RISK FACTOR ANALYSIS AND PRECISION PREDICTION MODEL FOR DIABETES RISK ASSESSMENT

Joseph Anifowose, Kalp Patel, Eshani Shah, Lahari Kaja, Ushaswini Sunkara

fanifowo@iu.edu, kalpate@iu.edu, eshshah@iu.edu, lkaja@iu.edu, usunkara@iu.edu

Indiana University-Purdue University, Indianapolis, IN 46202, USA

ABSTRACT: Diabetes mellitus is a widespread health issue in the United States, affecting individuals of all ages and posing early indications of coronary heart disease while significantly impacting morbidity and mortality rates. This study aims to analyze the "Healthcare Diabetes" dataset to ascertain crucial risk factors associated with diabetes and construct an accurate predictive model for its occurrence. The project's primary objective involves in-depth exploration of the dataset, emphasizing factors like age, BMI, blood pressure, familial diabetes history, glucose, and insulin levels. Employing Python libraries such as Pandas, SQL, Matplotlib, Seaborn, and Plotly, the methodology encompasses descriptive data analysis, exploratory data analysis (EDA), and statistical tests for hypothesis validation. Descriptive statistics, including mean, median, and standard deviation, coupled with data visualization tools, facilitate trend identification and anomaly detection. Statistical analysis using NumPy aids in validating hypotheses and preparing data for machine learning modelling. Logistic regression, random forest, and Naïve Bayes models are utilized to uncover relationships between health-related attributes and the likelihood of diabetes. The project's hypothesis examines the influential role of these attributes on diabetes risk. Deliverables encompass descriptive statistics, a binary classification model for risk prediction, and performance evaluation using Scikit-Learn metrics. Expected outcomes entail the development of a precise diabetes prediction tool, identification of pivotal risk factors, and actionable insights for healthcare interventions and policy formulation. In conclusion, this study endeavors to offer a robust predictive model for assessing diabetes risk, providing critical insights into key risk factors and offering a foundation for proactive healthcare strategies and informed public health policies.

Keywords: Diabetes Risk Factors, Predictive Modeling, Health Data Analysis, Machine Learning, Public Health Impact, Healthcare Interventions

1. Project Scope

1.1 Introduction

Diabetes mellitus, a prevalent health concern across diverse demographics in the United States, serves as an early indicator of coronary heart disease and contributes significantly to morbidity and mortality rates. The projected rise in diabetes cases underscores the urgency to comprehend its risk factors and establish proactive measures for early intervention. This study focuses on analyzing the "Healthcare Diabetes" dataset to discern key health-related attributes associated with diabetes and construct a robust predictive model. The overarching aim of this project is to develop a comprehensive understanding of the pivotal factors influencing diabetes occurrence, encompassing parameters such as age, BMI, blood pressure, familial diabetes history, glucose, and

insulin levels. Leveraging advanced methodologies in data analysis and machine learning, this study seeks to unravel intricate relationships within the dataset to accurately predict an individual's risk of developing diabetes.

By employing descriptive data analysis, exploratory data analysis (EDA), and statistical tests, this study aims to uncover patterns, anomalies, and significant correlations within the dataset. Utilizing machine learning models including logistic regression, random forest, and Naïve Bayes, the objective is to establish a robust prediction tool for diabetes risk assessment.

The anticipated outcomes of this research include the creation of a precise predictive model, identification of critical risk factors influencing diabetes susceptibility, and insights pivotal for guiding healthcare interventions and formulating informed public health policies. This study holds the potential to provide valuable insights into diabetes prevention and management, contributing to enhanced healthcare outcomes and proactive measures in addressing this significant health challenge.

1.2 Aim

The primary aim of this study involves a comprehensive examination of the "Healthcare Diabetes" dataset to discern and comprehend essential health-related attributes linked with diabetes. The primary focus is on establishing a reliable predictive model capable of accurately predicting an individual's likelihood of developing diabetes, drawing insights from various health-related characteristics within the dataset. The investigation targets factors such as age, BMI, blood pressure, familial diabetes history, glucose, and insulin levels. The objective is to develop an efficient predictive tool facilitating early detection and proactive interventions for better healthcare outcomes. Ultimately, this endeavor aims to contribute to improved strategies for diabetes prevention and management through informed insights derived from the predictive model.

1.3 Purpose

The project's purpose revolves around attribute identification, predictive model development, and the utilization of insights for enhancing healthcare strategies concerning diabetes prevention and management.

Attribute Identification: The project intends to thoroughly examine the "Healthcare Diabetes" dataset to pinpoint vital health-related attributes linked with diabetes, encompassing elements such as age, BMI, blood pressure, familial diabetes history, glucose, and insulin levels.

Predictive Model Development: The goal is to create a reliable predictive model that accurately anticipates an individual's likelihood of developing diabetes by utilising a variety of health-related characteristics found within the dataset. This model aims to enable early detection and proactive interventions, thereby enhancing healthcare outcomes.

Healthcare Enhancement: The project aims to provide valuable insights and tools crucial for shaping strategies concerning diabetes prevention and management. By utilizing the predictive model's discoveries, it seeks to direct healthcare interventions and contribute to policy formulation, with the goal of improving healthcare outcomes associated with diabetes.

1.4 Research Hypothesis

Null Hypothesis:

There is no significant relationship between our independent variables (history of pregnancy, skin thickness, glucose levels, blood pressure, insulin, BMI, diabetes pedigree function, age) and their significant influence on the likelihood of developing diabetes.

Alternate Hypothesis:

There is a significant relationship between at least one of the independent health-related attributes and the likelihood of an individual developing diabetes.

2. Methodology

2.1 Steps of the project

The methodology outlines the step-by-step approach, starting from data collection and preprocessing to the implementation of machine learning models and hypothesis testing, ultimately leading to the project's deliverables.

- Descriptive Statistics
- Data cleaning
- Exploratory data analysis
- Data visualization
- Statistical analysis
- Machine learning models
- Performance analysis

2.2. Team members and their responsibilities

Name	Responsibility
Joseph Anifowose	Data collection and cleaning
Ushaswini Sunkara	Data cleaning and visualization

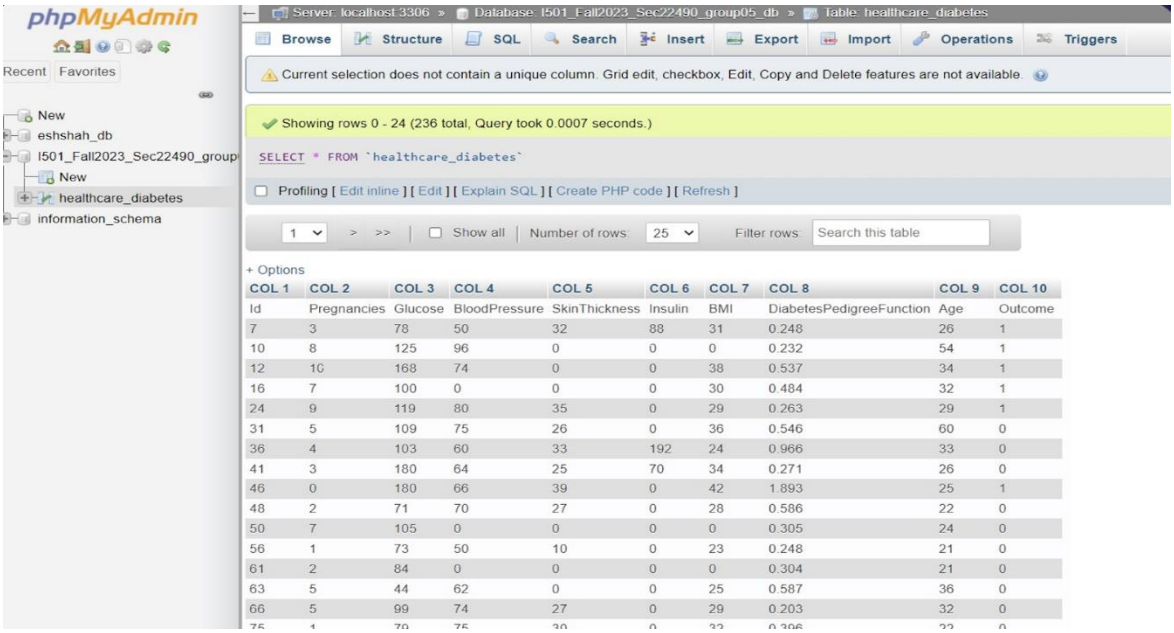
Lahari Kaja	Exploratory data analysis and Normality testing
Kalp Patel	Statistical analysis
Eshani Shah	Machine learning models

3. Data Collection

The healthcare diabetes dataset used for this project was obtained from Kaggle, specifically sourced from the dataset repository created by Nandita Pore. Accessible at the URL https://www.kaggle.com/datasets/nanditapore/healthcare-diabetes, this dataset served as the foundational source of health-related information. It encompasses a wide array of attributes pertinent to diabetes, including age, BMI, blood pressure, glucose, insulin levels, and familial diabetes history. Collected from Kaggle's platform, this dataset provided a comprehensive pool of diverse health-related features associated with diabetes occurrence. Its reliability and accessibility via Kaggle facilitated the thorough analysis and exploration of various health parameters critical for understanding and predicting the risk factors associated with diabetes within the scope of this project.

4. Data Importing

The Healthcare Diabetes Dataset was seamlessly imported into phpMyAdmin by initiating the process with the creation of a new database. Within the phpMyAdmin interface, the designated database was selected, and the Import tab was accessed. Through this tab, the dataset file was chosen for importation. Upon selection, the import process was executed, enabling the incorporation of the dataset into the database. This importation procedure ensured the dataset's availability within phpMyAdmin's database management system, facilitating efficient analysis and manipulation for further exploration and utilization in subsequent project phases.



COL 1	COL 2	COL 3	COL 4	COL 5	COL 6	COL 7	COL 8	COL 9	COL 10
Id	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
7	3	78	50	32	88	31	0.248	26	1
10	8	125	96	0	0	0	0.232	54	1
12	10	168	74	0	0	38	0.537	34	1
16	7	100	0	0	0	30	0.484	32	1
24	9	119	80	35	0	29	0.263	29	1
31	5	109	75	26	0	36	0.546	60	0
36	4	103	60	33	192	24	0.966	33	0
41	3	180	64	25	70	34	0.271	26	0
46	0	180	66	39	0	42	1.893	25	1
48	2	71	70	27	0	28	0.586	22	0
50	7	105	0	0	0	0	0.305	24	0
56	1	73	50	10	0	23	0.248	21	0
61	2	84	0	0	0	0	0.304	21	0
63	5	44	62	0	0	25	0.587	36	0
66	5	99	74	27	0	29	0.203	32	0

Fig1: Importing dataset to phpMyAdmin by creating a new database

5. Data Extraction

Establishing an SQL connection in Jupyter notebook enabled direct dataset access for extraction. This connection streamlined querying and facilitated efficient data retrieval for analysis within the notebook.

```
import MySQLdb
import pandas as pd

# Read MySQL credentials from file
myvars = {}
with open("eshshah-mysql-password") as myfile:
    for line in myfile:
        name, var = line.partition(":")[::2]
        myvars[name.strip()] = var.strip()

# Connect to MySQL database
conn = MySQLdb.connect(
    host="localhost",
    user=myvars['DB username'],
    passwd=myvars['DB password'],
    db=myvars['DB databasename']
)

cursor = conn.cursor()

# Select the specified database
cursor.execute('USE I501_Fall2023_Sec22490_group05_db;')

# Fetch data from the "healthcare_diabetes" table
cursor.execute('SELECT * FROM healthcare_diabetes;')
data = pd.DataFrame(list(cursor.fetchall()))

# Display the first few rows of the DataFrame
print(data.head())
```

Fig 2: Code showing the SQL connection in Jupyter notebook

The Pandas Python library was employed to read the CSV file, transforming it into a DataFrame format for comprehensive data handling. This conversion process enabled efficient manipulation

and analysis of the dataset within the Python environment, laying the groundwork for in-depth exploration and subsequent insights extraction.

IMPORTING CSV FILE AS DATAFRAME USING PANDAS

```
import pandas as pd
```

```
# reading the data frame
df = pd.read_csv('Healthcare-Diabetes.csv')# often works
# df = pd.read_csv('Healthcare-Diabetes.csv', header=0, quotechar='"',sep=',', na_values = ['na', '-', '.', ''])
df
```

	Id	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	1	6	148	72	35	0	33.6	0.627	50	1
1	2	1	85	66	29	0	26.6	0.351	31	0
2	3	8	183	64	0	0	23.3	0.672	32	1
3	4	1	89	66	23	94	28.1	0.167	21	0
4	5	0	137	40	35	168	43.1	2.288	33	1
...	...	...	...	...	...	...	...	...	...	...
2763	2764	2	75	64	24	55	29.7	0.370	33	0
2764	2765	8	179	72	42	130	32.7	0.719	36	1
2765	2766	6	85	78	0	0	31.2	0.382	42	0
2766	2767	0	129	110	46	130	67.1	0.319	26	1
2767	2768	2	81	72	15	76	30.1	0.547	25	0

2768 rows × 10 columns

Fig 3: Importing a Pandas DataFrame from a CSV file

The command "df.dtypes" was executed to inspect the data types of each column within the DataFrame, revealing whether the values were represented as either floating-point numbers (float) or integers (int). This allowed for a comprehensive understanding of the data structure, aiding in subsequent data processing and analysis.

```
# Series column data types
s = df.dtypes
s
```

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

Fig 4: Code showing the data types of all the columns

- Df.shape is a method used to retrieve the dimensions of a DataFrame.
- In Pandas, a DataFrame is a two-dimensional labeled data structure with rows and columns, similar to a table or spreadsheet. The shape attribute of a DataFrame returns a tuple representing its dimensions. Specifically, it returns a tuple containing the number of rows and columns in the DataFrame, in the format (number_of_rows, number_of_columns)

```
[9]: # Number of Rows and Columns
     (r, c) = df.shape
     print('number_of_rows:', r, '\nnumber_of_columns:' ,c)
```

```
number_of_rows: 2768
number_of_columns: 10
```

Fig 5: Code that checks the number of rows and columns

- Df.size attribute is used to retrieve the total number of elements present in a DataFrame. Unlike df.shape, which returns a tuple representing the number of rows and columns, df.size simply returns a single integer value indicating the total number of elements in the DataFrame.

```
# size of the dataset
i = df.size # row-count * column-count
i
```

```
27680
```

Fig 6: Code that calculates the size of the dataset

5.1 Descriptive Statistics

The df.describe() function provides summary statistics of the numerical columns in the DataFrame. It provides us the following statistical values:

- Count: Number of non-null values in each column.
- Mean: Average value for each column.
- Std: Standard deviation, a measure of the amount of variation or dispersion in each column's values.
- Min: The minimum value in each column.
- 25%: The 25th percentile (lower quartile), where 25% of the data falls below this value.
- 50%: The median or 50th percentile, where half of the data falls below this value.
- 75%: The 75th percentile (upper quartile), where 75% of the data falls below this value.
- Max: The maximum value in each column.

```
# Descriptive statistics
(df.describe())
```

	Id	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	2768.000000	2768.000000	2768.000000	2768.000000	2768.000000	2768.000000	2768.000000	2768.000000	2768.000000	2768.000000
mean	1384.500000	3.742775	121.102601	69.134393	20.824422	80.127890	32.137392	0.471193	33.132225	0.343931
std	799.197097	3.323801	32.036508	19.231438	16.059596	112.301933	8.076127	0.325669	11.777230	0.475104
min	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	692.750000	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.244000	24.000000	0.000000
50%	1384.500000	3.000000	117.000000	72.000000	23.000000	37.000000	32.200000	0.375000	29.000000	0.000000
75%	2076.250000	6.000000	141.000000	80.000000	32.000000	130.000000	36.625000	0.624000	40.000000	1.000000
max	2768.000000	17.000000	199.000000	122.000000	110.000000	846.000000	80.600000	2.420000	81.000000	1.000000

Fig 7: Descriptive statistics for all the columns

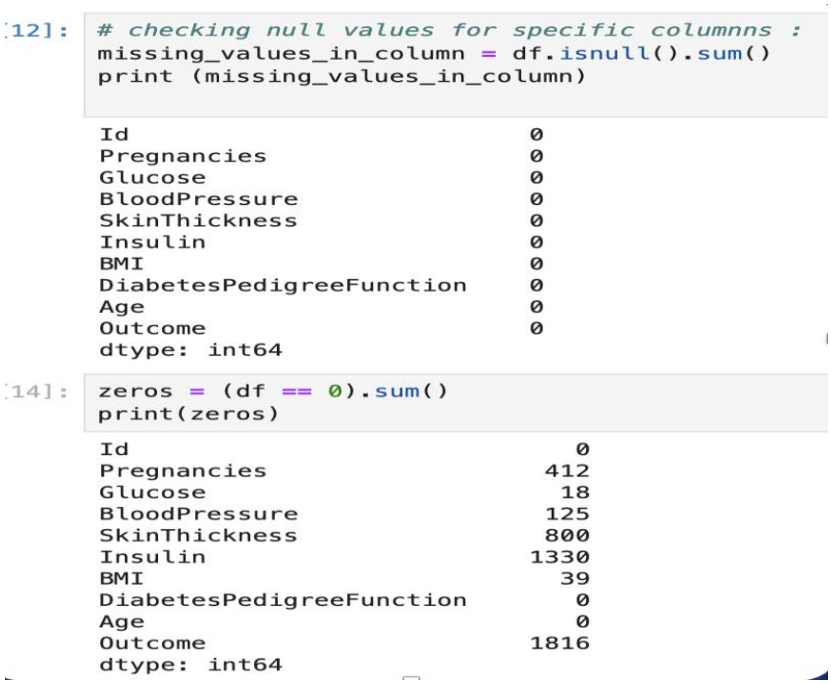
6. Data Cleaning

6.1 Checking for Null and missing values

After conducting an assessment for null (missing) values in each column of our pandas DataFrame (df), we confirmed the absence of any such instances. Our diligent examination indicated that all potentially missing values had been diligently substituted with 'zeroes', ensuring a complete dataset with no explicit null entries present.

Following this verification, our analysis delved further into the data by computing and unveiling the count of zero values existing within each column of the DataFrame (df). This step allowed us to discern the prevalence and distribution of zero values across the dataset. By scrutinizing the

counts within each column, we gained insight into the frequency and potential significance of these zeros within the context of the dataset's various attributes.



```
[12]: # checking null values for specific columnns :
      missing_values_in_column = df.isnull().sum()
      print (missing_values_in_column)

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

[14]: zeros = (df == 0).sum()
      print(zeros)

Id                          0
Pregnancies               412
Glucose                    18
BloodPressure             125
SkinThickness             800
Insulin                  1330
BMI                        39
DiabetesPedigreeFunction    0
Age                         0
Outcome                  1816
dtype: int64
```

Fig 8: Code that checks for null values and zeroes

6.2 Addressing Outliers

In pursuit of a more accurate and resilient analysis, our next crucial step involves addressing outliers to safeguard the integrity of our data. Outliers, being extreme values within the dataset, can significantly influence statistical measures and model performance. To effectively mitigate their impact, we've employed boxplots, a visual tool enabling us to inspect the distribution of data points across different variables. This graphical representation facilitates the identification and evaluation of potential outliers, empowering us to make informed decisions regarding outlier detection.

Subsequently, to actively address these outliers, we've implemented a statistical technique known as Winsorization. Winsorization is a method utilized to limit the influence of outliers by reducing their impact on the dataset. This approach involves capping extreme values at a predefined percentile threshold, effectively replacing values beyond this threshold with values closer to the threshold itself. By implementing Winsorization, we aim to minimize the disruptive effects of outliers on subsequent analyses, ensuring a more robust and reliable dataset for our analytical processes. Winsorization was performed for columns such as Insulin, BMI, Blood Pressure, Age, Diabetes Pedigree Function in the dataset. We plotted box plots before and after winsorization to check for outliers.

```python
# Treating outliers by Winsorization
import numpy as np
from scipy.stats.mstats import winsorize
import pandas as pd

columns_of_interest = ['Insulin', 'BMI', 'SkinThickness', 'BloodPressure', 'Age','DiabetesPedigreeFunction']

# Set the winsorizing limits (capping at the 5th and 95th percentiles)
winsorizing_limits = [0.05, 0.05]

# Winsorize each specified column
for column in columns_of_interest:
    df[column] = winsorize(df[column], limits=winsorizing_limits)

# Print or use the winsorized DataFrame
print(df)
```

Fig 9: Code that shows winsorization

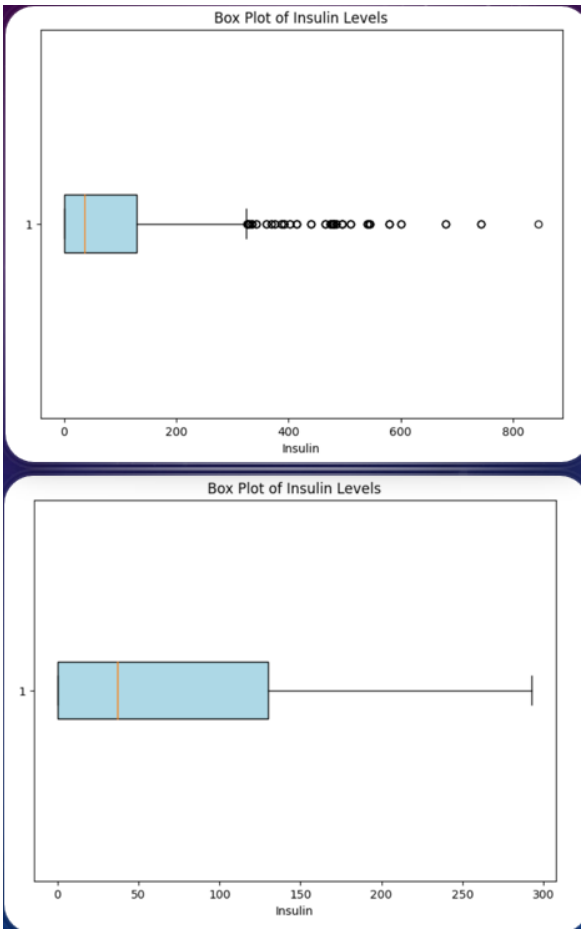
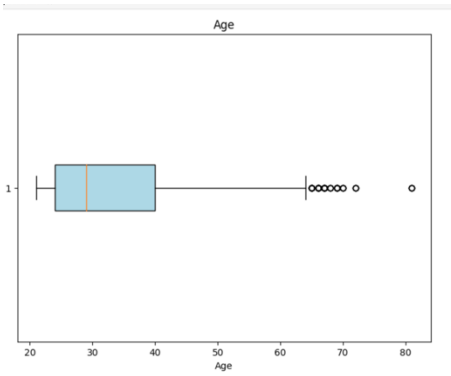


Fig 10: Boxplots before and after winsorization



6.3 Imputing zeroes

To replace zero values in each column of a DataFrame with the mean of the non-zero values, we computed the mean for each column excluding zeros. Next, iterating through each column, we replaced the zeros with this mean. This process ensures that zero values are substituted with a more representative value based on the non-zero data distribution. By employing this approach, you effectively mitigate the potential distortion of statistical measures caused by zero entries in the dataset, providing a more accurate summary of the data's central tendencies and variability.

df.describe() was done to evaluate the descriptive statistics such as count, mean, std, min, max, and quartiles after addressing outliers and imputing zeroes with mean.

```
EXPLORATORY DATA ANALYSIS : REPLACING THE ZEROES WITH MEAN FOR THE REQUIRED COLUMNS
```

```
# replacing zeroes with mean for specified columns
import numpy as np

num_cols = ['Glucose', 'BloodPressure', 'SkinThickness', 'BMI', 'DiabetesPedigreeFunction', 'Insulin']

for col in num_cols:
    mean_val = df[col].mean()
    df[col] = df[col].replace(0, mean_val)

print(df[num_cols].isin([0]).sum())
```

```
Glucose                   0
BloodPressure             0
SkinThickness             0
BMI                       0
DiabetesPedigreeFunction  0
Insulin                   0
dtype: int64
```

Fig 11: Code that replaces zeroes with column mean

	Id	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	2768.000000	2768.000000	2768.000000	2768.000000	2768.000000	2768.000000	2768.000000	2768.000000	2768.000000	2768.000000
mean	1384.500000	3.742775	121.102601	70.502529	20.482298	73.448699	32.244328	0.458191	32.819725	0.343931
std	799.197097	3.323801	32.036508	12.664216	15.279579	90.524390	6.396167	0.276790	10.955887	0.475104
min	1.000000	0.000000	0.000000	40.000000	0.000000	0.000000	21.800000	0.141000	21.000000	0.000000
25%	692.750000	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.244000	24.000000	0.000000
50%	1384.500000	3.000000	117.000000	72.000000	23.000000	37.000000	32.200000	0.375000	29.000000	0.000000
75%	2076.250000	6.000000	141.000000	80.000000	32.000000	130.000000	36.625000	0.624000	40.000000	1.000000
max	2768.000000	17.000000	199.000000	90.000000	44.000000	293.000000	44.600000	1.136000	58.000000	1.000000

Fig 12: Descriptive statistics after replacing zeroes and removing outliers

7. Exploratory Data analysis:

After data cleaning, we conducted exploratory data analysis on our project, focusing on data visualization techniques using Seaborn and Matplotlib. Specifically, we created frequency distribution histograms for each independent variable in our dataset. These visualizations provided insights into the distribution of key features, including the diabetes pedigree function, age, skin thickness, and insulin.

```
# Creating subplots of histogram of specified columns showing their frequency distribution
import seaborn as sns
import matplotlib.pyplot as plt
# Set the figure size
plt.figure(figsize=(12, 16))
# Histogram for DiabetesPedigreeFunction
plt.subplot(4, 2, 1)
sns.histplot(data=df, x='DiabetesPedigreeFunction', bins=30, kde=True)
plt.title('DiabetesPedigreeFunction Distribution')
plt.xlabel('DiabetesPedigreeFunction')
plt.ylabel('Frequency')
# Histogram for Age
plt.subplot(4, 2, 2)
sns.histplot(data=df, x='Age', bins=30, kde=True)
plt.title('Age Distribution')
plt.xlabel('Age')
plt.ylabel('Frequency')
# Histogram for SkinThickness
plt.subplot(4, 2, 3)
sns.histplot(data=df, x='SkinThickness', bins=30, kde=True)
plt.title('SkinThickness Distribution')
plt.xlabel('SkinThickness')
plt.ylabel('Frequency')
# Histogram for Insulin
plt.subplot(4, 2, 4)
sns.histplot(data=df, x='Insulin', bins=30, kde=True)
plt.title('Insulin Distribution')
plt.xlabel('Insulin')
plt.ylabel('Frequency')
```

```
# Histogram for BMI
plt.subplot(4, 2, 5)
sns.histplot(data=df, x='BMI', bins=30, kde=True)
plt.title('BMI Distribution')
plt.xlabel('BMI')
plt.ylabel('Frequency')
# Adjust layout and show the plot
plt.tight_layout()
plt.show()
```

Fig 13: Code snippet showing the frequency histogram plotting

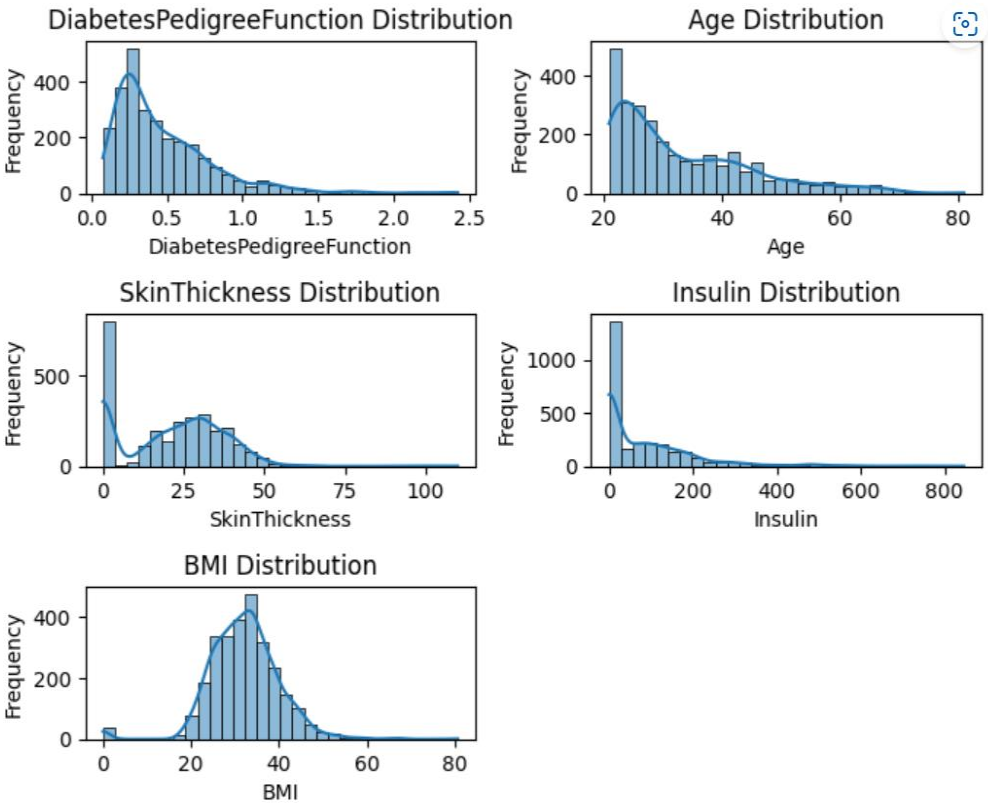


Fig 14: Frequency Distribution Histogram

Upon analyzing the histograms, we observed that the frequency distributions for the diabetes pedigree function, age, skin thickness, and insulin variables exhibit a right-skewed pattern. This suggests that a significant portion of the data points in these variables tends to cluster towards lower values, with a tail extending towards higher values. Understanding the skewness of these variables is crucial for gaining insights into the underlying patterns within our dataset. By leveraging Seaborn and Matplotlib, we have effectively communicated the distributional characteristics of key independent variables, paving the way for further analysis and decision-making in our project. These visualizations serve as a valuable tool for identifying trends, patterns, and potential outliers within the data, aiding in the formulation of informed strategies and hypotheses.

Stacked Histogram:

We have expanded our project's data visualization by integrating stacked histograms to explore the connection between each independent variable and the outcome (whether an individual is diabetic or non-diabetic). For this analysis, we employed Seaborn and Matplotlib to generate visual representations that vividly depict the interactions between independent variables and the outcome variable.

```python
# Creating subplots of histogram of specified columns by Outcome
import pandas as pd
import matplotlib.pyplot as plt
# Separate data based on Outcome
outcome_0 = df[df['Outcome'] == 0]
outcome_1 = df[df['Outcome'] == 1]
# List of numerical columns to plot
numerical_columns = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age']
# Plot histograms for each numerical column
plt.figure(figsize=(15, 10))
for i, column in enumerate(numerical_columns, 1):
    plt.subplot(3, 3, i)
    plt.hist(outcome_0[column], bins=20, label='Outcome 0', alpha=0.5, color='blue')
    plt.hist(outcome_1[column], bins=20, label='Outcome 1', alpha=0.5, color='orange')
    plt.title(f'Histogram of {column} by Outcome')
    plt.xlabel(column)
    plt.ylabel('Frequency')
    plt.legend()
plt.tight_layout()
plt.show()
```

Fig 15: code snippet showing the stacked histogram plotting

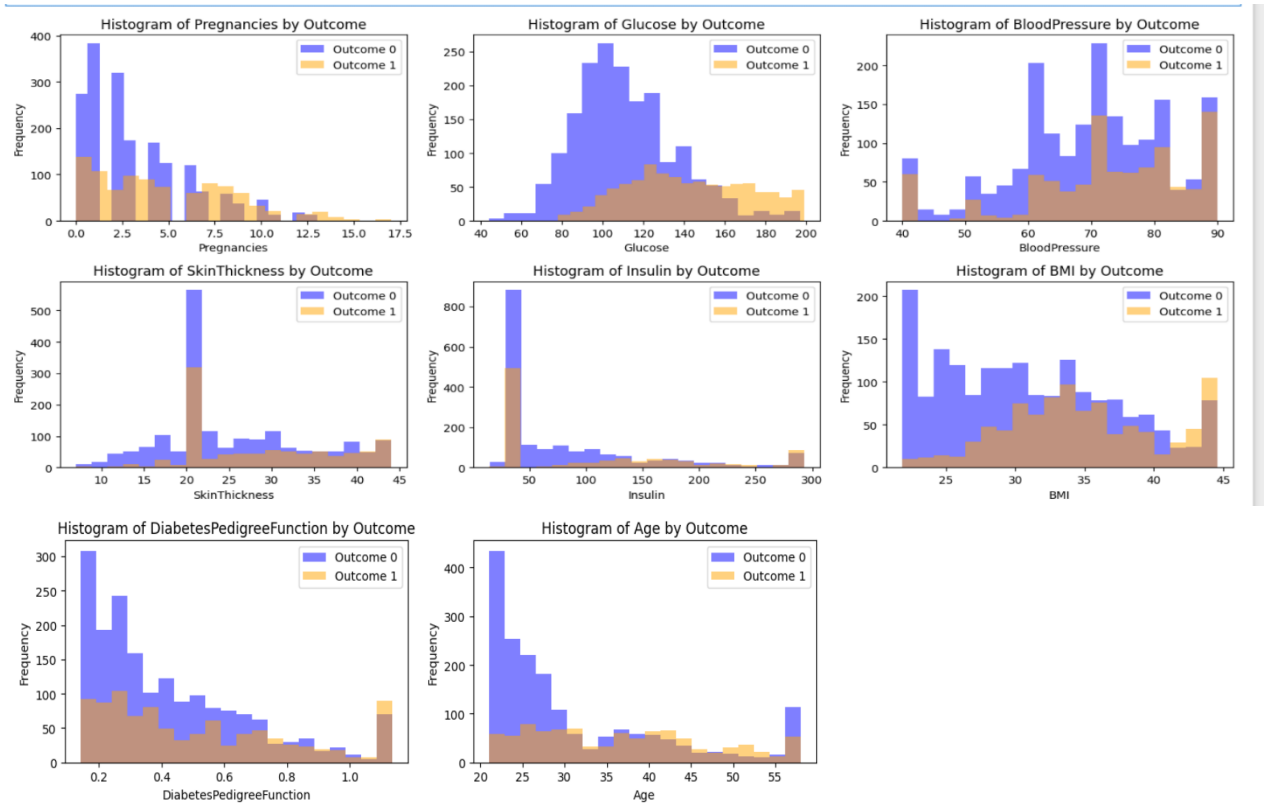


Fig: 16: Stacked histograms of independent variables with dependent variable (outcome)

The stacked histograms reveal a notable linear relationship between each independent variable and the outcome. Specifically, as the values of individual independent variables increase, there is a corresponding increase in the ratio of diabetic to non-diabetic cases. This insight is crucial for understanding how variations in independent variables may influence the likelihood of an individual being diabetic. These visualizations provide a clear and intuitive depiction of the impact of independent variables on the outcome variable, offering a valuable foundation for further analysis and interpretation.

Bar Graph:

We have further enhanced our project's data visualization by incorporating a bar plot specifically designed to illustrate the frequency distribution of the outcome variable. For this visualization, we utilized the Matplotlib library to create a bar plot that distinctly depicts the number of diabetic and non-diabetic cases in our dataset.

```
# Barplot showing Diabetes distribution based on Outome
import pandas as pd
import matplotlib.pyplot as plt
# Group the data by the 'Outcome' column and count the occurrences of 0 (no diabetes) and 1 (diabetes)
outcome_counts = df['Outcome'].value_counts()
# Create a bar chart
plt.figure(figsize=(8, 6))
bars = plt.bar(outcome_counts.index, outcome_counts.values, color=['skyblue', 'lightcoral'])
# Annotate each bar with its value count
for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, yval + 0.1, round(yval, 2), ha='center', va='bottom')
# Customize the plot
plt.title('Diabetes Distribution')
plt.xlabel('Outcome')
plt.ylabel('Count')
plt.xticks(outcome_counts.index, ['No Diabetes (0)', 'Diabetes (1)'])
# Show the bar chart
plt.show()
```

Fig 17: code snippet showing bar plot plotting

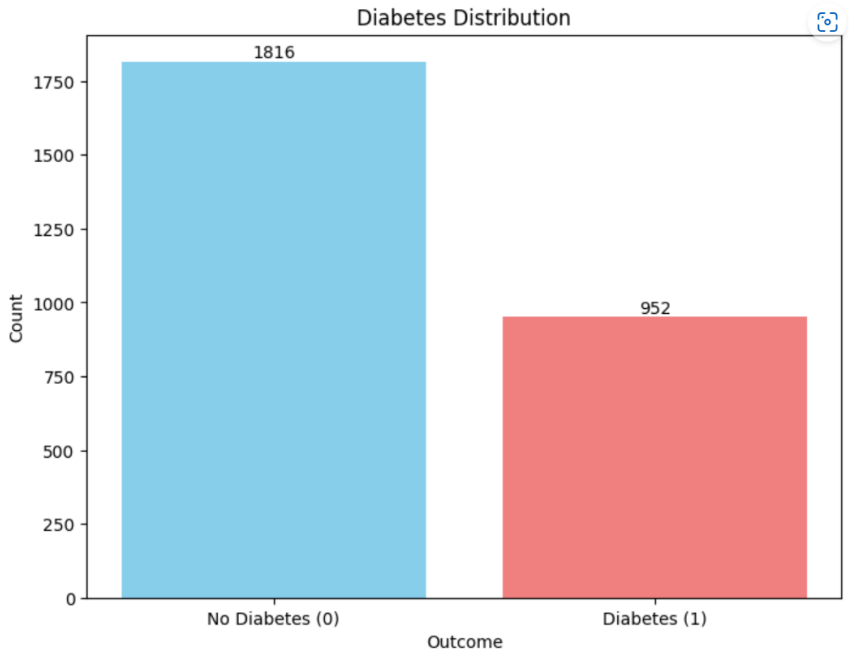


Fig:18: Bar plot of Diabetes distribution

The bar plot effectively communicates that there are 1,816 non-diabetic cases and 952 diabetic cases in our dataset. This straightforward representation allows for a quick and clear understanding of the distribution of the outcome variable, shedding light on the balance or imbalance between the two classes.

Correlation matrix

We have conducted a comprehensive analysis of the correlation between features and the outcome variable in our project. To visually represent these relationships, we employed correlation heatmaps and barplot, utilizing the Matplotlib library.

```python
#Creating a heatmap to show the corelation matrix amongst independant and dependant variables
import pandas as pd
import matplotlib.pyplot as plt
# Remove the 'Id' column
df = df.drop(columns=['Id'])
# Calculate the correlation matrix for all remaining columns
correlation_matrix = df.corr()
# Define the number of rows and columns for the grid
n_rows, n_cols = correlation_matrix.shape
# Create a pink-colored correlation matrix heatmap with labeled cells, a larger size, and an increased grid size
plt.figure(figsize=(16, 12))
cax = plt.matshow(correlation_matrix, cmap='coolwarm', vmin=-1, vmax=1, aspect='auto', origin='lower',
                  extent=[-0.5, n_cols - 0.5, -0.5, n_rows - 0.5])  # Adjust grid size
cax.set_cmap('coolwarm_r')  # Adjust the colormap to get a pinkish appearance
plt.colorbar(cax)
# Set column and row labels
plt.xticks(range(n_cols), correlation_matrix.columns, rotation=90)
plt.yticks(range(n_rows), correlation_matrix.columns)
# Label the cells with correlation coefficients
for i in range(n_rows):
    for j in range(n_cols):
        plt.text(j, i, f"{correlation_matrix.iloc[i, j]:.2f}", ha='center', va='center', color='black')
plt.title('Correlation matrix')
# Show the heatmap
plt.show()
```

Fig 19: code snippet showing correlation matrix

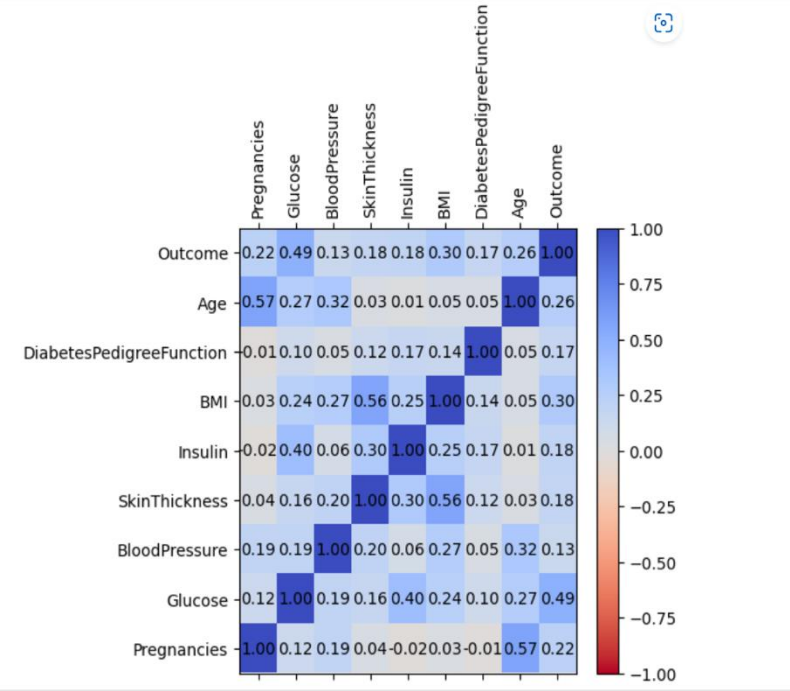


Fig 20: Correlation matrix

Through the correlation heatmap, we get a holistic understanding of both the relationships between features and the outcome and the inherent characteristics of individual features. This

ranking indicates the degree of association between each feature and the likelihood of an individual being diabetic

Additionally, horizontal bar plot was utilized to provide a visual representation of the correlation of each feature with outcome. We discerned that glucose exhibits the strongest correlation with the outcome variable, followed by BMI, age, pregnancies, insulin, skin thickness, diabetes pedigree function, and blood pressure.

```
#Barplot showing correlation of variables with outcome
import matplotlib.pyplot as plt
import pandas as pd
fig = plt.figure(figsize=(10, 4))
# Calculate correlation
outcome_corr = pd.DataFrame(df.corr()['Outcome'].sort_values(ascending=True))
# Set color for Outcome variable to red and others to blue
colors = ['orange' if var != 'Outcome' else 'red' for var in outcome_corr.index]
# Create bar plot
plt.barh(outcome_corr.index, outcome_corr['Outcome'], color=colors)
plt.title('Correlation with Outcome')
plt.show()
```

Fig 21: code snippet showing correlation of variables with outcome

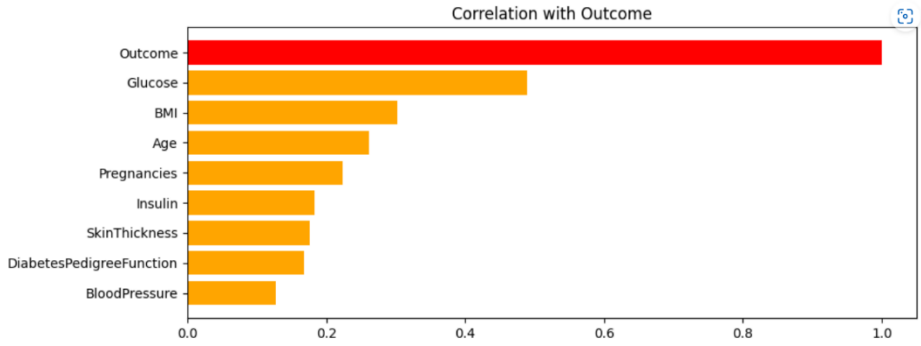


Fig 22: Bar graph depicting correlation of variables with outcome

These visualizations enable us to identify key features that may play a significant role in predicting the outcome variable. The insights derived from the correlation heatmap, and bar plots serve as a foundation for feature selection, guiding subsequent modeling and analysis decisions in our project.

Normality Testing:

We have conducted a normality testing process as part of our project to assess the distribution characteristics of each feature in the Healthcare Diabetes dataset. The objective was to evaluate whether the data follows a normal distribution or not. For this analysis, we utilized the Shapiro-Wilk test, a statistical test designed for assessing normality. The null hypothesis for each test was that the data is normally distributed, while the alternative hypothesis stated that the data does not follow a normal distribution. By applying the Shapiro-Wilk test to each feature, we aimed to gather statistical evidence to either accept or reject the assumption of normality. To complement the statistical tests, we employed histograms and QQ plots for visualizing the distribution of data for each feature. These graphical representations provided additional insights into the shape and

characteristics of the data, allowing us to visually confirm or challenge the results obtained from the Shapiro-Wilk tests.

```
# checking the normality using Shapiro-Wilk test and plotting histogram
import pandas as pd
from scipy.stats import shapiro
import matplotlib.pyplot as plt
import numpy as np
# Create a list of column indices to exclude (e.g., first and last columns)
exclude_columns = [-1]
# Create a copy of the DataFrame with excluded columns for visualization
df_copy = df.drop(df.columns[exclude_columns], axis=1).copy()
# Set up subplots
fig, axes = plt.subplots(nrows=2, ncols=4, figsize=(16, 8))
# Flatten the 2D array of subplots into a 1D array
axes = axes.flatten()
# Loop through columns, perform Shapiro-Wilk test, and plot line plot or histogram for normally distributed data
for i, column in enumerate(df_copy.columns):
    # Shapiro-Wilk test
    stat, p_value = shapiro(df_copy[column])
    # Check if data is normally distributed based on the significance level (e.g., 0.05)
    alpha = 0.05
    is_normally_distributed = p_value > alpha
    # Line plot for normally distributed data
    if is_normally_distributed:
        sorted_data = np.sort(df_copy[column])
        axes[i].plot(sorted_data, np.linspace(0, 1, len(sorted_data), endpoint=False), color='blue')
        axes[i].set_title(f'Line Plot - {column}\nData is normally distributed (Shapiro p-value: {p_value})')
    else:
        # If data is not normally distributed, you might want to handle it differently (e.g., use a histogram)
        axes[i].set_title(f'Histogram - {column}\nData is not normally distributed \n(Shapiro p-value: {p_value})')
        axes[i].hist(df_copy[column], bins='auto', color='purple', edgecolor='white', alpha=0.7)
    # Rotate x-axis labels
    axes[i].tick_params(axis='x', rotation=45)
# Adjust layout
plt.tight_layout()
# Show plots
plt.show()
```

Fig 23: code snippet showing normality test (Shapiro-wilk test)

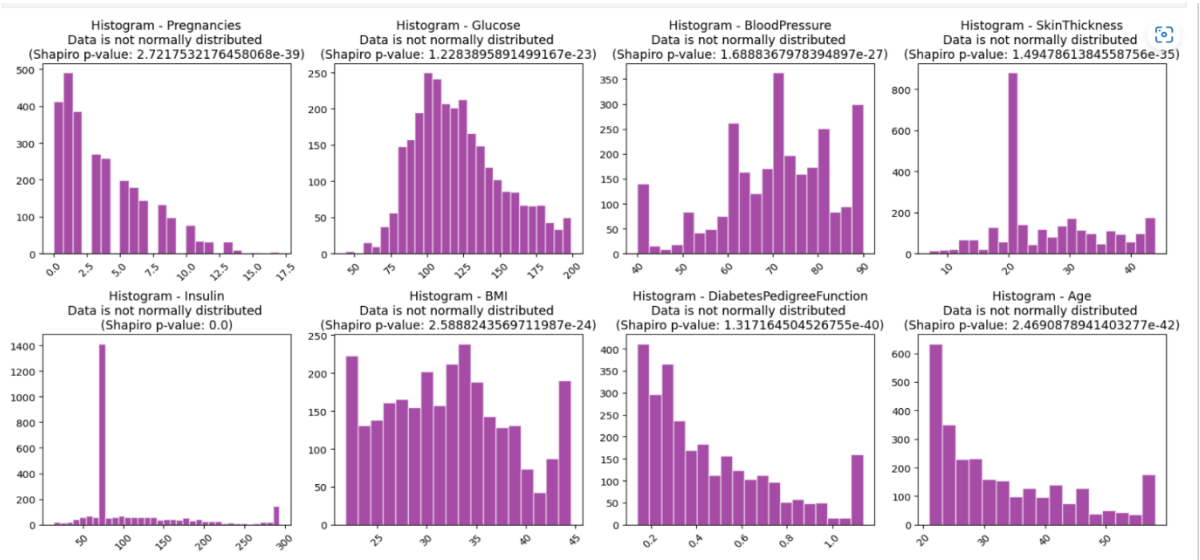


Fig 24: Histograms depicting the data distribution

Upon statistical testing with the Shapiro-Wilk test, we obtained p-values consistently below 0.05 for each feature. Consequently, we have rejected the null hypothesis for each test, indicating that the data in each column is not normally distributed. The histograms visually displayed the distribution characteristics of the data, and the rejection of the null hypothesis aligned with the observed shapes and patterns in the histograms.

Furthermore, QQ plots were utilized to provide a graphical representation of the data distribution and show that the p-values were consistently below 0.05, leading to the rejection of the null hypothesis. This reaffirms that the data in each column does not adhere to a normal distribution.

```python
# checking the normality using Shapiro-Wilk test and plotting QQ Plots
import pandas as pd
from scipy.stats import shapiro, probplot
import matplotlib.pyplot as plt
 # Create a list of column indices to exclude (e.g., first and last columns)
exclude_columns = [-1]
# Create a copy of the DataFrame with excluded columns for visualization
df_copy = df.drop(df.columns[exclude_columns], axis=1).copy()
# Set up subplots with constrained layout
fig, axes = plt.subplots(nrows=2, ncols=4, figsize=(20, 8), constrained_layout=True)
# Flatten the 2D array of subplots into a 1D array
axes = axes.flatten()
# Loop through columns
for i, column in enumerate(df.columns[:8]):  # Adjust the number of columns as needed
    # Shapiro-Wilk test
    stat, p_value = shapiro(df[column])
    # Check if data is normally distributed based on the significance level (e.g., 0.05)
    alpha = 0.05
    is_normally_distributed = p_value > alpha
    #QQ plot
    probplot(df[column], plot=axes[i], rvalue=True)
    axes[i].set_title(f'QQ Plot - {column}\nData is normally distributed (Shapiro p-value: {p_value})' if is_normally_distributed else f'QQ Plot -
    {column}\nData is not normally distributed \n(Shapiro p-value: {p_value:})')
# Adjust layout
# plt.tight_layout()
# Show plots
plt.show()
```

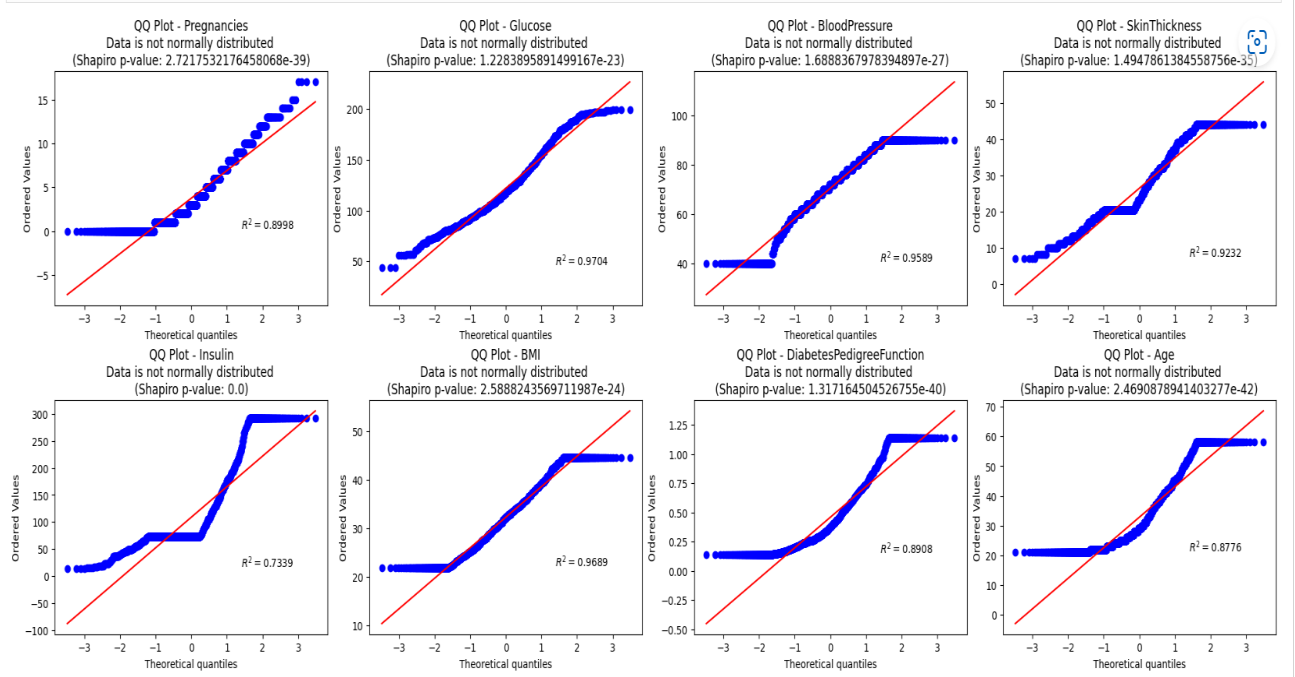


Fig 25: QQ plots of variables

Additionally, we conducted normality testing using the normal test, and again, the obtained p-values were below 0.05, confirming the rejection of the null hypothesis and reinforcing the conclusion that the data for each column is not normally distributed.

```python
import pandas as pd
from scipy.stats import normaltest
# Set up a list to store results
normality_results = []
# Loop through columns and perform normality test
for column in df.columns:
    # Normality test
    statistic, p_value = normaltest(df[column])
    # Store results
    normality_results.append({
        'Column': column,
        'Statistic': statistic,
        'P-value': p_value,
        'Is Normally Distributed': p_value > 0.05  # Adjust the significance level as needed
    })
# Convert results to DataFrame for easier inspection
results_df = pd.DataFrame(normality_results)
# Print or inspect the results DataFrame
print(results_df)
```

Fig 26: code snippet showing normal test

```
                     Column     Statistic        P-value  \
0               Pregnancies    319.215030   4.823220e-70
1                   Glucose    127.615069   1.944196e-28
2             BloodPressure    118.926419   1.497808e-26
3             SkinThickness    256.941607   1.606346e-56
4                   Insulin    731.647498  1.332799e-159
5                       BMI    255.283583   3.680224e-56
6  DiabetesPedigreeFunction    311.017942   2.906109e-68
7                       Age    290.067361   1.029610e-63
8                   Outcome  15242.908285   0.000000e+00

   Is Normally Distributed
0                    False
1                    False
2                    False
3                    False
4                    False
5                    False
6                    False
7                    False
8                    False
```

Fig 27: Normal test results

This thorough normality assessment equips us with crucial insights into the distributional nature of our dataset, enabling us to make informed decisions about the choice of statistical methods and model assumptions in subsequent analyses within our project.

Statistical Analysis

As it can be inferred from the normality test that the data is not normally distributed, we selected the non-parametric test for analysis. We have conducted a statistical analysis in our project, employing logistic regression to assess the impact of predictor variables on the outcome and chi-square test to evaluate the relationship between two categorical variables.

Logistic Regression:

1. Logistic regression is a statistical analysis used to model the relationship between a binary outcome variable and one or more independent variables.
2. Odds ratios in logistic regression quantify the change in odds of the outcome event occurring when the independent variable increases by one unit, holding all other variables constant.

```python
import pandas as pd
import statsmodels.api as sm
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming you have your training and testing data loaded into X_train_resampled, y_train_resampled, X_test, and y_test

# Add a constant term to the features for intercept in statsmodels
X_train_resampled = sm.add_constant(X_train_resampled)
X_test = sm.add_constant(X_test)

# Create logistic regression model
logit_model = sm.Logit(y_train_resampled, X_train_resampled)

# Fit the model
result = logit_model.fit()

print(result.summary())

odds_ratios = pd.DataFrame({'Odds Ratio': result.params.apply(lambda x: round(np.exp(x), 3)),
                            '95% CI (lower)': result.conf_int()[0].apply(lambda x: round(np.exp(x), 3)),
                            '95% CI (upper)': result.conf_int()[1].apply(lambda x: round(np.exp(x), 3))})

print("\nOdds Ratios:")
print(odds_ratios)
```

Fig 28: code snippet depicting logistic regression and odds ratio

```python
import matplotlib.pyplot as plt
# Assuming 'result' is the fitted logistic regression result
coefficients = result.params.drop('const').sort_values(ascending=False)
# Plotting in descending order
fig, ax = plt.subplots(figsize = (10,6))
bars = ax.barh(coefficients.index, coefficients, color=np.where(coefficients >= 0, 'green', 'red'))
# Add values as text annotations
for bar in bars:
    xval = bar.get_width()
    plt.text(xval +0.05, bar.get_y() + bar.get_height()/2, round(xval, 3), ha='left', va='center', color='black')
# Set even x-axis ticks from -0.2 to 1.2
ax.set_xticks([-0.2, 0, 0.2, 0.4, 0.6, 0.8, 1, 1.2, 1.4])
plt.title('Logistic Regression Coefficients (Descending Order)')
plt.xlabel('Coefficient Value')
plt.show()
```

Fig 29: code snippet depicting visual representation of logistic regression coefficients

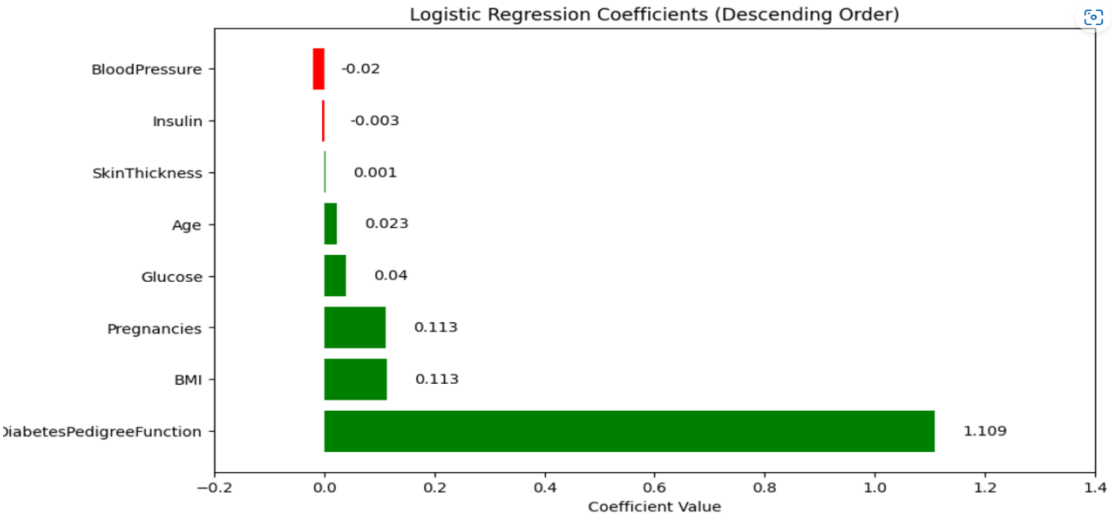


Fig 30: Logistic regression coefficeints horizontal bar plot

Positive coefficients indicate an increase in the log-odds of the outcome, while negative coefficients suggest a decrease.

Additionally, odds ratios greater than 1 signify that an increase in the predictor variable is associated with an increase in the odds of the outcome.

```
import seaborn as sns
# Assuming odds_ratios is a DataFrame containing odds ratios and CIs
plt.errorbar(odds_ratios['Odds Ratio'], range(len(odds_ratios)),
             xerr=(odds_ratios['Odds Ratio'] - odds_ratios['95% CI (lower)'], odds_ratios['95% CI (upper)'] - odds_ratios['Odds Ratio']),
             fmt='o', linestyle='None', color = 'magenta')
plt.yticks(range(len(odds_ratios)), odds_ratios.index)
plt.xscale('log')  # Use log scale for better visualization
plt.title('Odds Ratios with 95% Confidence Intervals')
plt.show()
```

Fig 31: code showing visual representation of odds ratio

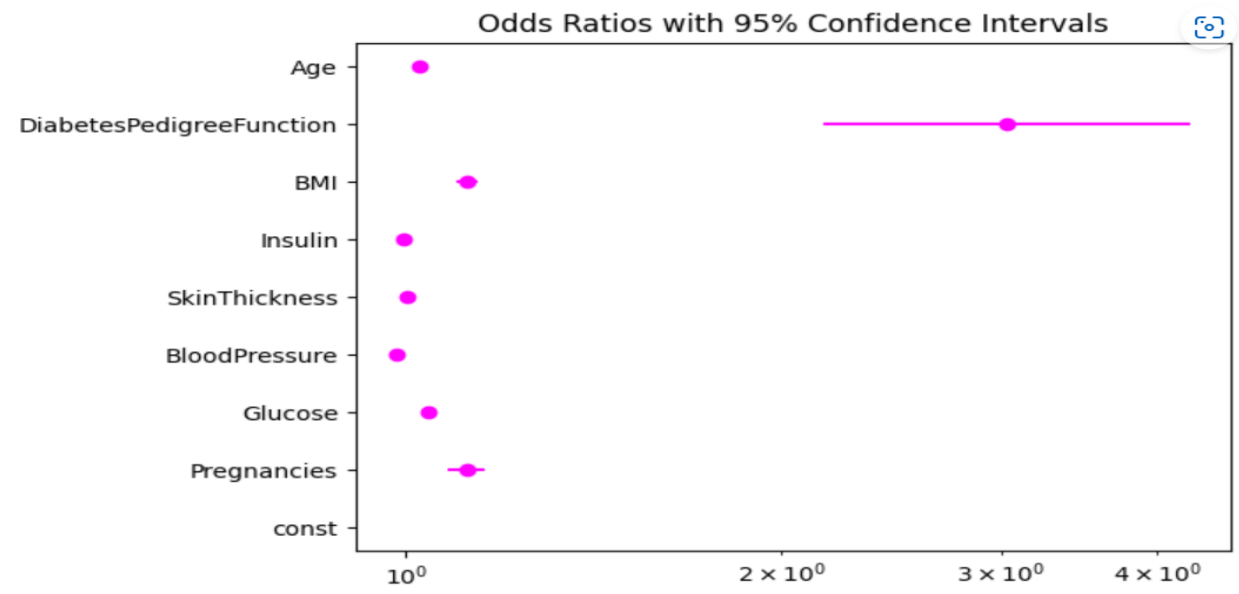


Fig 32: Errorbar showing Odds ratio with confidence ratio

Chi-square test:

Furthermore, we utilized the Chi-square test to examine the relationship between two categorical variables, specifically pregnancy and age group, BMI, and glucose. We converted the numerical variables such as pregnancy, age, BMI and glucose into categories and then compared it with our outcome variable. The analysis was based on p-values, and the rejection of the null hypothesis suggests a significant association between these variables and the outcome.

```python
import pandas as pd
from scipy.stats import chi2_contingency
import seaborn as sns
import matplotlib.pyplot as plt
df_Copy = df.copy()
# Create age groups
age_bins = [20, 30, 40, 50, 60]
age_labels = ['20-29', '30-39', '40-49', '50-59']
df_Copy['Age_Group'] = pd.cut(df_Copy['Age'], bins=age_bins, labels=age_labels, right=False)
# Perform Chi-Square Test of Independence
contingency_table = pd.crosstab(df_Copy['Age_Group'], df_Copy['Outcome'])
chi2, p_value, _, _ = chi2_contingency(contingency_table)
# Print results
print(f"Chi-Square Value: {chi2}")
print(f"P-value: {p_value}")
if p_value < 0.05:
    print("Reject the null hypothesis. There is evidence of an association between age groups and the diabetes outcome.")
else:
    print("Fail to reject the null hypothesis. There is insufficient evidence to claim an association.")
# Calculate percentages for each category within each group
group_percentages = (contingency_table.T / contingency_table.sum(axis=1)).T * 100
# Visualize the relationship between Age Group and Outcome with percentages
plt.figure(figsize=(12, 6))
ax = sns.barplot(x='Age_Group', y='Percentage', hue='Outcome', data=group_percentages.reset_index().melt(id_vars='Age_Group', var_name='Outcome', value_n
plt.title('Distribution of Outcome by Age Group')
plt.xlabel('Age Group')
plt.ylabel('Percentage')
plt.legend(title='Outcome', loc='upper right')
# Add percentage labels to each bar
for p in ax.patches:
    percentage = '{:.1f}%'.format(p.get_height())
    x = p.get_x() + p.get_width() / 2 - 0.15
    y = p.get_y() + p.get_height() + 0.1
    ax.annotate(percentage, (x, y), fontsize=10)
# Show the plot
plt.show()
```

Fig 33: code snippet showing Chi square test

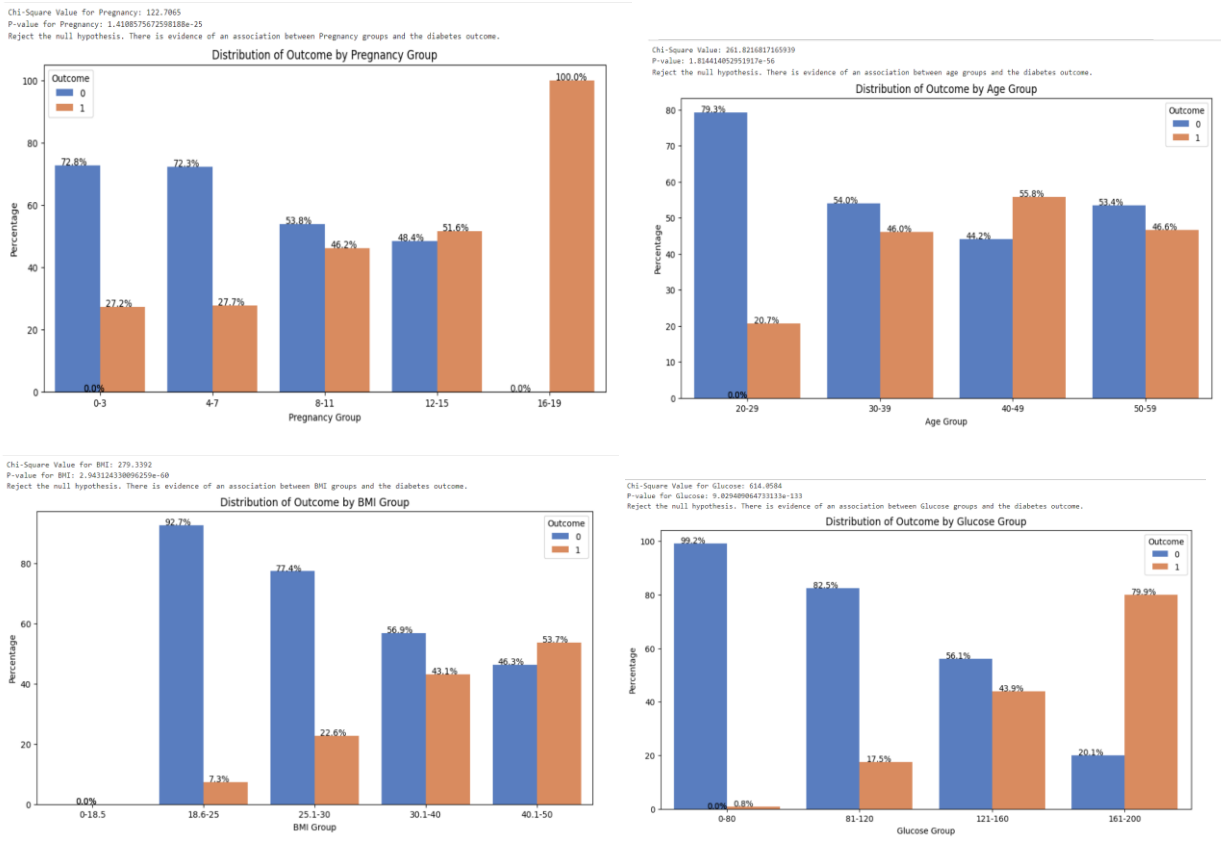


Fig 34: Distribution of outcome by Pregnancy Group, Age group, BMI Group, Glucose Group

In summary, we have employed logistic regression to understand the impact of predictor variables, considering both coefficient signs and odds ratios. Additionally, the Chi-square test has been utilized to assess associations between categorical variables and the outcome, with the rejection of null hypotheses indicating significant relationships.

Train-Test Split:

We have integrated the dataset into the `sklearn.model_selection` module, specifically utilizing the `train_test_split` function. In this process, we partitioned the dataset into training and testing sets. We have performed the split for both the predictor variables (X) and the outcome variable (Y), allocating 80% of the data for training purposes, resulting in `X_train` and `y_train`. The remaining 20% was designated for testing, leading to the creation of `X_test` and `y_test`.

```
import pandas as pd
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
# 'Outcome' is our binary target variable
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)
```

Fig 35: code snippet showing train_test split

This strategic division allows us to train our model on a substantial portion of the data, enhancing its learning capabilities, and then evaluate its performance on a separate and previously unseen dataset. Such practices contribute to robust model assessment and validation within our project.

SMOTE (Synthetic Minority Over-sampling Technique):

We used SMOTE for oversampling the minority class to address the class imbalance in our outcome column. After the train-test split, we used SMOTE only on the training set. The imblearn package is to be installed through 'pip install imblearn' to use SMOTE.

```
import pandas as pd
# Display value counts for the 'Outcome' column
outcome_counts = df['Outcome'].value_counts()
print(outcome_counts)
```

```
Outcome
0    1816
1     952
Name: count, dtype: int64
```

Fig 36: code snippet to check the data imbalance in the outcome column

```
pip install -U imbalanced-learn
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: imbalanced-learn in ./.local/lib/python3.10/site-packages (0.11.0)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn) (1.24.1)
Requirement already satisfied: scipy>=1.5.0 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn) (1.9.3)
Requirement already satisfied: scikit-learn>=1.0.2 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn) (1.3.0)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn) (1.2.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn) (3.1.0)

[notice] A new release of pip is available: 23.2.1 -> 23.3.1
[notice] To update, run: python3 -m pip install --upgrade pip
Note: you may need to restart the kernel to use updated packages.
```

Fig 37: code snippet depicting U-imbalanced installation

```
import pandas as pd
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
# 'Outcome' is our binary target variable
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)
# Print class distribution before SMOTE on the training set
original_train_class_distribution = pd.Series(y_train).value_counts()
print("Class Distribution of Training Set Before SMOTE:")
print(original_train_class_distribution)
# Apply SMOTE only to the training set
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)
# Print class distribution after SMOTE on the training set
resampled_train_class_distribution = pd.Series(y_train_resampled).value_counts()
print("\nClass Distribution of Training Set After SMOTE:")
print(resampled_train_class_distribution)
```

```
Class Distribution of Training Set Before SMOTE:
Outcome
0    1449
1     765
Name: count, dtype: int64

Class Distribution of Training Set After SMOTE:
Outcome
1    1449
0    1449
Name: count, dtype: int64
```

Fig 38: code snippet showing class distribution before and after SMOTE

Machine Learning Models

We used machine learning models like logistic regression, Support Vector classification, K-nearest neighbor, decision tree and random forest to understand the relationship between our independent variables (history of pregnancy, skin thickness, glucose levels, blood pressure, insulin, BMI, diabetes pedigree function, age) and dependent variable (risk of acquiring diabetes). For all the models, after the train-test split and SMOTE, we built the model, performed 5-fold cross validation on the resampled training data, fitted the model on the resampled training data and made predictions on the testing data. Cross-validation was done to obtain a more robust estimate about

the model's performance and to reduce variability and assess the model's generalizability performance. For cross-validation score, from the model_selection module within the scikit-learn library, we imported cross_val_score and from the metrics module within the scikit-learn library, we imported accuracy_score, classification report and confusion matrix. We printed the cross-validation scores, mean cross-validation scores, accuracy, classification report, confusion matrix and roc curve.

Logistic Regression:

Logistic Regression is interpretable and provides probabilities, making it suitable for understanding the impact of features on the likelihood of an event (e.g., diabetes). The logistic regression model provided the relationship between variables and the likelihood of diabetes. Evaluation metrics like accuracy and ROC curves were used to assess its performance.

To build the logistic regression model, we imported the LogisticRegression class from the linear_model module within the scikit-learn library.

```
import pandas as pd
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
## Create a logistic regression model
model = LogisticRegression()
# Perform k-fold cross-validation (e.g., k=5)
cv_scores = cross_val_score(model, X_train_resampled, y_train_resampled, cv=5)
# Train the model on the resampled training data
model.fit(X_train_resampled, y_train_resampled)
# Make predictions on the testing data
y_pred_lr = model.predict(X_test)
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred_lr)
conf_matrix = confusion_matrix(y_test, y_pred_lr)
class_report = classification_report(y_test, y_pred_lr)
# Print the results
print(f"Cross-Validation Scores: {cv_scores}")
print(f"Mean CV Score: {cv_scores.mean()}")
print(f"Accuracy: {accuracy}")
print(f"Confusion Matrix:\n{conf_matrix}")
print(f"Classification Report:\n{class_report}")
# Plot the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=['Non-Diabetic', 'Diabetic'], yticklabels=['Non-Diabetic', 'Diabetic'])
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```

Figure 39: code snippet for Logistic Regression model

The classification report for logistic regression shows an accuracy of 76% with an f-1 score of 0.81 for non-diabetic and 0.66 for diabetic. The visual representation of confusion matrix shows that the model correctly predicted 290 true negative, 129 true positive, 58 false negative and 77 false positive instances. The mean cross validation score for logistic regression is also 76%.

```
Cross-Validation Scores: [0.77413793 0.79310345 0.76206897 0.75647668 0.75302245]
Mean CV Score: 0.7677618962539456
Accuracy: 0.7563176895306859
Confusion Matrix:
[[290  77]
 [ 58 129]]
Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.79      0.81       367
           1       0.63      0.69      0.66       187

    accuracy                           0.76       554
   macro avg       0.73      0.74      0.73       554
weighted avg       0.76      0.76      0.76       554
```

Figure 40: Output showing classification report for Logistic Regression model

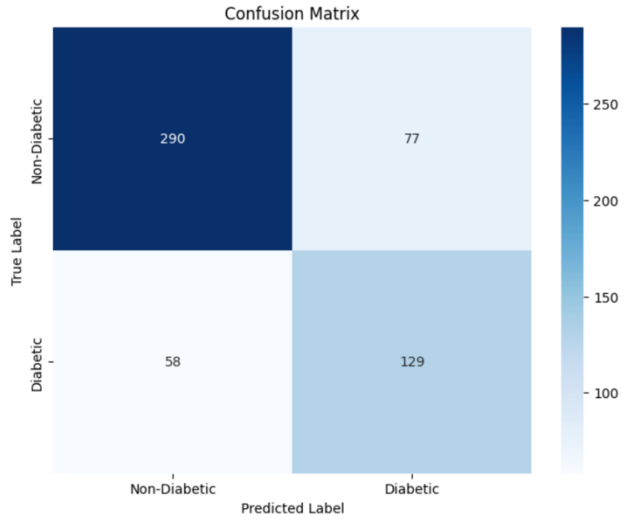


Fig 41: Confusion matrix for logistic regression

For roc curve, from the metrics module within the scikit-learn library we imported roc_curve and auc, and for the visual representation we used the matplotlib library to generate and display the Receiver Operating Characteristic (ROC) curve for a Logistic Regression model. Initially, it calculates probability estimates for the positive class using the predict_proba method on the Logistic Regression (model). Subsequently, it computes the ROC curve and the Area Under the Curve (AUC) using true labels (y_test) and predicted probabilities. The resulting ROC curve is plotted in dark orange, with the AUC region shaded in light blue and a dashed line represents the ROC curve for random guessing. False Positive Rate (FPR) is plotted on the x-axis, and the True Positive Rate (TPR) is plotted on the y-axis. The ROC for support vector classification shows an AUC of 0.84.

```python
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# Make predictions on the test set
y_pred_proba_lr = model.predict_proba(X_test)[:, 1]

# Calculate false positive rate and true positive rate
fpr_lr, tpr_lr, thresholds_lr = roc_curve(y_test, y_pred_proba_lr)

# Calculate AUC
roc_auc_lr = auc(fpr_lr, tpr_lr)

# Plot ROC Curve with shading
plt.figure(figsize=(8, 6))
plt.plot(fpr_lr, tpr_lr, label='ROC curve (AUC = {:.2f})'.format(roc_auc_lr), color='darkorange', lw=2)
plt.fill_between(fpr_lr, 0, tpr_lr, color='skyblue', alpha=0.3, label='AUC')  # Shading the area under the ROC curve
plt.plot([0, 1], [0, 1], 'k--', label='Random Guess')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve (Logistic Regression)')
plt.legend(loc="lower right")
plt.show()
```

Fig 42: code for plotting ROC curve for logistic regression

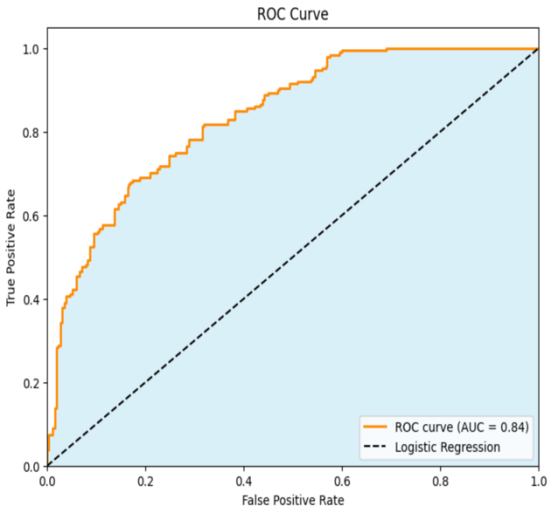


Fig 43: ROC curve for Logistic regression

Support Vector Classification:

Support Vector Classification (SVC) model with a linear kernel and probability estimates was trained to classify instances as diabetic or non-diabetic. Cross-validation and evaluation metrics were employed to assess its performance.

To build the support vector classification model, we imported the SVC class from the svm module within the scikit-learn library.

```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Create an SVC model
svc_model = SVC(probability = True)

# Perform k-fold cross-validation (e.g., k=5)
cv_scores = cross_val_score(svc_model, X_train_resampled, y_train_resampled, cv=5)

# Train the model on the training data
svc_model.fit(X_train_resampled, y_train_resampled)

# Make predictions on the testing data
y_pred = svc_model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

# Print the results
print(f"Cross-Validation Scores: {cv_scores}")
print(f"Mean CV Score: {cv_scores.mean()}")
print(f"Accuracy: {accuracy}")
print(f"Confusion Matrix:\n{conf_matrix}")
print(f"Classification Report:\n{class_report}")

# Plot the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Greens", xticklabels=['Non-Diabetic', 'Diabetic'], yticklabels=['Non-Diabetic', 'Diabetic'])
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```

Figure 44: code snippet for Support Vector Classification Technique

The classification report shows an accuracy of 75% with an f-1 score of 0.80 for non-diabetic and 0.67 for diabetic. The visual representation of confusion matrix shows that the model correctly predicted 280 true negative, 138 true positive, 49 false negative and 87 false positive instances. The mean cross validation score for SVC is 74%.

```
Cross-Validation Scores: [0.76034483 0.75689655 0.72413793 0.74438687 0.73402418]
Mean CV Score: 0.7439580727770829
Accuracy: 0.7545126353790613
Confusion Matrix:
[[280  87]
 [ 49 138]]
Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.76      0.80       367
           1       0.61      0.74      0.67       187

    accuracy                           0.75       554
   macro avg       0.73      0.75      0.74       554
weighted avg       0.77      0.75      0.76       554
```

Figure 45: Output showing classification report for SVC model

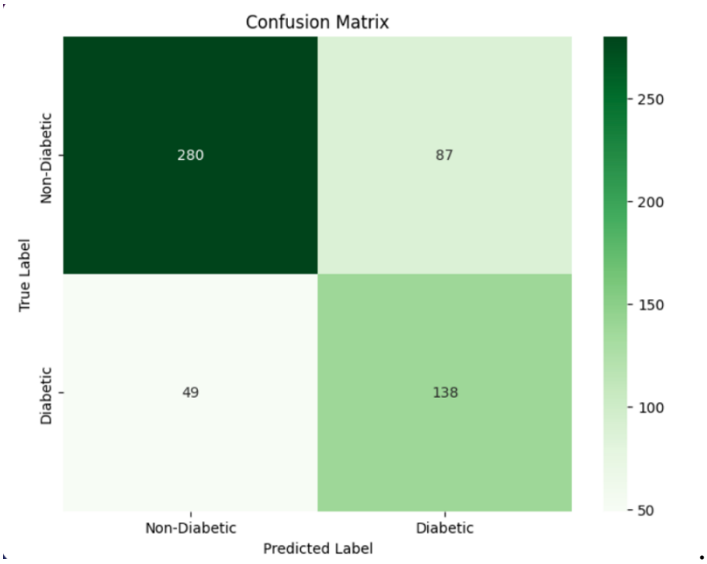


Fig 46: Confusion matrix for SVC

For roc curve, from the metrics module within the scikit-learn library we imported roc_curve and auc, and for the visual representation we used the matplotlib library to generate and display the Receiver Operating Characteristic (ROC) curve for a Support Vector Classification model. Initially, it calculates probability estimates for the positive class using the predict_proba method on the SVC (svc_model). Subsequently, it computes the ROC curve and the Area Under the Curve (AUC) using true labels (y_test) and predicted probabilities. The resulting ROC curve is plotted in dark orange, with the AUC region shaded in light blue and a dashed line represents the ROC curve for random guessing. False Positive Rate (FPR) is plotted on the x-axis, and the True Positive Rate (TPR) is plotted on the y-axis. The ROC for support vector classification shows an AUC of 0.84.

```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc

# Calculate probability estimates for the positive class
y_pred_proba_svc = svc_model.predict_proba(X_test)[:, 1]

# Calculate ROC curve
fpr_svc, tpr_svc, thresholds_svc = roc_curve(y_test, y_pred_proba_svc)
roc_auc_svc = auc(fpr_svc, tpr_svc)

# Plot ROC curve with shading
plt.figure(figsize=(8, 6))
plt.plot(fpr_svc, tpr_svc, color='darkorange', lw=2, label='ROC curve (AUC = {:.2f})'.format(roc_auc_svc))
plt.fill_between(fpr_svc, 0, tpr_svc, color='skyblue', alpha=0.3, label='AUC')  # Shading the area under the ROC curve
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--', label='Random Guess')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve (Support Vector Classifier)')
plt.legend(loc='lower right')
plt.show()
```

Fig 47: code for plotting ROC curve for SVC

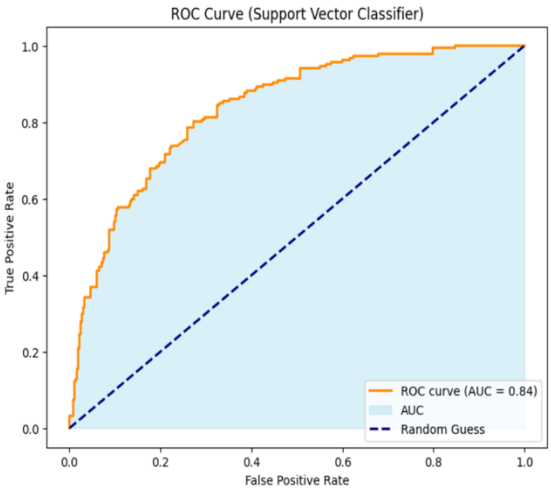


Fig 48: ROC curve for SVC

K-Nearest Neighbors (KNN):

The KNN model was used to classify instances as diabetic or non-diabetic. Cross-validation and evaluation metrics were used to assess its performance. We have created models using two different values of 'k'- 5 and 3.

To build the knn model, we imported the KNeighborsClassifier class from the neighbors module within the scikit-learn library.

```
import pandas as pd
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
# Create a k-Nearest Neighbors model (Let's use k=5 as an example)
knn_model = KNeighborsClassifier(n_neighbors=5)
# Perform k-fold cross-validation (e.g., k=5)
cv_scores = cross_val_score(knn_model, X_train_resampled, y_train_resampled, cv=5)
# Train the model on the training data
knn_model.fit(X_train_resampled, y_train_resampled)
# Make predictions on the testing data
y_pred = knn_model.predict(X_test)
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)
# Print the results
print(f"Cross-Validation Scores: {cv_scores}")
print(f"Mean CV Score: {cv_scores.mean()}")
print(f"Accuracy: {accuracy}")
print(f"Confusion Matrix:\n{conf_matrix}")
print(f"Classification Report:\n{class_report}")
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=['Non-Diabetic', 'Diabetic'], yticklabels=['Non-Diabetic', 'Diabetic'])
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```

Figure 49: code snippet for KNN model where k = 5

```python
import pandas as pd
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

knn_model = KNeighborsClassifier(n_neighbors=3)

# Perform k-fold cross-validation (e.g., k=5)
cv_scores = cross_val_score(knn_model, X_train_resampled, y_train_resampled, cv=5)

# Train the model on the training data
knn_model.fit(X_train_resampled, y_train_resampled)

# Make predictions on the testing data
y_pred = knn_model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

# Print the results
print(f"Cross-Validation Scores: {cv_scores}")
print(f"Mean CV Score: {cv_scores.mean()}")
print(f"Accuracy: {accuracy}")
print(f"Confusion Matrix:\n{conf_matrix}")
print(f"Classification Report:\n{class_report}")

plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="BuGn", xticklabels=['Non-Diabetic', 'Diabetic'], yticklabels=['Non-Diabetic', 'Diabetic'])
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```

Figure 50: code snippet for KNN model where k = 3

The classification report for knn model with k = 5 shows an accuracy of 88% with an f-1 score of 0.91 for non-diabetic and 0.85 for diabetic. The visual representation of confusion matrix shows that the model correctly predicted 315 true negative, 176 true positive, 11 false negative and 52 false positive instances. The mean cross validation score is 89%. However, the classification report for knn model with k = 3 shows an accuracy of 95% with an f-1 score of 0.97 for non-diabetic and 0.93 for diabetic. The visual representation of confusion matrix shows that the model correctly predicted 351 true negative, 178 true positive, 9 false negative and 16 false positive instances. The mean cross validation score is 94%.

As the model with k=3 gave us better accuracy, we selected that model and plotted the roc curve for the same.

```
Cross-Validation Scores: [0.88103448 0.9        0.88793103 0.89982729 0.90328152]
Mean CV Score: 0.8944148651063071
Accuracy: 0.8862815884476535
Confusion Matrix:
[[315  52]
 [ 11 176]]
Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.86      0.91       367
           1       0.77      0.94      0.85       187

    accuracy                           0.89       554
   macro avg       0.87      0.90      0.88       554
weighted avg       0.90      0.89      0.89       554
```

Fig 51: Output showing Classification report for KNN where k=5

```
Cross-Validation Scores: [0.93793103 0.94482759 0.94137931 0.94645941 0.93782383]
Mean CV Score: 0.9416842356024061
Accuracy: 0.9548736462093863
Confusion Matrix:
[[351  16]
 [  9 178]]
Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.96      0.97       367
           1       0.92      0.95      0.93       187

    accuracy                           0.95       554
   macro avg       0.95      0.95      0.95       554
weighted avg       0.96      0.95      0.96       554
```

Figure 52: Output showing Classification report for KNN where k=3

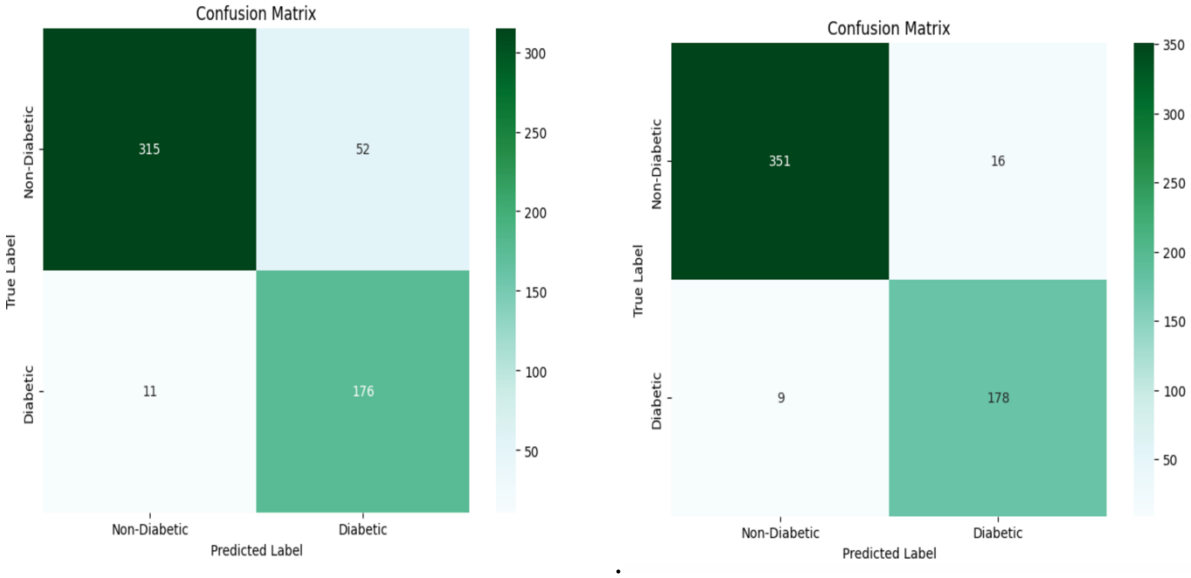


Fig 53: Confusion matrix for k=5 and k=3 respectively

For roc curve, from the metrics module within the scikit-learn library we imported roc_curve and auc, and for the visual representation we used the matplotlib library to generate and display the Receiver Operating Characteristic (ROC) curve for a K-Nearest Neighbor model. Initially, it calculates probability estimates for the positive class using the predict_proba method on the KNN (knn_model). Subsequently, it computes the ROC curve and the Area Under the Curve (AUC) using true labels (y_test) and predicted probabilities. The resulting ROC curve is plotted in dark orange, with the AUC region shaded in light blue and a dashed line represents the ROC curve for random guessing. False Positive Rate (FPR) is plotted on the x-axis, and the True Positive Rate (TPR) is plotted on the y-axis. The ROC for support vector classification shows an AUC of 0.97.

```python
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc

# Calculate probability estimates for the positive class
y_pred_proba_knn = knn_model.predict_proba(X_test)[:, 1]

# Calculate ROC curve and AUC
fpr_knn, tpr_knn, thresholds_knn = roc_curve(y_test, y_pred_proba_knn)
roc_auc_knn = auc(fpr_knn, tpr_knn)

# Plot ROC curve with shaded AUC
plt.figure(figsize=(8, 6))
plt.plot(fpr_knn, tpr_knn, color='darkorange', lw=2, label='ROC curve (AUC = {:.2f})'.format(roc_auc_knn))
plt.fill_between(fpr_knn, 0, tpr_knn, color='skyblue', alpha=0.3, label='AUC')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--', label='Random Guess')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve (K-nearest Neighbor)')
plt.legend(loc='lower right')
plt.show()
```

Fig 54: code for plotting ROC curve for KNN with k = 3

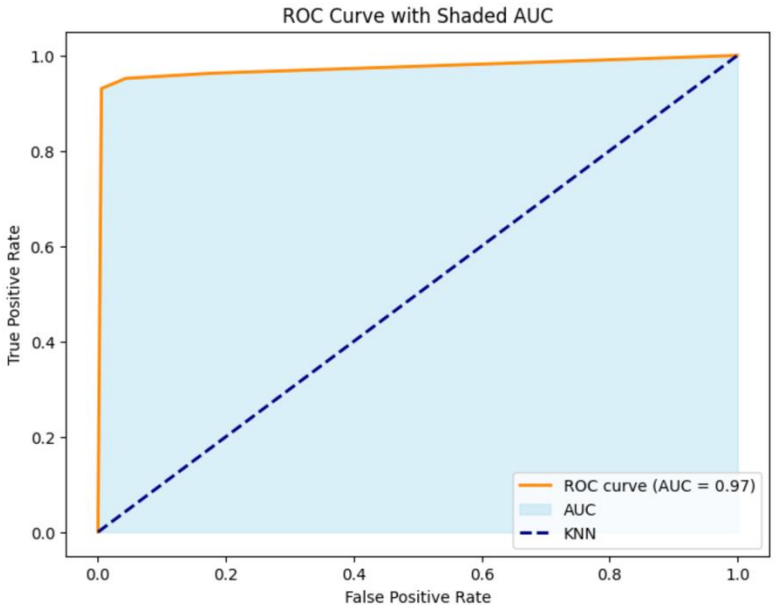


Fig 55: ROC curve for KNN with k=3

Decision Tree:

For our diabetes prediction task, the decision tree considers factors such as glucose levels, BMI, age, and other relevant features to classify individuals as diabetic or non-diabetic. Evaluation metrics like accuracy and ROC curves were used to assess its performance.

To build the decision tree model, we imported the DecisionTreeClassifier class from the tree module within the scikit-learn library.

```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from imblearn.over_sampling import SMOTE
# Create a decision tree model
tree_model = DecisionTreeClassifier(random_state=42)
# Perform k-fold cross-validation (e.g., k=5)
cv_scores = cross_val_score(tree_model, X_train_resampled, y_train_resampled, cv=5)
# Train the decision tree model on the resampled training data
tree_model.fit(X_train_resampled, y_train_resampled)
# Make predictions on the testing data
y_pred = tree_model.predict(X_test)
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)
# Print the results
print(f"Cross-Validation Scores: {cv_scores}")
print(f"Mean CV Score: {cv_scores.mean()}")
print(f"Accuracy: {accuracy}")
print(f"Confusion Matrix:\n{conf_matrix}")
print(f"Classification Report:\n{class_report}")
# Plot the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=['Non-Diabetic', 'Diabetic'], yticklabels=['Non-Diabetic', 'Diabetic'])
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```

Figure 56: code snippet for Decision Tree model

The classification report for decision tree shows an accuracy of 98% with an f-1 score of 0.98 for non-diabetic and 0.97 for diabetic. The visual representation of confusion matrix shows that the model correctly predicted 361 true negative, 180 true positive, 7 false negative and 6 false positive instances. The mean cross validation score for decision tree is 97%.

```
Cross-Validation Scores: [0.97241379 0.97931034 0.97586207 0.97236615 0.97236615]
Mean CV Score: 0.9744637007920911
Accuracy: 0.9765342960288809
Confusion Matrix:
[[361   6]
 [  7 180]]
Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.98      0.98       367
           1       0.97      0.96      0.97       187

    accuracy                           0.98       554
   macro avg       0.97      0.97      0.97       554
weighted avg       0.98      0.98      0.98       554
```

Figure 57: Output showing classification report for decision tree

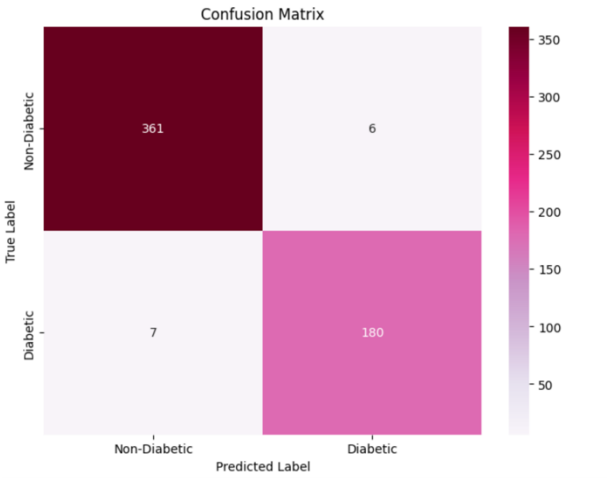


Fig 58: Confusion matrix for Decision tree model

For roc curve, from the metrics module within the scikit-learn library we imported roc_curve and auc, and for the visual representation we used the matplotlib library to generate and display the Receiver Operating Characteristic (ROC) curve for a Decision Tree model. Initially, it calculates probability estimates for the positive class using the predict_proba method on the Decision Tree (tree_model). Subsequently, it computes the ROC curve and the Area Under the Curve (AUC) using true labels (y_test) and predicted probabilities. The resulting ROC curve is plotted in dark orange, with the AUC region shaded in light blue and a dashed line represents the ROC curve for random guessing. False Positive Rate (FPR) is plotted on the x-axis, and the True Positive Rate (TPR) is plotted on the y-axis. The ROC for support vector classification shows an AUC of 0.97.

```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc

# Calculate probability estimates for the positive class
y_pred_proba_dt = tree_model.predict_proba(X_test)[:, 1]

# Calculate ROC curve and AUC
fpr_dt, tpr_dt, thresholds_dt = roc_curve(y_test, y_pred_proba_dt)
roc_auc_dt = auc(fpr_dt, tpr_dt)

# Plot ROC curve with shaded AUC
plt.figure(figsize=(8, 6))
plt.plot(fpr_dt, tpr_dt, color='darkorange', lw=2, label='ROC curve (AUC = {:.2f})'.format(roc_auc_dt))
plt.fill_between(fpr_dt, 0, tpr_dt, color='skyblue', alpha=0.3, label='AUC')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--', label='Random Guess')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve (Decision Tree)')
plt.legend(loc='lower right')
plt.show()
```

Fig 59: code for plotting ROC curve for Decision Tree model

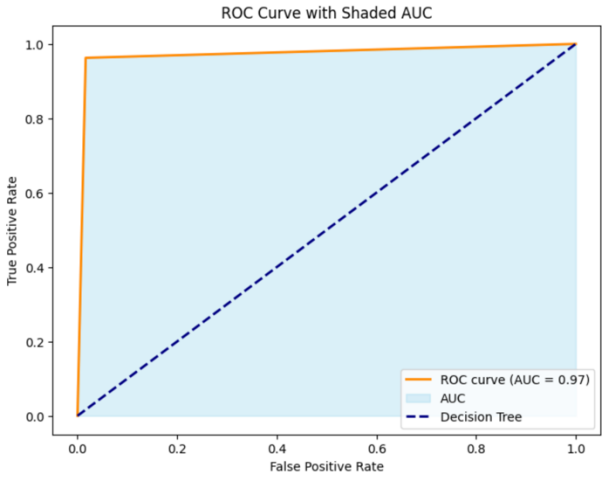


Fig 60: ROC curve for Decision Tree model

Random Forest:

Random Forest is an ensemble method that builds multiple decision trees and combines their predictions. The Random Forest model was trained, and its performance was evaluated. It combined the strengths of multiple decision trees to predict diabetes.

To build the random forest model, we imported the RandomForestClassifier class from the ensemble module within the scikit-learn library.

```
import pandas as pd
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
# Create a Random Forest model
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
# Perform k-fold cross-validation (e.g., k=5)
cv_scores = cross_val_score(rf_model, X_train_resampled, y_train_resampled, cv=5)
# Train the model on the training data
rf_model.fit(X_train_resampled, y_train_resampled)
# Make predictions on the testing data
y_pred = rf_model.predict(X_test)
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)
# Print the results
print(f"Cross-Validation Scores: {cv_scores}")
print(f"Mean CV Score: {cv_scores.mean()}")
print(f"Accuracy: {accuracy}")
print(f"Confusion Matrix:\n{conf_matrix}")
print(f"Classification Report:\n{class_report}")
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=['Non-Diabetic', 'Diabetic'], yticklabels=['Non-Diabetic', 'Diabetic'])
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```

Figure 61: code snippet for Decision Tree model

The classification report for random forest shows an accuracy of 98% with an f-1 score of 0.99 for non-diabetic and 0.97 for diabetic. The visual representation of confusion matrix shows that the model correctly predicted 363 true negative, 180 true positive, 7 false negative and 4 false positive instances. The mean cross validation score for decision tree is 99%.

```
Cross-Validation Scores: [0.98448276 0.99137931 0.9862069  0.98272884 0.98963731]
Mean CV Score: 0.9868870228098388
Accuracy: 0.98014440433213
Confusion Matrix:
[[363   4]
 [  7 180]]
Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.99      0.99       367
           1       0.98      0.96      0.97       187

    accuracy                           0.98       554
   macro avg       0.98      0.98      0.98       554
weighted avg       0.98      0.98      0.98       554
```

Figure 62: Output showing classification report for random forest

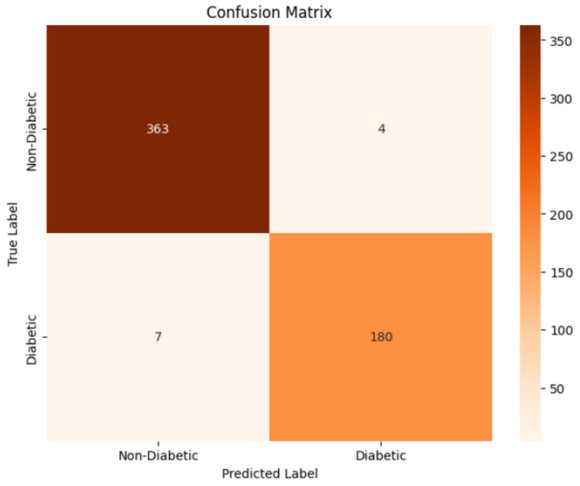


Fig 63: Confusion matrix for Random Forest

For roc curve, from the metrics module within the scikit-learn library we imported roc_curve and auc, and for the visual representation we used the matplotlib library to generate and display the Receiver Operating Characteristic (ROC) curve for a Random Forest model. Initially, it calculates probability estimates for the positive class using the predict_proba method on the random forest (rf_model). Subsequently, it computes the ROC curve and the Area Under the Curve (AUC) using true labels (y_test) and predicted probabilities. The resulting ROC curve is plotted in dark orange, with the AUC region shaded in light blue and a dashed line represents the ROC curve for random guessing. False Positive Rate (FPR) is plotted on the x-axis, and the True Positive Rate (TPR) is plotted on the y-axis. The ROC for support vector classification shows an AUC of 1.

```python
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc

# Calculate probability estimates for the positive class
y_pred_proba_rf = rf_model.predict_proba(X_test)[:, 1]

# Calculate ROC curve and AUC
fpr_rf, tpr_rf, thresholds_rf = roc_curve(y_test, y_pred_proba_rf)
roc_auc_rf = auc(fpr_rf, tpr_rf)

# Plot ROC curve with shaded AUC
plt.figure(figsize=(8, 6))
plt.plot(fpr_rf, tpr_rf, color='darkorange', lw=2, label='ROC curve (AUC = {:.2f})'.format(roc_auc_rf))
plt.fill_between(fpr_rf, 0, tpr_rf, color='skyblue', alpha=0.3, label='AUC')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--', label='Random Guess')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve(Random Forest)')
plt.legend(loc='lower right')
plt.show()
```

Fig 64: code for plotting ROC curve for Decision Tree model

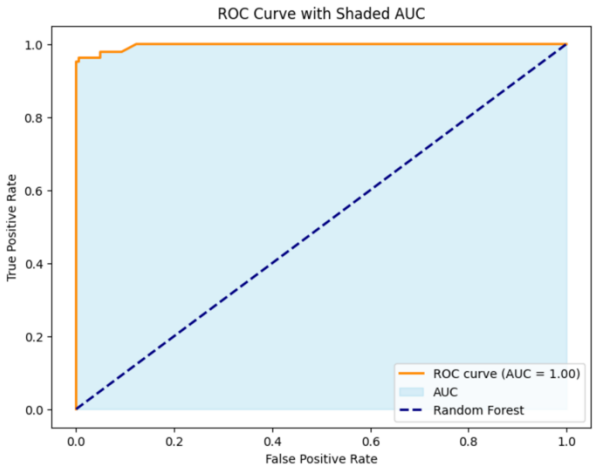


Fig 65: ROC curve for Decision Tree model

Comparison of ROC for all Models

This Python code uses the matplotlib library to create a Receiver Operating Characteristic (ROC) curve for multiple machine learning models. The models considered are Logistic Regression, Support Vector Classifier (SVC), K-Nearest Neighbors (KNN), Decision Tree, and Random Forest. For each model, the code calculates the false positive rate (fpr), true positive rate (tpr), and the area under the ROC curve (AUC) using the roc_curve and auc functions from the metrics module of scikit-learn library The resulting ROC curves are then plotted on a single graph using different colors for each model. The legend displays the AUC values for each model, and a dashed line represents the ROC curve for random guessing. The plot provides a visual representation of the trade-off between sensitivity and specificity for each model, aiding in the comparison of their performance in binary classification tasks.

```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc

# Logistic Regression
fpr_lr, tpr_lr, thresholds_lr = roc_curve(y_test, y_pred_proba_lr)
roc_auc_lr = auc(fpr_lr, tpr_lr)
# Support Vector Classifier
fpr_svc, tpr_svc, thresholds_svc = roc_curve(y_test, y_pred_proba_svc)
roc_auc_svc = auc(fpr_svc, tpr_svc)
# KNN
fpr_knn, tpr_knn, thresholds_knn = roc_curve(y_test, y_pred_proba_knn)
roc_auc_knn = auc(fpr_knn, tpr_knn)
# Decision Tree
fpr_dt, tpr_dt, thresholds_dt = roc_curve(y_test, y_pred_proba_dt)
roc_auc_dt = auc(fpr_dt, tpr_dt)
# Random Forest
fpr_rf, tpr_rf, thresholds_rf = roc_curve(y_test, y_pred_proba_rf)
roc_auc_rf = auc(fpr_rf, tpr_rf)

# Plot ROC curves for each model
plt.figure(figsize=(10, 8))
plt.plot(fpr_rf, tpr_rf, color='red', lw=2, label='Random Forest (AUC = {:.2f})'.format(roc_auc_rf))
plt.plot(fpr_dt, tpr_dt, color='green', lw=2, label='Decision Tree (AUC = {:.2f})'.format(roc_auc_dt))
plt.plot(fpr_knn, tpr_knn, color='blue', lw=2, label='KNN (AUC = {:.2f})'.format(roc_auc_knn))
plt.plot(fpr_lr, tpr_lr, color='darkorange', lw=2, label='Logistic Regression (AUC = {:.2f})'.format(roc_auc_lr))
plt.plot(fpr_svc, tpr_svc, color='purple', lw=2, label='Support Vector Classifier (AUC = {:.2f})'.format(roc_auc_svc))

# Plot the random guess line
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--', label='Random Guess')

# Set plot properties
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curves')
plt.legend(loc='lower right')
plt.show()
```

Figure 66: code snippet for plotting roc curve for all models

The visual representation compares the ROC for the models showing AUC of 1 for random forest, 0.97 for decision tree, 0.97 for k-nearest neighbor when k = 3, 0.84 for logistic regression and 0.84 for support vector classification.

An AUC of 1 for ROC of random forest suggests that it has better predictive ability compared to other models followed by decision tree, knn, logistic regression and svc.

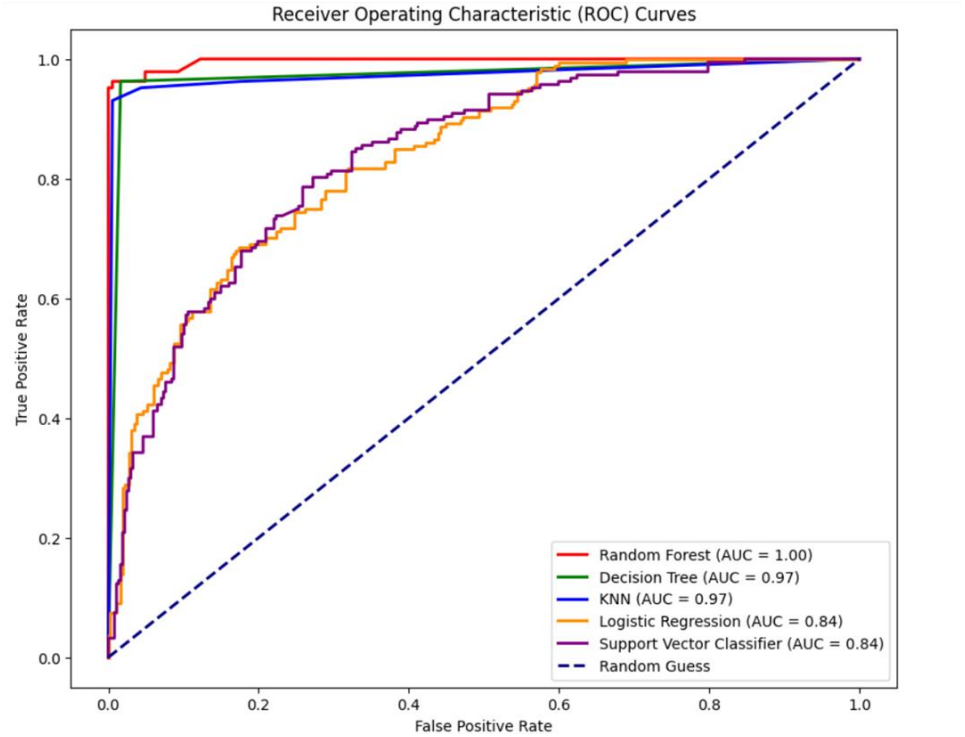


Fig 67: ROC curve for all models

Comparison of Accuray and Mean CV Scores for all Models

The model's performance for compared by plotting accuracy and mean cross-validation scores for all models. Python libraries such as the matplotlib and seaborn libraries were used to create a grouped bar plot comparing the performance of different machine learning models. The models considered are Logistic Regression, Support Vector Classifier, k-Nearest Neighbors, Decision Tree, and Random Forest. For each model, two bars are plotted side by side. The first set of bars (in purple) represents the accuracy scores of the models, while the second set (in coral) represents the mean cross-validated scores. The data for accuracy and mean cross-validated scores is provided in the accuracy_scores and mean_cv_scores lists, respectively. The bar plot is organized with distinct positions for each model along the x-axis, and values are rounded to two decimals for clarity. The legend, axis labels, and title enhance the interpretability of the plot, and numerical values are displayed on top of each bar for both accuracy and mean cross-validated scores. The overall visualization facilitates a quick comparison of model performance across the specified metrics.

```python
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# Assuming you have accuracy and mean_cv_scores for each model
models = ['Logistic Regression','Support Vector Classifier', 'k-Nearest Neighbors', 'Decision Tree','Random Forest']
accuracy_scores = [0.7888086642599278, 0.7545126353790613, 0.855595667870036, 0.9765342960288809, 0.9819494584837545]
mean_cv_scores = [0.7738400976622427, 0.7439580727770829, 0.8963317904962104, 0.9744637007920911, 0.9920499278631162]
# Round values to two decimals
accuracy_scores = [round(score, 2) for score in accuracy_scores]
mean_cv_scores = [round(score, 2) for score in mean_cv_scores]

# Create positions for grouped bar plot
bar_width = 0.35
index = np.arange(len(models))

# Plotting
plt.figure(figsize=(12, 6))
bars1 = plt.bar(index, accuracy_scores, width=bar_width, color='purple', label='Accuracy', align='center')
bars2 = plt.bar(index + bar_width, mean_cv_scores, width=bar_width, color='coral', label='Mean CV Score', align='center')

plt.title('Comparison of Models')
plt.xlabel('Model')
plt.ylabel('Score')
plt.xticks(index + bar_width/2, models)
plt.legend()

# Add values on top of the bars
for bar, score in zip(bars1, accuracy_scores):
    plt.text(bar.get_x() + bar.get_width()/2, bar.get_height(), str(score),
             ha='center', va='bottom', color='black', fontweight='bold')

for bar, score in zip(bars2, mean_cv_scores):
    plt.text(bar.get_x() + bar.get_width()/2, bar.get_height(), str(score),
             ha='center', va='bottom', color='black', fontweight='bold')

plt.show()
```

Figure 68: code snippet for plotting accuracy and mean cv scores for all models

Based on the visual representation of the grouped bar plot, it is evident that the Random Forest model stands out as the top-performing model for this particular dataset. It achieves an accuracy of 98% and a mean cross-validation score of 99%. These results highlight the Random Forest's robust performance and reliability in making accurate predictions on the given dataset, surpassing the other considered models, including Logistic Regression, Support Vector Classifier, k-Nearest Neighbors, and Decision Tree.

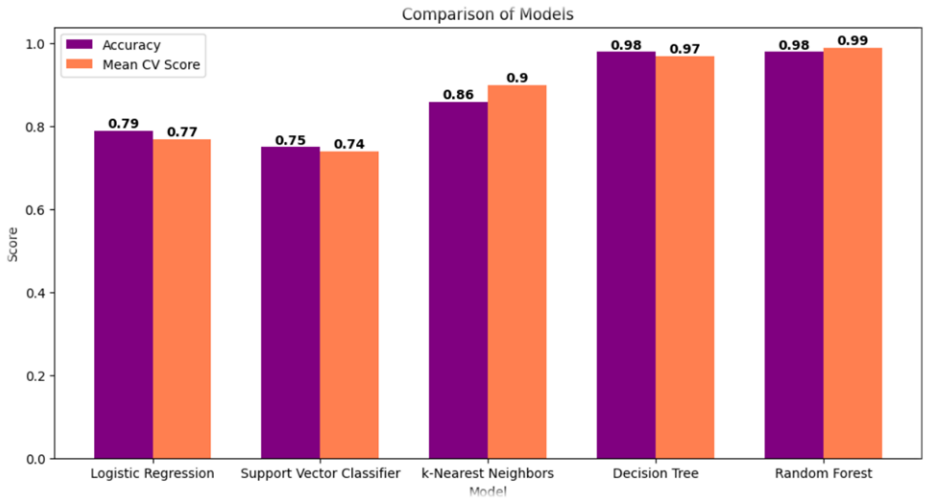


Figure 69: Bar plot showing Comparison of Accuracy and Mean CV scores for all Machine Learning Model

Result

The result is that we can reject the null hypothesis indicating that there is a significant relationship between health-related attributes and the likelihood of an individual developing diabetes.

Discussion

The rejection of the null hypothesis, indicating a significant relationship between health-related attributes and the likelihood of developing diabetes, holds paramount importance in the realm of healthcare and predictive modeling. This finding underscores the substantial impact that factors such as age, BMI, blood pressure, glucose levels, insulin levels, and familial diabetes history exert on an individual's susceptibility to diabetes. It reinforces the fundamental role of these health indicators in predicting and understanding diabetes risk.

This discovery carries profound clinical implications, emphasizing the critical nature of these attributes for healthcare professionals in risk assessment and early detection of diabetes. The correlation established between these factors and diabetes risk advocates for tailored interventions and vigilant monitoring for individuals exhibiting these risk indicators, enabling proactive healthcare management.

Moreover, the validation of these attributes as significant contributors to diabetes risk fortifies the reliability and accuracy of predictive models developed in this study. It bolsters confidence in the model's predictive capabilities, providing a robust foundation for precise risk assessment and informed decision-making in clinical settings.

Looking ahead, this outcome underscores the need for further in-depth research to explore the intricate relationships and nuanced impact of these attributes on diabetes development. Additionally, it accentuates the importance of personalized healthcare interventions, emphasizing the unique risk profiles of individuals for tailored preventive measures and targeted healthcare strategies.

In summary, the rejection of the null hypothesis regarding the relationship between health-related attributes and diabetes risk serves as a cornerstone, affirming the pivotal role of these indicators in predictive modeling. It empowers healthcare professionals with crucial insights for proactive interventions, paving the way for more effective diabetes prevention and management approaches in clinical practice.

Limitations

Some of the Limitations we experienced are:

- We imputed zero values in several of our rows with column means. This could introduce bias, affecting the models' reliability.
- Several of our columns had outliers, which although we addressed using winsorization, we believe might have affected the model's performance.
- There is a limiting factor of generalizability of our findings due to the exclusive inclusion of females in the dataset.

- The Reciever Operating Curve (ROC) for a Random Forest model shows an AUC of 1, indicating potential overfitting.

References

Bailes B. K. (2002). Diabetes mellitus and its chronic complications. AORN Journal, 76(2), 266–286. https://doi.org/10.1016/s0001-2092(06)61065-x

Ismail, L., Materwala, H., & Al Kaabi, J. (2021). Association of risk factors with type 2 diabetes: A systematic review. Computational and structural biotechnology journal, 19, 1759–1785. https://doi.org/10.1016/j.csbj.2021.03.003

Keita, Z. (n.d.). *Classification in Machine Learning: An Introduction*. Datacamp.com. Retrieved December 8, 2023, from https://www.datacamp.com/blog/classification-machine-learning

Nahm, F. S. (2016). Nonparametric statistical tests for the continuous data: the basic concept and the practical use. *Korean Journal of Anesthesiology*, *69*(1), 8. https://doi.org/10.4097/kjae.2016.69.1.8

Winsorization. (2021, May 27). GeeksforGeeks. https://www.geeksforgeeks.org/winsorization/

APPENDIX:

```
import MySQLdb
import pandas as pd

# Read MySQL credentials from file
myvars = {}
with open("eshshah-mysql-password") as myfile:
    for line in myfile:
        name, var = line.partition(":")[::2]
        myvars[name.strip()] = var.strip()

# Connect to MySQL database
conn = MySQLdb.connect(
    host="localhost",
    user=myvars['DB username'],
    passwd=myvars['DB password'],
    db=myvars['DB databasename']
)

cursor = conn.cursor()

# Select the specified database
cursor.execute('USE I501_Fall2023_Sec22490_group05_db;')

# Fetch data from the "healthcare_diabetes" table
cursor.execute('SELECT * FROM healthcare_diabetes;')
data = pd.DataFrame(list(cursor.fetchall()))

# Display the first few rows of the DataFrame
print(data.head())
```

```
import pandas as pd
```

```
# reading the data frame
df = pd.read_csv('Healthcare-Diabetes.csv')# often works
# df = pd.read_csv('Healthcare-Diabetes.csv', header=0, quotechar='"',sep=',', na_values = ['na', '-', '.', ''])
df
```

```
#showing the first 5 rows of the dataset
df.head()
```

```
# Series column data types
s = df.dtypes
s
```

```
# df.axes gives a list of axis labels
axes = df.axes
```

```
# number of axes
i = df.ndim
i
```

```
import pandas as pd
import matplotlib.pyplot as plt

# Creating a box plot for the 'Insulin' column
plt.figure(figsize=(8, 6))
plt.boxplot(df['Insulin'], vert=False,patch_artist=True, boxprops=dict(facecolor='lightblue'))
plt.title('Box Plot of Insulin Levels')
plt.xlabel('Insulin')
plt.show()
```

```
import pandas as pd
import matplotlib.pyplot as plt

# Creating a box plot for the 'Diabetes Pedigree Function' column
plt.figure(figsize=(8, 6))
plt.boxplot(df['DiabetesPedigreeFunction'], vert=False, patch_artist=True, boxprops=dict(facecolor='lightblue'))
plt.title('DiabetesPedigreeFunction')
plt.xlabel('DiabetesPedigreeFunction')
plt.show()
```

```
]: import pandas as pd
   import matplotlib.pyplot as plt

   # Creating a box plot for the 'BMI' column
   plt.figure(figsize=(8, 6))
   plt.boxplot(df['BMI'], vert=False, patch_artist=True, boxprops=dict(facecolor='lightblue'))
   plt.title('Box Plot of BMI')
   plt.xlabel('BMI')
   plt.show()
```

```python
import pandas as pd
import matplotlib.pyplot as plt

# Creating a box plot for the 'Glucose' column
plt.figure(figsize=(8, 6))
plt.boxplot(df['Glucose'], vert=False, patch_artist=True, boxprops=dict(facecolor='lightblue'))
plt.title('Box Plot of Glucose')
plt.xlabel('Glucose')
plt.show()
```

```python
import pandas as pd
import matplotlib.pyplot as plt

# Creating a box plot for the 'Blood Pressure' column
plt.figure(figsize=(8, 6))
plt.boxplot(df['BloodPressure'], vert=False, patch_artist=True, boxprops=dict(facecolor='lightblue'))
plt.title('Box Plot of BloodPressure')
plt.xlabel('BloodPressure')
plt.show()
```

```python
import pandas as pd
import matplotlib.pyplot as plt

# Creating a box plot for the 'Age' column
plt.figure(figsize=(8, 6))
plt.boxplot(df['Age'], vert=False, patch_artist=True, boxprops=dict(facecolor='lightblue'))
plt.title('Age')
plt.xlabel('Age')
plt.show()
```

```python
import pandas as pd
import matplotlib.pyplot as plt

# Create a box plot for the 'Skin Thickness' column
plt.figure(figsize=(8, 6))
plt.boxplot(df['SkinThickness'], vert=False, patch_artist=True, boxprops=dict(facecolor='lightblue'))
plt.title('SkinThickness')
plt.xlabel('SkinThickness')
plt.show()
```

```python
# finding the Interquantile Range of the columns
import numpy as np
import pandas as pd

columns_of_interest = ['Insulin', 'BMI', 'SkinThickness', 'BloodPressure', 'Age','DiabetesPedigreeFunction']

# Calculating the Interquantile Range for each specified column
for column in columns_of_interest:
    Q1 = np.percentile(df[column], 25)
    Q3 = np.percentile(df[column], 75)
    IQR = Q3 - Q1

    print(f"IQR for {column}: {IQR}")
```

```python
# Treating outliers by Winsorization
import numpy as np
from scipy.stats.mstats import winsorize
import pandas as pd

columns_of_interest = ['Insulin', 'BMI', 'SkinThickness', 'BloodPressure', 'Age','DiabetesPedigreeFunction']

# Set the winsorizing limits (capping at the 5th and 95th percentiles)
winsorizing_limits = [0.05, 0.05]

# Winsorize each specified column
for column in columns_of_interest:
    df[column] = winsorize(df[column], limits=winsorizing_limits)

# Print or use the winsorized DataFrame
print(df)
```

```python
# Creating subplots of histogram of specified columns showing their frequency distribution
import seaborn as sns
import matplotlib.pyplot as plt

# Set the figure size
plt.figure(figsize=(12, 16))

# Histogram for DiabetesPedigreeFunction
plt.subplot(4, 2, 1)
sns.histplot(data=df, x='DiabetesPedigreeFunction', bins=30, kde=True)
plt.title('DiabetesPedigreeFunction Distribution')
plt.xlabel('DiabetesPedigreeFunction')
plt.ylabel('Frequency')

# Histogram for Age
plt.subplot(4, 2, 2)
sns.histplot(data=df, x='Age', bins=30, kde=True)
plt.title('Age Distribution')
plt.xlabel('Age')
plt.ylabel('Frequency')

# Histogram for SkinThickness
plt.subplot(4, 2, 3)
sns.histplot(data=df, x='SkinThickness', bins=30, kde=True)
plt.title('SkinThickness Distribution')
plt.xlabel('SkinThickness')
plt.ylabel('Frequency')

# Histogram for Insulin
plt.subplot(4, 2, 4)
sns.histplot(data=df, x='Insulin', bins=30, kde=True)
plt.title('Insulin Distribution')
plt.xlabel('Insulin')
plt.ylabel('Frequency')
```

```python
# Histogram for BMI
plt.subplot(4, 2, 5)
sns.histplot(data=df, x='BMI', bins=30, kde=True)
plt.title('BMI Distribution')
plt.xlabel('BMI')
plt.ylabel('Frequency')

# Adjust layout and show the plot
plt.tight_layout()
plt.show()
```

```python
# Creating subplots of histogram of specified columns by Outcome
import pandas as pd
import matplotlib.pyplot as plt


# Separate data based on Outcome
outcome_0 = df[df['Outcome'] == 0]
outcome_1 = df[df['Outcome'] == 1]

# List of numerical columns to plot
numerical_columns = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age']

# Plot histograms for each numerical column
plt.figure(figsize=(15, 10))

for i, column in enumerate(numerical_columns, 1):
    plt.subplot(3, 3, i)
    plt.hist(outcome_0[column], bins=20, label='Outcome 0', alpha=0.5, color='blue')
    plt.hist(outcome_1[column], bins=20, label='Outcome 1', alpha=0.5, color='orange')
    plt.title(f'Histogram of {column} by Outcome')
    plt.xlabel(column)
    plt.ylabel('Frequency')
    plt.legend()

plt.tight_layout()
plt.show()
```

```python
#Barplot showing correlation of variables with outcome
import matplotlib.pyplot as plt
import pandas as pd

fig = plt.figure(figsize=(10, 4))

# Calculate correlation
outcome_corr = pd.DataFrame(df.corr()['Outcome'].sort_values(ascending=True))

# Set color for Outcome variable to red and others to blue
colors = ['orange' if var != 'Outcome' else 'red' for var in outcome_corr.index]

# Create bar plot
plt.barh(outcome_corr.index, outcome_corr['Outcome'], color=colors)
plt.title('Correlation with Outcome')
plt.show()
```

```python
# Barplot showing Diabetes distribution based on Outcome
import pandas as pd
import matplotlib.pyplot as plt

# Group the data by the 'Outcome' column and count the occurrences of 0 (no diabetes) and 1 (diabetes)
outcome_counts = df['Outcome'].value_counts()

# Create a bar chart
plt.figure(figsize=(8, 6))
bars = plt.bar(outcome_counts.index, outcome_counts.values, color=['skyblue', 'lightcoral'])

# Annotate each bar with its value count
for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, yval + 0.1, round(yval, 2), ha='center', va='bottom')

# Customize the plot
plt.title('Diabetes Distribution')
plt.xlabel('Outcome')
plt.ylabel('Count')
plt.xticks(outcome_counts.index, ['No Diabetes (0)', 'Diabetes (1)'])

# Show the bar chart
plt.show()
```

```
#Creating a heatmap to show the corelation matrix amongst independant and dependant variables
import pandas as pd
import matplotlib.pyplot as plt

# Remove the 'Id' column
df = df.drop(columns=['Id'])

# Calculate the correlation matrix for all remaining columns
correlation_matrix = df.corr()

# Define the number of rows and columns for the grid
n_rows, n_cols = correlation_matrix.shape

# Create a pink-colored correlation matrix heatmap with labeled cells, a larger size, and an increased grid size
plt.figure(figsize=(16, 12))
cax = plt.matshow(correlation_matrix, cmap='coolwarm', vmin=-1, vmax=1, aspect='auto', origin='lower',
                  extent=[-0.5, n_cols - 0.5, -0.5, n_rows - 0.5])  # Adjust grid size
cax.set_cmap('coolwarm_r')  # Adjust the colormap to get a pinkish appearance
plt.colorbar(cax)

# Set column and row labels
plt.xticks(range(n_cols), correlation_matrix.columns, rotation=90)
plt.yticks(range(n_rows), correlation_matrix.columns)

# Label the cells with correlation coefficients
for i in range(n_rows):
    for j in range(n_cols):
        plt.text(j, i, f"{correlation_matrix.iloc[i, j]:.2f}", ha='center', va='center', color='black')

plt.title('Correlation matrix')

# Show the heatmap
plt.show()
```

```
import pandas as pd
from scipy.stats import shapiro
import matplotlib.pyplot as plt
import numpy as np

# Create a list of column indices to exclude (e.g., first and last columns)
exclude_columns = [-1]

# Create a copy of the DataFrame with excluded columns for visualization
df_copy = df.drop(df.columns[exclude_columns], axis=1).copy()

# Set up subplots
fig, axes = plt.subplots(nrows=2, ncols=4, figsize=(18, 10))

# Flatten the 2D array of subplots into a 1D array
axes = axes.flatten()

# Loop through columns, perform Shapiro-Wilk test, and plot line plot or histogram for normally distributed data
for i, column in enumerate(df_copy.columns):
    # Shapiro-Wilk test
    stat, p_value = shapiro(df_copy[column])

    # Check if data is normally distributed based on the significance level (e.g., 0.05)
    alpha = 0.05
    is_normally_distributed = p_value > alpha

    # Line plot for normally distributed data
    if is_normally_distributed:
        sorted_data = np.sort(df_copy[column])
        axes[i].plot(sorted_data, np.linspace(0, 1, len(sorted_data), endpoint=False), color='blue')
        axes[i].set_title(f'Line Plot - {column}\nData is normally distributed (Shapiro p-value: {p_value})')
    else:
        # If data is not normally distributed, you might want to handle it differently (e.g., use a histogram)
        axes[i].set_title(f'Histogram - {column}\nData is not normally distributed \n(Shapiro p-value: {p_value})')
        axes[i].hist(df_copy[column], bins='auto', color='purple', edgecolor='white', alpha=0.7)
```

```
    # Rotate x-axis labels
    axes[i].tick_params(axis='x', rotation=45)

# Adjust layout
plt.tight_layout()

# Show plots
plt.show()
```

```python
# checking the normality using normal test

import pandas as pd
from scipy.stats import normaltest

# Set up a list to store results
normality_results = []

# Loop through columns and perform normality test
for column in df.columns:
    # Normality test
    statistic, p_value = normaltest(df[column])

    # Store results
    normality_results.append({
        'Column': column,
        'Statistic': statistic,
        'P-value': p_value,
        'Is Normally Distributed': p_value > 0.05  # Adjust the significance level as needed
    })

# Convert results to DataFrame for easier inspection
results_df = pd.DataFrame(normality_results)

# Print or inspect the results DataFrame
print(results_df)
```

```python
# checking the normality using Shapiro-Wilk test and plotting QQ Plots

import pandas as pd
from scipy.stats import shapiro, probplot
import matplotlib.pyplot as plt

 # Create a list of column indices to exclude (e.g., first and last columns)
exclude_columns = [-1]

# Create a copy of the DataFrame with excluded columns for visualization
df_copy = df.drop(df.columns[exclude_columns], axis=1).copy()
# Set up subplots with constrained layout
fig, axes = plt.subplots(nrows=2, ncols=4, figsize=(20, 8), constrained_layout=True)

# Flatten the 2D array of subplots into a 1D array
axes = axes.flatten()

# Loop through columns
for i, column in enumerate(df.columns[:8]):  # Adjust the number of columns as needed
    # Shapiro-Wilk test
    stat, p_value = shapiro(df[column])

    # Check if data is normally distributed based on the significance level (e.g., 0.05)
    alpha = 0.05
    is_normally_distributed = p_value > alpha

    #QQ plot
    probplot(df[column], plot=axes[i], rvalue=True)
    axes[i].set_title(f'QQ Plot - {column}\nData is normally distributed (Shapiro p-value: {p_value})' if is_normally_distributed else f'QQ Pl

# Adjust layout
# plt.tight_layout()

# Show plots
plt.show()
```

```python
for p in ax.patches:
    percentage = '{:.1f}%'.format(p.get_height())
    x = p.get_x() + p.get_width() / 2 - 0.15
    y = p.get_y() + p.get_height() + 0.1
    ax.annotate(percentage, (x, y), fontsize=10)

# Show the plot
plt.show()
```

```
import seaborn as sns
import matplotlib.pyplot as plt

# Create Pregnancy groups
pregnancy_bins = [0, 2, 5, 10, 15, 19]
pregnancy_labels = ['0-3', '4-7', '8-11', '12-15', '16-19']
df_Copy['Pregnancy_Group'] = pd.cut(df_Copy['Pregnancies'], bins=pregnancy_bins, labels=pregnancy_labels, right=False)

# Perform Chi-Square Test of Independence for Pregnancy groups
contingency_table_pregnancy = pd.crosstab(df_Copy['Pregnancy_Group'], df_Copy['Outcome'])
chi2_pregnancy, p_value_pregnancy, _, _ = chi2_contingency(contingency_table_pregnancy)

# Print results for Pregnancy
print(f"Chi-Square Value for Pregnancy: {chi2_pregnancy:.4f}")
print(f"P-value for Pregnancy: {p_value_pregnancy}")

# Interpretation for Pregnancy
if p_value_pregnancy < 0.05:
    print("Reject the null hypothesis. There is evidence of an association between Pregnancy groups and the diabetes outcome.")
else:
    print("Fail to reject the null hypothesis. There is insufficient evidence to claim an association for Pregnancy.")

# Calculate percentages for each category within each group
group_percentages_pregnancy = (contingency_table_pregnancy.T / contingency_table_pregnancy.sum(axis=1)).T * 100

# Fill NaN values with 0
group_percentages_pregnancy = group_percentages_pregnancy.fillna(0)

# Visualize the relationship between Pregnancy Group and Outcome with percentages
plt.figure(figsize=(12, 6))
ax = sns.barplot(x='Pregnancy_Group', y='Percentage', hue='Outcome', data=group_percentages_pregnancy.reset_index().melt(id_vars='Pregnancy_Gr

plt.title('Distribution of Outcome by Pregnancy Group')
plt.xlabel('Pregnancy Group')
plt.ylabel('Percentage')
plt.legend(title='Outcome', loc='upper left')
```

```
# Add percentage labels to each bar
for p in ax.patches:
    percentage = '{:.1f}%'.format(p.get_height())
    x = p.get_x() + p.get_width() / 2 - 0.15
    y = p.get_y() + p.get_height() + 0.1
    ax.annotate(percentage, (x, y), fontsize=10)

# Show the plot
plt.show()
```

```
import seaborn as sns
import matplotlib.pyplot as plt
from scipy.stats import chi2_contingency
import pandas as pd


# Create Glucose groups
glucose_bins = [0, 80, 120, 160, 200]
glucose_labels = ['0-80', '81-120', '121-160', '161-200']
df_Copy['Glucose_Group'] = pd.cut(df_Copy['Glucose'], bins=glucose_bins, labels=glucose_labels, right=False)

# Perform Chi-Square Test of Independence for Glucose groups
contingency_table_glucose = pd.crosstab(df_Copy['Glucose_Group'], df_Copy['Outcome'])
chi2_glucose, p_value_glucose, _, _ = chi2_contingency(contingency_table_glucose)

# Print results for Glucose
print(f"Chi-Square Value for Glucose: {chi2_glucose:.4f}")
print(f"P-value for Glucose: {p_value_glucose}")

# Interpretation for Glucose
if p_value_glucose < 0.05:
    print("Reject the null hypothesis. There is evidence of an association between Glucose groups and the diabetes outcome.")
else:
    print("Fail to reject the null hypothesis. There is insufficient evidence to claim an association for Glucose.")

# Calculate percentages for each category within each group
group_percentages = (contingency_table_glucose.T / contingency_table_glucose.sum(axis=1)).T * 100

# Visualize the relationship between Glucose Group and Outcome with percentages
plt.figure(figsize=(12, 6))
ax = sns.barplot(x='Glucose_Group', y='Percentage', hue='Outcome', data=group_percentages.reset_index().melt(id_vars='Glucose_Group', va

plt.title('Distribution of Outcome by Glucose Group')
plt.xlabel('Glucose Group')
plt.ylabel('Percentage')
plt.legend(title='Outcome', loc='upper right')
```

```
# Add percentage labels to each bar
for p in ax.patches:
    percentage = '{:.1f}%'.format(p.get_height())
    x = p.get_x() + p.get_width() / 2 - 0.15
    y = p.get_y() + p.get_height() + 0.1
    ax.annotate(percentage, (x, y), fontsize=10)

plt.show()
```

```
import seaborn as sns
import matplotlib.pyplot as plt

# Create BMI groups
bmi_bins = [0, 18.5, 25, 30, 40, 50]
bmi_labels = ['0-18.5', '18.6-25', '25.1-30', '30.1-40', '40.1-50']
df_Copy['BMI_Group'] = pd.cut(df_Copy['BMI'], bins=bmi_bins, labels=bmi_labels, right=False)

# Perform Chi-Square Test of Independence for BMI groups
contingency_table_bmi = pd.crosstab(df_Copy['BMI_Group'], df_Copy['Outcome'])
chi2_bmi, p_value_bmi, _, _ = chi2_contingency(contingency_table_bmi)

# Print results for BMI
print(f"Chi-Square Value for BMI: {chi2_bmi:.4f}")
print(f"P-value for BMI: {p_value_bmi}")

# Interpretation for BMI
if p_value_bmi < 0.05:
    print("Reject the null hypothesis. There is evidence of an association between BMI groups and the diabetes outcome.")
else:
    print("Fail to reject the null hypothesis. There is insufficient evidence to claim an association for BMI.")

# Calculate percentages for each category within each group
group_percentages_bmi = (contingency_table_bmi.T / contingency_table_bmi.sum(axis=1)).T * 100

# Fill NaN values with 0
group_percentages_bmi = group_percentages_bmi.fillna(0)

# Visualize the relationship between BMI Group and Outcome with percentages
plt.figure(figsize=(12, 6))
ax_bmi = sns.barplot(x='BMI_Group', y='Percentage', hue='Outcome', data=group_percentages_bmi.reset_index().melt(id_vars='BMI_Group', var_name

plt.title('Distribution of Outcome by BMI Group')
plt.xlabel('BMI Group')
plt.ylabel('Percentage')
plt.legend(title='Outcome', loc='upper right')
```

```
# Add percentage labels to each bar
for p in ax_bmi.patches:
    percentage = '{:.1f}%'.format(p.get_height())
    x = p.get_x() + p.get_width() / 2 - 0.15
    y = p.get_y() + p.get_height() + 0.1
    ax_bmi.annotate(percentage, (x, y), fontsize=10)

# Show the plot
plt.show()
```

```
pip install -U imbalanced-learn
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: imbalanced-learn in ./.local/lib/python3.10/site-packages (0.11.0)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn) (1.24.1)
Requirement already satisfied: scipy>=1.5.0 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn) (1.9.3)
Requirement already satisfied: scikit-learn>=1.0.2 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn) (1.3.0)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn) (1.2.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn) (3.1.0)

[notice] A new release of pip is available: 23.2.1 -> 23.3.1
[notice] To update, run: python3 -m pip install --upgrade pip
Note: you may need to restart the kernel to use updated packages.
```

SPLITTING THE DATA INTO TRAIN-TEST SPLIT FOLLOWED BY SMOTE (Synthetic Minority Over-sampling Technique) FOR MANAGING IMBALANCED DATA

```python
import pandas as pd
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split

# 'Outcome' is our binary target variable
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)

# Print class distribution before SMOTE on the training set
original_train_class_distribution = pd.Series(y_train).value_counts()
print("Class Distribution of Training Set Before SMOTE:")
print(original_train_class_distribution)

# Apply SMOTE only to the training set
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

# Print class distribution after SMOTE on the training set
resampled_train_class_distribution = pd.Series(y_train_resampled).value_counts()
print("\nClass Distribution of Training Set After SMOTE:")
print(resampled_train_class_distribution)
```

```python
import pandas as pd
import statsmodels.api as sm
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming you have your training and testing data loaded into X_train_resampled, y_train_resampled, X_test, and y_test

# Add a constant term to the features for intercept in statsmodels
X_train_resampled = sm.add_constant(X_train_resampled)
X_test = sm.add_constant(X_test)

# Create logistic regression model
logit_model = sm.Logit(y_train_resampled, X_train_resampled)

# Fit the model
result = logit_model.fit()

print(result.summary())

odds_ratios = pd.DataFrame({'Odds Ratio': result.params.apply(lambda x: round(np.exp(x), 3)),
                            '95% CI (lower)': result.conf_int()[0].apply(lambda x: round(np.exp(x), 3)),
                            '95% CI (upper)': result.conf_int()[1].apply(lambda x: round(np.exp(x), 3))})

print("\nOdds Ratios:")
print(odds_ratios)
```

```python
import matplotlib.pyplot as plt

# Assuming 'result' is the fitted logistic regression result
coefficients = result.params.drop('const').sort_values(ascending=False)

# Plotting in descending order
fig, ax = plt.subplots(figsize = (10,6))
bars = ax.barh(coefficients.index, coefficients, color=np.where(coefficients >= 0, 'green', 'red'))

# Add values as text annotations
for bar in bars:
    xval = bar.get_width()
    plt.text(xval +0.05, bar.get_y() + bar.get_height()/2, round(xval, 3), ha='left', va='center', color='black')

# Set even x-axis ticks from -0.2 to 1.2
ax.set_xticks([-0.2, 0, 0.2, 0.4, 0.6, 0.8, 1, 1.2, 1.4])

plt.title('Logistic Regression Coefficients (Descending Order)')
plt.xlabel('Coefficient Value')
plt.show()
```

```
import seaborn as sns

# Assuming odds_ratios is a DataFrame containing odds ratios and CIs
plt.errorbar(odds_ratios['Odds Ratio'], range(len(odds_ratios)),
             xerr=(odds_ratios['Odds Ratio'] - odds_ratios['95% CI (lower)'], odds_ratios['95% CI (upper)'] - odds_ratios['Odds Ratio']),
             fmt='o', linestyle='None', color = 'magenta')
plt.yticks(range(len(odds_ratios)), odds_ratios.index)
plt.xscale('log')  # Use log scale for better visualization
plt.title('Odds Ratios with 95% Confidence Intervals')
plt.show()
```

```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Create a logistic regression model
model = LogisticRegression()

# Perform k-fold cross-validation (e.g., k=5)
cv_scores = cross_val_score(model, X_train_resampled, y_train_resampled, cv=5)

# Train the model on the resampled training data
model.fit(X_train_resampled, y_train_resampled)

# Make predictions on the testing data
y_pred_lr = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred_lr)
conf_matrix = confusion_matrix(y_test, y_pred_lr)
class_report = classification_report(y_test, y_pred_lr)

# Print the results
print(f"Cross-Validation Scores: {cv_scores}")
print(f"Mean CV Score: {cv_scores.mean()}")
print(f"Accuracy: {accuracy}")
print(f"Confusion Matrix:\n{conf_matrix}")
print(f"Classification Report:\n{class_report}")

# Plot the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=['Non-Diabetic', 'Diabetic'], yticklabels=['Non-Diabetic', 'Diabetic']
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```

```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# Make predictions on the test set
y_pred_proba_lr = model.predict_proba(X_test)[:, 1]

# Calculate false positive rate and true positive rate
fpr_lr, tpr_lr, thresholds_lr = roc_curve(y_test, y_pred_proba_lr)

# Calculate AUC
roc_auc_lr = auc(fpr_lr, tpr_lr)

# Plot ROC Curve with shading
plt.figure(figsize=(8, 6))
plt.plot(fpr_lr, tpr_lr, label='ROC curve (AUC = {:.2f})'.format(roc_auc_lr), color='darkorange', lw=2)
plt.fill_between(fpr_lr, 0, tpr_lr, color='skyblue', alpha=0.3, label='AUC')  # Shading the area under the ROC curve
plt.plot([0, 1], [0, 1], 'k--', label='Random Guess')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve (Logistic Regression)')
plt.legend(loc="lower right")
plt.show()
```

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Create an SVC model
svc_model = SVC(probability = True)

# Perform k-fold cross-validation (e.g., k=5)
cv_scores = cross_val_score(svc_model, X_train_resampled, y_train_resampled, cv=5)

# Train the model on the training data
svc_model.fit(X_train_resampled, y_train_resampled)

# Make predictions on the testing data
y_pred = svc_model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

# Print the results
print(f"Cross-Validation Scores: {cv_scores}")
print(f"Mean CV Score: {cv_scores.mean()}")
print(f"Accuracy: {accuracy}")
print(f"Confusion Matrix:\n{conf_matrix}")
print(f"Classification Report:\n{class_report}")

# Plot the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Greens", xticklabels=['Non-Diabetic', 'Diabetic'], yticklabels=['Non-Diabetic', 'Diabetic'])
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```

```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc

# Calculate probability estimates for the positive class
y_pred_proba_svc = svc_model.predict_proba(X_test)[:, 1]

# Calculate ROC curve
fpr_svc, tpr_svc, thresholds_svc = roc_curve(y_test, y_pred_proba_svc)
roc_auc_svc = auc(fpr_svc, tpr_svc)

# Plot ROC curve with shading
plt.figure(figsize=(8, 6))
plt.plot(fpr_svc, tpr_svc, color='darkorange', lw=2, label='ROC curve (AUC = {:.2f})'.format(roc_auc_svc))
plt.fill_between(fpr_svc, 0, tpr_svc, color='skyblue', alpha=0.3, label='AUC')  # Shading the area under the ROC curve
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--', label='Random Guess')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve (Support Vector Classifier)')
plt.legend(loc='lower right')
plt.show()
```

4]:
```
import pandas as pd
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Create a k-Nearest Neighbors model (let's use k=5 as an example)
knn_model = KNeighborsClassifier(n_neighbors=5)

# Perform k-fold cross-validation (e.g., k=5)
cv_scores = cross_val_score(knn_model, X_train_resampled, y_train_resampled, cv=5)

# Train the model on the training data
knn_model.fit(X_train_resampled, y_train_resampled)

# Make predictions on the testing data
y_pred = knn_model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

# Print the results
print(f"Cross-Validation Scores: {cv_scores}")
print(f"Mean CV Score: {cv_scores.mean()}")
print(f"Accuracy: {accuracy}")
print(f"Confusion Matrix:\n{conf_matrix}")
print(f"Classification Report:\n{class_report}")

plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=['Non-Diabetic', 'Diabetic'], yticklabels=['Non-Diabetic', 'Diabetic']
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```

```python
import pandas as pd
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

knn_model = KNeighborsClassifier(n_neighbors=3)

# Perform k-fold cross-validation (e.g., k=5)
cv_scores = cross_val_score(knn_model, X_train_resampled, y_train_resampled, cv=5)

# Train the model on the training data
knn_model.fit(X_train_resampled, y_train_resampled)

# Make predictions on the testing data
y_pred = knn_model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

# Print the results
print(f"Cross-Validation Scores: {cv_scores}")
print(f"Mean CV Score: {cv_scores.mean()}")
print(f"Accuracy: {accuracy}")
print(f"Confusion Matrix:\n{conf_matrix}")
print(f"Classification Report:\n{class_report}")

plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="BuGn", xticklabels=['Non-Diabetic', 'Diabetic'], yticklabels=['Non-Diabetic', 'Diabetic'])
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```

```python
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc

# Calculate probability estimates for the positive class
y_pred_proba_knn = knn_model.predict_proba(X_test)[:, 1]

# Calculate ROC curve and AUC
fpr_knn, tpr_knn, thresholds_knn = roc_curve(y_test, y_pred_proba_knn)
roc_auc_knn = auc(fpr_knn, tpr_knn)

# Plot ROC curve with shaded AUC
plt.figure(figsize=(8, 6))
plt.plot(fpr_knn, tpr_knn, color='darkorange', lw=2, label='ROC curve (AUC = {:.2f})'.format(roc_auc_knn))
plt.fill_between(fpr_knn, 0, tpr_knn, color='skyblue', alpha=0.3, label='AUC')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--', label='Random Guess')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve (K-nearest Neighbor)')
plt.legend(loc='lower right')
plt.show()
```

```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Create a decision tree model
tree_model = DecisionTreeClassifier(random_state=42)

# Perform k-fold cross-validation (e.g., k=5)
cv_scores = cross_val_score(tree_model, X_train_resampled, y_train_resampled, cv=5)

# Train the decision tree model on the resampled training data
tree_model.fit(X_train_resampled, y_train_resampled)

# Make predictions on the testing data
y_pred = tree_model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

# Print the results
print(f"Cross-Validation Scores: {cv_scores}")
print(f"Mean CV Score: {cv_scores.mean()}")
print(f"Accuracy: {accuracy}")
print(f"Confusion Matrix:\n{conf_matrix}")
print(f"Classification Report:\n{class_report}")

# Plot the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="PuRd", xticklabels=['Non-Diabetic', 'Diabetic'], yticklabels=['Non-Diabetic', 'Diabetic'])
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```

```python
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc

# Calculate probability estimates for the positive class
y_pred_proba_dt = tree_model.predict_proba(X_test)[:, 1]

# Calculate ROC curve and AUC
fpr_dt, tpr_dt, thresholds_dt = roc_curve(y_test, y_pred_proba_dt)
roc_auc_dt = auc(fpr_dt, tpr_dt)

# Plot ROC curve with shaded AUC
plt.figure(figsize=(8, 6))
plt.plot(fpr_dt, tpr_dt, color='darkorange', lw=2, label='ROC curve (AUC = {:.2f})'.format(roc_auc_dt))
plt.fill_between(fpr_dt, 0, tpr_dt, color='skyblue', alpha=0.3, label='AUC')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--', label='Random Guess')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve (Decision Tree)')
plt.legend(loc='lower right')
plt.show()
```

```python
import pandas as pd
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Create a Random Forest model
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)

# Perform k-fold cross-validation (e.g., k=5)
cv_scores = cross_val_score(rf_model, X_train_resampled, y_train_resampled, cv=5)

# Train the model on the training data
rf_model.fit(X_train_resampled, y_train_resampled)

# Make predictions on the testing data
y_pred = rf_model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

# Print the results
print(f"Cross-Validation Scores: {cv_scores}")
print(f"Mean CV Score: {cv_scores.mean()}")
print(f"Accuracy: {accuracy}")
print(f"Confusion Matrix:\n{conf_matrix}")
print(f"Classification Report:\n{class_report}")

plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=['Non-Diabetic', 'Diabetic'], yticklabels=['Non-Diabetic', 'Diabetic']
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```

```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc

# Assuming you have already trained the Random Forest model (rf_model) and made predictions (y_pred) on the test set

# Calculate probability estimates for the positive class
y_pred_proba_rf = rf_model.predict_proba(X_test)[:, 1]

# Calculate ROC curve and AUC
fpr_rf, tpr_rf, thresholds_rf = roc_curve(y_test, y_pred_proba_rf)
roc_auc_rf = auc(fpr_rf, tpr_rf)

# Plot ROC curve with shaded AUC
plt.figure(figsize=(8, 6))
plt.plot(fpr_rf, tpr_rf, color='darkorange', lw=2, label='ROC curve (AUC = {:.2f})'.format(roc_auc_rf))
plt.fill_between(fpr_rf, 0, tpr_rf, color='skyblue', alpha=0.3, label='AUC')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--', label='Random Guess')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve(Random Forest)')
plt.legend(loc='lower right')
plt.show()
```

```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc

# Logistic Regression
fpr_lr, tpr_lr, thresholds_lr = roc_curve(y_test, y_pred_proba_lr)
roc_auc_lr = auc(fpr_lr, tpr_lr)
# Support Vector Classifier
fpr_svc, tpr_svc, thresholds_svc = roc_curve(y_test, y_pred_proba_svc)
roc_auc_svc = auc(fpr_svc, tpr_svc)
# KNN
fpr_knn, tpr_knn, thresholds_knn = roc_curve(y_test, y_pred_proba_knn)
roc_auc_knn = auc(fpr_knn, tpr_knn)
# Decision Tree
fpr_dt, tpr_dt, thresholds_dt = roc_curve(y_test, y_pred_proba_dt)
roc_auc_dt = auc(fpr_dt, tpr_dt)
# Random Forest
fpr_rf, tpr_rf, thresholds_rf = roc_curve(y_test, y_pred_proba_rf)
roc_auc_rf = auc(fpr_rf, tpr_rf)

# Plot ROC curves for each model
plt.figure(figsize=(10, 8))
plt.plot(fpr_rf, tpr_rf, color='red', lw=2, label='Random Forest (AUC = {:.2f})'.format(roc_auc_rf))
plt.plot(fpr_dt, tpr_dt, color='green', lw=2, label='Decision Tree (AUC = {:.2f})'.format(roc_auc_dt))
plt.plot(fpr_knn, tpr_knn, color='blue', lw=2, label='KNN (AUC = {:.2f})'.format(roc_auc_knn))
plt.plot(fpr_lr, tpr_lr, color='darkorange', lw=2, label='Logistic Regression (AUC = {:.2f})'.format(roc_auc_lr))
plt.plot(fpr_svc, tpr_svc, color='purple', lw=2, label='Support Vector Classifier (AUC = {:.2f})'.format(roc_auc_svc))

# Plot the random guess line
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--', label='Random Guess')

# Set plot properties
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curves')
plt.legend(loc='lower right')
plt.show()
```

```python
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# Assuming you have accuracy and mean_cv_scores for each model
models = ['Logistic Regression','Support Vector Classifier', 'k-Nearest Neighbors', 'Decision Tree','Random Forest']
accuracy_scores = [0.7888086642599278, 0.7545126353790613, 0.855595667870036, 0.9765342960288809, 0.9819494584837545]
mean_cv_scores = [0.7738400976622427, 0.7439580727770829, 0.8963317904962104, 0.9744637007920911, 0.9920499278631162]
# Round values to two decimals
accuracy_scores = [round(score, 2) for score in accuracy_scores]
mean_cv_scores = [round(score, 2) for score in mean_cv_scores]

# Create positions for grouped bar plot
bar_width = 0.35
index = np.arange(len(models))

# Plotting
plt.figure(figsize=(12, 6))
bars1 = plt.bar(index, accuracy_scores, width=bar_width, color='purple', label='Accuracy', align='center')
bars2 = plt.bar(index + bar_width, mean_cv_scores, width=bar_width, color='coral', label='Mean CV Score', align='center')

plt.title('Comparison of Models')
plt.xlabel('Model')
plt.ylabel('Score')
plt.xticks(index + bar_width/2, models)
plt.legend()

# Add values on top of the bars
for bar, score in zip(bars1, accuracy_scores):
    plt.text(bar.get_x() + bar.get_width()/2, bar.get_height(), str(score),
             ha='center', va='bottom', color='black', fontweight='bold')

for bar, score in zip(bars2, mean_cv_scores):
    plt.text(bar.get_x() + bar.get_width()/2, bar.get_height(), str(score),
             ha='center', va='bottom', color='black', fontweight='bold')

plt.show()
```