In [1]:
```
import pandas as pd
import seaborn as sns
```

In [2]:
```
data = pd.read_csv('diabetes.csv')
```

In [3]:
```
data.head()
```

Out[3]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Pedigree	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]:
```
data.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   Pedigree       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]:
```
data.shape
```

Out[5]:
```
(768, 9)
```

In [6]:
```
#Check for null or missing values
data.isnull().sum()
```

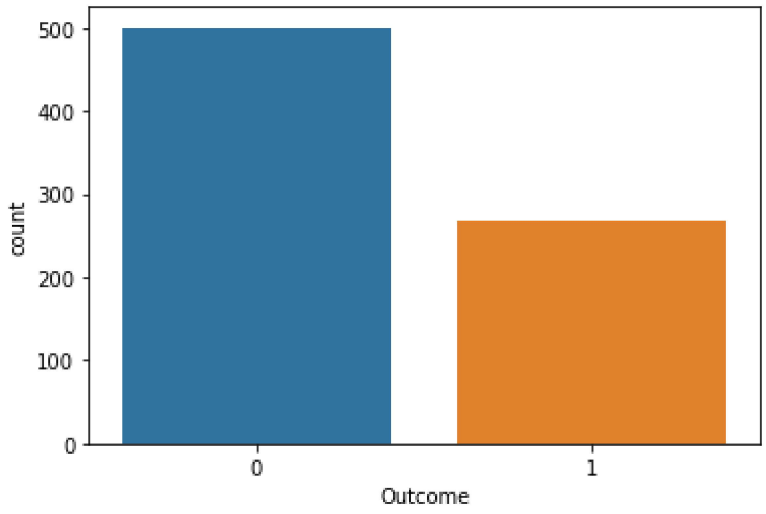
Out[6]:
```
Pregnancies      0
Glucose          0
BloodPressure    0
SkinThickness    0
Insulin          0
BMI              0
Pedigree         0
Age              0
Outcome          0
dtype: int64
```

In [7]:
```
#input data
x= data.drop('Outcome',axis=1)

#output data
y=data['Outcome']
```

In [8]:
```
sns.countplot(x=y)
```

Out[8]:
```
<AxesSubplot:xlabel='Outcome', ylabel='count'>
```



In [9]:
```
y.value_counts()
```

Out[9]:
```
0    500
1    268
Name: Outcome, dtype: int64
```

In [10]:
```
#Feature Scaling---->feature are scaled in one range (0-1)
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler()
x_scaled=scaler.fit_transform(x)
```

In [11]:
```
x_scaled
```

Out[11]:
```
array([[0.35294118, 0.74371859, 0.59016393, ..., 0.50074516, 0.23441503,
        0.48333333],
       [0.05882353, 0.42713568, 0.54098361, ..., 0.39642325, 0.11656704,
        0.16666667],
       [0.47058824, 0.91959799, 0.52459016, ..., 0.34724292, 0.25362938,
        0.18333333],
       ...,
       [0.29411765, 0.6080402 , 0.59016393, ..., 0.390462  , 0.07130658,
        0.15      ],
       [0.05882353, 0.63316583, 0.49180328, ..., 0.4485842 , 0.11571307,
        0.43333333],
       [0.05882353, 0.46733668, 0.57377049, ..., 0.45305514, 0.10119556,
        0.03333333]])
```

In [12]:
```
#Cross Validation
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x_scaled, y, test_size=0.25, random_
```

In [13]:
```
x.shape
```

Out[13]: (768, 8)

In [14]:
```
x_train.shape
```

Out[14]: (576, 8)

In [15]:
```
x_test.shape
```

Out[15]: (192, 8)

K-Nearest Neighbors algorithm

In [16]:
```
#KNN
#import class
from sklearn.neighbors import KNeighborsClassifier
```

In [17]:
```
#Create the object---num of neighbours 2
knn=KNeighborsClassifier(n_neighbors= 33)
```

In [18]:
```
#Train the algorithm
knn.fit(x_train,y_train)
```

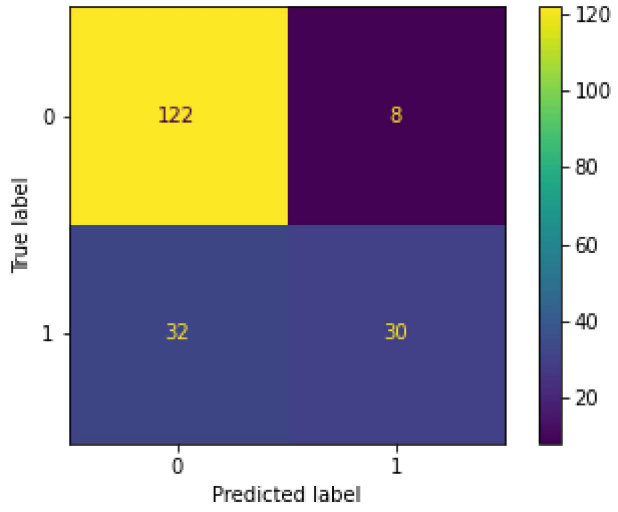
Out[18]: KNeighborsClassifier(n_neighbors=33)

In [19]:
```
#Predict on test data
y_pred=knn.predict(x_test)
```

In [20]:
```
#import the evaluation matrix
from sklearn.metrics import ConfusionMatrixDisplay, accuracy_score, classification_repor
```

In [21]:
```
ConfusionMatrixDisplay.from_predictions(y_test,y_pred)
```

Out[21]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2037377d3d0>



In [22]:
```
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of Logistic Regression: {accuracy}")
```

```
Accuracy of Logistic Regression: 0.7916666666666666
```

In [23]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.79      0.94      0.86       130
           1       0.79      0.48      0.60        62

    accuracy                           0.79       192
   macro avg       0.79      0.71      0.73       192
weighted avg       0.79      0.79      0.78       192
```

Naive Bayes algorithm

In [24]:
```
from sklearn.naive_bayes import GaussianNB
```

In [25]:
```
# Create a Naive Bayes classifier
naive_bayes = GaussianNB()
```

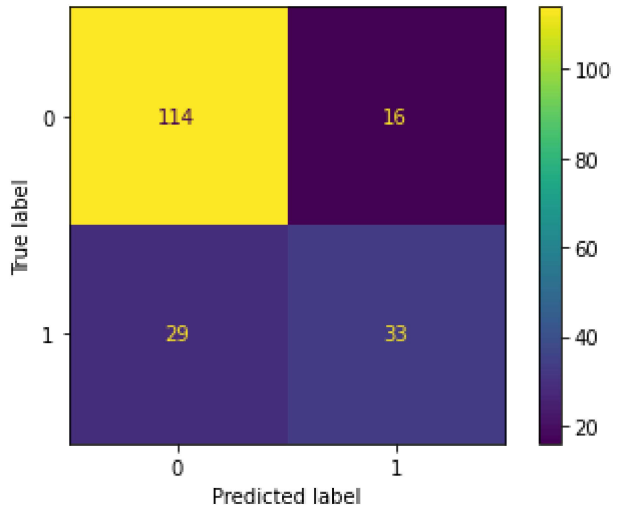
In [26]:
```
# Train the algorithm
naive_bayes.fit(x_train, y_train)
```

Out[26]: GaussianNB()

In [27]:
```
# Predict on test data
y_pred = naive_bayes.predict(x_test)
```

In [28]:
```
ConfusionMatrixDisplay.from_predictions(y_test,y_pred)
```

Out[28]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x203733ec820>



In [29]:
```
# Evaluate the performance
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of Naive Bayes: {accuracy}")
```

```
Accuracy of Naive Bayes: 0.765625
```

In [30]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.80      0.88      0.84       130
           1       0.67      0.53      0.59        62

    accuracy                           0.77       192
   macro avg       0.74      0.70      0.71       192
weighted avg       0.76      0.77      0.76       192
```

Decision Tree algorithm

In [31]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [32]:
```
# Create a Decision Tree classifier
decision_tree = DecisionTreeClassifier()
```

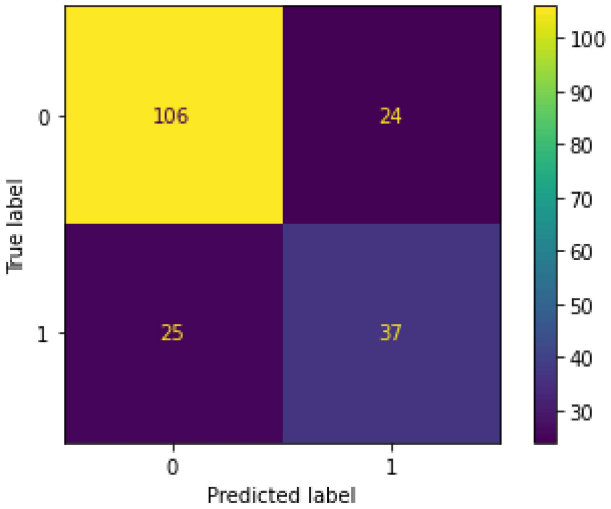
In [33]:
```
# Train the algorithm
decision_tree.fit(x_train, y_train)
```

Out[33]: DecisionTreeClassifier()

In [34]:
```
# Predict on test data
y_pred = decision_tree.predict(x_test)
```

In [35]:
```
ConfusionMatrixDisplay.from_predictions(y_test,y_pred)
```

Out[35]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x20373b57580>



In [36]:
```
# Evaluate the performance
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of Decision Tree: {accuracy}")
```

```
Accuracy of Decision Tree: 0.7447916666666666
```

In [37]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.81      0.82      0.81       130
           1       0.61      0.60      0.60        62

    accuracy                           0.74       192
   macro avg       0.71      0.71      0.71       192
weighted avg       0.74      0.74      0.74       192
```

Random Forest algorithm

In [38]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [39]:
```
# Create a Random Forest classifier
random_forest = RandomForestClassifier(n_estimators=100)  # You can adjust the number of
```

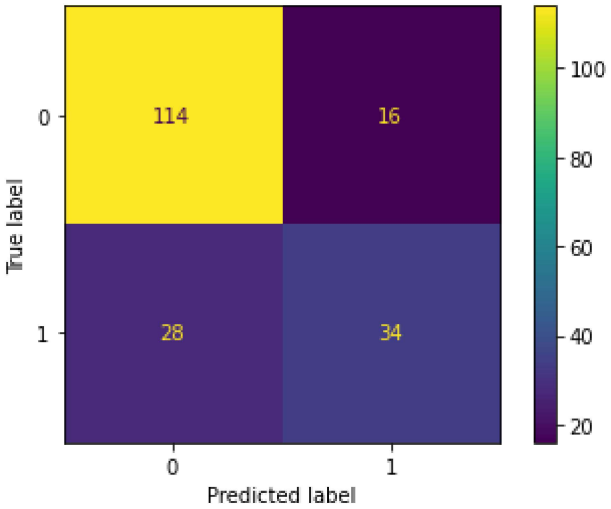
In [40]:
```
# Train the algorithm
random_forest.fit(x_train, y_train)
```

Out[40]:
```
RandomForestClassifier()
```

In [41]:
```
# Predict on test data
y_pred = random_forest.predict(x_test)
```

In [42]:
```
ConfusionMatrixDisplay.from_predictions(y_test,y_pred)
```

Out[42]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x20373a6db50>



In [43]:
```
# Evaluate the performance
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of Random Forest: {accuracy}")
```

```
Accuracy of Random Forest: 0.7708333333333334
```

In [44]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.80      0.88      0.84       130
           1       0.68      0.55      0.61        62

    accuracy                           0.77       192
   macro avg       0.74      0.71      0.72       192
weighted avg       0.76      0.77      0.76       192
```