Assignment-3: K- Nearest Neighbors

A. Get the Data:

1. Import pandas, seaborn, and the usual libraries.

In [1]:

```
import pandas as pd
import seaborn as sb
import matplotlib as plt
```

2. Read the KNN_Project_Data csv file into a dataframe.

In [2]:

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

Out[2]:

	XVPM	GWYH	TRAT	TLLZ	IGGA	HYKR	EDFS
0	1636.670614	817.988525	2565.995189	358.347163	550.417491	1618.870897	2147.641254
1	1013.402760	577.587332	2644.141273	280.428203	1161.873391	2084.107872	853.404981
2	1300.035501	820.518697	2025.854469	525.562292	922.206261	2552.355407	818.676686
3	1059.347542	1066.866418	612.000041	480.827789	419.467495	685.666983	852.867810
4	1018.340526	1313.679056	950.622661	724.742174	843.065903	1370.554164	905.469453
...	...	...	...	...	...	...	..
995	1343.060600	1289.142057	407.307449	567.564764	1000.953905	919.602401	485.269059
996	938.847057	1142.884331	2096.064295	483.242220	522.755771	1703.169782	2007.548635
997	921.994822	607.996901	2065.482529	497.107790	457.430427	1577.506205	1659.197738
998	1157.069348	602.749160	1548.809995	646.809528	1335.737820	1455.504390	2788.366441
999	1287.150025	1303.600085	2247.287535	664.362479	1132.682562	991.774941	2007.676371

1000 rows × 11 columns

3. 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
1	1013.402760	577.587332	2644.141273	280.428203	1161.873391	2084.107872	853.404981
2	1300.035501	820.518697	2025.854469	525.562292	922.206261	2552.355407	818.676686
3	1059.347542	1066.866418	612.000041	480.827789	419.467495	685.666983	852.867810
4	1018.340526	1313.679056	950.622661	724.742174	843.065903	1370.554164	905.469453

In [4]:

```
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
```

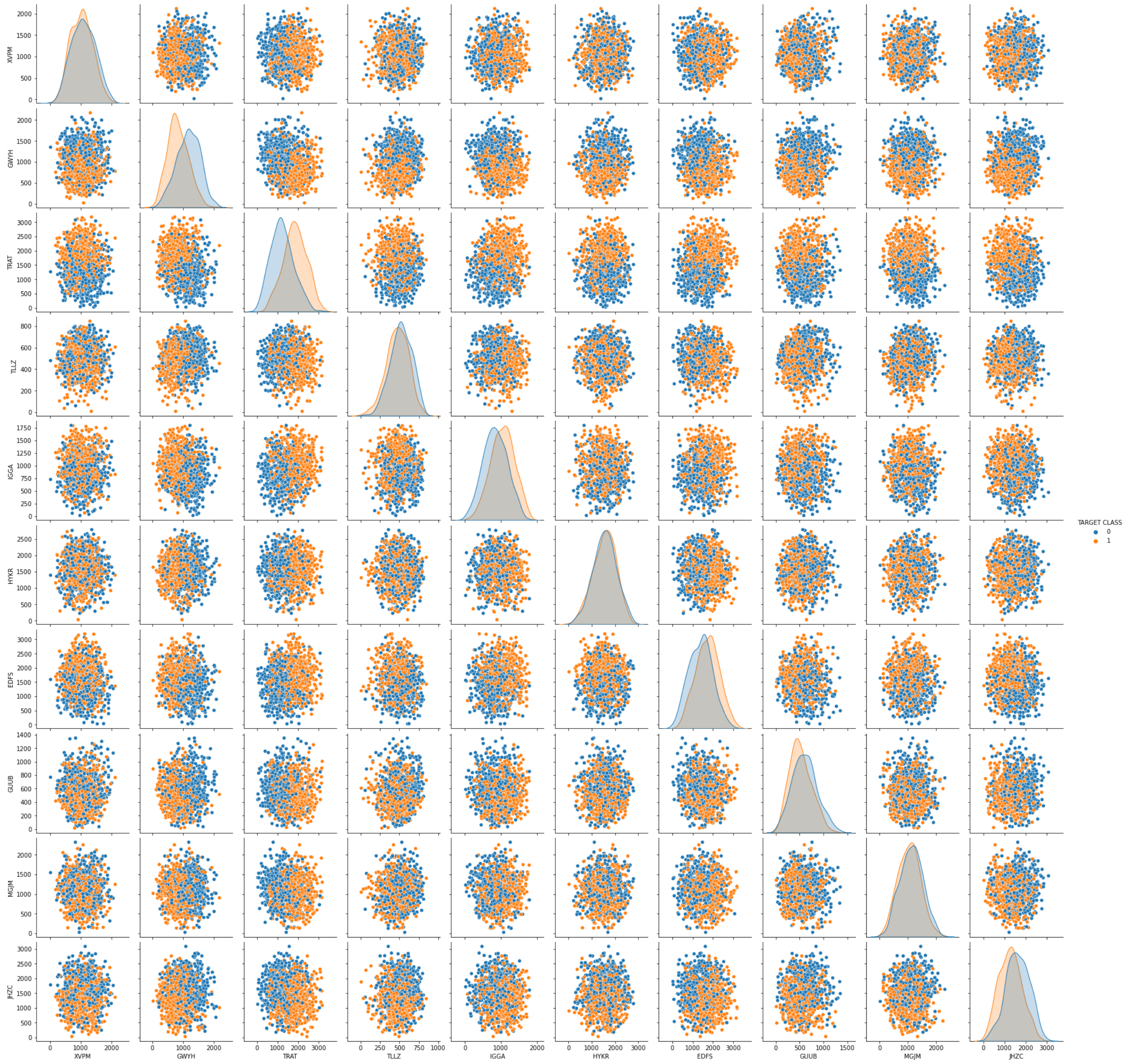
4. Create a pairplot with the hue indicated by the TARGET CLASS column using seaborn on the dataframe.

In [5]:

```
sb.pairplot(df,hue = 'TARGET CLASS')
```

Out[5]:

<seaborn.axisgrid.PairGrid at 0x13c773d01c0>



B. Standardize the Variables:

1. Import StandardScaler from Scikit learn.

In [6]:

```
from sklearn.preprocessing import StandardScaler
```

2. Create a StandardScaler() object called scaler.

In [7]:

```
scaler = StandardScaler()
```

3. Fit scaler to the features.

In [8]:

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

Out[8]:

```
StandardScaler()
```

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

In [9]:

```
scaled_features = scaler.transform(df.drop("TARGET CLASS",axis=1))
scaled_features
```

Out[9]:

```
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]])
```

5. Convert the scaled features to a dataframe and check the head of this dataframe to make sure the scaling worked.

In [10]:

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

Out[10]:

	XVPM	GWYH	TRAT	TLLZ	IGGA	HYKR	EDFS	GUUB	M
0	1.568522	-0.443435	1.619808	-0.958255	-1.128481	0.138336	0.980493	-0.932794	1.0
1	-0.112376	-1.056574	1.741918	-1.504220	0.640009	1.081552	-1.182663	-0.461864	0.2
2	0.660647	-0.436981	0.775793	0.213394	-0.053171	2.030872	-1.240707	1.149298	2.1
3	0.011533	0.191324	-1.433473	-0.100053	-1.507223	-1.753632	-1.183561	-0.888557	0.1
4	-0.099059	0.820815	-0.904346	1.609015	-0.282065	-0.365099	-1.095644	0.391419	-1.3
...	...	...	...	...	...	...	...	...	
995	0.776682	0.758234	-1.753322	0.507699	0.174588	-1.279354	-1.797957	0.431419	0.0
996	-0.313446	0.385206	0.885502	-0.083136	-1.208486	0.309242	0.746346	-0.112571	-1.7
997	-0.358895	-0.979015	0.837715	0.014018	-1.397424	0.054473	0.164120	-1.514726	-0.2
998	0.275080	-0.992399	0.030371	1.062954	1.142871	-0.192872	2.051386	-0.036233	0.4
999	0.625896	0.795109	1.121800	1.185944	0.555582	-1.133032	0.746559	-1.251565	-0.6

1000 rows × 10 columns

6. Use train_test_split to split your data into a training set and a testing set.(Make a test set considering 30% of total data)

In [12]:

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

In [13]:

```
x_train,x_test,y_train,y_test = train_test_split(scaled_features,df["TARGET CLASS"])
```

In [14]:

```
x_train,x_test,y_train,y_test
```

Out[14]:

```
(array([[-0.75932987, -0.54733953,  1.26902142, ..., -0.27150425,
          1.54218242, -1.50410876],
        [ 1.75117496, -1.98734674, -0.09533969, ...,  0.12328865,
          0.26701879, -0.74928025],
        [-0.11770832,  0.12501291,  1.44328362, ..., -0.16693329,
         -1.38493243,  1.28466961],
        ...,
        [ 0.27331146, -1.30707464, -1.22382368, ..., -0.73389958,
          0.42021991, -0.49288768],
        [ 0.60707114, -1.7447693 , -0.64349598, ...,  0.42033276,
          0.04461469,  1.0133517 ],
        [ 0.23676761,  0.23177147,  0.06111853, ...,  1.14636895,
          0.92842414,  2.88189577]]),
 array([[-0.05421249, -0.73652072,  0.44749513, ..., -0.86293443,
          0.12217923, -0.33651487],
        [-0.93959937,  0.36160398,  1.07883621, ..., -1.02934544,
         -0.22441486,  0.73022565],
        [ 1.11428478, -1.75461161,  0.59615646, ...,  0.08694836,
         -0.29535009, -0.96230747],
        ...,
        [-2.21893145,  1.17740997,  1.41027194, ..., -0.09243996,
          0.6484117 , -0.1131297 ],
        [ 0.81299895, -0.67990747, -0.0148825 , ...,  0.02154848,
          0.74900056, -0.53964585],
        [ 0.90597661,  0.59026744, -0.26815407, ..., -1.87908857,
          2.05568014,  0.5080592 ]]),
 348    1
 825    0
 85     0
 351    1
 485    1
       ..
 763    0
 616    1
 414    1
 608    0
 115    0
 Name: TARGET CLASS, Length: 750, dtype: int64,
 120    1
 495    1
 229    1
 86     1
 671    0
       ..
 535    0
 325    0
 469    1
 654    0
 704    0
 Name: TARGET CLASS, Length: 250, dtype: int64)
```

C. Using KNN:

1. Import KNeighborsClassifier from scikit learn

In [15]:

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

2. Create a KNN model instance with n_neighbors=1

In [16]:

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

3. Fit this KNN model to the training data.

In [17]:

```
knn.fit(x_train,y_train)
```

Out[17]:

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

D. Predictions and Evaluations

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

In [18]:

```
pred = knn.predict(x_test)
```

2. Create a confusion matrix and classification report.

In [20]:

```
from sklearn.metrics import classification_report,confusion_matrix
```

In [21]:

```
print(confusion_matrix(y_test,pred))
```

```
[[ 85  38]
 [ 26 101]]
```

In [22]:

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

```
              precision    recall  f1-score   support

           0       0.77      0.69      0.73       123
           1       0.73      0.80      0.76       127

    accuracy                           0.74       250
   macro avg       0.75      0.74      0.74       250
weighted avg       0.75      0.74      0.74       250
```

In [ ]:

```

```