# Import libraries

In [1]:

```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score,f1_score,precision_score,recall_score,plot_confusion
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler

import warnings
warnings.filterwarnings('ignore')
```

# Data preparation

In [2]:

```
df=pd.read_csv("C:\Akshit\MOR\projects\logistic\diabetes.csv")
```

ABOUT COLUMNS

-Pregnancies: Number of times pregnant

-Glucose: Plasma glucose concentration a 2 hours in an oral glucose tolerance test

-BloodPressure: Diastolic blood pressure (mm Hg)

-SkinThickness: Triceps skin fold thickness (mm)

-Insulin: 2-Hour serum insulin (mu U/ml)

-BMI: Body mass index (weight in kg/(height in m)^2)

-DiabetesPedigreeFunction: Diabetes pedigree function

-Age: Age (years)

-Outcome: Class variable (0 or 1)

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.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 [5]:

```
df.describe()
```

Out[5]:

| | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | DiabetesPedigreeFunction |
|---|---|---|---|---|---|---|---|
| **count** | 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 |
| **std** | 3.369578 | 31.972618 | 19.355807 | 15.952218 | 115.244002 | 7.884160 | 0.331329 |
| **min** | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.078000 |
| **25%** | 1.000000 | 99.000000 | 62.000000 | 0.000000 | 0.000000 | 27.300000 | 0.243750 |
| **50%** | 3.000000 | 117.000000 | 72.000000 | 23.000000 | 30.500000 | 32.000000 | 0.372500 |
| **75%** | 6.000000 | 140.250000 | 80.000000 | 32.000000 | 127.250000 | 36.600000 | 0.626250 |
| **max** | 17.000000 | 199.000000 | 122.000000 | 99.000000 | 846.000000 | 67.100000 | 2.420000 |

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
```

No missing values are present. However from the above functions we can see that Glucose, BloodPressure, SkinThickness, Insulin and BMI have min values of 0 which are not real clinical values. Lets look at counts. A value of 0 for Pregnancies is a real value.

In [7]:

```
print("Number of 0's for Glucose:", df['Glucose'].isin([0]).sum())
print("Number of 0's for Blood Pressure:", df['BloodPressure'].isin([0]).sum())
print("Number of 0's for Skin Thickness:", df['SkinThickness'].isin([0]).sum())
print("Number of 0's for Insulin:", df['Insulin'].isin([0]).sum())
print("Number of 0's for BMI:", df['BMI'].isin([0]).sum())
```

```
Number of 0's for Glucose: 5
Number of 0's for Blood Pressure: 35
Number of 0's for Skin Thickness: 227
Number of 0's for Insulin: 374
Number of 0's for BMI: 11
```

Replacing 0 values in these columns with mean.

In [8]:
```python
diabetes_clean = df.copy()
```

In [9]:
```python
diabetes_clean['Glucose'] = diabetes_clean['Glucose'].replace(0,df['Glucose'].mean())
diabetes_clean['BloodPressure'] = diabetes_clean['BloodPressure'].replace(0,df['BloodPressure']
diabetes_clean['SkinThickness'] = diabetes_clean['SkinThickness'].replace(0,df['SkinThickness']
diabetes_clean['Insulin'] = diabetes_clean['Insulin'].replace(0,df['Insulin'].mean())
diabetes_clean['BMI'] = diabetes_clean['BMI'].replace(0,df['BMI'].mean())
```

In [10]:
```python
diabetes_clean.head()
```

Out[10]:

| | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | DiabetesPedigreeFunction | Age | Outcon |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 6 | 148.0 | 72.0 | 35.000000 | 79.799479 | 33.6 | 0.627 | 50 | |
| **1** | 1 | 85.0 | 66.0 | 29.000000 | 79.799479 | 26.6 | 0.351 | 31 | |
| **2** | 8 | 183.0 | 64.0 | 20.536458 | 79.799479 | 23.3 | 0.672 | 32 | |
| **3** | 1 | 89.0 | 66.0 | 23.000000 | 94.000000 | 28.1 | 0.167 | 21 | |
| **4** | 0 | 137.0 | 40.0 | 35.000000 | 168.000000 | 43.1 | 2.288 | 33 | |

In [11]:
```python
diabetes_clean.describe()
```

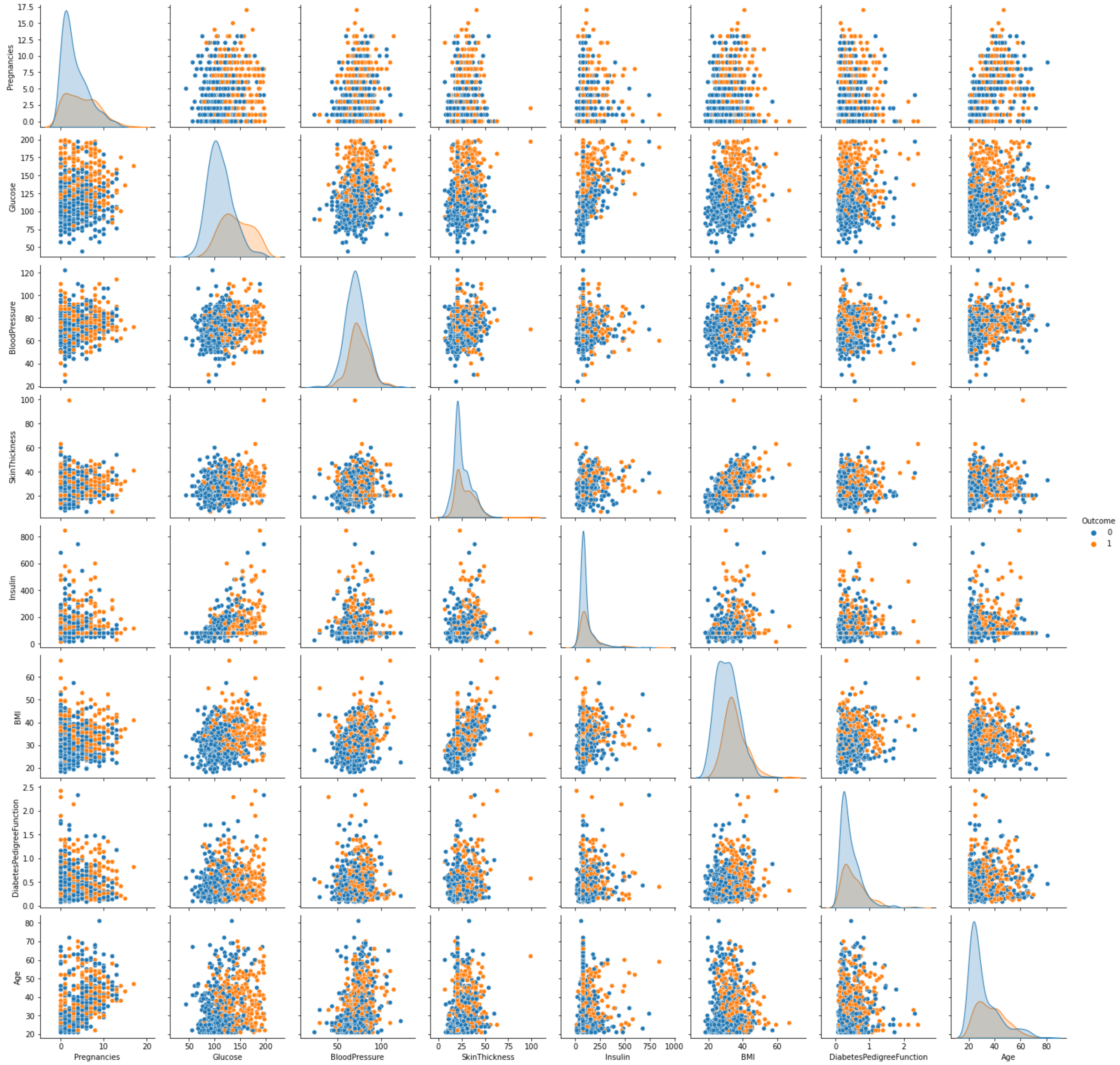
Out[11]:

| | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | DiabetesPedigreeFunction |
|---|---|---|---|---|---|---|---|
| **count** | 768.000000 | 768.000000 | 768.000000 | 768.000000 | 768.000000 | 768.000000 | 768.000000 |
| **mean** | 3.845052 | 121.681605 | 72.254807 | 26.606479 | 118.660163 | 32.450805 | 0.471876 |
| **std** | 3.369578 | 30.436016 | 12.115932 | 9.631241 | 93.080358 | 6.875374 | 0.331329 |
| **min** | 0.000000 | 44.000000 | 24.000000 | 7.000000 | 14.000000 | 18.200000 | 0.078000 |
| **25%** | 1.000000 | 99.750000 | 64.000000 | 20.536458 | 79.799479 | 27.500000 | 0.243750 |
| **50%** | 3.000000 | 117.000000 | 72.000000 | 23.000000 | 79.799479 | 32.000000 | 0.372500 |
| **75%** | 6.000000 | 140.250000 | 80.000000 | 32.000000 | 127.250000 | 36.600000 | 0.626250 |
| **max** | 17.000000 | 199.000000 | 122.000000 | 99.000000 | 846.000000 | 67.100000 | 2.420000 |

# Exploratory Data Analysis

In [12]:
```python
sns.pairplot(diabetes_clean, diag_kind='kde', hue='Outcome')
```

Out[12]:
```
<seaborn.axisgrid.PairGrid at 0x1278241cbb0>
```




In [13]:

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

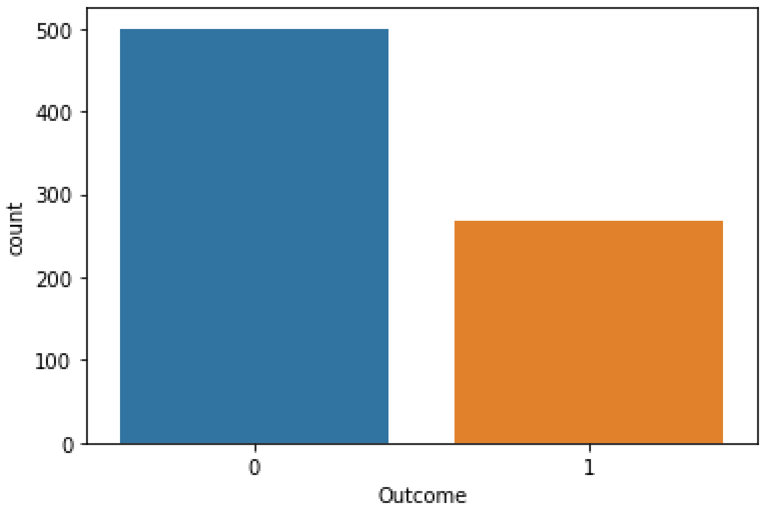
Out[13]:

```
0    500
1    268
Name: Outcome, dtype: int64
```

In [14]:

```
sns.countplot(data = diabetes_clean, x = 'Outcome')
```

Out[14]: <AxesSubplot:xlabel='Outcome', ylabel='count'>




In [15]:

```
plt.figure(figsize = [10, 10])
sns.heatmap(diabetes_clean.corr(), annot = True, fmt = '.3f', cmap = 'vlag_r', center = 0)
```

Out[15]: <AxesSubplot:>

# Model Fitting

In [16]:

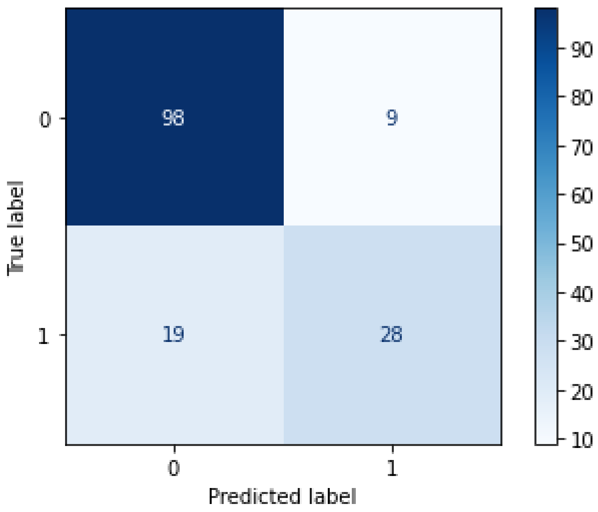
```
# define X and y
y = diabetes_clean['Outcome']
X = diabetes_clean.drop('Outcome', axis=1)

# Splitting the data so 20% is for testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

#feature scaling
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
```

In [17]:

```
# Logistic Regression
# Fitting the model
LRmodel = LogisticRegression()
LRmodel.fit(X_train, y_train)
y_predict = LRmodel.predict(X_test)
#Confusion matrix
plot_confusion_matrix(LRmodel, X_test, y_test, cmap=plt.cm.Blues)
```

Out[17]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x127ff1cfd00>
```




