In [2]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

In [3]:

```
df = pd.read_csv('knn.csv', index_col=0)
```

In [4]:

```
df
```

Out[4]:

	WTT	PTI	EQW	SBI	LQE	QWG	FDJ	PJF	HQE
0	0.913917	1.162073	0.567946	0.755464	0.780862	0.352608	0.759697	0.643798	0.879422
1	0.635632	1.003722	0.535342	0.825645	0.924109	0.648450	0.675334	1.013546	0.621552
2	0.721360	1.201493	0.921990	0.855595	1.526629	0.720781	1.626351	1.154483	0.957877
3	1.234204	1.386726	0.653046	0.825624	1.142504	0.875128	1.409708	1.380003	1.522692
4	1.279491	0.949750	0.627280	0.668976	1.232537	0.703727	1.115596	0.646691	1.463812
...	...	...	...	...	...	...	...	...	...
995	1.010953	1.034006	0.853116	0.622460	1.036610	0.586240	0.746811	0.319752	1.117340
996	0.575529	0.955786	0.941835	0.792882	1.414277	1.269540	1.055928	0.713193	0.958684
997	1.135470	0.982462	0.781905	0.916738	0.901031	0.884738	0.386802	0.389584	0.919191
998	1.084894	0.861769	0.407158	0.665696	1.608612	0.943859	0.855806	1.061338	1.277456
999	0.837460	0.961184	0.417006	0.799784	0.934399	0.424762	0.778234	0.907962	1.257190

1000 rows × 11 columns

In [5]:

```
df.head()
```

Out[5]:

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

In [9]:

```
#Standardize variables
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
```

In [10]:

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

Out[10]:

```
StandardScaler()
```

In [11]:

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

In [13]:

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

In [14]:

```
df_feat.head()
```

Out[14]:

	WTT	PTI	EQW	SBI	LQE	QWG	FDJ	PJF	H
0	-0.123542	0.185907	-0.913431	0.319629	-1.033637	-2.308375	-0.798951	-1.482368	-0.949
1	-1.084836	-0.430348	-1.025313	0.625388	-0.444847	-1.152706	-1.129797	-0.202240	-1.828
2	-0.788702	0.339318	0.301511	0.755873	2.031693	-0.870156	2.599818	0.285707	-0.682
3	0.982841	1.060193	-0.621399	0.625299	0.452820	-0.267220	1.750208	1.066491	1.241
4	1.139275	-0.640392	-0.709819	-0.057175	0.822886	-0.936773	0.596782	-1.472352	1.040

In [15]:

```
df_feat.shape
```

Out[15]:

```
(1000, 10)
```

In [16]:

```
#Train test split
from sklearn.model_selection import train_test_split
```

In [17]:

```
X_train, X_test, y_train, y_test =train_test_split(scaled_features, df['TARGET CLASS'], tes
```

In [18]:

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

In [19]:

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

In [20]:

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

Out[20]:

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

In [22]:

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

```
#Prediction and Evaluation
```

In [25]:

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

In [27]:

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

```
[[132  15]
 [ 11 142]]
```

In [50]:

```
#Choosing K Value
error_rate=[]
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)))
```

In [33]:

```
plt.figure(figsize=(10,6))
```

Out[33]:

```
<Figure size 720x432 with 0 Axes>
```

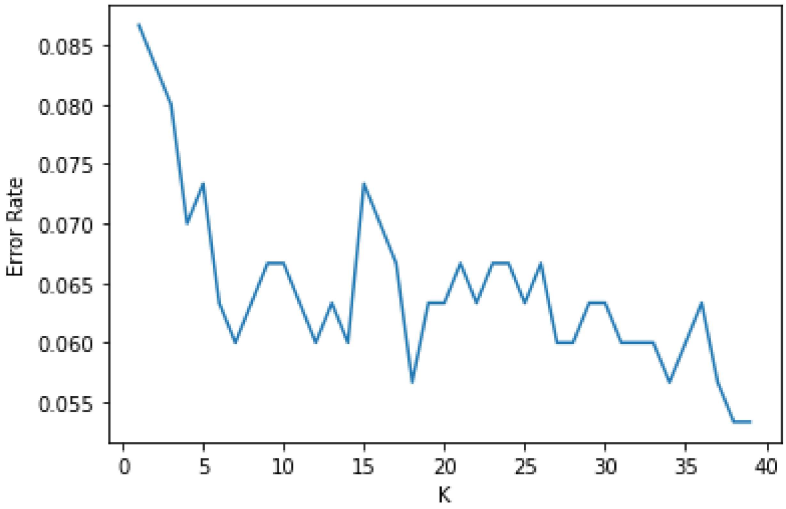
```
<Figure size 720x432 with 0 Axes>
```

In [53]:

```
plt.plot(range(1,40),error_rate)
plt.xlabel('K')
plt.ylabel('Error Rate')
```

Out[53]:

```
Text(0, 0.5, 'Error Rate')
```



In [55]:

```
#K 23 Looks better
knn=KNeighborsClassifier(n_neighbors=23)
knn.fit(X_train,y_train)
pred=knn.predict(X_test)
print('with K=13 Confusion matrix is =')
print(confusion_matrix(y_test, pred))
print(classification_report(y_test, pred))
```

```
with K=13 Confusion matrix is =
[[133  14]
 [  6 147]]
              precision    recall  f1-score   support

           0       0.96      0.90      0.93       147
           1       0.91      0.96      0.94       153

    accuracy                           0.93       300
   macro avg       0.93      0.93      0.93       300
weighted avg       0.93      0.93      0.93       300
```

In [ ]: