KNN : ' K NEAREST NEIBOUR ' :

PAGE 01

In [1]:
```
# KNN : ' K NEAREST NEIBOUR '
# 'K' is an VALUE / FEATURE / SIMILARITIES :
# KNN : is  a CLASSIFICATION MODEL :

# 1-0 : 1 of 0
# T-F : TRUE OR FALSE
# What exactly we are going to be work it out.
```

NOTE :

PAGE 05

In [2]:
```
# KNN : 'K NEAREST NEIBOUR' - is an a Classification Algorithm, that Operates on a very simple principle.
# 'k' we can Consider as an a 'New Data Point'(or)'Value'(or)'Features'(or)'Similarities of an a Existing'(or)'Trainec
# KNN : is an a Classification Model, Where 'K' is an a Features Identification Point:
# Here, 'K' Point -> Will identify the features, with the Nearest Data Points. Based on "EUCLIDEAN DISTNCE CALICULATI(

# "EUCLIDEAN DISTANCE FORMULA" :

# Euclidean Distance Formula :
# The Euclidean distance formula helps to find the distance of a line segment. Let us assume two points,
# such as (x1, y1) and (x2, y2) in the two-dimensional coordinate plane.

# Thus, the Euclidean distance formula is given by:

# d =√[(x2 – x1)2 + (y2 – y1)2]

# Where,

# “d” is the Euclidean distance

# (x1, y1) is the coordinate of the first point

# (x2, y2) is the coordinate of the second point.
```

In [ ]:
```
```

Now let's call the ' DATA SET ' and ' Few More LIBRARIES ' :

PAGE 06

In [16]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

In [17]:
```
df = pd.read_csv('DataS/38.Classified Data',index_col = 0)
```

In [18]:
```
df.head()
```

Out[18]:

	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 [19]:
```
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 [20]:
```
#
# A scaling transformation alters the size of an object.
# In the scaling process, we either compress or expand the dimension of the object.
# The scaling operation can be achieved by multiplying each vertex coordinate (x, y)
# of the polygon by scaling factor sx and sy to produce the transformed coordinates as (x', y')25
```

In [21]:
```
# Why are scaling techniques important?
# Importance of Scaling

# It helps in measuring. and analyzing attitudes of different individuals.
# The exact behavior of an individual is reflected by such attitude analysis.
# Number of attitude measuring scales has been developed by researchers
```

In [22]:
```
# What are the 4 types of scaling?
# Scales of Measurement- Nominal, Ordinal, Interval and Ratio
# The four types of scales are:
# Nominal Scale.
# Ordinal Scale.
# Interval Scale.
# Ratio Scale.
```

In [23]:
```
# What are the 4 pillars of scaling techniques?
# Scaling Techniques or Measurement:

# All measurement methods are based on four pillars, namely, order, definition, distance, and origin.
```

In [24]:
```
# What are the 2 types of scaling techniques?
# The various types of scaling techniques used in research can be classified into two categories:
#    (a) comparative scales, and (b) Non-comparative scales.
```

In [32]:
```
from sklearn.preprocessing import StandardScaler
```

In [33]:
```
scaler = StandardScaler()
```

In [42]:
```
scaler.fit(df.drop('TARGET CLASS', axis = 1))
```

Out[42]:
```
StandardScaler()
```

In [43]:
```
scaled_features = scaler.transform(df.drop('TARGET CLASS',axis = 1))
```

In [44]:
```
scaled_features
```

Out[44]:
```
array([[-0.12354188,  0.18590747, -0.91343069, ..., -1.48236813,
        -0.9497194 , -0.64331425],
       [-1.08483602, -0.43034845, -1.02531333, ..., -0.20224031,
        -1.82805088,  0.63675862],
       [-0.78870217,  0.33931821,  0.30151137, ...,  0.28570652,
        -0.68249379, -0.37784986],
       ...,
       [ 0.64177714, -0.51308341, -0.17920486, ..., -2.36249443,
        -0.81426092,  0.11159651],
       [ 0.46707241, -0.98278576, -1.46519359, ..., -0.03677699,
         0.40602453, -0.85567   ],
       [-0.38765353, -0.59589427, -1.4313981 , ..., -0.56778932,
         0.3369971 ,  0.01034996]])
```

In [45]:
```
# Now let's call Directly a Data Frame :
# page 09
```

In [46]:
```
df_feat = pd.DataFrame(scaled_features, columns = df.columns[ : -1])
```

In [47]:
```
df_feat.head()
```

Out[47]:

	WTT	PTI	EQW	SBI	LQE	QWG	FDJ	PJF	HQE	NXJ
0	-0.123542	0.185907	-0.913431	0.319629	-1.033637	-2.308375	-0.798951	-1.482368	-0.949719	-0.643314
1	-1.084836	-0.430348	-1.025313	0.625388	-0.444847	-1.152706	-1.129797	-0.202240	-1.828051	0.636759
2	-0.788702	0.339318	0.301511	0.755873	2.031693	-0.870156	2.599818	0.285707	-0.682494	-0.377850
3	0.982841	1.060193	-0.621399	0.625299	0.452820	-0.267220	1.750208	1.066491	1.241325	-1.026987
4	1.139275	-0.640392	-0.709819	-0.057175	0.822886	-0.936773	0.596782	-1.472352	1.040772	0.276510

In [48]:
```
from sklearn.model_selection import train_test_split
```

In [49]:
```
X = df_feat
y = df['TARGET CLASS']
```

In [50]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
```

In [53]:
```
# Now how many neighbours, we need to compare with,
# Means, 'K' Value is going to be compared with an a Nearest Values,
# with how many Values and Features, we want to Compare in Data.

from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors = 1)
```

Now let's see with the Multiple in the Next Step :

page 10

In [59]:
```
knn.fit(X_train,y_train)
```

Out[59]:
```
KNeighborsClassifier(n_neighbors=1)
```

In [60]:
```
pred = knn.predict(X_test)    # Here, prediction of 'X_test'
```

```
C:\Users\my pc\anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:228: FutureWarning: Unlike other red
uction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts alon
g. In SciPy 1.11.0, this behavior will change: the default value of `keepdims` will become False, the `axis` over wh
ich the statistic is taken will be eliminated, and the value None will no longer be accepted. Set `keepdims` to True
or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```

In [61]:
```
from sklearn.metrics import classification_report, confusion_matrix
```

In [62]:
```
print(confusion_matrix(y_test,pred))   # Here, print between confusion matrix
print(classification_report(y_test,pred)) # Here print of classification report
```

```
[[146   9]
 [ 11 164]]
              precision    recall  f1-score   support

           0       0.93      0.94      0.94       155
           1       0.95      0.94      0.94       175

    accuracy                           0.94       330
   macro avg       0.94      0.94      0.94       330
weighted avg       0.94      0.94      0.94       330
```

In [63]:
```
# But, Here i want to get a value compare between with How many neighbors, we want to compare here,,,
# Here in 59th sum, we have tried with only '1' neighbor.
# Now i want to compare with 'Multiple Neighbors' - At the time, we need to write Same particular Logic.
# Where 'error_rate = [] empty list i'm taking'
# for 'i' in range (1,40), Now i'm building a particular Logic - knn = KNeighbor......
```

In [64]:
```
error_rate = []
for i in range (1,40):
    knn = KNeighborsClassifier(n_neighbors = i)  # Here, we gave n_neighbors = 'i' for Multiple Neighbors
    knn.fit(X_train,y_train)
    pred_i = knn.predict(X_test)
    error_rate.append(np.mean(pred_i != y_test))   # Same data we called
```

