GOVERNMENT COLLEGE OF ENGINEERING BARGUR ( AUTONOMOUS)

Project : Cloud Application Development

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

Team members:

VIJAY P

SANJITH T

NISHANTH S

HARIHARAN P

KABILESHWARAN M

**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

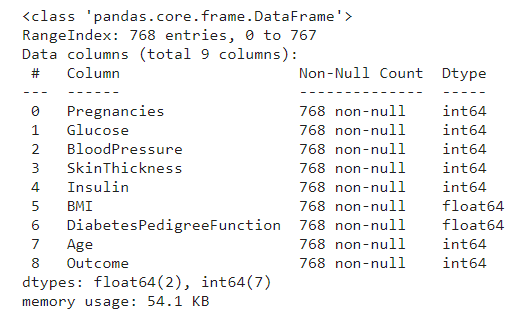
filterwarnings("ignore")

**Exploratory Data**

dataset = pd.read\_csv("/kaggle/input/diabetes-dataset/diabetes.csv")

## **Information of Dataset**

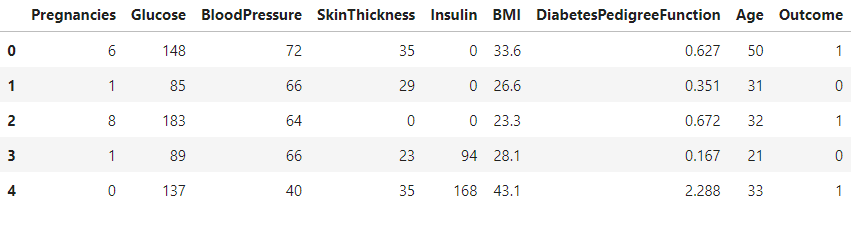
dataset.info()



dataset.shape

(768, 9)

dataset.head(5)

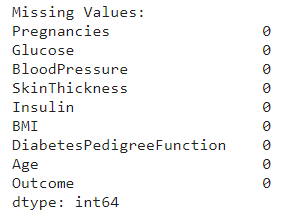


## **Checking for missing values:**

missing\_values = dataset.isnull().sum()

print("Missing Values:")

print(missing\_values)



## **Diabetical and Non-diabetical Persons**

dataset["Outcome"].value\_counts()

print(100 \* dataset["Outcome"].value\_counts() / len(dataset))

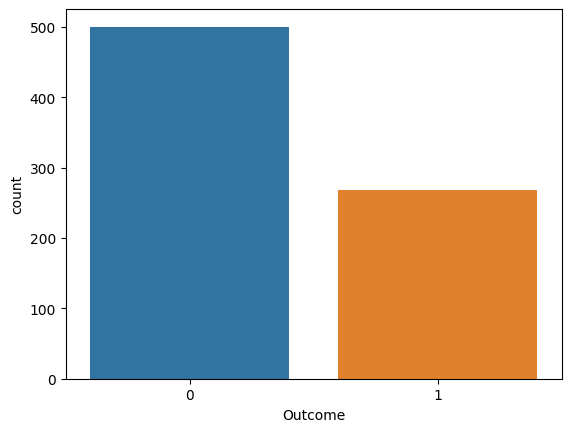
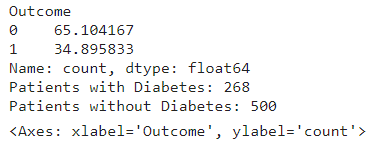
with\_diabetes = dataset['Outcome'].value\_counts()[1]

without\_diabetes = dataset['Outcome'].value\_counts()[0]

print(f"Patients with Diabetes: **{**with\_diabetes**}**

**\n**Patients without Diabetes: **{**without\_diabetes**}**")

sns.countplot(x="Outcome", data=dataset)

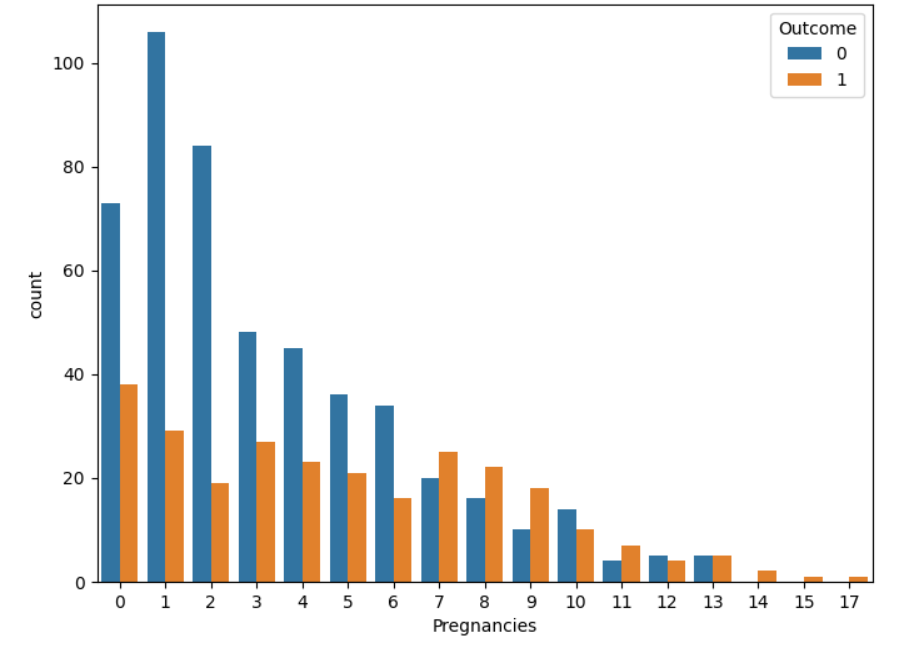


## Visualizing pregnancies average outcome

## 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



**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')

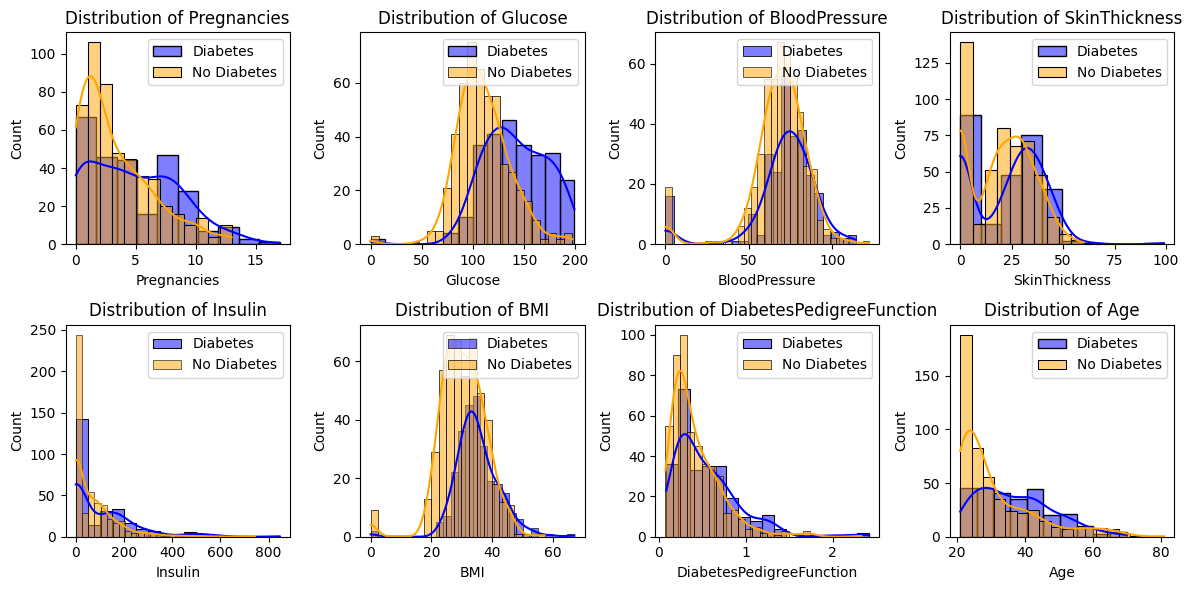
sns.histplot(dataset[dataset['Outcome'] == 0][col], kde=True, label='No Diabetes', color='orange')

plt.title(f"Distribution of **{**col**}**")

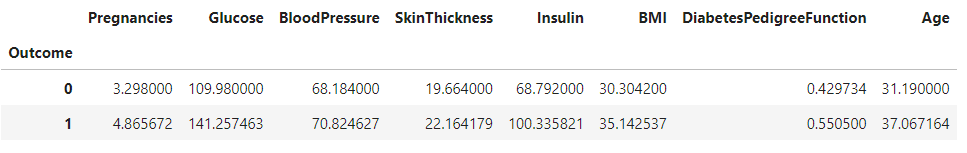
plt.legend()

plt.tight\_layout()

plt.show()



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



## Target vs Features

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

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

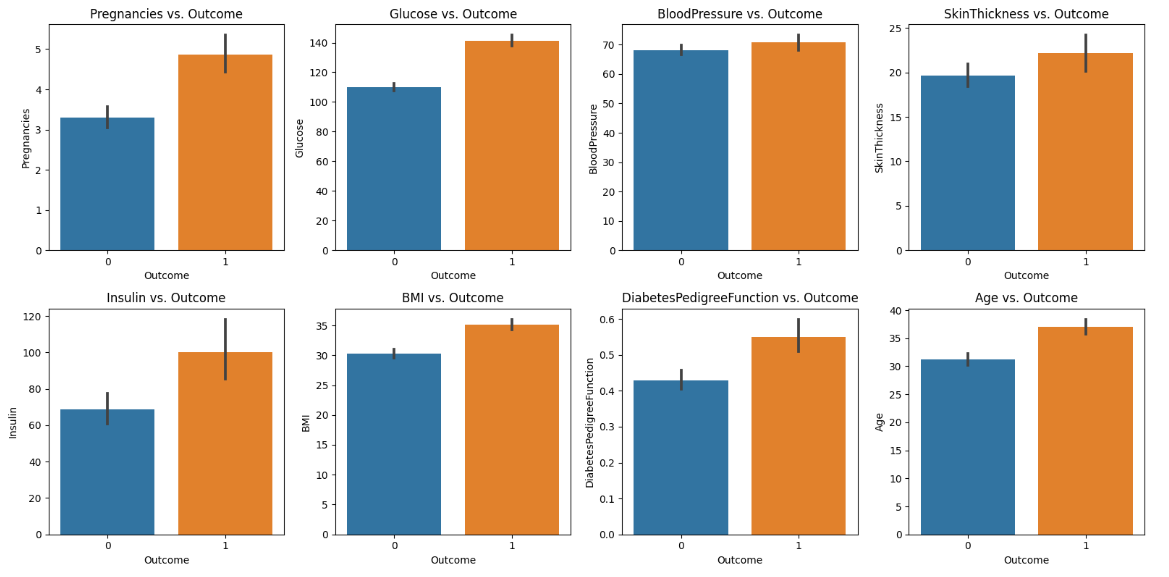
plt.subplot(2, 4, i + 1)

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

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

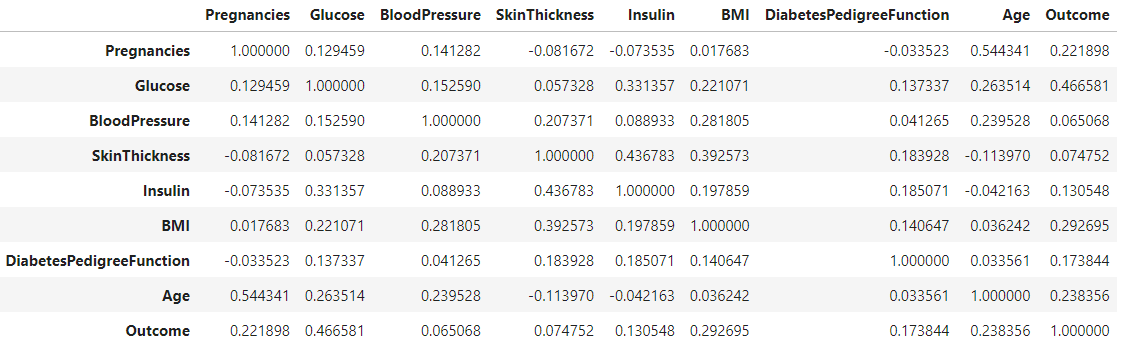
plt.tight\_layout()

plt.show()



## Dataset Correlation

dataset.corr()

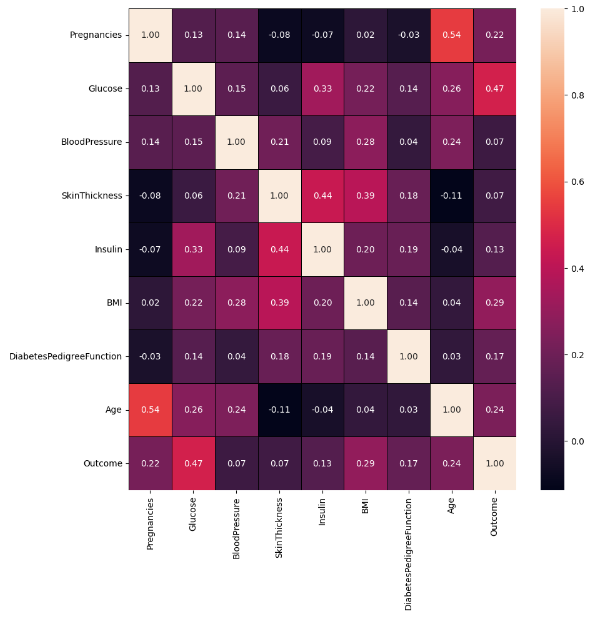


## Visualizing the correlation of the dataset with Heat Map.

f, ax = plt.subplots(figsize=(10, 10))

sns.heatmap(dataset.corr(), annot=True, linewidths=0.5, linecolor="black", fmt=".2f", ax=ax)

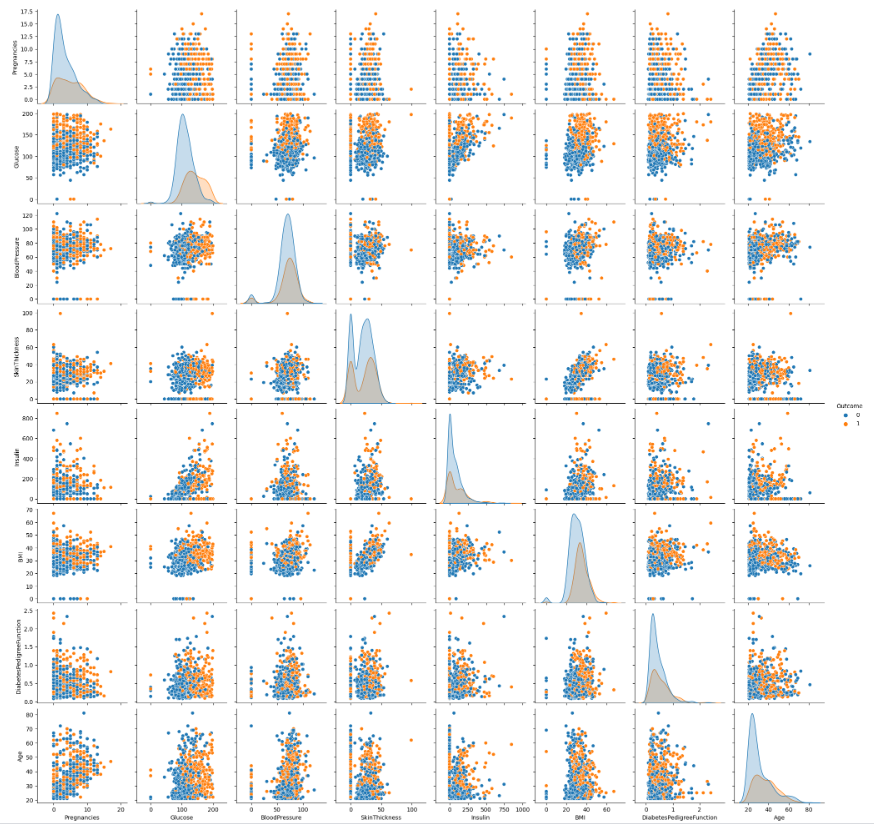
plt.show()



## Visualizing the correlation of the dataset with Pair Plot

sns.pairplot(dataset, hue="Outcome")

plt.show()



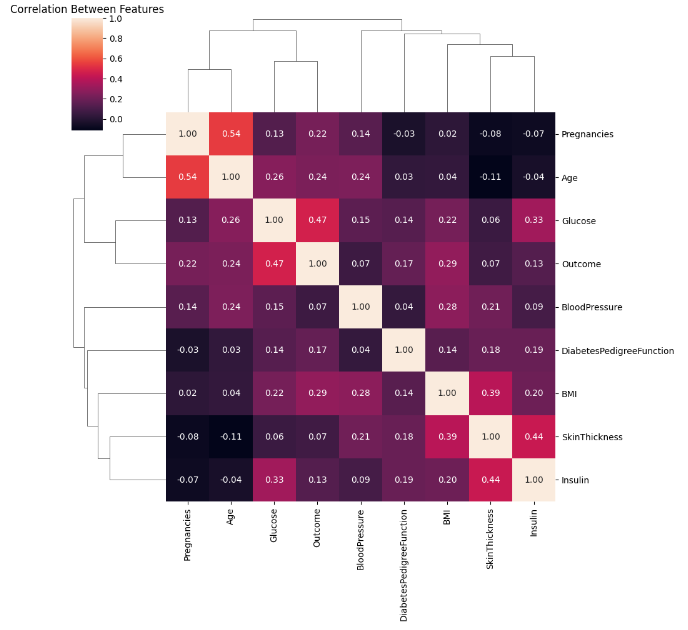
## Visualizing the correlation of the dataset with Cluster Map.

corr\_matrix = dataset.corr()

sns.clustermap(corr\_matrix, annot = True, fmt = ".2f")

plt.title("Correlation Between Features")

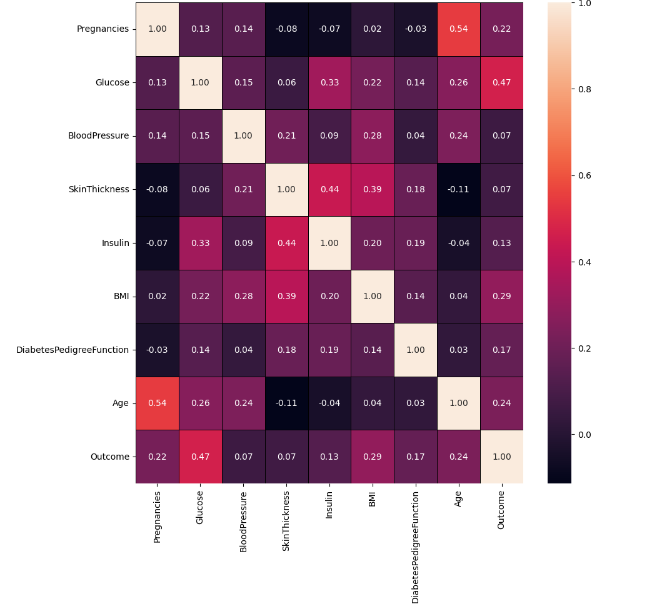
plt.show()



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**

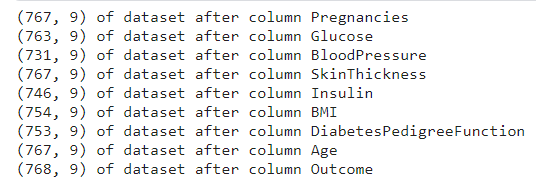
Data processing (data preprocessing) are the steps taken to clean, organize and prepare raw data before it is used in analysis or modeling. This aims to ensure the data is ready to be used in machine learning algorithms so that the data provides more accurate results.

**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)



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



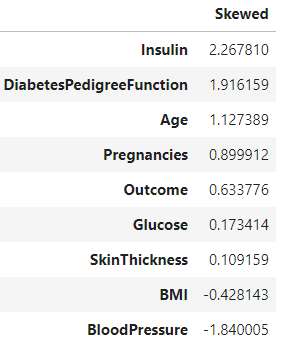
**Feature Engineering**

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



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

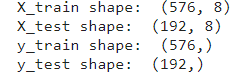
(X, y, test\_size=0.25, random\_state=0)

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)



**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

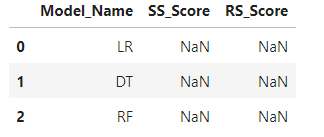
**Create a dataframe for the ML models**

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



**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)

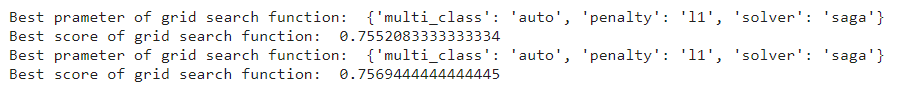
print("Best score of grid search function: ", best\_score)

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)

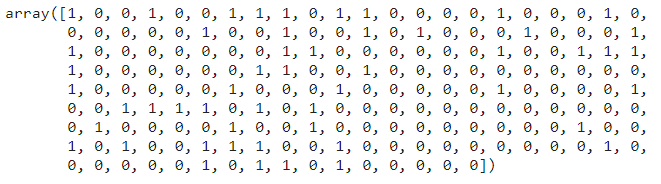


These results show that after performing parameter search with GridSearchCV, two different sets of parameters provide similar results in terms of accuracy. This can happen because some parameter combinations can produce similar results in terms of model performance.

**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))



**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")

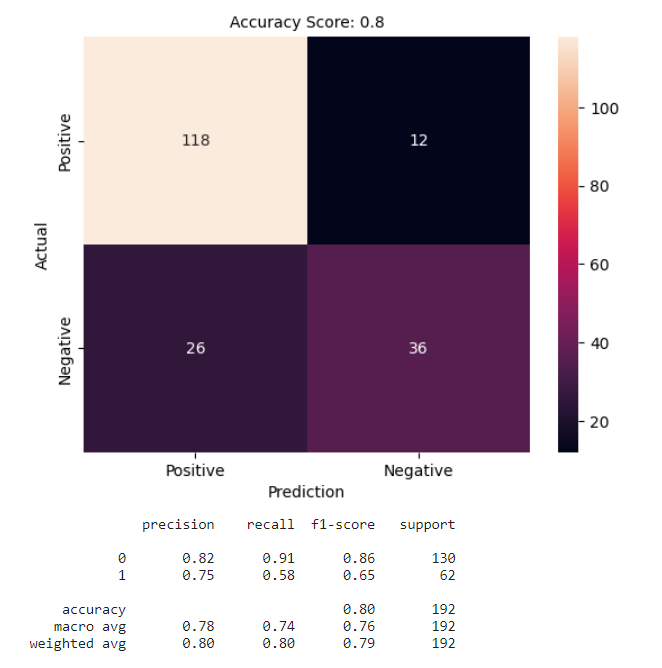
plt.ylabel("Actual")

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))



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])

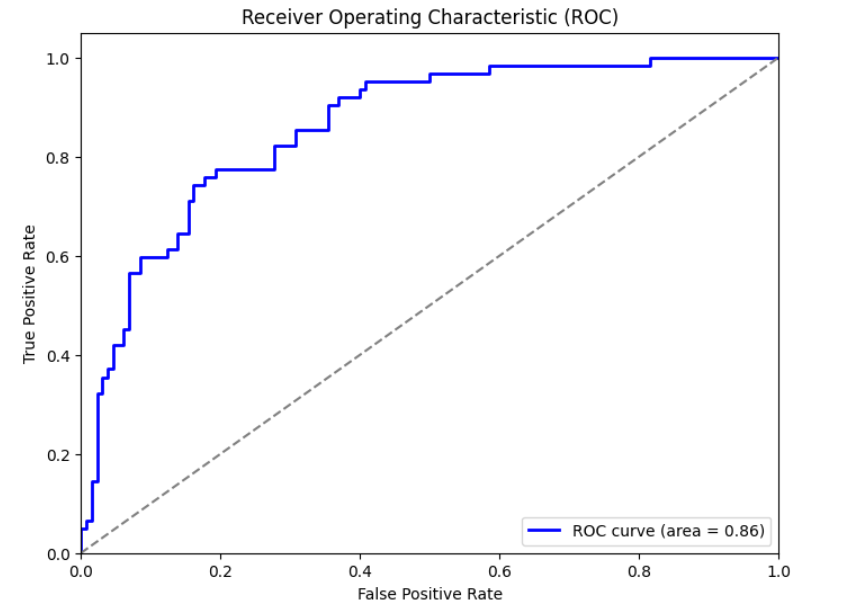
plt.xlabel('False Positive Rate')

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)

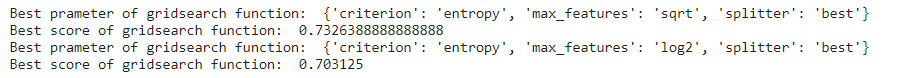
print("Best score of gridsearch function: ",best\_score\_grid\_dtc)

if (i == 0):

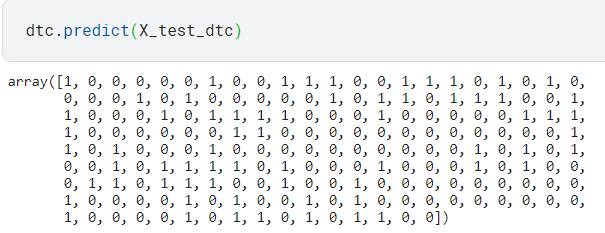
result\_ml\_data["SS\_Score"][4] = accuracy\_score(y\_pred\_dtc, y\_test\_dtc)

else:

result\_ml\_data["RS\_Score"][4] = accuracy\_score(y\_pred\_dtc, y\_test\_dtc)



**Predicting the Test Result**



**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")

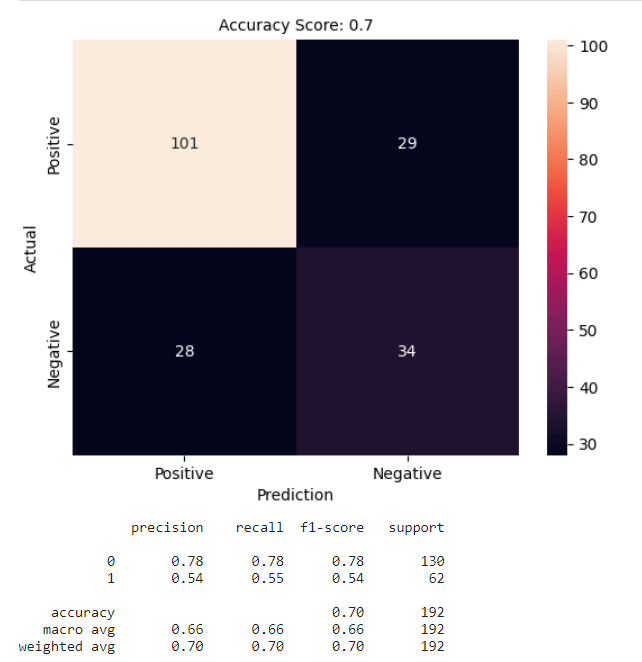
plt.ylabel("Actual")

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))



**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])

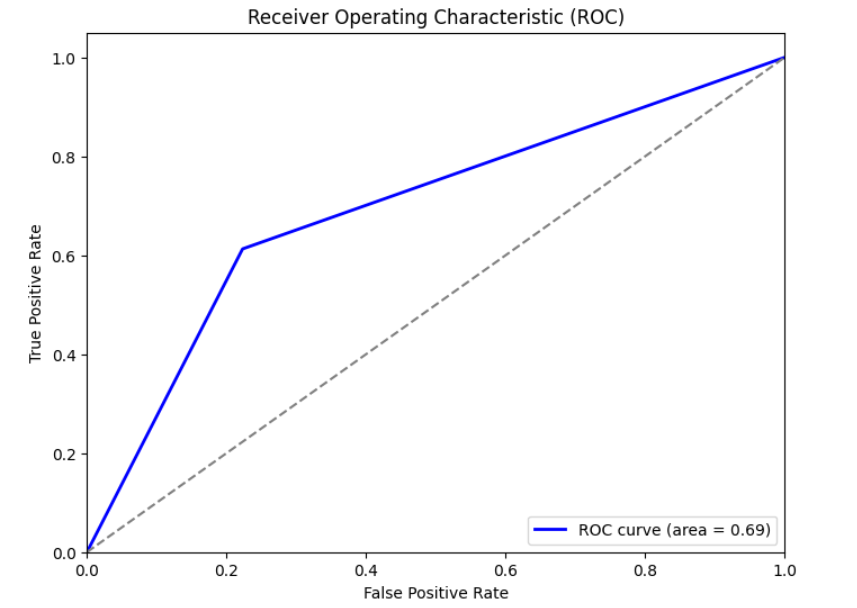
plt.xlabel('False Positive Rate')

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")

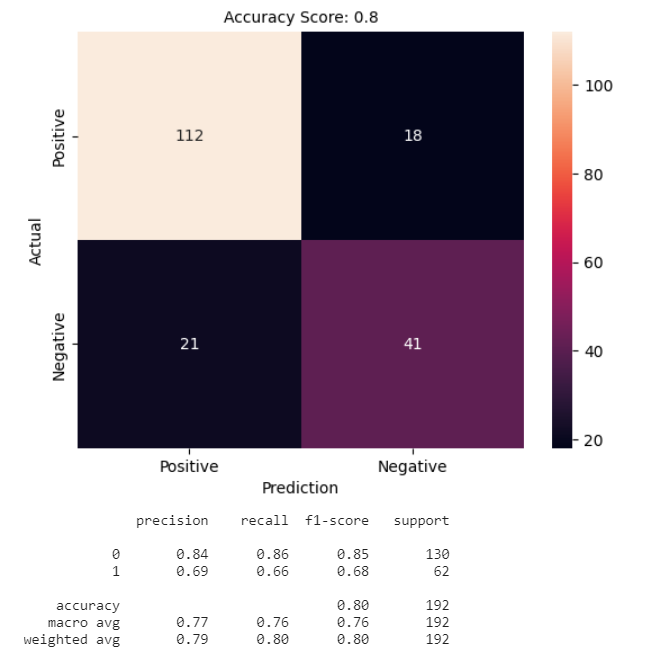
plt.ylabel("Actual")

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))



