Diabetes Detection Using Machine Learning Classification Methods

Lab Group Assignment

Prepared by

Siddarth Arora (23110009)
Manraj Singh Cheema (23110008)
Piyush Kanda (23110012)
Antriksh Rana (23110002)

Task 1: Dataset Identification and EDA (in R)

We are trying to research on early detection of Type 2 diabetes. The Dataset to be used is: Pima Indians dataset from the National Institute of Diabetes and Digestive and Kidney Diseases

Patients in the dataset to be included need to have:

- Age >= 21
- Sex = Female
- Heritage = Pima Indian Heritage

There are 8 Predictor Variables (Inputs) for which we have 1 Target Variable (Output), whether patient is diagnosed with Type 2 diabetes or not

```
# Reading the file to import the database

data <- read.csv("C:/Users/kanda/OneDrive/Desktop/Coding/Semester 4/Data Exploration for AI/Assignments/Group Assignment/diabetes.csv")
head(data)
```

A data.frame: 6 × 9

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
	<int>	<int>	<int>	<int>	<int>	<dbl>	<dbl>	<int>	<int>
1	6	148	72	35	0	33.6	0.627	50	1
2	1	85	66	29	0	26.6	0.351	31	0
3	8	183	64	0	0	23.3	0.672	32	1
4	1	89	66	23	94	28.1	0.167	21	0
5	0	137	40	35	168	43.1	2.288	33	1
6	5	116	74	0	0	25.6	0.201	30	0

0 = Healthy ; 1 = Diagnosed with Type 2 diabetes

In [5]:

```
# Filtering the data:

filtered_data <- subset(data, Age >= 21)
head(filtered_data)
```

A data.frame: 6 × 9

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
	<int>	<int>	<int>	<int>	<int>	<dbl>	<dbl>	<int>	<int>
1	6	148	72	35	0	33.6	0.627	50	1
2	1	85	66	29	0	26.6	0.351	31	0
3	8	183	64	0	0	23.3	0.672	32	1
4	1	89	66	23	94	28.1	0.167	21	0
5	0	137	40	35	168	43.1	2.288	33	1
6	5	116	74	0	0	25.6	0.201	30	0

In [7]:

```
nrow (data)
```

```
768
```

In [9]:

```
nrow(filtered_data)
```

```
768
```

The purpose of Exploratory Data Analysis is to understand the Data and apply required editing, manipulations and preprocessing to the data for Machine Learning and Statistical Modelling

As a part of Data Exploration and Analysis, we shall firstly clean the data before any other analysis

Data Cleaning includes tasks like:

```
1. Handling missing values

2. Removing Duplicate data

3. Imputing data values etc.
```

Checking the structure of the data:

In [11]:

```
print(str(filtered_data))
'data.frame':	768 obs. of  9 variables:
 $ Pregnancies             : int  6 1 8 1 0 5 3 10 2 8 ...
 $ Glucose                 : int  148 85 183 89 137 116 78 115 197 125 ...
 $ BloodPressure           : int  72 66 64 66 40 74 50 0 70 96 ...
 $ SkinThickness           : int  35 29 0 23 35 0 32 0 45 0 ...
 $ Insulin                 : int  0 0 0 94 168 0 88 0 543 0 ...
 $ BMI                     : num  33.6 26.6 23.3 28.1 43.1 25.6 31 35.3 30.5 0 ...
 $ DiabetesPedigreeFunction: num  0.627 0.351 0.672 0.167 2.288 ...
 $ Age                     : int  50 31 32 21 33 30 26 29 53 54 ...
 $ Outcome                 : int  1 0 1 0 1 0 1 0 1 1 ...
NULL
```

In [13]:

```
print(summary(filtered_data))
  Pregnancies        Glucose      BloodPressure    SkinThickness  
 Min.   : 0.000   Min.   :  0.0   Min.   :  0.00   Min.   : 0.00  
 1st Qu.: 1.000   1st Qu.: 99.0   1st Qu.: 62.00   1st Qu.: 0.00  
 Median : 3.000   Median :117.0   Median : 72.00   Median :23.00  
 Mean   : 3.845   Mean   :120.9   Mean   : 69.11   Mean   :20.54  
 3rd Qu.: 6.000   3rd Qu.:140.2   3rd Qu.: 80.00   3rd Qu.:32.00  
 Max.   :17.000   Max.   :199.0   Max.   :122.00   Max.   :99.00  
    Insulin           BMI        DiabetesPedigreeFunction      Age       
 Min.   :  0.0   Min.   : 0.00   Min.   :0.0780           Min.   :21.00  
 1st Qu.:  0.0   1st Qu.:27.30   1st Qu.:0.2437           1st Qu.:24.00  
 Median : 30.5   Median :32.00   Median :0.3725           Median :29.00  
 Mean   : 79.8   Mean   :31.99   Mean   :0.4719           Mean   :33.24  
 3rd Qu.:127.2   3rd Qu.:36.60   3rd Qu.:0.6262           3rd Qu.:41.00  
 Max.   :846.0   Max.   :67.10   Max.   :2.4200           Max.   :81.00  
    Outcome     
 Min.   :0.000  
 1st Qu.:0.000  
 Median :0.000  
 Mean   :0.349  
 3rd Qu.:1.000  
 Max.   :1.000  
```

In [15]:

```
print(names(data))
[1] "Pregnancies"              "Glucose"                 
[3] "BloodPressure"            "SkinThickness"           
[5] "Insulin"                  "BMI"                     
[7] "DiabetesPedigreeFunction" "Age"                     
[9] "Outcome"
```

Checking for the Missing Values

We will be imputing the missing values in the different predictor attributes with their respective medians, since medians are more robust as comapred to the means

In 5 attributes:

- `Glucose`
- `Blood Pressure`
- `SkinThickness`
- `Insulin`
- `BMI`

0 values signify nothing but missing values and hence they shall first be converted to Null values (NA) and then inputation will be done to them

In [17]:

```
# For the 5 columns listed above

# Step 1: Columns where 0 = missing
zero_cols <- c("Glucose", "BloodPressure", "SkinThickness", "Insulin", "BMI")

# Step 2: Replacing 0 with NA
filtered_data[, zero_cols] <- lapply(filtered_data[, zero_cols], function(x) ifelse(x == 0, NA, x))

# Step 3: Imputing NA values using column medians
for (col in zero_cols) {
 filtered_data[[col]][is.na(filtered_data[[col]])] <- median(filtered_data[[col]], na.rm = TRUE)
}
```

In [19]:

```
# For the remaining 3 columns as well, we shall apply imputation

# Optional: Imputing any remaining NAs in the remaining columns
filtered_data <- data.frame(lapply(filtered_data, function(x) {
 if (is.numeric(x)) {
  x[is.na(x)] <- median(x, na.rm = TRUE)
 }
 return(x)
}))
```

In [21]:

```
head(filtered_data)
```

A data.frame: 6 × 9

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1	6	148	72	35	125	33.6	0.627	50	1
2	1	85	66	29	125	26.6	0.351	31	0
3	8	183	64	29	125	23.3	0.672	32	1
4	1	89	66	23	94	28.1	0.167	21	0
5	0	137	40	35	168	43.1	2.288	33	1
6	5	116	74	29	125	25.6	0.201	30	0

Removing the Duplicate rows

In [23]:

```
sum(duplicated(filtered_data)) # helps to find the total number of duplicate rows
```

```
0
```

Thereby, there is no need to remove any duplicate rows, since there no duplicated rows

Converting the Outcome column into a factor for easier classification

In [25]:

```
filtered_data$Outcome <- as.factor(filtered_data$Outcome) # Convert target variable to factor
```

In [27]:

```
filtered_data$Outcome
```

1. 1
2. 0
3. 1
4. 0
5. 1

Levels: 0, 1

The Data Cleaning portion has been completed

For the analysis of the Dataset after cleaning:

There are two types of analysis:

1. Univariate Analysis

2. Bivariate Analysis

UNIVARIATE ANALYSIS

In Univariate Analysis we understand the distribution and behaviour of each individual column

In [29]:

```
# Summary Statistics
print(summary(filtered_data$Pregnancies))
```

```
  Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
 0.000   1.000   3.000   3.845   6.000  17.000 
```

In [31]:

```
print(summary(filtered_data$Glucose))
```

```
  Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
 44.00   99.75  117.00  121.66  140.25  199.00 
```

In [33]:

```
print(summary(filtered_data$BloodPressure))
```

```
  Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
 24.00   64.00   72.00   72.39   80.00  122.00 
```

In [35]:

```
print(summary(filtered_data$SkinThickness))
```

```
  Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  7.00   25.00   29.00   29.11   32.00   99.00 
```

In [37]:

```
print(summary(filtered_data$Insulin))
```

```
  Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  14.0   121.5   125.0   140.7   127.2   846.0 
```

In [39]:

```
print(summary(filtered_data$BMI))
```

```
  Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
 18.20   27.50   32.30   32.46   36.60   67.10 
```

In [41]:

```
print(summary(filtered_data$DiabetesPedigreeFunction))
```

```
  Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
0.0780  0.2437  0.3725  0.4719  0.6262  2.4200 
```

In [43]:

```
print(summary(filtered_data$Age))
```

```
  Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
 21.00   24.00   29.00   33.24   41.00   81.00 
```

MULITVARIATE ANALYSIS

In Multivariate Analysis, we understand the Relationships between Variables

In [49]:

```
# Correlation Matrix
cor(filtered_data[, sapply(filtered_data, is.numeric)])
```

A matrix: 8 × 8 of type dbl

	Pregnancies	**Glucose**	**BloodPressure**	**SkinThickness**	**Insulin**	**BMI**	**DiabetesPedigreeFunction**	**Age**
Pregnancies	1.00000000	0.1282130	0.208615412	0.08176982	0.02504748	0.02155873	-0.033522673	0.54434123
Glucose	0.12821296	1.0000000	0.218937186	0.19261490	0.41945051	0.23104855	0.137326919	0.26690916
BloodPressure	0.20861541	0.2189372	1.000000000	0.19189239	0.04536330	0.28125656	-0.002378336	0.32491539
SkinThickness	0.08176982	0.1926149	0.191892388	1.00000000	0.15561028	0.54320507	0.102188267	0.12610719
Insulin	0.02504748	0.4194505	0.045363305	0.15561028	1.00000000	0.18024114	0.126503086	0.09710125
BMI	0.02155873	0.2310486	0.281256564	0.54320507	0.18024114	1.00000000	0.153437673	0.02559691
DiabetesPedigreeFunction	-0.03352267	0.1373269	-0.002378336	0.10218827	0.12650309	0.15343767	1.000000000	0.03356131
Age	0.54434123	0.2669092	0.324915391	0.12610719	0.09710125	0.02559691	0.033561312	1.00000000

In [51]:

```
# Heatmap
install.packages("ggcorrplot")  # Run this only once
library(ggcorrplot)
```

```
Installing package into 'C:/Users/kanda/AppData/Local/R/win-library/4.4'
(as 'lib' is unspecified)

package 'ggcorrplot' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        C:\Users\kanda\AppData\Local\Temp\Rtmpe6lKM4\downloaded_packages
Warning message:
```

```
"package 'ggcorrplot' was built under R version 4.4.3"
Loading required package: ggplot2

Warning message:
"package 'ggplot2' was built under R version 4.4.3"
```

In [54]:

```
# Since Outcome was converted to a factor, converting it back temporarily
filtered_data$Outcome <- as.numeric(as.character(filtered_data$Outcome))
```

In [64]:

```
# Creating correlation matrix using absolute values
cor_matrix <- abs(cor(filtered_data_numeric[, sapply(filtered_data_numeric, is.numeric)]))
```

In [72]:

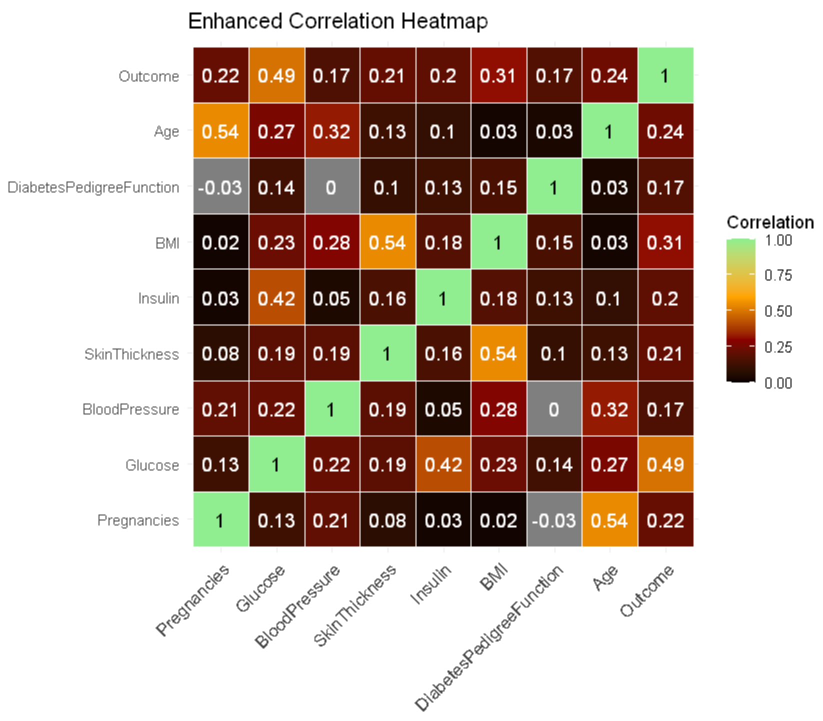
```
cor_matrix
```

A matrix: 9 × 9 of type dbl

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
Pregnancies	1.00000000	0.1282130	0.208615412	0.08176982	0.02504748	0.02155873	-0.033522673	0.54434123	0.2218982
Glucose	0.12821296	1.0000000	0.218937186	0.19261490	0.41945051	0.23104855	0.137326919	0.26690916	0.4927824
BloodPressure	0.20861541	0.2189372	1.000000000	0.19189239	0.04536330	0.28125656	-0.002378336	0.32491539	0.1657229
SkinThickness	0.08176982	0.1926149	0.191892388	1.00000000	0.15561028	0.54320507	0.102188267	0.12610719	0.2148732
Insulin	0.02504748	0.4194505	0.045363305	0.15561028	1.00000000	0.18024114	0.126503086	0.09710125	0.2037903
BMI	0.02155873	0.2310486	0.281256564	0.54320507	0.18024114	1.00000000	0.153437673	0.02559691	0.3120383
DiabetesPedigreeFunction	-0.03352267	0.1373269	-0.002378336	0.10218827	0.12650309	0.15343767	1.000000000	0.03356131	0.1738441
Age	0.54434123	0.2669092	0.324915391	0.12610719	0.09710125	0.02559691	0.033561312	1.00000000	0.2383560
Outcome	0.22189815	0.4927824	0.165722913	0.21487322	0.20379034	0.31203834	0.173844066	0.23835598	1.0000000

In [78]:

```
library(ggplot2)
library(reshape2)
library(dplyr)

# Step 1: Convert Outcome to numeric for correlation
filtered_data_numeric <- filtered_data
filtered_data_numeric$Outcome <- as.numeric(as.character(filtered_data$Outcome))

# Step 2: Create correlation matrix
cor_matrix <- cor(filtered_data_numeric[, sapply(filtered_data_numeric, is.numeric)])

# Step 3: Melt to long format
melted_cor <- melt(cor_matrix)

# Step 4: Create label and text color column
melted_cor <- melted_cor %>%
 mutate(label = round(value, 2),
     is_diag = ifelse(Var1 == Var2, TRUE, FALSE),
     text_color = ifelse(abs(value) > 0.7, "black", "white"))




# Step 5: Heatmap with custom gradient and light green diagonals
ggplot(data = melted_cor, aes(x = Var1, y = Var2, fill = value)) +
 geom_tile(color = "white") +
 geom_text(aes(label = label, color = text_color), size = 4) +
 scale_fill_gradientn(
  colors = c("black", "darkred", "orange", "lightgreen"),  # custom palette
  values = scales::rescale(c(0, 0.3, 0.6, 1)),           # smooth transition
  limits = c(0, 1),
  name = "Correlation"
 ) +
 scale_color_identity() +  # use provided colors
 coord_fixed() +
 labs(title = "Enhanced Correlation Heatmap", x = NULL, y = NULL) +
 theme_minimal() +
 theme(axis.text.x = element_text(angle = 45, vjust = 1, size = 10, hjust = 1))
```


