Mymeningh Rainfall Classification using KNN Algorithm

In [1]:

```
"""
@author: Manoj Roy, ID: 20216039"""

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report, confusion_matrix
```

In [2]:

```
#read data
df=pd.read_csv('Mymensingh.csv')
df.head()
```

Out[2]:

	ID	Station	Year	Month	TEM	DPT	WIS	HUM	SLP	T_RAN	A_RAIN	RAN
0	1	Mymensingh	1960	1	16.9	11.3	2.0	73.39	1016.0	15	0.48	NRT
1	2	Mymensingh	1960	2	21.4	12.6	1.7	66.34	1013.0	0	0.00	NRT
2	3	Mymensingh	1960	3	24.1	14.9	2.3	64.13	1011.4	69	2.23	LTR
3	4	Mymensingh	1960	4	29.9	17.6	2.2	59.03	1007.1	27	0.90	NRT
4	5	Mymensingh	1960	5	29.6	23.2	2.4	73.45	1003.4	187	6.03	LTR

In [3]:

```
#predict class
Y=df['RAN']
```

In [4]:

```
#removing unnecessary columns
del df["ID"]
del df["Station"]
del df['Year']
del df['Month']
del df['RAN']
```

In [5]:

```
df.shape
```

Out[5]:

```
(672, 7)
```

In [6]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 672 entries, 0 to 671
Data columns (total 7 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   TEM     654 non-null    float64
 1   DPT     657 non-null    float64
```

```
 2   WIS     672 non-null    float64
 3   HUM     654 non-null    float64
 4   SLP     654 non-null    float64
 5   T_RAN   672 non-null    int64
 6   A_RAIN  672 non-null    float64
dtypes: float64(6), int64(1)
memory usage: 36.9 KB
```

In [7]:

```
#Full missing values
df['TEM']= df['TEM'].fillna(df['TEM'].mean())
df['DPT']= df['DPT'].fillna(df['DPT'].mean())
df['HUM']= df['HUM'].fillna(df['HUM'].mean())
df['SLP']= df['SLP'].fillna(df['SLP'].mean())
```

In [8]:

```
#feauture class
X=df
X.head()
```

Out[8]:

	TEM	DPT	WIS	HUM	SLP	T_RAN	A_RAIN
0	**16.9**	**11.3**	**2.0**	**73.39**	**1016.0**	**15**	**0.48**
1	**21.4**	**12.6**	**1.7**	**66.34**	**1013.0**	**0**	**0.00**
2	**24.1**	**14.9**	**2.3**	**64.13**	**1011.4**	**69**	**2.23**
3	**29.9**	**17.6**	**2.2**	**59.03**	**1007.1**	**27**	**0.90**
4	**29.6**	**23.2**	**2.4**	**73.45**	**1003.4**	**187**	**6.03**

In [9]:

```
df.isnull
```

Out[9]:

```
<bound method DataFrame.isnull of        TEM   DPT  WIS    HUM     SLP  T_RAN  A_RAIN
0     16.9  11.3  2.0  73.39  1016.0     15    0.48
1     21.4  12.6  1.7  66.34  1013.0      0    0.00
2     24.1  14.9  2.3  64.13  1011.4     69    2.23
3     29.9  17.6  2.2  59.03  1007.1     27    0.90
4     29.6  23.2  2.4  73.45  1003.4    187    6.03
..     ...   ...  ...    ...     ...    ...     ...
667   28.7  26.2  2.5  87.10  1003.3    349   11.26
668   28.8  25.2  2.0  85.63  1006.0    263    8.77
669   27.0  23.5  2.0  82.48  1011.3    180    5.81
670   23.1  18.7  1.7  81.73  1013.7     13    0.43
671   18.3  14.9  1.8  82.68  1015.9      5    0.16

[672 rows x 7 columns]>
```

In [10]:

```
#splitting the dataset
#assuming initial k value to be 15
X_train, X_test, Y_train, Y_test=train_test_split(X, Y, test_size=0.25, random_state=216
039)
KNN=KNeighborsClassifier(n_neighbors=3,metric='euclidean')
KNN.fit(X_train, Y_train)
P=KNN.predict(X_test)
P
```

Out[10]:

```
array(['MHR', 'MHR', 'NRT', 'MHR', 'LTR', 'NRT', 'NRT', 'LTR', 'LTR',
       'NRT', 'NRT', 'LTR', 'MHR', 'NRT', 'NRT', 'LTR', 'MHR', 'LTR',
       'NRT', 'NRT', 'NRT', 'NRT', 'MHR', 'NRT', 'LTR', 'LTR', 'MHR',
       'NRT', 'NRT', 'MHR', 'LTR', 'LTR', 'LTR', 'LTR', 'LTR', 'MHR',
```

```
       'LTR', 'NRT', 'LTR', 'NRT', 'MHR', 'NRT', 'NRT', 'LTR', 'NRT',
       'NRT', 'LTR', 'NRT', 'NRT', 'NRT', 'NRT', 'MHR', 'LTR', 'MHR',
       'NRT', 'NRT', 'MHR', 'MHR', 'NRT', 'LTR', 'LTR', 'NRT', 'MHR',
       'MHR', 'LTR', 'LTR', 'LTR', 'MHR', 'LTR', 'LTR', 'NRT', 'MHR',
       'NRT', 'MHR', 'MHR', 'NRT', 'LTR', 'NRT', 'LTR', 'LTR', 'NRT',
       'LTR', 'LTR', 'MHR', 'NRT', 'MHR', 'MHR', 'LTR', 'NRT', 'MHR',
       'LTR', 'LTR', 'MHR', 'LTR', 'NRT', 'LTR', 'NRT', 'NRT', 'NRT',
       'NRT', 'LTR', 'NRT', 'LTR', 'LTR', 'NRT', 'LTR', 'NRT', 'MHR',
       'NRT', 'MHR', 'NRT', 'NRT', 'MHR', 'NRT', 'MHR', 'NRT', 'MHR',
       'LTR', 'NRT', 'NRT', 'NRT', 'NRT', 'NRT', 'NRT', 'LTR', 'MHR',
       'MHR', 'NRT', 'NRT', 'NRT', 'NRT', 'LTR', 'MHR', 'LTR', 'LTR',
       'MHR', 'MHR', 'LTR', 'LTR', 'MHR', 'MHR', 'NRT', 'NRT', 'LTR',
       'NRT', 'NRT', 'LTR', 'NRT', 'MHR', 'MHR', 'NRT', 'LTR', 'MHR',
       'MHR', 'LTR', 'NRT', 'MHR', 'NRT', 'MHR', 'MHR', 'MHR', 'LTR',
       'MHR', 'LTR', 'NRT', 'LTR', 'LTR', 'NRT'], dtype=object)
```

In [11]:

```
accuracy_score(Y_test, P)
```

Out[11]:

```
1.0
```

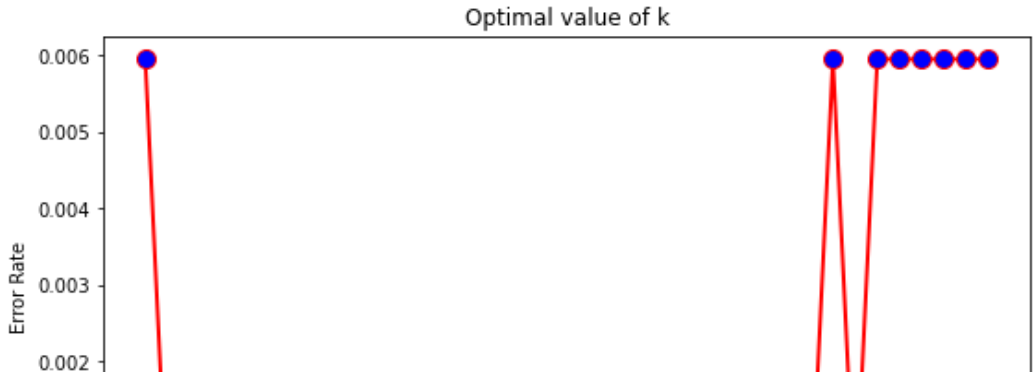
In [12]:

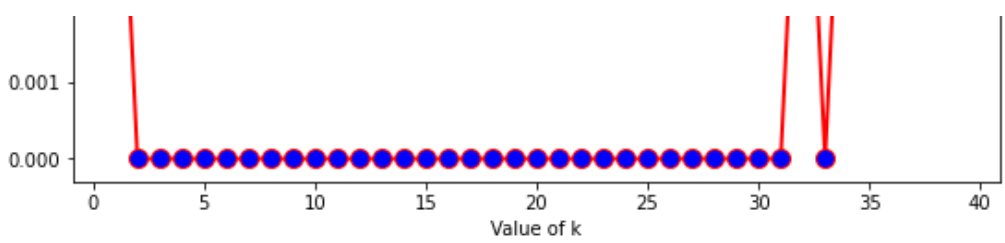
```
# Optimal Value of k and Accuracy Rate for Optimal k

error = []
accuracy = []
# Calculating error for K values between 1 and 40
for i in range(1, 40):
    knn = KNeighborsClassifier(n_neighbors=i,metric='euclidean')
    knn.fit(X_train, Y_train)
    pred_i = knn.predict(X_test)
    error.append(np.mean(pred_i != Y_test))
    accuracy.append(np.mean(pred_i == Y_test))
print(error)
print(accuracy)
```

```
[0.005952380952380952, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0
.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.005
952380952380952, 0.0, 0.005952380952380952, 0.005952380952380952, 0.005952380952380952, 0
.005952380952380952, 0.005952380952380952, 0.005952380952380952]
[0.9940476190476191, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0
, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.99404
76190476191, 1.0, 0.9940476190476191, 0.9940476190476191, 0.9940476190476191, 0.994047619
0476191, 0.9940476190476191, 0.9940476190476191]
```

In [13]:

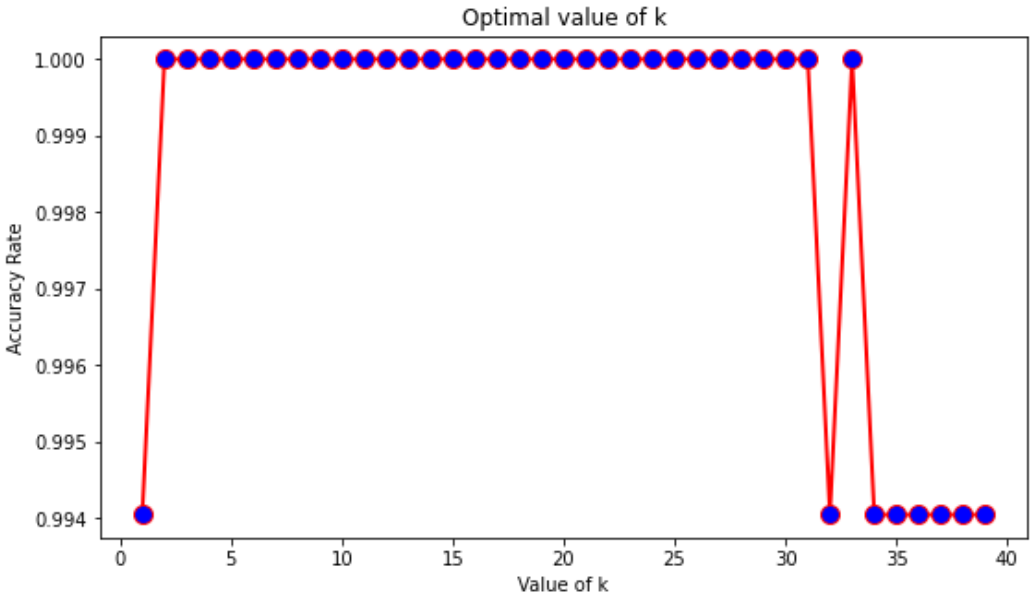
```
#plotting the optimal value of k vs Error Rate
plt.figure(figsize=(9, 5))
plt.plot(range(1, 40), error, color='red', linestyle='solid', marker='o',
         linewidth =2, markerfacecolor='blue', markersize=10)
plt.title('Optimal value of k')
plt.xlabel('Value of k')
plt.ylabel('Error Rate')
plt.savefig('ErrorRateMN.jpg',dpi=1200)
```





In [14]:

