_Project_Data csv file into a dataframe

In [83]:
```
# Sample dataset used to just explain how KNN works

df=pd.read_csv("KNN_Project_Data")
```

In [84]:
```
df.head()   # *Check the head of the dataframe.**
```

Out[84]:

	XVPM	GWYH	TRAT	TLLZ	IGGA	HYKR	EDFS	GU
0	1636.670614	817.988525	2565.995189	358.347163	550.417491	1618.870897	2147.641254	330.727
1	1013.402760	577.587332	2644.141273	280.428203	1161.873391	2084.107872	853.404981	447.157
2	1300.035501	820.518697	2025.854469	525.562292	922.206261	2552.355407	818.676686	845.491
3	1059.347542	1066.866418	612.000041	480.827789	419.467495	685.666983	852.867810	341.664
4	1018.340526	1313.679056	950.622661	724.742174	843.065903	1370.554164	905.469453	658.118

Exploratory Data Analysis:

In [87]:
```
#Gather information about the datatypes,column Info,Null value count of a dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 11 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   XVPM          1000 non-null   float64
 1   GWYH          1000 non-null   float64
 2   TRAT          1000 non-null   float64
 3   TLLZ          1000 non-null   float64
 4   IGGA          1000 non-null   float64
 5   HYKR          1000 non-null   float64
 6   EDFS          1000 non-null   float64
 7   GUUB          1000 non-null   float64
 8   MGJM          1000 non-null   float64
 9   JHZC          1000 non-null   float64
 10  TARGET CLASS  1000 non-null   int64
dtypes: float64(10), int64(1)
memory usage: 86.1 KB
```

In [89]:
```
df.describe()
```

Out[89]:

	XVPM	GWYH	TRAT	TLLZ	IGGA	HYKR	EDFS	
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	10
mean	1055.071157	991.851567	1529.373525	495.107156	940.590072	1550.637455	1561.003252	5
std	370.980193	392.278890	640.286092	142.789188	345.923136	493.491988	598.608517	2
min	21.170000	21.720000	31.800000	8.450000	17.930000	27.930000	31.960000	
25%	767.413366	694.859326	1062.600806	401.788135	700.763295	1219.267077	1132.097865	3
50%	1045.904805	978.355081	1522.507269	500.197421	939.348662	1564.996551	1565.882879	5
75%	1326.065178	1275.528770	1991.128626	600.525709	1182.578166	1891.937040	1981.739411	7
max	2117.000000	2172.000000	3180.000000	845.000000	1793.000000	2793.000000	3196.000000	13

In [91]:
```
#Transpose of a dataset to view the Summary Statistics columnwise
df.describe().T
```

Out[91]:

	count	mean	std	min	25%	50%	75%	max
XVPM	1000.0	1055.071157	370.980193	21.17	767.413366	1045.904805	1326.065178	2117.0
GWYH	1000.0	991.851567	392.278890	21.72	694.859326	978.355081	1275.528770	2172.0
TRAT	1000.0	1529.373525	640.286092	31.80	1062.600806	1522.507269	1991.128626	3180.0
TLLZ	1000.0	495.107156	142.789188	8.45	401.788135	500.197421	600.525709	845.0
IGGA	1000.0	940.590072	345.923136	17.93	700.763295	939.348662	1182.578166	1793.0
HYKR	1000.0	1550.637455	493.491988	27.93	1219.267077	1564.996551	1891.937040	2793.0
EDFS	1000.0	1561.003252	598.608517	31.96	1132.097865	1565.882879	1981.739411	3196.0
GUUB	1000.0	561.346117	247.357552	13.52	381.704293	540.420379	725.762027	1352.0
MGJM	1000.0	1089.067338	402.666953	23.21	801.849802	1099.087954	1369.923665	2321.0
JHZC	1000.0	1452.521629	568.132005	30.89	1059.499689	1441.554053	1864.405512	3089.0
TARGET CLASS	1000.0	0.500000	0.500250	0.00	0.000000	0.500000	1.000000	1.0

We'll just do a large pairplot with seaborn to check for the distribution of attributes with respect to the target class variable.

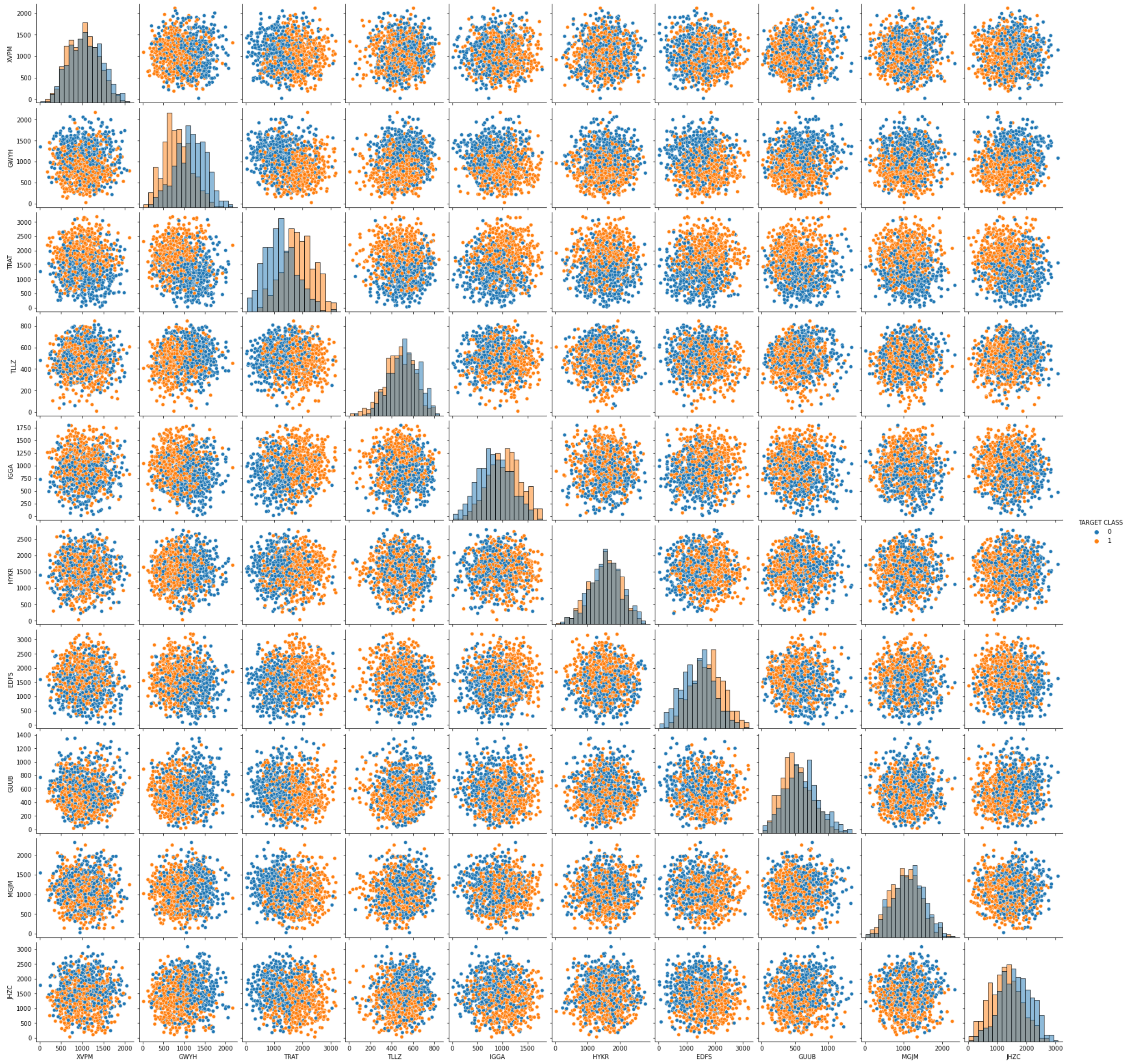
Use seaborn on the dataframe to create a pairplot with the hue indicated by the TARGET CLASS column.

In [103…

```
sns.pairplot(df,hue='TARGET CLASS',diag_kind='hist')
```

Out[103]:

```
<seaborn.axisgrid.PairGrid at 0x1c1cb62a490>
```



Standardize the Variables

Time to standardize the variables.

Import StandardScaler from Scikit learn.

In [18]:
```
from sklearn.preprocessing import StandardScaler
```

In [74]:
```
#** Create a StandardScaler() object called scaler.**

scaler=StandardScaler()
```

In [23]:
```
scaler.fit(df.drop('TARGET CLASS',axis=1))
```

Out[23]:
```
StandardScaler()
```

In [73]:
```
# **Use the .transform() method to transform the features to a scaled version.**
```

```
scaled_feat=scaler.transform(df.drop('TARGET CLASS',axis=1))
scaled_feat
```

Out[73]:

```
array([[ 1.56852168, -0.44343461,  1.61980773, ..., -0.93279392,
         1.00831307, -1.06962723],
       [-0.11237594, -1.05657361,  1.7419175 , ..., -0.46186435,
         0.25832069, -1.04154625],
       [ 0.66064691, -0.43698145,  0.77579285, ...,  1.14929806,
         2.1847836 ,  0.34281129],
       ...,
       [-0.35889496, -0.97901454,  0.83771499, ..., -1.51472604,
        -0.27512225,  0.86428656],
       [ 0.27507999, -0.99239881,  0.0303711 , ..., -0.03623294,
         0.43668516, -0.21245586],
       [ 0.62589594,  0.79510909,  1.12180047, ..., -1.25156478,
        -0.60352946, -0.87985868]])
```

In [75]:

```
#Convert the scaled features to a dataframe and check the head of this dataframe to mc

df1=pd.DataFrame(scaled_feat,columns=df.columns[:-1])
df1.head()
df1.columns
```

Out[75]:

```
Index(['XVPM', 'GWYH', 'TRAT', 'TLLZ', 'IGGA', 'HYKR', 'EDFS', 'GUUB', 'MGJM',
       'JHZC'],
      dtype='object')
```

Train Test Split

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

In [32]:

```
from sklearn.model_selection import train_test_split
```

In [35]:

```
X_train, X_test, y_train, y_test = train_test_split(scaled_feat,df['TARGET CLASS'], te
```

Using KNN

Import KNeighborsClassifier from scikit learn.

In [76]:

```
from sklearn.neighbors import KNeighborsClassifier
```

Initially, Create a KNN model instance with n_neighbors=1 to see the classification

In [38]:

```
knn=KNeighborsClassifier(n_neighbors=1)
```

In [78]:

```
#Fit this KNN model to the training data.
knn.fit(X_train,y_train)
```

Out[78]:

```
KNeighborsClassifier(n_neighbors=28)
```

Predictions and Evaluations¶

In [79]:
```
#Use the predict method to predict values using your KNN model and X_test.
pred=knn.predict(X_test)
```

In [44]:
```
#** Create a confusion matrix and classification report.**

from sklearn.metrics import classification_report,confusion_matrix
print(confusion_matrix(y_test,pred))
```

```
[[120  47]
 [ 29 104]]
```

