In [1]:

```python
# import the necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:

```python
data=pd.read_csv \
('C:/Users/kriti/OneDrive/Desktop/machine Learning/experiments\
/diabetes.csv')
data
```

Out[2]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
...	...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

768 rows × 9 columns

In [3]:

```python
print(len(data))
```

```
768
```

In [4]:

```python
data.columns
```

Out[4]:

```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

There are certain columns where 0 values are there like in SkinThickness,Insulin,glucose

Replace zeros with mean of those column

In [5]:

```python
(data == 0).sum()
```

Out[5]:

```
Pregnancies                 111
Glucose                       5
BloodPressure                35
SkinThickness               227
Insulin                     374
BMI                          11
DiabetesPedigreeFunction      0
Age                           0
Outcome                     500
dtype: int64
```

In [6]:

```
non_zero =['Glucose', 'BloodPressure','SkinThickness','BMI', 'Insulin']
#Iterating all columns wherever 0 is there & substituting with NaN
#NaN defined in NUMPY LIBRARY
# Then we are replacing NaN with mean of the column
for column in non_zero:
    data[column] = data[column].replace(0, np.NaN)
    mean = int(data[column].mean(skipna = True))
    data[column] = data[column].replace(np.NaN, mean)
```

In [7]:

```
#NaN is short for Not a number.
#It is used to represent entries that are undefined or
#missing values
#example
v= np.array([1, np.NaN, 3, 4])
np.sum(v)
```

Out[7]: nan

In [8]:

```
np.nansum(v)
```

Out[8]: 8.0

In [9]:

```
print(data['SkinThickness'])
```

```
0      35.0
1      29.0
2      29.0
3      23.0
4      35.0
       ...
763    48.0
764    27.0
765    23.0
766    29.0
767    31.0
Name: SkinThickness, Length: 768, dtype: float64
```

In [10]:

```
x = data.iloc[:, 0:8]
x
```

Out[10]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	6	148.0	72.0	35.0	155.0	33.6	0.627	50
1	1	85.0	66.0	29.0	155.0	26.6	0.351	31
2	8	183.0	64.0	29.0	155.0	23.3	0.672	32
3	1	89.0	66.0	23.0	94.0	28.1	0.167	21
4	0	137.0	40.0	35.0	168.0	43.1	2.288	33
...	...	...	...	...	...	...	...	...
763	10	101.0	76.0	48.0	180.0	32.9	0.171	63
764	2	122.0	70.0	27.0	155.0	36.8	0.340	27
765	5	121.0	72.0	23.0	112.0	26.2	0.245	30
766	1	126.0	60.0	29.0	155.0	30.1	0.349	47
767	1	93.0	70.0	31.0	155.0	30.4	0.315	23

768 rows × 8 columns

-sklearn Library: Scikit-learn is probably the most useful library for machine learning in Python. The sklearn library contains a lot of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction.

•Supervised learning algorithms •Cross-validation •Unsupervised learning algorithms •Various toy datasets: (e.g. IRIS dataset, Boston House prices dataset). •Feature extraction: Scikit-learn for extracting features from images

In [11]:
```
y = data.iloc[:, 8]
y
```

Out[11]:
```
0      1
1      0
2      1
3      0
4      1
      ..
763    0
764    0
765    0
766    1
767    0
Name: Outcome, Length: 768, dtype: int64
```

-model_selection: It is a method for setting a blueprint to analyze data and then using it to measure new data. Selecting a proper model allows you to generate accurate results when making a prediction.

-train_test_split is a function in Sklearn model selection for splitting data arrays into two subsets: for training data and for testing data. With this function, you don't need to divide the dataset manually.

In [27]:
```
#Split the data into train and test
from sklearn.model_selection import train_test_split
x = data.iloc[:, 0:8]
y = data.iloc[:, 8]
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,\
                                               random_state=89)
```

In [28]:
```
print(x)
```

```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0              6    148.0           72.0           35.0    155.0  33.6
1              1     85.0           66.0           29.0    155.0  26.6
2              8    183.0           64.0           29.0    155.0  23.3
3              1     89.0           66.0           23.0     94.0  28.1
4              0    137.0           40.0           35.0    168.0  43.1
..           ...      ...            ...            ...      ...   ...
763           10    101.0           76.0           48.0    180.0  32.9
764            2    122.0           70.0           27.0    155.0  36.8
765            5    121.0           72.0           23.0    112.0  26.2
766            1    126.0           60.0           29.0    155.0  30.1
767            1     93.0           70.0           31.0    155.0  30.4

     DiabetesPedigreeFunction  Age
0                       0.627   50
1                       0.351   31
2                       0.672   32
3                       0.167   21
4                       2.288   33
..                        ...  ...
763                     0.171   63
764                     0.340   27
765                     0.245   30
766                     0.349   47
767                     0.315   23

[768 rows x 8 columns]
```

