

# **Project Final Report**

## **Diabetes Prediction with Ensemble and Advanced Machine Learning Algorithms**

### **Team members:**

Aparna. Devabhaktuni

Neeraja Reddy. Renati

Sudham. Rajamolla

Sai Spandana. Sabbavarapu

### **Introduction:**

Diabetes is a major health issue, especially among Pima Indian women, and it can lead to severe complications if not managed early. Accurate prediction of diabetes onset based on diagnostic measurements is crucial for timely intervention and treatment. The persistent challenge of diabetes, coupled with its high prevalence among certain populations like the Pima Indians, motivates this exploration of advanced machine learning techniques. This project aims to develop and compare various machine learning models, including ensemble methods and advanced algorithms, to predict the onset of diabetes. Our goal is to contribute to the development of more sophisticated predictive models for healthcare applications, aiming to achieve superior classification accuracy compared to baseline techniques, evaluate and compare a range of machine learning algorithms, and gain insights into the key features and patterns that distinguish diabetic from non-diabetic patients.

### **Related Work:**

Extensive research has been conducted on diabetes prediction using the Pima Indians Diabetes Database, which is known for its robust and well-documented characteristics. The dataset, originally from the National Institute of Diabetes and Digestive and Kidney Diseases, is frequently used to benchmark machine learning models aimed at diagnosing diabetes based on various medical predictor variables. Recent advancements in machine learning (ML) and deep learning (DL) have revolutionized the field of healthcare, particularly in predicting and managing chronic diseases like diabetes. Diabetes, a metabolic disorder characterized by increased blood glucose levels, poses significant risks to various organs, including the kidneys, eyes, heart, nerves, and blood vessels. Early prediction, prognosis, and management of diabetes

are crucial to mitigate these risks and recommend effective treatments. In recent years, ML and DL algorithms have garnered considerable attention for their potential in addressing these objectives. A comprehensive review by Afsaneh et al. (2022) surveyed the landscape of ML and DL models, highlighting their promising outcomes in controlling blood glucose and managing diabetes. Furthermore, individual studies, such as the work by Tasin et al. (2022) and Suchi et al. (2023), have demonstrated the effectiveness of specific ML techniques in diabetes prediction. Tasin et al. (2022) developed an automated diabetes prediction system using XGBoost with the ADASYN approach, while Suchi et al. (2023) focused on feature selection and ensemble learning to achieve high accuracy in diabetes prediction. These studies collectively underscore the importance of leveraging advanced ML algorithms and ensemble techniques for accurate diabetes prediction, essential for effective prognosis and management strategies in clinical settings. Building on these studies, our project will apply and compare a range of machine learning classification algorithms, specifically focusing on K-Nearest Neighbors (KNN), Naive Bayes, Support Vector Machine (SVM), Decision Tree, Random Forest, and Logistic Regression, aiming to identify the most effective model for accurately predicting diabetes onset.

## **Methods:**

The Pima Indians Diabetes Dataset, collected by the National Institute of Diabetes and Digestive and Kidney Diseases, consists of 768 instances from the Pima Indian population. The dataset aims to predict whether a patient has diabetes based on diagnostic measurements. Below is a detailed overview of the dataset:

1. Pregnancies: Number of times the patient has been pregnant.
2. Glucose: Plasma glucose concentration 2 hours in an oral glucose tolerance test.
3. BloodPressure: Diastolic blood pressure (mm Hg).
4. SkinThickness: Triceps skinfold thickness (mm).
5. Insulin: 2-Hour serum insulin ( $\mu$ U/ml).
6. BMI: Body mass index ( $\text{weight in kg} / (\text{height in m})^2$ ).
7. DiabetesPedigreeFunction: A function which scores likelihood of diabetes based on family history.
8. Age: Age of the patient (years).
9. Outcome (target variable): Class variable (0 or 1), where 0 indicates no diabetes and 1 indicates diabetes.

The dataset contains several instances where values are zero for features such as Glucose, BloodPressure, SkinThickness, Insulin, and BMI, which are physiologically impossible and considered as missing values.

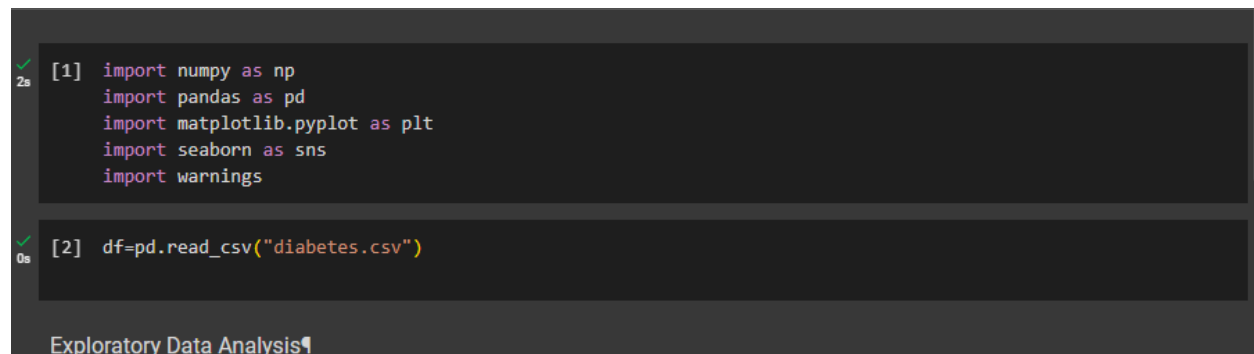
The class distribution shows 768 instances, 500 instances (65.1%) are labeled as non-diabetic (0), while 268 instances (34.9%) are labeled as diabetic (1).

## Implementation:

The core programming language for this project will be Python, using libraries such as Scikit-learn for machine learning tools, Pandas for data manipulation, NumPy for numerical computations, and Matplotlib or Seaborn for creating visualizations.

### 1. Data Collection

The dataset used in this project is the Pima Indians Diabetes Database, sourced from the UCI Machine Learning Repository. This dataset includes several medical predictor variables and one target variable, Outcome. The dataset is loaded using Pandas:



```
[1] import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings

[2] df=pd.read_csv("diabetes.csv")
```

Exploratory Data Analysis

### 2. Data Preprocessing

Preprocessing involves cleaning the data to ensure it is suitable for analysis and modeling. Key steps include:

- **Handling Missing Values:** Certain columns in the dataset have zero values, which are not medically plausible and need to be replaced with more appropriate statistics.
- **Removing Duplicates:** Ensuring there are no duplicate rows in the dataset.
- **Checking for Null Values:** Confirming there are no missing values after initial cleaning.

```
Data Cleaning

[9] df=df.drop_duplicates()

[10] df.isnull().sum()
Pregnancies      0
Glucose           0
BloodPressure     0
SkinThickness     0
Insulin           0
BMI              0
DiabetesPedigreeFunction  0
Age              0
Outcome          0
dtype: int64

[11] print(df[df['BloodPressure']==0].shape[0])
print(df[df['Glucose']==0].shape[0])
print(df[df['SkinThickness']==0].shape[0])
print(df[df['Insulin']==0].shape[0])
print(df[df['BMI']==0].shape[0])

35
5
227
374
11
```

### 3. Exploratory Data Analysis (EDA)

EDA is performed to understand the data distribution, identify patterns, and detect outliers. Key activities include:

## Exploratory Data Analysis

[3] df.head()

	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

Next steps:

[Generate code with df](#)

[View recommended plots](#)

[4] df.shape

(768, 9)

[5] df.columns

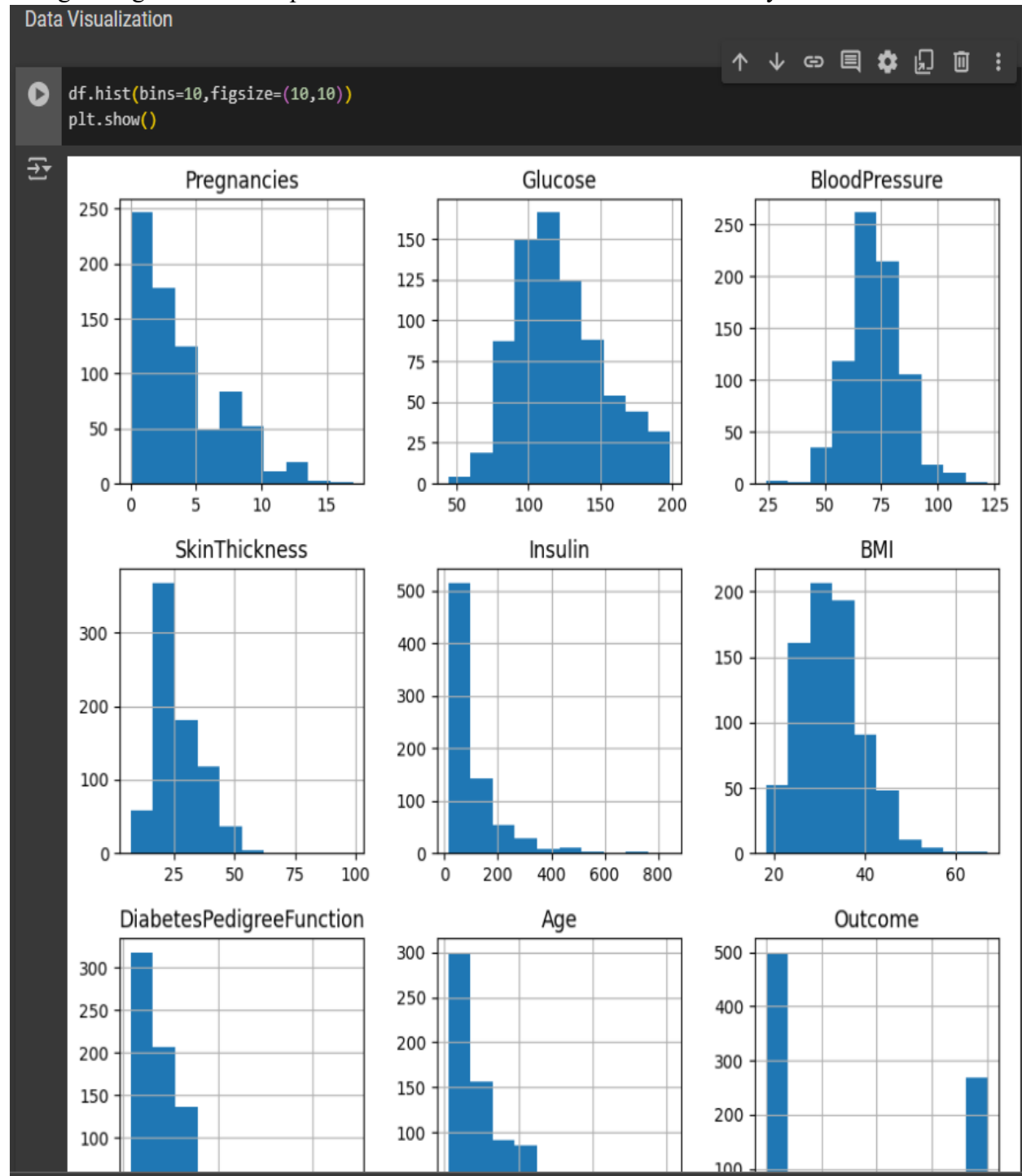
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',  
 'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],  
 dtype='object')

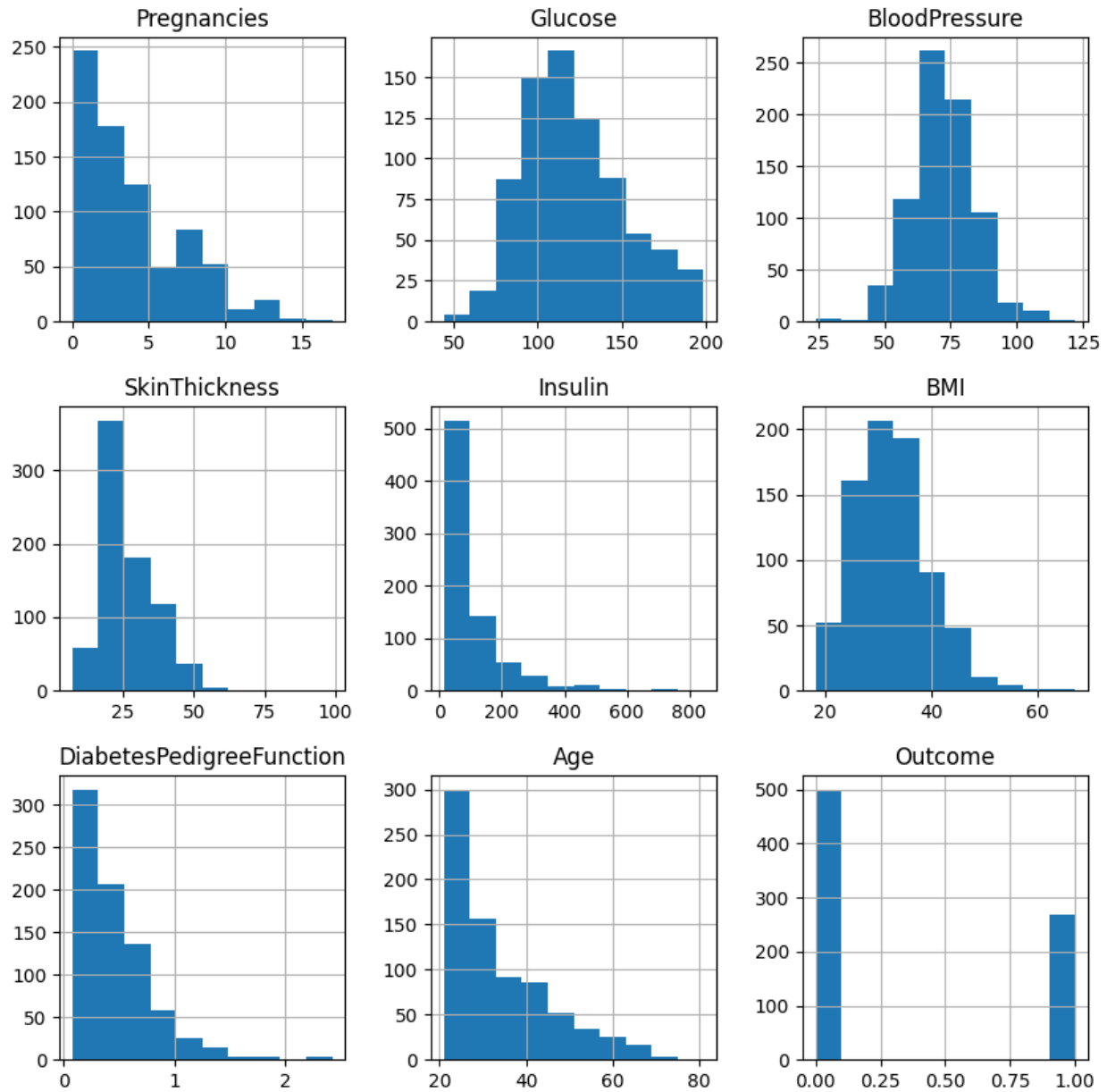
[6] df.dtypes

Pregnancies int64  
Glucose int64  
BloodPressure int64  
SkinThickness int64  
Insulin int64  
BMI float64  
DiabetesPedigreeFunction float64  
Age int64  
Outcome int64  
dtype: object