```
C:\Users\my pc\anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:228: FutureWarning: Unlike other
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it act
s along. In SciPy 1.11.0, this behavior will change: the default value of `keepdims` will become False, the `axis
` over which the statistic is taken will be eliminated, and the value None will no longer be accepted. Set `keepd
ims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
C:\Users\my pc\anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:228: FutureWarning: Unlike other
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it act
s along. In SciPy 1.11.0, this behavior will change: the default value of `keepdims` will become False, the `axis
` over which the statistic is taken will be eliminated, and the value None will no longer be accepted. Set `keepd
ims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
C:\Users\my pc\anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:228: FutureWarning: Unlike other
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it act
s along. In SciPy 1.11.0, this behavior will change: the default value of `keepdims` will become False, the `axis
` over which the statistic is taken will be eliminated, and the value None will no longer be accepted. Set `keepd
ims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
C:\Users\my pc\anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:228: FutureWarning: Unlike other
```

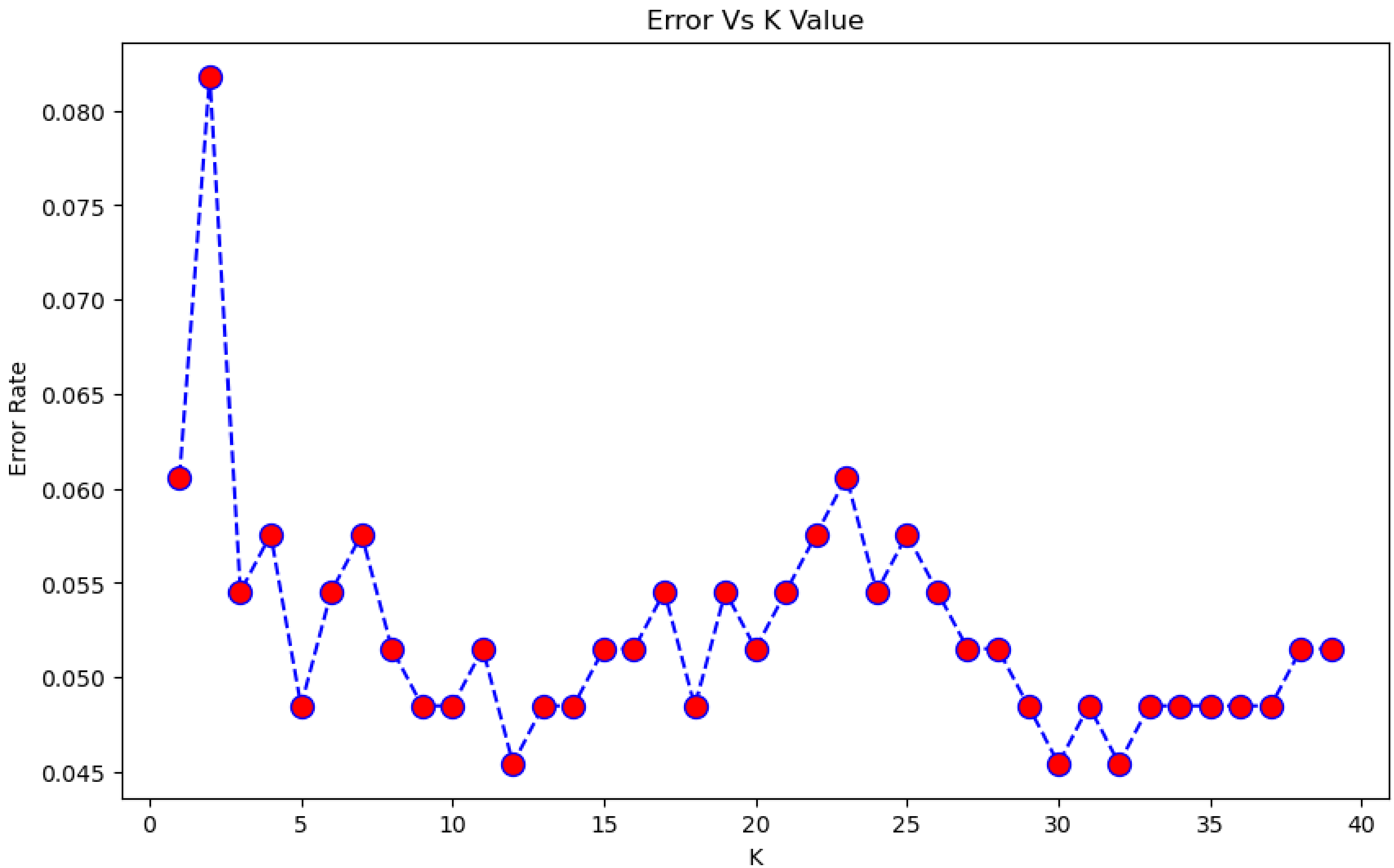
Now we need to see this one in the 'GRAPHICLE REPRESENTATION' :

PAGE 12 BOOK 5

In [66]:
```
# Here, We can see, Where the features very near here :
# This is the particular, How we are going to workout with an a 'KNN' :


plt.figure(figsize = (10,6))
plt.plot(range(1,40), error_rate, color = 'blue', linestyle = 'dashed', marker = 'o', markerfacecolor='red', markersi
plt.title("Error Vs K Value")
plt.xlabel(" K ")
plt.ylabel(" Error Rate")
```

Out[66]:
```
Text(0, 0.5, ' Error Rate')
```


Error Vs K Value
Error Rate
0.080
0.075
0.070
0.065
0.060
0.055
0.050
0.045
K
0
5
10
15
20
25
30
35
40

In [ ]: