Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)

Machine Learning Mini-Project

DIABETES PREDICTION

Submitted in partial fulfilment of the requirement of
the Machine Learning Laboratory

Department of Computer Science and Engineering (Data Science)

By
Advay Sharma 60009220147

A.Y. 2023 – 2024

Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)

Aim:

This project aims to develop a machine learning model capable of predicting whether a patient has diabetes based on various diagnostic measurements. It falls under the category of binary classification problems. The model will analyse the provided data and classify a new patient as either diabetic (positive class) or non-diabetic (negative class).

Data Description:

Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years old of Pima Indian heritage.

- Pregnancies: Number of times pregnant
- Glucose: Plasma glucose concentration a 2 hours in an oral glucose tolerance test
- BloodPressure: Diastolic blood pressure (mm Hg)
- SkinThickness: Triceps skin fold thickness (mm)
- Insulin: 2-Hour serum insulin (mu U/ml)
- BMI: Body mass index (weight in kg/(height in m)^2)
- DiabetesPedigreeFunction: Diabetes pedigree function
- Age: Age (years)
- Outcome: Class variable (0 or 1)

Data Preprocessing:

Cell 1: Finding Null values

```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
df = pd.read_csv('/content/drive/MyDrive/diabetes(1).csv')
# Find null values
df.isnull().sum()
```

```
Pregnancies              75
Glucose               5
BloodPressure            35
SkinThickness           227
Insulin             376
BMI                 11
DiabetesPedigreeFunction     0
Age                 10
Outcome                 0
dtype: int64
```

Cell 2: Filling Null values

```python
 df.fillna({'Pregnancies':df['Pregnancies'].median()},inplace=True)
df.fillna({'Glucose':'Medium'},inplace=True)
df.fillna({'BloodPressure':df['BloodPressure'].median()},inplace=True)
df.fillna({'SkinThickness':df['SkinThickness'].median()},inplace=True)
df.fillna({'Insulin':df['Insulin'].median()},inplace=True)
df.fillna({'BMI':df['BMI'].median()},inplace=True)
df.fillna({'Age':df['Age'].median()},inplace=True)
# Check for the null values
df.isnull().sum()
```

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

Cell 3: Summary of Data Frame Columns

```python
output = []
for col in df.columns:
    unique = df[col].nunique()
    colType = str(df[col].dtype)
    categories=df[col].unique()
    output.append([col, unique, colType,categories])
output = pd.DataFrame(output)
output.columns = ['colName','unique','dtype','categories']
output
```

	colName	unique	dtype	categories
0	Pregnancies	16	float64	[6.0, 1.0, 8.0, 0.0, 5.0, 2.0, 10.0, 4.0, 7.0,...
1	Glucose	3	category	['High', 'Low', 'Medium'] Categories (3, objec...
2	BloodPressure	46	float64	[72.0, 66.0, 64.0, 40.0, 74.0, 50.0, 70.0, 96....
3	SkinThickness	50	float64	[35.0, 29.0, 23.0, 32.0, 45.0, 19.0, 47.0, 38....
4	Insulin	184	float64	[125.5, 94.0, 168.0, 88.0, 543.0, 846.0, 175.0...
5	BMI	247	float64	[33.6, 26.6, 23.3, 28.1, 43.1, 25.6, 31.0, 35....
6	DiabetesPedigreeFunction	517	float64	[0.627, 0.351, 0.672, 0.167, 2.288, 0.201, 0.2...
7	Age	52	float64	[50.0, 31.0, 32.0, 21.0, 33.0, 30.0, 26.0, 29....
8	Outcome	2	int64	[1, 0]

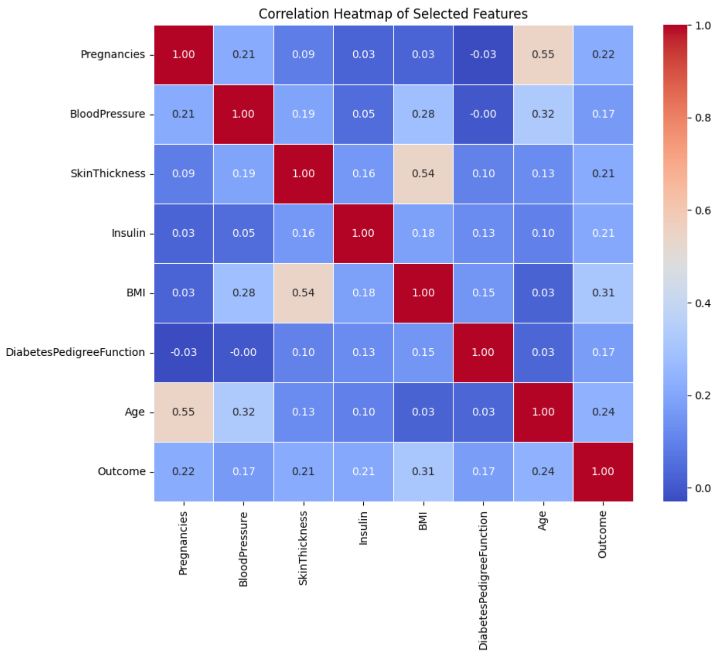
Cell 4: Count of Low, Medium, High in Glucose

```
df['Glucose'].value_counts()
```

```
Glucose
Low     259
High    258
Medium   251
Name: count, dtype: int64
```

Cell 5: Correlation Heat Map

```
import seaborn as sns
import matplotlib.pyplot as plt
# Define the subset of features to keep
selected_features                                                    =
['Pregnancies','BloodPressure','SkinThickness','Insulin','BMI','DiabetesPedi
greeFunction','Age','Outcome']
# Create a new DataFrame with only the selected features
reduced_df = df[selected_features]
# Plot the correlation heatmap for the reduced DataFrame
plt.figure(figsize=(10, 8))
sns.heatmap(reduced_df.corr(),   annot=True,   cmap='coolwarm',   fmt=".2f",
linewidths=0.5)
plt.title('Correlation Heatmap of Selected Features')
plt.show()
```



Cell 6: Relation between Age and its highly related column

```
pd.crosstab(df['Age'], df['Pregnancies'])
```

```
Pregnancies  0.0  1.0  2.0  4.0  5.0  6.0  7.0  8.0  9.0  10.0  11.0  12.0  13.0  14.0  15.0  17.0
       Age
  21.0        21   23   17    2    0    0    0    0    0     0     0     0     0     0     0     0
  22.0        17   22   28    4    0    0    0    1    0     0     0     0     0     0     0     0
  23.0         8   13   13    3    0    1    0    0    0     0     0     0     0     0     0     0
  24.0         9   14   16    4    1    1    1    0    0     0     0     0     0     0     0     0
  25.0        13    7   23    2    3    0    0    0    0     0     0     0     0     0     0     0
  26.0         9    7   10    4    1    2    0    0    0     0     0     0     0     0     0     0
  27.0         5    4   14    3    4    2    0    0    0     0     0     0     0     0     0     0
  28.0         2    8   11    5    4    5    0    0    0     0     0     0     0     0     0     0
  29.0         3    6    7    5    2    3    1    0    1     1     0     0     0     0     0     0
  30.0         1    3    6    4    5    1    0    0    0     1     0     0     0     0     0     0
  31.0         4    2    4    6    0    3    3    1    0     1     0     0     0     0     0     0
  32.0         2    2    2    1    2    3    2    1    1     0     0     0     0     0     0     0
  33.0         2    4    1    3    2    1    0    1    3     0     0     0     0     0     0     0
  34.0         0    0    3    3    1    1    2    1    1     2     0     0     0     0     0     0
  35.0         2    0    1    1    3    1    0    0    0     0     2     0     0     0     0     0
  36.0         1    2    2    2    1    1    3    2    1     1     0     0     0     0     0     0
  37.0         0    1    1    6    4    1    4    1    0     1     0     0     0     0     0     0
  38.0         1    2    1    1    3    0    0    1    1     3     0     1     1     1     0     0
```

Cell 7: Save Data Frame as a CSV File in Google Drive

```
df.to_csv('/content/drive/MyDrive/Cleaned_diabetes(1).csv', index=False)
```

Performance Evaluation:

Cell 8: Final Report (Logistic Regression, Naïve Bayes, Decision Tree, Random Forest AdaBoost, Support Vector Machine) to chose which model is the best classifier

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import make_pipeline
from sklearn.metrics import accuracy_score, precision_recall_fscore_support
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.svm import SVC
# Load the dataset
df = pd.read_csv('/content/drive/MyDrive/Cleaned_diabetes(1).csv')
# Define features and target
X = df.drop('Outcome', axis=1)
y = df['Outcome']
# Identify categorical columns (assuming all columns other than 'Outcome' are
features)
categorical_cols = X.select_dtypes(include=['object']).columns
# Create a column transformer to apply one-hot encoding to categorical columns
column_transformer = ColumnTransformer([
             ('one_hot_encoder',     OneHotEncoder(handle_unknown='ignore'),
categorical_cols)
], remainder='passthrough')
# Apply the column transformer to the features
X_transformed = column_transformer.fit_transform(X)
# Split the data into training and testing sets
X_train,  X_test,  y_train,  y_test  =  train_test_split(X_transformed,  y,
test_size=0.2, random_state=42)
# Create a pipeline for logistic regression with StandardScaler and set
max_iter and solver
logistic_regression_pipeline = make_pipeline(
    StandardScaler(),
    LogisticRegression(max_iter=1000, solver='liblinear')
)
# Define the models to evaluate
models = {
    'Logistic Regression': logistic_regression_pipeline,
    'Naive Bayes': GaussianNB(),
    'Decision Tree': DecisionTreeClassifier(),
    'Random Forest': RandomForestClassifier(),
    'AdaBoost': AdaBoostClassifier(),
    'Support Vector Machine': make_pipeline(
        StandardScaler(),
        SVC()
    )
}
# Create a dictionary to hold the performance metrics
metrics = {}
# Train and evaluate each model
```

```
for name, model in models.items():
    # Train the model
    model.fit(X_train, y_train)
    # Predict on the test set
    y_pred = model.predict(X_test)
    # Calculate accuracy
    accuracy = accuracy_score(y_test, y_pred)
     # Calculate precision, recall, and F1 score for the positive class
(diabetes = 1)
    precision, recall, f1, _ = precision_recall_fscore_support(y_test, y_pred,
average='binary')
    # Store the metrics in the dictionary
    metrics[name] = {
        'Accuracy': accuracy,
        'Precision': precision,
        'Recall': recall,
        'F1 Score': f1
    }
# Create a report table
report_table = pd.DataFrame(metrics).transpose()
# Print the report table
report_table
```

	Accuracy	Precision	Recall	F1 Score
Logistic Regression	0.753247	0.660377	0.636364	0.648148
Naive Bayes	0.753247	0.644068	0.690909	0.666667
Decision Tree	0.694805	0.562500	0.654545	0.605042
Random Forest	0.746753	0.648148	0.636364	0.642202
AdaBoost	0.759740	0.645161	0.727273	0.683761
Support Vector Machine	0.759740	0.673077	0.636364	0.654206

Cell 9: Find model with best Accuracy and F1 Score

```
# Determine the best algorithm based on the highest accuracy
best_algorithm = report_table.loc[report_table['Accuracy'].idxmax(), :]
print(f"The best algorithm based on accuracy is: {best_algorithm.name}")
# Determine the best algorithm based on the highest F1 Score
best_f1_algorithm = report_table.loc[report_table['F1 Score'].idxmax(), :]
print(f"The best algorithm based on F1 Score is: {best_f1_algorithm.name}")
```

```
The best algorithm based on accuracy is: AdaBoost
The best algorithm based on F1 Score is: AdaBoost
```

Data Modelling (AdaBoost Classifier):

Cell 10: AdaBoost Classifier with Decision Tree Estimator and Naïve Bayes Estimator

```
# Define a dictionary for mapping 'Glucose' levels to numeric values
glucose_mapping = {
    'High': 2,
    'Medium': 1,
    'Low': 0
}
# Convert 'Glucose' column from categorical values to numerical values
df['Glucose'] = df['Glucose'].map(glucose_mapping)
# After mapping the 'Glucose' column, the DataFrame is cleaned and ready for use
new_df = df
# Import necessary libraries
import numpy as np
import pandas as pd
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, f1_score
from  sklearn.model_selection  import  train_test_split,  cross_val_score,
RepeatedStratifiedKFold
from sklearn.preprocessing import StandardScaler
from numpy import mean, std
# Load your cleaned diabetes dataset (assuming it's stored as 'new_df')
X = new_df.iloc[:, :-1]  # All columns except the last column (Outcome)
y = new_df.iloc[:, -1]   # Last column (Outcome)
# Split the data into training and testing sets
X_train,  X_test,  y_train,  y_test  =  train_test_split(X,  y,  test_size=0.3,
random_state=42)
# Standardize the features
scaler = StandardScaler()
X_train_std = scaler.fit_transform(X_train)
X_test_std = scaler.transform(X_test)
# Create the AdaBoost classifier with Decision Tree as the base estimator
abc_dt   =   AdaBoostClassifier(estimator=DecisionTreeClassifier(max_depth=1),
n_estimators=50, random_state=42)
# Create the AdaBoost classifier with Naive Bayes as the base estimator
abc_nb    =    AdaBoostClassifier(estimator=GaussianNB(),    n_estimators=50,
random_state=42)
# Train the model
model_dt = abc_dt.fit(X_train_std, y_train)
# Predict on the test set
```

```
y_pred_dt = model_dt.predict(X_test_std)
# Evaluate the model performance
print("AdaBoost Classifier with Decision Tree Base Estimator:")
print(" Accuracy:", accuracy_score(y_test, y_pred_dt))
print(" F1 Score:", f1_score(y_test, y_pred_dt))
# Train the model
model_nb = abc_nb.fit(X_train_std, y_train)
# Predict on the test set
y_pred_nb = model_nb.predict(X_test_std)
# Evaluate the model performance
print("\nAdaBoost Classifier with Naive Bayes Base Estimator:")
print(" Accuracy:", accuracy_score(y_test, y_pred_nb))
print(" F1 Score:", f1_score(y_test, y_pred_nb))
```

```
AdaBoost Classifier with Decision Tree Base Estimator:
 Accuracy: 0.7316017316017316
 F1 Score: 0.6172839506172839

AdaBoost Classifier with Naive Bayes Base Estimator:
 Accuracy: 0.683982683982684
 F1 Score: 0.5828571428571427
```

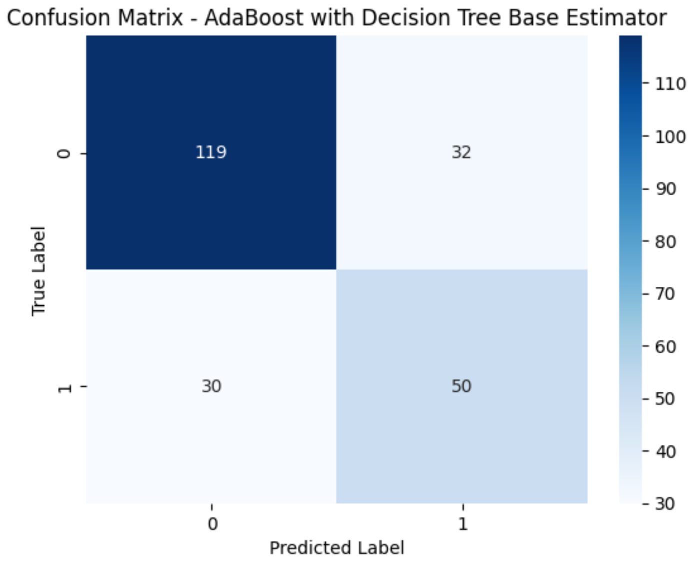
Cell 11: AdaBoost Classifier Cross-Validation

```
# Use cross-validation to evaluate AdaBoost classifier
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
model = AdaBoostClassifier(random_state=42)
n_scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1)
# Report performance
print("\nAdaBoost Classifier Cross-validation:")
print('Accuracy: %.3f (%.3f)' % (mean(n_scores), std(n_scores)))
```

```
 AdaBoost Classifier Cross-validation:
Accuracy: 0.748 (0.044)
```

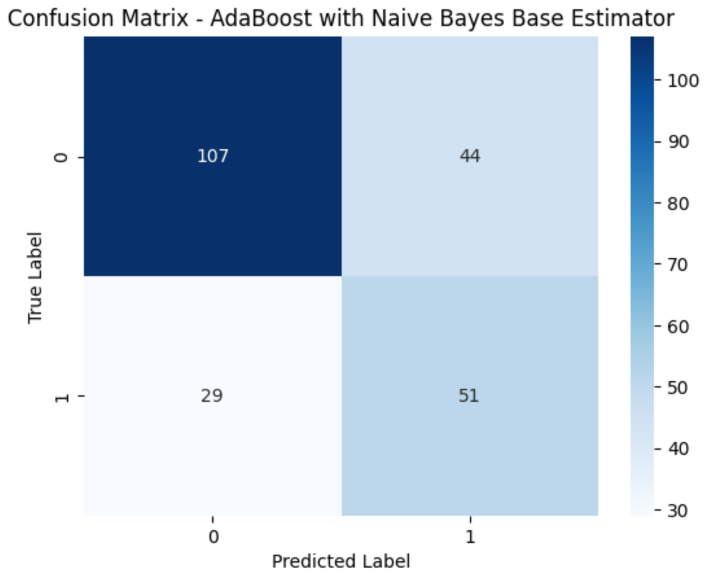
Cell 12: AdaBoost Classifier with Decision Tree Base Estimator

```
from sklearn import metrics
import seaborn as sns
# Plot confusion matrix for Decision Tree base estimator
cm_dt = metrics.confusion_matrix(y_test, y_pred_dt)
sns.heatmap(cm_dt, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix - AdaBoost with Decision Tree Base Estimator')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```



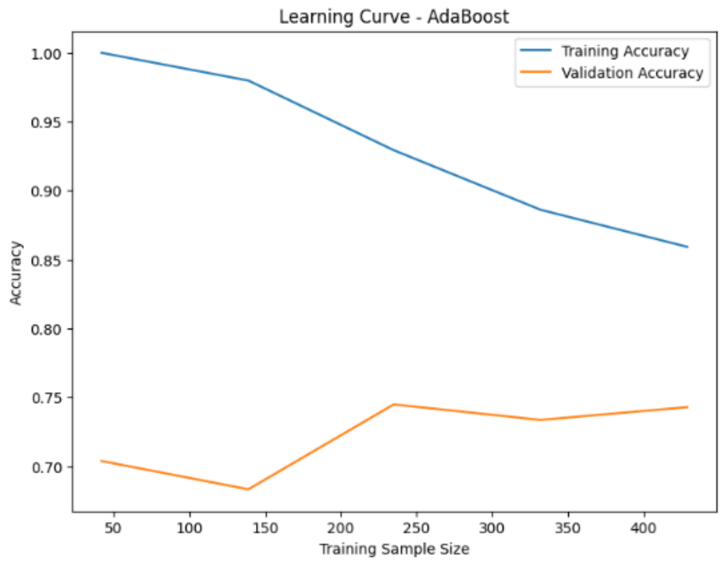
Cell 13: AdaBoost Classifier with Naïve Bayes Base Estimator

```
import seaborn as sns
# Plot confusion matrix for Naive Bayes base estimator
cm_nb = metrics.confusion_matrix(y_test, y_pred_nb)
sns.heatmap(cm_nb, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix - AdaBoost with Naive Bayes Base Estimator')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```



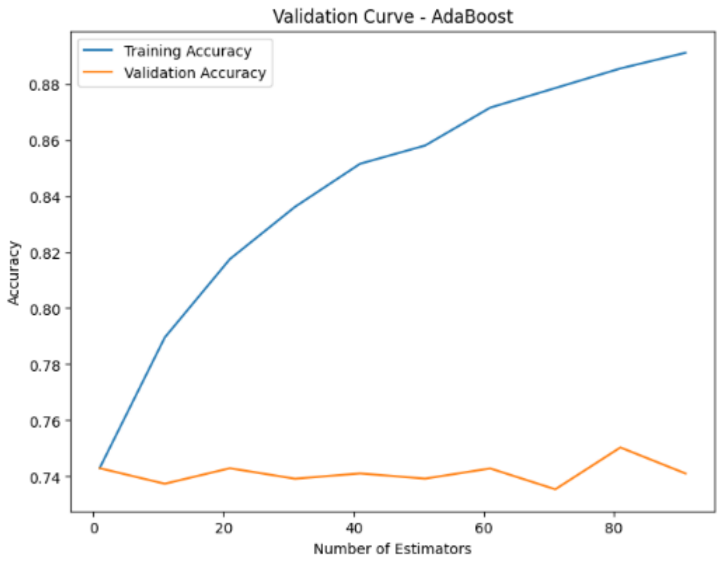
Cell 14: AdaBoost Learning Curve

```
from sklearn.model_selection import learning_curve
def plot_learning_curve(estimator, X, y):
     train_sizes, train_scores, test_scores = learning_curve(estimator, X, y, cv=5, n_jobs=-1)
    train_means = mean(train_scores, axis=1)
    test_means = mean(test_scores, axis=1)
    plt.figure(figsize=(8, 6))
    plt.plot(train_sizes, train_means, label='Training Accuracy')
    plt.plot(train_sizes, test_means, label='Validation Accuracy')
    plt.xlabel('Training Sample Size')
    plt.ylabel('Accuracy')
    plt.title('Learning Curve - AdaBoost')
    plt.legend()
    plt.show()
plot_learning_curve(model_dt, X_train_std, y_train)
```



Cell 15: AdaBoost Validation Curve

```
from sklearn.model_selection import validation_curve
def plot_validation_curve(estimator, X, y):
    param_range = np.arange(1, 100, 10)
        train_scores, test_scores = validation_curve(estimator, X, y, param_name='n_estimators', param_range=param_range, cv=5, scoring='accuracy', n_jobs=-1)
    train_means = mean(train_scores, axis=1)
    test_means = mean(test_scores, axis=1)
    plt.figure(figsize=(8, 6))
    plt.plot(param_range, train_means, label='Training Accuracy')
    plt.plot(param_range, test_means, label='Validation Accuracy')
    plt.xlabel('Number of Estimators')
    plt.ylabel('Accuracy')
    plt.title('Validation Curve - AdaBoost')
    plt.legend()
    plt.show()
plot_validation_curve(AdaBoostClassifier(estimator=DecisionTreeClassifier(max
_depth=1), random_state=42), X_train_std, y_train)
```



Cell 16: ROC-AUC Score

```
# Import necessary libraries
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
# Load your cleaned diabetes dataset
# Assuming `new_df` is the cleaned diabetes dataset DataFrame
X = new_df.iloc[:, :-1]  # Features
y = new_df.iloc[:, -1]   # Target
# Split the data into training and testing sets
X_train,  X_test,  y_train,  y_test  =  train_test_split(X,  y,  test_size=0.3,
random_state=42)
# Standardize the features
scaler = StandardScaler()
X_train_std = scaler.fit_transform(X_train)
X_test_std = scaler.transform(X_test)
# Define the base estimator for AdaBoost
base_estimator = DecisionTreeClassifier(max_depth=1)
# Define the AdaBoost classifier
```

```
ada_model = AdaBoostClassifier(base_estimator=base_estimator, n_estimators=50,
learning_rate=1.0, random_state=42)
# Train the AdaBoost model
ada_model.fit(X_train_std, y_train)
```

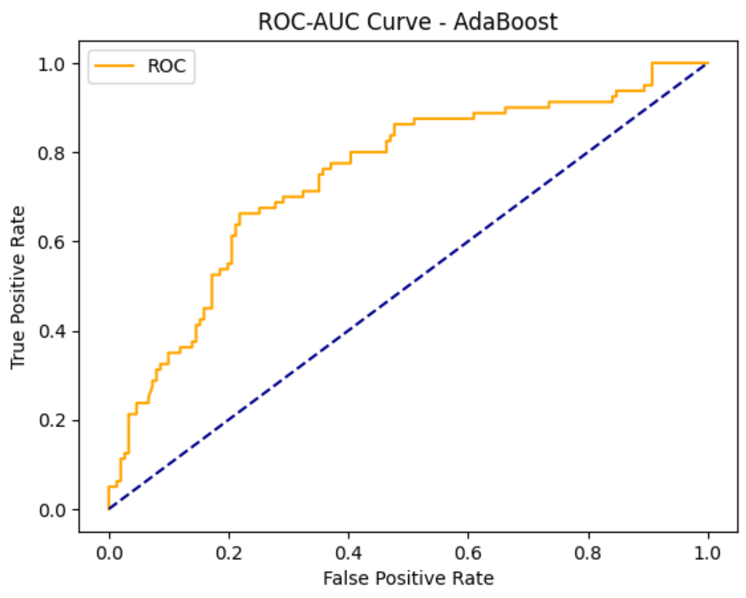
```
▸        AdaBoostClassifier
▸ base_estimator: DecisionTreeClassifier
      ▸ DecisionTreeClassifier
```

```
# Predict probabilities on the test set
y_pred_proba = ada_model.predict_proba(X_test_std)[:, 1]
# Calculate the ROC-AUC score
roc_auc = roc_auc_score(y_test, y_pred_proba)
# Print the ROC-AUC score
print("ROC-AUC score:", roc_auc)
```

```
ROC-AUC score: 0.7443708609271523
```

Cell 17: ROC-AUC Curve

```
from sklearn.metrics import roc_curve
# Plot the ROC-AUC curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
plt.plot(fpr, tpr, color='orange', label='ROC')
plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC-AUC Curve - AdaBoost')
plt.legend()
plt.show()
```



Cell 18: Flask Task (Optional) -> Model made on Google Collab without flask

```
# Import necessary libraries
import numpy as np
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import roc_auc_score
# Load your cleaned diabetes dataset
# Assuming `new_df` is the cleaned diabetes dataset DataFrame
X = new_df.iloc[:, :-1]  # Features
y = new_df.iloc[:, -1]   # Target
# Split the data into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)
# Standardize the features
scaler = StandardScaler()
X_train_std = scaler.fit_transform(X_train)
X_test_std = scaler.transform(X_test)
# Define the base estimator for AdaBoost
base_estimator = DecisionTreeClassifier(max_depth=1)
# Define the AdaBoost classifier
ada_model = AdaBoostClassifier(base_estimator=base_estimator, n_estimators=50,
learning_rate=1.0, random_state=42)
# Train the AdaBoost model
ada_model.fit(X_train_std, y_train)
def predict_outcome(ada_model, scaler):
    # Prompt user for input
    pregnancies = int(input("Enter the number of pregnancies (range: 0 to 17):
"))
    glucose = int(input("Enter glucose level (0 for low, 1 for medium, 2 for
high): "))
    blood_pressure = int(input("Enter blood pressure level (range: 0 to 122):
"))
    skin_thickness = int(input("Enter skin thickness (range: 0 to 99): "))
    insulin = int(input("Enter insulin level (range: 0 to 846): "))
    bmi = float(input("Enter BMI (range: 0 to 67.1): "))
    diabetes_pedigree_function = float(input("Enter diabetes pedigree function
(range: 0.078 to 2.42): "))
    age = int(input("Enter age (range: 21 to 81): "))
    # Combine user input into a DataFrame with feature names
    user_input = pd.DataFrame([[pregnancies, glucose, blood_pressure,
skin_thickness, insulin, bmi, diabetes_pedigree_function, age]],
                                    columns=['Pregnancies', 'Glucose',
'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction',
'Age'])
    # Standardize the user input using the scaler
    features_std = scaler.transform(user_input)
    # Predict the outcome using the AdaBoost model
    outcome = ada_model.predict(features_std)
    # Output the predicted outcome
    if outcome[0] == 0:
        print("The predicted outcome is: 0 (no diabetes)")
    else:
        print("The predicted outcome is: 1 (diabetes)")
# Call the function to predict outcome
predict_outcome(ada_model, scaler)
```

