Import necessary libraries

In [1]:

```
import numpy as np

import matplotlib.pyplot as plt
import pandas as pd

import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import mean_absolute_error,accuracy_score, mean_squared_error,confu
from sklearn.neighbors import KNeighborsClassifier
from sklearn import svm
from sklearn.tree import DecisionTreeClassifier

from sklearn.preprocessing import LabelEncoder

from sklearn.preprocessing import MinMaxScaler


import joblib
```

# Define the Objective for Building the KNN Model

Objective

To classify tumours as Benign or Malignant

Read the cancer data as a csv

In [3]:

```
cancer = pd.read_csv(r'C:\Users\user\OneDrive\Desktop\PC\Machine Learning\cancer_data.cs
```

In [4]:

```
cancer.head()
```

Out[4]:

| | id | diagnosis | radius_mean | texture_mean | perimeter_mean | area_mean | smoothness_mean | c |
|---|---|---|---|---|---|---|---|---|
| **0** | 842302 | M | 17.99 | 10.38 | 122.80 | 1001.0 | 0.11840 | |
| **1** | 842517 | M | 20.57 | 17.77 | 132.90 | 1326.0 | 0.08474 | |
| **2** | 84300903 | M | 19.69 | 21.25 | 130.00 | 1203.0 | 0.10960 | |
| **3** | 84348301 | M | 11.42 | 20.38 | 77.58 | 386.1 | 0.14250 | |
| **4** | 84358402 | M | 20.29 | 14.34 | 135.10 | 1297.0 | 0.10030 | |

5 rows × 33 columns

In [5]:

```
cancer.isnull().sum()
```

Out[5]:

```
id                          0
diagnosis                   0
radius_mean                 0
texture_mean                0
perimeter_mean              0
area_mean                   0
smoothness_mean             0
compactness_mean            0
concavity_mean              0
concave points_mean         0
symmetry_mean               0
fractal_dimension_mean      0
radius_se                   0
texture_se                  0
perimeter_se                0
area_se                     0
smoothness_se               0
compactness_se              0
concavity_se                0
concave points_se           0
symmetry_se                 0
fractal_dimension_se        0
radius_worst                0
texture_worst               0
perimeter_worst             0
area_worst                  0
smoothness_worst            0
compactness_worst           0
concavity_worst             0
concave points_worst        0
symmetry_worst              0
fractal_dimension_worst     0
Unnamed: 32               569
dtype: int64
```

In [6]:

```
cancer.columns
```

Out[6]:

```
Index(['id', 'diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst',
       'symmetry_worst', 'fractal_dimension_worst', 'Unnamed: 32'],
      dtype='object')
```

In [7]:

```
cancer.drop('Unnamed: 32', axis = 1, inplace = True)
```

In [8]:

```
cancer.columns
```

Out[8]:

```
Index(['id', 'diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst',
       'symmetry_worst', 'fractal_dimension_worst'],
      dtype='object')
```

In [9]:

```
cancer['diagnosis'].unique()
```

Out[9]:

```
array(['M', 'B'], dtype=object)
```

In [10]:

```
cancer.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 32 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   id                       569 non-null    int64  
 1   diagnosis                569 non-null    object 
 2   radius_mean              569 non-null    float64
 3   texture_mean             569 non-null    float64
 4   perimeter_mean           569 non-null    float64
 5   area_mean                569 non-null    float64
 6   smoothness_mean          569 non-null    float64
 7   compactness_mean         569 non-null    float64
 8   concavity_mean           569 non-null    float64
 9   concave points_mean      569 non-null    float64
 10  symmetry_mean            569 non-null    float64
 11  fractal_dimension_mean   569 non-null    float64
 12  radius_se                569 non-null    float64
 13  texture_se               569 non-null    float64
 14  perimeter_se             569 non-null    float64
 15  area_se                  569 non-null    float64
 16  smoothness_se            569 non-null    float64
 17  compactness_se           569 non-null    float64
 18  concavity_se             569 non-null    float64
 19  concave points_se        569 non-null    float64
 20  symmetry_se              569 non-null    float64
 21  fractal_dimension_se     569 non-null    float64
 22  radius_worst             569 non-null    float64
 23  texture_worst            569 non-null    float64
 24  perimeter_worst          569 non-null    float64
 25  area_worst               569 non-null    float64
 26  smoothness_worst         569 non-null    float64
 27  compactness_worst        569 non-null    float64
 28  concavity_worst          569 non-null    float64
 29  concave points_worst     569 non-null    float64
 30  symmetry_worst           569 non-null    float64
 31  fractal_dimension_worst  569 non-null    float64
dtypes: float64(30), int64(1), object(1)
memory usage: 142.4+ KB
```

# Encoding

In classification, categories are normally changed to numerical values.

This is because not all algorithms can deal with non-numeric categories

In [11]:
```
#Create an instance of the encoder
le = LabelEncoder()
```

In [12]:
```
#Convert the diagonis column to numerical categories
cancer['diagnosis'] = le.fit_transform(cancer['diagnosis'])
```

In [13]:
```
cancer.head()
```

Out[13]:

| | id | diagnosis | radius_mean | texture_mean | perimeter_mean | area_mean | smoothness_mean | c |
|---|---|---|---|---|---|---|---|---|
| **0** | 842302 | 1 | 17.99 | 10.38 | 122.80 | 1001.0 | 0.11840 | |
| **1** | 842517 | 1 | 20.57 | 17.77 | 132.90 | 1326.0 | 0.08474 | |
| **2** | 84300903 | 1 | 19.69 | 21.25 | 130.00 | 1203.0 | 0.10960 | |
| **3** | 84348301 | 1 | 11.42 | 20.38 | 77.58 | 386.1 | 0.14250 | |
| **4** | 84358402 | 1 | 20.29 | 14.34 | 135.10 | 1297.0 | 0.10030 | |

5 rows × 32 columns

In [14]:
```
cancer['diagnosis'].unique()
```

Out[14]:
```
array([1, 0])
```

Normally, the encoding is done in ascending order

After Encoding,

Benign --- 0

Malignant ---- 1

# Choose the target and predictors variables

Target --- diagnosis

Predictors --- all the other columns

In [15]:

```
y = cancer['diagnosis']
X = cancer.drop('diagnosis', axis = 1)
```

In [16]:

```
X.head()
```

Out[16]:

| | id | radius_mean | texture_mean | perimeter_mean | area_mean | smoothness_mean | compactness |
|---|---|---|---|---|---|---|---|
| **0** | 842302 | 17.99 | 10.38 | 122.80 | 1001.0 | 0.11840 | |
| **1** | 842517 | 20.57 | 17.77 | 132.90 | 1326.0 | 0.08474 | |
| **2** | 84300903 | 19.69 | 21.25 | 130.00 | 1203.0 | 0.10960 | |
| **3** | 84348301 | 11.42 | 20.38 | 77.58 | 386.1 | 0.14250 | |
| **4** | 84358402 | 20.29 | 14.34 | 135.10 | 1297.0 | 0.10030 | |

5 rows × 31 columns

In [17]:

```
y
```

Out[17]:

```
0      1
1      1
2      1
3      1
4      1
      ..
564    1
565    1
566    1
567    1
568    0
Name: diagnosis, Length: 569, dtype: int32
```

# Split the data to train and test

In [18]:

```
X_train,X_test,y_train,y_test = train_test_split(X, y, test_size = 0.2, random_state = 4
```

In [19]:

```
cancer.shape
```

Out[19]:

```
(569, 32)
```

In [20]:

```
X_train.shape
```

Out[20]:

```
(455, 31)
```

In [21]:

```
X_test.shape
```

Out[21]:
(114, 31)

In [22]:
```
y_test.shape
```

Out[22]:
(114,)

In [23]:
```
y_train.shape
```

Out[23]:
(455,)

# Building the Classification Model - KNN

In [24]:
```
#Build the model - KNN Algorithm
#Set the number of neighbors or k value
knn = KNeighborsClassifier(n_neighbors = 2)
knn.fit(X_train,y_train)
```

Out[24]:

▾ KNeighborsClassifier

KNeighborsClassifier(n_neighbors=2)

# Test the K-Nearest Neighbor Model

In [25]:
```
#Testing the model
#Using the variable y_pred, and testing data X_test, predict using the model to make com
y_pred = knn.predict(X_test)
```

Accuracy

In [26]:
```
acc = accuracy_score(y_test,y_pred)
```

In [27]:
```
print('The accuracy is ',acc*100)
```

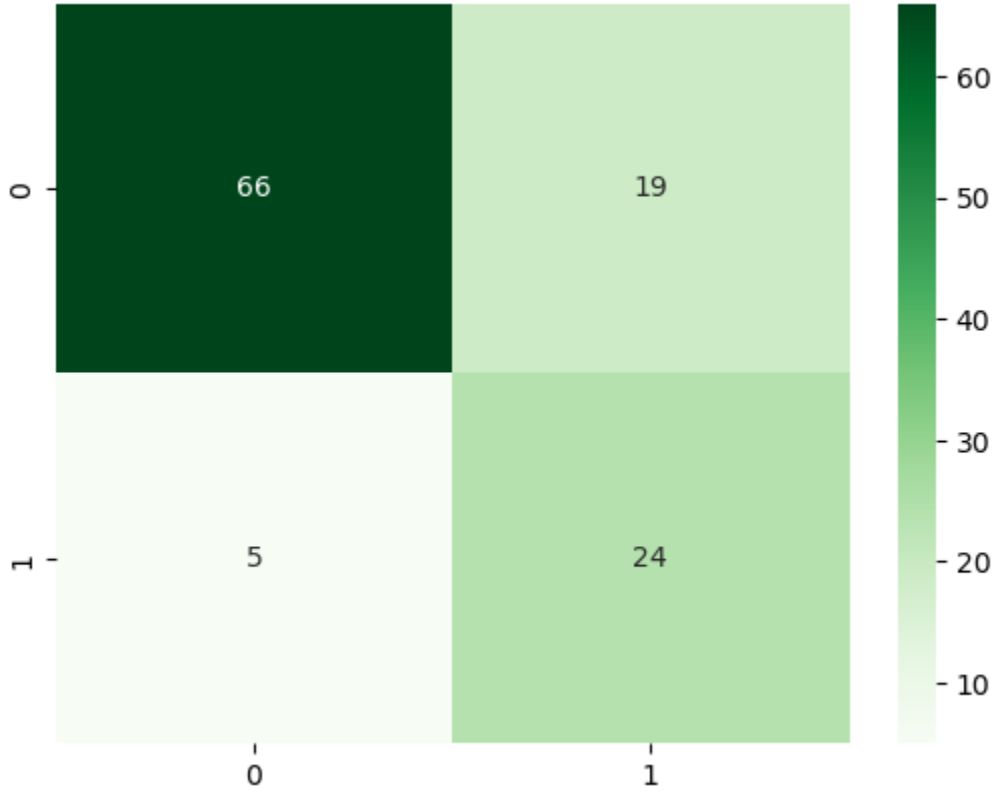
```
The accuracy is  78.94736842105263
```

Confusion Matrix

In [28]:
```
cnf_matrix = confusion_matrix(y_pred,y_test)
```

In [29]:
```
sns.heatmap(cnf_matrix, annot = True, cmap = 'Greens')
```

Out[29]:
<Axes: >




# Find the Optimal K value (ELBOW METHOD)

In [30]:

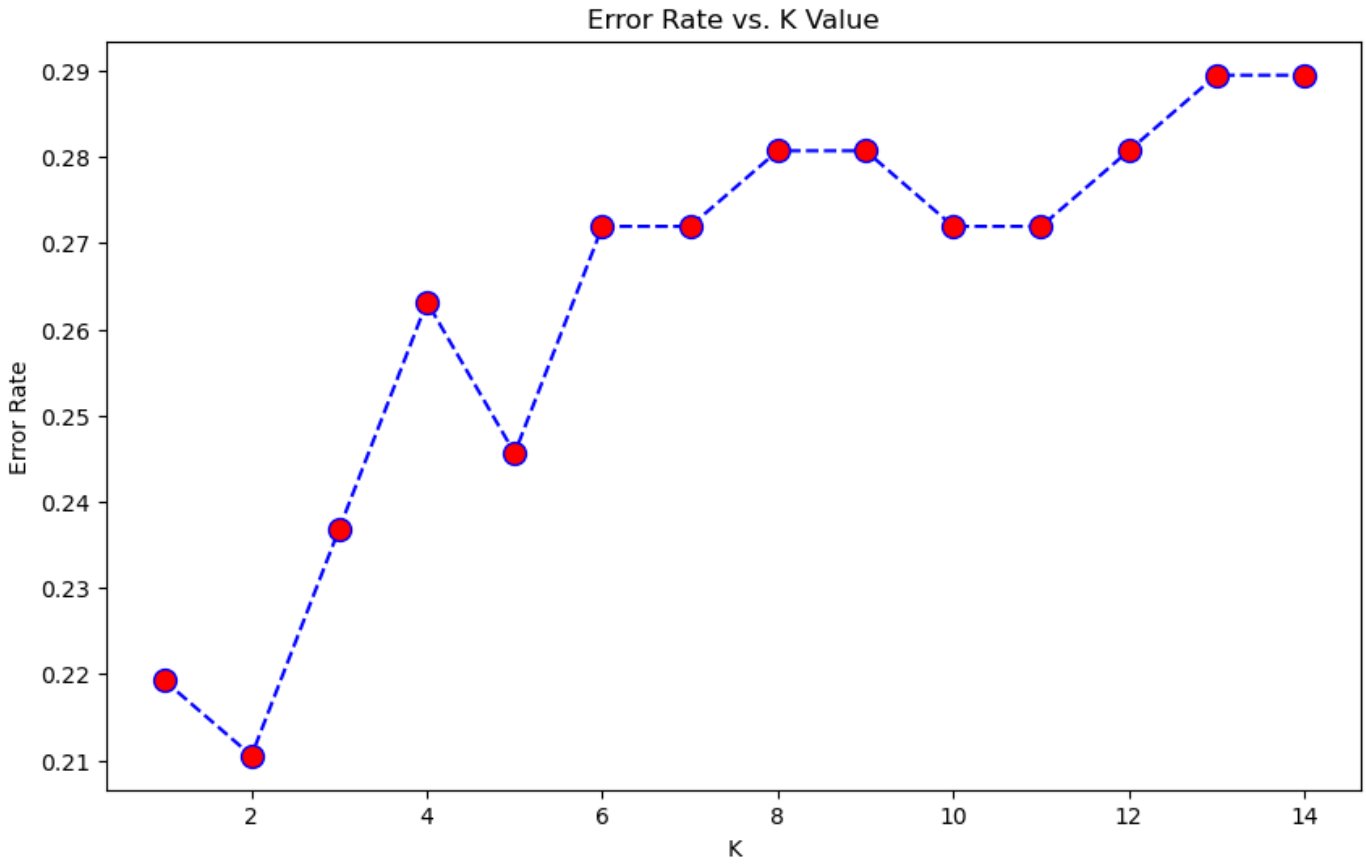
```
# Find the Optimal K value (ELBOW METHOD)
error_rate = []# Will take some time
for i in range(1,15):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train,y_train)
    pred_i = knn.predict(X_test)
    error_rate.append(np.mean(pred_i != y_test))
```

In [31]:

```
#Plotting
plt.figure(figsize=(10,6))
plt.plot(range(1,15),error_rate,color='blue', linestyle='dashed', marker='o',markerfacec
plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
```

Out[31]:

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




From our elbow method, the optimal K value is 11

In [ ]:

In [ ]:

In [ ]: