## Diabetes Dataset

### Objective

The objective of the dataset is to diagnostically predict whether or not a patient has diabetes

#### Import pandas, numpy, seaborn, matplotlib.pyplot packages

In [2]:

**import** numpy **as** np

**import** pandas **as** pd

**import** matplotlib.pyplot **as** plt

**%**matplotlib inline

**import** seaborn **as** sns

**from** warnings **import** filterwarnings

filterwarnings('ignore')

In [3]:

df **=** pd.read\_csv('Datasets/diabetes.csv')

df.head()

Out[3]:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **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 |

**It is a Classification Problem - the Dependent variable is Outcome**

* **Shape of Dataset**

In [4]:

df.shape

Out[4]:

(768, 9)

* **The dataset has total 768 rows & 9 Attributes**
* **Checking Information of Dataset**

In [5]:

df.info()

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 768 entries, 0 to 767  
Data columns (total 9 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 Pregnancies 768 non-null int64   
 1 Glucose 768 non-null int64   
 2 BloodPressure 768 non-null int64   
 3 SkinThickness 768 non-null int64   
 4 Insulin 768 non-null int64   
 5 BMI 768 non-null float64  
 6 DiabetesPedigreeFunction 768 non-null float64  
 7 Age 768 non-null int64   
 8 Outcome 768 non-null int64   
dtypes: float64(2), int64(7)  
memory usage: 54.1 KB

* Dataset has 2 Float columns, 7 integer columns

## Data preprocessing & EDA

#### checking null values

In [6]:

df.isnull().sum()

Out[6]:

Pregnancies 0  
Glucose 0  
BloodPressure 0  
SkinThickness 0  
Insulin 0  
BMI 0  
DiabetesPedigreeFunction 0  
Age 0  
Outcome 0  
dtype: int64

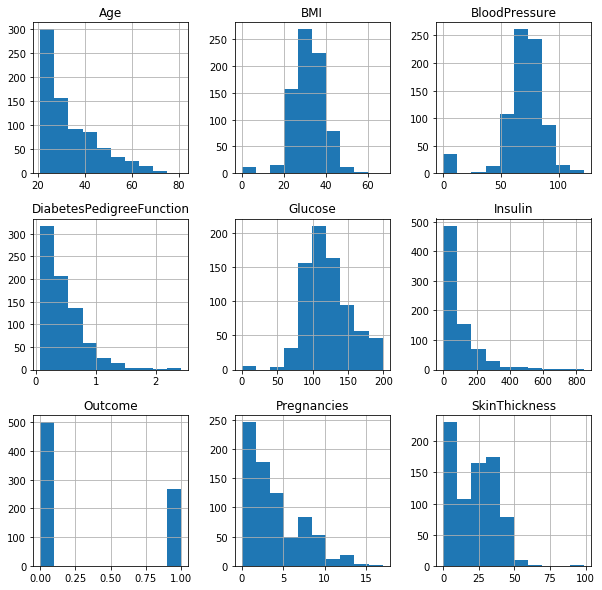
* There is no null values

**Plotting Histogram**

In [7]:

df.hist(figsize**=**(10,10))

plt.show()



### Inference from Histogram:

* Outcome Categorical Variables which is in Encoded format

### Checking for outliers

In [8]:

df.columns

Out[8]:

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

In [10]:

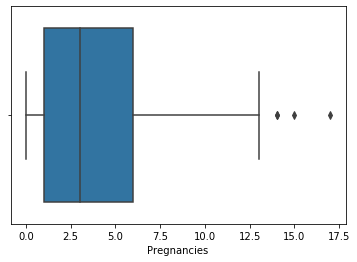
col **=** ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',

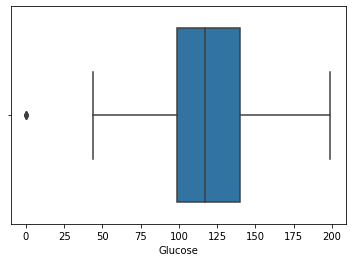
'BMI', 'DiabetesPedigreeFunction', 'Age']

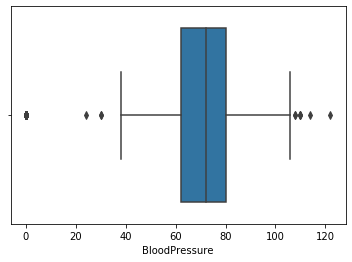
**for** i **in** col:

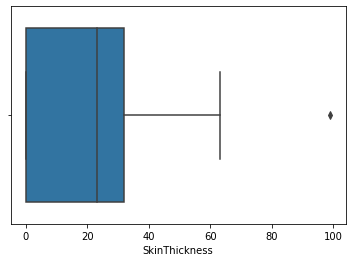
sns.boxplot(df[i])

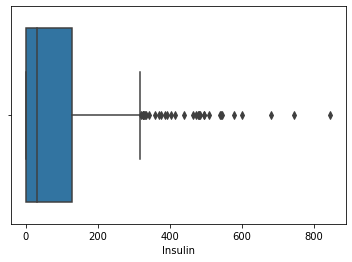
plt.show()

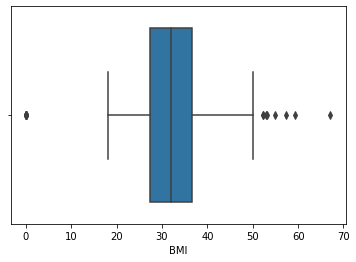


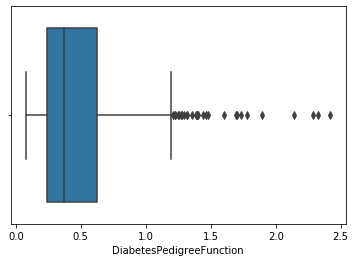


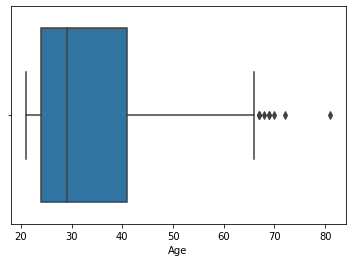












#### There are moderate outliers

## Correlation matrix

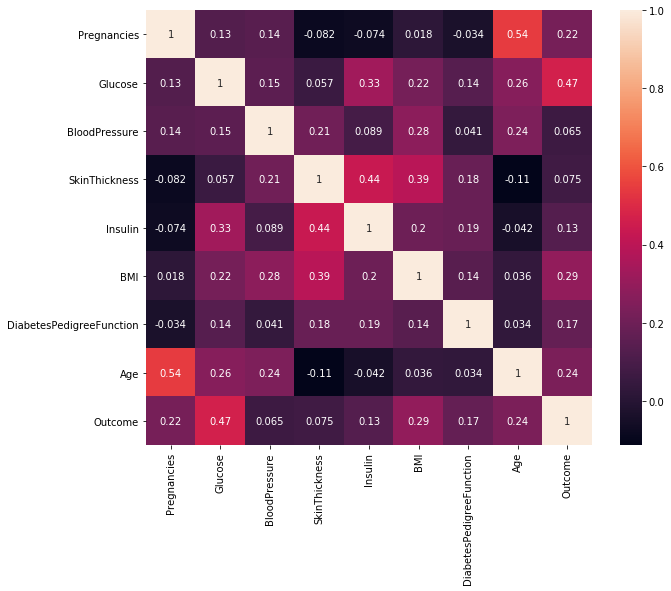
In [9]:

plt.figure(figsize**=**(10,8))

sns.heatmap(df.corr(), annot **=** **True**)

Out[9]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x1bc04253148>



### Analysis

In [46]:

df['Outcome'].value\_counts()

Out[46]:

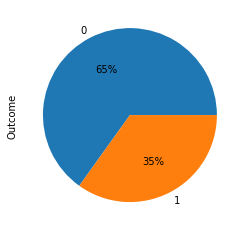
0 500  
1 268  
Name: Outcome, dtype: int64

In [37]:

df['Outcome'].value\_counts().plot(kind**=**'pie', autopct **=** "%1.0f%%")

Out[37]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x22c6d684448>



### Inference:

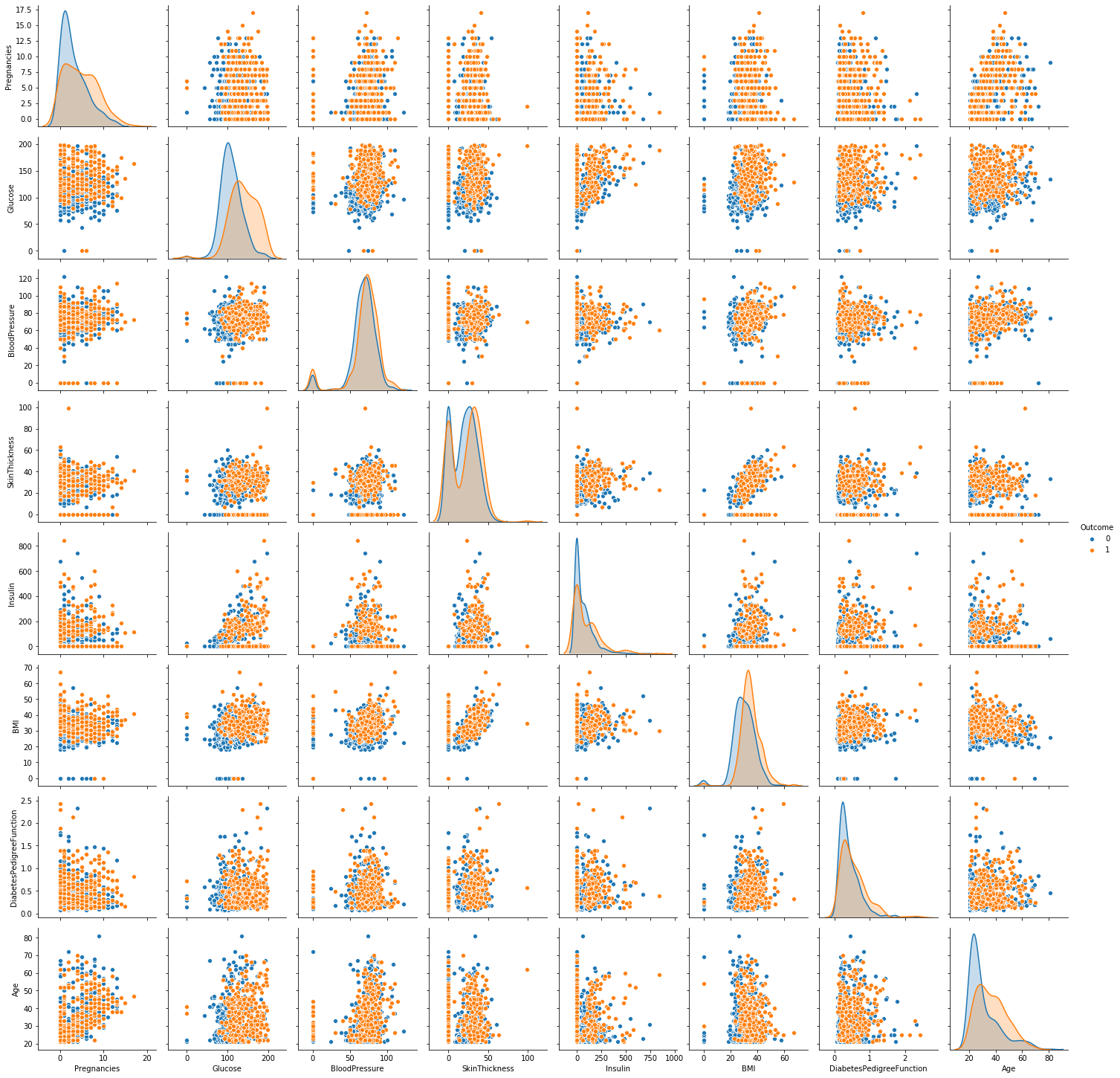
* 35% of the persons in dataset have diabetes

In [39]:

sns.pairplot(df, hue**=**'Outcome')

Out[39]:

<seaborn.axisgrid.PairGrid at 0x22c6f76ef88>



In [42]:

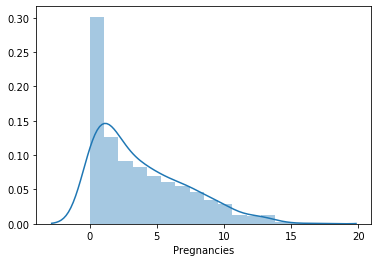
colm **=** ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',

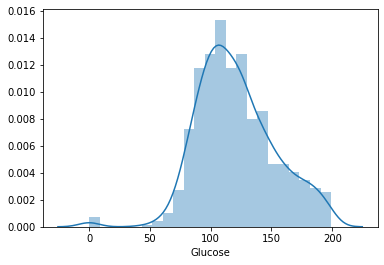
'BMI', 'DiabetesPedigreeFunction', 'Age']

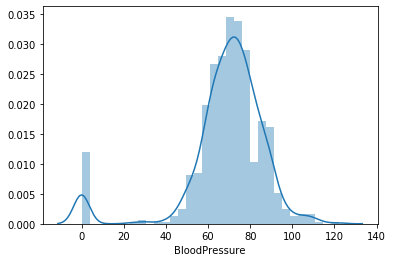
**for** col **in** colm:

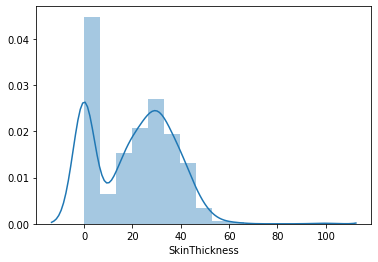
sns.distplot(df[col])

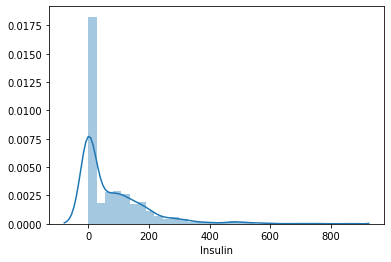
plt.show()

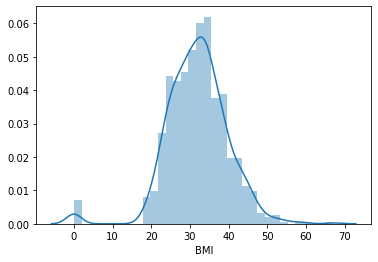


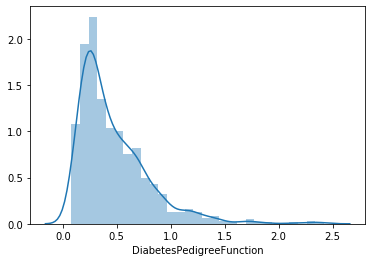


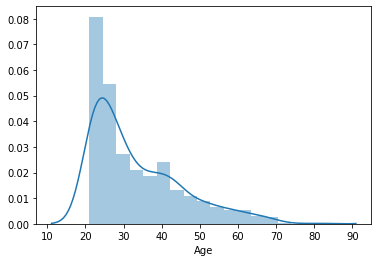












### Inference from Distplot

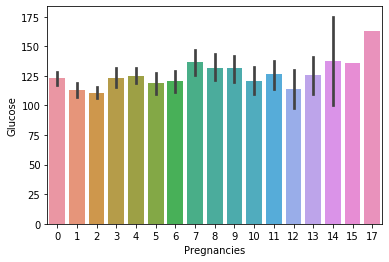
* Pregnencies, Glucose, Insulin , DiabetesPedigreeFunction, Age is Unimodal
* BloodPresure, SkinThickness, BMI from Home is Bimodal

In [43]:

sns.barplot(df['Pregnancies'], df['Glucose'])

Out[43]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x22c74ebcd48>

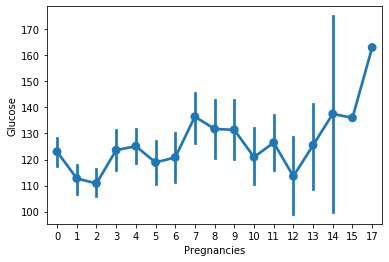


In [45]:

sns.pointplot(df['Pregnancies'], df['Glucose'])

Out[45]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x22c75149dc8>



In [47]:

pd.crosstab(df.Outcome , df.Pregnancies)

Out[47]:

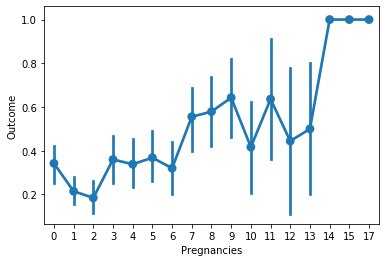
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Pregnancies** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** | **17** |
| **Outcome** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **0** | 73 | 106 | 84 | 48 | 45 | 36 | 34 | 20 | 16 | 10 | 14 | 4 | 5 | 5 | 0 | 0 | 0 |
| **1** | 38 | 29 | 19 | 27 | 23 | 21 | 16 | 25 | 22 | 18 | 10 | 7 | 4 | 5 | 2 | 1 | 1 |

In [55]:

sns.pointplot(df['Pregnancies'], df['Outcome'])

Out[55]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x22c790a8d88>



### Spliting data

In [15]:

x **=** df.drop(['Outcome'], axis**=**1)

x.head()

Out[15]:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Pregnancies** | **Glucose** | **BloodPressure** | **SkinThickness** | **Insulin** | **BMI** | **DiabetesPedigreeFunction** | **Age** |
| **0** | 6 | 148 | 72 | 35 | 0 | 33.6 | 0.627 | 50 |
| **1** | 1 | 85 | 66 | 29 | 0 | 26.6 | 0.351 | 31 |
| **2** | 8 | 183 | 64 | 0 | 0 | 23.3 | 0.672 | 32 |
| **3** | 1 | 89 | 66 | 23 | 94 | 28.1 | 0.167 | 21 |
| **4** | 0 | 137 | 40 | 35 | 168 | 43.1 | 2.288 | 33 |

In [16]:

y **=** df.iloc[:,**-**1:]

y.head()

Out[16]:

|  |  |
| --- | --- |
|  | **Outcome** |
| **0** | 1 |
| **1** | 0 |
| **2** | 1 |
| **3** | 0 |
| **4** | 1 |

### Split into test and train dataset (70-30 ratio)

In [17]:

**from** sklearn.model\_selection **import** train\_test\_split

x\_train, x\_test, y\_train, y\_test **=** train\_test\_split(x,y,test\_size**=**0.3,random\_state**=**0)

In [18]:

print("Dataset shape:", df.shape)

print("Input Features shape: ", x\_train.shape, y\_train.shape)

print("Output Features shape: ", x\_test.shape, y\_test.shape)

Dataset shape: (768, 9)  
Input Features shape: (537, 8) (537, 1)  
Output Features shape: (231, 8) (231, 1)

### Applying Logistic Regression

In [19]:

**from** sklearn.linear\_model **import** LogisticRegression

log **=** LogisticRegression()

#### Fitting model

In [20]:

log.fit(x\_train,y\_train)

Out[20]:

LogisticRegression(C=1.0, class\_weight=None, dual=False, fit\_intercept=True,  
 intercept\_scaling=1, l1\_ratio=None, max\_iter=100,  
 multi\_class='auto', n\_jobs=None, penalty='l2',  
 random\_state=None, solver='lbfgs', tol=0.0001, verbose=0,  
 warm\_start=False)

#### Predicting values

In [21]:

pred **=** log.predict(x\_test)

In [17]:

pred

Out[17]:

array([1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0,  
 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1,  
 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1,  
 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  
 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1,  
 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  
 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,  
 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,  
 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0,  
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,  
 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0], dtype=int64)

#### Finding accuracy score and confusion matrix

In [22]:

**from** sklearn.metrics **import** accuracy\_score, confusion\_matrix

In [23]:

accuracy\_score(y\_test,pred, normalize**=True**)

Out[23]:

0.7792207792207793

In [24]:

confusion\_matrix(y\_test,pred)

Out[24]:

array([[141, 16],  
 [ 35, 39]], dtype=int64)

In [ ]:

## Applying KNN

In [25]:

**from** sklearn.neighbors **import** KNeighborsClassifier

knn **=** KNeighborsClassifier()

#### Fitting model

In [26]:

knn.fit(x\_train,y\_train)

Out[26]:

KNeighborsClassifier(algorithm='auto', leaf\_size=30, metric='minkowski',  
 metric\_params=None, n\_jobs=None, n\_neighbors=5, p=2,  
 weights='uniform')

#### Predicting values

In [27]:

ypred **=** knn.predict(x\_test)

In [28]:

ypred

Out[28]:

array([1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0,  
 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1,  
 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1,  
 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  
 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,  
 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0,  
 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,  
 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0,  
 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,  
 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, 1, 1, 0, 0, 0, 0, 1], dtype=int64)

#### Finding accuracy score and confusion matrix

In [29]:

accuracy\_score(y\_test, ypred, normalize**=True**)

Out[29]:

0.7489177489177489

In [30]:

confusion\_matrix(y\_test, ypred)

Out[30]:

array([[134, 23],  
 [ 35, 39]], dtype=int64)

## Conclusion

KNN with k=5 has the accuracy of 74.89% on test dataset while Logistic Regression is having 77.92% accuracy on test dataset

Hence, Logistic Regression is predicting more accurately than KNN

## By Standardising features

In [1]:

**from** sklearn.preprocessing **import** StandardScaler

sc **=** StandardScaler()

In [4]:

df.head()

Out[4]:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **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 |

In [5]:

sc\_input **=** sc.fit\_transform(df.drop(['Outcome'], axis **=** 1))

sc\_input

Out[5]:

array([[ 0.63994726, 0.84832379, 0.14964075, ..., 0.20401277,  
 0.46849198, 1.4259954 ],  
 [-0.84488505, -1.12339636, -0.16054575, ..., -0.68442195,  
 -0.36506078, -0.19067191],  
 [ 1.23388019, 1.94372388, -0.26394125, ..., -1.10325546,  
 0.60439732, -0.10558415],  
 ...,  
 [ 0.3429808 , 0.00330087, 0.14964075, ..., -0.73518964,  
 -0.68519336, -0.27575966],  
 [-0.84488505, 0.1597866 , -0.47073225, ..., -0.24020459,  
 -0.37110101, 1.17073215],  
 [-0.84488505, -0.8730192 , 0.04624525, ..., -0.20212881,  
 -0.47378505, -0.87137393]])

In [6]:

df\_input **=** pd.DataFrame(sc\_input, columns**=** df.columns[:**-**1])

df\_input

Out[6]:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Pregnancies** | **Glucose** | **BloodPressure** | **SkinThickness** | **Insulin** | **BMI** | **DiabetesPedigreeFunction** | **Age** |
| **0** | 0.639947 | 0.848324 | 0.149641 | 0.907270 | -0.692891 | 0.204013 | 0.468492 | 1.425995 |
| **1** | -0.844885 | -1.123396 | -0.160546 | 0.530902 | -0.692891 | -0.684422 | -0.365061 | -0.190672 |
| **2** | 1.233880 | 1.943724 | -0.263941 | -1.288212 | -0.692891 | -1.103255 | 0.604397 | -0.105584 |
| **3** | -0.844885 | -0.998208 | -0.160546 | 0.154533 | 0.123302 | -0.494043 | -0.920763 | -1.041549 |
| **4** | -1.141852 | 0.504055 | -1.504687 | 0.907270 | 0.765836 | 1.409746 | 5.484909 | -0.020496 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... |
| **763** | 1.827813 | -0.622642 | 0.356432 | 1.722735 | 0.870031 | 0.115169 | -0.908682 | 2.532136 |
| **764** | -0.547919 | 0.034598 | 0.046245 | 0.405445 | -0.692891 | 0.610154 | -0.398282 | -0.531023 |
| **765** | 0.342981 | 0.003301 | 0.149641 | 0.154533 | 0.279594 | -0.735190 | -0.685193 | -0.275760 |
| **766** | -0.844885 | 0.159787 | -0.470732 | -1.288212 | -0.692891 | -0.240205 | -0.371101 | 1.170732 |
| **767** | -0.844885 | -0.873019 | 0.046245 | 0.656358 | -0.692891 | -0.202129 | -0.473785 | -0.871374 |

768 rows × 8 columns

### Spliting into training and test data

In [7]:

**from** sklearn.model\_selection **import** train\_test\_split

X\_train, X\_test, y\_train, y\_test **=** train\_test\_split(df\_input, df['Outcome'], test\_size**=**0.30)

#### Applying KNN for K=1

In [8]:

**from** sklearn.neighbors **import** KNeighborsClassifier

knn **=** KNeighborsClassifier(n\_neighbors**=** 1)

#### Training and Predicting

In [9]:

knn.fit(X\_train,y\_train)

pred **=** knn.predict(X\_test)

**from** sklearn.metrics **import** classification\_report,confusion\_matrix

**from** sklearn.model\_selection **import** cross\_val\_score

print("\n",confusion\_matrix(y\_test, pred))

print(classification\_report(y\_test, pred))

[[124 28]  
 [ 37 42]]  
 precision recall f1-score support  
  
 0 0.77 0.82 0.79 152  
 1 0.60 0.53 0.56 79  
  
 accuracy 0.72 231  
 macro avg 0.69 0.67 0.68 231  
weighted avg 0.71 0.72 0.71 231

### Analyzing visually with accuracy rate

In [10]:

accuracy\_rate **=** [ ]

**for** i **in** range(1,40):

k **=** KNeighborsClassifier(n\_neighbors**=**i)

score **=** cross\_val\_score(k, df\_input , df['Outcome'], cv**=**10)

accuracy\_rate.append(score.mean())

In [11]:

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

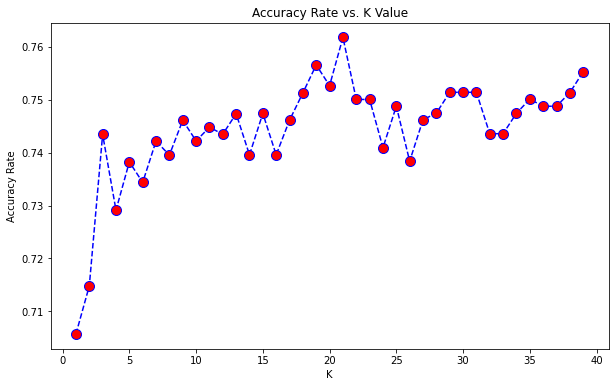
plt.plot(range(1,40), accuracy\_rate, color**=**'blue', linestyle**=**'dashed', marker**=**'o', markerfacecolor**=**'red', markersize**=**10)

plt.title('Accuracy Rate vs. K Value')

plt.xlabel('K')

plt.ylabel('Accuracy Rate')

plt.show()



### Analysing Visually by Error rate

In [12]:

error\_rate **=** [ ]

**for** i **in** range(1,40):

k **=** KNeighborsClassifier(n\_neighbors**=**i)

score **=** cross\_val\_score(k, df\_input , df['Outcome'], cv**=**10)

error\_rate.append(1**-**score.mean())

In [13]:

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

plt.plot(range(1,40), error\_rate, color**=**'blue', linestyle**=**'dashed', marker**=**'o', markerfacecolor**=**'red', markersize**=**10)

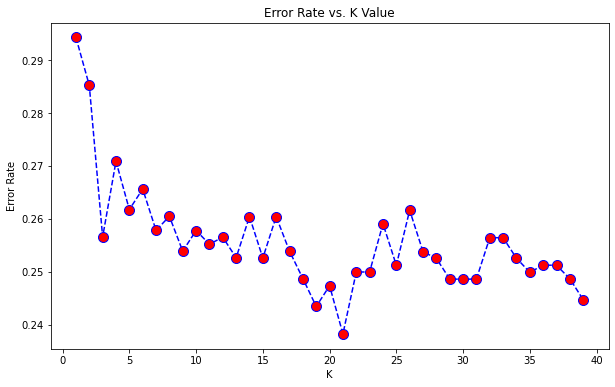
plt.title('Error Rate vs. K Value')

plt.xlabel('K')

plt.ylabel('Error Rate')

Out[13]:

Text(0, 0.5, 'Error Rate')



k > 27 error rate keeps decreasing. now finding clasification report at k=27

In [14]:

knn **=** KNeighborsClassifier(n\_neighbors**=**27)

knn.fit(X\_train,y\_train)

pred **=** knn.predict(X\_test)

print('WITH K=27')

print('\n')

print(confusion\_matrix(y\_test,pred))

print('\n')

print(classification\_report(y\_test,pred))

WITH K=27  
  
  
[[142 10]  
 [ 48 31]]  
  
  
 precision recall f1-score support  
  
 0 0.75 0.93 0.83 152  
 1 0.76 0.39 0.52 79  
  
 accuracy 0.75 231  
 macro avg 0.75 0.66 0.67 231  
weighted avg 0.75 0.75 0.72 231

### Applying Logistic Regression

In [19]:

**from** sklearn.linear\_model **import** LogisticRegression

log **=** LogisticRegression()

*# Fitting model*

log.fit(X\_train,y\_train)

*# Predicting values*

pred **=** log.predict(X\_test)

### Accuracy of logistic regression

In [20]:

accuracy\_score(y\_test, pred)

Out[20]:

0.7532467532467533

---------------------------------------------------------------------------------------------------------------------------------------------