K Nearest Neighbors (KNN)

1. Import data set

In [1]:
```
import pandas as pd
df = pd.read_csv("Classified Data",index_col=0)
```

In [2]:
```
df.head()
```

Out[2]:

	WTT	PTI	EQW	SBI	LQE	QWG	FDJ	PJF	HQE	NXJ	TARGET CLASS
0	0.913917	1.162073	0.567946	0.755464	0.780862	0.352608	0.759697	0.643798	0.879422	1.231409	1
1	0.635632	1.003722	0.535342	0.825645	0.924109	0.648450	0.675334	1.013546	0.621552	1.492702	0
2	0.721360	1.201493	0.921990	0.855595	1.526629	0.720781	1.626351	1.154483	0.957877	1.285597	0
3	1.234204	1.386726	0.653046	0.825624	1.142504	0.875128	1.409708	1.380003	1.522692	1.153093	1
4	1.279491	0.949750	0.627280	0.668976	1.232537	0.703727	1.115596	0.646691	1.463812	1.419167	1

2. Show basic information

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

Out[3]:

	WTT	PTI	EQW	SBI	LQE	QWG	FDJ	PJF	HQE	NXJ	TARGET CLASS
0	0.913917	1.162073	0.567946	0.755464	0.780862	0.352608	0.759697	0.643798	0.879422	1.231409	1
1	0.635632	1.003722	0.535342	0.825645	0.924109	0.648450	0.675334	1.013546	0.621552	1.492702	0
2	0.721360	1.201493	0.921990	0.855595	1.526629	0.720781	1.626351	1.154483	0.957877	1.285597	0
3	1.234204	1.386726	0.653046	0.825624	1.142504	0.875128	1.409708	1.380003	1.522692	1.153093	1
4	1.279491	0.949750	0.627280	0.668976	1.232537	0.703727	1.115596	0.646691	1.463812	1.419167	1

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1000 entries, 0 to 999
Data columns (total 11 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   WTT           1000 non-null   float64
 1   PTI           1000 non-null   float64
 2   EQW           1000 non-null   float64
 3   SBI           1000 non-null   float64
 4   LQE           1000 non-null   float64
 5   QWG           1000 non-null   float64
 6   FDJ           1000 non-null   float64
 7   PJF           1000 non-null   float64
 8   HQE           1000 non-null   float64
 9   NXJ           1000 non-null   float64
 10  TARGET CLASS  1000 non-null   int64  
dtypes: float64(10), int64(1)
memory usage: 93.8 KB
```

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

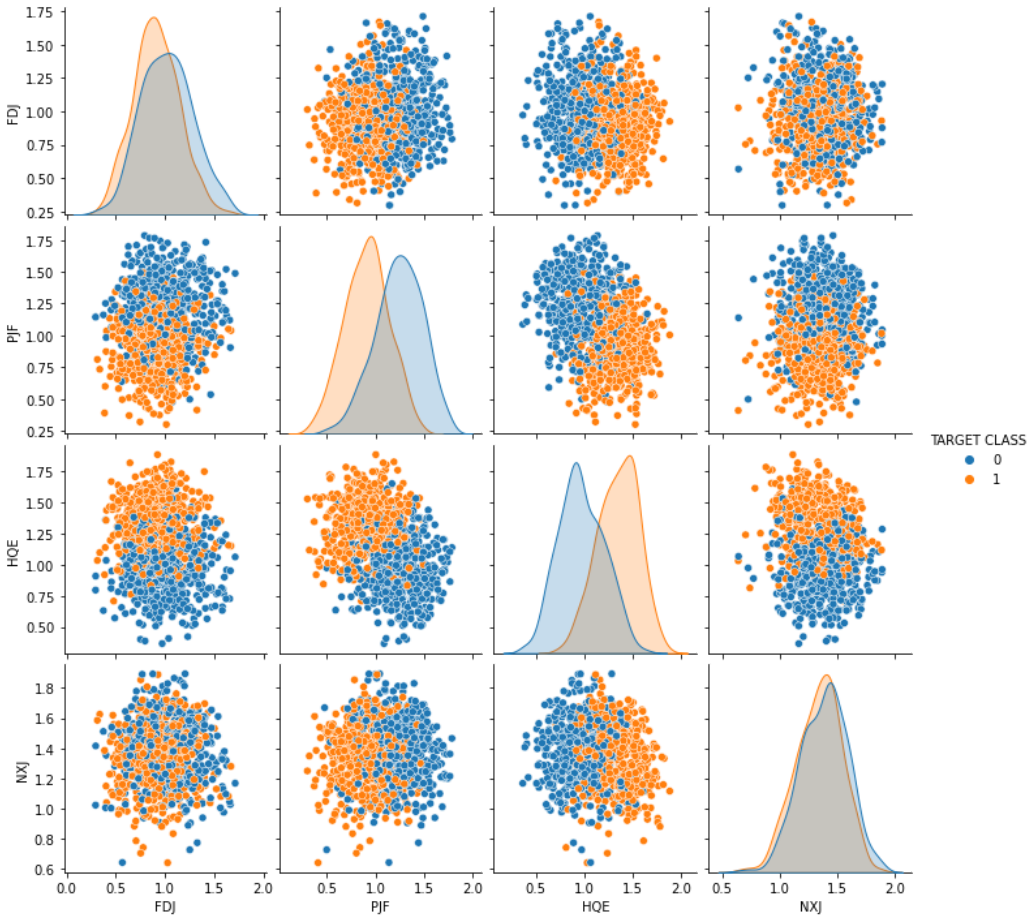
Out[5]:

	WTT	PTI	EQW	SBI	LQE	QWG	FDJ	PJF	HQE	NXJ	TARGET CLASS
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.00000
mean	0.949682	1.114303	0.834127	0.682099	1.032336	0.943534	0.963422	1.071960	1.158251	1.362725	0.50000
std	0.289635	0.257085	0.291554	0.229645	0.243413	0.256121	0.255118	0.288982	0.293738	0.204225	0.50025
min	0.174412	0.441398	0.170924	0.045027	0.315307	0.262389	0.295228	0.299476	0.365157	0.639693	0.00000
25%	0.742358	0.942071	0.615451	0.515010	0.870855	0.761064	0.784407	0.866306	0.934340	1.222623	0.00000
50%	0.940475	1.118486	0.813264	0.676835	1.035824	0.941502	0.945333	1.065500	1.165556	1.375368	0.50000
75%	1.163295	1.307904	1.028340	0.834317	1.198270	1.123060	1.134852	1.283156	1.383173	1.504832	1.00000
max	1.721779	1.833757	1.722725	1.634884	1.650050	1.666902	1.713342	1.785420	1.885690	1.893950	1.00000

3. Visualize data

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

Out[6]: <seaborn.axisgrid.PairGrid at 0x7fa953722d60>



4. Prepare data

In [7]:
```
df.head()
```

Out[7]:

	WTT	PTI	EQW	SBI	LQE	QWG	FDJ	PJF	HQE	NXJ	TARGET CLASS
0	0.913917	1.162073	0.567946	0.755464	0.780862	0.352608	0.759697	0.643798	0.879422	1.231409	1
1	0.635632	1.003722	0.535342	0.825645	0.924109	0.648450	0.675334	1.013546	0.621552	1.492702	0
2	0.721360	1.201493	0.921990	0.855595	1.526629	0.720781	1.626351	1.154483	0.957877	1.285597	0
3	1.234204	1.386726	0.653046	0.825624	1.142504	0.875128	1.409708	1.380003	1.522692	1.153093	1
4	1.279491	0.949750	0.627280	0.668976	1.232537	0.703727	1.115596	0.646691	1.463812	1.419167	1

In [8]:
```
x = df.drop('TARGET CLASS',axis=1)
y = df.loc[:, 'TARGET CLASS']
```

In [9]:
```
x.describe()
```

Out[9]:

	WTT	PTI	EQW	SBI	LQE	QWG	FDJ	PJF	HQE	NXJ
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000
mean	0.949682	1.114303	0.834127	0.682099	1.032336	0.943534	0.963422	1.071960	1.158251	1.362725
std	0.289635	0.257085	0.291554	0.229645	0.243413	0.256121	0.255118	0.288982	0.293738	0.204225
min	0.174412	0.441398	0.170924	0.045027	0.315307	0.262389	0.295228	0.299476	0.365157	0.639693
25%	0.742358	0.942071	0.615451	0.515010	0.870855	0.761064	0.784407	0.866306	0.934340	1.222623
50%	0.940475	1.118486	0.813264	0.676835	1.035824	0.941502	0.945333	1.065500	1.165556	1.375368
75%	1.163295	1.307904	1.028340	0.834317	1.198270	1.123060	1.134852	1.283156	1.383173	1.504832
max	1.721779	1.833757	1.722725	1.634884	1.650050	1.666902	1.713342	1.785420	1.885690	1.893950

In [10]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(x)
x_scaled = scaler.transform(x)
```

In [11]:
```
pd.DataFrame(x_scaled).describe()
```

Out[11]:

	0	1	2	3	4	5	6	7	8	9
count	1.000000e+03	1.000000e+03	1.000000e+03	1.000000e+03	1.000000e+03	1.000000e+03	1.000000e+03	1.000000e+03	1.000000e+03	1.000000e+03
mean	1.141309e-16	-3.198553e-16	-1.181277e-16	-1.766365e-16	-6.170064e-16	2.531308e-17	2.317035e-16	-4.826139e-16	3.438916e-16	4.525824e-16
std	1.000500e+00	1.000500e+00	1.000500e+00	1.000500e+00	1.000500e+00	1.000500e+00	1.000500e+00	1.000500e+00	1.000500e+00	1.000500e+00
min	-2.678050e+00	-2.618747e+00	-2.275858e+00	-2.775551e+00	-2.947206e+00	-2.660802e+00	-2.620466e+00	-2.674465e+00	-2.701361e+00	-3.542140e+00
25%	-7.161683e-01	-6.702761e-01	-7.504105e-01	-7.279635e-01	-6.637361e-01	-7.127975e-01	-7.020467e-01	-7.120098e-01	-7.626629e-01	-6.863610e-01
50%	-3.180217e-02	1.628137e-02	-7.159299e-02	-2.293699e-02	1.433731e-02	-7.940354e-03	-7.093937e-02	-2.236584e-02	2.488297e-02	6.194010e-02
75%	7.378939e-01	7.534412e-01	6.664646e-01	6.631695e-01	6.820374e-01	7.012930e-01	6.723000e-01	7.311915e-01	7.661087e-01	6.961851e-01
max	2.667092e+00	2.799904e+00	3.049325e+00	4.151021e+00	2.538987e+00	2.825739e+00	2.940974e+00	2.470109e+00	2.477734e+00	2.602476e+00

5. Split data

In [12]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x_scaled,y, test_size=0.30)
```

6. Train model

In [13]:
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(x_train,y_train)
```

Out[13]:
```
KNeighborsClassifier(n_neighbors=1)
```

7. Predict

In [14]:
```
y_pred = knn.predict(x_test)
```

In [15]:
```
y_pred
```

Out[15]:
```
array([0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1,
       1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0,
       1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0,
       1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0,
       1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0,
       1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1,
       0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1,
       0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1,
       1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0,
       1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1,
       1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0,
       0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1,
       0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1])
```

In [16]:
```
y_test
```

Out[16]:
```
587    0
663    1
820    0
268    0
889    0
      ..
401    0
178    1
752    0
407    0
613    0
Name: TARGET CLASS, Length: 300, dtype: int64
```

In [17]:
```
y_test == y_pred
```

Out[17]:
```
587     True
663     True
820     True
268     True
889     True
       ...
401     True
178     True
752     True
407     True
613    False
Name: TARGET CLASS, Length: 300, dtype: bool
```

In [18]:
```
# calculate error
import numpy as np
np.mean(y_test != y_pred)
```

Out[18]:
```
0.08666666666666667
```

In [19]:
```
# evaluate
from sklearn.metrics import classification_report,confusion_matrix
print( confusion_matrix( y_test , y_pred ) )
print( classification_report( y_test , y_pred ) )
```

```
[[143  15]
 [ 11 131]]
              precision    recall  f1-score   support

           0       0.93      0.91      0.92       158
           1       0.90      0.92      0.91       142

    accuracy                           0.91       300
   macro avg       0.91      0.91      0.91       300
weighted avg       0.91      0.91      0.91       300
```

8. Change the number of N

In [20]:
```
# Train model again (use N = 23)
knn = KNeighborsClassifier(n_neighbors=23)
knn.fit(x_train,y_train)
```

Out[20]:
```
KNeighborsClassifier(n_neighbors=23)
```

In [21]:
```
# Predict again
y_pred_new = knn.predict(x_test)
```

In [22]:
```
# Write performance of both N=1 and N=23 to compare
print('WITH K=1')
print(confusion_matrix(y_test,y_pred))
print(classification_report(y_test,y_pred))
print(np.mean(y_test != y_pred), '\n')
print('**************************************************')
print('WITH K=23')
print(confusion_matrix(y_test,y_pred_new),)
print(classification_report(y_test,y_pred_new))
print(np.mean(y_test != y_pred_new))
```

```
WITH K=1
[[143  15]
 [ 11 131]]
              precision    recall  f1-score   support

           0       0.93      0.91      0.92       158
           1       0.90      0.92      0.91       142

    accuracy                           0.91       300
   macro avg       0.91      0.91      0.91       300
weighted avg       0.91      0.91      0.91       300

0.08666666666666667

**************************************************
WITH K=23
[[144  14]
 [  5 137]]
              precision    recall  f1-score   support

           0       0.97      0.91      0.94       158
           1       0.91      0.96      0.94       142

    accuracy                           0.94       300
   macro avg       0.94      0.94      0.94       300
weighted avg       0.94      0.94      0.94       300

0.06333333333333334
```

8. Iteratively run the model for several N

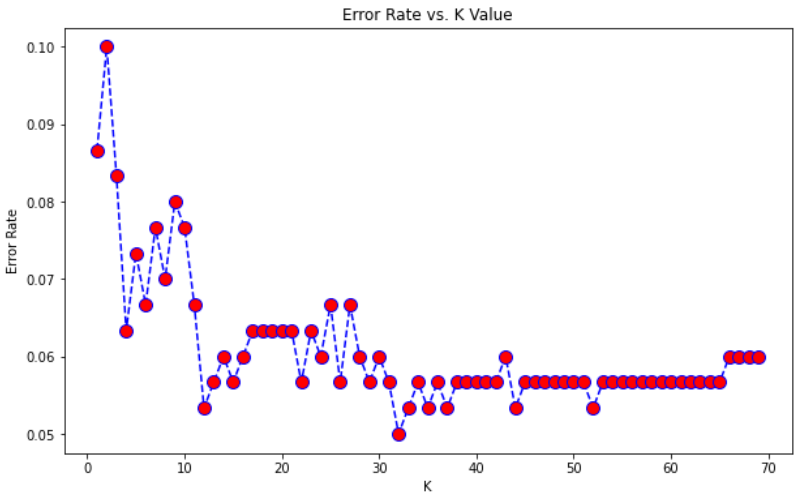
In [23]:
```
error_rate = []
for i in range(1,70):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(x_train,y_train)
    pred_i = knn.predict(x_test)
    error_rate.append(np.mean(pred_i != y_test))
```

In [24]:
```
error_rate
```

Out[24]:
```
[0.08666666666666667,
 0.1,
 0.08333333333333333,
 0.06333333333333334,
 0.07333333333333333,
 0.06666666666666667,
 0.07666666666666666,
 0.07,
 0.08,
 0.07666666666666666,
 0.06666666666666667,
 0.05333333333333334,
 0.056666666666666664,
 0.06,
 0.056666666666666664,
 0.06,
 0.06333333333333334,
 0.06333333333333334,
 0.06333333333333334,
 0.06333333333333334,
 0.06333333333333334,
 0.056666666666666664,
 0.06333333333333334,
 0.06,
 0.06666666666666667,
 0.056666666666666664,
 0.06666666666666667,
 0.06,
 0.056666666666666664,
 0.06,
 0.056666666666666664,
 0.05,
 0.05333333333333334,
 0.056666666666666664,
 0.05333333333333334,
 0.056666666666666664,
 0.05333333333333334,
 0.056666666666666664,
 0.056666666666666664,
 0.056666666666666664,
 0.056666666666666664,
 0.056666666666666664,
 0.06,
 0.05333333333333334,
 0.056666666666666664,
 0.056666666666666664,
 0.056666666666666664,
 0.056666666666666664,
 0.056666666666666664,
 0.056666666666666664,
 0.056666666666666664,
 0.05333333333333334,
 0.056666666666666664,
 0.056666666666666664,
 0.056666666666666664,
 0.056666666666666664,
 0.056666666666666664,
 0.056666666666666664,
 0.056666666666666664,
 0.056666666666666664,
 0.056666666666666664,
 0.056666666666666664,
 0.056666666666666664,
 0.056666666666666664,
 0.06,
 0.06,
 0.06,
 0.06]
```

In [25]:
```
import matplotlib.pyplot as plt
plt.figure(figsize=(10,6))
plt.plot(range(1,70),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[25]:
```
Text(0, 0.5, 'Error Rate')
```



In [ ]: