In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
```

In [2]:
```
d = r'C:\Users\Pratik Sonawane\Downloads\diabetes.csv'
df = pd.read_csv(d)
```

In [3]:
```
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

In [4]:
```
df.shape
```

Out[4]:
```
(768, 9)
```

In [5]:
```
df.isnull().sum()
```

Out[5]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

In [6]:
```
df.describe()
```

Out[6]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

In [7]:
```
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
```

In [8]:
```
# Target Variababsl

df['Outcome'].value_counts()
```

Out[8]:
```
Outcome
0    500
1    268
Name: count, dtype: int64
```

- 500 patients (about 65%) have Outcome = 0 (no diabetes).
- 268 patients (about 35%) have Outcome = 1 (diabetes).

In [9]:
```
grouped_means = df.groupby('Outcome').mean().T

# Calculate percentage difference
grouped_means['% diff'] = 100 * (grouped_means[1] - grouped_means[0]) / grouped_means[0]

print(grouped_means)
```

```
Outcome                            0           1     % diff
Pregnancies                 3.298000    4.865672  47.534010
Glucose                   109.980000  141.257463  28.439228
BloodPressure              68.184000   70.824627   3.872795
SkinThickness              19.664000   22.164179  12.714499
Insulin                    68.792000  100.335821  45.853909
BMI                        30.304200   35.142537  15.965897
DiabetesPedigreeFunction    0.429734    0.550500  28.102501
Age                        31.190000   37.067164  18.843104
```

High differences (above 25%):

- Pregnancies: Diabetic patients have almost 48% more pregnancies on average, suggesting a potential link between pre-pregnancy factors and diabetes risk.
- Glucose: Blood glucose levels are around 28% higher in the diabetic group, highlighting impaired glucose regulation as a major characteristic of the disease.
- Insulin: Diabetic patients show a significant 46% increase in average insulin levels, indicating insufficient or ineffective insulin production or utilization.
- DiabetesPedigreeFunction: A 28% increase in this genetic score suggests a possible hereditary influence on diabetes risk.

Moderate differences (around 10-25%):

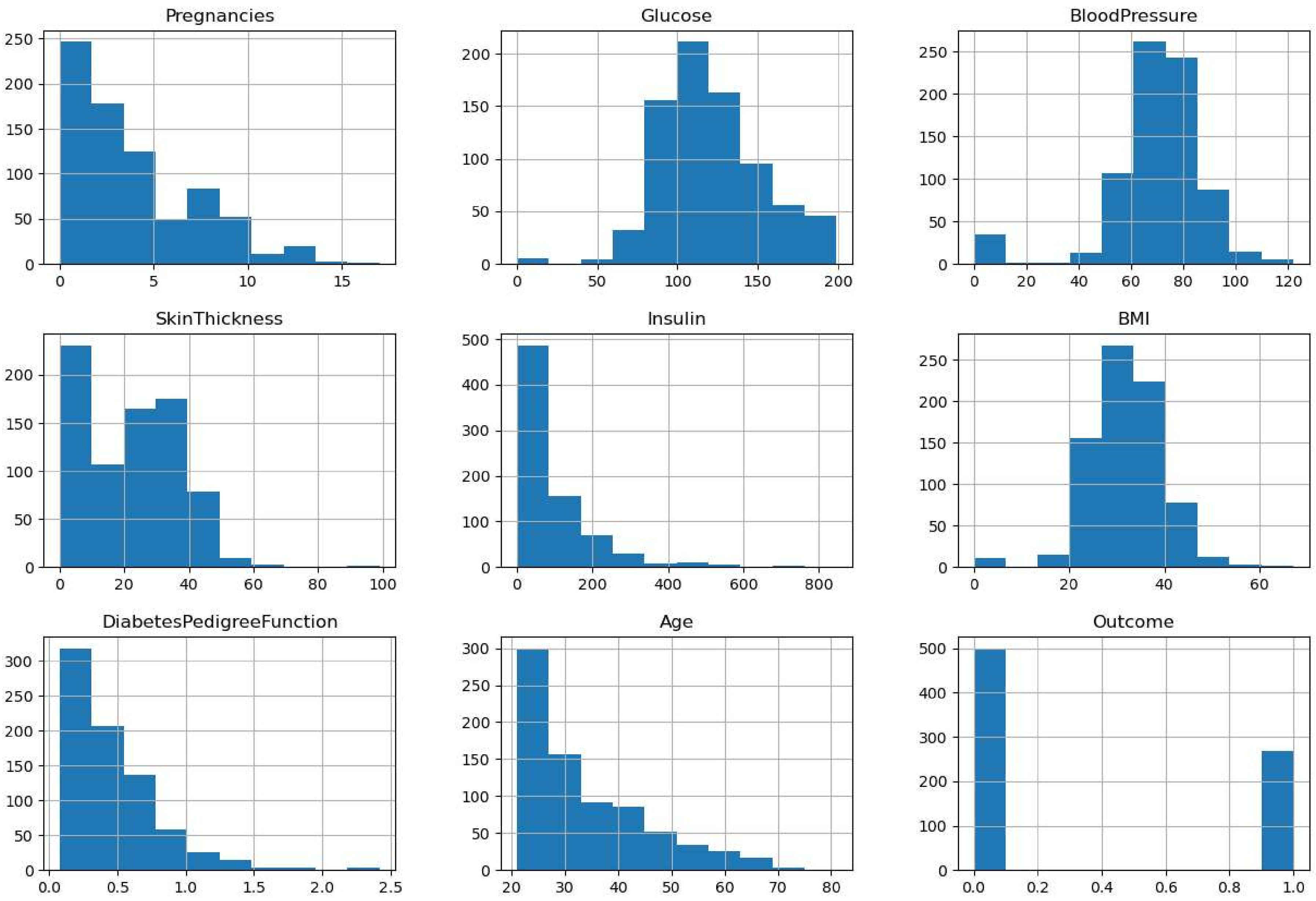
- BMI: Diabetic patients have a 16% higher average BMI, reflecting a potential role of body weight in diabetes development.
- SkinThickness: Increased skin thickness by 13% could be a marker for insulin resistance or metabolic changes associated with diabetes.
- Age: There's a 19% difference in average age, with diabetic patients being older on average. This could be due to age-related changes in metabolism or longer exposure to risk factors.

Low difference (below 5%):

- BloodPressure: While slightly higher in diabetic patients, the difference in average blood pressure is relatively small, indicating it might not be a primary factor for everyone.

```
In [10]: df.hist(bins=10,figsize = (15,10))
```

```
Out[10]: array([[<Axes: title={'center': 'Pregnancies'}>,
                 <Axes: title={'center': 'Glucose'}>,
                 <Axes: title={'center': 'BloodPressure'}>],
                [<Axes: title={'center': 'SkinThickness'}>,
                 <Axes: title={'center': 'Insulin'}>,
                 <Axes: title={'center': 'BMI'}>],
                [<Axes: title={'center': 'DiabetesPedigreeFunction'}>,
                 <Axes: title={'center': 'Age'}>,
                 <Axes: title={'center': 'Outcome'}>]], dtype=object)
```


Pregnancies
Glucose
BloodPressure
SkinThickness
Insulin
BMI
DiabetesPedigreeFunction
Age
Outcome

In [11]:
```
df.corr()['Outcome']
```

Out[11]:
```
Pregnancies                 0.221898
Glucose                     0.466581
BloodPressure               0.065068
SkinThickness               0.074752
Insulin                     0.130548
BMI                         0.292695
DiabetesPedigreeFunction    0.173844
Age                         0.238356
Outcome                     1.000000
Name: Outcome, dtype: float64
```

- A high correlation simply indicates a linear relationship between two variables, but it doesn't necessarily mean one directly causes the other.
- Features with seemingly weak correlations might still be valuable based on ones understanding of the problem and the underlying biology.

In [15]:
```
X = df.drop(columns = ['Outcome'])
Y = df['Outcome']
```

In [17]:
```
scaler = StandardScaler()
```

In [19]:
```
scaler.fit(X)
```

Out[19]:
```
StandardScaler()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [20]:
```
standard_data = scaler.transform(X)
```

```
In [22]: X_scaled = scaler.fit_transform(X)
         X_scaled
```

```
Out[22]: 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 [25]: X_train,X_test,y_train,y_test = train_test_split(X_scaled,Y,test_size = 0.2,stratify = Y,random_state=42)
```

```
In [27]: print(X.shape,X_train.shape,X_test.shape)
```

```
(768, 8) (614, 8) (154, 8)
```

```
In [28]: from sklearn.linear_model import LogisticRegression
         from sklearn.tree import DecisionTreeClassifier
         from sklearn.ensemble import RandomForestClassifier
         from sklearn.svm import SVC
         from sklearn.neighbors import KNeighborsClassifier
         from sklearn.ensemble import GradientBoostingClassifier
```

In [29]:
```
lr = LogisticRegression()

dtc = DecisionTreeClassifier()

rf = RandomForestClassifier()

svm = SVC()

knn = KNeighborsClassifier()

GBM = GradientBoostingClassifier()
```

In [31]:
```
lr.fit(X_train,y_train)
dtc.fit(X_train,y_train)
rf.fit(X_train,y_train)
svm.fit(X_train,y_train)
knn.fit(X_train,y_train)
GBM.fit(X_train,y_train)
```

Out[31]:
```
GradientBoostingClassifier()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [32]:
```
from sklearn.metrics import classification_report
```

```
In [34]: def model_evaluation(model):
             y_pred = model.predict(X_test)
             accuracy = accuracy_score(y_test, y_pred)
             return y_pred, accuracy

         lr_pred, lr_accuracy = model_evaluation(lr)
         dtc_pred, dtc_accuracy = model_evaluation(dtc)
         rf_pred, rf_accuracy = model_evaluation(rf)
         svm_pred, svm_accuracy = model_evaluation(svm)
         knn_pred, knn_accuracy = model_evaluation(knn)
         GBM_pred, GBM_accuracy = model_evaluation(GBM)

         print(f"Logistic Regression accuracy: {lr_accuracy:.4f}")
         print(f"Decision Tree accuracy: {dtc_accuracy:.4f}")
         print(f"Random Forest accuracy: {rf_accuracy:.4f}")
         print(f"Support Vector Machine accuracy: {svm_accuracy:.4f}")
         print(f"K-Nearest Neighbors accuracy: {knn_accuracy:.4f}")
         print(f"Gradient Boosting Model accuracy: {GBM_accuracy:.4f}")
```

```
Logistic Regression accuracy: 0.7143
Decision Tree accuracy: 0.7208
Random Forest accuracy: 0.7532
Support Vector Machine accuracy: 0.7468
K-Nearest Neighbors accuracy: 0.7078
Gradient Boosting Model accuracy: 0.7532
```

rf and GBM giving same score , preparing classification report and model with best recall score will be best for our project

In [39]:
```
from sklearn.metrics import classification_report

rf_report = classification_report(y_test, rf.predict(X_test))
print('rf :',rf_report)

GBM_report = classification_report(y_test, GBM.predict(X_test))
print('GBM :',GBM_report)
```

```
rf :               precision    recall  f1-score   support

           0       0.79      0.84      0.82       100
           1       0.67      0.59      0.63        54

    accuracy                           0.75       154
   macro avg       0.73      0.72      0.72       154
weighted avg       0.75      0.75      0.75       154

GBM :               precision    recall  f1-score   support

           0       0.79      0.84      0.82       100
           1       0.67      0.59      0.63        54

    accuracy                           0.75       154
   macro avg       0.73      0.72      0.72       154
weighted avg       0.75      0.75      0.75       154
```

In [40]:
```
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import cross_val_score
```

```
In [45]: lrcv = cross_val_score(lr,X_scaled,Y,cv = StratifiedKFold(n_splits=5)).mean()
         print('lr :', lrcv)
         dtccv = cross_val_score(dtc,X_scaled,Y,cv = StratifiedKFold(n_splits=5)).mean()
         print('dtccv :',dtccv)
         rfcv = cross_val_score(rf,X_scaled,Y,cv = StratifiedKFold(n_splits=5)).mean()
         print('rfcv :',rfcv)
         svmcv = cross_val_score(svm,X_scaled,Y,cv = StratifiedKFold(n_splits=5)).mean()
         print('svmcv :', svmcv)
         knncv = cross_val_score(knn,X_scaled,Y,cv = StratifiedKFold(n_splits=5)).mean()
         print('knncv :', knncv)
         GBMcv = cross_val_score(GBM,X_scaled,Y,cv = StratifiedKFold(n_splits=5)).mean()
         print('GBMcv :', GBMcv)
```

```
lr : 0.7708853238265002
dtccv : 0.7201680672268906
rfcv : 0.7643833290892115
svmcv : 0.7708938120702827
knncv : 0.733112638994992
GBMcv : 0.7578728461081402
```

Predictive System using SVM Model

```
In [46]: X.columns
```

```
Out[46]: Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
                'BMI', 'DiabetesPedigreeFunction', 'Age'],
               dtype='object')
```

```
In [48]: Pregnancies = int(input('Enter the Pregnancies =',))
         Glucose = float(input('Enter the Glucose =',))
         BloodPressure = float(input('Enter the BloodPressure =',))
         SkinThickness = float(input('Enter the SkinThickness =',))
         Insulin = float(input('Enter the Insulin =',))
         BMI = float(input('Enter the BMI =',))
         DiabetesPedigreeFunction = float(input('Enter the DiabetesPedigreeFunction =',))
         Age = float(input('Enter the Age =',))

         patient_data = [Pregnancies,Glucose,BloodPressure,SkinThickness,Insulin,
                         BMI,DiabetesPedigreeFunction,Age]
         patient_data

         Enter the Pregnancies =5
         Enter the Glucose =120
         Enter the BloodPressure =140
         Enter the SkinThickness =28
         Enter the Insulin =95
         Enter the BMI =36
         Enter the DiabetesPedigreeFunction =0.52
         Enter the Age =42

Out[48]: [5, 120.0, 140.0, 28.0, 95.0, 36.0, 0.52, 42.0]

In [49]: input_array = np.asarray(patient_data)
         input_array

Out[49]: array([  5.  , 120.  , 140.  ,  28.  ,  95.  ,  36.  ,   0.52,  42.  ])

In [50]: # Reshape in 2D
         input_data = input_array.reshape(1,-1)
         input_data

Out[50]: array([[  5.  , 120.  , 140.  ,  28.  ,  95.  ,  36.  ,   0.52,  42.  ]])
```

```
In [51]: std_data = scaler.transform(input_data)
         std_data
```

```
C:\Users\Pratik Sonawane\anaconda3\Lib\site-packages\sklearn\base.py:493: UserWarning: X does not have valid feature
names, but StandardScaler was fitted with feature names
  warnings.warn(
```

```
Out[51]: array([[ 0.3429808 , -0.02799627,  3.66508775,  0.4681735 ,  0.13198454,
                  0.50861896,  0.14533928,  0.74529338]])
```

```
In [52]: prediction = svm.predict(std_data)
         prediction
```

```
Out[52]: array([0], dtype=int64)
```

```
In [53]: prediction[0]
```

```
Out[53]: 0
```

Model has accurately predicted above pateint data with no diabetes(0)

Prediction System

In [57]:
```python
import warnings
warnings.filterwarnings('ignore')

Pregnancies = int(input('Enter the Pregnancies =',))
Glucose = float(input('Enter the Glucose =',))
BloodPressure = float(input('Enter the BloodPressure =',))
SkinThickness = float(input('Enter the SkinThickness =',))
Insulin = float(input('Enter the Insulin =',))
BMI = float(input('Enter the BMI =',))
DiabetesPedigreeFunction = float(input('Enter the DiabetesPedigreeFunction =',))
Age = float(input('Enter the Age =',))

patient_data = [Pregnancies,Glucose,BloodPressure,SkinThickness,Insulin,
                BMI,DiabetesPedigreeFunction,Age]

input_array = np.asarray(patient_data)
input_data = input_array.reshape(1,-1)
std_data = scaler.transform(input_data)
prediction = svm.predict(std_data)

print('***************Model Predictions*******************')
if (prediction[0] ==0):
    print('Result - Negative')
    print('The person is Not Diabetic')
else:
    print('Result - Positive')
    print('The Patient is Diabetic')
```

```
Enter the Pregnancies =6
Enter the Glucose =148
Enter the BloodPressure =72
Enter the SkinThickness =35
Enter the Insulin =0
Enter the BMI =33.6
Enter the DiabetesPedigreeFunction =0.627
Enter the Age =50
***************Model Predictions*******************
Result - Positive
The Patient is Diabetic
```