In [19]:

```
# Import libraries
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import seaborn as sns # for data visualization
import matplotlib.pyplot as plt # to plot charts
from collections import Counter
import os

# Modeling Libraries
from sklearn.preprocessing import QuantileTransformer
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, GradientBoostingClassifier, VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV, cross_val_score, StratifiedKFold, learning_curve, train_test_split
```

In [20]:

```
# Import dataset
df = pd.read_csv("diabetes.csv")
# Get familier with dataset structure
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 [21]:
```
# Show top 5 rows
df.head()
```

Out[21]:

	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 [22]:
```
# Explore missing values
df.isnull().sum()
```

Out[22]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

In [23]:
```
df['Glucose'] = df['Glucose'].replace(0, df['Glucose'].mean())
# Correcting missing values in blood pressure
df['BloodPressure'] = df['BloodPressure'].replace(0, df['BloodPressure'].mean()) # There are 35 records with 0 BloodPressure in da
# Correcting missing values in BMI
df['BMI'] = df['BMI'].replace(0, df['BMI'].median())
# Correct missing values in Insulin and SkinThickness

df['SkinThickness'] = df['SkinThickness'].replace(0, df['SkinThickness'].median())
df['Insulin'] = df['Insulin'].replace(0, df['Insulin'].median())
```

In [24]:
```
# Review dataset statistics
df.describe()
```

Out[24]:

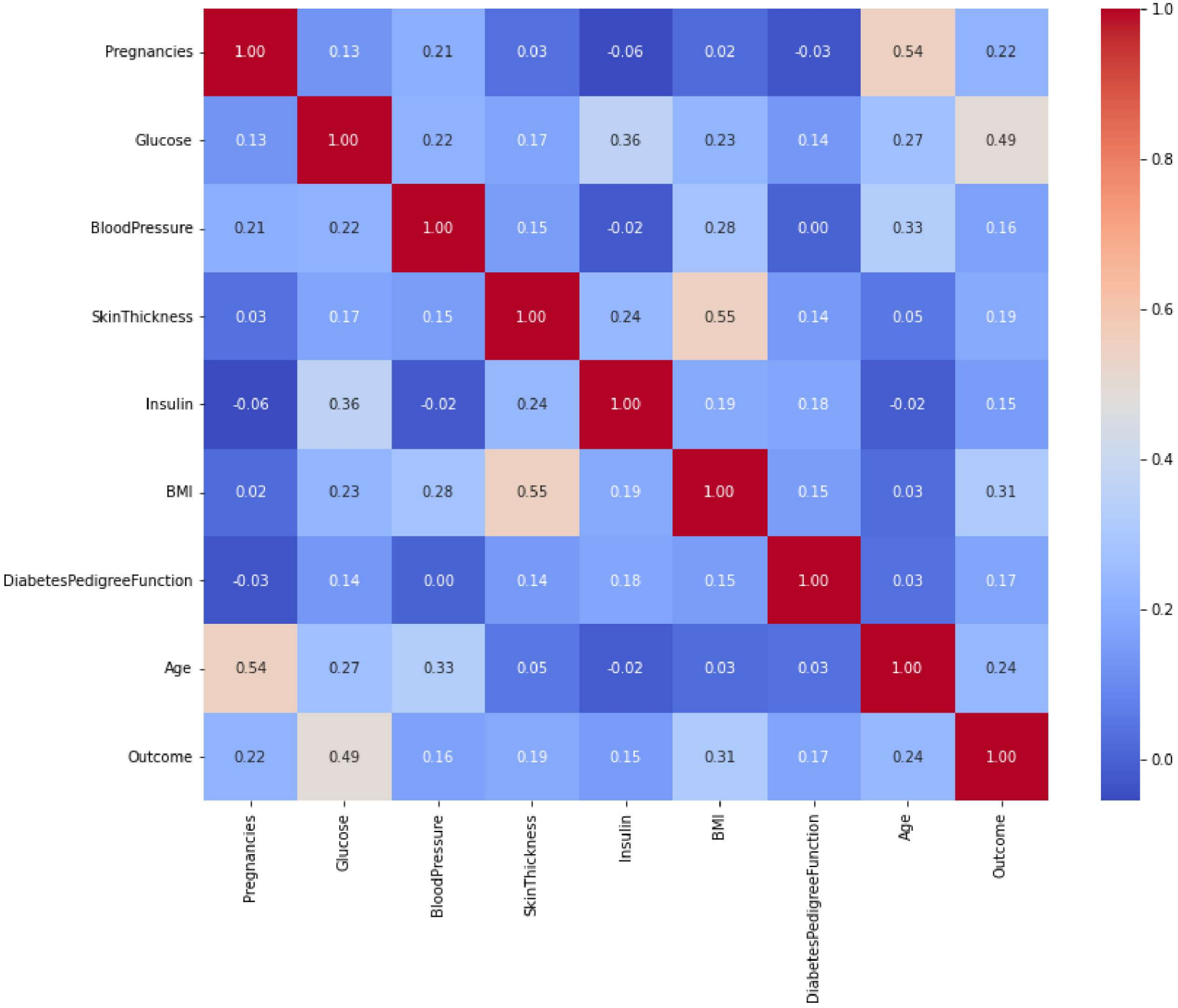
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	121.681605	72.254807	27.334635	94.652344	32.450911	0.471876	33.240885	0.348958
std	3.369578	30.436016	12.115932	9.229014	105.547598	6.875366	0.331329	11.760232	0.476951
min	0.000000	44.000000	24.000000	7.000000	14.000000	18.200000	0.078000	21.000000	0.000000
25%	1.000000	99.750000	64.000000	23.000000	30.500000	27.500000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	31.250000	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 [25]:

```
plt.figure(figsize=(13,10))
sns.heatmap(df.corr(),annot=True, fmt = ".2f", cmap = "coolwarm")
```

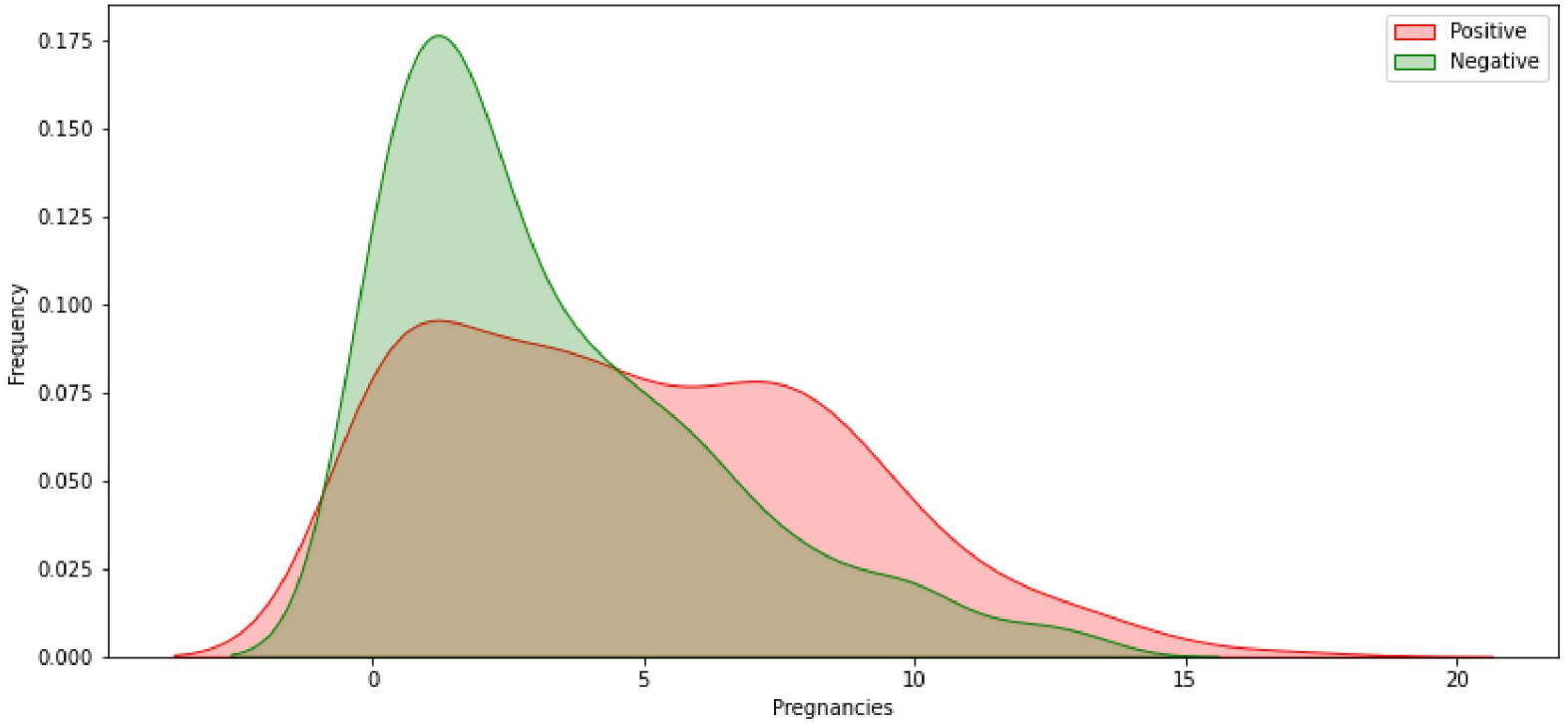
Out[25]:

```
<AxesSubplot:>
```


Pregnancies 1.00 0.13 0.21 0.03 -0.06 0.02 -0.03 0.54 0.22
Glucose 0.13 1.00 0.22 0.17 0.36 0.23 0.14 0.27 0.49
BloodPressure 0.21 0.22 1.00 0.15 -0.02 0.28 0.00 0.33 0.16
SkinThickness 0.03 0.17 0.15 1.00 0.24 0.55 0.14 0.05 0.19
Insulin -0.06 0.36 -0.02 0.24 1.00 0.19 0.18 -0.02 0.15
BMI 0.02 0.23 0.28 0.55 0.19 1.00 0.15 0.03 0.31
DiabetesPedigreeFunction -0.03 0.14 0.00 0.14 0.18 0.15 1.00 0.03 0.17
Age 0.54 0.27 0.33 0.05 -0.02 0.03 0.03 1.00 0.24
Outcome 0.22 0.49 0.16 0.19 0.15 0.31 0.17 0.24 1.00
Pregnancies
Glucose
BloodPressure
SkinThickness
Insulin
BMI
DiabetesPedigreeFunction
Age
Outcome
1.0
0.8
0.6
0.4
0.2
0.0

In [26]:
```
# Explore Pregnancies vs Outcome
plt.figure(figsize=(13,6))
g = sns.kdeplot(df["Pregnancies"][df["Outcome"] == 1],
     color="Red", shade = True)
g = sns.kdeplot(df["Pregnancies"][df["Outcome"] == 0],
     ax =g, color="Green", shade= True)
g.set_xlabel("Pregnancies")
g.set_ylabel("Frequency")
g.legend(["Positive","Negative"])
```

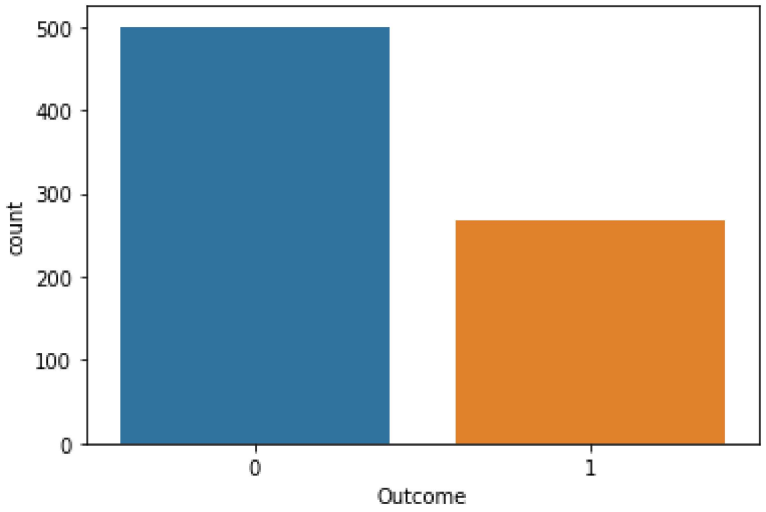
Out[26]:
```
<matplotlib.legend.Legend at 0x1ce42e44d30>
```



In [27]:
```
sns.countplot('Outcome', data = df)
```

```
C:\Users\kbala\AppData\Local\Programs\Python\Python310\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the follow
ing variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other argument
s without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
<AxesSubplot:xlabel='Outcome', ylabel='count'>
```

Out[27]:

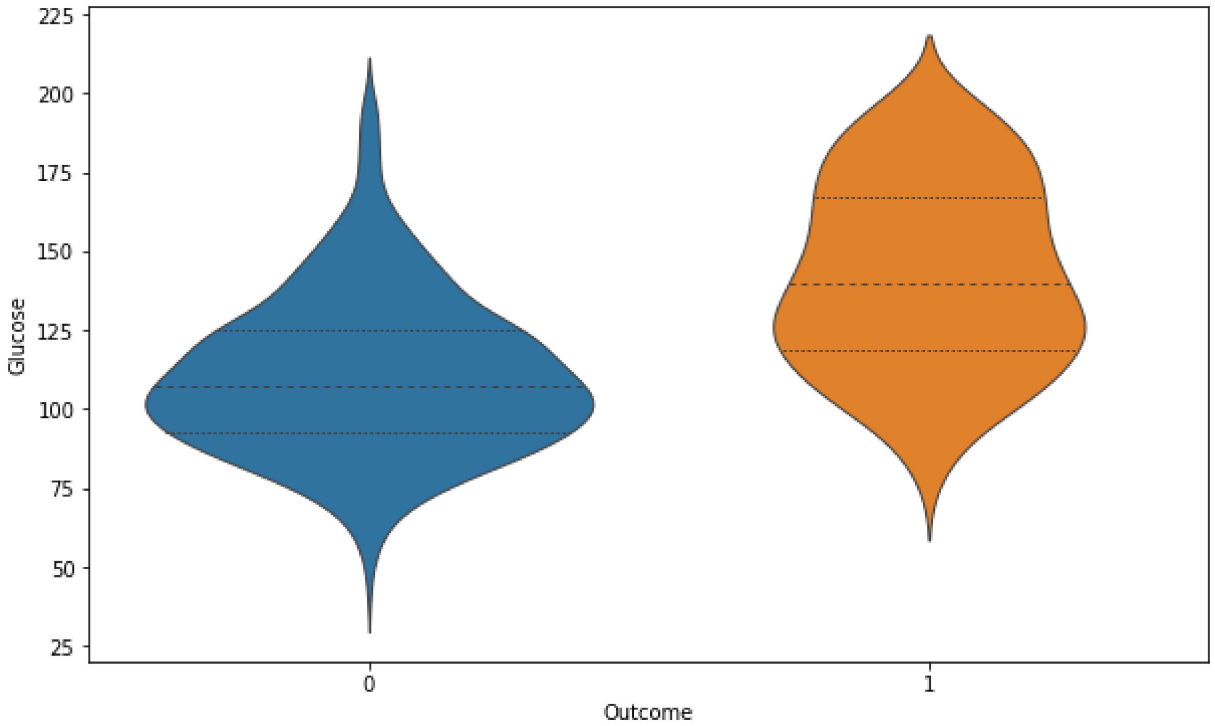


In [28]:

```
# Explore Gluecose vs Outcome
plt.figure(figsize=(10,6))
sns.violinplot(data=df, x="Outcome", y="Glucose",
               split=True, inner="quart", linewidth=1)
```

Out[28]:

```
<AxesSubplot:xlabel='Outcome', ylabel='Glucose'>
```



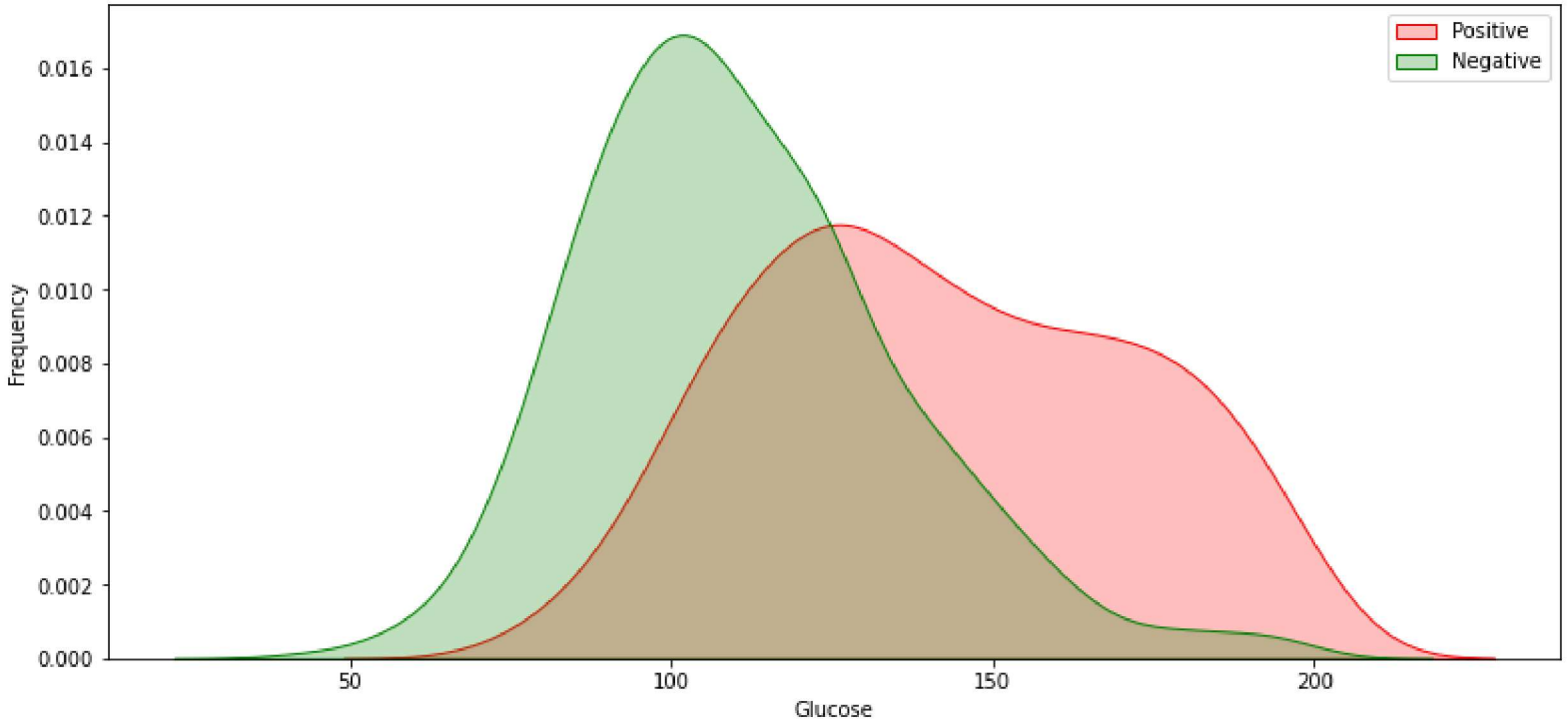
In [29]:

```
# Explore Glucose vs Outcome

plt.figure(figsize=(13,6))
g = sns.kdeplot(df["Glucose"][df["Outcome"] == 1], color="Red", shade = True)
g = sns.kdeplot(df["Glucose"][df["Outcome"] == 0], ax =g, color="Green", shade= True)
g.set_xlabel("Glucose")
g.set_ylabel("Frequency")
g.legend(["Positive","Negative"])
```

Out[29]:

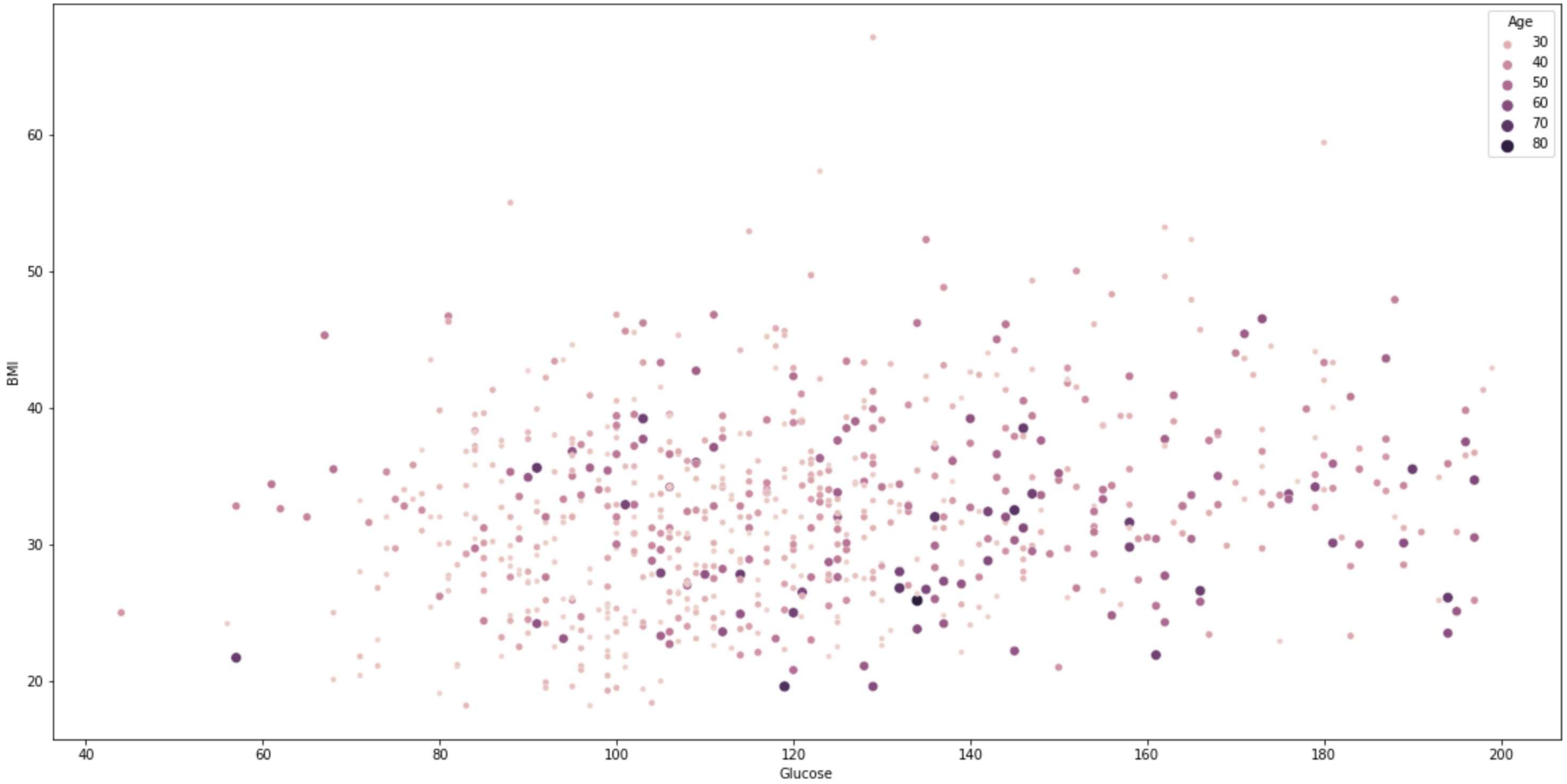
```
<matplotlib.legend.Legend at 0x1ce67147580>
```



In [30]:
```
# Glucose vs BMI vs Age

plt.figure(figsize=(20,10))
sns.scatterplot(data=df, x="Glucose", y="BMI", hue="Age", size="Age")
```

Out[30]:
```
<AxesSubplot:xlabel='Glucose', ylabel='BMI'>
```



In [31]:

```
def detect_outliers(df,n,features):
    outlier_indices = []
    """
    Detect outliers from given list of features. It returns a list of the indices
    according to the observations containing more than n outliers according
    to the Tukey method
    """
    # iterate over features(columns)
    for col in features:
        Q1 = np.percentile(df[col], 25)
        Q3 = np.percentile(df[col],75)
        IQR = Q3 - Q1
```

```
        # outlier step
        outlier_step = 1.5 * IQR

        # Determine a list of indices of outliers for feature col
        outlier_list_col = df[(df[col] < Q1 - outlier_step) | (df[col] > Q3 + outlier_step )].index

        # append the found outlier indices for col to the list of outlier indices
        outlier_indices.extend(outlier_list_col)

    # select observations containing more than 2 outliers
    outlier_indices = Counter(outlier_indices)
    multiple_outliers = list( k for k, v in outlier_indices.items() if v > n )

    return multiple_outliers

# detect outliers from numeric features
outliers_to_drop = detect_outliers(df, 2 ,["Pregnancies", 'Glucose', 'BloodPressure', 'BMI', 'DiabetesPedigreeFunction', 'SkinThic
```

In [32]:

```
# Data Transformation
q  = QuantileTransformer()
X = q.fit_transform(df)
transformedDF = q.transform(X)
transformedDF = pd.DataFrame(X)
transformedDF.columns =['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction',
# Show top 5 rows
transformedDF.head()
```

```
C:\Users\kbala\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\preprocessing\_data.py:2590: UserWarning: n_quant
iles (1000) is greater than the total number of samples (768). n_quantiles is set to n_samples.
  warnings.warn(
C:\Users\kbala\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\base.py:450: UserWarning: X does not have valid f
eature names, but QuantileTransformer was fitted with feature names
  warnings.warn(
```

Out[32]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	0.747718	0.810300	0.516949	0.801825	0.256193	0.591265	0.750978	0.889831	1.0
1	0.232725	0.091265	0.290091	0.644720	0.256193	0.213168	0.475880	0.558670	0.0
2	0.863755	0.956975	0.233377	0.357888	0.256193	0.077575	0.782269	0.585398	1.0
3	0.232725	0.124511	0.290091	0.357888	0.662973	0.284224	0.106258	0.000000	0.0

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
4	0.000000	0.721643	0.005215	0.801825	0.834420	0.926988	0.997392	0.606258	1.0

In [33]:

```
features = df.drop(["Outcome"], axis=1)
labels = df["Outcome"]
x_train, x_test, y_train, y_test = train_test_split(features, labels, test_size=0.30, random_state=7)
```

In [34]:

```
def evaluate_model(models):
    """
    Takes a list of models and returns chart of cross validation scores using mean accuracy
    """

    # Cross validate model with Kfold stratified cross val
    kfold = StratifiedKFold(n_splits = 10)

    result = []
    for model in models :
        result.append(cross_val_score(estimator = model, X = x_train, y = y_train, scoring = "accuracy", cv = kfold, n_jobs=4))

    cv_means = []
    cv_std = []
    for cv_result in result:
        cv_means.append(cv_result.mean())
        cv_std.append(cv_result.std())

    result_df = pd.DataFrame({
        "CrossValMeans":cv_means,
        "CrossValerrors": cv_std,
        "Models":[
            "LogisticRegression",
            "DecisionTreeClassifier",
            "AdaBoostClassifier",
            "SVC",
            "RandomForestClassifier",
            "GradientBoostingClassifier",
            "KNeighborsClassifier"
        ]
    })

    # Generate chart
```

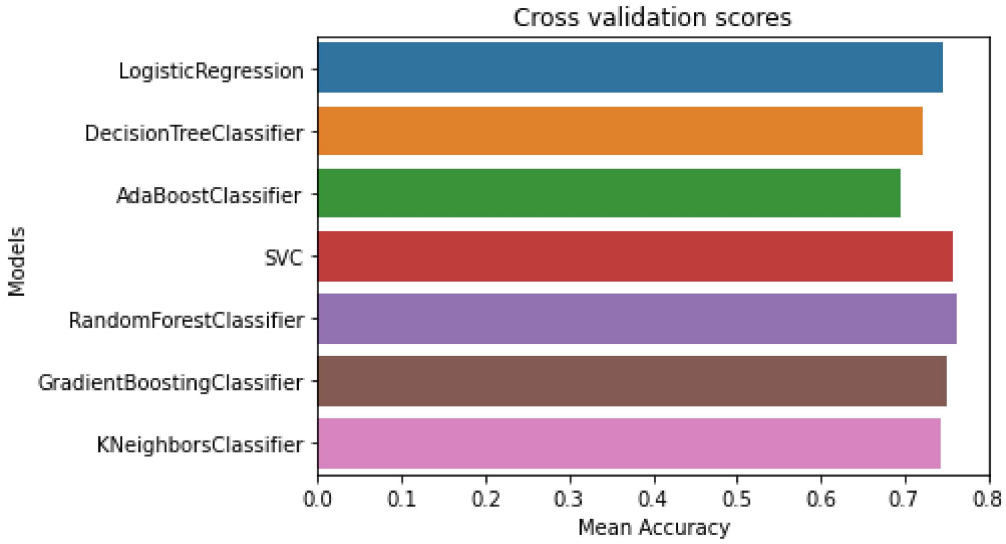
```
    bar = sns.barplot(x = "CrossValMeans", y = "Models", data = result_df, orient = "h")
    bar.set_xlabel("Mean Accuracy")
    bar.set_title("Cross validation scores")
    return result_df
```

In [35]:

```
# Modeling step Test differents algorithms
random_state = 30
models = [
    LogisticRegression(random_state = random_state, solver='liblinear'),
    DecisionTreeClassifier(random_state = random_state),
    AdaBoostClassifier(DecisionTreeClassifier(random_state = random_state), random_state = random_state, learning_rate = 0.2),
    SVC(random_state = random_state),
    RandomForestClassifier(random_state = random_state),
    GradientBoostingClassifier(random_state = random_state),
    KNeighborsClassifier(),
]
evaluate_model(models)
```

Out[35]:

	CrossValMeans	CrossValerrors	Models
0	0.746820	0.062856	LogisticRegression
1	0.722711	0.059852	DecisionTreeClassifier
2	0.696611	0.067270	AdaBoostClassifier
3	0.757966	0.031690	SVC
4	0.763452	0.061238	RandomForestClassifier
5	0.750314	0.083211	GradientBoostingClassifier
6	0.742802	0.053138	KNeighborsClassifier



In [36]:

```
# Import libraries
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report
def analyze_grid_result(grid_result):
    '''
    Analysis of GridCV result and predicting with test dataset
    Show classification report at last
    '''
    # Best parameters and accuracy
    print("Tuned hyperparameters: (best parameters) ", grid_result.best_params_)
    print("Accuracy :", grid_result.best_score_)

    means = grid_result.cv_results_["mean_test_score"]
    stds = grid_result.cv_results_["std_test_score"]
    for mean, std, params in zip(means, stds, grid_result.cv_results_["params"]):
        print("%0.3f (+/-%0.03f) for %r" % (mean, std * 2, params))
    print()
    print("Detailed classification report:")
    y_true, y_pred = y_test, grid_result.predict(x_test)
    print(classification_report(y_true, y_pred))
    print()
```

In [37]:

```
# Define models and parameters for LogisticRegression
```

```
model = LogisticRegression(solver='liblinear')
solvers = ['newton-cg', 'liblinear']
penalty = ['l2']
c_values = [100, 10, 1.0, 0.1, 0.01]
# Define grid search
grid = dict(solver = solvers, penalty = penalty, C = c_values)
cv = StratifiedKFold(n_splits = 50, random_state = 1, shuffle = True)
grid_search = GridSearchCV(estimator = model, param_grid = grid, cv = cv, scoring = 'accuracy', error_score = 0)
logi_result = grid_search.fit(x_train, y_train)
# Logistic Regression Hyperparameter Result
analyze_grid_result(logi_result)
```

```
Tuned hyperparameters: (best parameters)  {'C': 10, 'penalty': 'l2', 'solver': 'liblinear'}
Accuracy : 0.7727272727272728
0.769 (+/-0.253) for {'C': 100, 'penalty': 'l2', 'solver': 'newton-cg'}
0.767 (+/-0.250) for {'C': 100, 'penalty': 'l2', 'solver': 'liblinear'}
0.771 (+/-0.252) for {'C': 10, 'penalty': 'l2', 'solver': 'newton-cg'}
0.773 (+/-0.252) for {'C': 10, 'penalty': 'l2', 'solver': 'liblinear'}
0.773 (+/-0.255) for {'C': 1.0, 'penalty': 'l2', 'solver': 'newton-cg'}
0.750 (+/-0.250) for {'C': 1.0, 'penalty': 'l2', 'solver': 'liblinear'}
0.767 (+/-0.233) for {'C': 0.1, 'penalty': 'l2', 'solver': 'newton-cg'}
0.708 (+/-0.246) for {'C': 0.1, 'penalty': 'l2', 'solver': 'liblinear'}
0.764 (+/-0.235) for {'C': 0.01, 'penalty': 'l2', 'solver': 'newton-cg'}
0.693 (+/-0.271) for {'C': 0.01, 'penalty': 'l2', 'solver': 'liblinear'}

Detailed classification report:
              precision    recall  f1-score   support

           0       0.79      0.88      0.83       147
           1       0.73      0.58      0.65        84

    accuracy                           0.77       231
   macro avg       0.76      0.73      0.74       231
weighted avg       0.77      0.77      0.76       231
```

In [38]:

```
# Define models and parameters for LogisticRegression
model = SVC()
# Define grid search
tuned_parameters = [
    {"kernel": ["rbf"], "gamma": [1e-3, 1e-4], "C": [1, 10, 100, 1000]},
    {"kernel": ["linear"], "C": [1, 10, 100, 1000]},
```

```
]
cv = StratifiedKFold(n_splits = 2, random_state = 1, shuffle = True)
grid_search = GridSearchCV(estimator = model, param_grid = tuned_parameters, cv = cv, scoring = 'accuracy', error_score = 0)
scv_result = grid_search.fit(x_train, y_train)
# SVC Hyperparameter Result
analyze_grid_result(scv_result)
```

```
Tuned hyperparameters: (best parameters)  {'C': 100, 'kernel': 'linear'}
Accuracy : 0.7746768018642846
0.726 (+/-0.005) for {'C': 1, 'gamma': 0.001, 'kernel': 'rbf'}
0.752 (+/-0.034) for {'C': 1, 'gamma': 0.0001, 'kernel': 'rbf'}
0.708 (+/-0.003) for {'C': 10, 'gamma': 0.001, 'kernel': 'rbf'}
0.728 (+/-0.001) for {'C': 10, 'gamma': 0.0001, 'kernel': 'rbf'}
0.700 (+/-0.010) for {'C': 100, 'gamma': 0.001, 'kernel': 'rbf'}
0.728 (+/-0.051) for {'C': 100, 'gamma': 0.0001, 'kernel': 'rbf'}
0.700 (+/-0.012) for {'C': 1000, 'gamma': 0.001, 'kernel': 'rbf'}
0.713 (+/-0.021) for {'C': 1000, 'gamma': 0.0001, 'kernel': 'rbf'}
0.762 (+/-0.008) for {'C': 1, 'kernel': 'linear'}
0.760 (+/-0.020) for {'C': 10, 'kernel': 'linear'}
0.775 (+/-0.003) for {'C': 100, 'kernel': 'linear'}
0.758 (+/-0.029) for {'C': 1000, 'kernel': 'linear'}

Detailed classification report:
              precision    recall  f1-score   support

           0       0.80      0.83      0.81       147
           1       0.68      0.63      0.65        84

    accuracy                           0.76       231
   macro avg       0.74      0.73      0.73       231
weighted avg       0.75      0.76      0.76       231
```

In [39]:

```
# Define models and parameters for LogisticRegression
model = RandomForestClassifier(random_state=42)
# Define grid search
tuned_parameters = {
    'n_estimators': [200, 500],
    'max_features': ['auto', 'sqrt', 'log2'],
    'max_depth' : [4,5,6,7,8],
    'criterion' :['gini', 'entropy']
}
```

```python
cv = StratifiedKFold(n_splits = 2, random_state = 1, shuffle = True)
grid_search = GridSearchCV(estimator = model, param_grid = tuned_parameters, cv = cv, scoring = 'accuracy', error_score = 0)
grid_result = grid_search.fit(x_train, y_train)
# SVC Hyperparameter Result
analyze_grid_result(grid_result)
```

```
Tuned hyperparameters: (best parameters)  {'criterion': 'gini', 'max_depth': 5, 'max_features': 'auto', 'n_estimators': 500}
Accuracy : 0.7746559951173501
0.767 (+/-0.012) for {'criterion': 'gini', 'max_depth': 4, 'max_features': 'auto', 'n_estimators': 200}
0.775 (+/-0.027) for {'criterion': 'gini', 'max_depth': 4, 'max_features': 'auto', 'n_estimators': 500}
0.767 (+/-0.012) for {'criterion': 'gini', 'max_depth': 4, 'max_features': 'sqrt', 'n_estimators': 200}
0.775 (+/-0.027) for {'criterion': 'gini', 'max_depth': 4, 'max_features': 'sqrt', 'n_estimators': 500}
0.769 (+/-0.023) for {'criterion': 'gini', 'max_depth': 4, 'max_features': 'log2', 'n_estimators': 200}
0.767 (+/-0.034) for {'criterion': 'gini', 'max_depth': 4, 'max_features': 'log2', 'n_estimators': 500}
0.765 (+/-0.016) for {'criterion': 'gini', 'max_depth': 5, 'max_features': 'auto', 'n_estimators': 200}
0.775 (+/-0.019) for {'criterion': 'gini', 'max_depth': 5, 'max_features': 'auto', 'n_estimators': 500}
0.765 (+/-0.016) for {'criterion': 'gini', 'max_depth': 5, 'max_features': 'sqrt', 'n_estimators': 200}
0.775 (+/-0.019) for {'criterion': 'gini', 'max_depth': 5, 'max_features': 'sqrt', 'n_estimators': 500}
0.765 (+/-0.038) for {'criterion': 'gini', 'max_depth': 5, 'max_features': 'log2', 'n_estimators': 200}
0.762 (+/-0.031) for {'criterion': 'gini', 'max_depth': 5, 'max_features': 'log2', 'n_estimators': 500}
0.762 (+/-0.016) for {'criterion': 'gini', 'max_depth': 6, 'max_features': 'auto', 'n_estimators': 200}
0.765 (+/-0.023) for {'criterion': 'gini', 'max_depth': 6, 'max_features': 'auto', 'n_estimators': 500}
0.762 (+/-0.016) for {'criterion': 'gini', 'max_depth': 6, 'max_features': 'sqrt', 'n_estimators': 200}
0.765 (+/-0.023) for {'criterion': 'gini', 'max_depth': 6, 'max_features': 'sqrt', 'n_estimators': 500}
0.763 (+/-0.012) for {'criterion': 'gini', 'max_depth': 6, 'max_features': 'log2', 'n_estimators': 200}
0.760 (+/-0.005) for {'criterion': 'gini', 'max_depth': 6, 'max_features': 'log2', 'n_estimators': 500}
0.763 (+/-0.027) for {'criterion': 'gini', 'max_depth': 7, 'max_features': 'auto', 'n_estimators': 200}
0.767 (+/-0.034) for {'criterion': 'gini', 'max_depth': 7, 'max_features': 'auto', 'n_estimators': 500}
0.763 (+/-0.027) for {'criterion': 'gini', 'max_depth': 7, 'max_features': 'sqrt', 'n_estimators': 200}
0.767 (+/-0.034) for {'criterion': 'gini', 'max_depth': 7, 'max_features': 'sqrt', 'n_estimators': 500}
0.767 (+/-0.027) for {'criterion': 'gini', 'max_depth': 7, 'max_features': 'log2', 'n_estimators': 200}
0.769 (+/-0.031) for {'criterion': 'gini', 'max_depth': 7, 'max_features': 'log2', 'n_estimators': 500}
0.763 (+/-0.034) for {'criterion': 'gini', 'max_depth': 8, 'max_features': 'auto', 'n_estimators': 200}
0.760 (+/-0.034) for {'criterion': 'gini', 'max_depth': 8, 'max_features': 'auto', 'n_estimators': 500}
0.763 (+/-0.034) for {'criterion': 'gini', 'max_depth': 8, 'max_features': 'sqrt', 'n_estimators': 200}
0.760 (+/-0.034) for {'criterion': 'gini', 'max_depth': 8, 'max_features': 'sqrt', 'n_estimators': 500}
0.763 (+/-0.020) for {'criterion': 'gini', 'max_depth': 8, 'max_features': 'log2', 'n_estimators': 200}
0.767 (+/-0.019) for {'criterion': 'gini', 'max_depth': 8, 'max_features': 'log2', 'n_estimators': 500}
0.773 (+/-0.023) for {'criterion': 'entropy', 'max_depth': 4, 'max_features': 'auto', 'n_estimators': 200}
0.767 (+/-0.019) for {'criterion': 'entropy', 'max_depth': 4, 'max_features': 'auto', 'n_estimators': 500}
0.773 (+/-0.023) for {'criterion': 'entropy', 'max_depth': 4, 'max_features': 'sqrt', 'n_estimators': 200}
0.767 (+/-0.019) for {'criterion': 'entropy', 'max_depth': 4, 'max_features': 'sqrt', 'n_estimators': 500}
0.771 (+/-0.027) for {'criterion': 'entropy', 'max_depth': 4, 'max_features': 'log2', 'n_estimators': 200}
```

```
0.771 (+/-0.019) for {'criterion': 'entropy', 'max_depth': 4, 'max_features': 'log2', 'n_estimators': 500}
0.765 (+/-0.038) for {'criterion': 'entropy', 'max_depth': 5, 'max_features': 'auto', 'n_estimators': 200}
0.765 (+/-0.016) for {'criterion': 'entropy', 'max_depth': 5, 'max_features': 'auto', 'n_estimators': 500}
0.765 (+/-0.038) for {'criterion': 'entropy', 'max_depth': 5, 'max_features': 'sqrt', 'n_estimators': 200}
0.765 (+/-0.016) for {'criterion': 'entropy', 'max_depth': 5, 'max_features': 'sqrt', 'n_estimators': 500}
0.769 (+/-0.031) for {'criterion': 'entropy', 'max_depth': 5, 'max_features': 'log2', 'n_estimators': 200}
0.763 (+/-0.034) for {'criterion': 'entropy', 'max_depth': 5, 'max_features': 'log2', 'n_estimators': 500}
0.765 (+/-0.023) for {'criterion': 'entropy', 'max_depth': 6, 'max_features': 'auto', 'n_estimators': 200}
0.771 (+/-0.034) for {'criterion': 'entropy', 'max_depth': 6, 'max_features': 'auto', 'n_estimators': 500}
0.765 (+/-0.023) for {'criterion': 'entropy', 'max_depth': 6, 'max_features': 'sqrt', 'n_estimators': 200}
0.771 (+/-0.034) for {'criterion': 'entropy', 'max_depth': 6, 'max_features': 'sqrt', 'n_estimators': 500}
0.771 (+/-0.027) for {'criterion': 'entropy', 'max_depth': 6, 'max_features': 'log2', 'n_estimators': 200}
0.771 (+/-0.027) for {'criterion': 'entropy', 'max_depth': 6, 'max_features': 'log2', 'n_estimators': 500}
0.767 (+/-0.034) for {'criterion': 'entropy', 'max_depth': 7, 'max_features': 'auto', 'n_estimators': 200}
0.767 (+/-0.034) for {'criterion': 'entropy', 'max_depth': 7, 'max_features': 'auto', 'n_estimators': 500}
0.767 (+/-0.034) for {'criterion': 'entropy', 'max_depth': 7, 'max_features': 'sqrt', 'n_estimators': 200}
0.767 (+/-0.034) for {'criterion': 'entropy', 'max_depth': 7, 'max_features': 'sqrt', 'n_estimators': 500}
0.762 (+/-0.031) for {'criterion': 'entropy', 'max_depth': 7, 'max_features': 'log2', 'n_estimators': 200}
0.756 (+/-0.027) for {'criterion': 'entropy', 'max_depth': 7, 'max_features': 'log2', 'n_estimators': 500}
0.752 (+/-0.020) for {'criterion': 'entropy', 'max_depth': 8, 'max_features': 'auto', 'n_estimators': 200}
0.756 (+/-0.020) for {'criterion': 'entropy', 'max_depth': 8, 'max_features': 'auto', 'n_estimators': 500}
0.752 (+/-0.020) for {'criterion': 'entropy', 'max_depth': 8, 'max_features': 'sqrt', 'n_estimators': 200}
0.756 (+/-0.020) for {'criterion': 'entropy', 'max_depth': 8, 'max_features': 'sqrt', 'n_estimators': 500}
0.754 (+/-0.031) for {'criterion': 'entropy', 'max_depth': 8, 'max_features': 'log2', 'n_estimators': 200}
0.758 (+/-0.031) for {'criterion': 'entropy', 'max_depth': 8, 'max_features': 'log2', 'n_estimators': 500}

Detailed classification report:
              precision    recall  f1-score   support

           0       0.78      0.84      0.81       147
           1       0.68      0.60      0.64        84

    accuracy                           0.75       231
   macro avg       0.73      0.72      0.73       231
weighted avg       0.75      0.75      0.75       231
```

In [44]:

```
from sklearn.metrics import confusion_matrix
confusion_matrix = confusion_matrix(y_test, y_pred)
print(confusion_matrix)
```

```
[[129  18]
```

```
[ 35  49]]
```

In [45]:
```
# true positive = 129
# true negative = 18
# false positive = 35
# false negative = 49
```

In [42]:
```
# Test predictions
y_pred = logi_result.predict(x_test)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.79      0.88      0.83       147
           1       0.73      0.58      0.65        84

    accuracy                           0.77       231
   macro avg       0.76      0.73      0.74       231
weighted avg       0.77      0.77      0.76       231
```

In [43]:
```
x_test['pred'] = y_pred
print(x_test)
```

```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
353            1     90.0           62.0             12     43.0  27.2
236            7    181.0           84.0             21    192.0  35.9
323           13    152.0           90.0             33     29.0  26.8
98             6     93.0           50.0             30     64.0  28.7
701            6    125.0           78.0             31     30.5  27.6
..           ...      ...            ...            ...      ...   ...
188            8    109.0           76.0             39    114.0  27.9
351            4    137.0           84.0             23     30.5  31.2
120            0    162.0           76.0             56    100.0  53.2
108            3     83.0           58.0             31     18.0  34.3
616            6    117.0           96.0             23     30.5  28.7

     DiabetesPedigreeFunction  Age  pred
353                     0.580   24     0
236                     0.586   51     1
323                     0.731   43     1
```

```
98                    0.356   23     0
701                   0.565   49     0
..                      ...   ...   ...
188                   0.640   31     0
351                   0.252   30     0
120                   0.759   25     1
108                   0.336   25     0
616                   0.157   30     0

[231 rows x 9 columns]
```

Done by

Name: Balaji subrahmanyam K

November batch

Data science November mini project (with python)

Dataset used https://www.kaggle.com/uciml/pima-indians-diabetes-database

Create a machine learning model to predict that the person has diabetes or not.