PREDICTING DIABETES USING MACHINE LEARNING

Presented by Group 8:

K.Shailaja Reddy

Mythri Rathod

M. Kushali

Thakur Rohan Singh

CONTENTS:

DATA DESCRIPTION

This project aims to predict diabetes using machine learning. The dataset contains patient health details such as pregnancies, glucose, blood pressure, insulin, BMI, age, and more. The target variable is Outcome, which shows whether a person has diabetes (1) or not (0). It helps in analyzing health factors that influence diabetes risk..

OBJECTIVE

The objective of this project is to predict diabetes in patients using machine learning. It analyzes health factors like glucose, BMI, age, and blood pressure to support early diagnosis and better healthcare planning.

THE PATH

The project follows a path of data collection, preprocessing, and exploratory analysis, followed by building machine learning models and evaluating their performance. Finally, the best model is selected for predicting diabetes

ATTRIBUTE INFORMATION

Attribute	Description	Data Type
Pregnancies	Number of times the patient has been pregnant	Integer
Glucose	Plasma glucose concentration (mg/dL)	Integer
BloodPressure	Diastolic blood pressure (mm Hg)	Integer
SkinThickness	Triceps skin fold thickness (mm)	Integer
Insulin	2-Hour serum insulin (mu U/ml)	Integer
BMI	Body Mass Index (weight in kg/(height in m)^2)	Float
DiabetesPedigreeFunction	Likelihood of diabetes based on family history	Float
Age	Age of the patient (in years)	Integer
Outcome	Class label (1 = Diabetic, 0 = Non-diabetic)	Integer

ABOUT THE DATA

The dataset consists of 8 feature columns and 1 target column.
“Outcome” is the target variable.

- Rows (Observations): 768
- Columns (Features): 9

VARIABLES

Continuous Variables
Pregnancies
Glucose
BloodPressure
SkinThickness
DiabetesPedigreeFunction
Insulin
Age
BMI
Outcome

DATA PREPROSSESSING

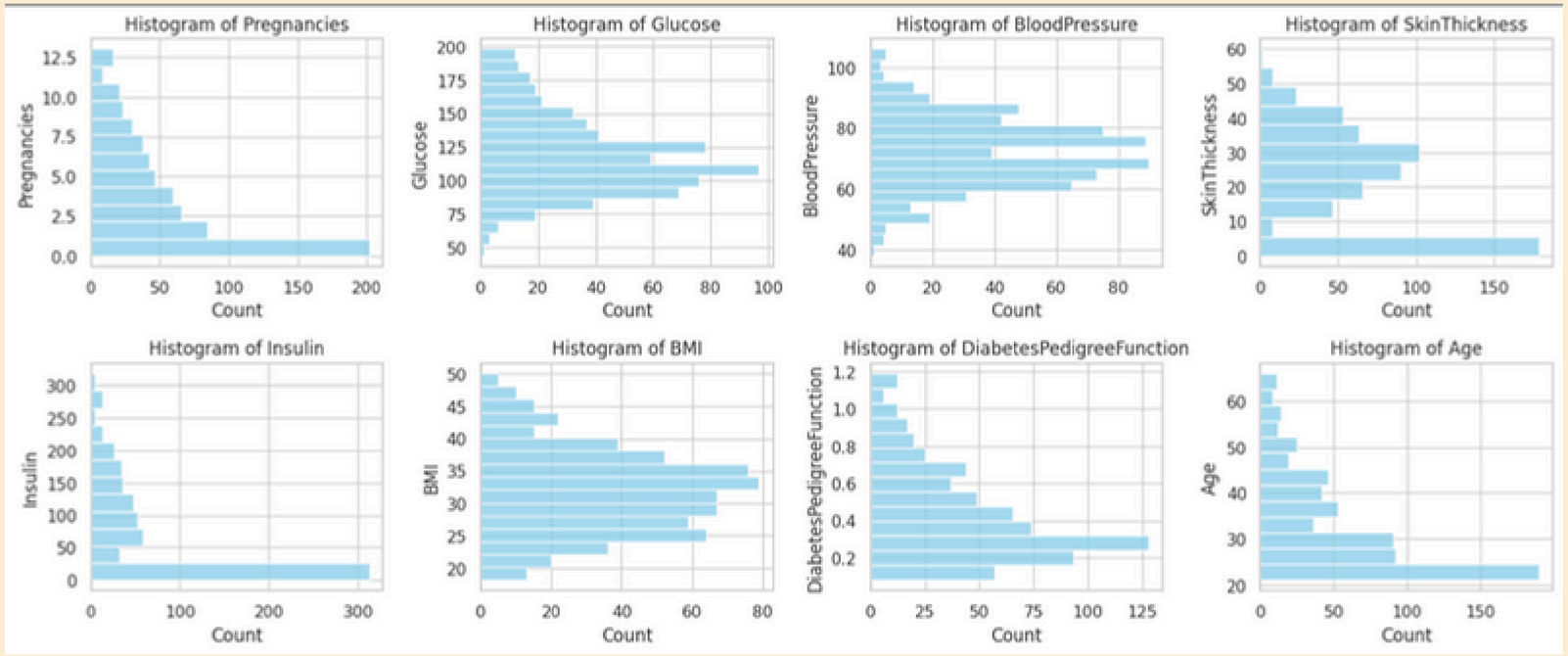
Pregnancies	0
Glucose	0
BloodPressure	0
SkinThickness	0
Insulin	0
BMI	0
DiabetesPedigreeFunction	0
Age	0
Outcome	0

dtype: int64

There are no missing values and no duplicate values in the given data

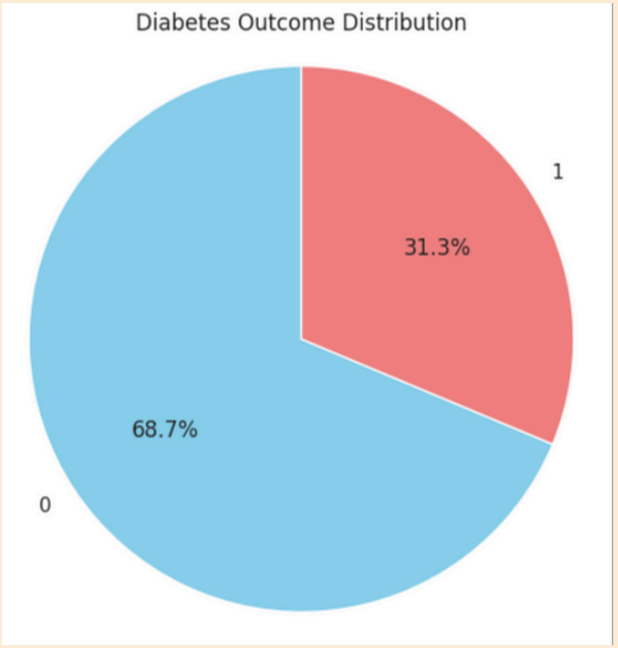
```
Number of duplicate rows:  0
Number of rows after removing duplicates:  768
```

EXPLORATORY DATA ANALYSIS(EDA)

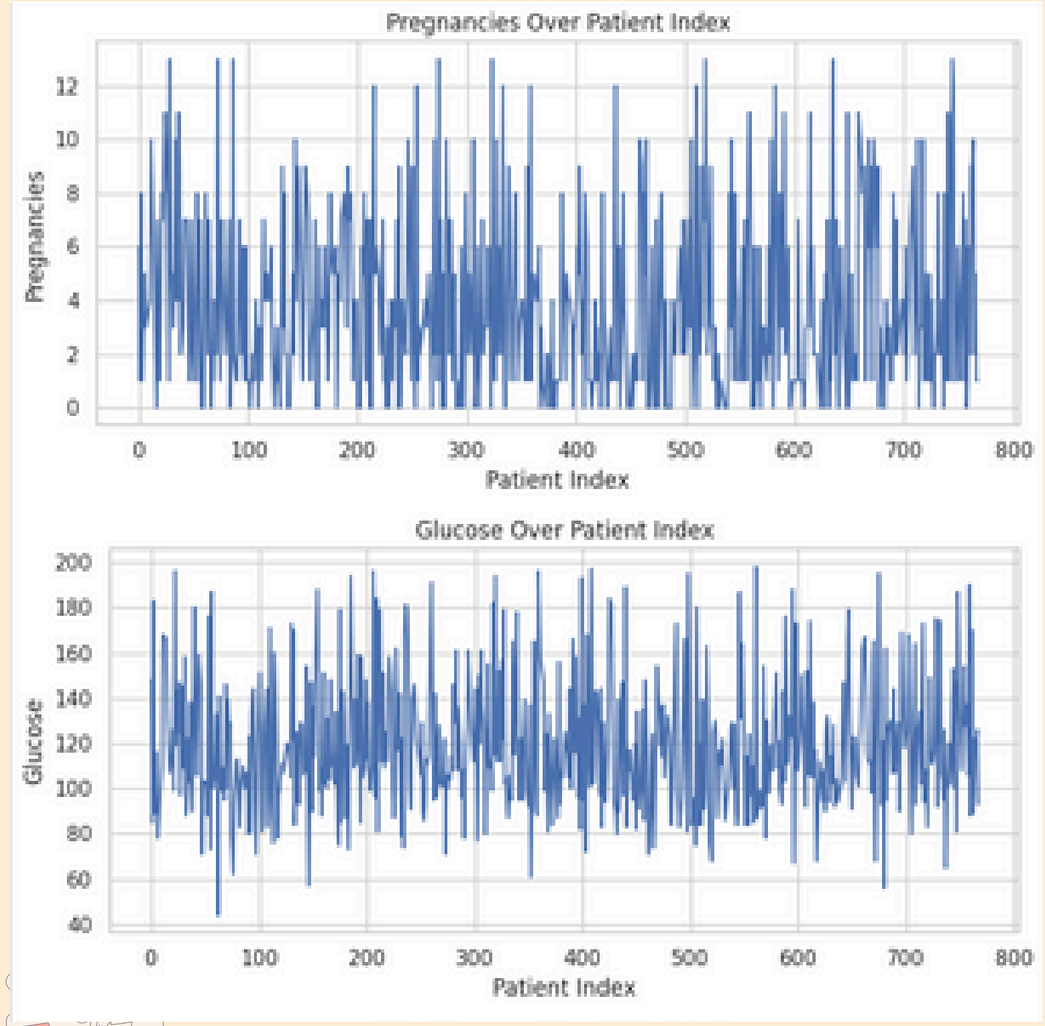


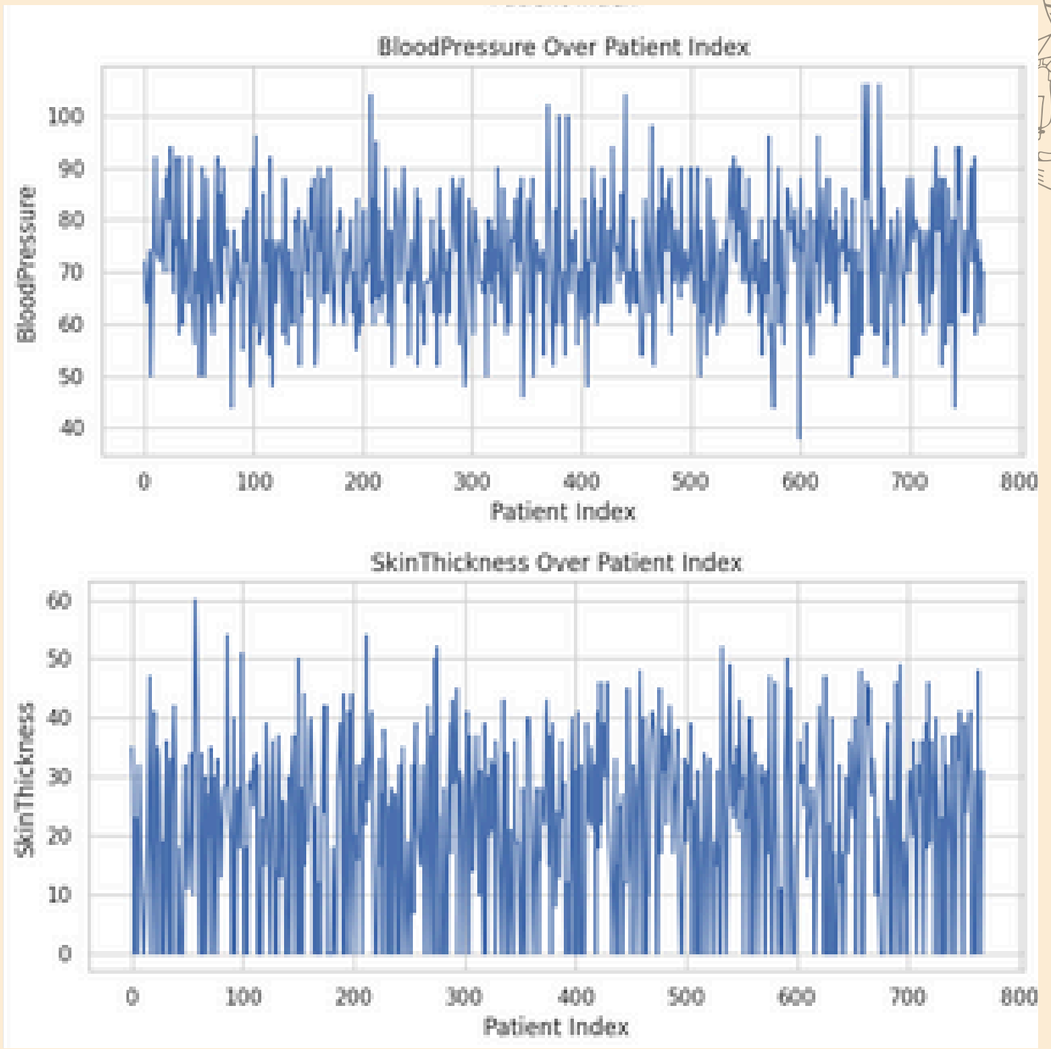
Histogram of all features variables

Pie chart of target variable



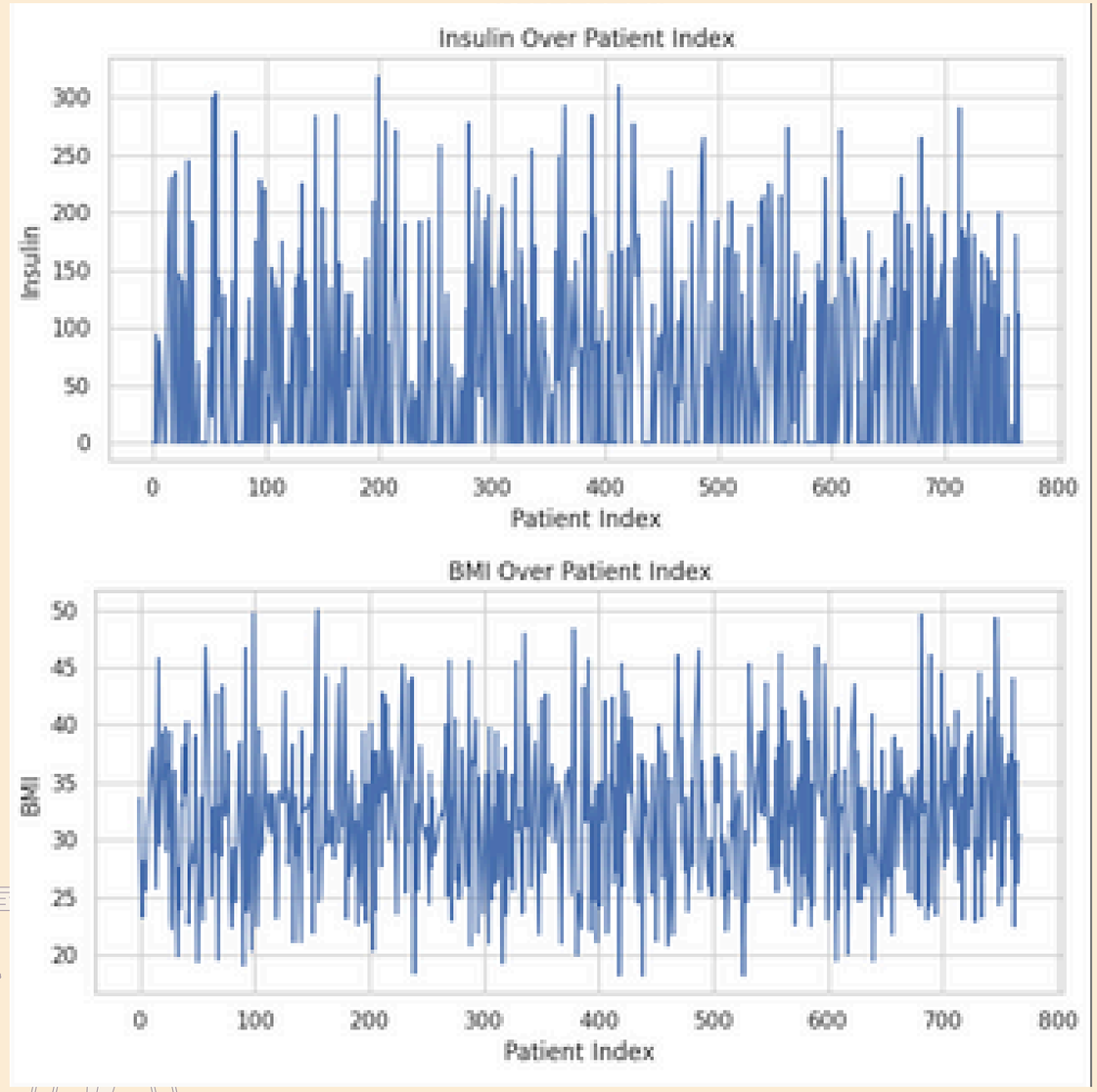
Line plot of pregnanices and Glucose

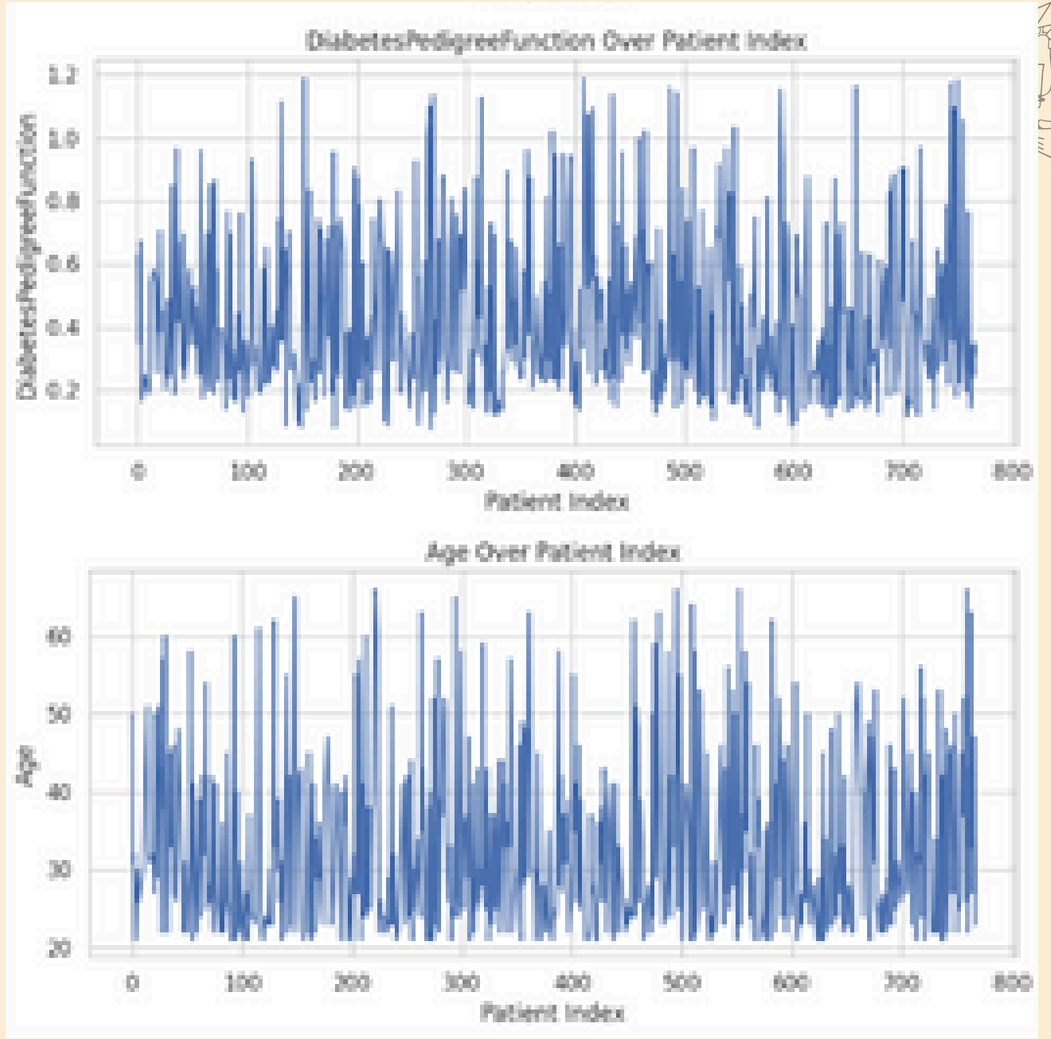




Line plot of Blood pressure and Skinthickness

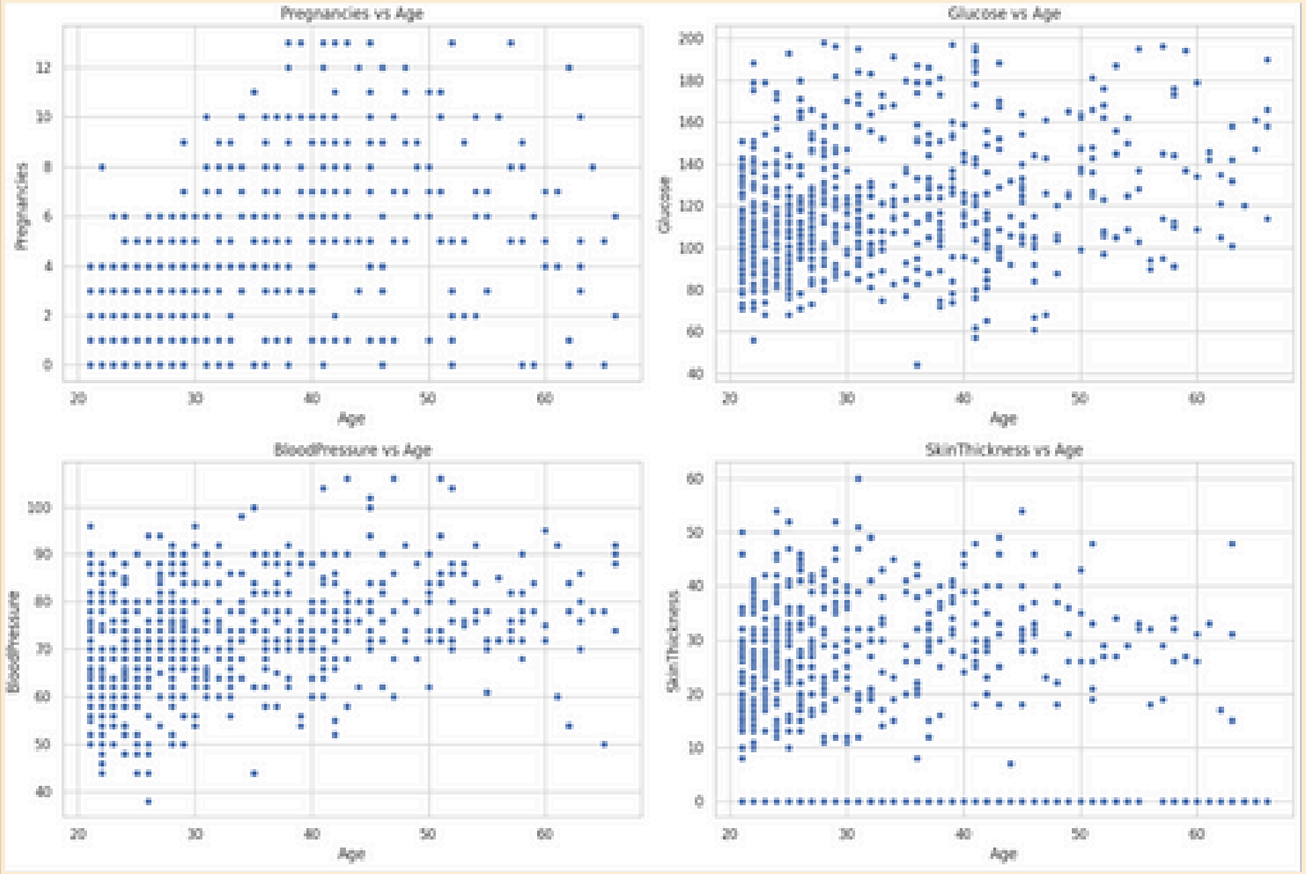
Line plot of Insulin and BMI

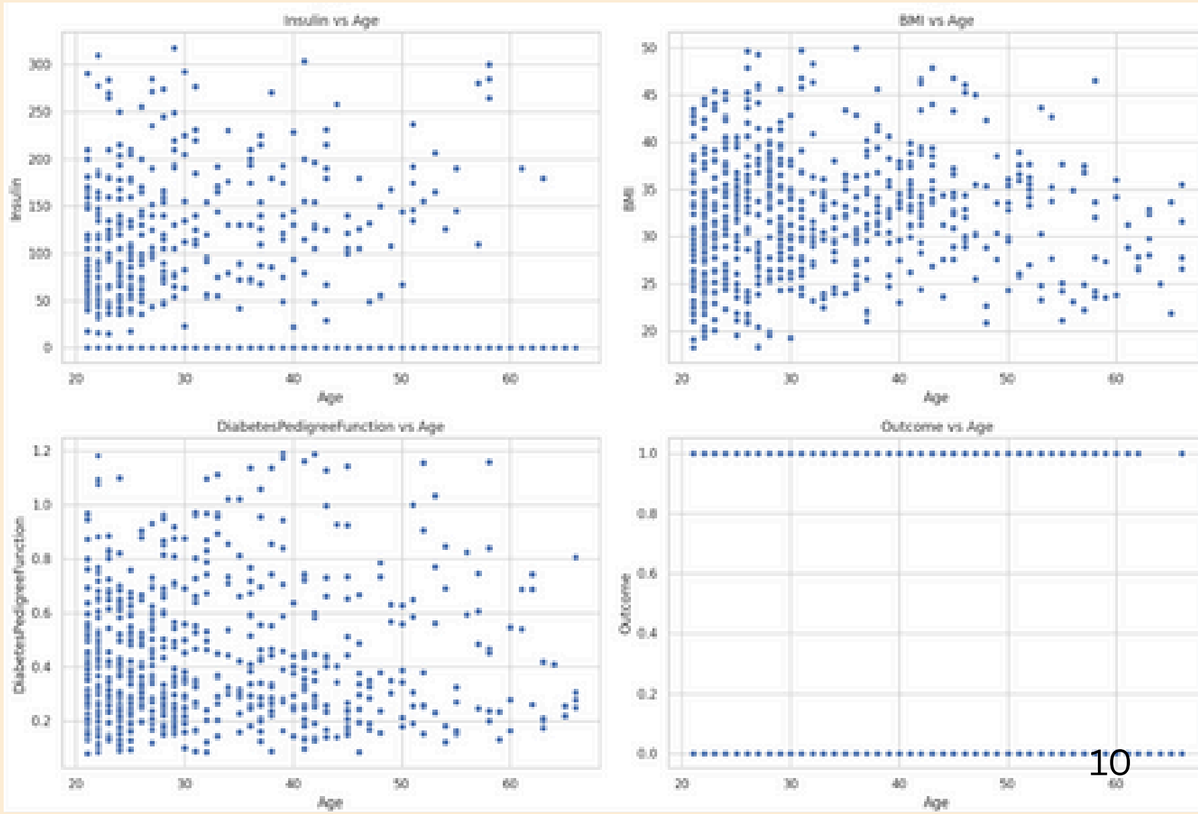




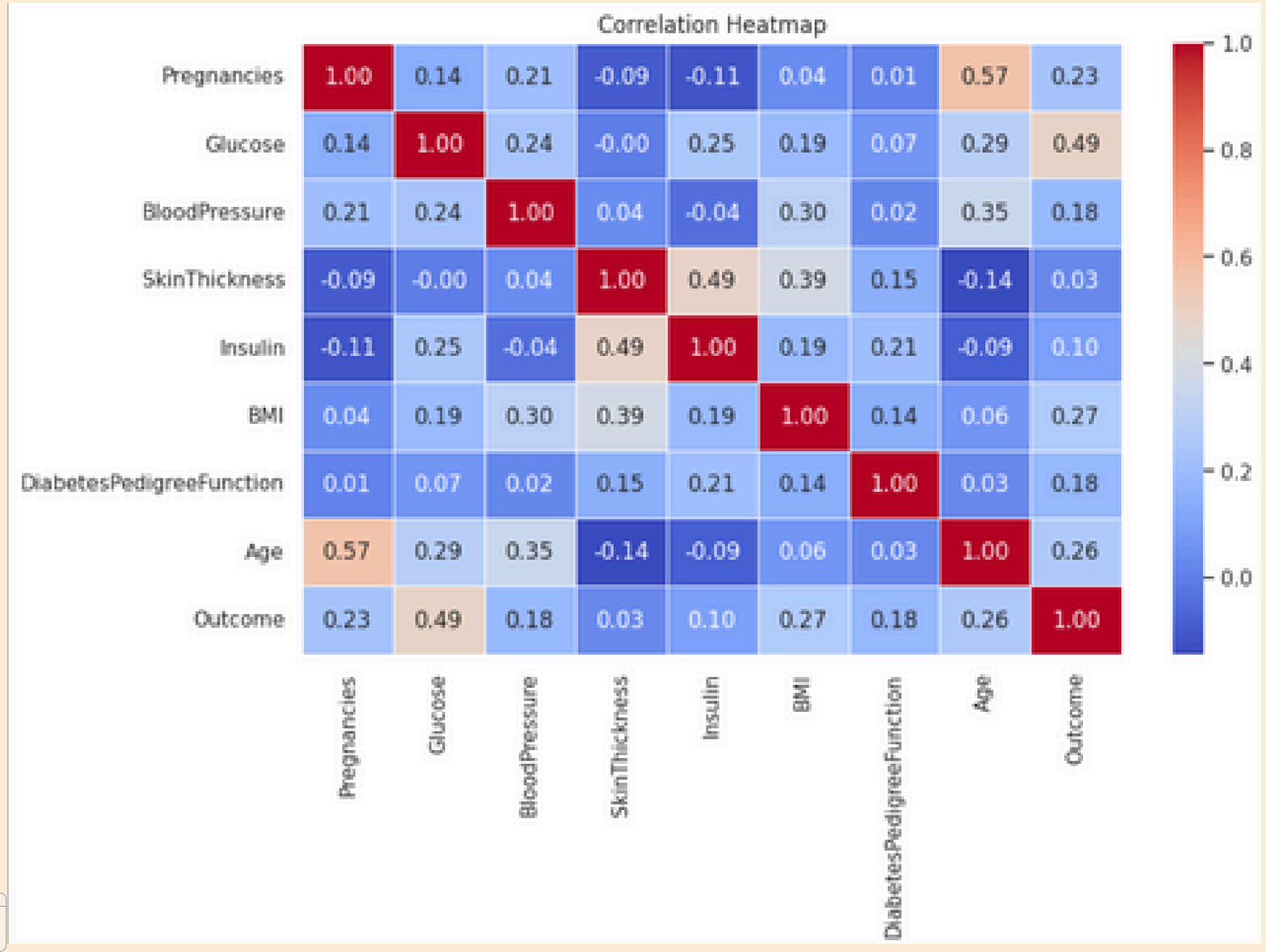
Line plot of DiabetesPedigree function and Age

SCATTER PLOT





HEAT MAP



"Glucose is the strongest predictor of diabetes, followed by BMI, Age, and Pregnancies."

VIF (Variance Inflation Factor)

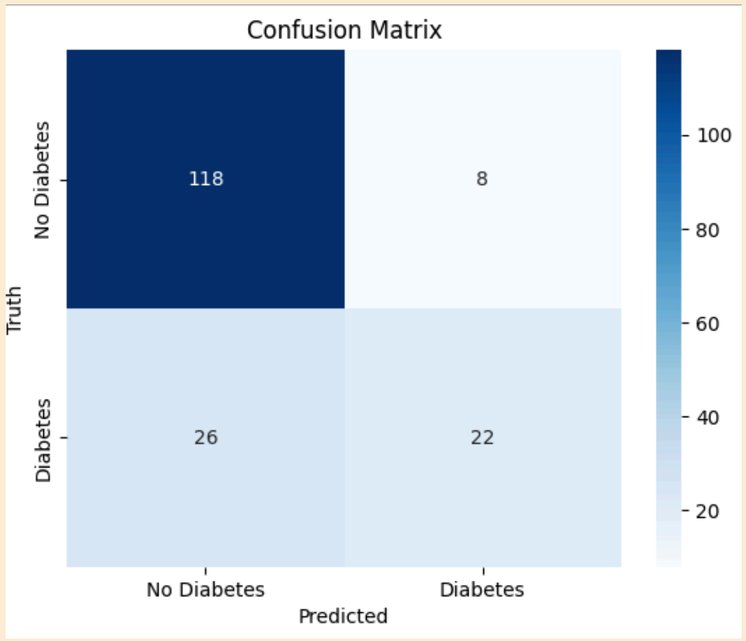
Measures multicollinearity between independent variables in classification; higher values indicate stronger correlation.

Feature	VIF
Pregnancies	3.484846
Glucose	20.947945
BloodPressure	37.356434
SkinThickness	4.430741
Insulin	2.554788
BMI	30.960665
DiabetesPedigreeFunction	4.100191
Age	16.588756

After VIF

Feature	VIF
Pregnancies	1.832635
SkinThickness	3.086644
Insulin	2.289419
DiabetesPedigreeFunction	2.810155

MODEL BUILDING



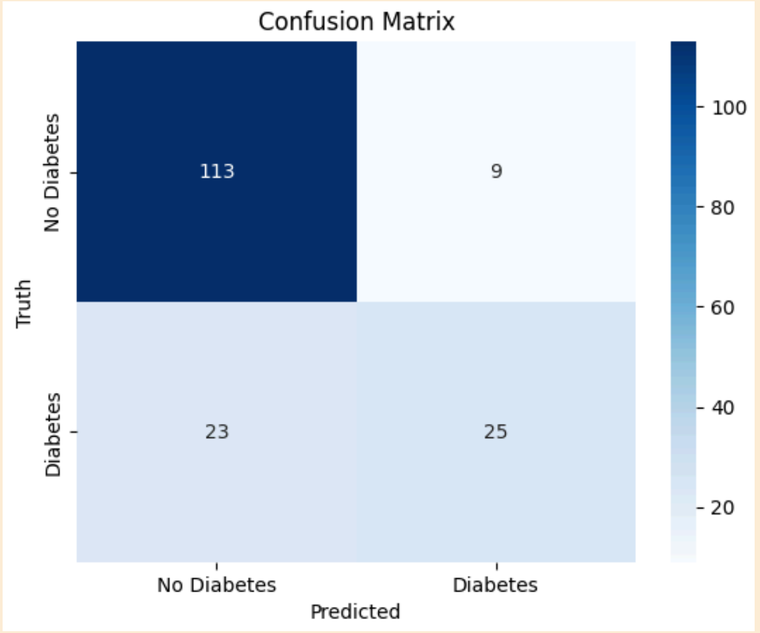
Logistic classification

Train – Test Ratio	Accuracy
80:20	0.78
75:25	0.74
70:30	0.74
65:35	0.75

A model used for predicting binary outcomes (yes/no, 0/1) based on input features. It predicts probabilities using the sigmoid function

SUPPORT VECTOR MACHINE

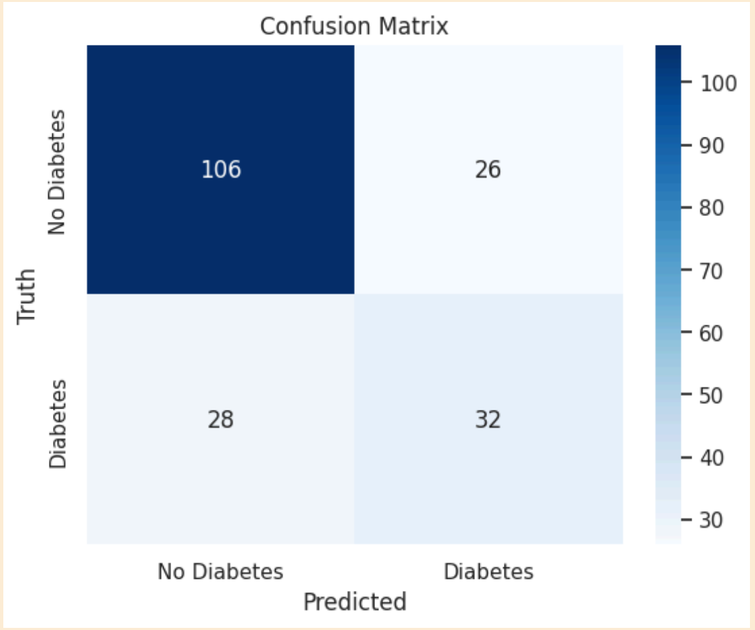
Train – Test Ratio	Accuracy
80:20	0.80
75:25	0.77
70:30	0.77
65:35	0.77



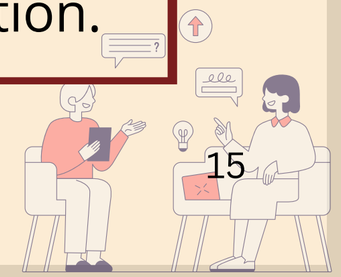
A supervised model that finds the best boundary (hyperplane) to separate classes.

NAIVE BAYES

Train – Test Ratio	Accuracy
80:20	0.78
75:25	0.75
70:30	0.74
65:35	0.75

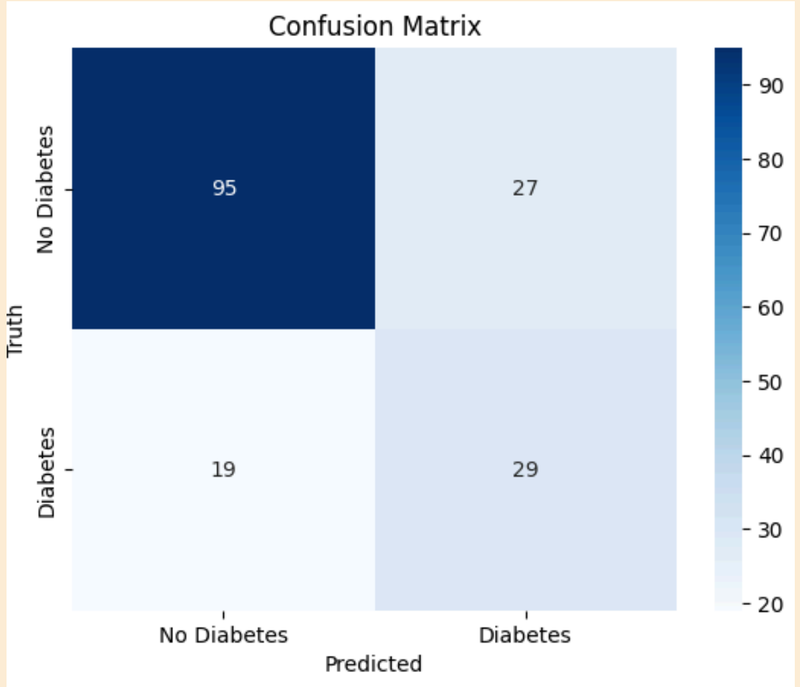


A simple probabilistic classifier based on Bayes' theorem. It assumes features are independent and works well for text classification like spam detection.



DECISION TREE

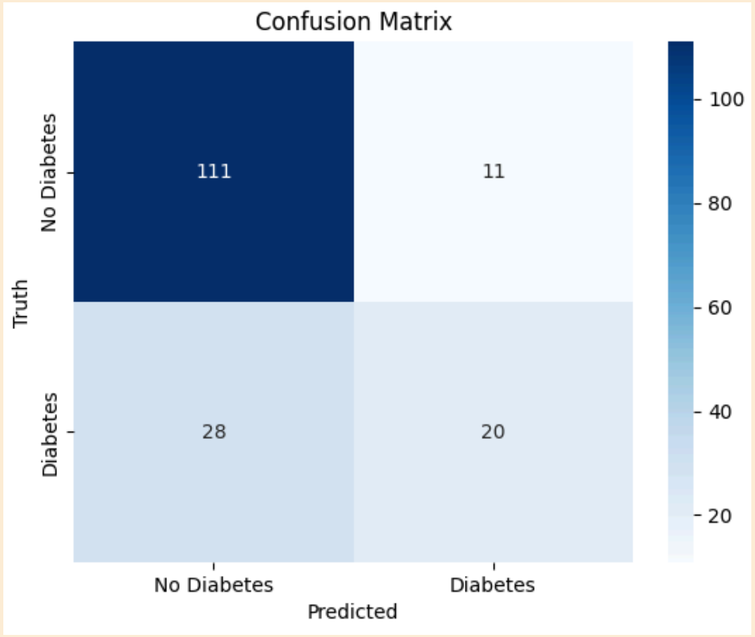
Train – Test Ratio	Accuracy
80:20	0.72
75:25	0.71
70:30	0.71
65:35	0.69



Splits data step by step based on conditions (if/else). Easy to understand and interpret visually. Can handle both numerical and categorical data. But prone to overfitting if not controlled (pruning helps).

RANDOM FOREST

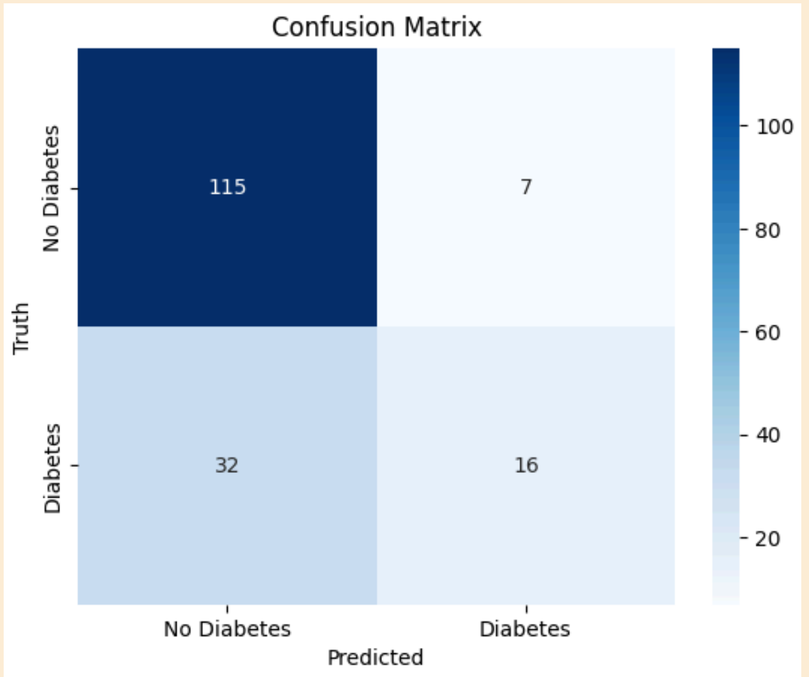
Train – Test Ratio	Accuracy
80:20	0.80
75:25	0.76
70:30	0.76
65:35	0.73



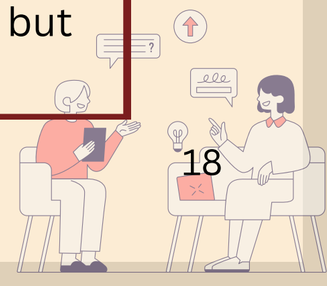
An ensemble of many decision trees combined together. Each tree is trained on random subsets of data. The final prediction is based on majority voting (classification) or averaging (regression).

K_NEAREST NEIGHBORS

Train – Test Ratio	Accuracy
80:20	0.78
75:25	0.71
70:30	0.71
65:35	0.73

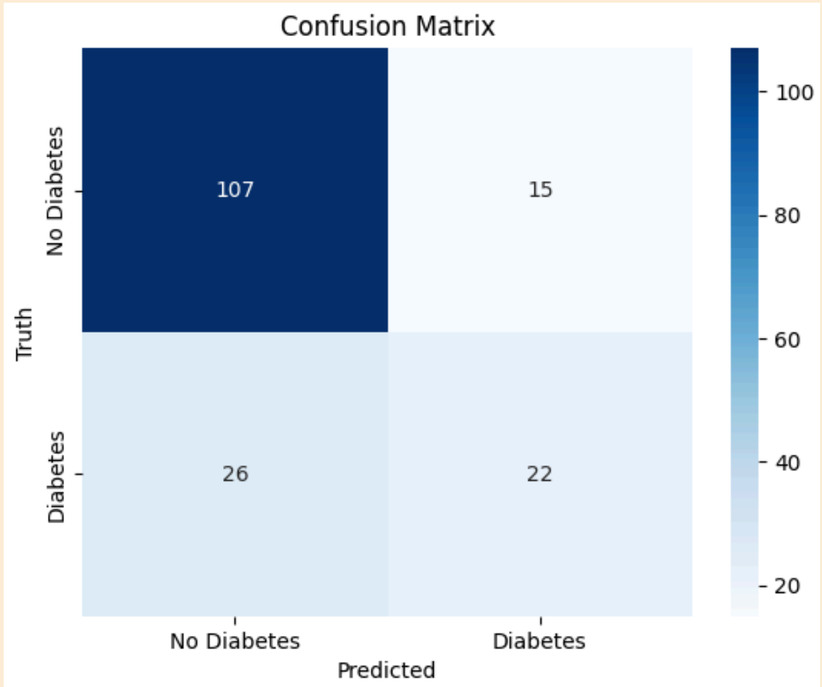


A simple algorithm that classifies based on the closest “k” data points. Distance metrics (like Euclidean) are used to find neighbors.
No training phase; prediction happens at runtime. Works well for small datasets but is slow for large ones.



BAGGING

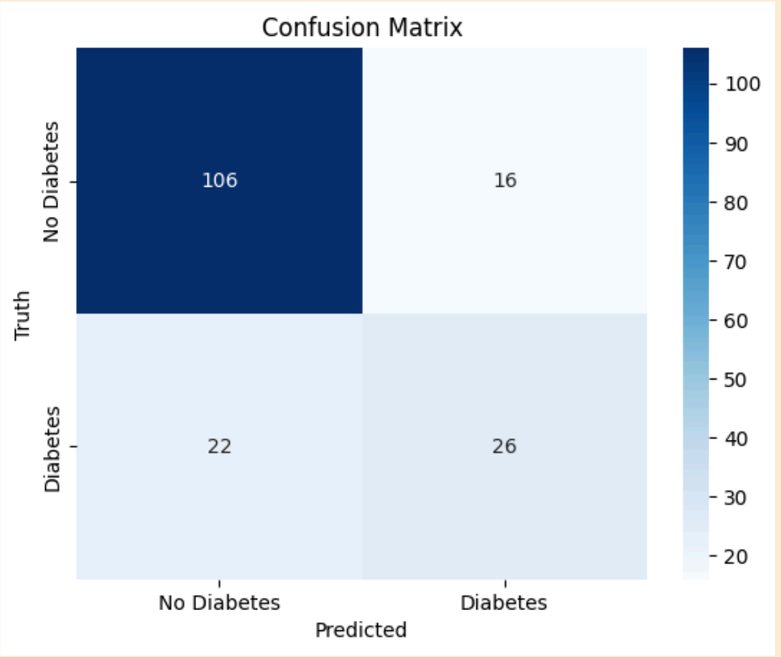
Train – Test Ratio	Accuracy
80:20	0.79
75:25	0.75
70:30	0.77
65:35	0.73



Uses random samples of data to train multiple models in parallel.
Predictions are combined by averaging (regression) or voting (classification).
Reduces variance and overfitting.

ADAPTIVE BOOSTING

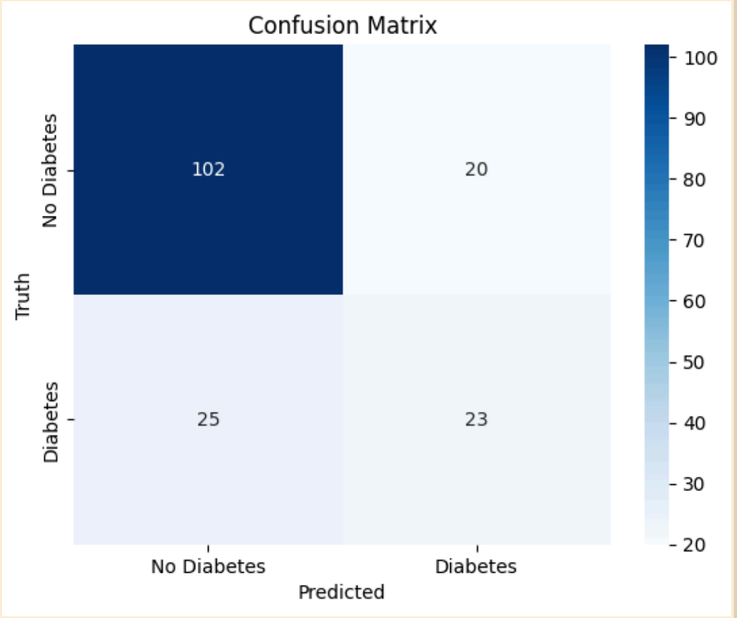
Train – Test Ratio	Accuracy
80:20	0.80
75:25	0.76
70:30	0.77
65:35	0.76



AdaBoost (Adaptive Boosting): It builds models sequentially, where each new model focuses more on the mistakes of the previous ones. By combining many weak learners, it creates a strong and accurate classifier.

GRADIENT BOOSTING

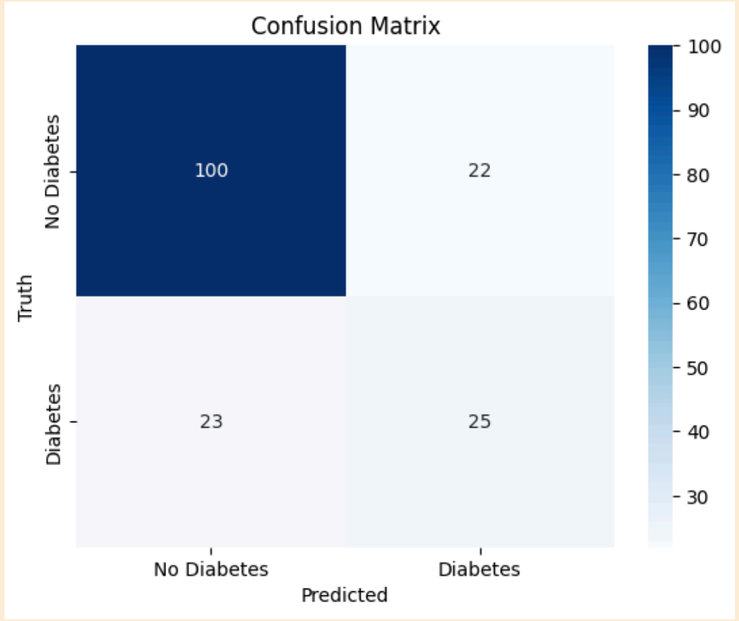
Train – Test Ratio	Accuracy
80:20	0.85
75:25	0.76
70:30	0.77
65:35	0.76



It builds models sequentially, where each new model corrects the errors of the previous one by minimizing a loss function. It combines weak learners (like small trees) into a strong model and often gives high accuracy.

EXTREME GRADIENT BOOSTING

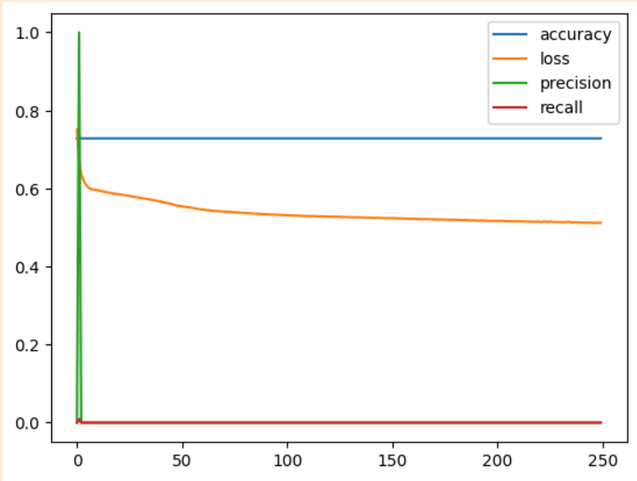
Train – Test Ratio	Accuracy
80:20	0.80
75:25	0.75
70:30	0.76
65:35	0.75



An advanced version of Gradient Boosting that is faster, more efficient, and handles large datasets well. It includes regularization to reduce overfitting and is widely used in ML competitions.

ANN

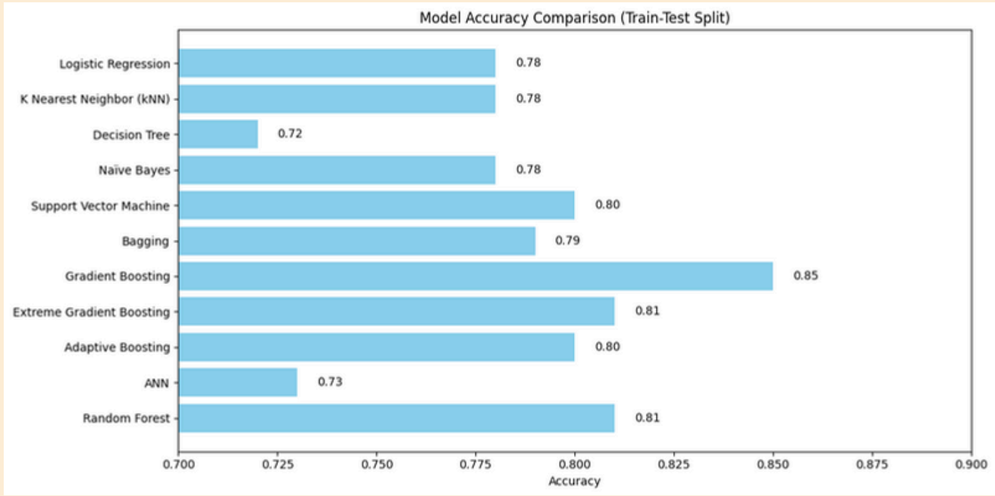
Train-Test	Architecture	Epochs	Accuracy
80:20	7-5-3-1	150	0.73
80:20	6-4-2-1	300	0.73
80:20	7-4-3-1	500	0.73
70:30	7-5-3-1	250	0.73
70:30	7-6-6-1	300	0.73
70:30	6-4-2-1	550	0.74
75:25	7-5-3-1	250	0.69
75:25	6-5-2-1	500	0.73
75:25	6-5-3-1	650	0.26
60:40	7-5-3-1	250	0.69
60:40	6-4-4-1	600	0.69
60:40	7-6-5-1	750	0.72



A brain-inspired model with interconnected layers of neurons that learn from data. It captures complex patterns and is the base for deep learning.

COMPARISON OF THE MODELS

Model	Train-Test	Accuracy
Logistic Regression	80-20	0.78
K Nearest Neighbor (kNN)	80:20	0.78
Decision Tree	80-20	0.72
Naïve Bayes	80-20	0.78
Support Vector Machine	80:20	0.8
Bagging	80:20	0.79
Gradient Boosting	80-20	0.85
Extreme Gradient Boosting	80-20	0.81
Adaptive Boosting	80:20	0.8
ANN	70-30	0.73
Random Forest	80-20	0.81



CONCLUSION :

After applying various ML algorithms to the dataset, the best accuracy is given by “Gradient Boosting“ algorithm for “80:20” train-test ratio with “Accuracy-0.85“ which is the highest among all the other algorithms.

INSIGHTS:

- Higher glucose levels strongly indicate diabetes.
- People with higher BMI are more likely to have diabetes.
- Age plays an important role – older individuals show higher chances of being diabetic.
- Pregnancies slightly increase the risk of diabetes in women.
- The Diabetes Pedigree Function shows family history as an important factor.

Thank You

COLAB LINK:

https://colab.research.google.com/drive/1heyC7jHxEPD5DT-d6Qs_Zur_CgsxTTVS?usp=sharing

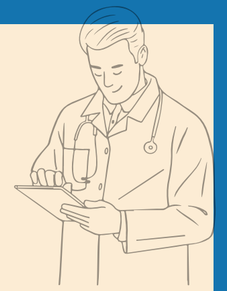
-BY GROUP 8

Mythri Rathod

K. Shailaja Reddy

M. Kushali

Thakur Rohan Singh



APPENDIX

Importing the libraries:

```
[ ]  import pandas as pd
     import numpy as np
     import matplotlib.pyplot as plt
     import seaborn as sns
     from sklearn.model_selection import train_test_split
     from sklearn.tree import DecisionTreeClassifier
     from sklearn.ensemble import RandomForestClassifier
     from sklearn.ensemble import BaggingClassifier
     from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
     from sklearn.ensemble import RandomForestClassifier
     from sklearn.ensemble import BaggingClassifier
     from sklearn.ensemble import GradientBoostingClassifier
     from sklearn.impute import SimpleImputer
     from sklearn.svm import SVC
     from sklearn.neighbors import KNeighborsClassifier
     from sklearn.linear_model import LogisticRegression
     from sklearn.model_selection import GridSearchCV, cross_val_score
     from xgboost import XGBClassifier
     from lightgbm import LGBMClassifier
     from sklearn.ensemble import AdaBoostClassifier
     from sklearn.ensemble import GradientBoostingClassifier
     from xgboost import XGBClassifier
     from sklearn.datasets import load_iris
     from sklearn.naive_bayes import GaussianNB
     from sklearn.metrics import accuracy_score
```

Training and Testing :

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, train_size=0.70, random_state = 42)
```

```
# fit a logistic regression model and store the class predictions
#70-30
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression(C=1e9)

logreg.fit(X_train, y_train)
predictions = logreg.predict(X_test)
print(predictions)
```

```
[1 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 1 0 1 0 1
 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 1
 0 0 1 0 1 1 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 1 0 0 1 0 0 0 0 1 0 0 0 0 0 0
 0 0 0 0 0 0 0 1 0 0 0 1 1 0 0 0 0 0 0 0 0 0]
```

Logistic Regression:

Accuracy

```
[ ]  from sklearn.metrics import accuracy_score
     accuracy_score(y_test,predictions)
```

```
0.7764705882352941
```

```
[ ]  from sklearn.model_selection import train_test_split
```

```
[ ]  from sklearn.linear_model import LogisticRegression
     model = LogisticRegression()
```

K Nearest Neighbour :

```
[ ] from sklearn.metrics import accuracy_score
    accuracy_score(y_test,y_pred)
```

```
0.7705882352941177
```

```
[ ] from sklearn.metrics import confusion_matrix
    cm = confusion_matrix(y_test,y_pred)
    cm
```

```
array([[115,   7],
       [ 32,  16]])
```

```
[ ] sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["No Diabetes", "Diabetes"], yticklabels=["No Diabetes", "Diabetes"
    plt.title('Confusion Matrix')
    plt.xlabel("Predicted")
    plt.ylabel("Truth")
    plt.show()
```

Decision Tree :

```
[ ]  from sklearn.metrics import accuracy_score
     accuracy_score(y_test,y_pred)
```

```
0.7294117647058823
```

```
[ ]  from sklearn.metrics import confusion_matrix
     cm = confusion_matrix(y_test,y_pred)
     cm
```

```
array([[95, 27],
       [19, 29]])
```

```
[ ]  sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["No Diabetes", "Diabetes"], yticklabels=["No Diabetes", "Di
     plt.title('Confusion Matrix')
     plt.xlabel("Predicted")
     plt.ylabel("Truth")
     plt.show()
```

Support Vector Machine:

```
[ ]  from sklearn.metrics import accuracy_score
     accuracy_score(y_test,y_pred)
```

```
0.8117647058823529
```

```
[ ]  from sklearn.metrics import confusion_matrix
     cm = confusion_matrix(y_test,y_pred)
     cm
```

```
array([[113,   9],
       [ 23,  25]])
```

```
[ ]  sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["No Diabetes", "Diabetes"], yticklabels=["No Diabetes", "Diabetes"])
     plt.title('Confusion Matrix')
     plt.xlabel("Predicted")
     plt.ylabel("Truth")
     plt.show()
```

RANDOM FOREST:

```
[ ]  from sklearn.metrics import accuracy_score
     accuracy_score(y_test,y_pred)
```

```
0.7705882352941177
```

```
[ ]  from sklearn.metrics import confusion_matrix
     cm = confusion_matrix(y_test,y_pred)
     cm
```

```
array([[111,  11],
       [ 28,  20]])
```

```
[ ]  sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["No Diabetes", "Diabetes"], yticklabels=["No Diabetes", "Dia
     plt.title('Confusion Matrix')
     plt.xlabel("Predicted")
     plt.ylabel("Truth")
     plt.show()
```

BAGGING:

```
[ ]  from sklearn.metrics import accuracy_score
     accuracy_score(y_test,y_pred)
```

```
0.7588235294117647
```

```
[ ]  from sklearn.metrics import confusion_matrix
     cm = confusion_matrix(y_test,y_pred)
     cm
```

```
array([[107,  15],
       [ 26,  22]])
```

```
[ ]  sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["No Diabetes", "Diabetes"], yticklabels=["No Diabetes", "Diabetes"])
     plt.title('Confusion Matrix')
     plt.xlabel("Predicted")
     plt.ylabel("Truth")
     plt.show()
```

ADAPTIVE BOOSTING:

```
[ ]  # Create Boosting Classifier object
     from sklearn.ensemble import RandomForestClassifier
     base_estimator = RandomForestClassifier(max_depth=12, random_state=42)
     br = AdaBoostClassifier(estimator=base_estimator, n_estimators=100, learning_rate=0.05)

     # Train Boosting Classifier
     br = br.fit(x_train,y_train)

     #Predict the response for test dataset
     y_pred = br.predict(x_test)
```

```
[ ]  accuracy_score(y_test,y_pred)
```

```
0.7529411764705882
```

GRADIENT BOOSTING:

```
[ ] from sklearn.metrics import accuracy_score
    accuracy_score(y_test,y_pred)
```

```
0.7352941176470589
```

```
[ ] from sklearn.metrics import confusion_matrix
    cm = confusion_matrix(y_test,y_pred)
    cm
```

```
array([[102,  20],
       [ 25,  23]])
```

```
[ ] sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["No Diabetes", "Diabetes"], yticklabels=["No Dia
    plt.title('Confusion Matrix')
    plt.xlabel("Predicted")
    plt.ylabel("Truth")
    plt.show()
```

EXTREME GRADIENT BOOSTING:

```
[ ]  from sklearn.metrics import accuracy_score
     accuracy_score(y_test,y_pred)
```

```
0.7352941176470589
```

```
[ ]  from sklearn.metrics import confusion_matrix
     cm = confusion_matrix(y_test,y_pred)
     cm
```

```
array([[100,  22],
       [ 23,  25]])
```

```
[ ]  sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["No Diabetes", "Diabetes"], yticklabels=["No Diabetes", "D
     plt.title('Confusion Matrix')
     plt.xlabel("Predicted")
     plt.ylabel("Truth")
     plt.show()
```

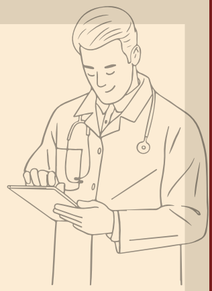
ANN:

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size=0.30, random_state=46)
```

```
#ReLU + Adam
tf.random.set_seed(42)

# STEP1: Creating the model

model1= tf.keras.Sequential([
                             tf.keras.layers.Dense(7, activation='relu'),
                             tf.keras.layers.Dense(5, activation='relu'),
                             tf.keras.layers.Dense(3, activation='relu'),
                             tf.keras.layers.Dense(1, activation='sigmoid')
])

# STEP2: Compiling the model

model1.compile(loss= tf.keras.losses.binary_crossentropy,
               optimizer= tf.keras.optimizers.Adam(learning_rate=0.001),
               metrics= [tf.keras.metrics.BinaryAccuracy(name='accuracy'),
                         tf.keras.metrics.Precision(name='precision'),
                         tf.keras.metrics.Recall(name='recall')
               ]
               )

# STEP1: Fit the model

history= model1.fit(x_train, y_train, epochs= 250)
```

```
[ ]  import pandas as pd

    # Evaluate all models
    results = []

    # Model 1
    res1 = model1.evaluate(x_test, y_test, verbose=0)
    results.append(["Model 1: RELU, relu, 24-12-5-1, 100 epochs", res1[0], res1[1], res1[2], res1[3]])

    # Model 2
    res2 = model2.evaluate(x_test, y_test, verbose=0)
    results.append(["Model 2: tanh, tanh, 10-8-7-1, 50 epochs", res2[0], res2[1], res2[2], res2[3]])

    # Model 3
    res3 = model3.evaluate(x_test, y_test, verbose=0)
    results.append(["Model 3: RMSprop, sigmoid, 22-15-8-1, 100 epochs", res3[0], res3[1], res3[2], res3[3]])

    # Model 4
    res4 = model4.evaluate(x_test, y_test, verbose=0)
    results.append(["Model 4: Adam (0.01 LR), relu, 22-15-8-1, 300 epochs", res4[0], res4[1], res4[2], res4[3]])

    # Create DataFrame
    df = pd.DataFrame(results, columns=["Model", "Loss", "Accuracy", "Precision", "Recall"])

    # Display table
    print(df)
```

ANN:



```
model1.evaluate(x_test, y_test)
```

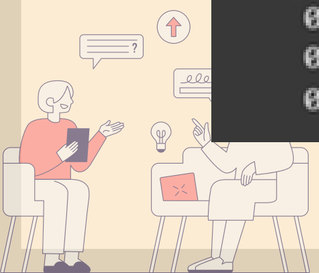
```
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.7300 - loss: 0.6144 - precision: 0.0000e+00 - recall: 0.0000e+00
[0.7006093859672546, 0.7176470756530762, 0.0, 0.0]
```

```
[ ] model2.evaluate(x_test, y_test)
```

```
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - accuracy: 0.7324 - loss: 0.5816 - precision: 0.0000e+00 - recall: 0.0000e+00
[0.5837113857269287, 0.730088472366333, 0.0, 0.0]
```

```
[ ] model3.evaluate(x_test, y_test)
```

```
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7419 - loss: 0.5258 - precision: 0.5241 - recall: 0.3010
[0.513127326965332,
 0.7433628439903259,
 0.5405405163764954,
 0.32786884903907776]
```



```
============================
Train-Test Split: 65-35
============================
SVC Accuracy: 0.7723
RandomForest Accuracy: 0.7366
Bagging Accuracy: 0.7366
KNN Accuracy: 0.7366
DecisionTree Accuracy: 0.6920
ada Accuracy: 0.7455
gb Accuracy: 0.7634
xgb Accuracy: 0.7545
naive Accuracy: 0.7500
LogisticRegression Accuracy: 0.7545

============================
Train-Test Split: 75-25
============================
SVC Accuracy: 0.7750
RandomForest Accuracy: 0.7688
Bagging Accuracy: 0.7500
KNN Accuracy: 0.7188
DecisionTree Accuracy: 0.7188
ada Accuracy: 0.7625
gb Accuracy: 0.7625
xgb Accuracy: 0.7562
naive Accuracy: 0.7500
LogisticRegression Accuracy: 0.7438

============================
Train-Test Split: 70-30
============================
SVC Accuracy: 0.7760
RandomForest Accuracy: 0.7656
Bagging Accuracy: 0.7708
KNN Accuracy: 0.7135
DecisionTree Accuracy: 0.7188
ada Accuracy: 0.7708
gb Accuracy: 0.7760
xgb Accuracy: 0.7604
naive Accuracy: 0.7448
LogisticRegression Accuracy: 0.7448
```