In [1]:

```
# Import libraries
import numpy as np # for linear algebra
import pandas as pd # for 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 data visualization charts
from collections import Counter
import os

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

In [4]:

```
# Importing the dataset from Kaggle
data = pd.read_csv("diabetes.csv")

# First step is getting familiar with the structure of the dataset
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   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]:

```
data.head()
```

Out[5]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunc
0	6	148	72	35	0	33.6	0
1	1	85	66	29	0	26.6	0
2	8	183	64	0	0	23.3	0
3	1	89	66	23	94	28.1	0
4	0	137	40	35	168	43.1	2

# Filling the Missing Values

In [6]:

```
data.isnull().sum()
```

Out[6]:

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

In [7]:

```
data['Glucose'] = data['Glucose'].replace(0, data['Glucose'].median())

# Filling 0 values of Blood Pressure
data['BloodPressure'] = data['BloodPressure'].replace(0, data['BloodPressure'].median()

# Replacing 0 values in BMI
data['BMI'] = data['BMI'].replace(0, data['BMI'].mean())

# Replacing the missing values of Insulin and SkinThickness
data['SkinThickness'] = data['SkinThickness'].replace(0, data['SkinThickness'].mean())
data['Insulin'] = data['Insulin'].replace(0, data['Insulin'].mean())
data.head()
```

Out[7]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigree
0	6	148	72	35.000000	79.799479	33.6	
1	1	85	66	29.000000	79.799479	26.6	
2	8	183	64	20.536458	79.799479	23.3	
3	1	89	66	23.000000	94.000000	28.1	
4	0	137	40	35.000000	168.000000	43.1	

In [8]:

```
data.describe()
```

Out[8]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Dia
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	
mean	3.845052	121.656250	72.386719	26.606479	118.660163	32.450805	
std	3.369578	30.438286	12.096642	9.631241	93.080358	6.875374	
min	0.000000	44.000000	24.000000	7.000000	14.000000	18.200000	
25%	1.000000	99.750000	64.000000	20.536458	79.799479	27.500000	
50%	3.000000	117.000000	72.000000	23.000000	79.799479	32.000000	
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	

In [ ]:

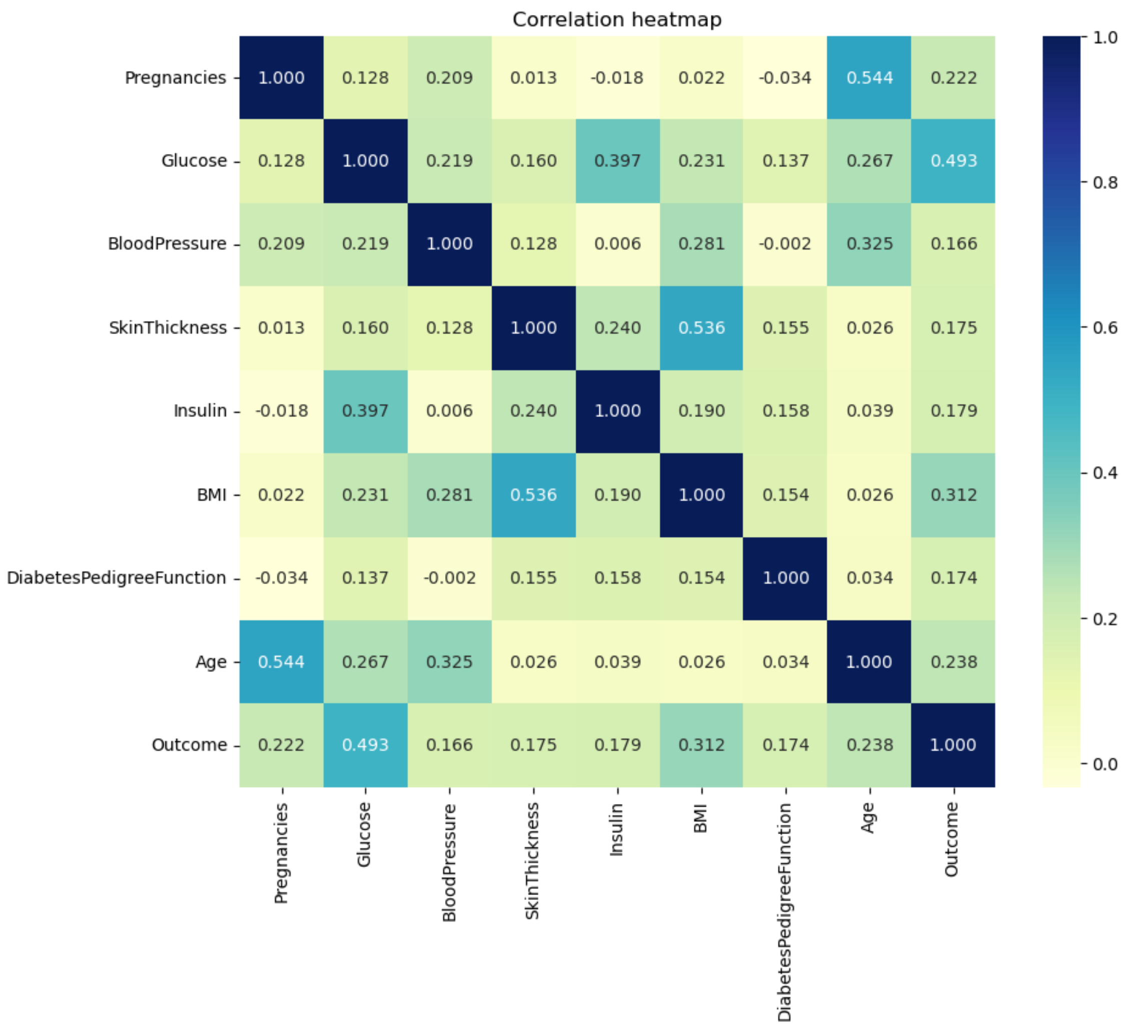
```
#Exploratory Data Analysis
```

In [9]:

```
plt.figure(figsize = (10, 8))
sns.heatmap(data.corr(), annot = True, fmt = ".3f", cmap = "YlGnBu")
plt.title("Correlation heatmap")
```

Out[9]:

```
Text(0.5, 1.0, 'Correlation heatmap')
```



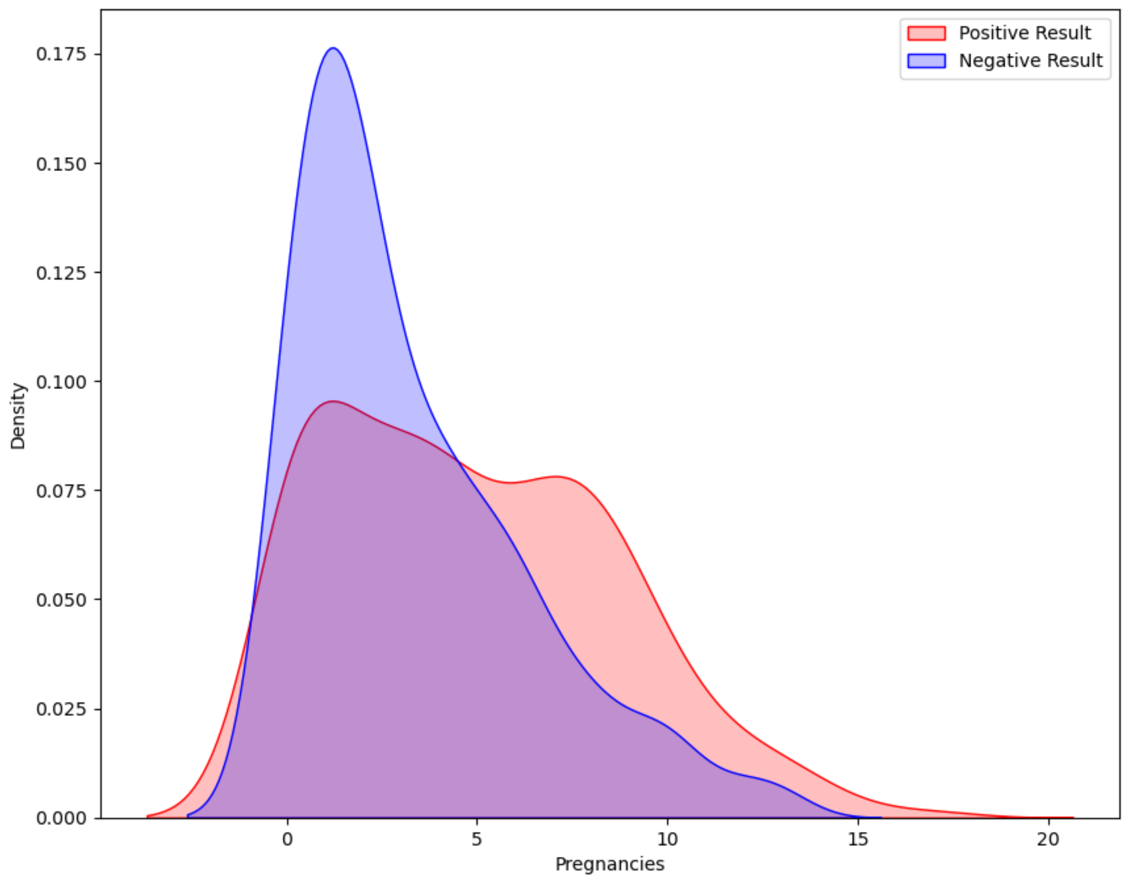
In [10]:

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

# Plotting density function graph of the pregnancies and the target variable
kde = sns.kdeplot(data["Pregnancies"][data["Outcome"] == 1], color = "Red", shade = True
kde = sns.kdeplot(data["Pregnancies"][data["Outcome"] == 0], ax = kde, color = "Blue", s
kde.set_xlabel("Pregnancies")
kde.set_ylabel("Density")
kde.legend(["Positive Result", "Negative Result"])
```

```
C:\Users\venut\AppData\Local\Temp\ipykernel_16960\827401410.py:4: FutureW
arning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  kde = sns.kdeplot(data["Pregnancies"][data["Outcome"] == 1], color = "R
ed", shade = True)
C:\Users\venut\AppData\Local\Temp\ipykernel_16960\827401410.py:5: FutureW
arning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  kde = sns.kdeplot(data["Pregnancies"][data["Outcome"] == 0], ax = kde,
color = "Blue", shade= True)
```

Out[10]:

```
<matplotlib.legend.Legend at 0x273f0fbbd60>
```

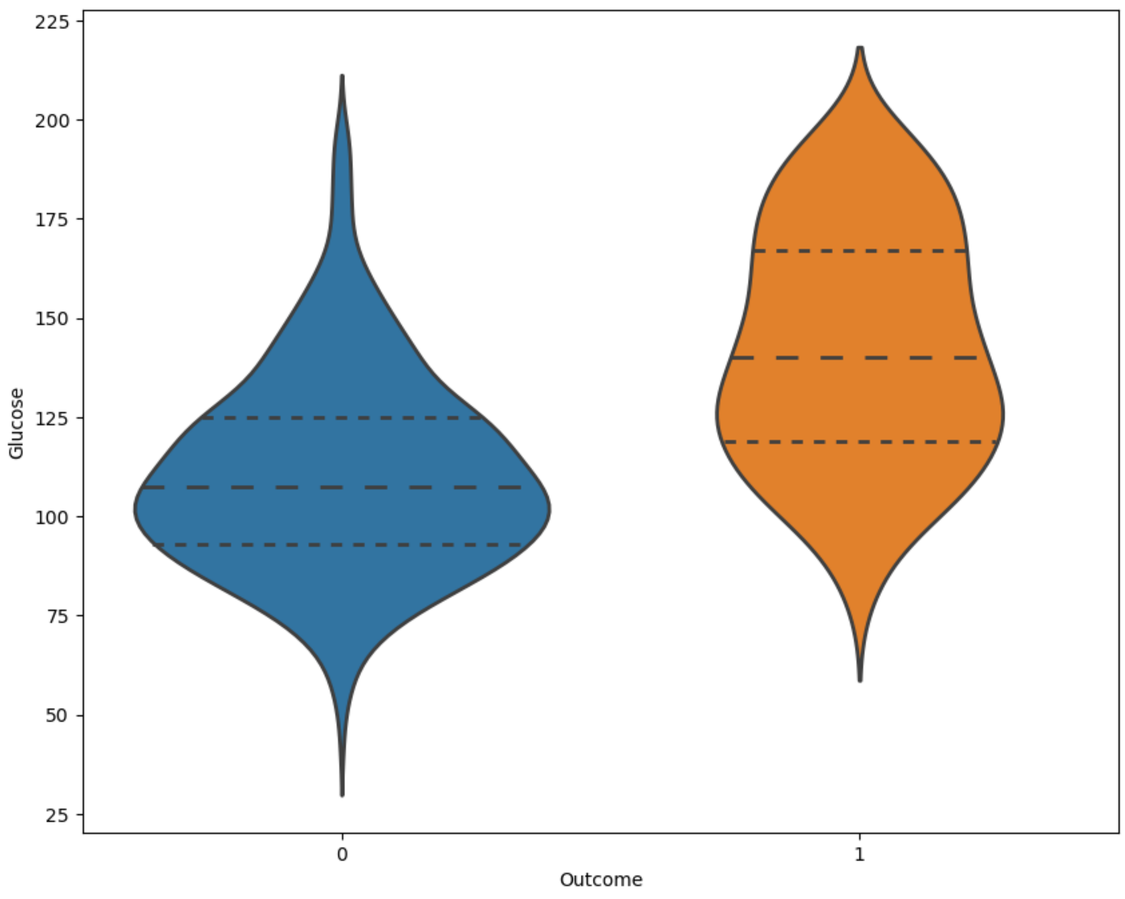


In [11]:

```
# Exploring the Glucose and the Target variables together
plt.figure(figsize = (10, 8))
sns.violinplot(data = data, x = "Outcome", y = "Glucose",
               split = True, inner = "quart", linewidth = 2)
```

Out[11]:

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



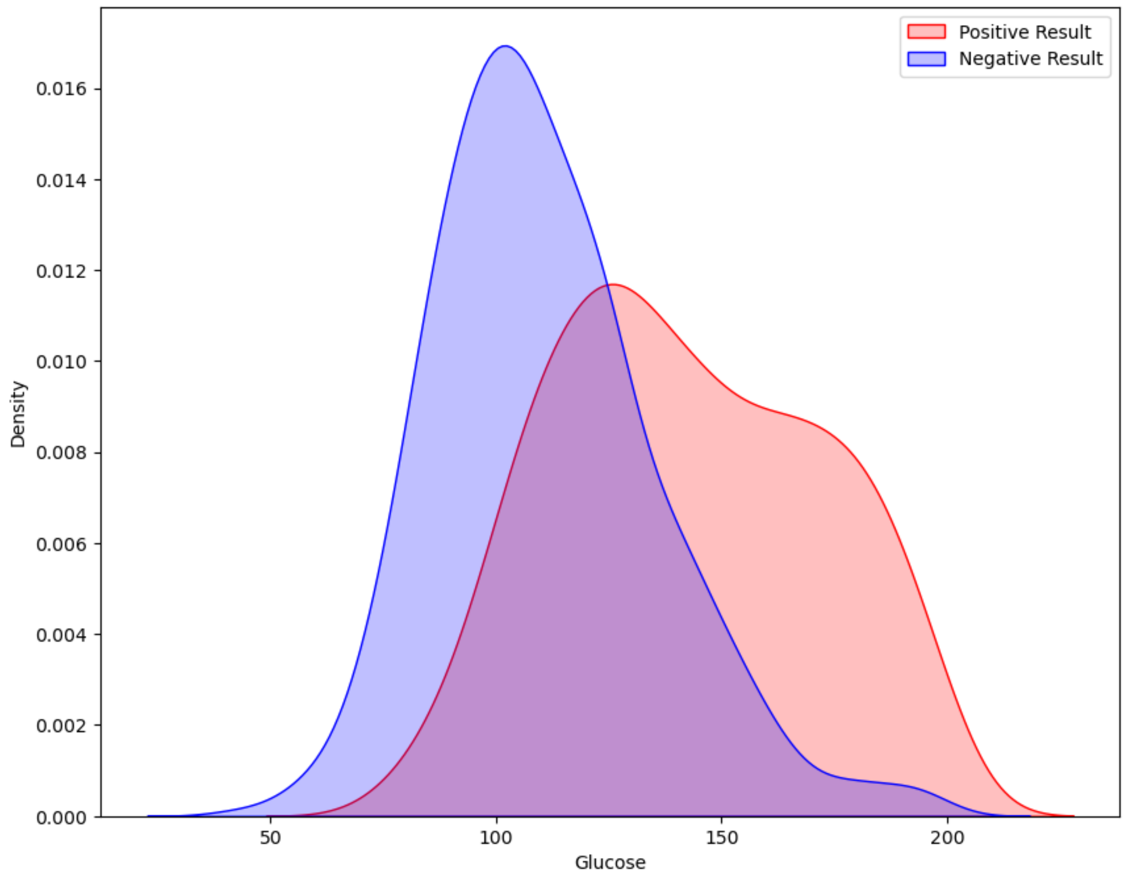
In [12]:

```
plt.figure(figsize = (10, 8))
kde = sns.kdeplot(data["Glucose"][data["Outcome"] == 1], color = "Red", shade = True)
kde = sns.kdeplot(data["Glucose"][data["Outcome"] == 0], ax = kde, color = "Blue", shade
kde.set_xlabel("Glucose")
kde.set_ylabel("Density")
kde.legend(["Positive Result","Negative Result"])
```

```
C:\Users\venut\AppData\Local\Temp\ipykernel_16960\2458365466.py:2: Future
Warning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  kde = sns.kdeplot(data["Glucose"][data["Outcome"] == 1], color = "Red",
shade = True)
C:\Users\venut\AppData\Local\Temp\ipykernel_16960\2458365466.py:3: Future
Warning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  kde = sns.kdeplot(data["Glucose"][data["Outcome"] == 0], ax = kde, colo
r = "Blue", shade= True)
```

Out[12]:

```
<matplotlib.legend.Legend at 0x273f0600be0>
```



In [ ]:

```
#Implementing Machine Learning Models
```

In [13]:

```
# Transforming the data into quartiles
quartile  = QuantileTransformer()
X = quartile.fit_transform(data)
dataset = quartile.transform(X)
dataset = pd.DataFrame(X)
dataset.columns =['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
# Showing the top 5 rows of the transformed dataset
dataset.head()
```

```
C:\Users\venut\anaconda3\lib\site-packages\sklearn\preprocessing\_data.p
y:2627: UserWarning: n_quantiles (1000) is greater than the total number
of samples (768). n_quantiles is set to n_samples.
  warnings.warn(
C:\Users\venut\anaconda3\lib\site-packages\sklearn\base.py:420: UserWarni
ng: X does not have valid feature names, but QuantileTransformer was fitt
ed with feature names
  warnings.warn(
```

Out[13]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigı
0	0.747718	0.810300	0.494133	0.801825	0.380052	0.591265	
1	0.232725	0.091265	0.290091	0.644720	0.380052	0.213168	
2	0.863755	0.956975	0.233377	0.308996	0.380052	0.077575	
3	0.232725	0.124511	0.290091	0.505867	0.662973	0.284224	
4	0.000000	0.721643	0.005215	0.801825	0.834420	0.926988	

In [ ]:

```
#Data Splitting
```

In [14]:

```
# Splitting the dependent and independent features
X = data.drop(["Outcome"], axis = 1)
Y = data["Outcome"]

# Splitting the dataset into the training and testing dataset
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.40, random_state

# Printing the size of the training and testing dataset
print("The size of the training dataset: ", X_train.size)
print("The size of the testing dataset: ", X_test.size)
```

```
The size of the training dataset:  3680
The size of the testing dataset:  2464
```

In [17]:

```
# Python program to create a function to validate models

def cv_model(models):
    """
    We will create a list of machine learning models and print graphs of cross-validatic
    """

    # Cross validating the model using the Kfold stratified cross-validation method
    k_fold = StratifiedKFold(n_splits = 15)

    r = []
    for m in models :
        r.append(cross_val_score(estimator = m, X = X_train, y = Y_train, scoring = "acc

    cross_val_means = []
    cross_val_std = []
    for result in r:
        cross_val_means.append(result.mean())
        cross_val_std.append(result.std())

    df_result = pd.DataFrame({
        "CrossValMean": cross_val_means,
        "CrossValStd": cross_val_std,
        "Model List":[
            "DecisionTreeClassifier",
            "LogisticRegression",
            "SVC",
            "AdaBoostClassifier",
            "GradientBoostingClassifier",
            "RandomForestClassifier",
            "KNeighborsClassifier"
        ]
    })

    # Generating the graph of cross-validation scores
    bar_plot = sns.barplot(x = cross_val_means, y = df_result["Model List"].values, data
    bar_plot.set_xlabel("Mean of Cross Validation Accuracy Scores")
    bar_plot.set_title("Cross Validation Scores of Models")
    return df_result
```

In [19]:

```
 # Importing the required libraries
from sklearn.metrics import classification_report
from sklearn.model_selection import GridSearchCV

# Defining a function to analyse the grid results
def analyze_grid(grid):
    '''''
    Analyzing the results of GridCV method and making predictions for the test data
    Presenting the classification report at the end
    '''
    # Printing the best parameter and accuracy score
    print("Tuned hyperparameters: ", grid.best_params_)
    print("Accuracy Score:", grid.best_score_)

    mean_values = grid.cv_results_["mean_test_score"]
    std_values = grid.cv_results_["std_test_score"]
    for m, s, p in zip(mean_values, std_values, grid.cv_results_["params"]):
      print(f"Mean: {m}, Std: {s} * 2, Params: {p}")
      print("The classification Report:")
    Y_true, Y_pred = Y_test, grid.predict(X_test)
    print(classification_report(Y_true, Y_pred))
    print()
```

In [21]:

```
# Defining the Logistic Regression model and its parameters
model = LogisticRegression(solver ='liblinear')
solver_list = ['liblinear']
penalty_type = ['l2']
c_values = [200, 100, 10, 1.0, 0.01]

# Defining the grid search
grid_lr = dict(solver = solver_list, penalty = penalty_type, C = c_values)
cross_val = StratifiedKFold(n_splits = 100, random_state = 10, shuffle = True)
grid_search_cv = GridSearchCV(estimator = model, param_grid = grid_lr, cv = cross_val, s
lr_result = grid_search_cv.fit(X_train, Y_train)

# Result of Hyper Parameters of Logistic Regression
analyze_grid(lr_result)
```

```
Tuned hyperparameters:  {'C': 200, 'penalty': 'l2', 'solver': 'liblinea
r'}
Accuracy Score: 0.7715000000000001
Mean: 0.7715000000000001, Std: 0.16556796187668676 * 2, Params: {'C': 20
0, 'penalty': 'l2', 'solver': 'liblinear'}
The classification Report:
Mean: 0.7715000000000001, Std: 0.16556796187668676 * 2, Params: {'C': 10
0, 'penalty': 'l2', 'solver': 'liblinear'}
The classification Report:
Mean: 0.7675, Std: 0.16961353129983467 * 2, Params: {'C': 10, 'penalty':
'l2', 'solver': 'liblinear'}
The classification Report:
Mean: 0.7675, Std: 0.17224619008848932 * 2, Params: {'C': 1.0, 'penalty':
'l2', 'solver': 'liblinear'}
The classification Report:
Mean: 0.711, Std: 0.1888888562091475 * 2, Params: {'C': 0.01, 'penalty':
'l2', 'solver': 'liblinear'}
The classification Report:
              precision    recall  f1-score   support

           0       0.78      0.88      0.83       201
           1       0.70      0.53      0.61       107

    accuracy                           0.76       308
   macro avg       0.74      0.71      0.72       308
weighted avg       0.75      0.76      0.75       308
```

In [22]:

```
Y_pred = lr_result.predict(X_test)
print(classification_report(Y_test, Y_pred))
```

```
              precision    recall  f1-score   support

           0       0.78      0.88      0.83       201
           1       0.70      0.53      0.61       107

    accuracy                           0.76       308
   macro avg       0.74      0.71      0.72       308
weighted avg       0.75      0.76      0.75       308
```

In [23]:

```
X_test['predictions'] = Y_pred
print(X_test)
```

```
     Pregnancies  Glucose  BloodPressure  SkinThickness     Insulin   BMI
\
568            4      154             72      29.000000  126.000000  31.3
620            2      112             86      42.000000  160.000000  38.4
456            1      135             54      20.536458   79.799479  26.7
197            3      107             62      13.000000   48.000000  22.9
714            3      102             74      20.536458   79.799479  29.5
..           ...      ...            ...            ...         ...   ...
70             2      100             66      20.000000   90.000000  32.9
679            2      101             58      17.000000  265.000000  24.2
375           12      140             82      43.000000  325.000000  39.2
700            2      122             76      27.000000  200.000000  35.9
505           10       75             82      20.536458   79.799479  33.3

     DiabetesPedigreeFunction  Age  predictions
568                     0.338   37            1
620                     0.246   28            0
456                     0.687   62            0
197                     0.678   23            0
714                     0.121   32            0
..                        ...  ...          ...
70                      0.867   28            0
679                     0.614   23            0
375                     0.528   58            1
700                     0.483   26            0
505                     0.263   38            0

[308 rows x 9 columns]
```