```
#plotting the optimal value of k vs Accuracy Rate
plt.figure(figsize=(9, 5))
plt.plot(range(1, 40), accuracy, color='red', linestyle='solid', marker='o',
         linewidth =2, markerfacecolor='blue', markersize=10)
plt.title('Optimal value of k')
plt.xlabel('Value of k')
plt.ylabel('Accuracy Rate')
plt.savefig('AccuracyRateMN.jpg',dpi=1200)
```



In [15]:

```
#training the model using optimal k value found from the graph
KNN1=KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='euclidean',
           metric_params=None, n_jobs=1, n_neighbors=5, p=2,
           weights='uniform')
KNN1.fit(X_train, Y_train)
P1=KNN1.predict(X_test)
P1
```

Out[15]:

```
array(['MHR', 'MHR', 'NRT', 'MHR', 'LTR', 'NRT', 'NRT', 'LTR', 'LTR',
       'NRT', 'NRT', 'LTR', 'MHR', 'NRT', 'NRT', 'LTR', 'MHR', 'LTR',
       'NRT', 'NRT', 'NRT', 'NRT', 'MHR', 'NRT', 'LTR', 'LTR', 'MHR',
       'NRT', 'NRT', 'MHR', 'LTR', 'LTR', 'LTR', 'LTR', 'LTR', 'MHR',
       'LTR', 'NRT', 'LTR', 'NRT', 'MHR', 'NRT', 'NRT', 'LTR', 'NRT',
       'NRT', 'LTR', 'NRT', 'NRT', 'NRT', 'NRT', 'MHR', 'LTR', 'MHR',
       'NRT', 'NRT', 'MHR', 'MHR', 'NRT', 'LTR', 'LTR', 'NRT', 'MHR',
       'MHR', 'LTR', 'LTR', 'LTR', 'MHR', 'LTR', 'LTR', 'NRT', 'MHR',
       'NRT', 'MHR', 'MHR', 'NRT', 'LTR', 'NRT', 'LTR', 'LTR', 'NRT',
       'LTR', 'LTR', 'MHR', 'NRT', 'MHR', 'MHR', 'LTR', 'NRT', 'MHR',
       'LTR', 'LTR', 'MHR', 'LTR', 'NRT', 'LTR', 'NRT', 'NRT', 'NRT',
       'NRT', 'LTR', 'NRT', 'LTR', 'LTR', 'NRT', 'LTR', 'NRT', 'MHR',
       'NRT', 'MHR', 'NRT', 'NRT', 'MHR', 'NRT', 'MHR', 'NRT', 'MHR',
       'LTR', 'NRT', 'NRT', 'NRT', 'NRT', 'NRT', 'NRT', 'LTR', 'MHR',
       'MHR', 'NRT', 'NRT', 'NRT', 'NRT', 'LTR', 'MHR', 'LTR', 'LTR',
       'MHR', 'MHR', 'LTR', 'LTR', 'MHR', 'MHR', 'NRT', 'NRT', 'LTR',
       'NRT', 'NRT', 'LTR', 'NRT', 'MHR', 'MHR', 'NRT', 'LTR', 'MHR',
       'MHR', 'LTR', 'NRT', 'MHR', 'NRT', 'MHR', 'MHR', 'MHR', 'LTR',
       'MHR', 'LTR', 'NRT', 'LTR', 'LTR', 'NRT'], dtype=object)
```

In [16]:

```
accuracy_score(Y_test, P1)
```

Out[16]:

```
1.0
```

In [17]:

```
print(confusion_matrix(Y_test, P1))
```

```
[[54  0  0]
 [ 0 46  0]
 [ 0  0 68]]
```

In [18]:

```
print(classification_report(Y_test, P1))
```

```
              precision    recall  f1-score   support

         LTR       1.00      1.00      1.00        54
         MHR       1.00      1.00      1.00        46
         NRT       1.00      1.00      1.00        68

    accuracy                           1.00       168
   macro avg       1.00      1.00      1.00       168
weighted avg       1.00      1.00      1.00       168
```

#Inference Here, I initially set the K value 3. Where I found the accuracy score of the classification 1. After iterating it 40 times, Here I found multiple optimal k values(3-31,33). I chose 5 among them, for which the accuracy score remained the same. From this assignment, I came to know about the optimal K value identification process & necessity.