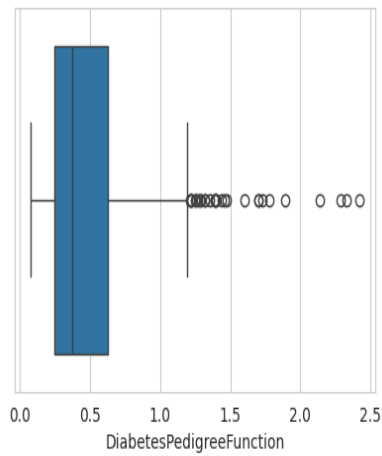
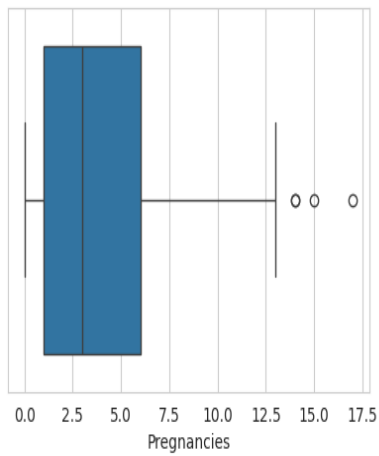
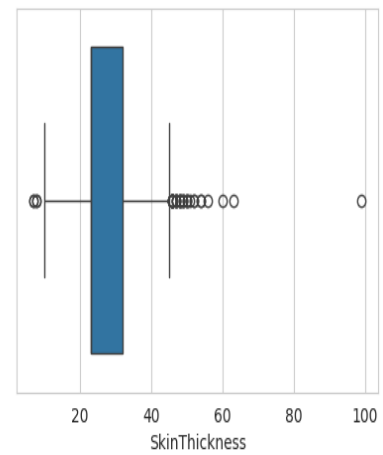
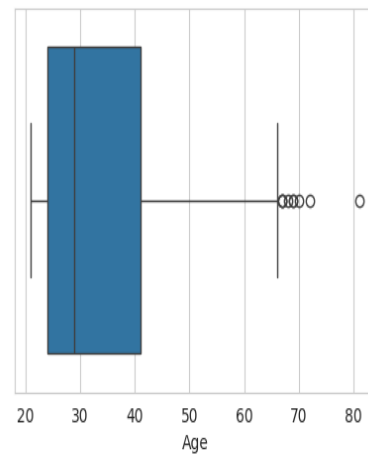
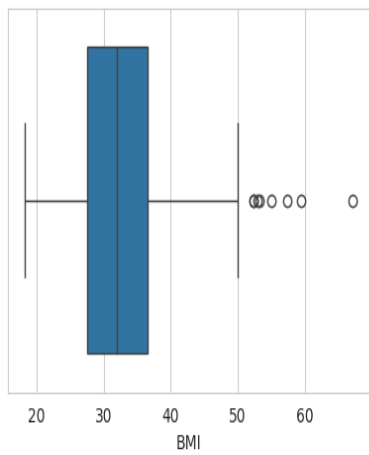
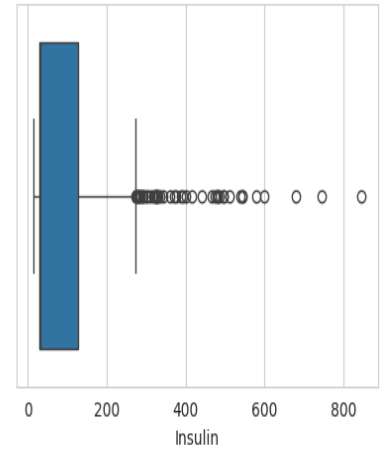
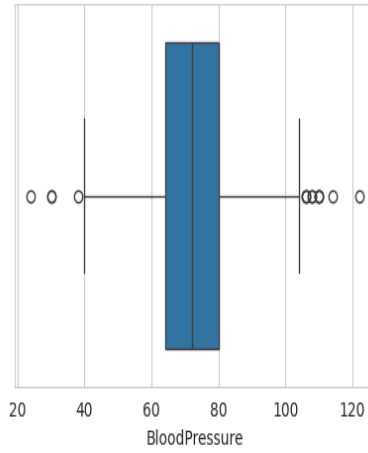
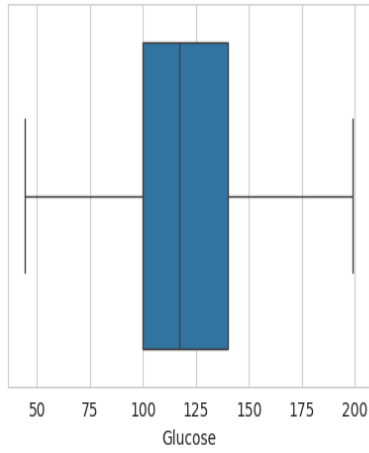
## 4. Data Visualization

Using histograms and boxplots to visualize the distribution and identify outliers in the dataset.



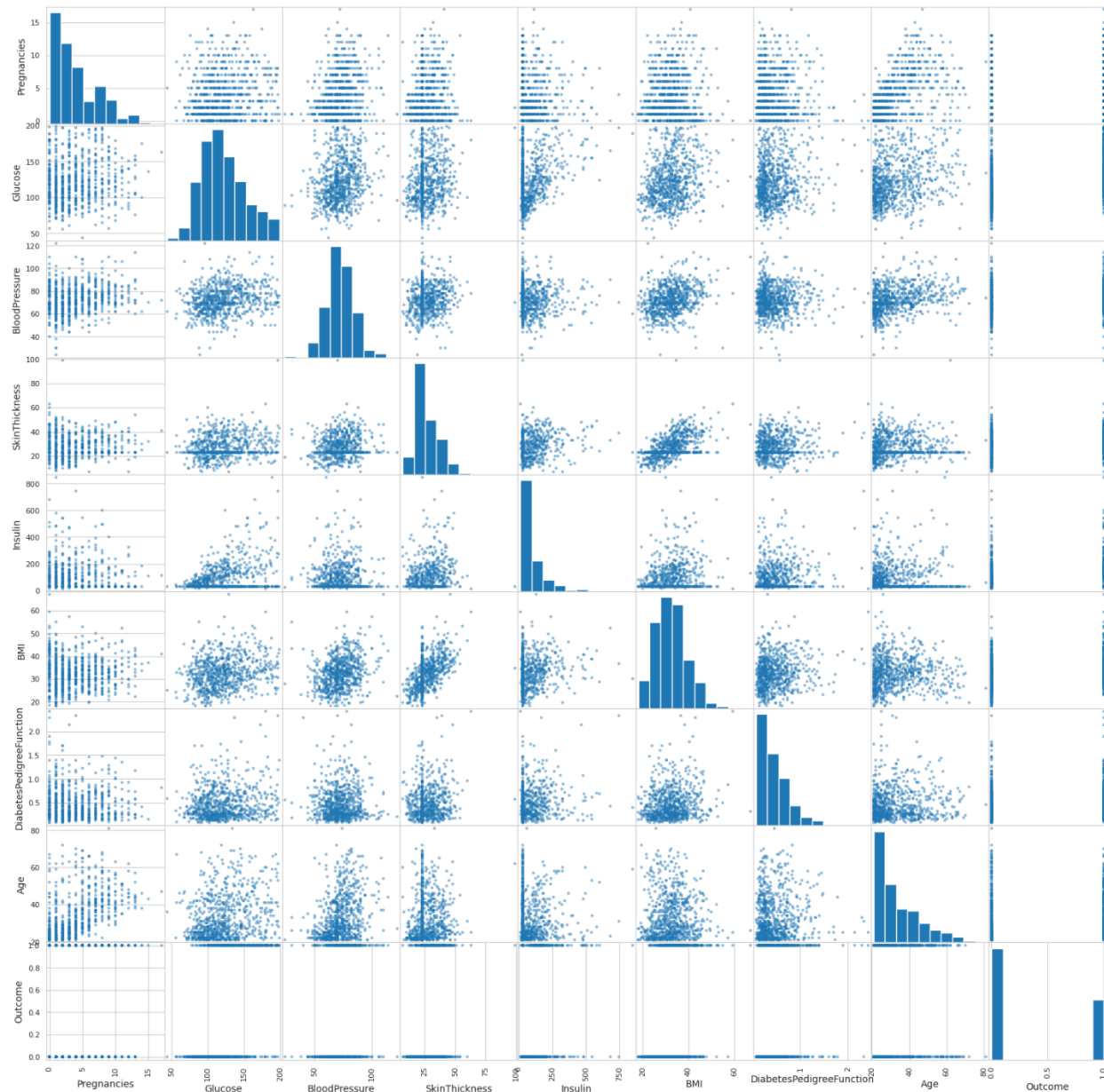


```
plt.figure(figsize=(16,12))
sns.set_style(style='whitegrid')
plt.subplot(3,3,1)
sns.boxplot(x='Glucose',data=df)
plt.subplot(3,3,2)
sns.boxplot(x='BloodPressure',data=df)
plt.subplot(3,3,3)
sns.boxplot(x='Insulin',data=df)
plt.subplot(3,3,4)
sns.boxplot(x='BMI',data=df)
plt.subplot(3,3,5)
sns.boxplot(x='Age',data=df)
plt.subplot(3,3,6)
sns.boxplot(x='SkinThickness',data=df)
plt.subplot(3,3,7)
sns.boxplot(x='Pregnancies',data=df)
plt.subplot(3,3,8)
sns.boxplot(x='DiabetesPedigreeFunction',data=df)
```





**Scatter Matrix:** To understand relationships between features.

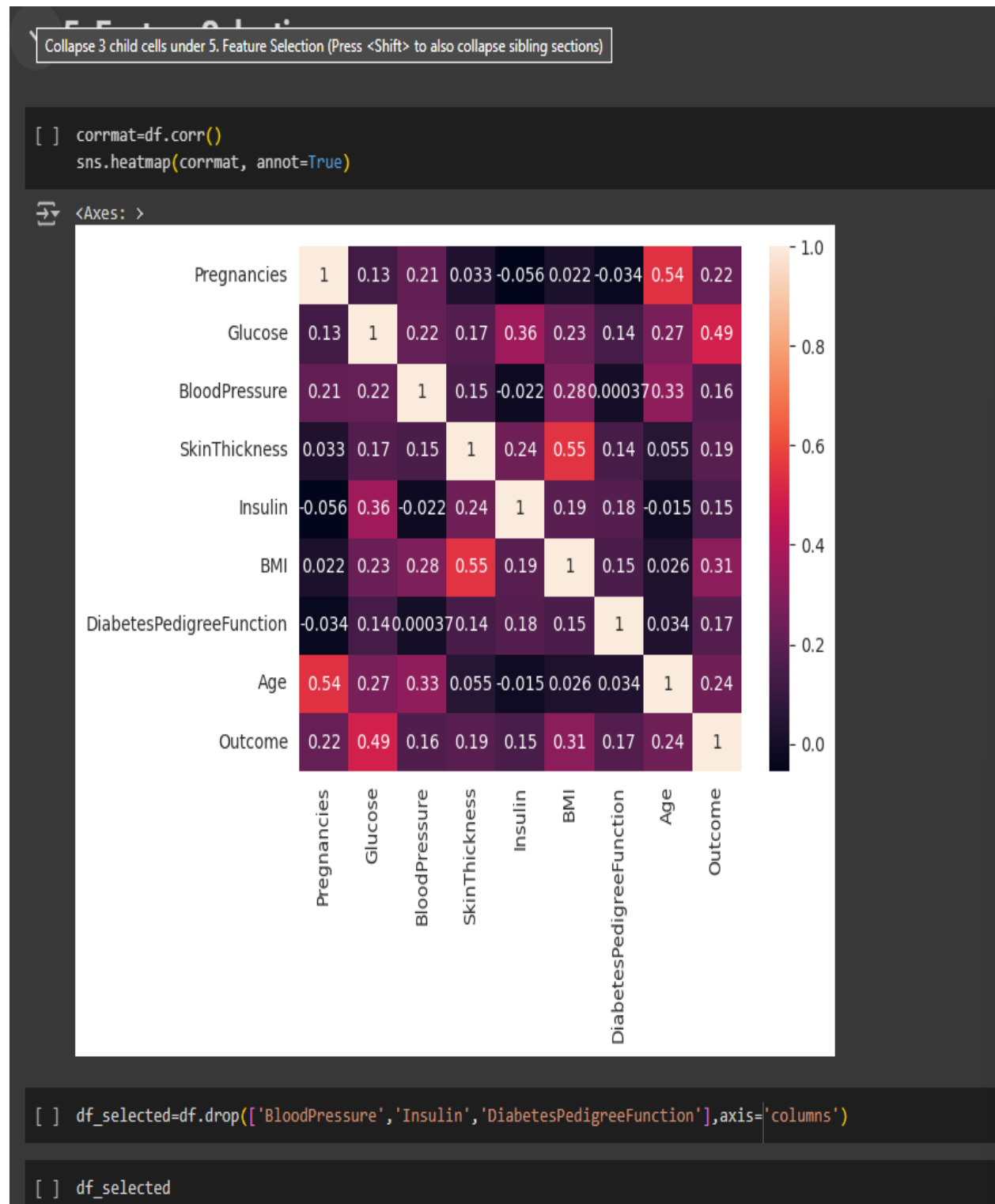


## 5. Data Splitting

The data post preprocessing is ready to be used in our machine learning analysis. For understanding the performance of the model, we require a testing set of data. Therefore, the current dataset is split into two parts: a training set and a test set. The training set is 80% of the data and is used to train machine learning models. The test set is 20% of the data and will be used for model evaluation purposes. The data is divided initially into X and y variables as part of attributes and classes. The Outcome variable is the class variable, and all other variables are predicting variables. The splitting is done using the sklearn library to obtain four variables related to training and testing.

## 6. Feature Selection

Using a correlation matrix, we identify highly correlated features. We drop less important features to reduce dimensionality and improve model performance.




## 7. Handling Outliers

To manage outliers and ensure all features are on a comparable scale, we apply the Quantile Transformer, which transforms features to follow a uniform or normal distribution.

### 6. Handling Outliers

#

```
[ ] from sklearn.preprocessing import QuantileTransformer
x=df_selected
quantile = QuantileTransformer()
X = quantile.fit_transform(x)
df_new=quantile.transform(X)
df_new=pd.DataFrame(X)
df_new.columns =['Pregnancies', 'Glucose','SkinThickness','BMI','Age','Outcome']
df_new.head()
```

 /opt/conda/lib/python3.10/site-packages/sklearn/preprocessing/\_data.py:2627: UserWarning: n\_quantiles (1000) is greater than the number of samples (5). The quantiles will be estimated with a small number of data points, this can lead to unreliable quantiles estimates and a degraded performance.  
warnings.warn(  
/opt/conda/lib/python3.10/site-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, using indices to access the data.  
warnings.warn(

	Pregnancies	Glucose	SkinThickness	BMI	Age	Outcome
0	0.747718	0.810300	0.801825	0.591265	0.889831	1.0
1	0.232725	0.091265	0.644720	0.213168	0.558670	0.0
2	0.863755	0.956975	0.357888	0.077575	0.585398	1.0
3	0.232725	0.124511	0.357888	0.284224	0.000000	0.0
4	0.000000	0.721643	0.801825	0.926988	0.606258	1.0

```
[ ] plt.figure(figsize=(16,12))
sns.set_style(style='whitegrid')
plt.subplot(3,3,1)
sns.boxplot(x=df_new['Glucose'],data=df_new)
plt.subplot(3,3,2)
sns.boxplot(x=df_new['BMI'],data=df_new)
plt.subplot(3,3,3)
sns.boxplot(x=df_new['Pregnancies'],data=df_new)
plt.subplot(3,3,4)
sns.boxplot(x=df_new['Age'],data=df_new)
plt.subplot(3,3,5)
sns.boxplot(x=df_new['SkinThickness'],data=df_new)
```

## 8. Splitting the Data into Features and Target

We split the dataset into features (X) and the target variable (y), where X includes all the columns except the target column (Outcome).

### 7. Split the Data Frame into X and y

```
[ ] target_name='Outcome'  
y= df_new[target_name]  
X=df_new.drop(target_name,axis=1)
```

```
[ ] X.head()
```



	Pregnancies	Glucose	SkinThickness	BMI	Age
0	0.747718	0.810300	0.801825	0.591265	0.889831
1	0.232725	0.091265	0.644720	0.213168	0.558670
2	0.863755	0.956975	0.357888	0.077575	0.585398
3	0.232725	0.124511	0.357888	0.284224	0.000000
4	0.000000	0.721643	0.801825	0.926988	0.606258

```
[ ] y.head()
```



```
0    1.0  
1    0.0  
2    1.0  
3    0.0  
4    1.0  
Name: Outcome, dtype: float64
```

## 9. Train-Test Split

The data is split into training and testing sets, typically in an 80-20 split. This helps in training the model on one subset and testing its performance on another, ensuring it generalizes well to new data.

### 8. TRAIN TEST SPLIT

```
[ ] from sklearn.model_selection import train_test_split
    X_train, X_test, y_train, y_test= train_test_split(X,y,test_size=0.2,random_state=0)
```

```
[ ] X_train.shape,y_train.shape
```

```
↔ ((614, 5), (614,))
```

```
[ ] X_test.shape,y_test.shape
```

```
↔ ((154, 5), (154,))
```

## 9. Model Training

This stage centers on training a diverse set of machine learning models for diabetes prediction using the Scikit-learn library. Our modeling strategy uses a spectrum of classification techniques:

- **Baseline Models:** Provide an initial benchmark for comparison.
  - Logistic Regression (LR)
  - K-Nearest Neighbors (KNN)
  - Decision Tree Classifier (DTC)
- **Established Algorithms:** Train Support Vector Machines (SVM), Random Forest (RF), and other robust models.
- **Ensemble Approaches:** Explore ensemble methods like AdaBoost (ABC), Bagging (BC), Extra Trees (ETC), Gradient Boosting (GBDT), and XGBoost (XGB), with a focus on evaluating XGBoost's potential for superior diabetes detection.

The model's performance will be assessed on the testing set. This will enable us to identify the most effective algorithms and feature engineering combinations for our diabetes prediction system.

### 9.1 K-Nearest Neighbors (KNN)

KNN classifies a data point based on the majority class among its k-nearest neighbors. We perform hyperparameter tuning to find the optimal number of neighbors and other parameters.

## 9.1 K Nearest Neighbours

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import RepeatedStratifiedFold
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import f1_score, precision_score, recall_score
from sklearn.model_selection import GridSearchCV

[ ] knn= KNeighborsClassifier()
n_neighbors = list(range(15,25))
p=[1,2]
weights = ['uniform', 'distance']
metric = ['euclidean', 'manhattan', 'minkowski']

hyperparameters = dict(n_neighbors=n_neighbors, p=p, weights=weights, metric=metric)

cv = RepeatedStratifiedFold(n_splits=10, n_repeats=3, random_state=1)
grid_search = GridSearchCV(estimator=knn, param_grid=hyperparameters, n_jobs=-1, cv=cv, scoring='f1', error_score=0)

[ ] best_model = grid_search.fit(X_train, y_train)

[ ] print('Best leaf_size:', best_model.best_estimator_.get_params()['leaf_size'])
print('Best p:', best_model.best_estimator_.get_params()['p'])
print('Best n_neighbors:', best_model.best_estimator_.get_params()['n_neighbors'])

Best leaf_size: 30
Best p: 1
Best n_neighbors: 19

[ ] knn_pred = best_model.predict(X_test)

[ ] print("Classification Report is:\n", classification_report(y_test, knn_pred))
print("\n F1:\n", f1_score(y_test, knn_pred))
print("\n Precision score is:\n", precision_score(y_test, knn_pred))
print("\n Recall score is:\n", recall_score(y_test, knn_pred))
print("\n Confusion Matrix:\n")
sns.heatmap(confusion_matrix(y_test, knn_pred))

Classification Report is:
precision    recall  f1-score   support

   0.0      0.85      0.88      0.86      187
   1.0      0.78      0.64      0.67       47

 accuracy      0.77      0.76      0.81      154
  macro avg      0.77      0.76      0.76      154
 weighted avg      0.80      0.81      0.80      154

F1:
0.6666666666666666

Precision score is:
0.6976744188946512

Recall score is:
0.6382978723404256

Confusion Matrix:
```

## 9.2 Naive Bayes

Naive Bayes uses Bayes' theorem to make predictions, assuming independence between features. We tune the smoothing parameter to improve performance.

### 9.2 Naive Bayes :-

```
[ ] from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import GridSearchCV

param_grid_nb = {
    'var_smoothing': np.logspace(0, -2, num=100)
}
nbModel_grid = GridSearchCV(estimator=GaussianNB(), param_grid=param_grid_nb, verbose=1, cv=10, n_jobs=-1)

[ ] best_model= nbModel_grid.fit(X_train, y_train)

Fitting 10 folds for each of 100 candidates, totalling 1000 fits

9.2 Naive Bayes :-

[ ] nb_pred=best_model.predict(X_test)

[ ] print("Classification Report is:\n", classification_report(y_test, nb_pred))
print("\n F1:\n", f1_score(y_test, nb_pred))
print("\n Precision score is:\n", precision_score(y_test, nb_pred))
print("\n Recall score is:\n", recall_score(y_test, nb_pred))
print("\n Confusion Matrix:\n")
sns.heatmap(confusion_matrix(y_test, nb_pred))

Classification Report is:
precision    recall  f1-score   support

   0.0      0.81      0.87      0.84      187
   1.0      0.64      0.53      0.58       47

 accuracy      0.72      0.70      0.77      154
  macro avg      0.72      0.70      0.71      154
 weighted avg      0.76      0.77      0.76      154

F1:
0.5813953488372893

Precision score is:
0.6418256418256411

Recall score is:
0.5319148936170213

Confusion Matrix:
```

## 9.3 Support Vector Machine (SVM)

SVM finds a hyperplane that best separates classes. We tune parameters like the kernel type, penalty parameter (C), and gamma.

```
9.3 Support Vector Machine :-

from sklearn.model_selection import RepeatedStratifiedFold
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import f1_score, precision_score, recall_score

[ ] model = SVC()
kernel = ['poly', 'rbf', 'sigmoid']
C = [50, 10, 1.0, 0.1, 0.01]
gamma = ['scale']

[ ] grid = dict(kernel=kernel, C=C, gamma=gamma)
cv = RepeatedStratifiedFold(n_splits=10, n_repeats=3, random_state=1)
grid_search = GridSearchCV(estimator=model, param_grid=grid, n_jobs=-1, cv=cv, scoring='f1', error_score=0)

[ ] grid_result = grid_search.fit(X, y)

[ ] svm_pred=grid_result.predict(X_test)

[ ] print("Classification Report is:\n",classification_report(y_test,svm_pred))
print("\n F1:\n",f1_score(y_test,svm_pred))
print("\n Precision score is:\n",precision_score(y_test,svm_pred))
print("\n Recall score is:\n",recall_score(y_test,svm_pred))
print("\n Confusion Matrix:\n")
sns.heatmap(confusion_matrix(y_test,svm_pred))
```

	precision	recall	f1-score	support
0.0	0.86	0.89	0.88	107
1.0	0.73	0.68	0.70	47
accuracy			0.82	154
macro avg	0.80	0.78	0.79	154
weighted avg	0.82	0.82	0.82	154

F1:  
0.6666666666666666  
Precision score is:  
0.697674418040512  
Recall score is:  
0.6382978723404256  
Confusion Matrix:

## 9.4 Decision Tree

Decision Tree splits the data into branches to make decisions. We tune parameters like max depth and min samples per leaf to prevent overfitting.

```
9.4 Decision Tree

[ ] from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import f1_score, precision_score, recall_score
from sklearn.model_selection import GridSearchCV
dt = DecisionTreeClassifier(random_state=42)

[ ] params = {
    'max_depth': [5, 10, 20, 25],
    'min_samples_leaf': [10, 20, 50, 100, 120],
    'criterion': ['gini', 'entropy']
}

[ ] grid_search = GridSearchCV(estimator=dt,
                             param_grid=params,
                             cv=4, n_jobs=-1, verbose=1, scoring = "accuracy")

[ ] best_model=grid_search.fit(X_train, y_train)

Fitting 4 folds for each of 40 candidates, totalling 160 fits

[ ] dt_pred=best_model.predict(X_test)

[ ] print("Classification Report is:\n",classification_report(y_test,dt_pred))
print("\n F1:\n",f1_score(y_test,dt_pred))
print("\n Precision score is:\n",precision_score(y_test,dt_pred))
print("\n Recall score is:\n",recall_score(y_test,dt_pred))
print("\n Confusion Matrix:\n")
sns.heatmap(confusion_matrix(y_test,dt_pred))
```

	precision	recall	f1-score	support
0.0	0.80	0.94	0.86	107
1.0	0.78	0.45	0.57	47
accuracy			0.79	154
macro avg	0.79	0.70	0.72	154
weighted avg	0.79	0.79	0.77	154

F1:  
0.5675675675675675  
Precision score is:  
0.7777777777777778  
Recall score is:  
0.446205186323785  
Confusion Matrix:

## 9.5 Random Forest

Random Forest builds multiple decision trees and averages their predictions. We tune the number of trees (estimators) and the maximum features to consider for splits.

```
9.5 Random Forest :-  
[ ] from sklearn.ensemble import RandomForestClassifier  
[ ] from sklearn.metrics import classification_report, confusion_matrix  
[ ] from sklearn.metrics import f1_score, precision_score, recall_score  
[ ] from sklearn.model_selection import RepeatedStratifiedKFold  
[ ] from sklearn.model_selection import GridSearchCV  
  
[ ] model = RandomForestClassifier()  
[ ] n_estimators = [1000]  
[ ] max_features = ['sqrt', 'log2']  
  
[ ] grid = dict(n_estimators=n_estimators, max_features=max_features)  
[ ] cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)  
[ ] grid_search = GridSearchCV(estimator=model, param_grid=grid, n_jobs=-1, cv=cv, scoring='accuracy', error_score=0)  
  
[ ] best_model = grid_search.fit(X_train, y_train)  
  
[ ] rf_pred = best_model.predict(X_test)  
  
[ ] print("Classification Report is:\n", classification_report(y_test, rf_pred))  
[ ] print("\n F1:\n", f1_score(y_test, knn_pred))  
[ ] print("\n Precision score is:\n", precision_score(y_test, knn_pred))  
[ ] print("\n Recall score is:\n", recall_score(y_test, knn_pred))  
[ ] print("\n Confusion Matrix:\n")  
[ ] sns.heatmap(confusion_matrix(y_test, rf_pred))
```

Classification Report is:				
	precision	recall	f1-score	support
0.0	0.86	0.84	0.85	187
1.0	0.65	0.68	0.67	47
accuracy			0.79	154
macro avg	0.76	0.76	0.76	154
weighted avg	0.79	0.79	0.79	154

```
F1:  
0.6666666666666666  
  
Precision score is:  
0.6976744186046512  
  
Recall score is:  
0.6382978723404256  
  
Confusion Matrix:
```

## 9.6 Logistic Regression

Logistic Regression models the probability of a binary outcome. We tune the regularization strength (C) and the type of regularization (penalty).



## 9.6 Logistic Regression:-

```
[ ] from sklearn.linear_model import LogisticRegression
    from sklearn.metrics import classification_report, confusion_matrix
    from sklearn.metrics import f1_score, precision_score, recall_score, accuracy_score
```

```
[ ] reg = LogisticRegression()
    reg.fit(X_train, y_train)
```

```
LogisticRegression
LogisticRegression()
```

```
[ ] lr_pred = reg.predict(X_test)
```

```
[ ] print("Classification Report is:\n", classification_report(y_test, lr_pred))
    print("\n F1:\n", f1_score(y_test, lr_pred))
    print("\n Precision score is:\n", precision_score(y_test, lr_pred))
    print("\n Recall score is:\n", recall_score(y_test, lr_pred))
    print("\n Confusion Matrix:\n")
    sns.heatmap(confusion_matrix(y_test, lr_pred))
```

```
Classification Report is:
precision    recall    f1-score   support

   0.0        0.83    0.89    0.86     107
   1.0        0.69    0.57    0.63      47

 accuracy          0.79    0.79    0.79     154
 macro avg          0.76    0.73    0.74     154
 weighted avg          0.79    0.79    0.79     154

F1:
0.627906976744186

Precision score is:
0.6923076923076923

Recall score is:
0.574468085106383

Confusion Matrix:
```

## 10. Results and discussion:

We compared the performance of various machine learning models for predicting diabetes in the Pima Indians dataset, using key metrics such as F1 Score, Precision, Recall, and Area Under the ROC Curve (AUC). These metrics provide insights into the models' ability to correctly identify positive cases (individuals with diabetes) while minimizing false positives and false negatives. The table below summarizes the performance of the evaluated models across these metrics:

Model	F1 Score	Precision	Recall	AUC
Logistic Regression	0.627	0.692	0.574	0.86
K-Nearest Neighbors	0.666	0.697	0.638	0.86
Naive Bayes	0.581	0.641	0.531	0.86
Support Vector Machine	0.666	0.697	0.638	0.91
Decision Tree	0.567	0.777	0.446	0.86
Random Forest	0.666	0.697	0.638	0.86

The Support Vector Machine (SVM) model achieved the highest AUC of 0.91, indicating excellent overall performance in separating diabetic and non-diabetic cases. It also shared the highest F1 Score (0.666), Precision (0.697), and Recall (0.638) with the K-Nearest Neighbors and Random Forest models.

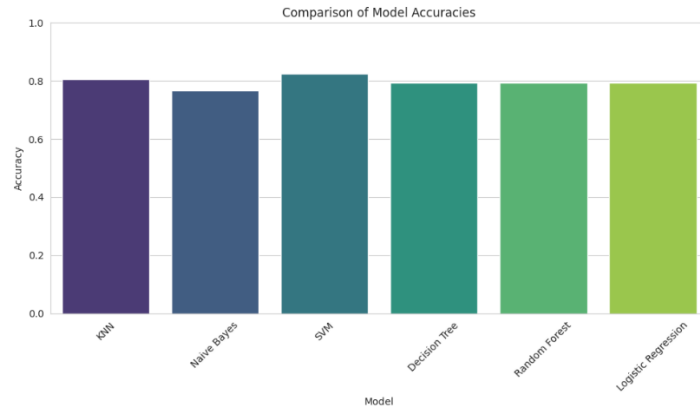
The K-Nearest Neighbors, Random Forest, and Logistic Regression models demonstrated a respectable AUC of 0.86, suggesting good overall performance in distinguishing between the classes. However, the K-Nearest Neighbors and Random Forest models outperformed Logistic Regression in terms of F1 Score, Precision, and Recall.

The Decision Tree model had the highest Precision of 0.777, indicating its strength in minimizing false positives. However, its low Recall of 0.446 and F1 Score of 0.567 suggest that it may struggle to identify a significant portion of true positive cases accurately.

The Naive Bayes model displayed the lowest performance among the evaluated models based on the F1 Score (0.581), Precision (0.641), and Recall (0.531) metrics.

Based on the metrics, including the AUC, the Support Vector Machine emerges as the top-performing model for this diabetes prediction task. It achieves the highest AUC of 0.91, indicating excellent overall performance in distinguishing between diabetic and non-diabetic cases, while also maintaining high F1 Score, Precision, and Recall.

It is important to note that model selection should also consider factors such as interpretability, computational efficiency, and robustness to overfitting, in addition to the performance metrics. Furthermore, if the dataset is imbalanced or if there are specific business requirements, the model choice may need to be adjusted accordingly.



```
knn_proba = best_model.predict_proba(X_test)[:, 1]
nb_proba = best_model.predict_proba(X_test)[:, 1]
svm_proba = grid_result.decision_function(X_test)
dt_proba = best_model.predict_proba(X_test)[:, 1]
rf_proba = best_model.predict_proba(X_test)[:, 1]
lr_proba = reg.predict_proba(X_test)[:, 1]

knn_accuracy = accuracy_score(y_test, knn_pred)
nb_accuracy = accuracy_score(y_test, nb_pred)
svm_accuracy = accuracy_score(y_test, svm_pred)
dt_accuracy = accuracy_score(y_test, dt_pred)
rf_accuracy = accuracy_score(y_test, rf_pred)
lr_accuracy = accuracy_score(y_test, lr_pred)

model_accuracies = {
    'KNN': knn_accuracy,
    'Naive Bayes': nb_accuracy,
    'SVM': svm_accuracy,
    'Decision Tree': dt_accuracy,
    'Random Forest': rf_accuracy,
    'Logistic Regression': lr_accuracy
}

accuracy_df = pd.DataFrame(list(model_accuracies.items()), columns=['Model', 'Accuracy'])

plt.figure(figsize=(10, 6))
sns.barplot(x='Model', y='Accuracy', data=accuracy_df, palette='viridis')

plt.xlabel('Model')
plt.ylabel('Accuracy')
plt.title('Comparison of Model Accuracies')
plt.xticks(rotation=45)
plt.ylim(0, 1)

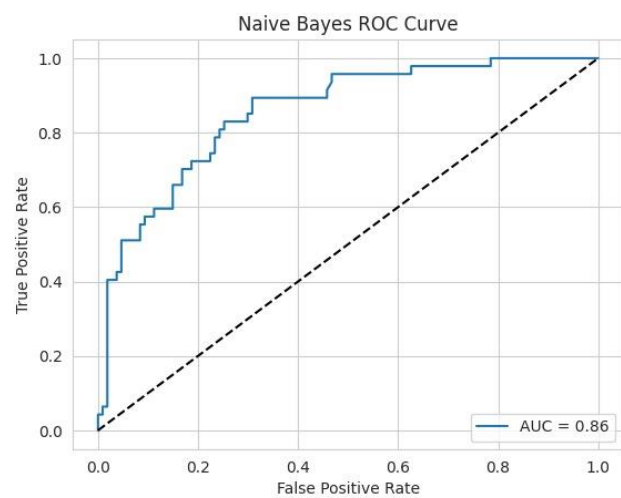
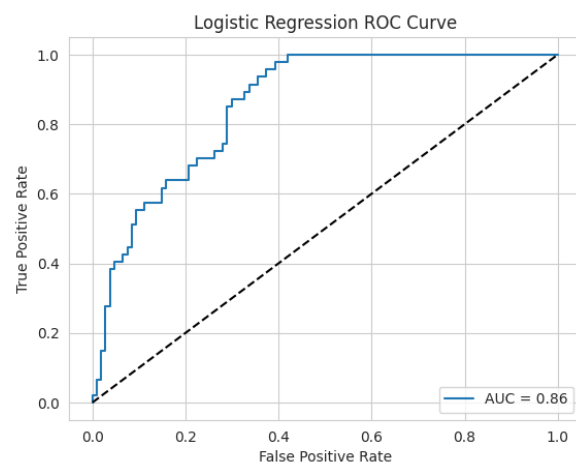
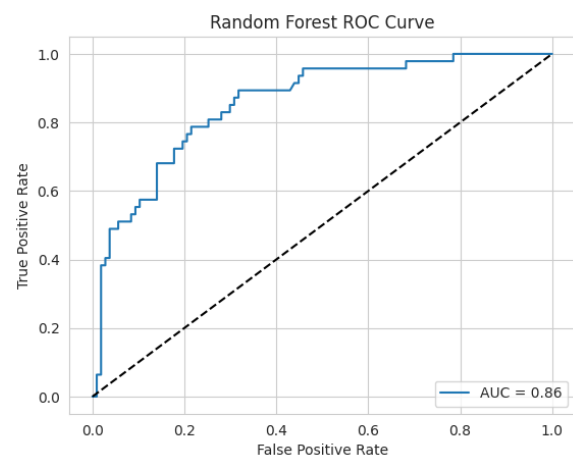
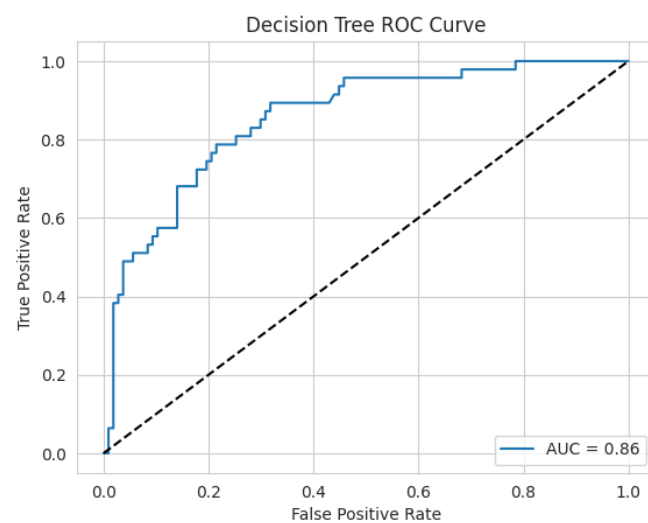
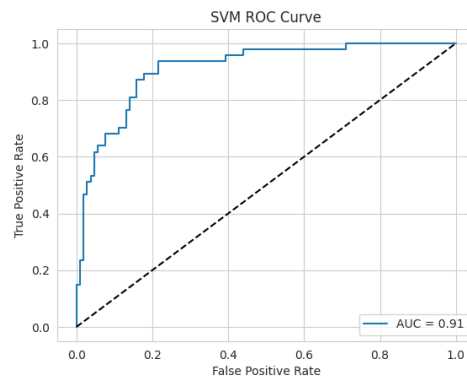
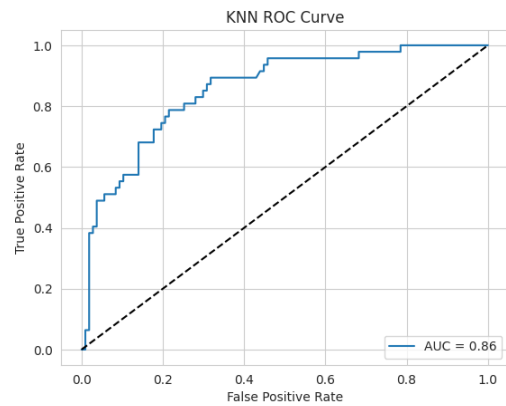
plt.tight_layout()
plt.show()

# ROC Curve
def plot_roc_curve(y_true, y_pred_proba, title):
    fpr, tpr, _ = roc_curve(y_true, y_pred_proba)
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, label=f'AUC = {roc_auc:.2f}')
    plt.plot([0, 1], [0, 1], 'k--')
    plt.title(title)
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.legend(loc='lower right')
    plt.show()

plot_roc_curve(y_test, knn_proba, 'KNN ROC Curve')
plot_roc_curve(y_test, nb_proba, 'Naive Bayes ROC Curve')
plot_roc_curve(y_test, svm_proba, 'SVM ROC Curve')
plot_roc_curve(y_test, dt_proba, 'Decision Tree ROC Curve')
plot_roc_curve(y_test, rf_proba, 'Random Forest ROC Curve')
plot_roc_curve(y_test, lr_proba, 'Logistic Regression ROC Curve')

# Precision-Recall Curve
def plot_precision_recall_curve(y_true, y_pred_proba, title):
    precision, recall, _ = precision_recall_curve(y_true, y_pred_proba)
    plt.plot(recall, precision, marker='.')
    plt.title(title)
    plt.xlabel('Recall')
    plt.ylabel('Precision')
    plt.show()

plot_precision_recall_curve(y_test, knn_proba, 'KNN Precision-Recall Curve')
plot_precision_recall_curve(y_test, nb_proba, 'Naive Bayes Precision-Recall Curve')
plot_precision_recall_curve(y_test, svm_proba, 'SVM Precision-Recall Curve')
plot_precision_recall_curve(y_test, dt_proba, 'Decision Tree Precision-Recall Curve')
plot_precision_recall_curve(y_test, rf_proba, 'Random Forest Precision-Recall Curve')
```



**Conclusion:**

Our project evaluated various machine learning models for diabetes prediction using the Pima Indians Diabetes Database. The Support Vector Machine (SVM) achieved the best performance with an AUC of 0.91, excelling in F1 Score, Precision, and Recall. K-Nearest Neighbors (KNN) and Random Forest also performed well, while Naive Bayes showed the lowest performance.

**Limitations:**

Advanced models like SVM, despite their accuracy, are less interpretable than simpler models, posing challenges for clinical application and the models were trained and tested on a specific dataset, raising concerns about their generalizability to other populations.

**Future work:**

Future work should address these limitations by implementing techniques to balance the dataset and enhance model robustness. Real-world testing on diverse datasets is essential to ensure generalizability and robustness. Additionally, applying advanced feature selection techniques could refine input features, potentially improving both accuracy and efficiency of the models.

**Data and Software availability:****Dataset link:**

**Pima Indians Diabetes Database:** <https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database>

**Collab notebook link:**

<https://colab.research.google.com/drive/1itjF49LPKoXz1WX4s74NkLxCb14Remhy>

**References:**

Suchi, T. A., Rabbi, M. A., & Layek, M. A. (2023, June 16). Effective Feature Selection and Soft Voting Classifier based Diabetes Detection Using Machine Learning Approaches. 2023 International Conference on Next-Generation Computing, IoT and Machine Learning (NCIM). <https://doi.org/10.1109/ncim59001.2023.10212616>

Tasin, I., Nabil, T. U., Islam, S., & Khan, R. (2022, December 14). Diabetes prediction using machine learning and explainable AI techniques. Healthcare Technology Letters, 10(1–2), 1–10. <https://doi.org/10.1049/htl2.12039>

Afsaneh, E., Sharifdini, A., Ghazzaghi, H. et al. Recent applications of machine learning and deep learning models in the prediction, diagnosis, and management of diabetes: a comprehensive review. Diabetol Metab Syndr 14, 196 (2022). <https://doi.org/10.1186/s13098-022-00969-9>