In [46]:
```
print(classification_report(y_test,pred))
```

```
              precision    recall  f1-score   support

           0       0.81      0.72      0.76       167
           1       0.69      0.78      0.73       133

    accuracy                           0.75       300
   macro avg       0.75      0.75      0.75       300
weighted avg       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.

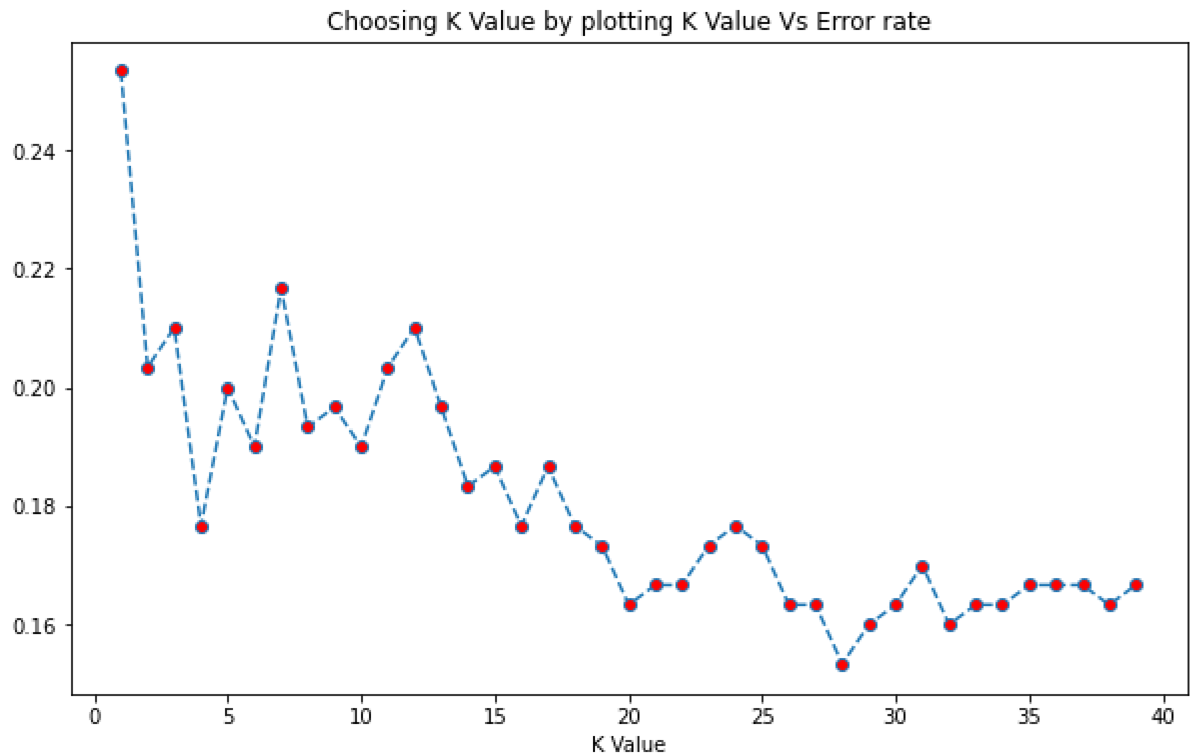
In [49]:
```
error_rate=[]

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

In [82]:
```
#Plot the Graph to show the error rate at various K values ranging from 1 to 40

plt.figure(figsize=(10,6))
plt.plot(range(1,40),error_rate,marker='o',markerfacecolor='red',linestyle='dashed')
plt.xlabel('K Value')
plt.ylabel=('Error_Rate')
plt.title("Choosing K Value by plotting K Value Vs Error rate")
```

Out[82]:
```
Text(0.5, 1.0, 'Choosing K Value by plotting K Value Vs 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 [70]:

```
knn=KNeighborsClassifier(n_neighbors=28)
knn.fit(X_train,y_train)
pred=knn.predict(X_test)
print("With K=28")
print("\n")
print("Confusion Matrix:")
print(confusion_matrix(y_test,pred))

print(classification_report(y_test,pred))
```

```
With K=28


Confusion Matrix:
[[135  32]
 [ 14 119]]
              precision    recall  f1-score   support

           0       0.91      0.81      0.85       167
           1       0.79      0.89      0.84       133

    accuracy                           0.85       300
   macro avg       0.85      0.85      0.85       300
weighted avg       0.85      0.85      0.85       300
```

Inference:

Higher the K Value, lesser the error rate here and hence,we were able to squeeze some more performance out of our model by tuning to a better K value!

Hope this project gives a basic idea on KNN Classification