In [14]:

```
print(y)
```

```
0      1
1      0
2      1
3      0
4      1
      ..
763    0
764    0
765    0
766    1
767    0
Name: Outcome, Length: 768, dtype: int64
```

```
Data standardization is the process of rescaling the attributes so that they have mean as 0 and variance as 1.
The ultimate goal to perform standardization is to bring down all the features to a common scale without distorting the differences in the range of the values.
In sklearn.preprocessing.StandardScaler(), centering and scaling happens independently on each feature.
The idea behind StandardScaler is that it will transform your data such that its distribution will have a mean value 0 and standard deviation of 1.
python sklearn library offers us with StandardScaler() function to standardize the data values into a standard format with mean 0 and standard deviation 1
```

```
feature scaling is done by making mean 0 and standard deviation 1 of every input column
fit_tranform function is used on x_train to learn paramters(mean and standard deviation)
so that standardization can happpen use the tranform function on x_text to use the paramters(mean and standard deviation)
of train set on set test
```

In [30]:

```

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

x_train = sc.fit_transform(x_train)

x_test = sc.transform(x_test)
```

```
object = StandardScaler()
object.fit_transform(data)
According to the above syntax, we initially create an object of the StandardScaler() function. Further, we use fit_transform()
fit_transform() is used on the training data so that we can scale the training data and also learn the scaling parameters of that data (mean and standard deviation) . Here, the model built by us will learn the mean and standard deviation of the features of the training set. These learned parameters are then used to scale our test data also.
Using the transform method we can use the same mean and standard deviation as it is calculated from our training data to transform our test data. Thus, the parameters learned by our model using the training data will help us to transform our test data also.
```

In [31]:
```
x_train
```

Out[31]:
```
array([[-0.83444704,  2.4607925 ,  0.2729567 , ...,  1.49054148,
         2.83104312, -0.95428911],
       [ 0.6549439 ,  0.46408671, -0.98652789, ..., -1.1976073 ,
        -0.97626322,  1.81271231],
       [-0.83444704,  0.23865218, -0.23083713, ..., -0.95323013,
         0.00695909, -0.95428911],
       ...,
       [-0.83444704, -1.46820922, -1.40635608, ...,  0.11052928,
         2.39303443, -0.78659205],
       [-0.23869066,  1.52684947, -0.73463097, ...,  0.2830308 ,
        -0.34834866, -0.28350089],
       [ 0.35706571,  1.04377549,  0.94468182, ...,  0.88678614,
         0.45721978,  0.05189323]])
```

Generating Model Let's build KNN classifier model.

First, import the KNeighborsClassifier module and create KNN classifier object by passing argument number of neighbors in KNeighborsClassifier() function.

In [32]:
```
# define the model
from sklearn.neighbors import KNeighborsClassifier
#classifier =  KNeighborsClassifier(n_neighbors = 11, p =2, metric = 'euclidean')
classifier =  KNeighborsClassifier(n_neighbors = 11, \
                                   metric = 'euclidean')
# p= power paramter 1 for manhatten and 2 for euclidian
```

Why k is odd: Let's think for a while: The k, in the KNN algorithm, represent the number of closest neighbors that you are comparing, right? So, no matter if you have 2 or n classes, if you choose an even k, there is a risk of a tie in the decision of which class you should set a new instance. This is why the k is usually odd - no ties.

Fit the model on the train set using fit() and perform prediction on the test set using predict().

In [18]:
```
classifier.fit(x_train, y_train)
```

Out[18]:
```
KNeighborsClassifier(metric='euclidean', n_neighbors=11)
```

In [19]:
```
# prediction on test set
y_pred = classifier.predict(x_test)
y_pred
```

Out[19]:
```
array([0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0,
       1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0,
       1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0,
       0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0,
       0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0],
      dtype=int64)
```

In [20]:
```
# Generate the confusion matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm);

```

```
[[82 21]
 [23 28]]
```

In [21]:
```
# Find accuracy
from sklearn.metrics import accuracy_score
print(accuracy_score(y_test, y_pred))
```

```
0.7142857142857143
```

In [22]:
```
# Find F1 score
from sklearn.metrics import f1_score
print(f1_score(y_test, y_pred))
```

```
0.5599999999999999
```

Elbow method: helps to find optimal value of k Elbow method helps data scientists to select the optimal number of neighbors for KNN. As K increases, the error usually goes down, then stabilizes, and then raises again. Pick the optimum K at the beginning of the stable zone. This is also called Elbow Method.

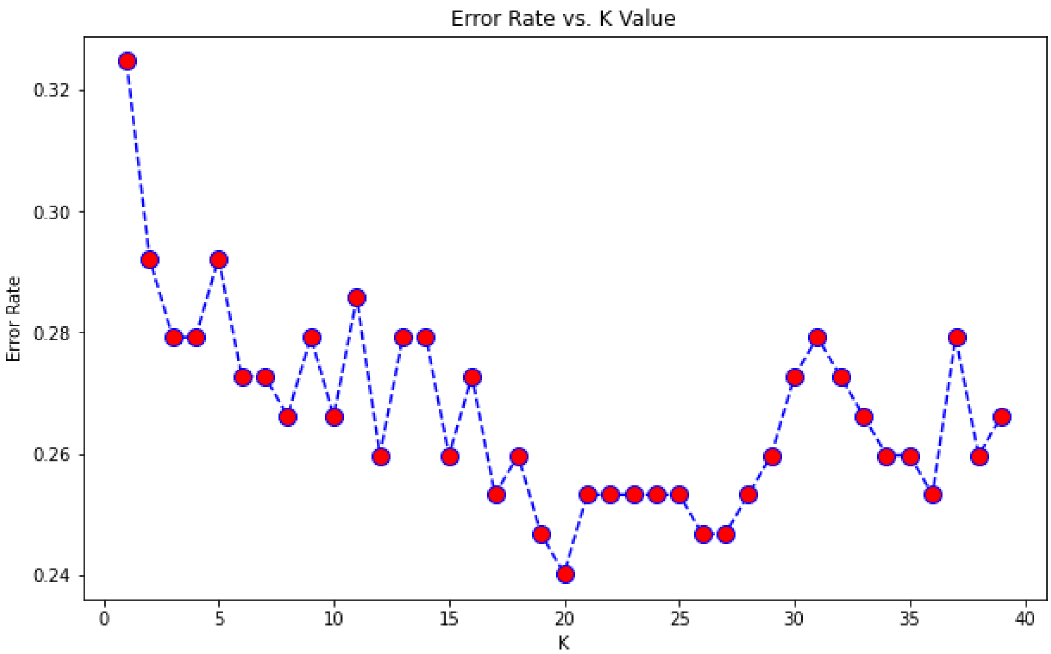
In [23]:
```
from sklearn.neighbors import KNeighborsClassifier
error_rate = []
for i in range(1,40):

 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 [24]:
```
error_rate
```

Out[24]:
```
[0.3246753246753247,
 0.2922077922077922,
 0.2792207792207792,
 0.2792207792207792,
 0.2922077922077922,
 0.2727272727272727,
 0.2727272727272727,
 0.2662337662337662,
 0.2792207792207792,
 0.2662337662337662,
 0.2857142857142857,
 0.2597402597402597,
 0.2792207792207792,
 0.2792207792207792,
 0.2597402597402597,
 0.2727272727272727,
 0.2532467532467532,
 0.2597402597402597,
 0.24675324675324675,
 0.24025974025974026,
 0.2532467532467532,
 0.2532467532467532,
 0.2532467532467532,
 0.2532467532467532,
 0.2532467532467532,
 0.24675324675324675,
 0.24675324675324675,
 0.2532467532467532,
 0.2597402597402597,
 0.2727272727272727,
 0.2792207792207792,
 0.2727272727272727,
 0.2662337662337662,
 0.2597402597402597,
 0.2597402597402597,
 0.2532467532467532,
 0.2792207792207792,
 0.2597402597402597,
 0.2662337662337662]
```

In [25]:
```
plt.figure(figsize=(10,6))
plt.plot(range(1,40),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')
```




```
Use the confusion_matrix method from sklearn.metrics to compute the confusion matrix.
 classification_report: Gives a text report showing the main classification metrics.
```

In [26]:
```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
# NOW WITH K=20
knn = KNeighborsClassifier(n_neighbors=20)
knn.fit(x_train,y_train)
pred = knn.predict(x_test)

print(confusion_matrix(y_test,pred))

print(classification_report(y_test,pred))
```

```
[[90 13]
 [24 27]]
              precision    recall  f1-score   support

           0       0.79      0.87      0.83       103
           1       0.68      0.53      0.59        51

    accuracy                           0.76       154
   macro avg       0.73      0.70      0.71       154
weighted avg       0.75      0.76      0.75       154
```

In [ ]:
```

```