```
Enter the number of pregnancies (range: 0 to 17): 0
Enter glucose level (0 for low, 1 for medium, 2 for high): 0
Enter blood pressure level (range: 0 to 122): 53
Enter skin thickness (range: 0 to 99): 62
Enter insulin level (range: 0 to 846): 521
Enter BMI (range: 0 to 67.1): 21
Enter diabetes pedigree function (range: 0.078 to 2.42): 1
Enter age (range: 21 to 81): 21
The predicted outcome is: 0 (no diabetes)
```

Exploratory Data Analysis (Optional):

Cell 19: Data type, Null count, Number of records of each column and Memory usage

```
   #understand the data type and information about data, including the number of
records in each column, data having null or not null, Data type, the memory
usage of the dataset
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    float64
 1   Glucose                   768 non-null    object 
 2   BloodPressure             768 non-null    float64
 3   SkinThickness             768 non-null    float64
 4   Insulin                   768 non-null    float64
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    float64
 8   Outcome                   768 non-null    int64  
dtypes: float64(7), int64(1), object(1)
memory usage: 54.1+ KB
```

Cell 20: Number of Unique Values

```
#several unique values in each column
df.nunique()
```

```
  Pregnancies                16
Glucose                     3
BloodPressure              46
SkinThickness              50
Insulin                   184
BMI                       247
DiabetesPedigreeFunction  517
Age                        52
Outcome                     2
dtype: int64
```

Cell 21: Print Duplicated Rows

```
df[df.duplicated()]
```

Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome

Cell 22: Discripted Statistics Summary

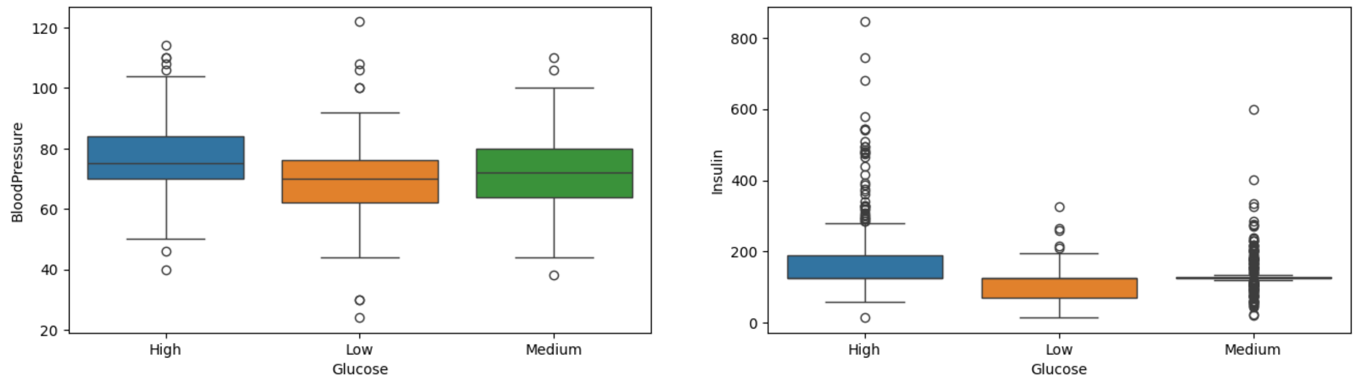
```
df.describe()
```

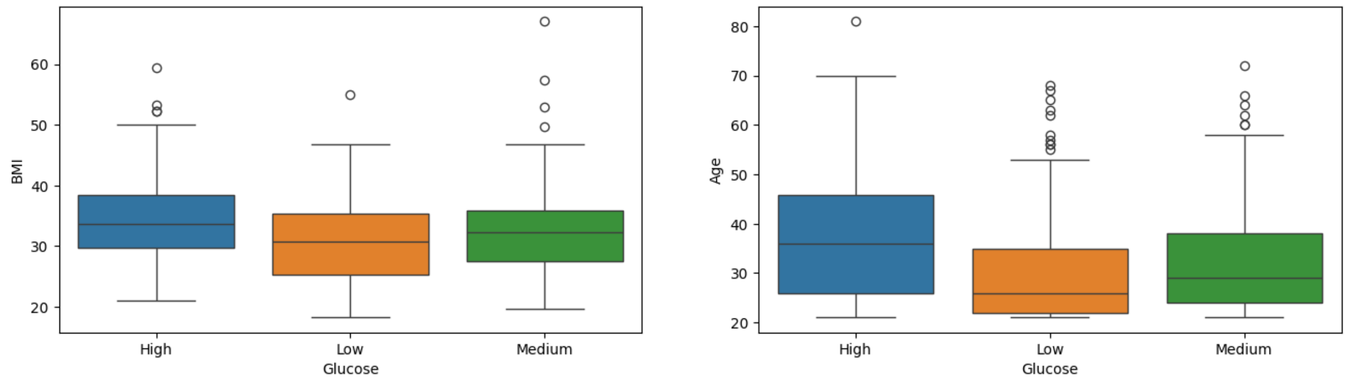
	Pregnancies	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.747396	72.386719	29.108073	141.162760	32.455208	0.471876	33.240885	0.348958
std	3.406986	12.096642	8.791221	86.158087	6.875177	0.331329	11.760232	0.476951
min	0.000000	24.000000	7.000000	14.000000	18.200000	0.078000	21.000000	0.000000
25%	1.000000	64.000000	25.000000	124.250000	27.500000	0.243750	24.000000	0.000000
50%	2.000000	72.000000	29.000000	125.500000	32.300000	0.372500	29.000000	0.000000
75%	6.000000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

Cell 23: Boxplot of Blood pressure, Insulin, BMI and Age with Glucose

```
import seaborn as sns
import matplotlib.pyplot as plt
fig, axes = plt.subplots(2, 2, figsize=(16, 9))
sns.boxplot(y='BloodPressure', x='Glucose', data=df, orient='v', ax=axes[0, 0],
hue='Glucose')
sns.boxplot(y='Insulin', x='Glucose', data=df, orient='v', ax=axes[0, 1],
hue='Glucose')
sns.boxplot(y='BMI', x='Glucose', data=df, orient='v', ax=axes[1, 0],
hue='Glucose')
```

```
sns.boxplot(y='Age', x='Glucose', data=df, orient='v', ax=axes[1, 1],
hue='Glucose')
plt.show()
```



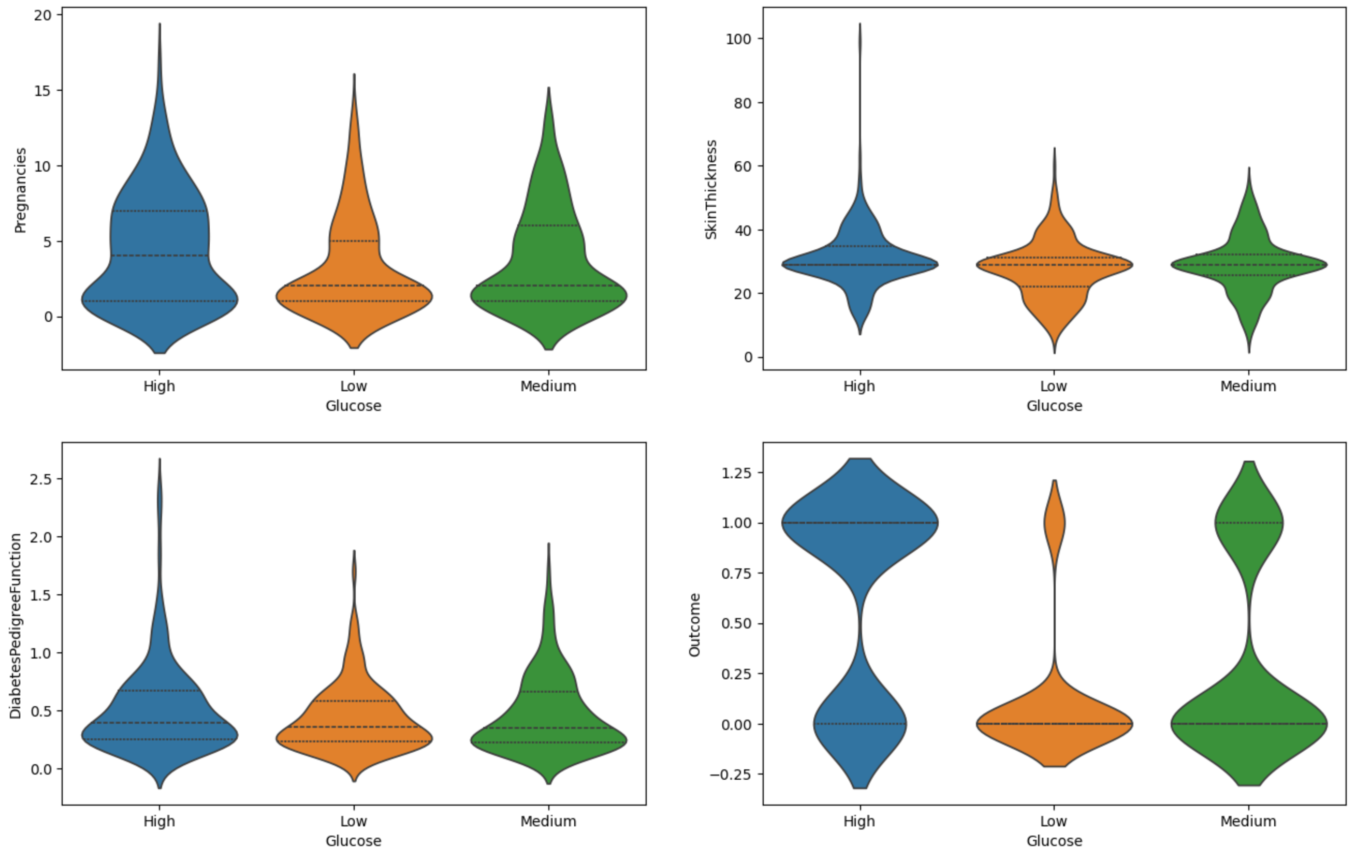


Key take aways from the graph:
Amount of glucose as it changes from low to medium and medium to high we observe that the max value continuously keeps increasing meaning aged people are likely to have high glucose level than the younger ones
From insulin, glucose graph it is clear people with high glucose are definitely more likely to have more doses of insulin than those with lower glucose levels
High glucose means excess carbohydrates and if not proper exercise it might as well get converted to fats and hence we observe BMI is maximum for people with high glucose levels

Cell 24: Violin Plot of Pregnancies, Skin Thickness, Diabetes Pedigree Function and Outcome with Glucose

```
import seaborn as sns
import matplotlib.pyplot as plt
fig, axes = plt.subplots(2, 2, figsize=(16, 10))
sns.violinplot(y='Pregnancies', x='Glucose', data=df, orient='v', ax=axes[0, 0],
hue='Glucose', inner='quartile')
sns.violinplot(y='SkinThickness', x='Glucose', data=df, orient='v', ax=axes[0,
1], hue='Glucose', inner='quartile')
sns.violinplot(y='DiabetesPedigreeFunction', x='Glucose', data=df, orient='v',
ax=axes[1, 0], hue='Glucose', inner='quartile')
```

```
sns.violinplot(y='Outcome', x='Glucose', data=df, orient='v', ax=axes[1, 1],
hue='Glucose', inner='quartile')
plt.show()
```



Cell 25: Mean and Median values for each glucose type

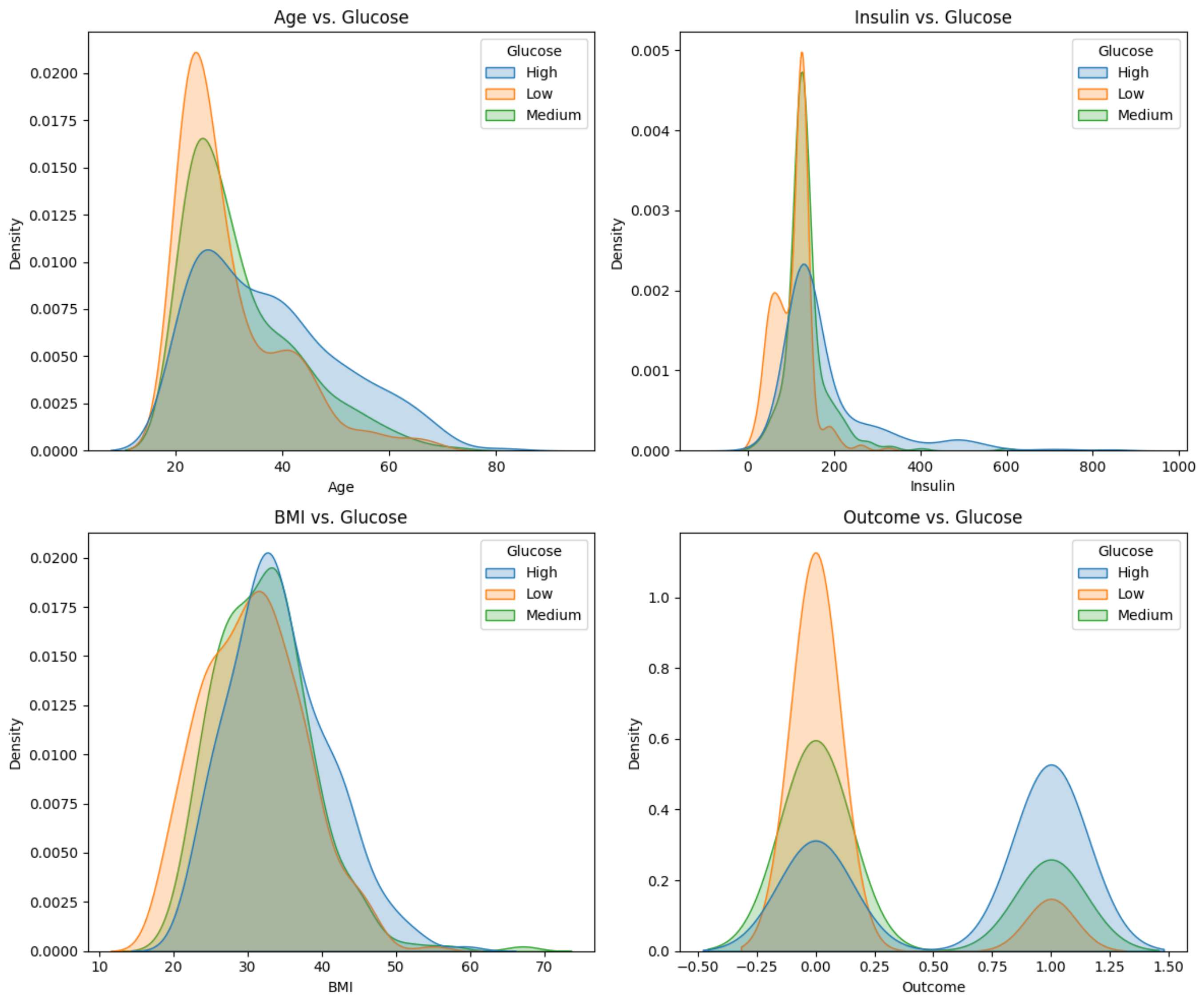
```
#Checking Mean & Median Values for each glucose type
df.groupby('Glucose').agg(['mean', 'median'])
```

	Pregnancies		BloodPressure		SkinThickness		Insulin		BMI		DiabetesPedigreeFunction		Age		Outcome	
	mean	median	mean	median	mean	median	mean	median	mean	median	mean	median	mean	median	mean	median
Glucose																
High	4.356589	4.0	75.782946	75.0	31.023256	29.0	182.558140	125.5	34.303101	33.6	0.514504	0.3985	37.534884	36.0	0.627907	1.0
Low	3.196911	2.0	69.339768	70.0	27.250965	29.0	104.967181	125.5	30.830116	30.8	0.435031	0.3640	29.996139	26.0	0.115830	0.0
Medium	3.689243	2.0	72.039841	72.0	29.055777	29.0	135.962151	125.5	32.232669	32.3	0.466080	0.3490	32.175299	29.0	0.302789	0.0

Cell 26: KDE plots of Age, Insulin, BMI, and Outcome with Glucose

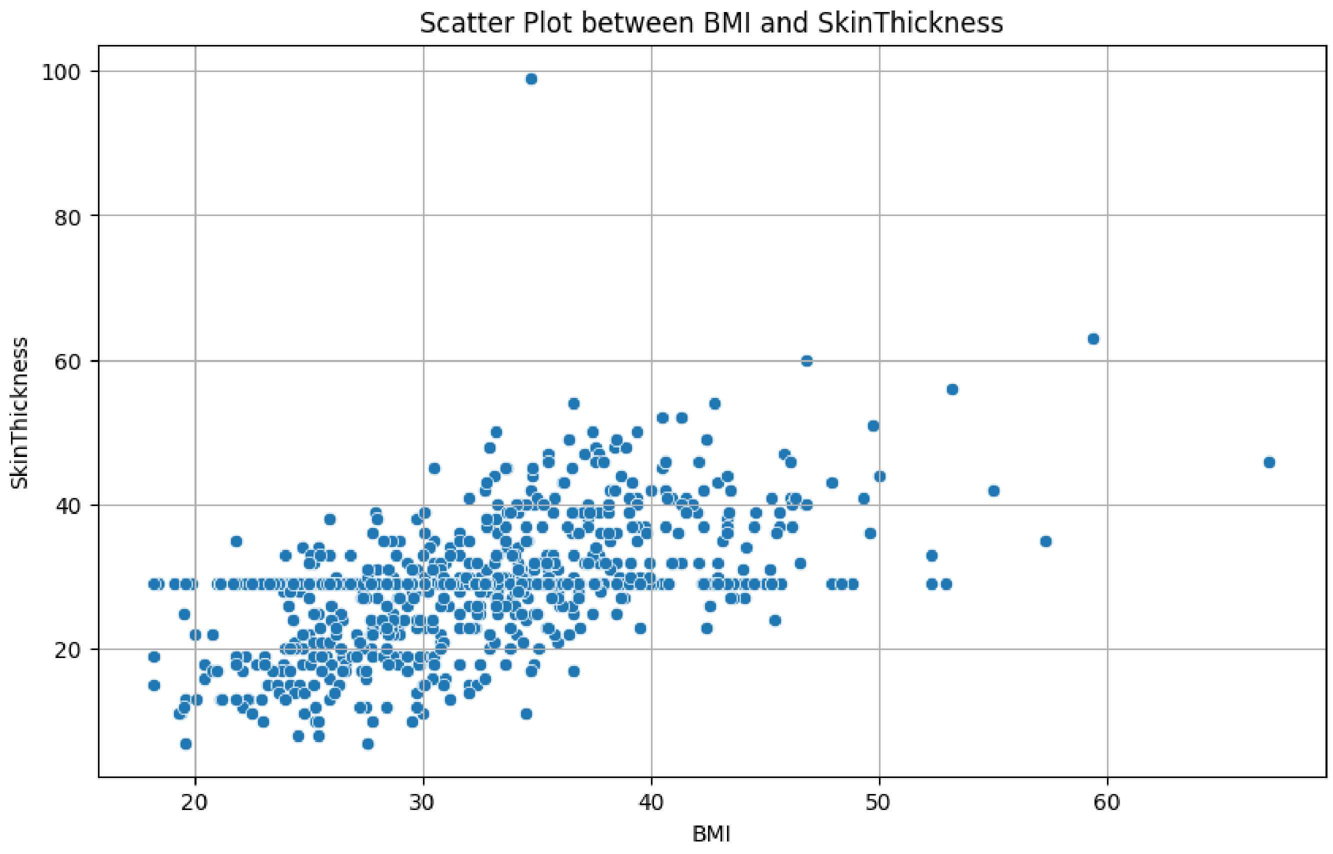
```
import seaborn as sns
import matplotlib.pyplot as plt
# Create a 2x2 grid of subplots
fig, axes = plt.subplots(2, 2, figsize=(12, 10))
# Plot Age vs. Glucose
sns.kdeplot(data=df, x="Age", hue="Glucose", ax=axes[0, 0], fill=True)
```

```
axes[0, 0].set_title("Age vs. Glucose")
# Plot Insulin vs. Glucose
sns.kdeplot(data=df, x="Insulin", hue="Glucose", ax=axes[0, 1], fill=True)
axes[0, 1].set_title("Insulin vs. Glucose")
# Plot BMI vs. Glucose
sns.kdeplot(data=df, x="BMI", hue="Glucose", ax=axes[1, 0], fill=True)
axes[1, 0].set_title("BMI vs. Glucose")
# Plot Outcome vs. Glucose
sns.kdeplot(data=df, x="Outcome", hue="Glucose", ax=axes[1, 1], fill=True)
axes[1, 1].set_title("Outcome vs. Glucose")
# Adjust layout
plt.tight_layout()
# Show the plot
plt.show()
```



Cell 27: Scatter plot between BMI and Skin Thickness

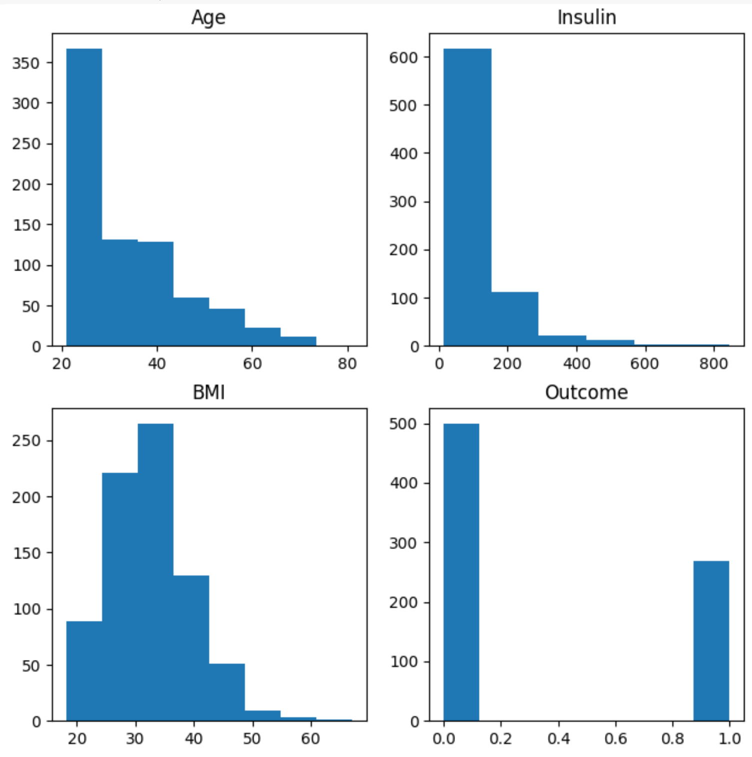
```
import seaborn as sns
import matplotlib.pyplot as plt
# Select the numerical columns for the scatter plot
x_column = 'BMI'  # Choose the first numerical column
y_column = 'SkinThickness'  # Choose the second numerical column
# Create a scatter plot
plt.figure(figsize=(10, 6))
sns.scatterplot(data=df, x=x_column, y=y_column)
plt.title(f'Scatter Plot between {x_column} and {y_column}')
plt.xlabel(x_column)
plt.ylabel(y_column)
plt.grid(True)
plt.show()
```



Cell 28: Histogram Plots of Age, Insulin, BMI and Outcome

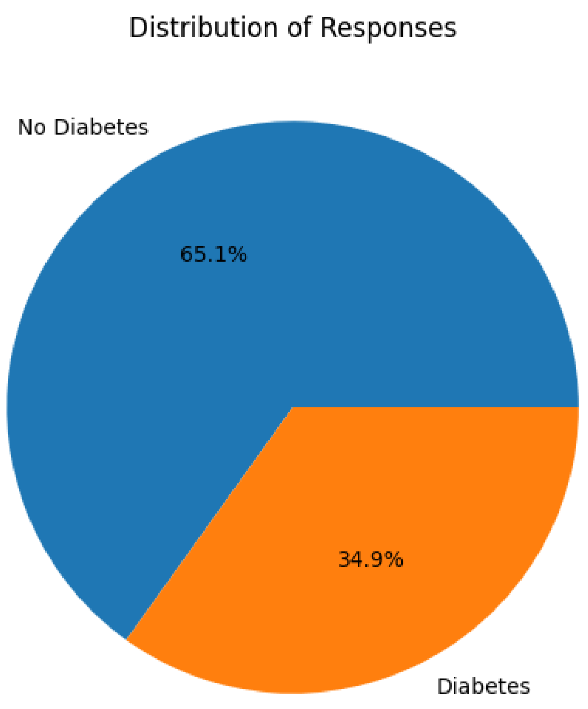
```
figure, ax = plt.subplots(2, 2, figsize=(8,8))
ax[0][0].set_title("Age")
ax[0][0].hist(df['Age'], bins=8)
ax[0][1].set_title("Insulin")
```

```
ax[0][1].hist(df['Insulin'], bins=6);
ax[1][0].set_title("BMI")
ax[1][0].hist(df['BMI'], bins=8)
ax[1][1].set_title("Outcome")
ax[1][1].hist(df['Outcome'], bins=8)
```



Cell 29: Distribution of Outcomes Responses (Diabetes/No Diabetes)

```
# Calculate the counts of each category
counts = df['Outcome'].value_counts()
# Create a pie chart
plt.figure(figsize=(6, 6))
plt.pie(counts, labels=['No Diabetes', 'Diabetes'], autopct='%1.1f%%')
plt.title('Distribution of Responses')
plt.show()
```



Conclusion:

In conclusion, the project aimed to develop a model to predict diabetes outcomes using data from a medical dataset. Various machine learning models, including logistic regression, Naive Bayes, decision trees, random forests, AdaBoost, and support vector machines, were evaluated based on their accuracy, precision, recall, and F1 score. Among the models, AdaBoost with a decision tree base estimator performed the best in terms of both accuracy and F1 score.
Further analysis included visualizing relationships between glucose levels and other features such as age, insulin, BMI, and outcomes, providing insights into potential correlations and patterns within the data.
The project successfully built a prediction model using the AdaBoost algorithm, achieving a ROC-AUC score of 0.7443, indicating the model's ability to distinguish between positive and negative outcomes effectively. The user interface for the model allows for inputting medical data and predicting whether the individual has diabetes.
Overall, the project demonstrates the potential of machine learning in medical diagnosis and highlights the AdaBoost classifier as a promising approach for predicting diabetes outcomes. Future improvements could involve fine-tuning the model parameters and exploring additional data features to enhance prediction accuracy further.