**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])

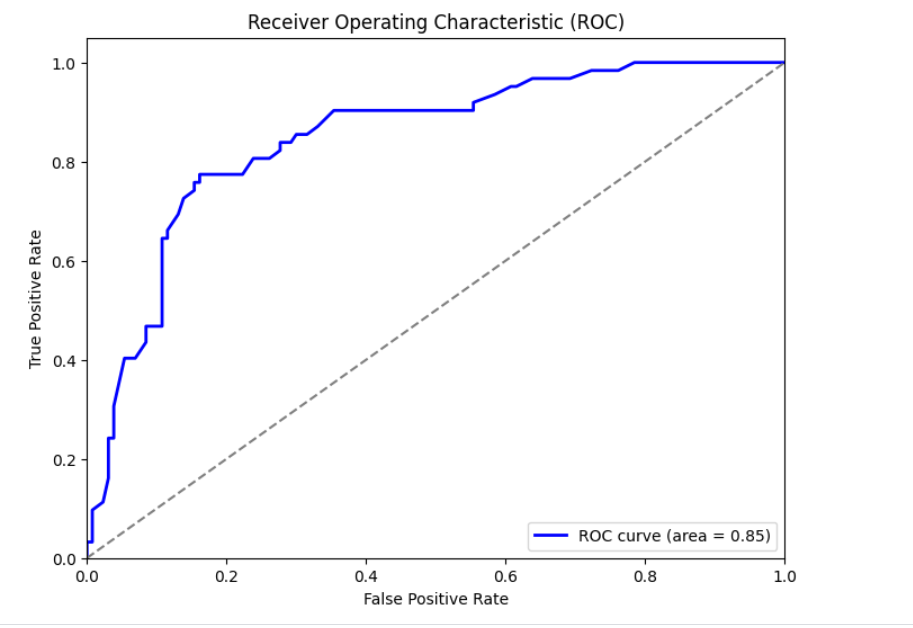
plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

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

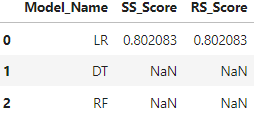
plt.legend(loc='lower right')

plt.show()



**Machine Learning Models Assestment**

result\_ml\_data



**Visualizing Result with Standart Scaler**

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

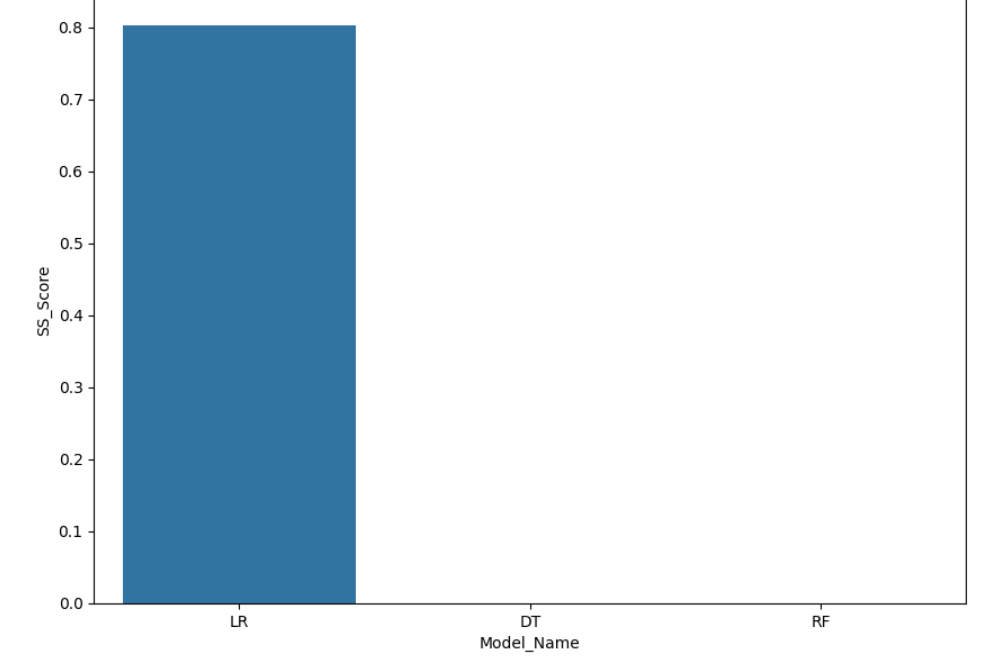
sns.barplot(x=result\_ml\_data["Model\_Name"], y=result\_ml\_data["SS\_Score"])

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")

