

Table of Contents:

1. **ABSTRATCT**
2. **INTRODUCTION**
3. **EXISTING SYSTEM**
4. **PROPOSED SYSTEMS**
5. **ARCHITECTURE TECHNIQUES**
6. **IMPLEMENTATION METHODOLOGY**
7. **RAW DATA**
8. **VISUALISATION**
9. **CODE**
10. **EXECUTION**
11. **OUTPUT**
12. **SPYDER**
13. **CONCLUSION**
14. **REFERENCES**

Diabetes Prediction Using Machine Learning

Abstract:

Diabetes mellitus is a prevalent chronic disease that affects millions of people worldwide, leading to significant healthcare challenges. Early detection and timely intervention are crucial for managing diabetes effectively and reducing its associated complications. This study explores the application of machine learning techniques to predict the risk of diabetes onset based on a comprehensive dataset of patient attributes, including demographic information, medical history, and lifestyle factors.

INTRODUCTION

Diabetes is a chronic medical condition characterized by elevated levels of glucose (sugar) in the blood. It occurs when the body has difficulty regulating blood sugar due to problems with insulin, a hormone produced by the pancreas that helps regulate glucose levels. There are two primary types of diabetes: Type 1 diabetes and Type 2 diabetes, as well as a less common form known as gestational diabetes that occurs during pregnancy.

1. Type 1 Diabetes:

- **Autoimmune Disorder:** Type 1 diabetes is an autoimmune disorder in which the immune system mistakenly attacks and destroys the insulin-producing beta cells in the pancreas.
- **Onset:** It typically develops during childhood or adolescence but can occur at any age.
- **Insulin Dependency:** People with Type 1 diabetes are entirely dependent on insulin injections or an insulin pump to manage their blood sugar levels.
- **Causes:** The exact cause is unknown, but genetic and environmental factors are believed to play a role.

2. Type 2 Diabetes:

- **Insulin Resistance:** Type 2 diabetes is characterized by insulin resistance, where the body's cells do not respond effectively to insulin, and the pancreas may not produce enough insulin.
- **Onset:** It often develops in adulthood, although it is increasingly diagnosed in children and adolescents due to rising obesity rates.
- **Risk Factors:** Obesity, poor diet, physical inactivity, genetics, and family history are major risk factors.
- **Management:** Initially managed with lifestyle changes (diet and exercise), and some individuals may require oral medications or insulin injections if blood sugar control cannot be achieved through lifestyle modifications.

3. Gestational Diabetes:

- **During Pregnancy:** Gestational diabetes occurs during pregnancy when hormonal changes affect insulin sensitivity. It usually resolves after childbirth.
- **Monitoring:** Pregnant women are screened for gestational diabetes, and if diagnosed, it is managed through diet, exercise, and sometimes insulin.

Common symptoms of diabetes include excessive thirst, frequent urination, unexplained weight loss, fatigue, and blurred vision. If left untreated or poorly managed, diabetes can lead to serious complications, including heart disease, kidney disease, nerve damage (neuropathy), vision problems (diabetic retinopathy), and slow wound healing.

Effective diabetes management involves maintaining blood sugar levels within a target range through a combination of medication, lifestyle changes (such as a balanced diet and regular physical activity), and regular monitoring of blood glucose levels. Early diagnosis, education, and ongoing medical care are essential to prevent or manage diabetes-related complications.

It's crucial for individuals at risk for diabetes or those with a family history to be aware of the condition's signs and seek medical advice for early intervention and management. Diabetes is a chronic condition that requires lifelong attention, but with proper care, many people with diabetes can lead healthy and fulfilling lives.

EXISTING SYSTEMS:

As of my last knowledge update in September 2021, there were several existing systems and platforms for diabetes prediction and management. These systems often use a combination of data analytics, machine learning, and patient data to provide predictions and support for individuals at risk of diabetes or those already diagnosed with diabetes. Here are some examples:

1. EHR (Electronic Health Records) Systems
2. Mobile Apps and Wearable Devices.
3. Continuous Glucose Monitoring (CGM) Systems
4. Telemedicine Platforms.
5. Research Studies and Clinical Trials.
6. Population Health Management Systems.
7. Artificial Intelligence (AI) in Healthcare.

PROPOSED SYSTEMS:

Proposing a system for diabetes prediction involves considering various factors, including data sources, predictive models, and user interfaces. Below is a conceptual outline of a proposed system for diabetes prediction:

1. Data Collection and Integration.
2. Data Preprocessing.
3. Predictive Models.
4. Model Training and Evaluation.
5. Risk Assessment.
6. User Interface.
7. Alerts and Notifications.
8. Integration with Healthcare Systems.
9. Continuous Improvement.
10. Research and Validation.
11. Accessibility and Education.

This proposed diabetes prediction system aims to leverage data-driven insights to assess diabetes risk accurately, empower individuals to take proactive measures for prevention, and support healthcare providers in delivering personalized care. It should be developed in collaboration with healthcare professionals and researchers to ensure its effectiveness and compliance with healthcare standards.

ARCHITECTURE TECHNIQUES:

The architecture and techniques used in diabetes prediction systems involve a combination of data processing, machine learning, and software design. Below, I outline key architectural techniques commonly employed in diabetes prediction systems:

1. Data Integration and Preprocessing.

- Data Sources Integration.
- Data Cleaning and Transformation.

2. Feature Engineering.

- Feature Selection.
- Feature Creation.

3. Machine Learning Models.

- Model Selection.
- Ensemble Techniques.

4. Model Training and Evaluation.

- Cross-Validation.
- Hyperparameter Tuning.

5. Real-Time Data Processing.

- Streaming Data.
- Data Pipelines

6. Scalability and Deployment.

- Cloud-Based Architecture.
- Containerization.

7. Security and Privacy.

- Data Encryption.
- Access Control.

8. User Interface and Communication.

- User-Friendly Interfaces.
- Notifications.

9. Continuous Monitoring and Maintenance.

- Monitoring.
- Regular Updates.

10. Compliance with Regulations.

Successful architecture and techniques in diabetes prediction systems require a multidisciplinary approach, involving data scientists, machine learning engineers, healthcare professionals, and software developers. Collaboration and ongoing evaluation are crucial to building accurate, secure, and user-friendly systems for diabetes risk assessment and management.

IMPLEMENTATION METHODOLOGY:

Data Collection:

This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. The objective of the dataset is to diagnostically predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset. 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.

RAW DATA:

Data Set Information:

- **Pregnancies**
Number of times pregnant
- **Glucose**
Plasma glucose concentration a 2 hours in an oral glucose tolerance test
- **Blood Pressure**
Diastolic blood pressure (mm Hg)
- **Skin Thickness**
Triceps skin fold thickness (mm)
- **Insulin**
2-Hour serum insulin (mu U/ml)

Pre-processing:

The collected data may contain missing values that may lead to inconsistency. To gain better results data need to be pre-processed and so it'll better the effectiveness of the algorithm. We should remove the outliers and we need to convert the variables. In order to flooring these issues we use chart function.

Train model on training data set:

Now we should train the model on the training dataset and make sooth sayings for the test dataset. We can divide our train dataset into two tract train and testimony. We can train the model on this training part and using that make sooth sayings for the testimony part. In this way, we can validate our sooth sayings as we've the true sooth sayings for the testimony part (which we don't have for the test dataset)

Correlating attributes:

Grounded on the correlation among attributes it was observed more likely to pay back their loans. The attributes that are individual and significant can include Property area, education, loan measure, and originally credit History, which is since by insight it's considered as important. The correlation among attributes can be associated using plot and boxplot in Python platform

Decision Tree:

Decision tree is a type of supervised education algorithm (having a pre- defined target variable) that is generally used in category problems. In this approach, we disassociate the population or sample into two or added homogeneous sets (or sub-populations) based on the most significant splitter/ differentiator in input variables.

Decision trees use multiple algorithms to decide to disunite a bump into two or added sub- knots. The creation of sub- knots increases the unsophistication of attendant sub- knots. In other words, we can say that chasteness of the bump increases with respect to the target variable.

Predicting the outcomes:

Using decision tree algorithm, the outcomes of all applicant can be stored in any file. Algorithm:

- Import all the required python modules
- Import the database for both TESTING and TRAINING.
- Check any NULLVALUES are exists
- If NULLVALUES exists, fill the table with corresponding coding
- Exploratory Data Analysis for all ATTRIBUTES from the table
- Plot all graphs using MATPLOTLIB module
- Build the DECISIONTREE MODEL for the coding
- Send that output to CSV FILE

Data Processing:

After the data has been acquired, the data preparation stage begins, when raw data is cleaned up and structured in preparation for the next stage of data processing. The pre-processing stage involves Data discretization, data transformation, data cleaning, data integration. After the dataset is transformed into a clean dataset, the dataset is divided into training and testing sets to evaluate. When data is collected and transformed into usable information, it is called data processing. Data processing is usually done by a data scientist or a team of data scientists, and it is critical that it is done correctly so that the end product, or data output, is not harmed. Data processing takes raw data and puts it into a more legible format (graphs, papers, etc.) so that computers can comprehend it and personnel throughout an organization can use it.

Determine the training and testing data:

Typically, Here the system separates a dataset into a training set and testing set, most of the data use for training, and a smaller portion of data is use for testing. after a system has been processed by using the training set, it makes the prediction against the test set.

Data cleaning and processing: In Data cleaning the system detect and correct corrupt or inaccurate records from database and refers to identifying incomplete, incorrect, inaccurate or irrelevant parts of the data and then replacing, modifying or detecting the dirty or coarse data. In Data processing the system convert data from a given form to a much more usable and desired form i.e., make it more meaningful and informative.

Feature Extraction:

Feature extraction refers to the process of transforming raw data into numerical features that can be processed while preserving the information in the original data set. It yields better results than applying machine learning directly to the raw data. When you need to reduce the number of resources needed for processing without losing significant or relevant information, feature extraction comes in handy. Feature extraction can also reduce the amount of redundant data for a given analysis. Furthermore, the reduction of data and the machine's

efforts in constructing variable combinations (features) speed up the learning and generalization stages of the machine learning process. Image Processing is one of the practical examples of feature extraction where the Algorithms are used to detect features such as shaped, edges, or motion in a digital image or video.

Logistic Regression:

Logistic regression is supervised learning classification algorithm (try to method connections and conditions between the target prediction output and input attributes) such that we are able to anticipate the yield values for new information based on those connections which it learned from the previous information sets.

Training Neural Network:

Backpropagation is the most common training algorithm for neural networks. It makes gradient descent feasible for multi-layer neural networks.

steps for training a neural network:

- Data Set.
- Neural Networks.
- Training Strategy.
- Model Selection.
- Testing Analysis.
- Model Development.

Output Generation:

The output/interpretation stage is where non-data scientists can finally use the data. It is translated, readable, and frequently takes the shape of graphs, videos, photos, plain text, and other formats. Members of the firm or institution can now use the data for their own data analytics initiatives by self-serving it.

Visualization:

Data visualization uses algorithms to generate visuals from data so that humans may better comprehend and respond to it. The goal of artificial intelligence research is to create algorithms that can "understand" and respond to data as well as a human — if not better. Data storage is the ultimate stage of data processing. After all of the data has been analyzed, it is saved for future reference. While some information will be useful right away, most of it will be useful later. Furthermore, complying with data protection legislation necessitates correctly maintained data. When data is correctly saved, individuals of the organization may access it quickly and readily when they need it. A rolling analysis of a time series model is frequently used to determine the model's long-term stability. When using a statistical model to analyze financial time series data, one crucial assumption is that the model's parameters remain constant across time.

Machine Learning:

Machine learning (ML) is a field of inquiry devoted to understanding and building methods that 'learn', that is, methods that leverage data to improve performance on some set of tasks. It is seen as a part of artificial intelligence. Machine learning algorithms build a model based on sample data, known as training data, in order to make predictions or decisions without being explicitly programmed to do so. Machine learning algorithms are used in a wide variety of applications, such as in medicine, email filtering, speech recognition and computer vision, where it is difficult or unfeasible to develop conventional algorithms to perform the needed tasks. A subset of machine learning is closely related to computational statistics, which focuses on making predictions using computers, but not all machine learning is statistical learning. The study of mathematical optimization delivers methods, theory and application domains to the field of machine learning. Data mining is a related field of study, focusing on exploratory data analysis through unsupervised

learning. Some implementations of machine learning use data and neural networks in a way that mimics the working of a biological learning. In its application across business problems, machine learning is also referred to as predictive learning.

CODE:

Essential

```
import numpy as np
```

```
import pandas as pd
```

Visualization

```
import seaborn as sns
```

```
import matplotlib.pyplot as plt
```

```
import matplotlib.style as style
```

```
import plotly.express as ex
```

```
import plotly.express as px
```

```
import plotly.graph_objects as go
```

```
import missingno as msno
```

Machine learning and statistical

```
from sklearn.preprocessing import StandardScaler
```

```
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
```

```
from sklearn.svm import SVC
```

```
from sklearn.neighbors import KNeighborsClassifier
```

```
from sklearn.neural_network import MLPClassifier
```

```
from sklearn.naive_bayes import GaussianNB
```

```
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

```
from xgboost import XGBClassifier
```

```
from imblearn.over_sampling import SMOTE
```

```
from sklearn.model_selection import train_test_split, cross_val_score, KFold
```

Ignore useless warnings

```
import warnings
```

```
warnings.filterwarnings('ignore')
```

```
df = pd.read_csv("D:\\Diabetes\\diabetes.csv") # read data frame.
```

```
df.shape
```

```
df.head(10)
```

```
df.info()
```

```
df.describe().T
```

#Correlation

```
corr = df.corr()
```

```
plt.figure() #plot the heatmap for the correlation
```

```
sns.heatmap(corr,fmt=".5f", linewidth=.5, cmap="coolwarm")
```

```
plt.show() #the more darker color the more stronger correlation.
```

#Data Preprocessing

Check for duplicates across all columns

```
duplicated = df.duplicated().sum()
```

Print the number of duplicated instances

```
if duplicated == 0:
```

```
    print("Number of duplicated instances:", duplicated)
```

Print the duplicated instances

```
else:
```

```
    print("Number of duplicated instances:", duplicated)
```

```
    print(df[duplicated])
```

print the percentage of missing values for instances.

```
total = df.isnull().sum().sort_values(ascending=False)[df.isnull().sum().sort_values(ascending=False) != 0]
```

```
percent = ((df.isnull().sum() / df.isnull().count()).sort_values(ascending=False)[df.isnull().sum() / df.isnull().count().sort_values(ascending=False) != 0]) * 100
```

```
missing = pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])
```

```
missing['Percent'] = missing['Percent'].apply(lambda x: "%.2f%%" % x)
```

```

print(missing)

# null count analysis

p=msno.bar(df)

#Plot relationship between variables

# Count the occurrences of each outcome (0: No
diabetes, 1: Diabetes)

outcome_counts = df['Outcome'].value_counts()

# Create subplots

fig, axes = plt.subplots(nrows=3, ncols=3,
figsize=(15, 10))

# Flatten the axes array for easier iteration

axes = axes.flatten()

# Features to plot (excluding 'Outcome' which is
the target variable)

features = df.columns[:-1]

# Plot histograms for each feature

for i, feature in enumerate(features):

    ax = axes[i]

    ax.hist(df[df['Outcome'] == 0][feature],
alpha=0.5, label='No Diabetes', color='blue',
bins=20)

    ax.hist(df[df['Outcome'] == 1][feature],
alpha=0.5, label='Diabetes', color='red', bins=20)

    ax.set_title(feature)

    ax.legend()

# Adjust layout and display

plt.tight_layout()

plt.show()

# Create a bar plot

plt.figure(figsize=(6, 4))

plt.bar(outcome_counts.index,
outcome_counts.values, tick_label=['No Diabetes',
'Diabetes'])

plt.xlabel('Outcome')

plt.ylabel('Count')

plt.title('Distribution of Outcome (Diabetes vs. No
Diabetes)')

# Show the plot

plt.show()

# Create subplots

fig, axes = plt.subplots(nrows=3, ncols=3,
figsize=(15, 10))

# Flatten the axes array for easier iteration

axes = axes.flatten()

# Features to plot (excluding 'Outcome' which is
the target variable)

features = df.columns[:-1]

# Plot histograms for each feature

for i, feature in enumerate(features):

    ax = axes[i]

    ax.hist(df[df['Outcome'] == 0][feature],
alpha=0.5, label='No Diabetes', color='blue',
bins=20)

    ax.hist(df[df['Outcome'] == 1][feature],
alpha=0.5, label='Diabetes', color='red', bins=20)

    ax.set_title(feature)

    ax.legend()

# Adjust layout and display

plt.tight_layout()

plt.show()

sns.pairplot(df, hue='Outcome', diag_kind='kde')

plt.show()

# Create a 3D scatter plot

fig = plt.figure(figsize=(10, 8))

ax = fig.add_subplot(111, projection='3d')

# Define colors for diabetes status (0 for No
Diabetes, 1 for Diabetes)

colors = {0: 'blue', 1: 'red'}

# Scatter plot with Age on the x-axis, BMI on the
y-axis, and Glucose on the z-axis

for outcome, color in colors.items():

    subset = df[df['Outcome'] == outcome]

```

```

    ax.scatter(subset['Age'], subset['BMI'],
subset['Glucose'], c=color, label=f'Diabetes
{outcome}')
# Set labels for each axis
ax.set_xlabel('Age')
ax.set_ylabel('BMI')
ax.set_zlabel('Glucose')
# Add a legend
ax.legend()
# Show the 3D scatter plot
plt.show()
# Create subplots with two violin plots
fig, axes = plt.subplots(1, 2, figsize=(14, 6))
# Violin Plot 1: Age by Outcome (Diabetes vs. No
Diabetes)
sns.violinplot(x='Outcome', y='Age', data=df,
palette='Set1', ax=axes[0])
axes[0].set_title('Violin Plot: Age by Outcome')
axes[0].set_xlabel('Outcome (0 = No Diabetes, 1 =
Diabetes)')
axes[0].set_ylabel('Age')
# Violin Plot 2: Diabetes Pedigree Function by
Outcome (Diabetes vs. No Diabetes)
sns.violinplot(x='Outcome',
y='DiabetesPedigreeFunction', data=df,
palette='Set1', ax=axes[1])
axes[1].set_title('Violin Plot: Diabetes Pedigree
Function by Outcome')
axes[1].set_xlabel('Outcome (0 = No Diabetes, 1 =
Diabetes)')
axes[1].set_ylabel('Diabetes Pedigree Function')
# Adjust layout
plt.tight_layout()
# Show the subplots
plt.show()
# Create subplots with two box plots side by side
fig, axes = plt.subplots(1, 2, figsize=(14, 6))

```

```

# Box Plot 1: Blood Pressure by Outcome
(Diabetes vs. No Diabetes)
sns.boxplot(x='Outcome', y='BloodPressure',
data=df, palette='Set1', ax=axes[0])
axes[0].set_title('Box Plot: Blood Pressure by
Outcome')
axes[0].set_xlabel('Outcome (0 = No Diabetes, 1 =
Diabetes)')
axes[0].set_ylabel('Blood Pressure')
# Box Plot 2: Insulin by Outcome (Diabetes vs. No
Diabetes)
sns.boxplot(x='Outcome', y='Insulin', data=df,
palette='Set1', ax=axes[1])
axes[1].set_title('Box Plot: Insulin by Outcome')
axes[1].set_xlabel('Outcome (0 = No Diabetes, 1 =
Diabetes)')
axes[1].set_ylabel('Insulin')
# Adjust layout
plt.tight_layout()
# Show the subplots
plt.show()
#Machine Learning
# Extract the features (X) and target variable (y)
X = df.drop('Outcome', axis=1)
y = df['Outcome']
# Split the data into training and testing sets
X_train, X_test, y_train, y_test =
train_test_split(X, y, test_size=0.2,
random_state=42)
# Standardize the features
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
fig = px.pie(y_train, names='Outcome')
fig.update_layout(title='<b>Outcome Proportion
before SMOTE Upsampling</b>')
fig.show()

```

```

fig = px.pie(y_train, names='Outcome')
fig.update_layout(title='<b>Outcome Proportion
before SMOTE Upsampling</b>')
fig.show()

# Define a list of classifiers to evaluate
classifiers = [
    ('Logistic Regression',
     LogisticRegression(max_iter=1000)),
    ('Random Forest',
     RandomForestClassifier(random_state=42)),
    ('XGBoost', XGBClassifier(random_state=42)),
    ('SVM', SVC(kernel='linear', C=1)),
    ('K-Nearest Neighbors',
     KNeighborsClassifier(n_neighbors=5)),
    ('Naive Bayes', GaussianNB())
]

# Print evaluation results with improved formatting
for name, classifier in classifiers:

    # Fit the model on the training data
    classifier.fit(X_train, y_train)

    # Make predictions on the test data
    y_pred = classifier.predict(X_test)

    print(f'{name}')

    accuracy = accuracy_score(y_test, y_pred)
    print(f'\nAccuracy: {accuracy:.4f}')

    cm = confusion_matrix(y_test, y_pred)
    print("\nConfusion Matrix:")

    print(cm)

    report = classification_report(y_test, y_pred)
    print("\nClassification Report:")

    print(report)

    print("-" * 50)

    # Perform 10-fold cross-validation for each
    model
    for name, classifier in classifiers:

```

```

        # Create a KFold cross-validator
        kf = KFold(n_splits=10, shuffle=True,
                    random_state=42)

        # Perform cross-validation
        scores = cross_val_score(classifier, X_train,
                                   y_train, cv=kf)

        # Print model name and cross-validation scores
        print(name)

        # Calculate and print the mean and standard
        deviation of the cross-validation scores

        mean_score = np.mean(scores)
        std_score = np.std(scores)

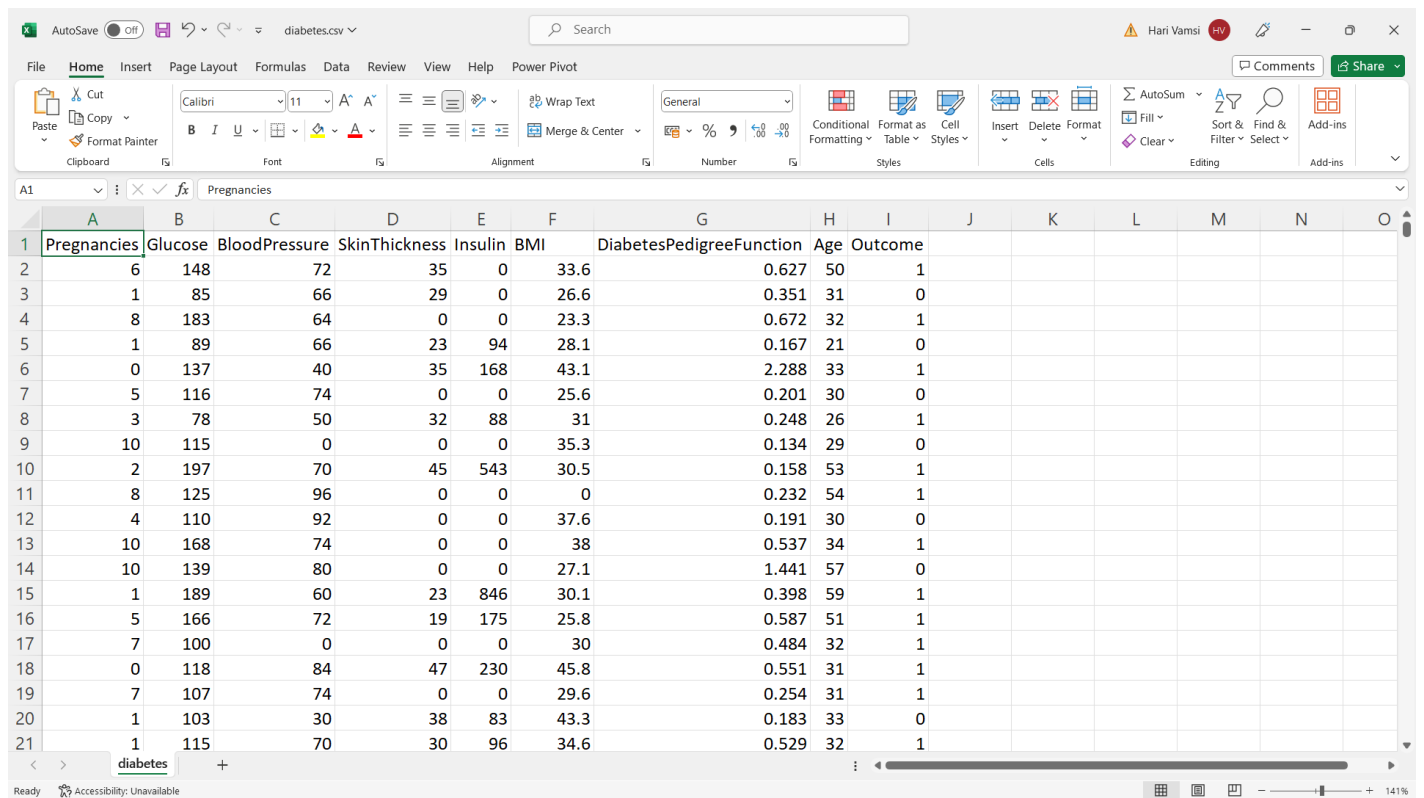
        print(f'Mean Accuracy: {mean_score:.4f}')
        print(f'Standard Deviation: {std_score:.4f}')

        # Print a separator for better readability
        print("-" * 50)

```

EXECUTION:

Data:



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome						
2	6	148	72	35	0	33.6	0.627	50	1						
3	1	85	66	29	0	26.6	0.351	31	0						
4	8	183	64	0	0	23.3	0.672	32	1						
5	1	89	66	23	94	28.1	0.167	21	0						
6	0	137	40	35	168	43.1	2.288	33	1						
7	5	116	74	0	0	25.6	0.201	30	0						
8	3	78	50	32	88	31	0.248	26	1						
9	10	115	0	0	0	35.3	0.134	29	0						
10	2	197	70	45	543	30.5	0.158	53	1						
11	8	125	96	0	0	0	0.232	54	1						
12	4	110	92	0	0	37.6	0.191	30	0						
13	10	168	74	0	0	38	0.537	34	1						
14	10	139	80	0	0	27.1	1.441	57	0						
15	1	189	60	23	846	30.1	0.398	59	1						
16	5	166	72	19	175	25.8	0.587	51	1						
17	7	100	0	0	0	30	0.484	32	1						
18	0	118	84	47	230	45.8	0.551	31	1						
19	7	107	74	0	0	29.6	0.254	31	1						
20	1	103	30	38	83	43.3	0.183	33	0						
21	1	115	70	30	96	34.6	0.529	32	1						

Code:

```
# Essential
import numpy as np
import pandas as pd

# Visualization
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.style as style
import plotly.express as ex
import plotly.express as px
import plotly.graph_objects as go
import missingno as msno

# Machine learning and statistical
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from xgboost import XGBClassifier
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split, cross_val_score, KFold

# Ignore useless warnings
import warnings
warnings.filterwarnings('ignore')

df = pd.read_csv("D:\\Diabetes\\diabetes.csv") # read data frame.
df.shape
df.head(10)
df.info()
df.describe().T

|

#Correlation
corr = df.corr()
plt.figure() #plot the heatmap for the correlation
sns.heatmap(corr,fmt=".5f", linewidth=.5, cmap="coolwarm")
plt.show() #the more darker color the more stronger correlation.
```

```

#Data Preprocessing
# Check for duplicates across all columns
duplicated = df.duplicated().sum()

# Print the number of duplicated instances
if duplicated == 0:
    print("Number of duplicated instances:", duplicated)
# Print the duplicated instances
else:
    print("Number of duplicated instances:", duplicated)
    print(df[duplicated])

# print the percentage of missing values for instances.
total = df.isnull().sum().sort_values(ascending=False)[df.isnull().sum().sort_values(ascending=False) != 0]
percent = ((df.isnull().sum() / df.isnull().count()).sort_values(ascending=False)[df.isnull().sum() / df.isnull().count().sort_values(ascending=False) != 0]) * 100
missing = pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])
missing['Percent'] = missing['Percent'].apply(lambda x: "%.2f%%" % x)
print(missing)
# null count analysis
p=msno.bar(df)

#Plot relationship between variables
# Count the occurrences of each outcome (0: No diabetes, 1: Diabetes)
outcome_counts = df['Outcome'].value_counts()
# Create subplots
fig, axes = plt.subplots(nrows=3, ncols=3, figsize=(15, 10))

# Flatten the axes array for easier iteration
axes = axes.flatten()

# Features to plot (excluding 'Outcome' which is the target variable)
features = df.columns[:-1]

# Plot histograms for each feature
for i, feature in enumerate(features):
    ax = axes[i]
    ax.hist(df[df['Outcome'] == 0][feature], alpha=0.5, label='No Diabetes', color='blue', bins=20)
    ax.hist(df[df['Outcome'] == 1][feature], alpha=0.5, label='Diabetes', color='red', bins=20)
    ax.set_title(feature)
    ax.legend()

# Adjust layout and display
plt.tight_layout()

```

```

plt.tight_layout()
plt.show()

# Create a bar plot
plt.figure(figsize=(6, 4))
plt.bar(outcome_counts.index, outcome_counts.values, tick_label=['No Diabetes', 'Diabetes'])
plt.xlabel('Outcome')
plt.ylabel('Count')
plt.title('Distribution of Outcome (Diabetes vs. No Diabetes)')

# Show the plot
plt.show()
# Create subplots
fig, axes = plt.subplots(nrows=3, ncols=3, figsize=(15, 10))

# Flatten the axes array for easier iteration
axes = axes.flatten()

# Features to plot (excluding 'Outcome' which is the target variable)
features = df.columns[:-1]

# Plot histograms for each feature
for i, feature in enumerate(features):
    ax = axes[i]
    ax.hist(df[df['Outcome'] == 0][feature], alpha=0.5, label='No Diabetes', color='blue', bins=20)
    ax.hist(df[df['Outcome'] == 1][feature], alpha=0.5, label='Diabetes', color='red', bins=20)
    ax.set_title(feature)
    ax.legend()

# Adjust layout and display
plt.tight_layout()
plt.show()
sns.pairplot(df, hue='Outcome', diag_kind='kde')
plt.show()

# Create a 3D scatter plot
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')

# Define colors for diabetes status (0 for No Diabetes, 1 for Diabetes)
colors = {0: 'blue', 1: 'red'}

```

```

axes[1].set_xlabel('Outcome (0 = No Diabetes, 1 = Diabetes)')
axes[1].set_ylabel('Insulin')

# Adjust layout
plt.tight_layout()

# Show the subplots
plt.show()

#Machine Learning
# Extract the features (X) and target variable (y)
X = df.drop('Outcome', axis=1)
y = df['Outcome']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Standardize the features
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
fig = px.pie(y_train, names='Outcome')
fig.update_layout(title='<b>Outcome Proportion before SMOTE Upsampling</b>')
fig.show()
fig = px.pie(y_train, names='Outcome')
fig.update_layout(title='<b>Outcome Proportion before SMOTE Upsampling</b>')
fig.show()
# Define a list of classifiers to evaluate
classifiers = [
    ('Logistic Regression', LogisticRegression(max_iter=1000)),
    ('Random Forest', RandomForestClassifier(random_state=42)),
    ('XGBoost', XGBClassifier(random_state=42)),
    ('SVM', SVC(kernel='linear', C=1)),
    ('K-Nearest Neighbors', KNeighborsClassifier(n_neighbors=5)),
    ('Naive Bayes', GaussianNB())
]

# Print evaluation results with improved formatting
for name, classifier in classifiers:
    # Fit the model on the training data
    classifier.fit(X_train, y_train)

# Scatter plot with Age on the x-axis, BMI on the y-axis, and Glucose on the z-axis
for outcome, color in colors.items():
    subset = df[df['Outcome'] == outcome]
    ax.scatter(subset['Age'], subset['BMI'], subset['Glucose'], c=color, label=f'Diabetes {outcome}')

# Set labels for each axis
ax.set_xlabel('Age')
ax.set_ylabel('BMI')
ax.set_zlabel('Glucose')

# Add a legend
ax.legend()

# Show the 3D scatter plot
plt.show()
# Create subplots with two violin plots
fig, axes = plt.subplots(1, 2, figsize=(14, 6))

# Violin Plot 1: Age by Outcome (Diabetes vs. No Diabetes)
sns.violinplot(x='Outcome', y='Age', data=df, palette='Set1', ax=axes[0])
axes[0].set_title('Violin Plot: Age by Outcome')
axes[0].set_xlabel('Outcome (0 = No Diabetes, 1 = Diabetes)')
axes[0].set_ylabel('Age')

# Violin Plot 2: Diabetes Pedigree Function by Outcome (Diabetes vs. No Diabetes)
sns.violinplot(x='Outcome', y='DiabetesPedigreeFunction', data=df, palette='Set1', ax=axes[1])
axes[1].set_title('Violin Plot: Diabetes Pedigree Function by Outcome')
axes[1].set_xlabel('Outcome (0 = No Diabetes, 1 = Diabetes)')
axes[1].set_ylabel('Diabetes Pedigree Function')

# Adjust layout
plt.tight_layout()

# Show the subplots
plt.show()

# Create subplots with two box plots side by side
fig, axes = plt.subplots(1, 2, figsize=(14, 6))

# Box Plot 1: Blood Pressure by Outcome (Diabetes vs. No Diabetes)
sns.boxplot(x='Outcome', y='BloodPressure', data=df, palette='Set1', ax=axes[0])
axes[0].set_title('Box Plot: Blood Pressure by Outcome')
axes[0].set_xlabel('Outcome (0 = No Diabetes, 1 = Diabetes)')

```

```

# Make predictions on the test data
y_pred = classifier.predict(X_test)
print(f"{name}")
accuracy = accuracy_score(y_test, y_pred)
print(f"\nAccuracy: {accuracy:.4f}")

cm = confusion_matrix(y_test, y_pred)
print("\nConfusion Matrix:")
print(cm)

report = classification_report(y_test, y_pred)
print("\nClassification Report:")
print(report)
print("-" * 50)

# Perform 10-fold cross-validation for each model
for name, classifier in classifiers:
    # Create a KFold cross-validator
    kf = KFold(n_splits=10, shuffle=True, random_state=42)

    # Perform cross-validation
    scores = cross_val_score(classifier, X_train, y_train, cv=kf)

    # Print model name and cross-validation scores
    print(name)

    # Calculate and print the mean and standard deviation of the cross-validation scores
    mean_score = np.mean(scores)
    std_score = np.std(scores)
    print(f"Mean Accuracy: {mean_score:.4f}")
    print(f"Standard Deviation: {std_score:.4f}")

    # Print a separator for better readability
    print("-" * 50)

```


Output:

```
In [2]: runfile('D:/Diabetes/Code.py', wdir='D:/Diabetes')
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
#   Column                Non-Null Count  Dtype
---  -
0   Pregnancies            768 non-null    int64
1   Glucose                768 non-null    int64
2   BloodPressure          768 non-null    int64
3   SkinThickness          768 non-null    int64
4   Insulin                768 non-null    int64
5   BMI                    768 non-null    float64
6   DiabetesPedigreeFunction 768 non-null    float64
7   Age                    768 non-null    int64
8   Outcome                768 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

Warning

Figures now render in the Plots pane by default. To make them also appear inline in the Console, uncheck "Mute Inline Plotting" under the Plots pane options menu.

```
Number of duplicated instances: 0
Empty DataFrame
Columns: [Total, Percent]
Index: []

Logistic Regression

Accuracy: 0.7532
```

Confusion Matrix:

```
[[79 20]
 [18 37]]
```

Classification Report:

	precision	recall	f1-score	support
0	0.81	0.80	0.81	99
1	0.65	0.67	0.66	55
accuracy			0.75	154
macro avg	0.73	0.74	0.73	154
weighted avg	0.76	0.75	0.75	154

Random Forest

Accuracy: 0.7208

Confusion Matrix:

```
[[77 22]
 [21 34]]
```

Classification Report:

	precision	recall	f1-score	support
0	0.79	0.78	0.78	99
1	0.61	0.62	0.61	55
accuracy			0.72	154
macro avg	0.70	0.70	0.70	154
weighted avg	0.72	0.72	0.72	154

XGBoost

Accuracy: 0.6883

Confusion Matrix:
[[70 29]
[19 36]]

Classification Report:		precision	recall	f1-score	support
	0	0.79	0.71	0.74	99
	1	0.55	0.65	0.60	55
	accuracy			0.69	154
	macro avg	0.67	0.68	0.67	154
	weighted avg	0.70	0.69	0.69	154

SVM

Accuracy: 0.7597

Confusion Matrix:
[[81 18]
[19 36]]

Classification Report:		precision	recall	f1-score	support
	0	0.81	0.82	0.81	99
	1	0.67	0.65	0.66	55
	accuracy			0.76	154
	macro avg	0.74	0.74	0.74	154
	weighted avg	0.76	0.76	0.76	154

K-Nearest Neighbors

Accuracy: 0.6948

Confusion Matrix:
[[79 20]
[27 28]]

Classification Report:		precision	recall	f1-score	support
	0	0.75	0.80	0.77	99
	1	0.58	0.51	0.54	55
	accuracy			0.69	154
	macro avg	0.66	0.65	0.66	154
	weighted avg	0.69	0.69	0.69	154

Naive Bayes

Accuracy: 0.7662

Confusion Matrix:
[[79 20]
[16 39]]

Classification Report:		precision	recall	f1-score	support
	0	0.83	0.80	0.81	99
	1	0.66	0.71	0.68	55
	accuracy			0.77	154
	macro avg	0.75	0.75	0.75	154

Classification Report:

	precision	recall	f1-score	support
0	0.83	0.80	0.81	99
1	0.66	0.71	0.68	55
accuracy			0.77	154
macro avg	0.75	0.75	0.75	154
weighted avg	0.77	0.77	0.77	154

----- Logistic Regression

Mean Accuracy: 0.7589

Standard Deviation: 0.0374

----- Random Forest

Mean Accuracy: 0.7589

Standard Deviation: 0.0246

----- XGBoost

Mean Accuracy: 0.7492

Standard Deviation: 0.0251

----- SVM

Mean Accuracy: 0.7654

Standard Deviation: 0.0316

----- K-Nearest Neighbors

Mean Accuracy: 0.7411

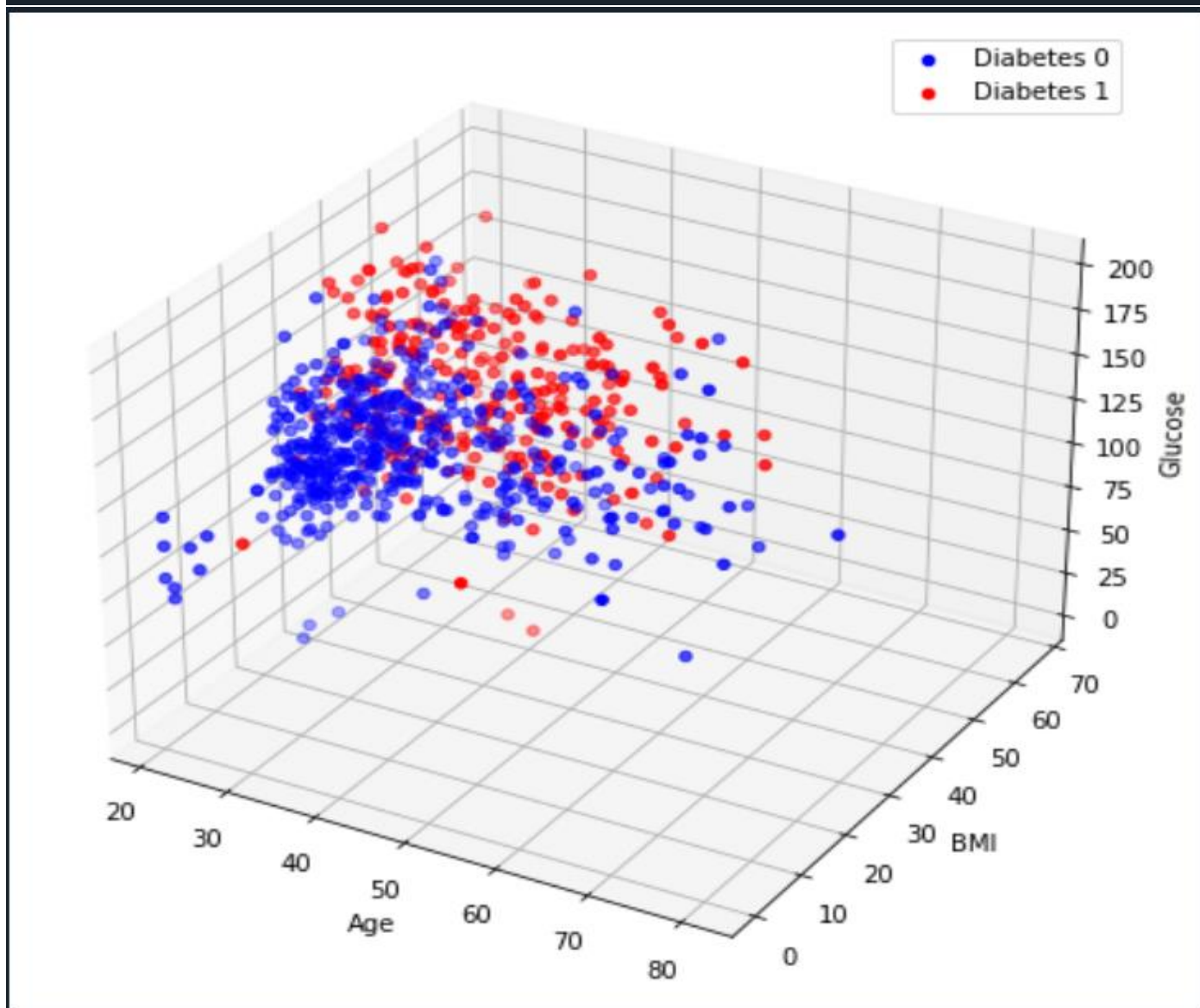
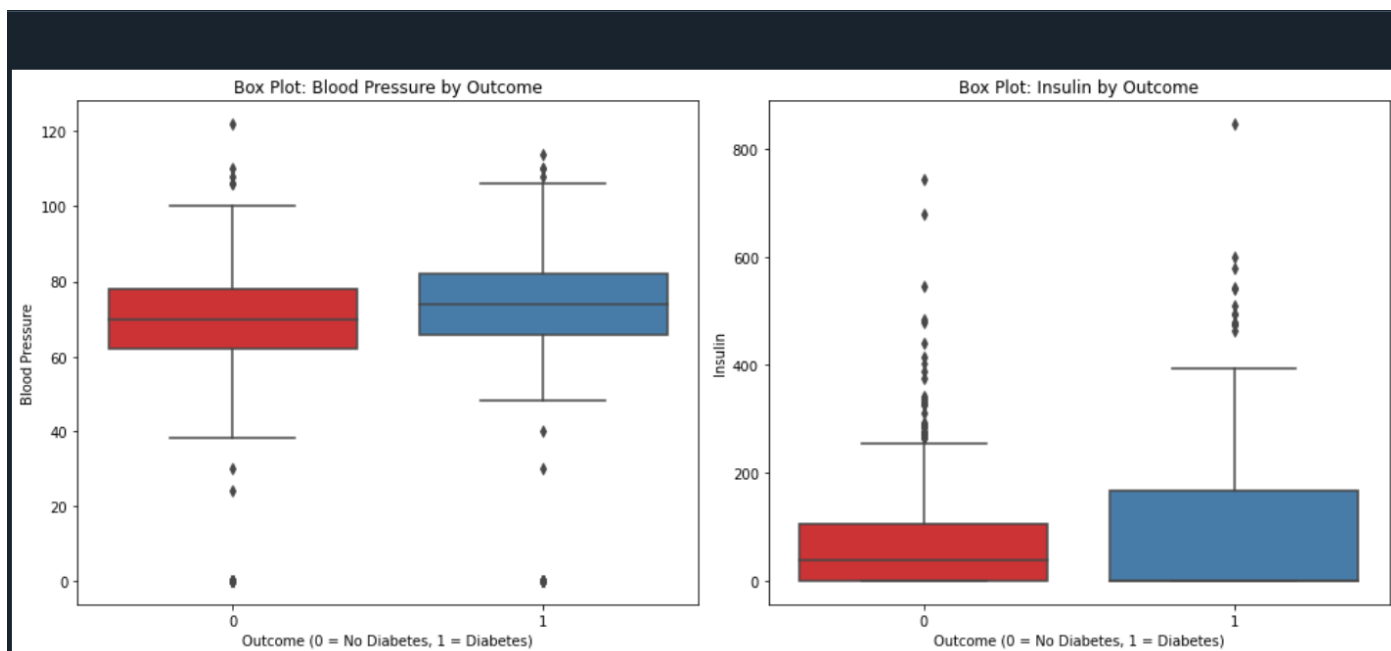
Standard Deviation: 0.0507

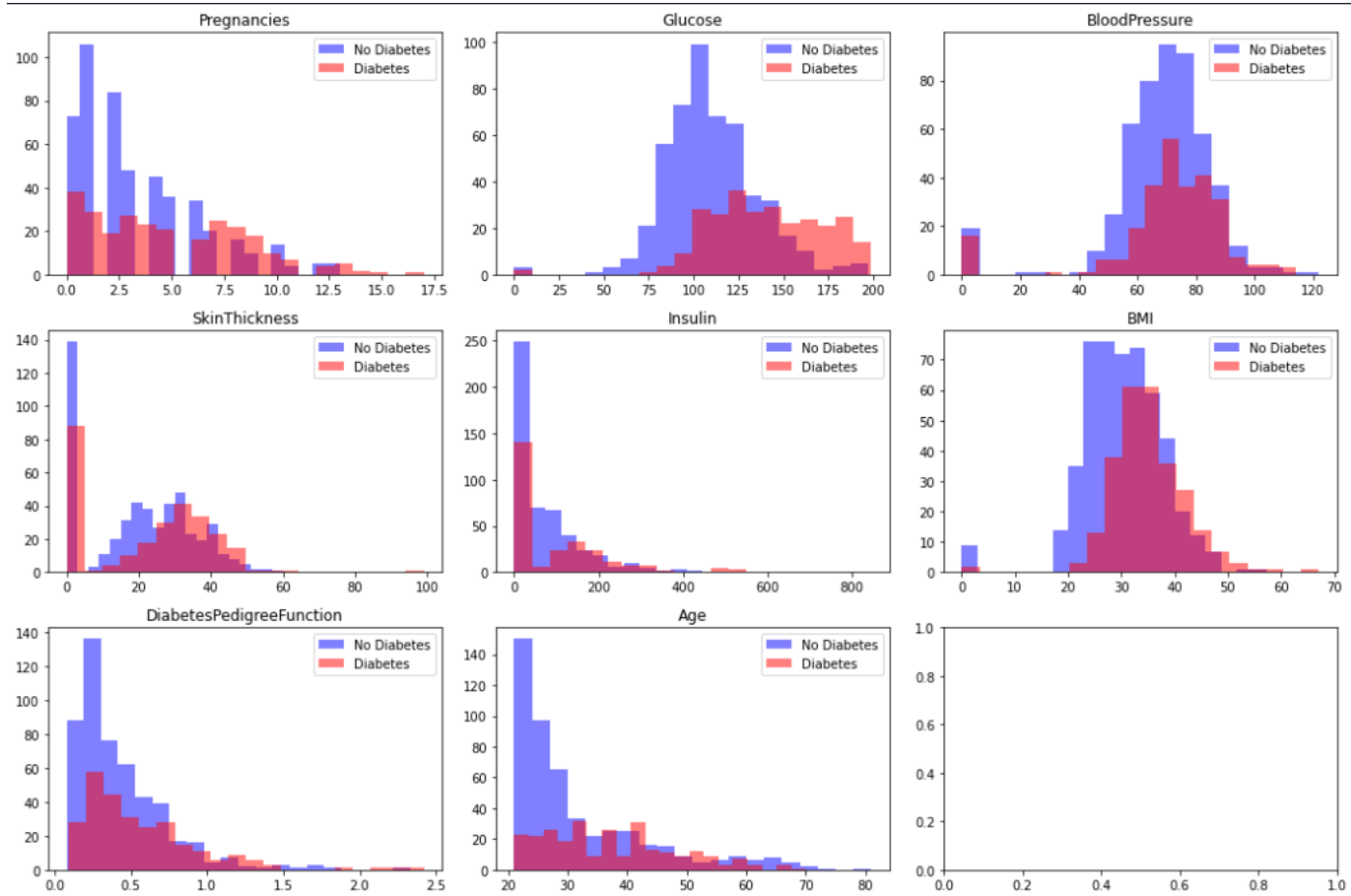
----- Naive Bayes

Mean Accuracy: 0.7362

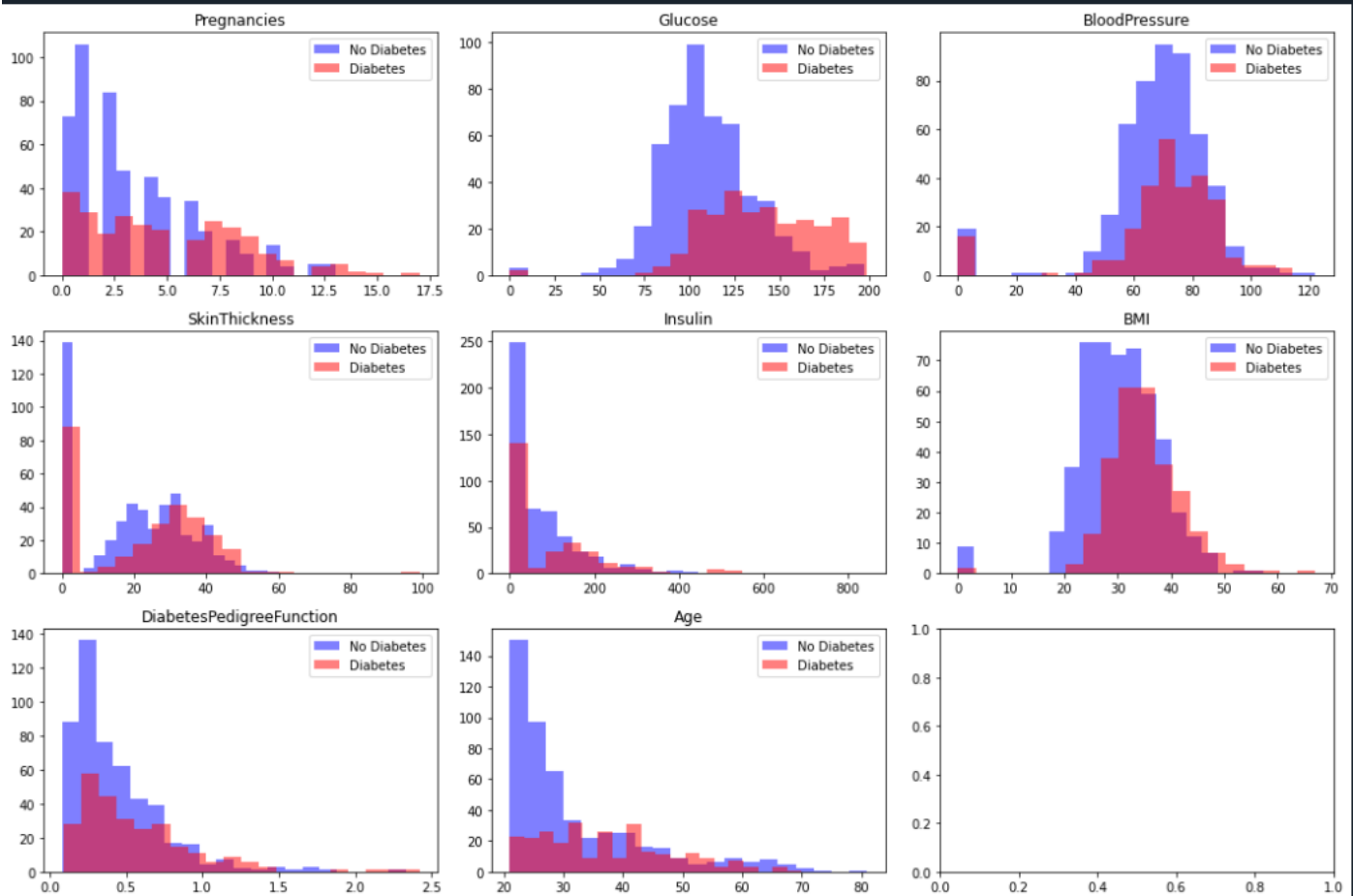
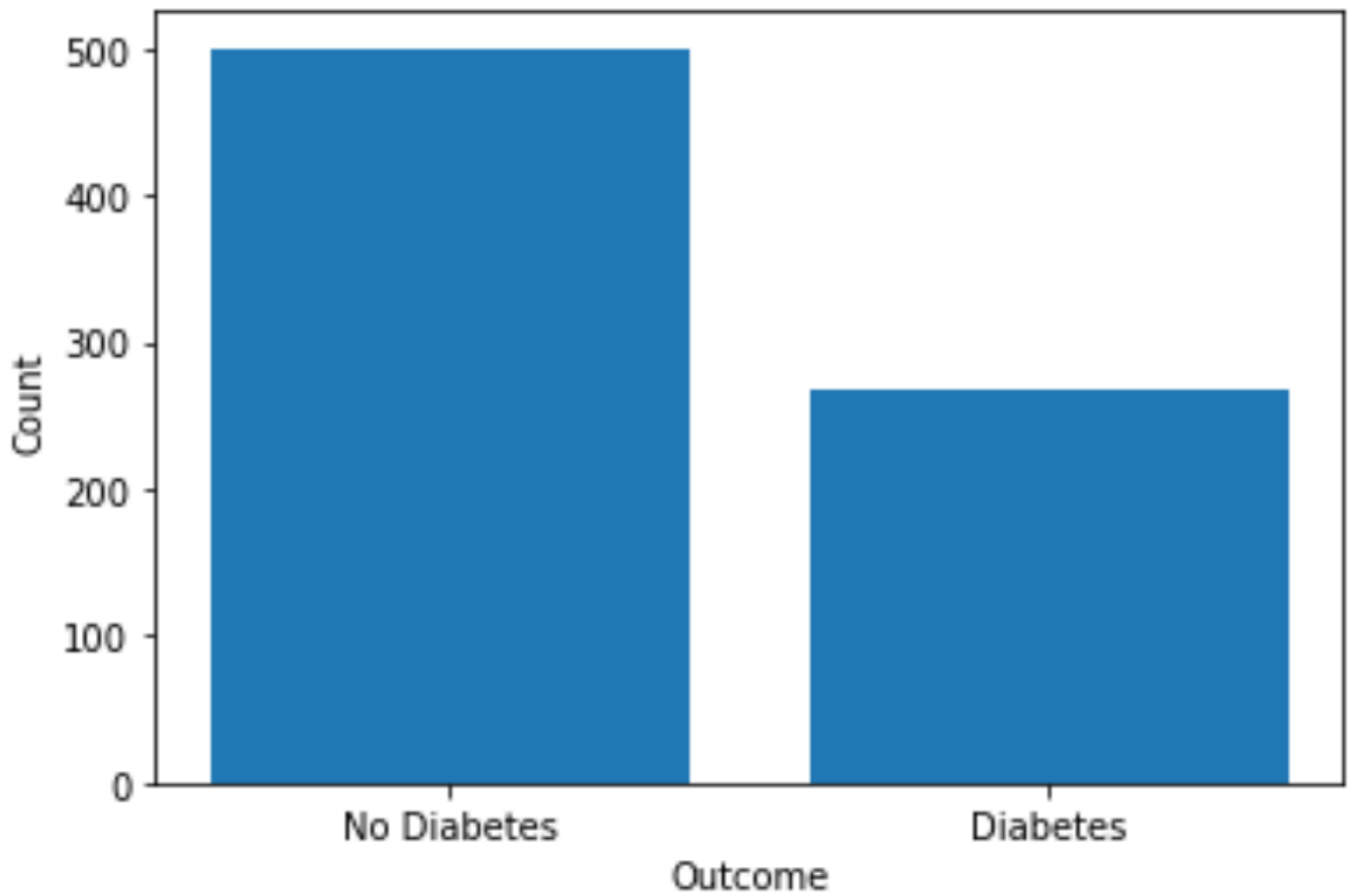
Standard Deviation: 0.0298

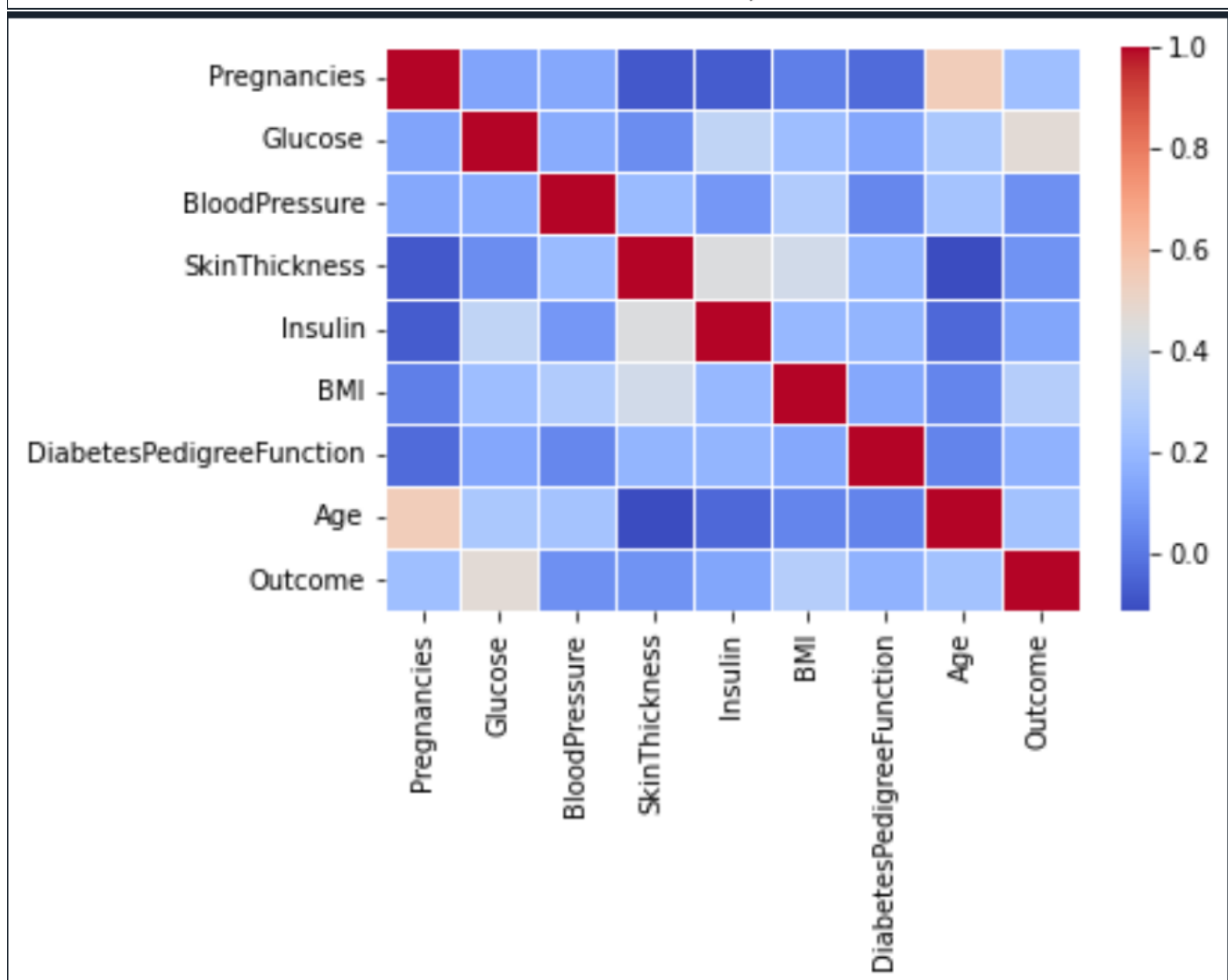
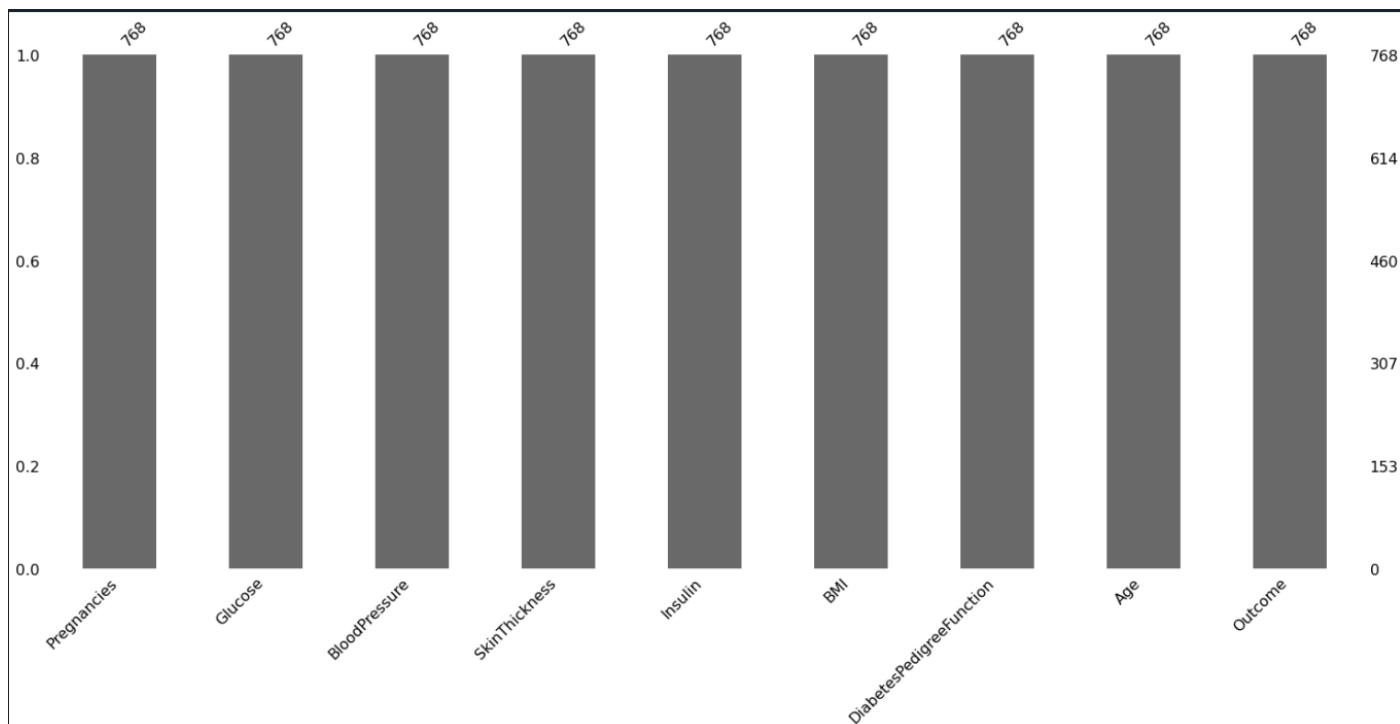
Graphs:





Distribution of Outcome (Diabetes vs. No Diabetes)





TOOLS USED FOR PROGRAMMING:

SPYDER:

Spyder is a free and open-source scientific environment written in Python, for Python, and designed by and for scientists, engineers and data analysts. It features a unique combination of the advanced editing, analysis, debugging, and profiling functionality of a comprehensive development tool with the data exploration, interactive execution, deep inspection, and beautiful visualization capabilities of a scientific package.



CONCLUSION:

In this project, we conducted a comprehensive analysis of a dataset comprising eight medical predictor variables and one target variable, 'Outcome.' These predictor variables encompassed pregnancy, glucose levels, blood pressure, skin thickness, insulin levels, BMI, diabetes prevalence function, and age, collectively aiding in predicting the presence or absence of diabetes in patients. Our exploratory data analysis (EDA) journey commenced with a meticulous examination of the dataset's characteristics, revealing eight features and 768 rows of data. Subsequent evaluation of data types confirmed the data's integrity and correctness, rendering any alterations unnecessary.

Further insights were garnered through a correlation heatmap, unveiling interrelationships between variables such as pregnancies and age, skin thickness, and BMI. A thorough assessment for data redundancy and missing values yielded no irregularities, instilling confidence in proceeding with data visualization. To elucidate the distribution of outcomes, we created plots, revealing a notable disparity: patients identified as non-diabetic outnumbered their diabetic counterparts by a substantial margin.

We constructed subplots featuring histograms to visualize feature distributions while distinguishing between 'No Diabetes' and 'Diabetes' cases within the dataset. A pairplot was also employed to unveil pairwise relationships among all features. Utilizing violin and box plots, we delved deeper into understanding these relationships and the dataset's intricacies, laying the foundation for subsequent machine learning endeavors.

In the machine learning phase, we initiated by partitioning the data into 80% for training and 20% for testing, followed by standardizing the features for optimal modeling performance. Addressing the imbalance issue—where 'No Diabetes' cases considerably outnumbered 'Diabetes' cases—we applied the Synthetic Minority Over-sampling Technique (SMOTE) to achieve balance.

Finally, we employed model selection techniques to identify the most accurate predictive model for our scenario. A range of models, including Logistic Regression, Random Forest, XGBoost, SVM, KNN, and Naive Bayes, were assessed. Through rigorous 10-fold cross-validation, we determined that the Random Forest and XGBoost models exhibited the highest accuracy, thereby concluding our project on a promising note.