ព្រះរាជាណាចក្រកម្ពុជា
ជាតិ សាសនា ព្រះមហាក្សត្រ

Introduction to Data Science

"Diabetes Predictor"

Group 6:

Name of Student	ID of student	Score
HUON SOPANHA	e20220209	……..
HANG MUYKHORNG	e20220166	……..
CHHOUK PHALTHUNIN	e20220467	……..
CHOEURN BROSPOV	e20221157	……..
HUN SOPHEAK	e20220446	……..

Professors: Dr. PHAUK Sokkhey (course and tp)

Academic year 2024-2025

Contents

1. Introduction

1.1. Background

In recent years, the prevalence of unhealthy dietary habits has become a significant concern in many societies. Many individuals prefer diets that are excessively sweet, salty, or high in unhealthy fats. Additionally, the consumption of energy drinks has surged, particularly among those working long hours or in high-stress environments. These dietary choices have been linked to an increased risk of chronic diseases, including diabetes.

Diabetes, a condition characterized by high blood sugar levels, has become increasingly common, posing serious health challenges to both individuals and healthcare systems. Statistics reveal a noticeable rise in diabetes cases across diverse demographics, emphasizing the urgent need for effective preventative measures and early detection. This trend is further exacerbated in societies where awareness of balanced nutrition and healthy lifestyles is limited.

Compounding the problem is the lack of access to advanced medical equipment and diagnostic tools in rural and underserved areas. While urban centers often benefit from modern healthcare facilities, rural communities face significant disparities in medical care. This gap hinders early diagnosis and timely treatment for diabetes patients, leading to severe complications and reduced quality of life.

The combination of unhealthy diets, a growing number of diabetes cases, and limited medical resources underscores the importance of innovative solutions. Addressing these challenges requires a multifaceted approach, including the development of accessible technologies for early diabetes detection. This project aims to contribute to this effort by creating a web-based tool that leverages machine learning to predict the likelihood of diabetes, providing a simple yet effective resource for individuals and healthcare providers alike.

1.2. Objective

The objectives of this project are:

- **Develop a machine learning model for diabetes prediction:** The primary goal is to develop a machine learning algorithms to predict the likelihood of diabetes using health-related information such as age, BMI, glucose levels, and other factors. This model will provide valuable insights and act as a decision support tool for early diagnosis.
- **Provide a user-friendly solution for rural hospitals:** develop a web-based application to forecast the risk of diabetes via machine learning methods. The application provides a user-friendly diagnostic tool for rural healthcare providers, bridging the gap in advanced medical equipment, enhancing access to early diabetes screening and supporting underserved communities.

2. Dataset and Data Description

- Dataset: “Diabetes Dataset”
- Source: Kaggle [diabetes/dataset]
- Purpose: The objective of the dataset is to diagnostically predict whether a patient has diabetes, based on certain diagnostic measurements included in the dataset.
- The dataset contains 768 rows and 9 columns.
- Data description:

- Pregnancies: to express the number of pregnancies
- Glucose: to express the glucose level in blood
- BloodPressure: to express the blood pressure measurement (mm Hg)
- SkinThickness: to express the thickness of the skin (mm)
- Insulin: to express the insulin level in blood (mu U/ml)
- BMI: Body mass index (weight in kg/ (height in m) ^2)
- DiabetesPedigreeFunction: to express the diabetes percentage
- Age: Age (years)
- Outcome: to express the final result 1 means positive (diabetes) and 0 means negative (no diabetes)

Feature Name	Data Type	Type of variable
Pregnancies	Integer	Numerical
Glucose	Integer	Numerical
BloodPressure	Integer	Numerical
SkinThickness	Integer	Numerical
Insulin	Integer	Numerical
BMI	Float	Numerical
DiabetesPedigreeFunction	Float	Numerical
Age	Integer	Numerical
Outcome	Integer	Binary

3. Methodology

3.1. Data Cleaning Process

Before building the machine learning model, it is essential to ensure the dataset is clean, consist, and ready for analysis. Therefore, data cleaning is the foundational step in the data science process because the model's dependability and performance are directly impacted by the quality of the data. The following are the process we do to clean the data:

- **Importing necessary libraries**

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.experimental import enable_iterative_imputer  # This line enables the feature
from sklearn.impute import IterativeImputer
%matplotlib inline
```

- **Load the dataset**

```
data = pd.read_csv("diabetes.csv")
```

- **Inspect the first rows of the dataset**

```
data.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

- **Inspect the ending rows of the dataset**

```
data.tail()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

- **Check the number of rows and columns**

```
data.shape
```

(768, 9)

- **Access the first row of the DataFrame**

```
data.iloc[0]
```

```
Pregnancies                 6.000
Glucose                   148.000
BloodPressure              72.000
SkinThickness              35.000
Insulin                     0.000
BMI                        33.600
DiabetesPedigreeFunction    0.627
Age                        50.000
Outcome                     1.000
Name: 0, dtype: float64
```

- **Check for missing values**

```
data.isna().sum()
```

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

- **Quick overview of the DataFrame's structure, data types, and missing values**

```
data.info()
```

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

- **Descriptive statistics for each column**

```
# Get summary statistics
summary = data.describe()

# Calculate mode
modes = data.mode().iloc[0]  # Get the first mode for each column

# Display describe() and mode
print("Summary Statistics:\n", summary)
print("\nMode for Each Column:\n", modes)
```

```
Summary Statistics:
       Pregnancies     Glucose  BloodPressure  SkinThickness     Insulin  \
count   768.000000  768.000000     768.000000     768.000000  768.000000
mean      3.845052  120.894531      69.105469      20.536458   79.799479
std       3.369578   31.972618      19.355807      15.952218  115.244002
min       0.000000    0.000000       0.000000       0.000000    0.000000
25%       1.000000   99.000000      62.000000       0.000000    0.000000
50%       3.000000  117.000000      72.000000      23.000000   30.500000
75%       6.000000  140.250000      80.000000      32.000000  127.250000
max      17.000000  199.000000     122.000000      99.000000  846.000000

              BMI  DiabetesPedigreeFunction         Age     Outcome
count  768.000000                768.000000  768.000000  768.000000
mean    31.992578                  0.471876   33.240885    0.348958
std      7.884160                  0.331329   11.760232    0.476951
min      0.000000                  0.078000   21.000000    0.000000
25%     27.300000                  0.243750   24.000000    0.000000
50%     32.000000                  0.372500   29.000000    0.000000
75%     36.600000                  0.626250   41.000000    1.000000
max     67.100000                  2.420000   81.000000    1.000000

Mode for Each Column:
 Pregnancies                 1.000
Glucose                     99.000
BloodPressure               70.000
SkinThickness                0.000
Insulin                      0.000
BMI                         32.000
DiabetesPedigreeFunction     0.254
Age                         22.000
Outcome                      0.000
Name: 0, dtype: float64
```

- **Check for data imbalance**

```
data['Outcome'].value_counts()
```

```
Outcome
0    500
1    268
Name: count, dtype: int64
```

- **Data imbalance in term of probability (percentage)**

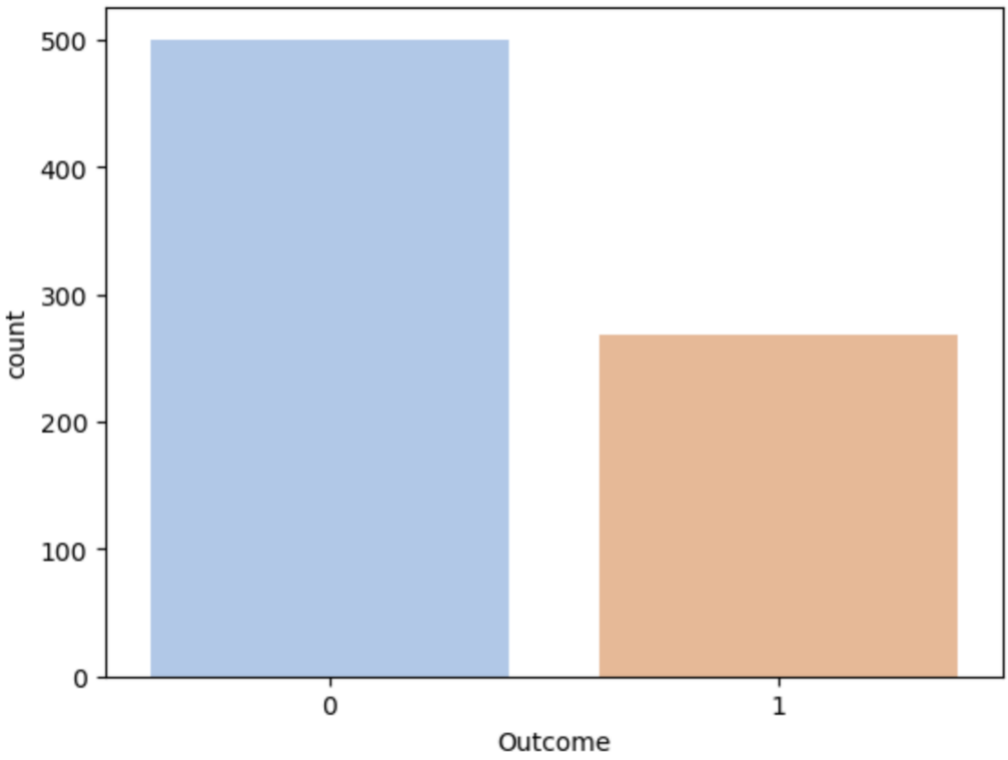
```
data['Outcome'].value_counts(normalize=True)
```

```
Outcome
0    0.651042
1    0.348958
Name: proportion, dtype: float64
```

- **Outcome values visualizing**

```
sns.countplot(x='Outcome', data=data, palette = 'pastel')
```

```
<Axes: xlabel='Outcome', ylabel='count'>
```



- **Variable explorations + identifying shape & outliers**

```
def find_outlier(cols,data):#outlier
    Q1 = data[cols].quantile(0.25)
    Q3 = data[cols].quantile(0.75)
    IQR = Q3 - Q1

    # Calculate bounds
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    # Find outliers
    outliers_list = data[(data[cols] < lower_bound) | (data[cols] >
upper_bound)][cols].unique().tolist()

    # Print results
    print(f"Lower Bound: {lower_bound}")
    print(f"Upper Bound: {upper_bound}")
    print(f"Outliers:\n{outliers_list}")
    return outliers_list
```

```
def clean_outlier(col,outlier,df,MMM ):
    for i in outlier :
        ind =df.index[df[col]==i].tolist()
        df.iloc[ind, df.columns.get_loc(col)] = MMM
```

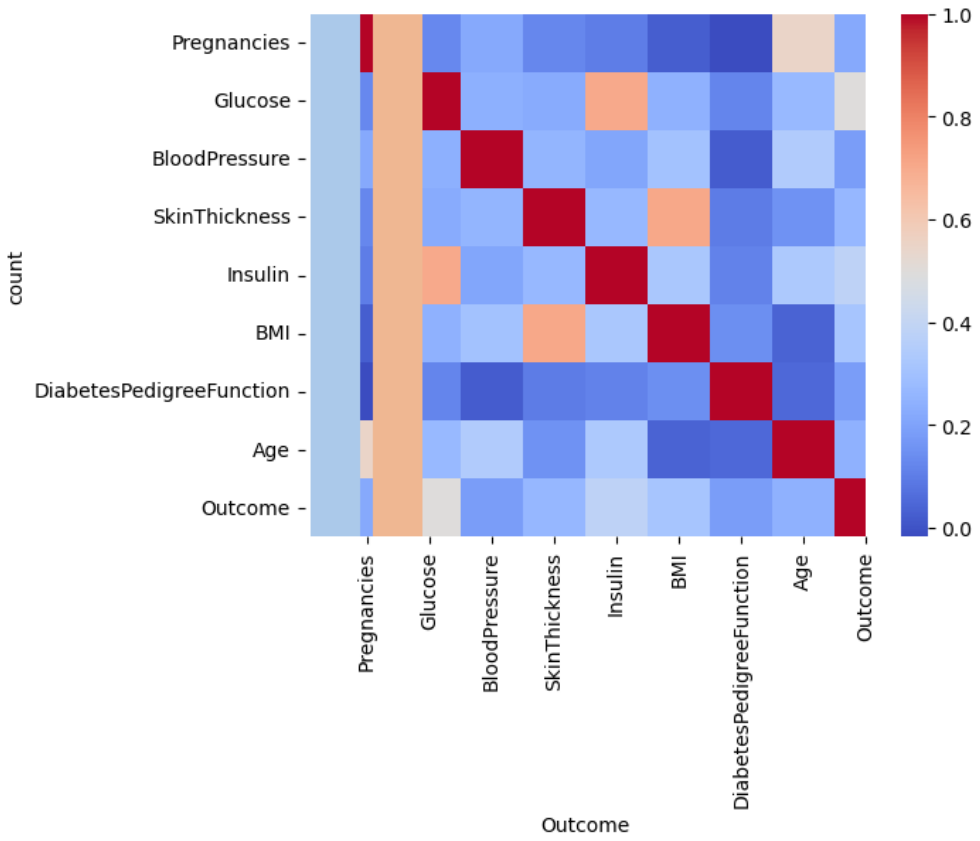
```
    return df
def clean_outlier_remove(col,outlier,df ):
    for i in outlier :
        ind =df.index[df[col]==i].tolist()
        df = df.drop(ind,axis=0)
    return df
```

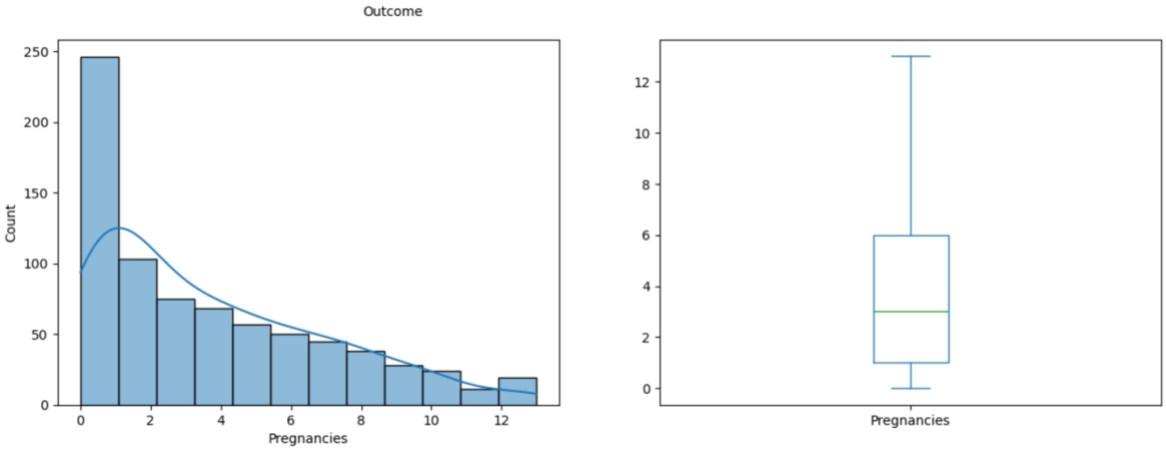
```
def impute_missing_values(data, cols,round=True):
    data[cols] = data[cols].replace(0, np.nan)

    # Apply MICE imputation
    rf = RandomForestRegressor(n_estimators=100, random_state=42)
    mice_imputer = IterativeImputer(estimator=rf,verbose=0,max_iter=1,tol=1e-
10,imputation_order='roman',random_state=0)
    data.iloc[:, :] = mice_imputer.fit_transform(data)

    #Round imputed SkinThickness values
    if round==True:
        data[cols] = data[cols].round(0)
    else:
        pass

    #Work with the cleaned data in memory
    print(data.head())
    return data
```

- **Outlier detection, removal, and visualization for the 'Pregnancies' column**

```
a = find_outlier('Pregnancies', data)
data= clean_outlier_remove('Pregnancies',a , data)

plt.figure(2)
plt.subplot(121) #histogram
sns.histplot(data['Pregnancies'], kde = True)

plt.subplot(122) #box plot
box_plot = data['Pregnancies'].plot.box(figsize=(15, 5))
plt.show()
```

Lower Bound: -6.5
Upper Bound: 13.5
Outliers:
[15, 17, 14]





- **Fill missing values in the 'Glucose' column**

```
data=impute_missing_values(data, 'Glucose',round=False)
```

```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6    148.0             72             35        0  33.6
1            1     85.0             66             29        0  26.6
2            8    183.0             64              0        0  23.3
3            1     89.0             66             23       94  28.1
4            0    137.0             40             35      168  43.1

   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
```

- **Quick overview (number of rows and columns)**

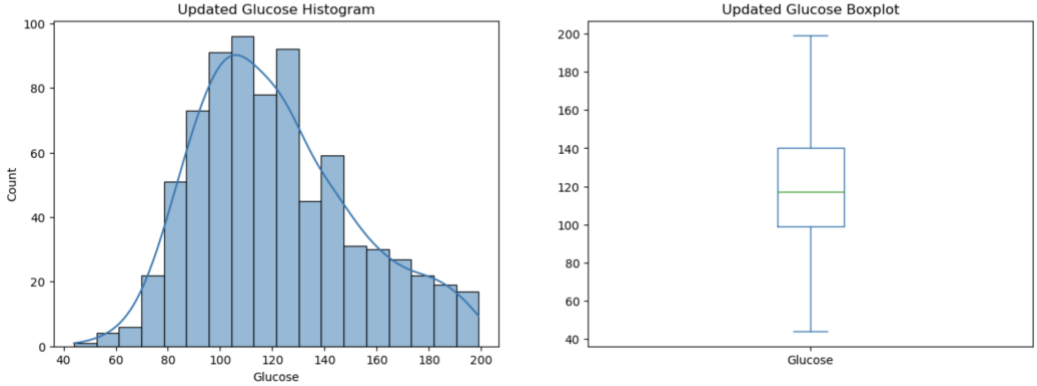
```
data.shape
```

(764, 9)

- **Outlier detection, removal, and visualization for the ‘Glucose’ column**

```
mean_glucose = 121.68

a = find_outlier('Glucose', data)
data = clean_outlier_remove('Glucose', a, data )

# Plot the updated histogram and boxplot
plt.figure(2)
# Histogram
plt.subplot(121)
sns.histplot(data['Glucose'], kde=True)
plt.title('Updated Glucose Histogram')

# Boxplot
plt.subplot(122)
data['Glucose'].plot.box(figsize=(15, 5))
plt.title('Updated Glucose Boxplot')

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

```
Lower Bound: 37.125
Upper Bound: 202.125
Outliers:
[]
```




```
data.shape
```

(764, 9)

- **Fill missing values in the ‘BloodPressure’ column**

```
data=impute_missing_values(data, 'BloodPressure',round=False)
```

```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6    148.0           72.0             35        0  33.6
1            1     85.0           66.0             29        0  26.6
2            8    183.0           64.0              0        0  23.3
3            1     89.0           66.0             23       94  28.1
4            0    137.0           40.0             35      168  43.1

   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
```

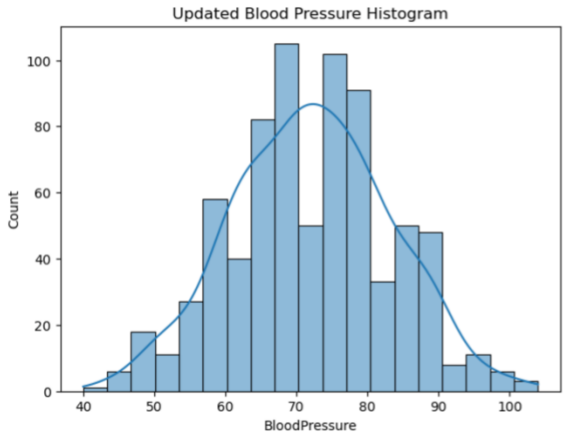
- **Outlier detection and removal for the ‘BloodPressure’ column**

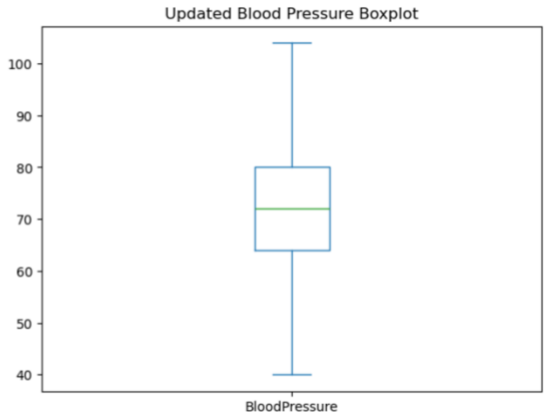
```
a = find_outlier('BloodPressure', data)
a.sort()
print(a)
data = clean_outlier_remove('BloodPressure', a[3:], data)
data = clean_outlier_remove('BloodPressure', a[0:3], data)
```

Lower Bound: 40.0
Upper Bound: 104.0
Outliers:
[30.0, 110.0, 108.0, 122.0, 24.0, 38.0, 106.0, 114.0]
[24.0, 30.0, 38.0, 106.0, 108.0, 110.0, 114.0, 122.0]

- **Visualizing the distribution of the ‘BloodPressure’**

```
mean_bp = 69.1
median_bp = 72  # Optional: Use this if you prefer median

# Plot the updated histogram and boxplot
plt.figure(2)

# Histogram
plt.subplot(121)
sns.histplot(data['BloodPressure'], kde=True)
plt.title('Updated Blood Pressure Histogram')

# Boxplot
plt.subplot(122)
data['BloodPressure'].plot.box(figsize=(15, 5))
plt.title('Updated Blood Pressure Boxplot')

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






```
data.shape
```

(750, 9)

- **Outlier detection, removal, and visualization for the ‘SkinThickness’ column**

```
a= find_outlier('SkinThickness', data)
```

Lower Bound: -48.0
Upper Bound: 80.0
Outliers:
[99]

```
# Before imputation
print("Before Cleaning:")
print(data['SkinThickness'].describe())
```

Before Cleaning:

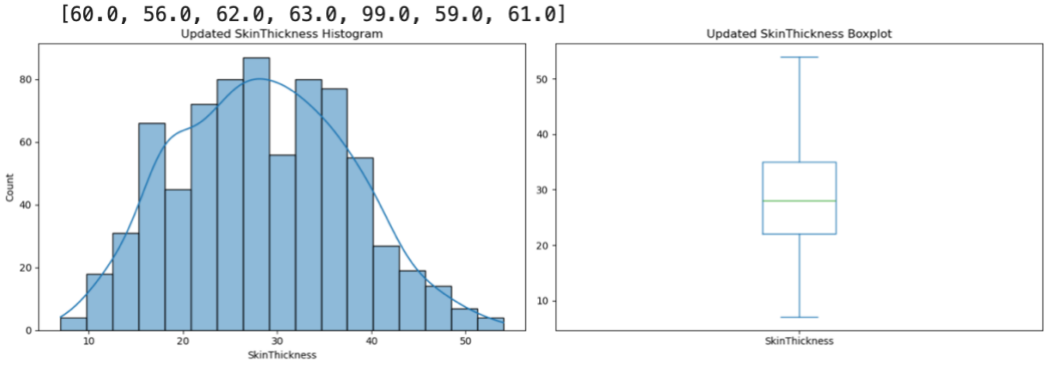
count	750.000000
mean	20.426667
std	15.941962
min	0.000000
25%	0.000000
50%	23.000000
75%	32.000000
max	99.000000

Name: SkinThickness, dtype: float64

```
data=impute_missing_values(data,'SkinThickness')
a = find_outlier('SkinThickness', data)
data = clean_outlier_remove('SkinThickness', a, data)
# Plot the updated histogram and boxplot
plt.figure(figsize=(16, 6))

# Histogram
```

```
plt.subplot(121)
sns.histplot(data['SkinThickness'], kde=True)
plt.title('Updated SkinThickness Histogram')

# Boxplot
plt.subplot(122)
data['SkinThickness'].plot.box(figsize=(15, 5))
plt.title('Updated SkinThickness Boxplot')

# Show the plots
plt.tight_layout()
plt.show()
```

```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6    148.0           72.0           35.0        0  33.6
1            1     85.0           66.0           29.0        0  26.6
2            8    183.0           64.0           19.0        0  23.3
3            1     89.0           66.0           23.0       94  28.1
4            0    137.0           40.0           35.0      168  43.1

   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
Lower Bound: 2.5
Upper Bound: 54.5
Outliers:
[60.0, 56.0, 62.0, 63.0, 99.0, 59.0, 61.0]
```




```
# After imputation
print("\nAfter Cleaning:")
print(data['SkinThickness'].describe())
```

After Cleaning:

count	742.000000
mean	28.497305
std	9.089115
min	7.000000
25%	22.000000
50%	28.000000
75%	35.000000
max	54.000000

Name: SkinThickness, dtype: float64

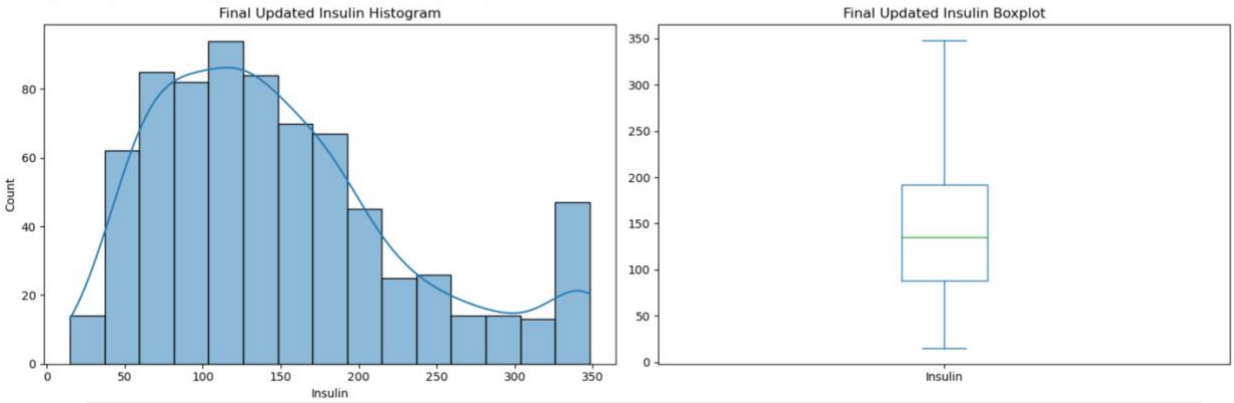
- **Outlier detection, removal, and visualization for the 'Insulin' column**

```
data=impute_missing_values(data,'Insulin',False)
```

```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6    148.0           72.0           35.0   193.32  33.6
1            1     85.0           66.0           29.0    60.82  26.6
2            8    183.0           64.0           19.0   229.85  23.3
3            1     89.0           66.0           23.0    94.00  28.1
4            0    137.0           40.0           35.0   168.00  43.1

   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
```

```
a = find_outlier('Insulin', data)
data.reset_index(drop=True, inplace=True)
data = clean_outlier('Insulin', a, data, 348)
 # Reset the index after cleaning outliers

# Histogram
plt.subplot(121)
sns.histplot(data['Insulin'], kde=True)
plt.title('Final Updated Insulin Histogram')

# Boxplot
plt.subplot(122)
data['Insulin'].plot.box(figsize=(15, 5))
plt.title('Final Updated Insulin Boxplot')

# Show plots
plt.tight_layout()
plt.show()
```

```
Lower Bound: -68.0
Upper Bound: 348.0
Outliers:
[543.0, 846.0, 495.0, 485.0, 349.09, 416.37, 478.0, 395.66, 744.0, 370.0, 680.0, 402.0, 375.0, 545.0, 405.81, 360.0, 540.35, 465.0, 364.55, 415.0, 579.0, 474.0, 480.0,
462.35, 359.62, 391.51, 600.0, 350.02, 440.0, 540.0, 392.78, 387.0, 392.0, 510.0]
```




```
data.shape
```

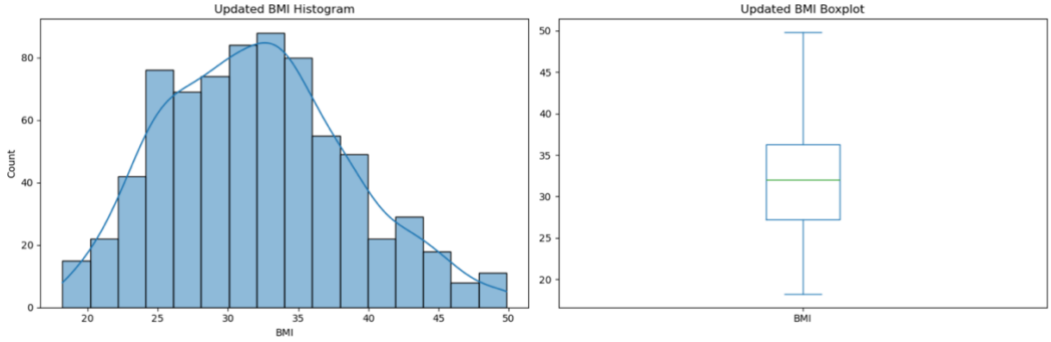
(742, 9)

- **Outlier removal, and visualization for the 'BMI' column**

```
data=impute_missing_values(data,'BMI',False)
a = find_outlier('BMI', data)
data = clean_outlier('BMI', a, data, 49.84 )
# Clean the specific outlier (0) using your clean_outlier function
#verify the cleaning
```

```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6    148.0           72.0           35.0   193.32  33.6
1            1     85.0           66.0           29.0    60.82  26.6
2            8    183.0           64.0           19.0   229.85  23.3
3            1     89.0           66.0           23.0    94.00  28.1
4            0    137.0           40.0           35.0   168.00  43.1

   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
Lower Bound: 13.650000000000006
Upper Bound: 49.849999999999994
Outliers:
[50.0, 52.3, 52.9, 57.3]
```

```
# Define mean and median values
mean_bmi = 31.99
median_bmi = 32

# Plot the updated histogram and boxplot
plt.figure(figsize=(16, 6))

# Histogram
plt.subplot(121)
sns.histplot(data['BMI'], kde=True)
plt.title('Updated BMI Histogram')

# Boxplot
plt.subplot(122)
data['BMI'].plot.box(figsize=(15, 5))
plt.title('Updated BMI Boxplot')

# Show the plots
plt.tight_layout()
plt.show()
```



```
data.shape
```

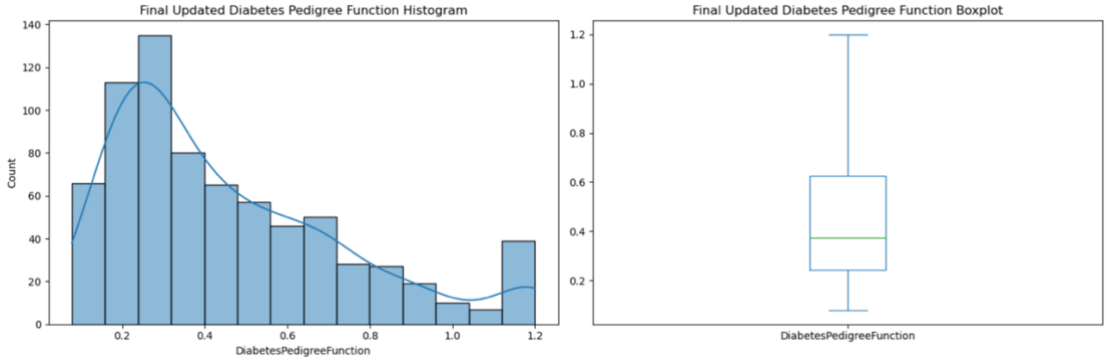
(742, 9)

- **Outlier detection, removal, and visualization for the 'DiabetesPedigreeFunction' column**

```
a = find_outlier('DiabetesPedigreeFunction', data)
a.sort()
print(a)
print('Mode: ',data['DiabetesPedigreeFunction'].mode())
data = clean_outlier('DiabetesPedigreeFunction', a, data, 1.2)
```

Lower Bound: -0.3295000000000001
Upper Bound: 1.2005000000000001
Outliers:
[2.288, 1.441, 1.39, 1.893, 1.781, 1.222, 1.4, 1.321, 1.224, 2.329, 1.318, 1.213, 1.353, 1.391, 1.476, 2.137, 1.731, 1.268, 1.6, 1.251, 1.699, 1.258, 1.282, 1.698, 1.461, 1.292, 1.394]
[1.213, 1.222, 1.224, 1.251, 1.258, 1.268, 1.282, 1.292, 1.318, 1.321, 1.353, 1.39, 1.391, 1.394, 1.4, 1.441, 1.461, 1.476, 1.6, 1.698, 1.699, 1.731, 1.781, 1.893, 2.137, 2.288, 2.329]
Mode: 0 0.254
1 0.258
Name: DiabetesPedigreeFunction, dtype: float64

```
# Plot the updated histogram and boxplot
plt.figure(figsize=(16, 6))

# Histogram
plt.subplot(121)
sns.histplot(data['DiabetesPedigreeFunction'], kde=True)
plt.title('Final Updated Diabetes Pedigree Function Histogram')

# Boxplot
plt.subplot(122)
data['DiabetesPedigreeFunction'].plot.box(figsize=(15, 5))
plt.title('Final Updated Diabetes Pedigree Function Boxplot')

# Show the plots
plt.tight_layout()
plt.show()
```

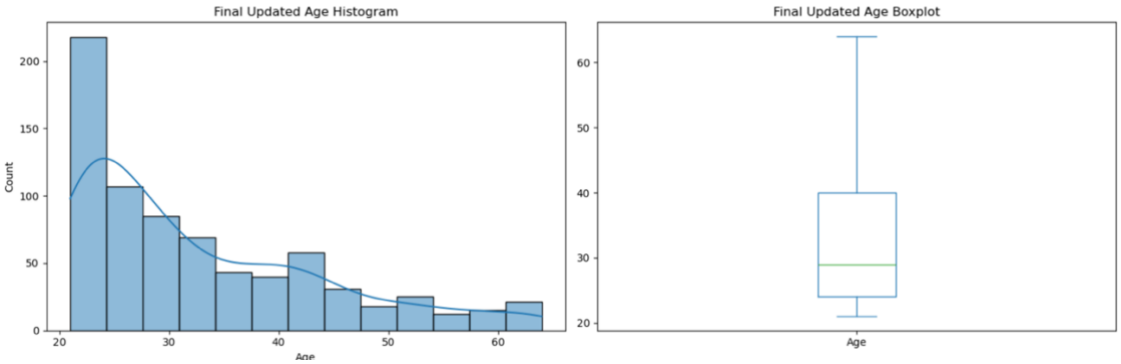


- **Outlier detection, removal, and visualization for the ‘Age’ column**

```
a = find_outlier('Age', data)
data = clean_outlier('Age', a, data, 64 )
```

Lower Bound: 0.0
Upper Bound: 64.0
Outliers:
[69, 66, 65, 72, 81, 67, 70]

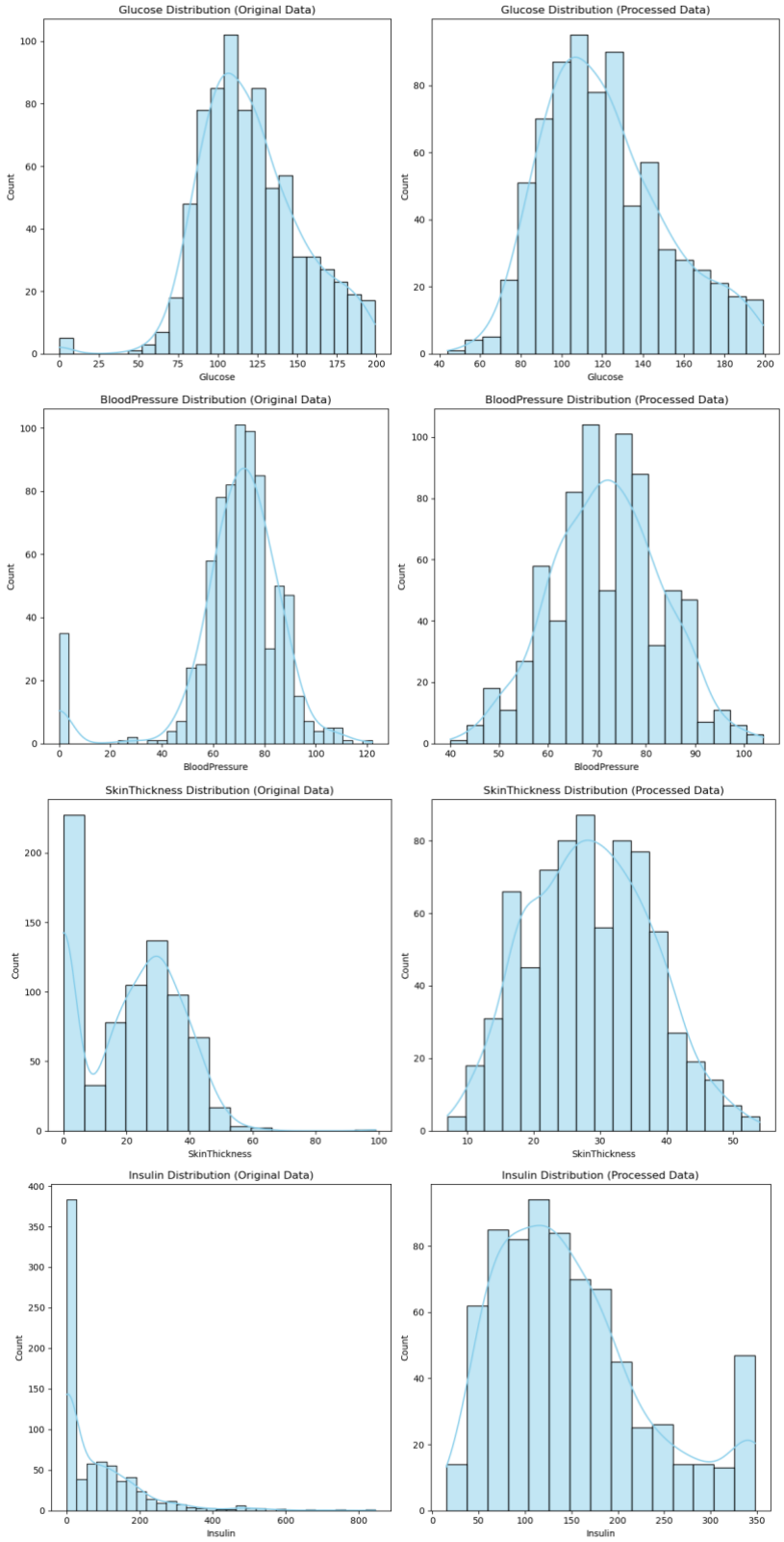
```
# Plot the updated histogram and boxplot
plt.figure(figsize=(16, 6))

# Histogram
plt.subplot(121)
sns.histplot(data['Age'], kde=True)
plt.title('Final Updated Age Histogram')

# Boxplot
plt.subplot(122)
data['Age'].plot.box(figsize=(15, 5))
plt.title('Final Updated Age Boxplot')

# Show the plots
plt.tight_layout()
plt.show()
```

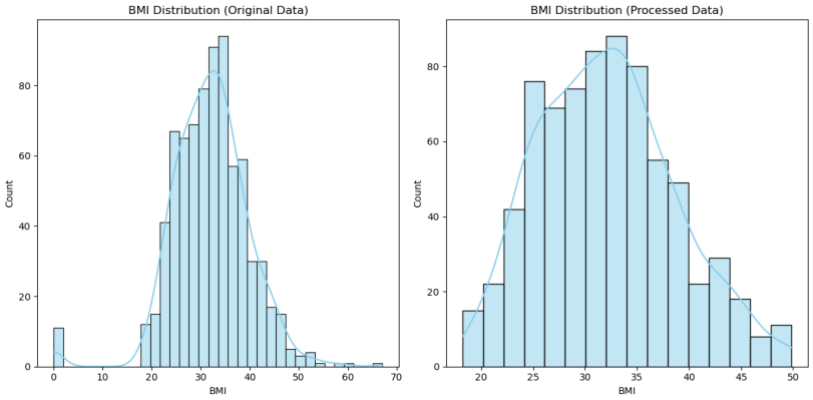


- **Generate correlation for each variable**

```
cols = ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI',
'DiabetesPedigreeFunction', 'Age','Pregnancies']
for i in cols:
    plt.figure(figsize=(12, 6))  # Set the figure size for each column

    # Plot the first distribution
    plt.subplot(121)
    plt.title(f'{i} Distribution (Original Data)')
    sns.histplot(df[i], color='skyblue', kde=True)

    # Plot the second distribution
    plt.subplot(122)
    plt.title(f'{i} Distribution (Processed Data)')
```

```
    sns.histplot(data[i], color='skyblue', kde=True)

    plt.tight_layout()  # Ensure the plots fit nicely within the figure
    plt.show()
```






```
data.shape
```

(742, 9)

3.2. Modeling and Training

After carefully cleaning and preparing the dataset, we convert it into a CSV file "proceed_data.csv". Then, we move on to the main goal of the project, which is to use cutting-edge machine learning algorithms to find significant patterns in the data. This phase is essential because it allows us to estimate diabetes risk with accuracy and dependability, paving the way for impactful insights and potential real-world applications. Here's the process of build the model:

- **Load necessary libraries**

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
import seaborn as sns
import scipy.stats as stat
import pylab
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from imblearn.over_sampling import RandomOverSampler
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB,MultinomialNB
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import accuracy_score, classification_report
from sklearn import metrics
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.preprocessing import MinMaxScaler
```

```
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense, Dropout
from plotly.subplots import make_subplots
import plotly.graph_objects as go
from math import ceil
import plotly.express as px
```

- **Load the cleaned dataset**

```
df = pd.read_csv('processed_data.csv')
```

- **Inspect the first rows of the dataset**

```
df.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148.0	72.0	35	193.32	33.6	0.627	50	1
1	1	85.0	66.0	29	60.82	26.6	0.351	31	0
2	8	183.0	64.0	19	229.85	23.3	0.672	32	1
3	1	89.0	66.0	23	94.00	28.1	0.167	21	0
4	0	137.0	40.0	35	168.00	43.1	1.200	33	1

- **Quick overview of the DataFrame's structure, data types**

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 742 entries, 0 to 741
Data columns (total 9 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   Pregnancies              742 non-null    int64  
 1   Glucose                  742 non-null    float64
 2   BloodPressure            742 non-null    float64
 3   SkinThickness            742 non-null    int64  
 4   Insulin                  742 non-null    float64
 5   BMI                      742 non-null    float64
 6   DiabetesPedigreeFunction 742 non-null    float64
 7   Age                      742 non-null    int64  
 8   Outcome                  742 non-null    int64  
dtypes: float64(5), int64(4)
memory usage: 52.3 KB
```

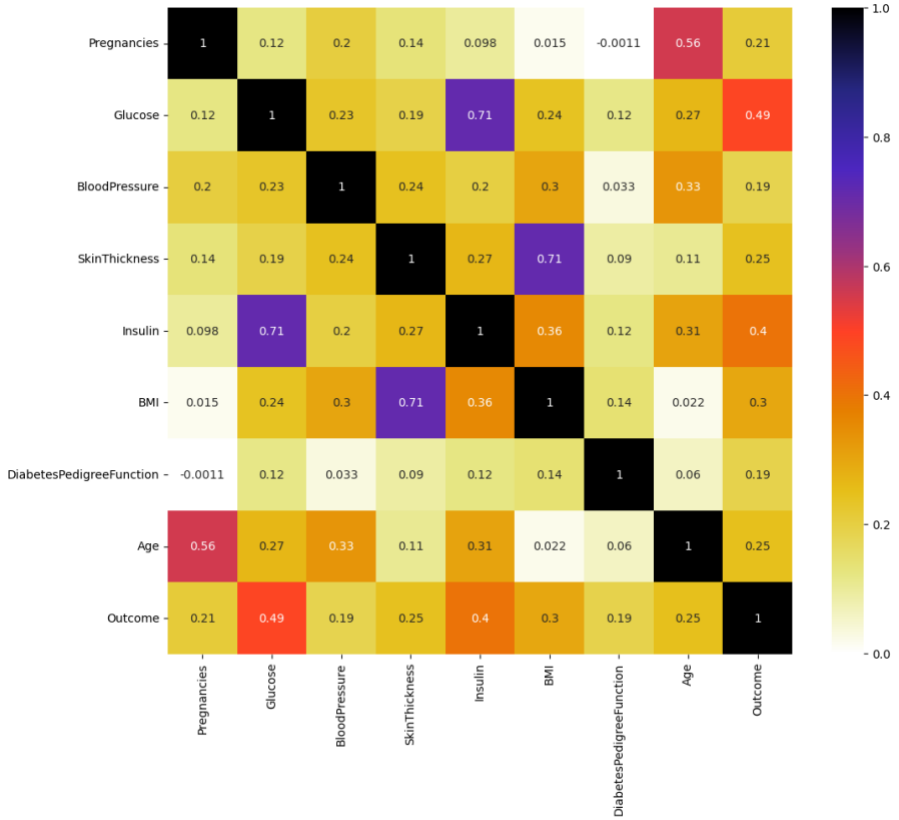
- **Check for duplicate row in the DataFrame**

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

```
0
```

- **Visualizing the correlation matrix of the DataFrame**

```
plt.figure(figsize=(12,10))
corr = df.corr()
sns.heatmap(corr,annot = True,cmap=plt.cm.CMRmap_r)
plt.show()
```



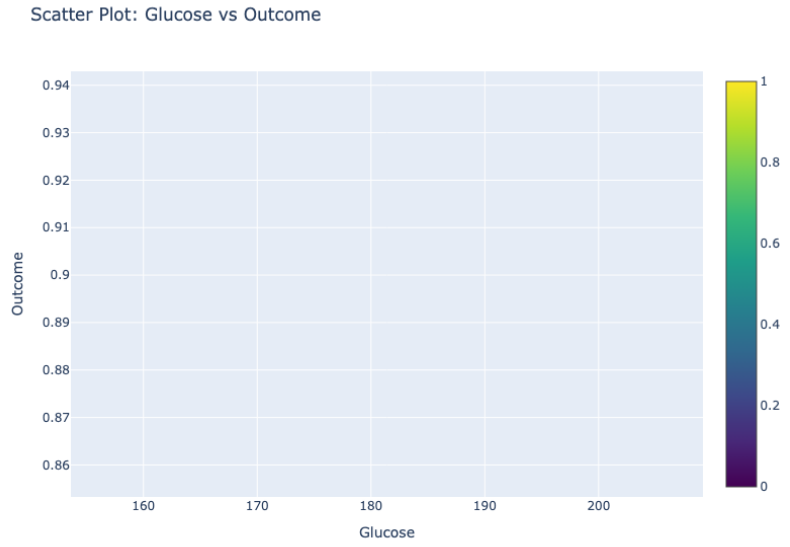
- **Feature analysis or Feature selection**

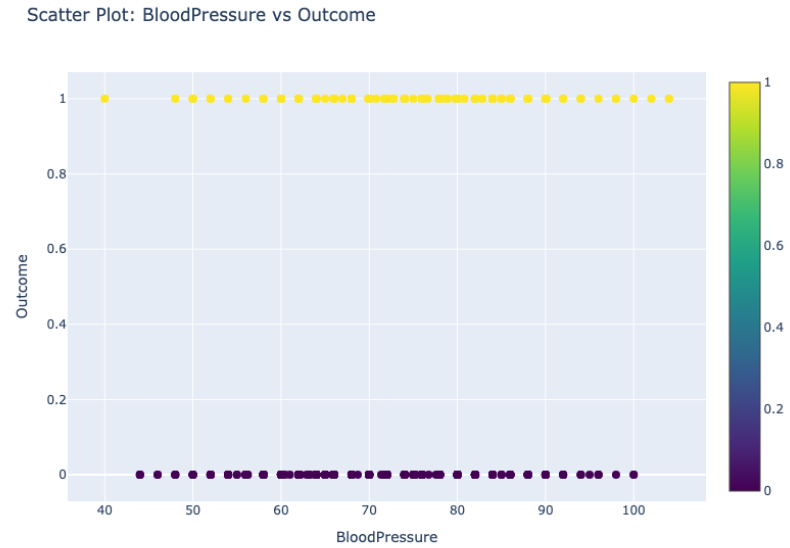
```
cols=['Glucose','BloodPressure','Insulin','BMI','SkinThickness','DiabetesPedigree
Function','Age','Pregnancies','Outcome',]
```

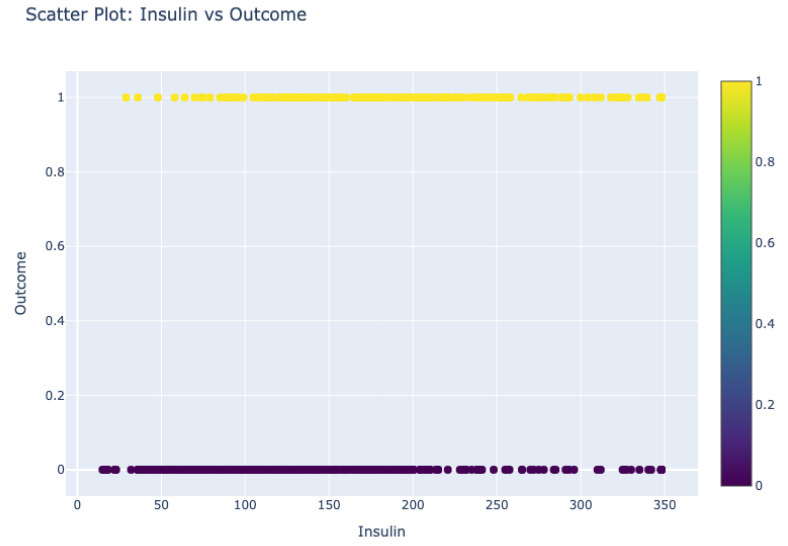
- **Visualizing how each feature correlates with the Outcome**

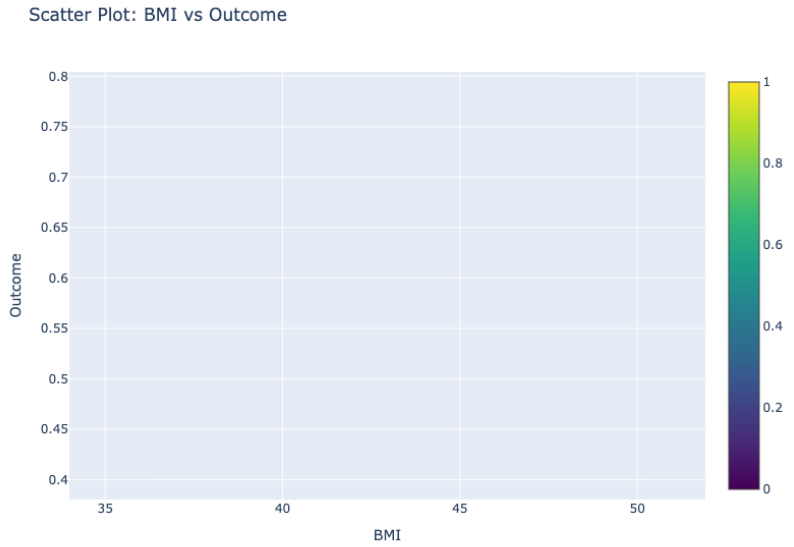
```
import plotly.graph_objects as go

# Loop through each feature in cols[:-1]
for label in cols[:-1]:
    if label not in df.columns:
        print(f"Column '{label}' not found in the DataFrame. Skipping...")
        continue

    fig = go.Figure()

    # Add scatter plot for the current feature vs. 'Outcome'
    fig.add_trace(
        go.Scatter(
            x=df[label],
            y=df['Outcome'],
            mode='markers',
            marker=dict(
                size=8,
                color=df['Outcome'],  # Color points based on 'Outcome' values
                colorscale='Viridis',  # Color scale
                showscale=True        # Show color scale legend
            ),
            name=label
```

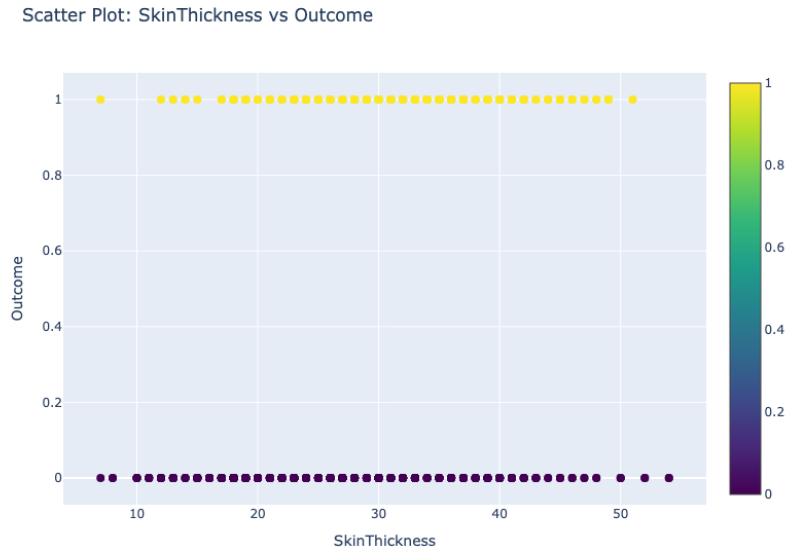
```
    )
)

# Update the layout for the current feature
fig.update_layout(
    title=f"Scatter Plot: {label} vs Outcome",
    xaxis_title=label,
    yaxis_title="Outcome",
    height=600,
    width=800
)

# Display the plot
fig.show()
```

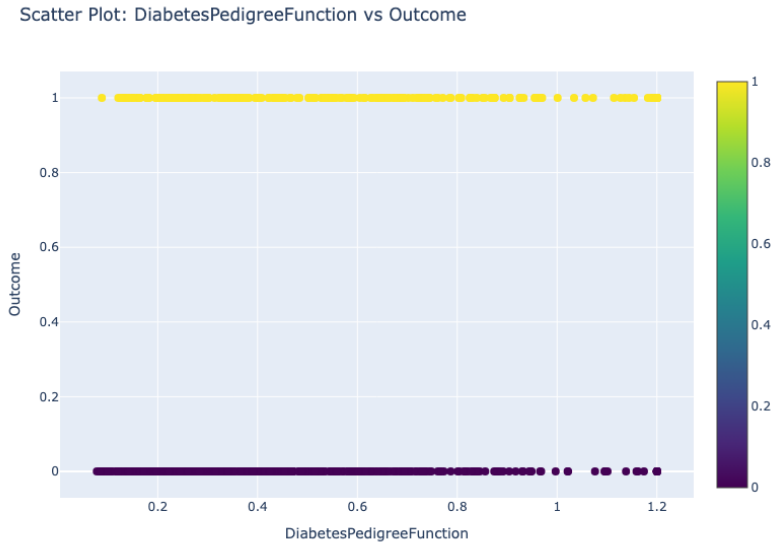


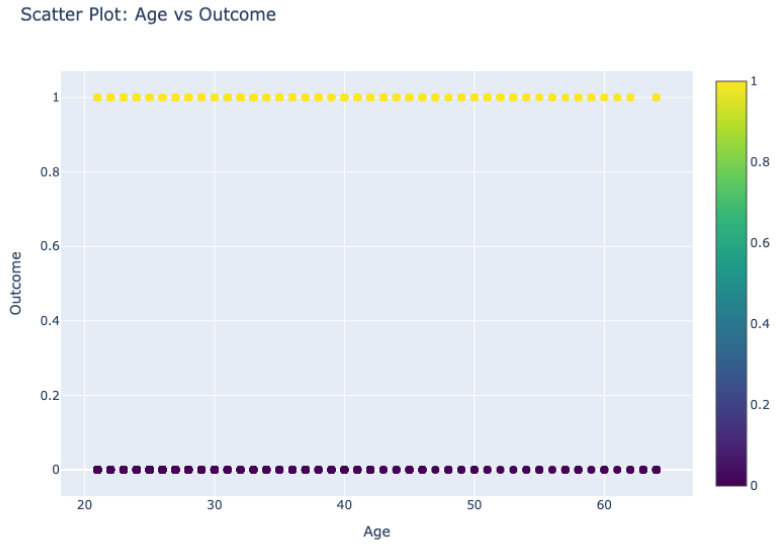


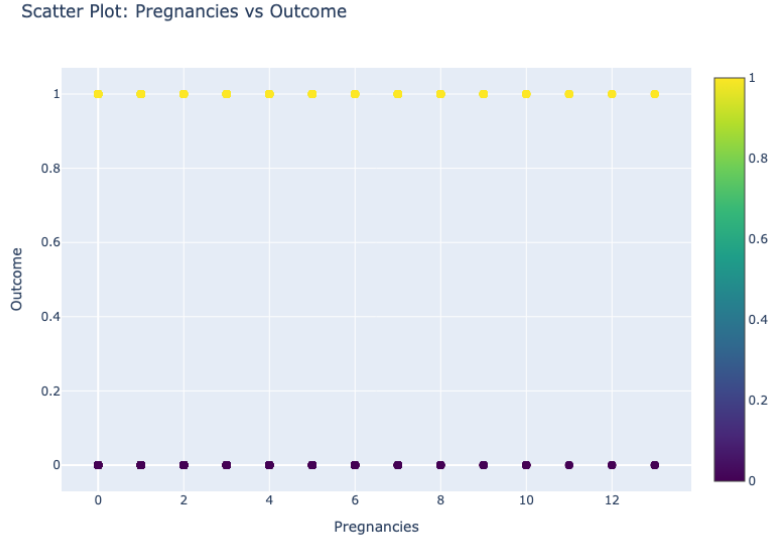

Scatter Plot: Insulin vs Outcome
Outcome
Insulin
0
0.2
0.4
0.6
0.8
1
0
50
100
150
200
250
300
350


Scatter Plot: BMI vs Outcome
Outcome
BMI
0.4
0.45
0.5
0.55
0.6
0.65
0.7
0.75
0.8
35
40
45
50


Scatter Plot: SkinThickness vs Outcome
Outcome
SkinThickness
0
0.2
0.4
0.6
0.8
1
10
20
30
40
50


Scatter Plot: DiabetesPedigreeFunction vs Outcome
Outcome
DiabetesPedigreeFunction
0
0.2
0.4
0.6
0.8
1
1.2


Scatter Plot: Age vs Outcome
Outcome
Age
0
0.2
0.4
0.6
0.8
1
20
30
40
50
60


Scatter Plot: Pregnancies vs Outcome
Outcome
Pregnancies
0
0.2
0.4
0.6
0.8
1
0
2
4
6
8
10
12

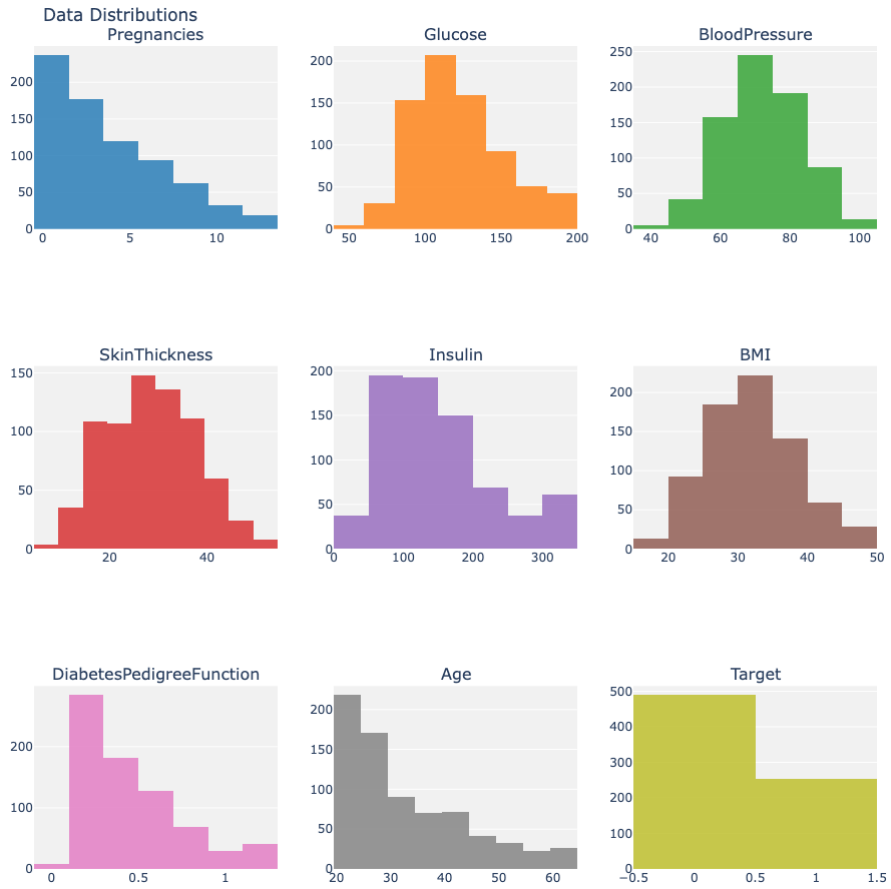
- **Global variables**

```
# Global variables
features = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness',
        'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age']
target = 'Outcome'
```

- **Visualizing the distributions of each feature and the target ‘Outcome’**

```
# Create a 3x3 grid of subplots
fig = make_subplots(rows=3, cols=3, subplot_titles=features + ['Target'])

# Define a color palette
colors = [
   'rgba(31, 119, 180, 0.8)', 'rgba(255, 127, 14, 0.8)', 'rgba(44, 160, 44, 0.8)',
   'rgba(214, 39, 40, 0.8)', 'rgba(148, 103, 189, 0.8)', 'rgba(140, 86, 75, 0.8)',
   'rgba(227, 119, 194, 0.8)', 'rgba(127, 127, 127, 0.8)', 'rgba(188, 189, 34, 0.8)'
]

# Add histograms for each feature
for i, feature in enumerate(features):
   row, col = (i // 3) + 1, (i % 3) + 1
   fig.add_trace(
      go.Histogram(
         x=df[feature],
         name=feature,
         marker_color=colors[i],
         hoverinfo="x+y",
         nbinsx=10
      ),
      row=row,
      col=col
   )

# Add target distribution
fig.add_trace(
   go.Histogram(
      x=df[target],
      name=target,
      marker_color=colors[-1],
      hoverinfo="x+y",
      nbinsx=2
   ),
   row=3,
   col=3
)

# Update layout for better appearance
fig.update_layout(
   height=900,
   width=900,
```

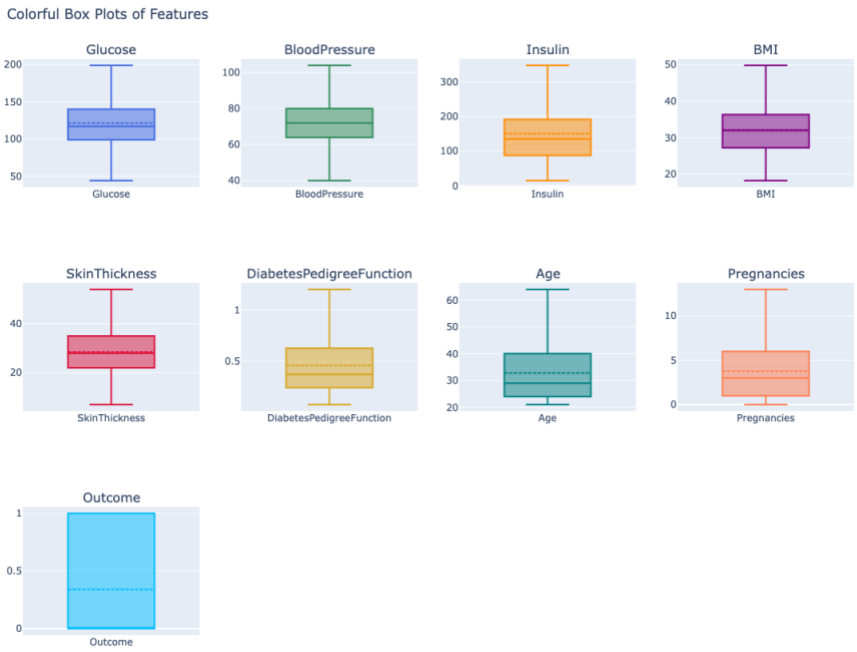
```
    title_text="Data Distributions",
    showlegend=False,
    plot_bgcolor='rgba(240, 240, 240, 0.9)',  # Light grey background
    margin=dict(t=50, l=20, r=20, b=20)
)

# Show the interactive figure
fig.show()
```



- **Visualizing the distribution of each feature, including the spread, median, and potential outliers**

```
# Define a list of colors for the box plots
colors = [
    "royalblue", "seagreen", "darkorange", "purple",
    "crimson", "goldenrod", "teal", "coral", "deepskyblue"
]

# Ensure the colors list is long enough for all features
colors = (colors * (len(cols) // len(colors) + 1))[:len(cols)]

# Determine the number of rows and columns based on the number of features
num_features = len(cols)
num_cols = 4  # Fixed number of columns
```

```
num_rows = ceil(num_features / num_cols)

# Create a subplot grid with the calculated number of rows and columns
fig = make_subplots(rows=num_rows, cols=num_cols, subplot_titles=cols)

# Loop through each feature and add a box plot with its color
for i, feature in enumerate(cols):
    row, col = (i // num_cols) + 1, (i % num_cols) + 1  # Calculate row and column index
    fig.add_trace(
        go.Box(
            y=df[feature],
            name=feature,
            boxmean=True,
            marker_color=colors[i]
        ),
        row=row,
        col=col
    )

# Update layout for the entire figure
fig.update_layout(
    height=300 * num_rows,  # Adjust height dynamically
    width=1200,          # Fixed width
    title_text="Colorful Box Plots of Features",
    showlegend=False
)

# Display the figure
fig.show()
```



- **Handle outliers**

```
def clean_outlier_re(col,outlier,df,value ):
    for i in outlier :
        ind =df.index[df[col]==i].tolist()
        for j in ind :
            df.loc[j, col] = value
    return df
```

```
def clean_outlier_drop(col,outlier,df ):
    for i in outlier :
        ind =df.index[df[col]==i].tolist()
        df=df.drop(index=ind,axis=0)
    return df
```

```
def find_outlier(cols,data):#outlier
    Q1 = data[cols].quantile(0.25)
    Q3 = data[cols].quantile(0.75)
    IQR = Q3 - Q1

    # Calculate bounds
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    # Find outliers
    outliers_list = data[(data[cols] < lower_bound) | (data[cols] >
upper_bound)][cols].unique().tolist()
    return outliers_list,lower_bound,upper_bound
```

```
for i in cols[:-1]:
    outlier,l,u = find_outlier(i,df)
    print(i," :",outlier)
```

```
Glucose  : []
BloodPressure  : []
Insulin  : []
BMI  : []
SkinThickness  : []
DiabetesPedigreeFunction  : []
Age  : []
Pregnancies  : []
```

```
df.shape
```

```
(742, 9)
```

- **Function to visualize the distribution of a feature in the dataset**

```
def plot_data(df,feature):
    plt.figure(figsize=(10,5))
```

```
    plt.subplot(1,2,1)
    df[feature].hist()
    plt.title('Histogram of '+feature)
    plt.subplot(1,2,2)
    stat.probplot(df[feature],dist='norm',plot=pylab)
    plt.title('Probability Plot of '+feature)
    plt.show()
```

- **Make a deep copy**

```
data = df.copy()
```

- **Inspect the first rows of the dataset again**

```
df.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148.0	72.0	35	193.32	33.6	0.627	50	1
1	1	85.0	66.0	29	60.82	26.6	0.351	31	0
2	8	183.0	64.0	19	229.85	23.3	0.672	32	1
3	1	89.0	66.0	23	94.00	28.1	0.167	21	0
4	0	137.0	40.0	35	168.00	43.1	1.200	33	1

- **Load necessary libraries for handling imbalanced the datasets**

```
from imblearn.over_sampling import SMOTE
from imblearn.over_sampling import ADASYN
from imblearn.under_sampling import RandomUnderSampler
from imblearn.under_sampling import TomekLinks
from imblearn.under_sampling import ClusterCentroids
from imblearn.combine import SMOTEENN
from imblearn.combine import SMOTETomek
```

- **Function for scaling the features**

```
def scale(df,oversample=False):
    x = df[df.columns[:-1]].values
    y = df[df.columns[-1]].values
    scaler = StandardScaler()
    x = scaler.fit_transform(x)
    scaler = MinMaxScaler()
    x = scaler.fit_transform(x)
    if oversample:

        # ros = TomekLinks()
        # ros = ADASYN()
        # ros = RandomOverSampler(random_state=41)
        # x, y = ros.fit_resample(x, y)
        ros=SMOTEENN(random_state=41)
        # ros=SMOTE()
```

```
    x, y = ros.fit_resample(x,y)
  #data = np.hstack([x,np.reshape(y,(-1,1))])
  return x, y
```

- **Remove unnecessary features**

```
reduced_features = ['BloodPressure', 'DiabetesPedigreeFunction', 'Insulin',
'SkinThickness']  # Keep these
df = df.drop(columns=reduced_features, axis=1)
```

- **Count the occurrences of the 'Outcome'**

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

```
Outcome
0    489
1    253
Name: count, dtype: int64
```

- **Check the number of rows and columns**

```
df.shape
```

```
(742, 5)
```

- **Check for missing or null values**

```
df.isnull().sum()
```

```
Pregnancies    0
Glucose        0
BMI            0
Age            0
Outcome        0
dtype: int64
```

- **Inspect the first rows**

```
df.head()
```

	Pregnancies	Glucose	BMI	Age	Outcome
0	6	148.0	33.6	50	1
1	1	85.0	26.6	31	0
2	8	183.0	23.3	32	1
3	1	89.0	28.1	21	0
4	0	137.0	43.1	33	1

- **Split data**

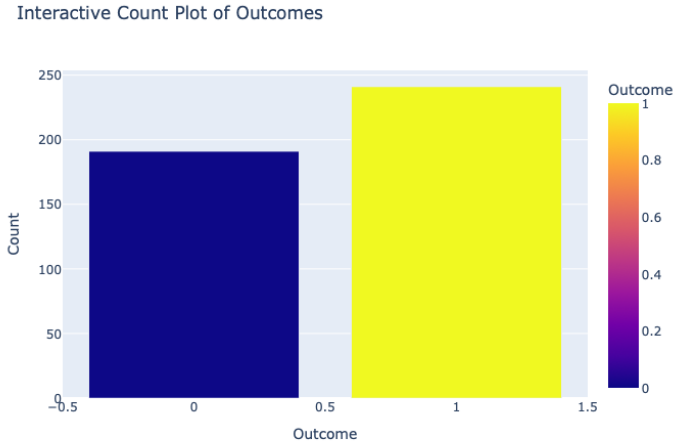
```
train,test = train_test_split(df,test_size=0.20)
dtrain,dtest = train_test_split(data,test_size=0.15)
```

```
train['Outcome'].value_counts()
```

```
Outcome
0    390
1    203
Name: count, dtype: int64
```

```
xtrain, ytrain = scale(dtrain, True)
xtest, ytest = scale(dtest, False)
x_train,y_train = scale(train,True)
x_test,y_test = scale(test,False)
```

```
outcome_counts = y_train_df['Outcome'].value_counts().reset_index()

outcome_counts.columns = ['Outcome', 'Count']
# Create an interactive bar plot
fig = px.bar(
    outcome_counts,  # Use the corrected DataFrame
    x='Outcome',
    y='Count',
    color='Outcome',  # Color by outcome
    labels={'Outcome': 'Outcome', 'Count': 'Count'},  # Customize axis labels
    color_discrete_sequence=px.colors.qualitative.Pastel,  # Pastel color palette
    title="Interactive Count Plot of Outcomes"
)

# Customize the layout
fig.update_layout(
    xaxis_title="Outcome",
    yaxis_title="Count",
    showlegend=False,  # Hide legend (optional)
    height=500,
    width=700
)

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



- **Deep learning**

```
def plot_history(history):
  plt.plot(history.history['loss'],label='loss')
  plt.plot(history.history['val_loss'],label='val_loss')
  plt.xlabel('Epoch')
  plt.ylabel('Binary Crossentropy')
  plt.legend()
  plt.grid(True)
  plt.show()
```

```
dl_model = Sequential()
dl_model.add(Dense(256,  activation = 'relu' ,input_shape=([8]))) #input layer
dl_model.add(Dense(256,  activation = 'relu'))
dl_model.add(Dense(1,activation = 'sigmoid'))
dl_model.summary()
dl_model.compile(optimizer = 'adam' , loss = 'binary_crossentropy' ,metrics =
['accuracy','Precision','Recall','AUC'])
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
dense_3 (Dense)	(None, 256)	2,304
dense_4 (Dense)	(None, 256)	65,792
dense_5 (Dense)	(None, 1)	257

Total params: 68,353 (267.00 KB)

Trainable params: 68,353 (267.00 KB)

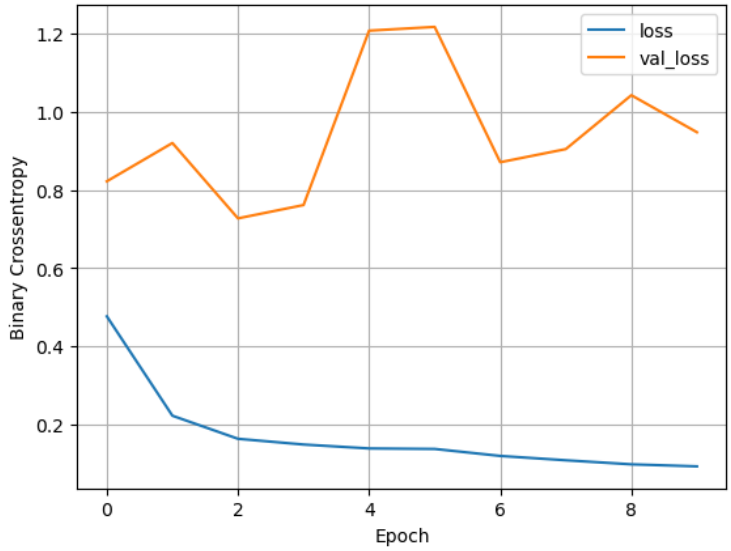
Non-trainable params: 0 (0.00 B)

```
num_epochs = 10
history = dl_model.fit(xtrain,
            ytrain,
            epochs=num_epochs,
            steps_per_epoch=200,
            validation_data=(xtest, ytest),
            verbose=0)
```

```
dl_model.evaluate(xtest,ytest)
```

4/4 ━━━━━━━━━━━━━━━━━━━━ **0s** 10ms/step - AUC: 0.7994 - Precisio
n: 0.5018 - Recall: 0.6459 - accuracy: 0.7500 - loss: 0.9313
[0.9472659826278687,
0.75,
0.5428571701049805,
0.6129032373428345,
0.786141037940979]

```
plot_history(history)
```



- **Build model**

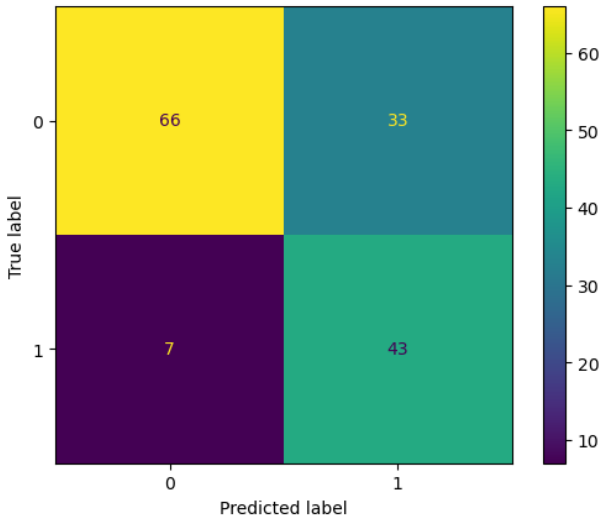
Logistic Regression

```
lg_model = LogisticRegression()
lg_model = lg_model.fit(x_train,y_train)
```

```
y_pre = lg_model.predict(x_test)
print(classification_report(y_test,y_pre))
```

```
              precision    recall  f1-score   support

           0       0.90      0.67      0.77        99
           1       0.57      0.86      0.68        50

    accuracy                           0.73       149
   macro avg       0.73      0.76      0.72       149
weighted avg       0.79      0.73      0.74       149
```

```
confusion_matrix = metrics.confusion_matrix(y_test, y_pre)

cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix =
confusion_matrix, display_labels = [0, 1])

cm_display.plot()
plt.show()
```



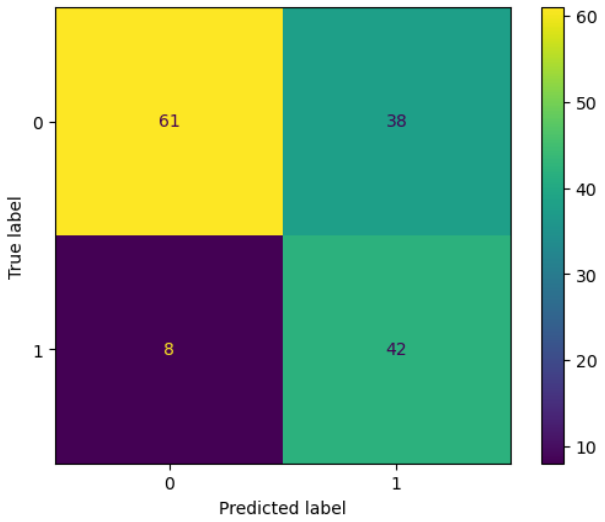
Supporting Vector Machine

```
svm_model = SVC()
svm_model = svm_model.fit(x_train,y_train)
```

```
y_pre = svm_model.predict(x_test)
print(classification_report(y_test,y_pre))
```

```
              precision    recall  f1-score   support

           0       0.88      0.62      0.73        99
           1       0.53      0.84      0.65        50

    accuracy                           0.69       149
   macro avg       0.70      0.73      0.69       149
weighted avg       0.76      0.69      0.70       149
```

```
confusion_matrix = metrics.confusion_matrix(y_test, y_pre)

cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix =
confusion_matrix, display_labels = [0, 1])

cm_display.plot()
plt.show()
```



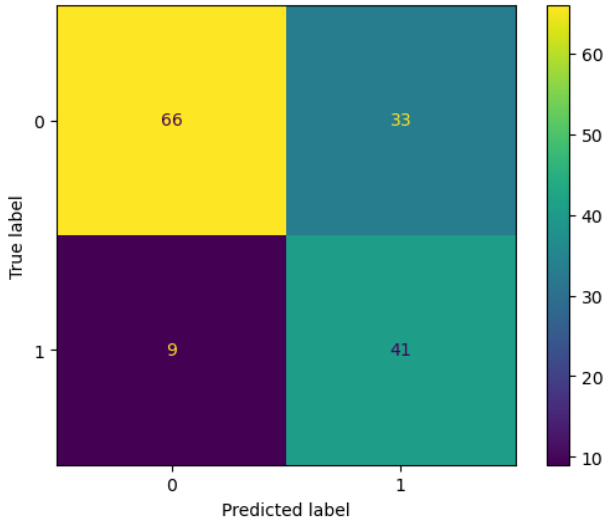
Gaussian Naive Bays

```
nb_model = GaussianNB()
nb_model = nb_model.fit(x_train,y_train)
```

```
y_pre = nb_model.predict(x_test)
print(classification_report(y_test,y_pre))
```

```
              precision    recall  f1-score   support

           0       0.88      0.67      0.76        99
           1       0.55      0.82      0.66        50

    accuracy                           0.72       149
   macro avg       0.72      0.74      0.71       149
weighted avg       0.77      0.72      0.73       149
```

```
confusion_matrix = metrics.confusion_matrix(y_test, y_pre)

cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix =
confusion_matrix, display_labels = [0, 1])

cm_display.plot()
plt.show()
```



Tuning for GaussianNB

```
parameters = {
   'var_smoothing': [1e-9, 1e-8, 1e-7, 1e-6]  # The regularization term (for
smoothing the variance)

}

nb_model = GaussianNB()
grid_search = GridSearchCV(estimator = nb_model,
                           param_grid = parameters,
                           scoring = 'accuracy',
                           cv = 5,
                           verbose=0)
# Fit the model
```

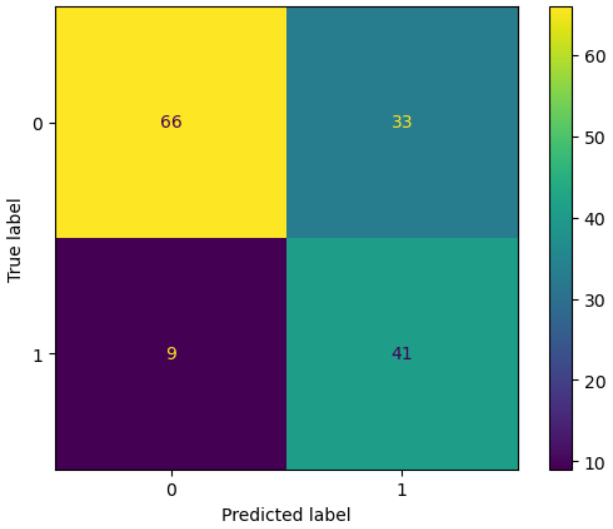
```
grid_search.fit(x_train, y_train)
grid_search.best_params_
```

{'var_smoothing': 1e-09}

```
y_pre = grid_search.predict(x_test)
print(classification_report(y_test,y_pre))
```

	precision	recall	f1-score	support
0	0.88	0.67	0.76	99
1	0.55	0.82	0.66	50
accuracy			0.72	149
macro avg	0.72	0.74	0.71	149
weighted avg	0.77	0.72	0.73	149

```
confusion_matrix = metrics.confusion_matrix(y_test, y_pre)

cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix =
confusion_matrix, display_labels = [0, 1])

cm_display.plot()
plt.show()
```



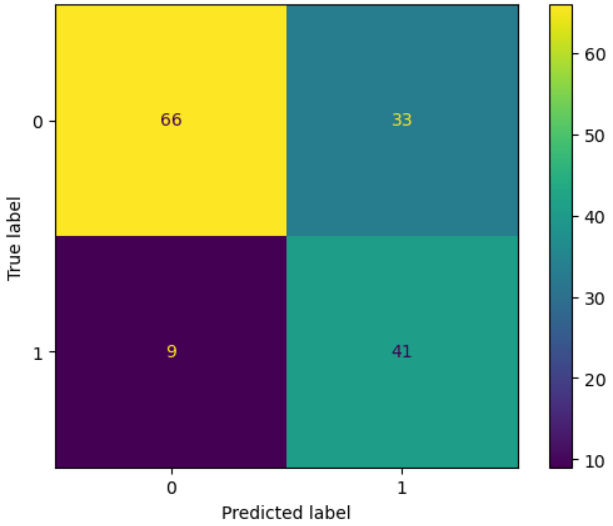
Random Forest

```
rf_model = RandomForestClassifier(criterion='gini',
                        max_depth=8,
                        min_samples_split=10)
rf_model.fit(x_train, y_train)
y_pre = grid_search.predict(x_test)
print(classification_report(y_test,y_pre))
```

	precision	recall	f1-score	support
0	0.88	0.67	0.76	99
1	0.55	0.82	0.66	50
accuracy			0.72	149
macro avg	0.72	0.74	0.71	149
weighted avg	0.77	0.72	0.73	149

```
confusion_matrix = metrics.confusion_matrix(y_test, y_pre)
```

```
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix =
confusion_matrix, display_labels = [0, 1])

cm_display.plot()
plt.show()
```



3.3. Evaluate the Model

In the model-building process, we evaluate a variety of machine learning models to find the best one for our task. These models include:

- Logistic Regression
- Support Vector Machine (SVM)
- Gaussian Naive Bayes (GaussianNB) (with tuning)
- Random Forest

Each model is evaluated based on several important performance metrics, including precision, accuracy, and Type I error (also known as the false negative rate). **Type I error** refers to the situation where the true label is positive (indicating that the individual actually has diabetes), but the model prediction is negative (indicating that the model incorrectly predicts no diabetes).

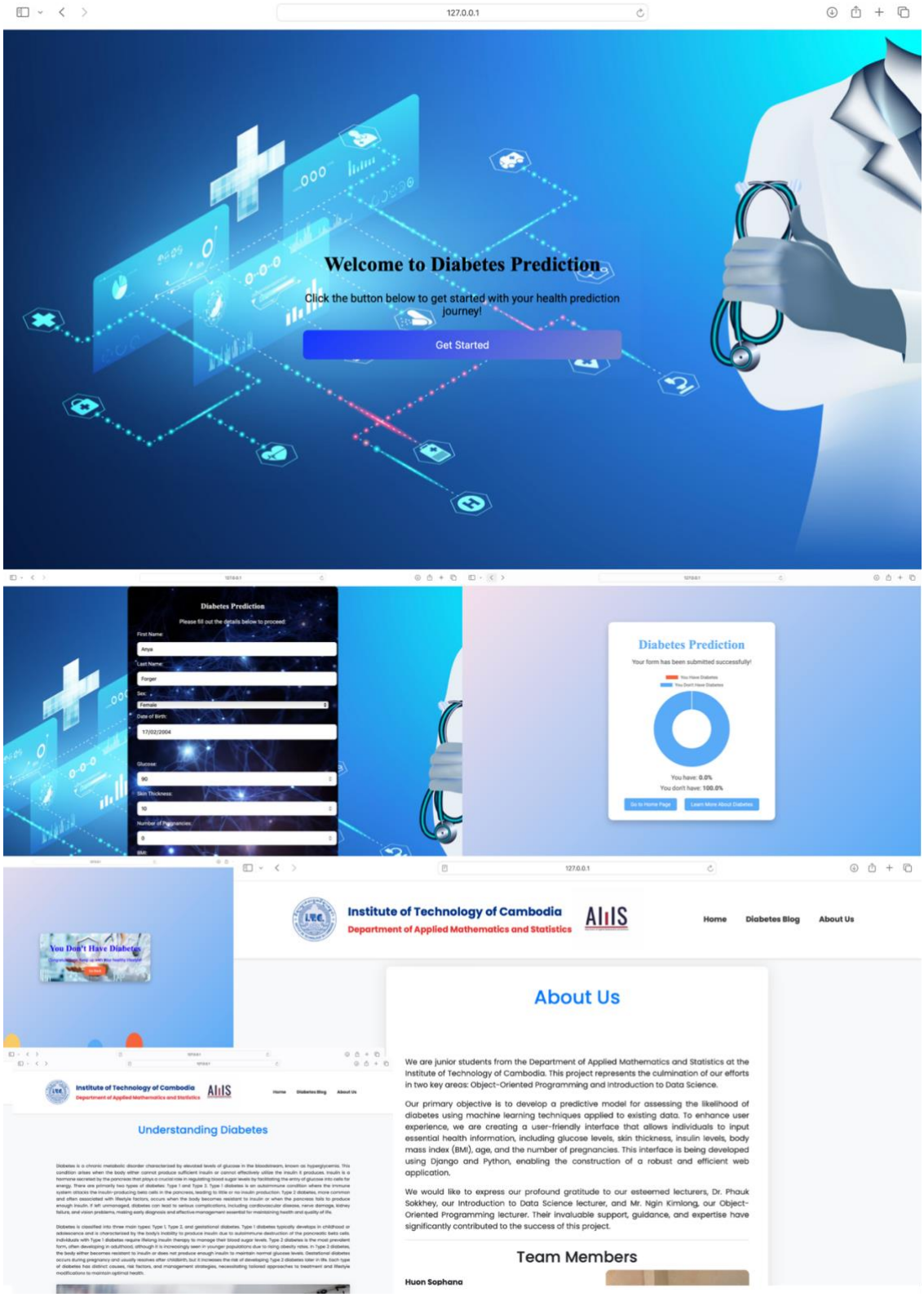
In medical diagnostics, a false negative can have severe consequences. For instance, if the model incorrectly predicts that a person with diabetes does not have diabetes (when they actually do), it could result in a lack of necessary treatment or mismanagement of the condition. This could lead to serious health risks for the individual. Therefore, minimizing Type I error is especially important in this case to ensure that individuals who truly have diabetes are correctly identified and treated.

After comparing the models, we decided to choose **Support Vector Machine (SVM)** because it consistently performed better in terms of precision, accuracy, and especially Type I error. SVM balances the trade-off between underfitting and overfitting while handling high-dimensional data well, which made it the ideal choice for our task.

4. Deploy the Model

To take our model to the next level, we have decided to integrate it into a web-based platform. This will allow people to access and use the model for free, making it widely

available. Most importantly, our goal is to provide a valuable tool for rural hospitals, especially those that lack the necessary medical equipment. By offering an accessible and easy-to-use platform, we aim to bridge the gap in healthcare availability, enabling healthcare providers in underserved areas to make more accurate diabetes predictions and provide better care to their patients.



5. Conclusion

In conclusion, this project has successfully demonstrated the transformative value of machine learning in enhancing the early diagnosis of diabetes. By employing the Support Vector Machine (SVM) model and deploying it via a web application, we have created a powerful tool that empowers healthcare professionals to detect diabetes at an earlier stage. This early detection allows patients to receive timely treatment, which is crucial in preventing serious health complications associated with diabetes.

What makes this approach particularly impactful is its potential to greatly improve patient outcomes, especially in under-resourced areas where access to advanced medical equipment may be limited or unavailable. By making this tool accessible through a web application, we ensure that healthcare providers in rural or underserved communities can leverage modern technology to make informed, data-driven diagnoses, ultimately leading to better health outcomes for individuals who may otherwise go undiagnosed. This project showcases how machine learning can bridge the gap between technological advancements and global healthcare disparities.

6. Reference

- Dataset from Kaggle: Diabetes Dataset
- Slide presentation in Canva: IDS Slide
- Source code for diabetes predictor modeling in GitHup: Diabetes Predictor Code
- Source code for diabetes predictor website in GitHup: Diabetes Predictor Website Code