In [18]:

```
print('F1 score: ',f1_score(y_predict,y_test)*100)
print('Accuracy: ',accuracy_score(y_predict,y_test)*100)
print('Precision score: ',precision_score(y_predict,y_test)*100)
print('Recall score: ',recall_score(y_predict,y_test)*100)
```

```
F1 score:  66.66666666666666
Accuracy:  81.81818181818183
Precision score:  59.57446808510638
Recall score:  75.67567567567568
```

In [19]:

```
#Decision tree classifier
DTmodel=DecisionTreeClassifier()
DTmodel.fit(X_train,y_train)
prediction=DTmodel.predict(X_test)
print('F1 score: ',f1_score(prediction,y_test)*100)
print('Accuracy: ',accuracy_score(prediction,y_test)*100)
print('Precision score: ',precision_score(prediction,y_test)*100)
print('Recall score: ',recall_score(prediction,y_test)*100)
```

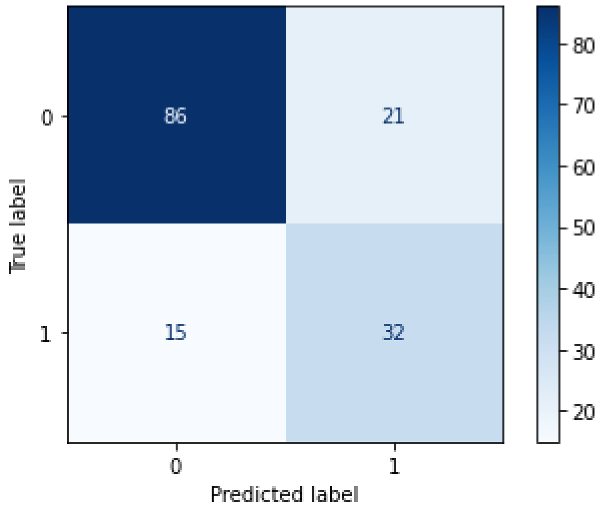
```
F1 score:  64.0
Accuracy:  76.62337662337663
```

```
Precision score:  68.08510638297872
Recall score:  60.37735849056604
```

In [20]:

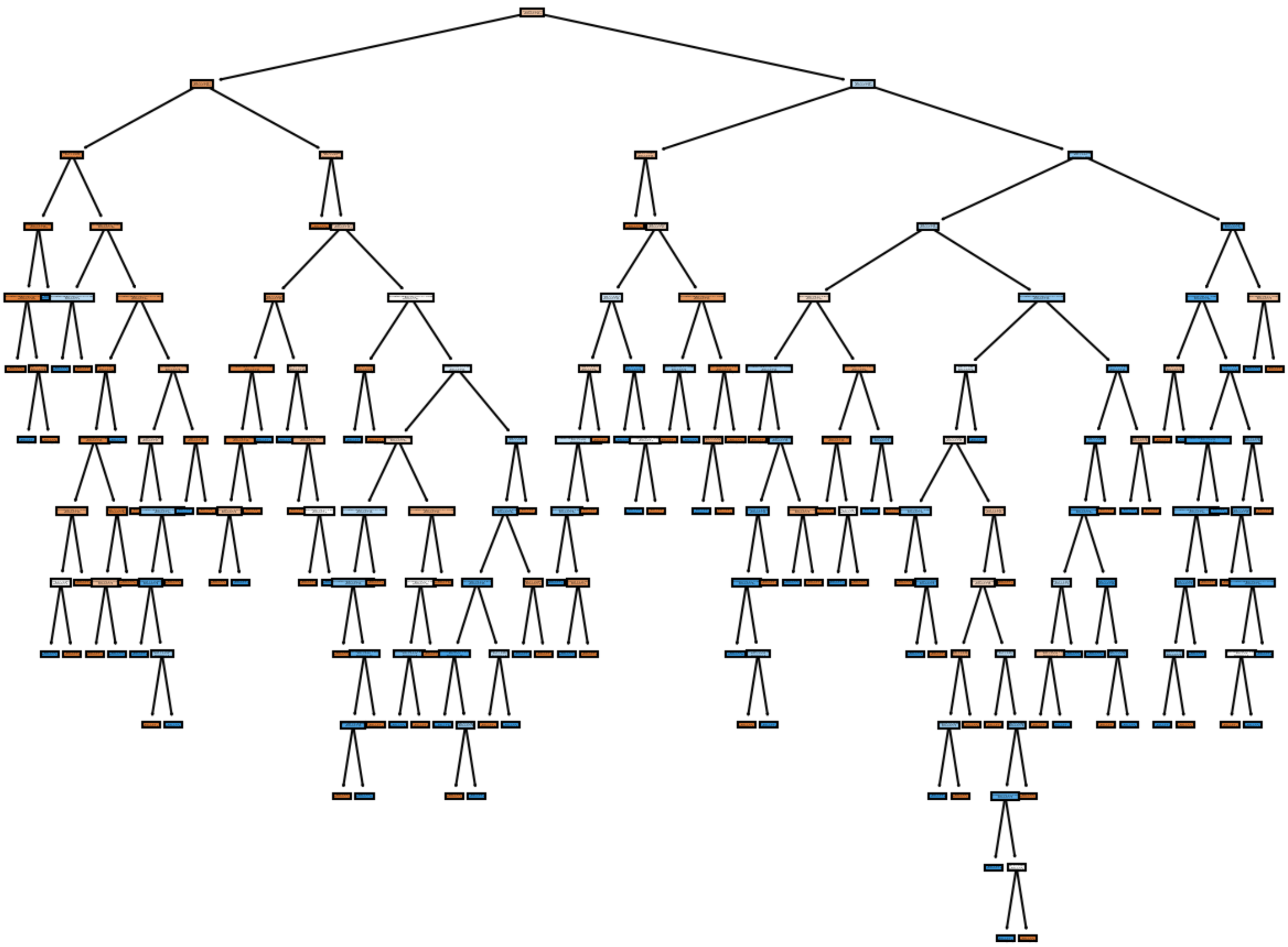
```python
plot_confusion_matrix(DTmodel, X_test, y_test, cmap=plt.cm.Blues)
```

