Project : Cloud Application Development

Project Statement: Machine Learning Model Deployment with IBM Cloud Watson Studio

Phase-4

Phase 4: Development Part 2

In this part you will begin building your project.

Continue building the project by deploying the model and integrating it into applications.

Deploy the trained model as a web service in IBM Cloud Watson Studio.

Integrate the deployed model into applications using the provided API endpoint.

Import the libraries:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
from scipy.stats import norm, skew
from sklearn.preprocessing import RobustScaler, StandardScaler
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score
from sklearn.metrics import roc_auc_score, roc_curve, classification_report
```

```
from warnings import filterwarnings
filterwarnings("ignore")
```

Exploratory Data

```
dataset = pd.read_csv("/kaggle/input/diabetes-dataset/diabetes.csv")
```

9

Information of Dataset

```
dataset.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
```

```
dataset.shape
```

```
(768, 9)
```

```
dataset.head(5)
```

	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

Checking for missing values:

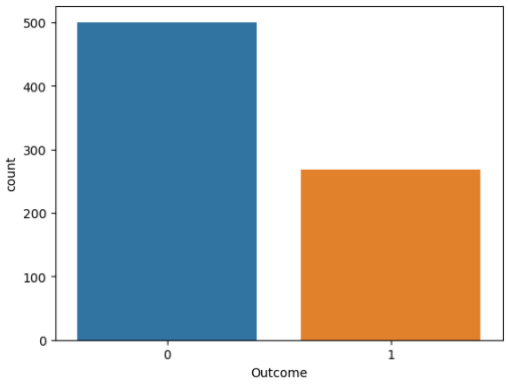
```
missing_values = dataset.isnull().sum()
print("Missing Values:")
print(missing_values)
```

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

Diabetical and Non-diabetical Persons

```
dataset["Outcome"].value_counts()
print(100 * dataset["Outcome"].value_counts() / len(dataset))
with_diabetes = dataset['Outcome'].value_counts()[1]
without_diabetes = dataset['Outcome'].value_counts()[0]
print(f"Patients with Diabetes: {with_diabetes}
          \nPatients without Diabetes: {without_diabetes}")
sns.countplot(x="Outcome", data=dataset)
```

```
Outcome
0    65.104167
1    34.895833
Name: count, dtype: float64
Patients with Diabetes: 268
Patients without Diabetes: 500
<Axes: xlabel='Outcome', ylabel='count'>
```

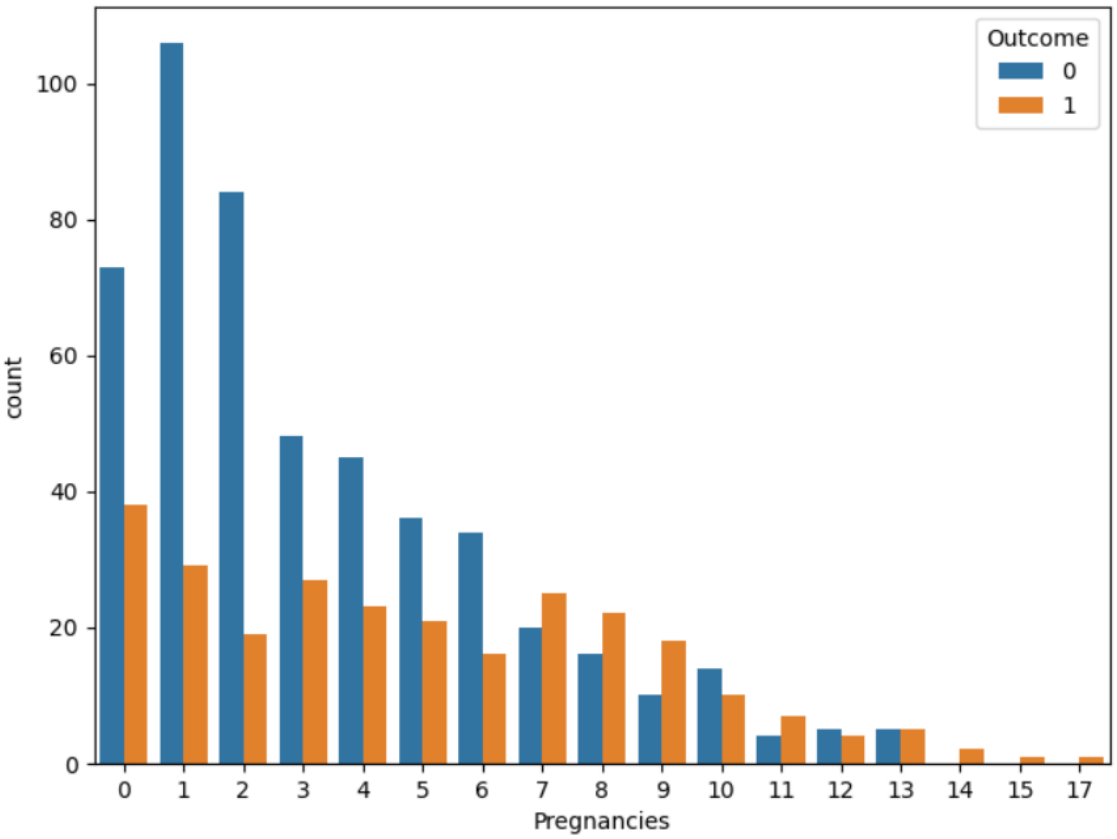


Visualizing pregnancies average outcome

plt.figure(figsize=(8,6))

sns.countplot(x='Pregnancies', hue='Outcome', data = dataset)

plt.show()



Describing data in dataset(mean, std, max)

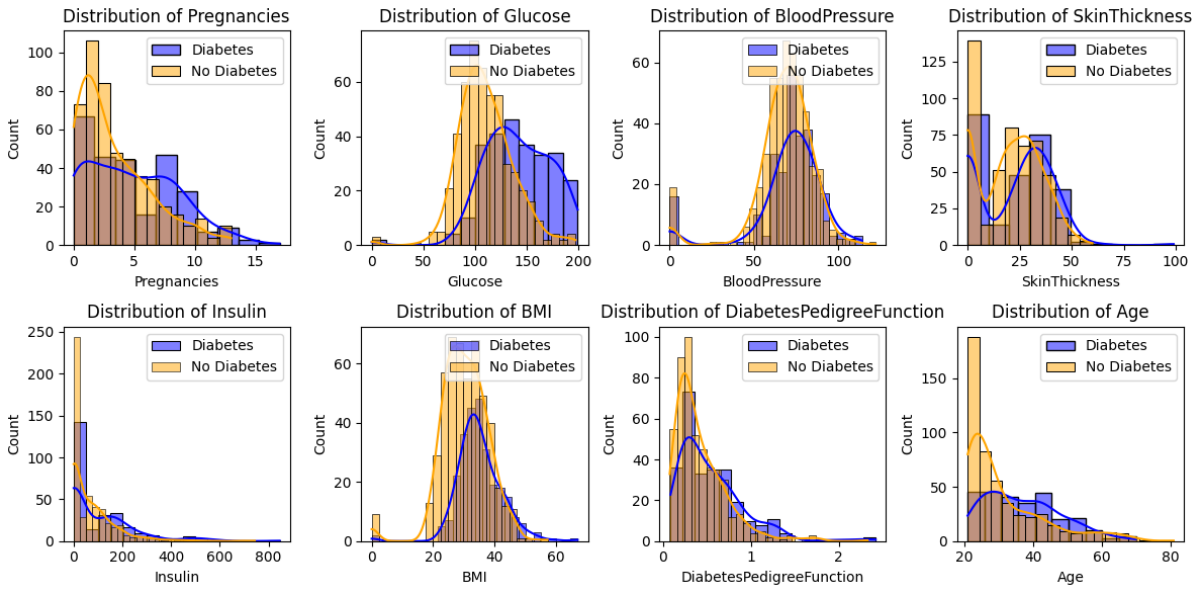
describe = dataset.describe().T

describe

	count	mean	std	min	25%	50%	75%	max
Pregnancies	768.0	3.845052	3.369578	0.000	1.00000	3.0000	6.00000	17.00
Glucose	768.0	120.894531	31.972618	0.000	99.00000	117.0000	140.25000	199.00
BloodPressure	768.0	69.105469	19.355807	0.000	62.00000	72.0000	80.00000	122.00
SkinThickness	768.0	20.536458	15.952218	0.000	0.00000	23.0000	32.00000	99.00
Insulin	768.0	79.799479	115.244002	0.000	0.00000	30.5000	127.25000	846.00
BMI	768.0	31.992578	7.884160	0.000	27.30000	32.0000	36.60000	67.10
DiabetesPedigreeFunction	768.0	0.471876	0.331329	0.078	0.24375	0.3725	0.62625	2.42
Age	768.0	33.240885	11.760232	21.000	24.00000	29.0000	41.00000	81.00
Outcome	768.0	0.348958	0.476951	0.000	0.00000	0.0000	1.00000	1.00

Visualizing the distribution of data in each column

```
plt.figure(figsize=(12, 6))
for i, col in enumerate(dataset.columns[:-1]):
    plt.subplot(2, 4, i + 1)
    sns.histplot(dataset[dataset['Outcome'] == 1][col], kde=True, label='Diabetes', color='blue')
    sns.histplot(dataset[dataset['Outcome'] == 0][col], kde=True, label='No Diabetes', color='o
range')
    plt.title(f"Distribution of {col}")
    plt.legend()
plt.tight_layout()
plt.show()
```



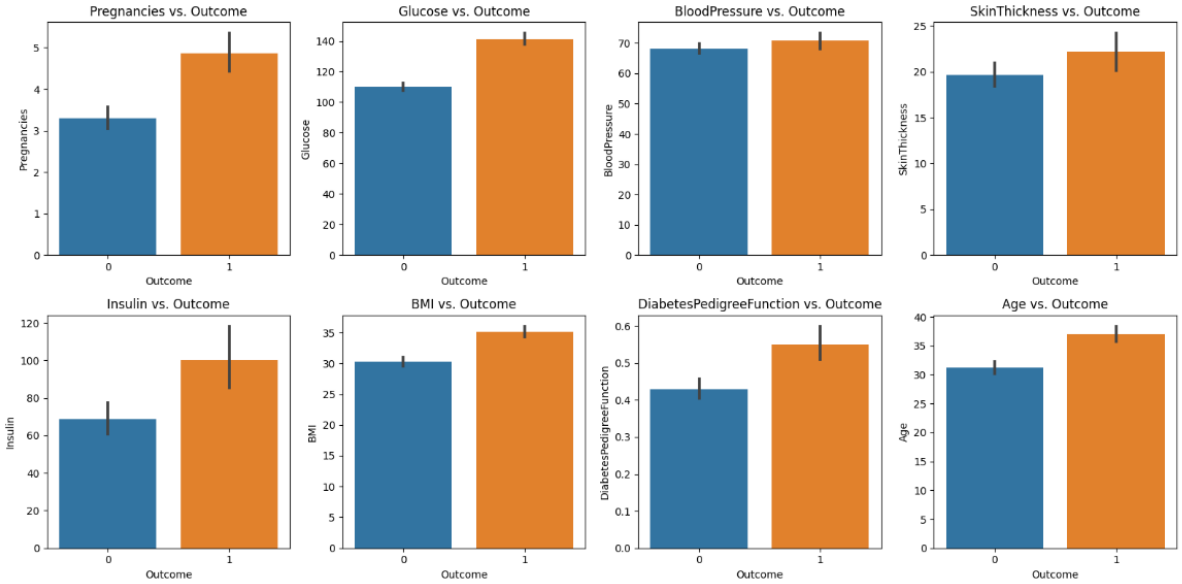
dataset.groupby('Outcome').mean()

Outcome	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	3.298000	109.980000	68.184000	19.664000	68.792000	30.304200	0.429734	31.190000
1	4.865672	141.257463	70.824627	22.164179	100.335821	35.142537	0.550500	37.067164

Target vs Features

```
plt.figure(figsize=(16,8))

for i, col in enumerate(dataset.columns[:-1]):

    plt.subplot(2, 4, i + 1)

    sns.barplot(x='Outcome', y=dataset[col], data=dataset)

    plt.title(f"{col} vs. Outcome")

    plt.tight_layout()
```

plt.show()



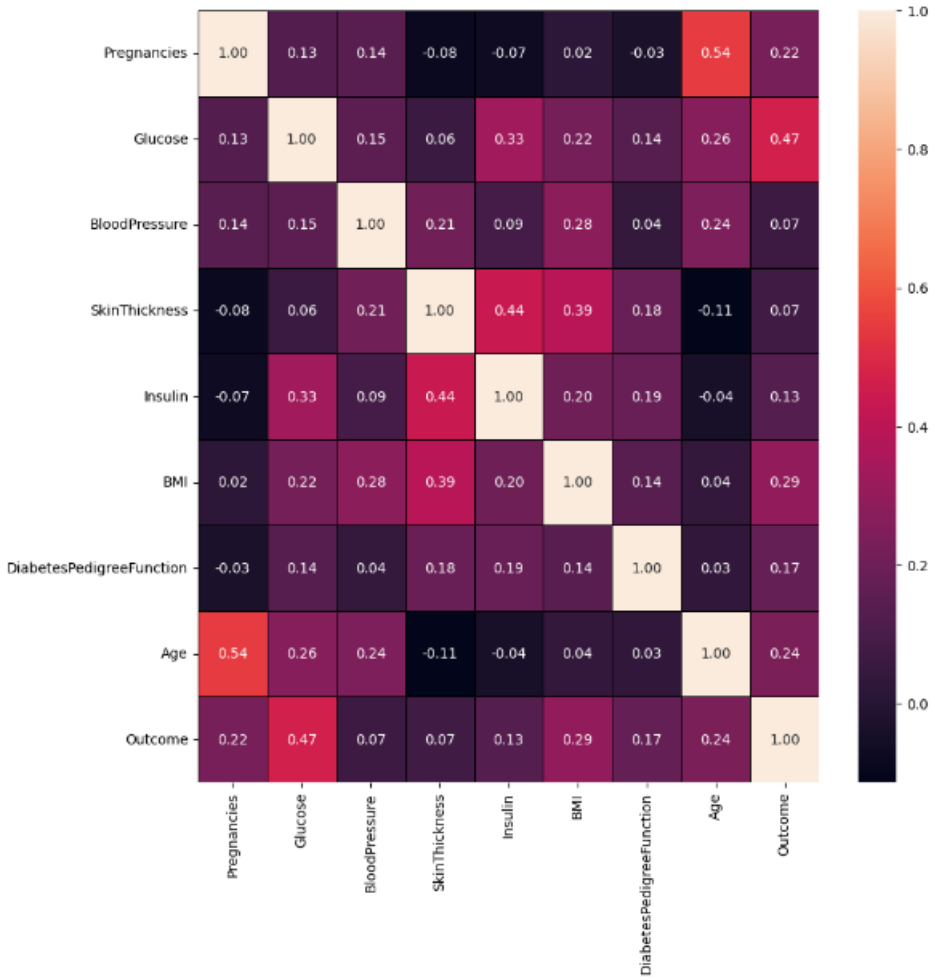
Dataset Correlation

dataset.corr()

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
Pregnancies	1.000000	0.129459	0.141282	-0.081672	-0.073535	0.017683	-0.033523	0.544341	0.221898
Glucose	0.129459	1.000000	0.152590	0.057328	0.331357	0.221071	0.137337	0.263514	0.466581
BloodPressure	0.141282	0.152590	1.000000	0.207371	0.088933	0.281805	0.041265	0.239528	0.065068
SkinThickness	-0.081672	0.057328	0.207371	1.000000	0.436783	0.392573	0.183928	-0.113970	0.074752
Insulin	-0.073535	0.331357	0.088933	0.436783	1.000000	0.197859	0.185071	-0.042163	0.130548
BMI	0.017683	0.221071	0.281805	0.392573	0.197859	1.000000	0.140647	0.036242	0.292695
DiabetesPedigreeFunction	-0.033523	0.137337	0.041265	0.183928	0.185071	0.140647	1.000000	0.033561	0.173844
Age	0.544341	0.263514	0.239528	-0.113970	-0.042163	0.036242	0.033561	1.000000	0.238356
Outcome	0.221898	0.466581	0.065068	0.074752	0.130548	0.292695	0.173844	0.238356	1.000000

Visualizing the correlation of the dataset with Heat Map.

```
f, ax = plt.subplots(figsize=(10, 10))
sns.heatmap(dataset.corr(), annot=True, linewidths=0.5, linecolor="black", fmt=".2f", ax=ax)
plt.show()
```

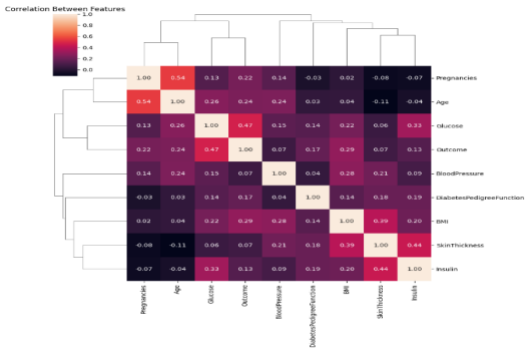


Visualizing the correlation of the dataset with Pair Plot

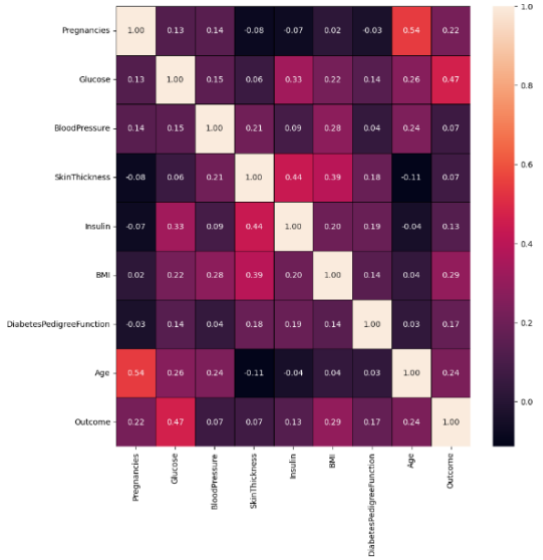
```
sns.pairplot(dataset, hue="Outcome")
plt.show()
```

Visualizing the correlation of the dataset with Cluster Map.

```
corr_matrix = dataset.corr()
sns.clustermap(corr_matrix, annot = True, fmt = ".2f")
plt.title("Correlation Between Features")
plt.show()
```




```
f, ax = plt.subplots(figsize=(10, 10))
sns.heatmap(dataset.corr(), annot=True, linewidths=0.5, linecolor="black", fmt=".2f", ax=ax)
plt.show()
```



Data Pre-Processing

Data processing (data preprocessing) are the steps taken to clean, organize and prepare raw data before it is used in analysis or modeling. This aims to ensure the data is ready to be used in machine learning algorithms so that the data provides more accurate results.

Outliers Analysis

Outliers analysis is the process of identifying and handling extreme or unusual values in a dataset. Outliers are data that is significantly different from most of the other data in the dataset. Outlier analysis is important in data processing and statistical analysis because outliers can significantly influence the analysis results.

for i in dataset.columns:

data = cleanColumn(dataset, i)

```
(767, 9) of dataset after column Pregnancies
(763, 9) of dataset after column Glucose
(731, 9) of dataset after column BloodPressure
(767, 9) of dataset after column SkinThickness
(746, 9) of dataset after column Insulin
(754, 9) of dataset after column BMI
(753, 9) of dataset after column DiabetesPedigreeFunction
(767, 9) of dataset after column Age
(768, 9) of dataset after column Outcome
```

print("New data shape: ", dataset.shape)

```
New data shape:  (768, 9)
```

Feature Engineering

The process of creating additional variables (features) from existing data or changing existing features in order to improve the quality of a machine learning model.

skewed_feats = dataset.apply(lambda x:
skew(x.dropna())).sort_values(ascending = False)

skewness = pd.DataFrame(skewed_feats, columns = ["Skewed"])

skewness

	Skewed
Insulin	2.267810
DiabetesPedigreeFunction	1.916159
Age	1.127389
Pregnancies	0.899912
Outcome	0.633776
Glucose	0.173414
SkinThickness	0.109159
BMI	-0.428143
BloodPressure	-1.840005

If skewness is positive, then this indicates that the distribution of data on that feature tends to be skewed to the right (positive), which means there is a lot of data spread across higher values. Negative skewness will indicate that the data distribution tends to be skewed to the left (negative), with more data at lower

values. Skewness close to zero indicates that the data distribution is less skewed and more symmetrical.

Splitting the Dataset into the Training set and Test Set

```
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
```

```
X_train, X_test, y_train, y_test = train_test_split
                    (X, y, test_size=0.25, random_state=0)

print("X_train shape: ", X_train.shape)
print("X_test shape: ", X_test.shape)
print("y_train shape: ", y_train.shape)
print("y_test shape: ", y_test.shape)
```

```
X_train shape:  (576, 8)
X_test shape:  (192, 8)
y_train shape:  (576,)
y_test shape:  (192,)
```

Machine Learning Classification Models

Import Logistic Regression Models

from sklearn.linear_model import LogisticRegression

Import Decision Tree Classification Models

from sklearn.tree import DecisionTreeClassifier

Import Random Forest Classification Models

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy_score, confusion_matrix

Create a dataframe for the ML models

result_ml_data = pd.DataFrame(columns = ["Model_Name", "SS_Score", "RS_Score"])

model_name = ["LR", "DT", "RF"]

result_ml_data["Model_Name"] = model_name

result_ml_data

	Model_Name	SS_Score	RS_Score
0	LR	NaN	NaN
1	DT	NaN	NaN
2	RF	NaN	NaN

Logistic Regression

Logistic Regression is a way to predict something that only has two possible outcomes. For example, whether the patient has a certain disease or not. These algorithms use mathematics to find relationships between the factors you have (such as age, weight, blood pressure, etc.) and the probability of the outcome you are interested in occurring (for example, the probability that a person has a disease). This produces an output that is in the range 0 to 1 and can be interpreted as a probability.

for i in range(0,2):

    logreg = LogisticRegression(random_state = 0)

    X_train_logreg, X_test_logreg, y_train_logreg, y_test_logreg = X_train, X_test, y_train, y_test

    logreg.fit(X_train_logreg, y_train_logreg)

    logreg_params_grid = [{"penalty" : ["l1","l2"], "solver" : ["newton-cg", "lbfgs", "liblinear", "sag", "saga"],

            "multi_class" : ["auto","ovr","multinomial"]}]

 grid_search_logreg = GridSearchCV(estimator = logreg, param_grid = logreg_params_grid, scoring = "accuracy", cv = 4)

```
    grid_search_logreg_result = grid_search_logreg.fit(train_test[i],
y_train_logreg)

    y_pred_logreg = grid_search_logreg_result.predict(train_test[i+2])

    best_param = grid_search_logreg_result.best_params_

    best_score = grid_search_logreg_result.best_score_

    print("Best prameter of grid search function: ", best_param)

    print("Best score of grid search function: ", best_score)

    if (i == 0):

        result_ml_data["SS_Score"][0] = accuracy_score(y_pred_logreg,
y_test_logreg)

    else:

        result_ml_data["RS_Score"][0] = accuracy_score(y_pred_logreg,
y_test_logreg)
```

```
Best prameter of grid search function:  {'multi_class': 'auto', 'penalty': 'l1', 'solver': 'saga'}
Best score of grid search function:  0.7552083333333334
Best prameter of grid search function:  {'multi_class': 'auto', 'penalty': 'l1', 'solver': 'saga'}
Best score of grid search function:  0.7569444444444445
```

These results show that after performing parameter search with GridSearchCV, two different sets of parameters provide similar results in terms of accuracy. This can happen because some parameter combinations can produce similar results in terms of model performance.

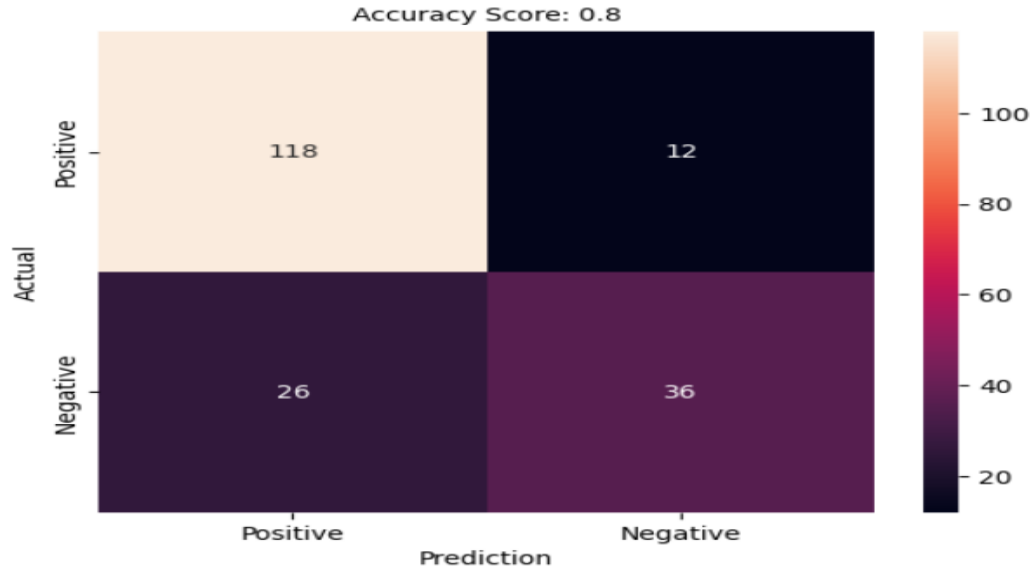
Predicting the Test results

```
logreg.predict(X_test_logreg)

# print(np.concatenate((y_pred_logreg.reshape(len(y_pred_logreg), 1),
y_test.reshape(len(y_test_logreg),1)),1))
```

```
array([1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1,
       1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1,
       1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1,
       0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0])
```

Visualizing Actual Values and Prediction Values with Confussion Matrix

```
def plot_confussion_matrix_logreg(y_test_logreg, y_pred_logreg):
  acc = round(accuracy_score(y_test_logreg, y_pred_logreg), 2)
  cm = confusion_matrix(y_pred=y_pred_logreg, y_true=y_test_logreg)
  sns.heatmap(cm, annot=True, fmt=".0f", xticklabels=["Positive", "Negative"], yticklabels=["Positive", "Negative"])
  plt.xlabel("Prediction")
  plt.ylabel("Actual")
  plt.title("Accuracy Score: {0}".format(acc), size=10)
  plt.show()
plot_confussion_matrix_logreg(y_test_logreg, y_pred_logreg)
print(classification_report(y_test_logreg, y_pred_logreg))
```




```
              precision    recall  f1-score   support

           0       0.82      0.91      0.86       130
           1       0.75      0.58      0.65        62

    accuracy                           0.80       192
   macro avg       0.78      0.74      0.76       192
weighted avg       0.80      0.80      0.79       192
```

1.True Positives (TP): This indicates that the logistic regression model correctly predicted 118 patients as positive (having diabetes).

2.False Positives (FP): This indicates that the logistic regression model incorrectly predicted 12 patients as positive (having diabetes), even though they were actually negative.

3.False Negatives (FN): This indicates that the logistic regression model incorrectly predicted 26 patients as negative (not having diabetes), even though they were actually positive.

4.True Negatives (TN): This indicates that the logistic regression model correctly predicted 36 patients as negative (not having diabetes).

Accuracy: Accuracy measures the extent to which the model is correct as a whole. It is calculated as (TP + TN) / (TP + TN + FP + FN).

Precision: Precision measures the extent to which a model's positive predictions are correct. It is calculated as TP / (TP + FP).

Recall (Sensitivity): Recall measures the extent to which your model can identify all true positive cases. It is calculated as TP / (TP + FN).

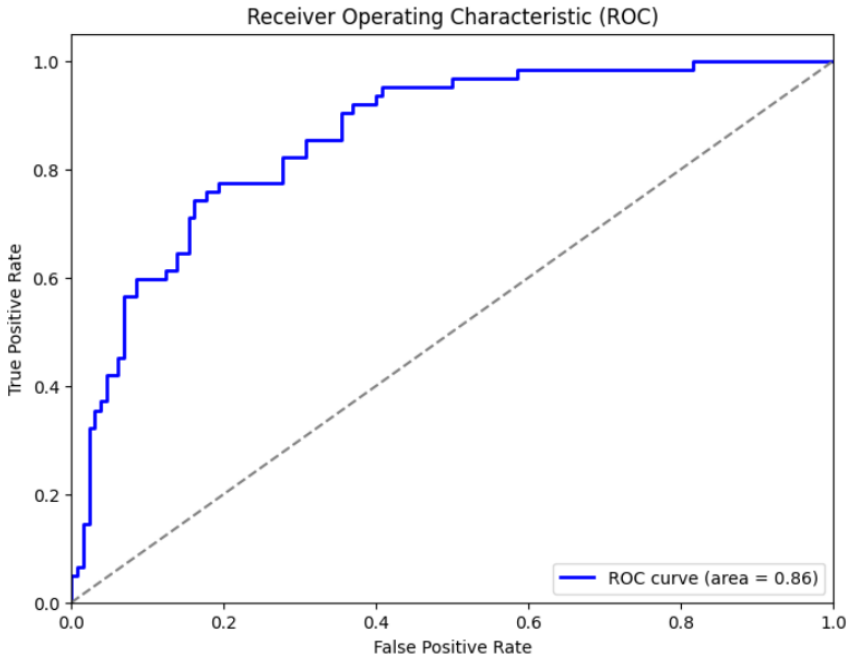
F1-Score (F1-Score): F1-Score is the harmonic average of precision and recall. This provides a balance between the two metrics.

ROC Curve Visualization

ROC is a graph used to assess the performance of classification models, such as those used in machine learning. It measures the model's ability to differentiate between positive and negative classes. This graph displays a comparison between the True Positive rate (accuracy of predicting the positive class) and the False Positive rate (error in predicting the positive class) at various different thresholds. The closer the ROC curve is to the upper left corner, the better the model's ability to differentiate between the two classes.

```
y_pred_prob = logreg.predict_proba(X_test_logreg)[:, 1]

fpr, tpr, thresholds = roc_curve(y_test_logreg, y_pred_prob)

roc_auc = roc_auc_score(y_test_logreg, y_pred_prob)

plt.figure(figsize=(8, 6))
```

```
plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC curve (area = {roc_auc:.2f})')

plt.plot([0, 1], [0, 1], color='gray', linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic (ROC)')

plt.legend(loc='lower right')

plt.show()
```



Decision Tree Classification

Decision Trees are algorithms that make decisions based on a series of questions. This is similar to playing a guessing game where each question helps us reach a final decision. For example, in a classification problem, a decision tree might ask, "Does the patient have a fever?" If the answer is "Yes," the tree will lead to the next question, and so on, until it reaches a final prediction such as "The patient has the flu." This tree helps us understand how decisions are made from data.

```
for i in range(0,2):

     dtc = DecisionTreeClassifier()

    X_train_dtc, X_test_dtc, y_train_dtc, y_test_dtc = X_train, X_test, y_train, y_test

 dtc.fit(X_train_dtc, y_train_dtc)

dtc_params = {"criterion" : ["gini", "entropy","log_loss"], "splitter" : ["best", "random"], "max_features" : ["auto", "sqrt", "log2"]}

 grid_dtc = GridSearchCV(estimator = dtc, param_grid = dtc_params, scoring = "accuracy", cv = 4)

   grid_dtc_search = grid_dtc.fit(train_test[i], y_train_dtc)

    y_pred_dtc = grid_dtc.predict(train_test[i+2])

   best_param_grid_dtc = grid_dtc_search.best_params_

    best_score_grid_dtc = grid_dtc_search.best_score_

    print("Best prameter of gridsearch function: ", best_param_grid_dtc)

    print("Best score of gridsearch function: ",best_score_grid_dtc)

    if (i == 0):

        result_ml_data["SS_Score"][4] = accuracy_score(y_pred_dtc, y_test_dtc)

    else:

        result_ml_data["RS_Score"][4] = accuracy_score(y_pred_dtc, y_test_dtc)
```

```
Best prameter of gridsearch function:  {'criterion': 'entropy', 'max_features': 'sqrt', 'splitter': 'best'}
Best score of gridsearch function:  0.7326388888888888
Best prameter of gridsearch function:  {'criterion': 'entropy', 'max_features': 'log2', 'splitter': 'best'}
Best score of gridsearch function:  0.703125
```

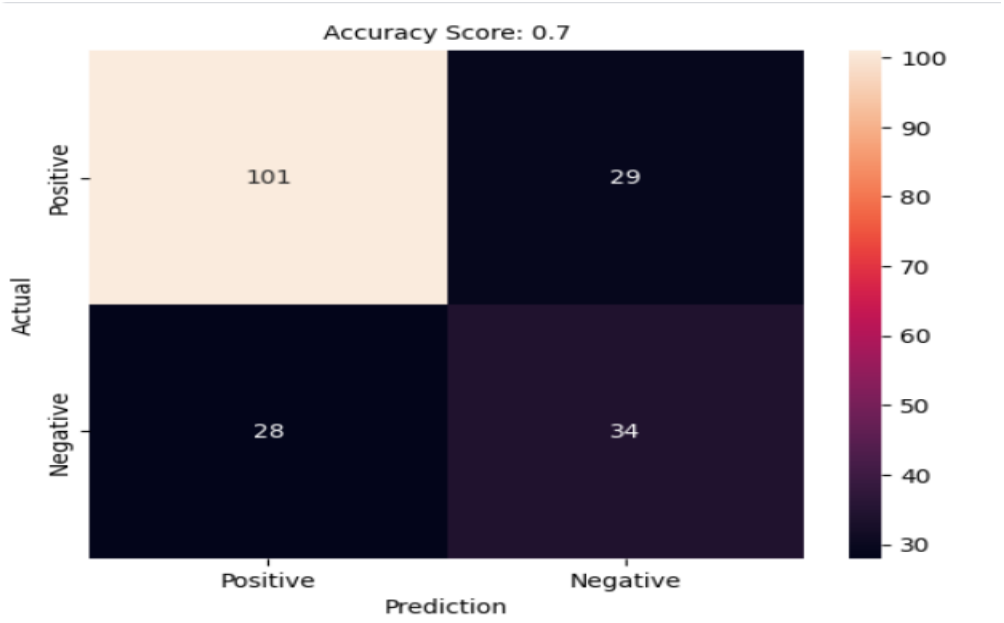
Predicting the Test Result

```
dtc.predict(X_test_dtc)
```

```
array([1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0,
       0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1,
       1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1,
       1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1,
       0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0,
       0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0])
```

Visualizing Actual Values and Prediction Values with Confussion Matrix

```
def plot_confussion_matrix_dtc(y_test_dtc, y_pred_dtc):
  acc = round(accuracy_score(y_test_dtc, y_pred_dtc), 2)
  cm = confusion_matrix(y_test_dtc, y_pred_dtc)
  sns.heatmap(cm, annot=True, fmt=".0f", xticklabels=["Positive", "Negative"], yticklabels=["Positive", "Negative"])
  plt.xlabel("Prediction")
  plt.ylabel("Actual")
  plt.title("Accuracy Score: {0}".format(acc), size=10)
  plt.show()
plot_confussion_matrix_dtc(y_test_dtc, y_pred_dtc)
print(classification_report(y_test_dtc, y_pred_dtc))
```

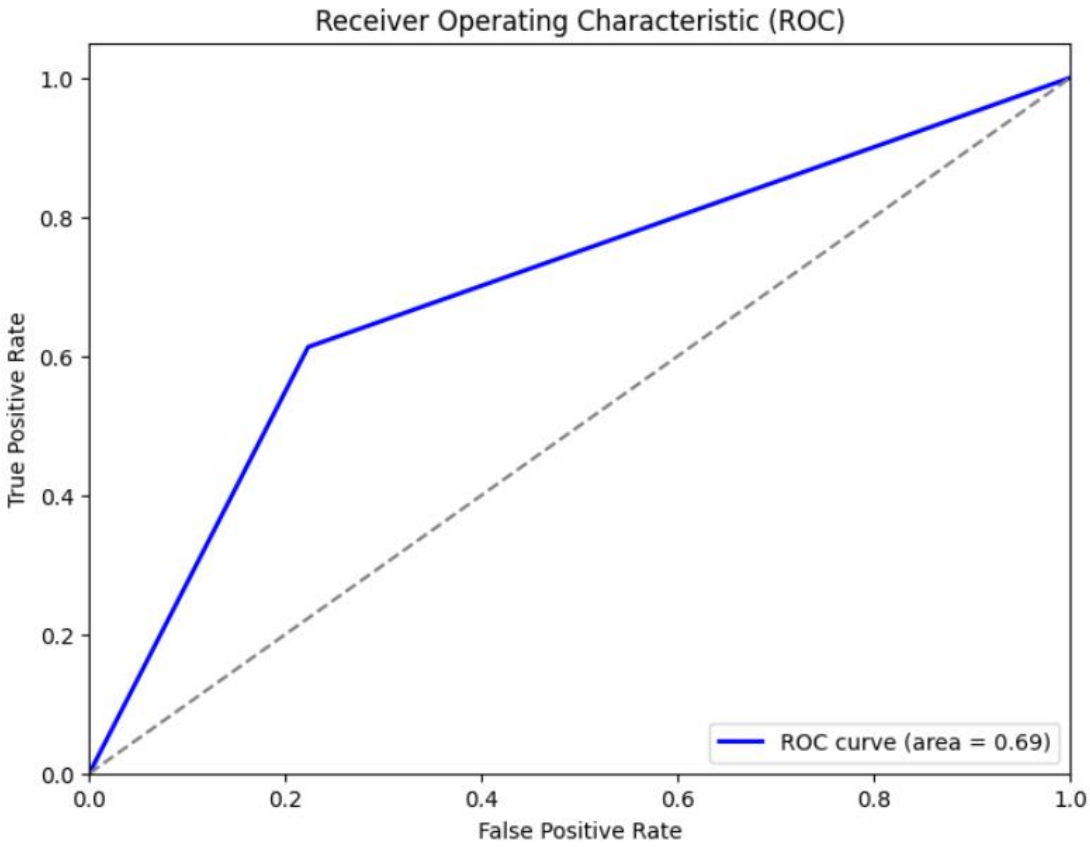


	precision	recall	f1-score	support
0	0.78	0.78	0.78	130
1	0.54	0.55	0.54	62
accuracy			0.70	192
macro avg	0.66	0.66	0.66	192
weighted avg	0.70	0.70	0.70	192

ROC Curve Visualization

```
y_pred_prob = dtc.predict_proba(X_test_dtc)[:, 1]

fpr, tpr, thresholds = roc_curve(y_test_dtc, y_pred_prob)

roc_auc = roc_auc_score(y_test_dtc, y_pred_prob)

plt.figure(figsize=(8, 6))

plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC curve (area = {roc_auc:.2f})')

plt.plot([0, 1], [0, 1], color='gray', linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')
```

```
plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic (ROC)')

plt.legend(loc='lower right')

plt.show()
```



Random Forest Classification

Random Forest is a machine learning algorithm that combines many decision trees to produce more accurate predictions. It's like gathering many different views to make better decisions. Rather than relying on a single decision tree, Random Forest uses a “forest” containing many trees that collectively provide more reliable results. This helps overcome overfitting and improve model performance.

```
for i in range(0,2):

    rfc = RandomForestClassifier()
```

```
    X_train_rfc, X_test_rfc, y_train_rfc, y_test_rfc = X_train, X_test, y_train, y_test

    rfc.fit(X_train_rfc, y_train_rfc)

    param_rfc = {"n_estimators" : range(1,50), "criterion" : ["gini", "entropy","log_loss"], "max_features" : ["sqrt","log2", None],

            "class_weight" : ["balanced", "balanced_subsample"]}

    grid_rfc = GridSearchCV(estimator = rfc, param_grid = param_rfc, scoring = "accuracy", cv = 4)

    grid_rfc_search = grid_rfc.fit(train_test[i], y_train_rfc)

    y_pred_rfc = grid_rfc.predict(train_test[i+2])

    best_parm_grid_rfc = grid_rfc_search.best_params_

    best_score_grid_rfc = grid_rfc_search.best_score_

    print("Best prameter of gridsearch function: ", best_parm_grid_rfc)

    print("Best score of gridsearch function: ",best_score_grid_rfc)

    if (i == 0):

        result_ml_data["SS_Score"][5] = accuracy_score(y_pred_rfc, y_test_rfc)

    else:

        result_ml_data["RS_Score"][5] = accuracy_score(y_pred_rfc, y_test_rfc)
```

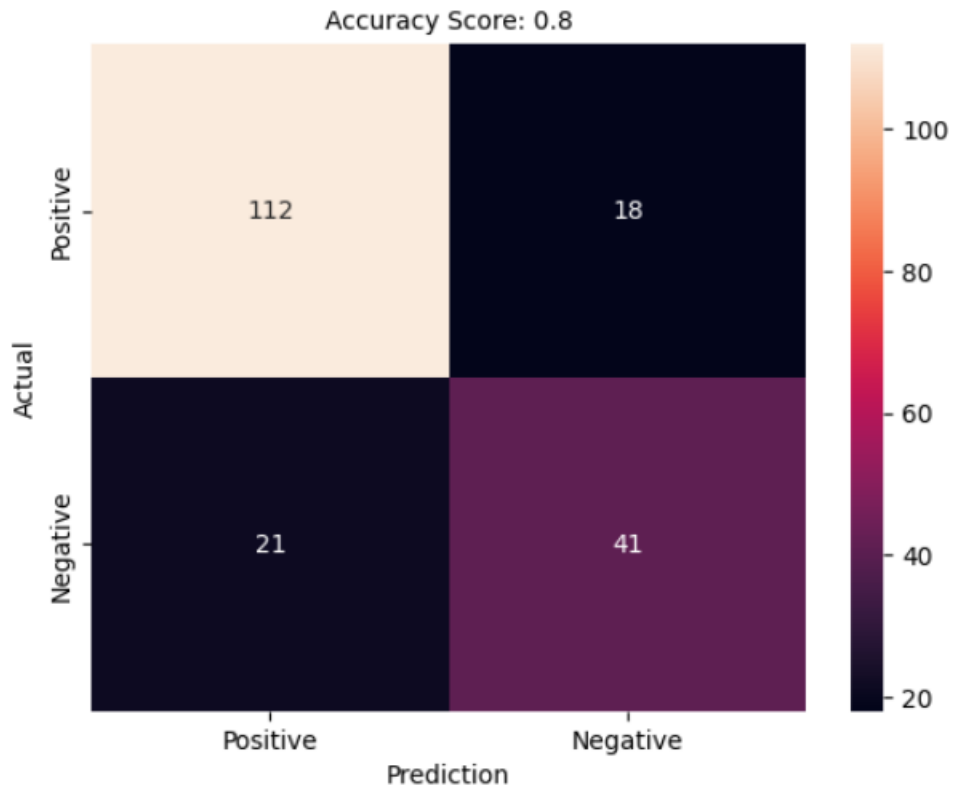
Predicting the Test Result

```
rfc.predict(X_test_rfc)
```

Visualizing Actual Values and Prediction Values with Confussion Matrix

```
def plot_confussion_matrix_rfc(y_test_rfc, y_pred_rfc):

  acc = round(accuracy_score(y_test_rfc, y_pred_rfc), 2)
```

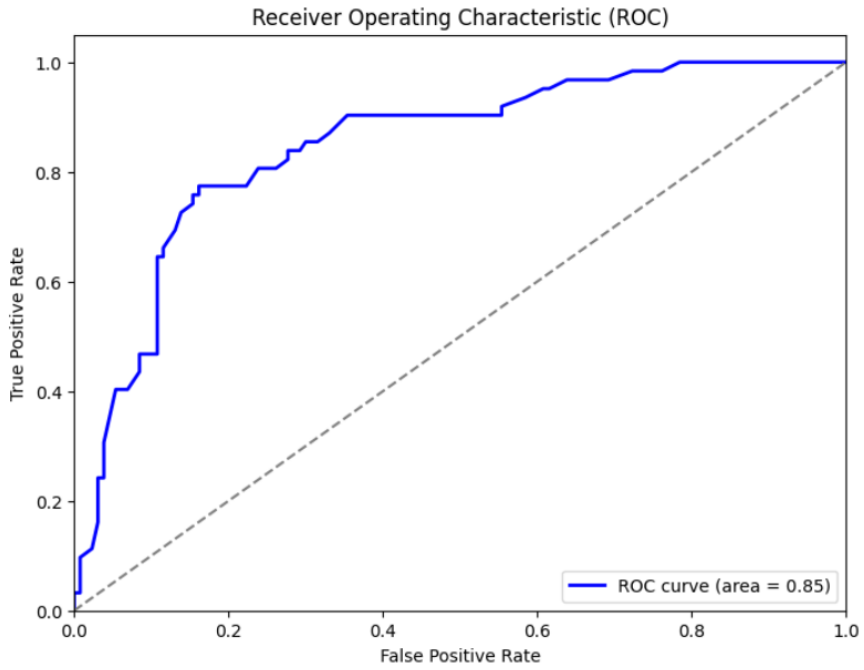
```
  cm = confusion_matrix(y_pred=y_pred_rfc, y_true=y_test_rfc)

  sns.heatmap(cm, annot=True, fmt=".0f", xticklabels=["Positive", "Negative"],
yticklabels=["Positive", "Negative"])

  plt.xlabel("Prediction")

  plt.ylabel("Actual")

  plt.title("Accuracy Score: {0}".format(acc), size=10)

  plt.show()

plot_confussion_matrix_dtc(y_test_rfc, y_pred_rfc)

print(classification_report(y_test_rfc, y_pred_rfc))
```




```
              precision    recall  f1-score   support

           0       0.84      0.86      0.85       130
           1       0.69      0.66      0.68        62

    accuracy                           0.80       192
   macro avg       0.77      0.76      0.76       192
weighted avg       0.79      0.80      0.80       192
```

ROC Curve Visualization

```
y_pred_prob = rfc.predict_proba(X_test_rfc)[:, 1]

fpr, tpr, thresholds = roc_curve(y_test_rfc, y_pred_prob)

roc_auc = roc_auc_score(y_test_rfc, y_pred_prob)

plt.figure(figsize=(8, 6))

plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC curve (area = {roc_auc:.2f})')

plt.plot([0, 1], [0, 1], color='gray', linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic (ROC)')

plt.legend(loc='lower right')

plt.show()
```



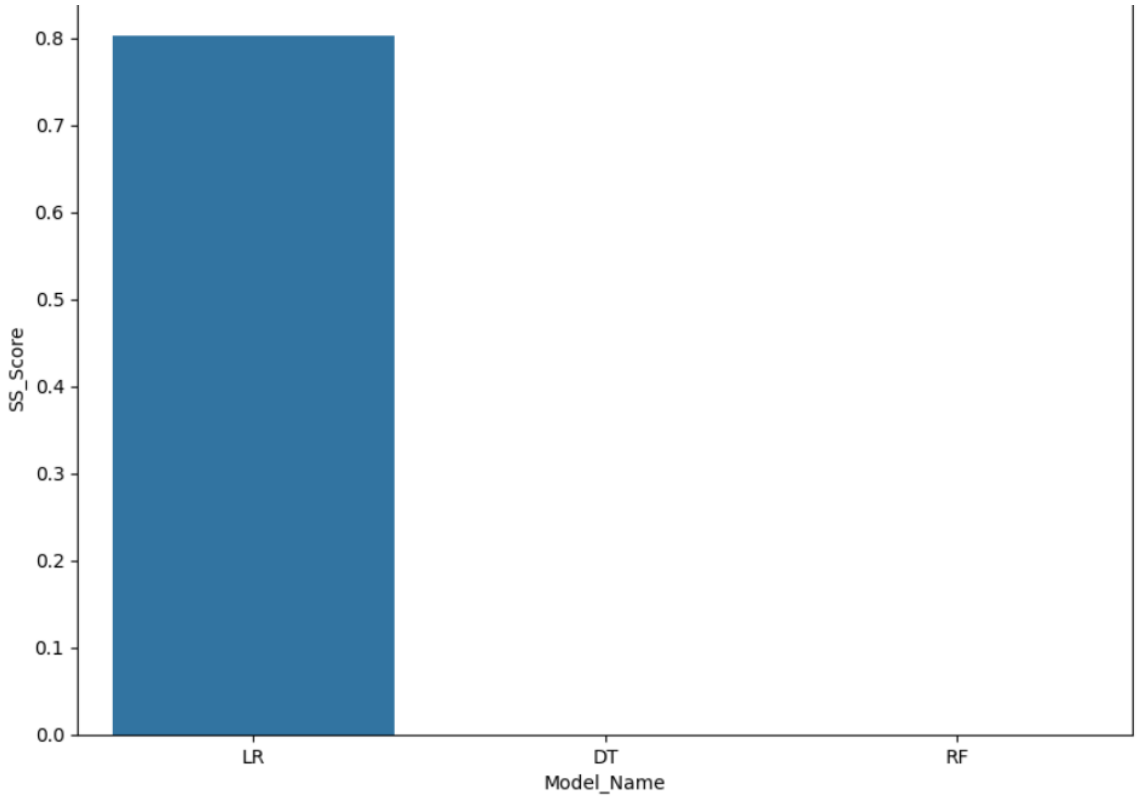
Machine Learning Models Assestment

result_ml_data

	Model_Name	SS_Score	RS_Score
0	LR	0.802083	0.802083
1	DT	NaN	NaN
2	RF	NaN	NaN

Visualizing Result with Standart Scaler

```
plt.figure(figsize=(10,7))

sns.barplot(x=result_ml_data["Model_Name"], y=result_ml_data["SS_Score"])

plt.xticks(rotation=0)

plt.xlabel("Model_Name")

plt.ylabel("SS_Score")

plt.title("Result with Standart Scaler")
```



Visualizing Result with Robust Scaller

```
plt.figure(figsize=(10,7))

sns.barplot(x=result_ml_data["Model_Name"], y=result_ml_data["RS_Score"])

plt.xticks(rotation=0)

plt.xlabel("Model_Name")

plt.ylabel("RS_Score")

plt.title("Result with Robust Scaler")
```