Out[20]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x12785c70220>




In [21]:

```python
from sklearn.tree import plot_tree
plt.figure(figsize=(10,8), dpi=150)
plot_tree(DTmodel, feature_names=X.columns, filled=True);
```



In [22]:

```python
#Random forest classifier
RFmodel=RandomForestClassifier(n_estimators=100,random_state=0)
RFmodel.fit(X_train,y_train)
RF_prediction=RFmodel.predict(X_test)
```

```
print('F1 score: ',f1_score(RF_prediction,y_test)*100)
print('Accuracy: ',accuracy_score(RF_prediction,y_test)*100)
print('Precision score: ',precision_score(RF_prediction,y_test)*100)
print('Recall score: ',recall_score(RF_prediction,y_test)*100)
```

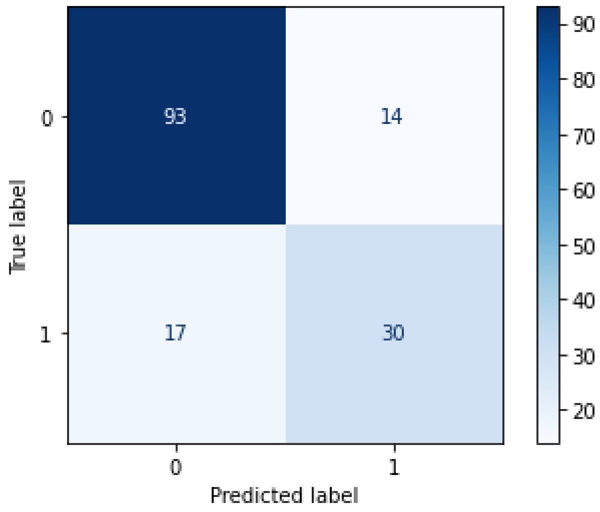
```
F1 score:  65.93406593406593
Accuracy:  79.87012987012987
Precision score:  63.829787234042556
Recall score:  68.18181818181817
```

In [23]:

```
plot_confusion_matrix(RFmodel, X_test, y_test, cmap=plt.cm.Blues)
```

Out[23]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x127862a8bb0>
```




Here we need output sensitive predictions which means it is ok if a non-diabetic person is labeled as diabetic but a diabetic person should not be labeled as non-diabetic,so, as the cost of false positives and false negatives are very different ,we will prefer the model with highest F1 score and recall score. The logistic regression model has the highest F1 score and recall score, so we will consider that model for predictions.