Enhanced Correlation Heatmap
Outcome 0.22 0.49 0.17 0.21 0.2 0.31 0.17 0.24 1
Age 0.54 0.27 0.32 0.13 0.1 0.03 0.03 1 0.24
DiabetesPedigreeFunction -0.03 0.14 0 0.1 0.13 0.15 1 0.03 0.17
BMI 0.02 0.23 0.28 0.54 0.18 1 0.15 0.03 0.31
Insulin 0.03 0.42 0.05 0.16 1 0.18 0.13 0.1 0.2
SkinThickness 0.08 0.19 0.19 1 0.16 0.54 0.1 0.13 0.21
BloodPressure 0.21 0.22 1 0.19 0.05 0.28 0 0.32 0.17
Glucose 0.13 1 0.22 0.19 0.42 0.23 0.14 0.27 0.49
Pregnancies 1 0.13 0.21 0.08 0.03 0.02 -0.03 0.54 0.22
Pregnancies
Glucose
BloodPressure
SkinThickness
Insulin
BMI
DiabetesPedigreeFunction
Age
Outcome
Correlation
1.00
0.75
0.50
0.25
0.00

Task 2: Recreate Visualizations (in R)

```
df <- read.csv("/content/filtered_diabetes_data.csv")
head(df)
```

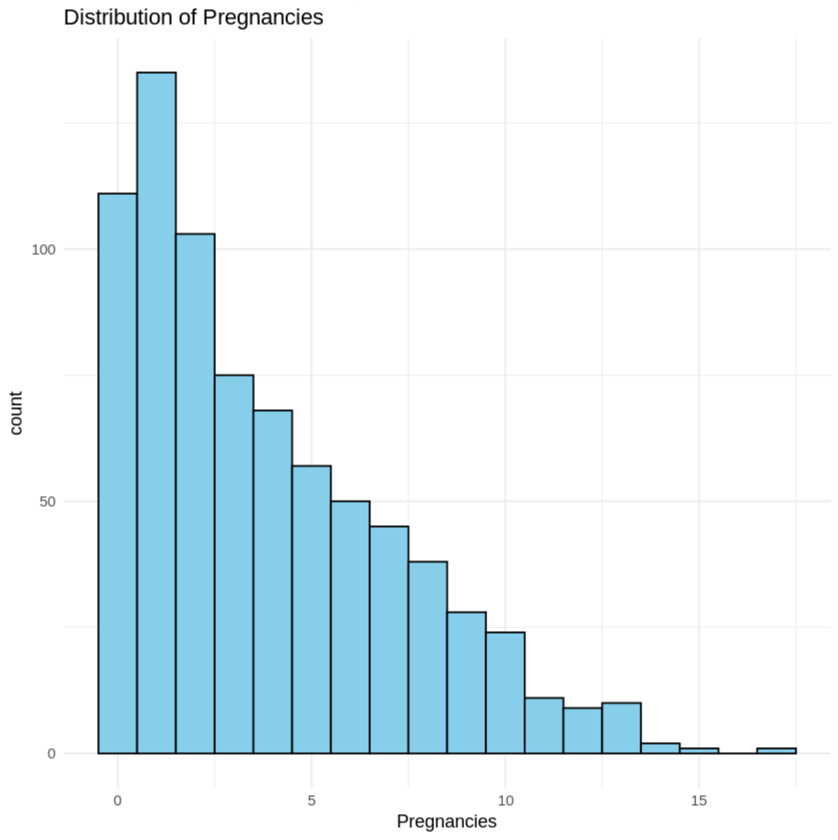
A data.frame: 6 × 9

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
	<int>	<int>	<int>	<int>	<int>	<dbl>	<dbl>	<int>	<int>
1	6	148	72	35	125	33.6	0.627	50	1
2	1	85	66	29	125	26.6	0.351	31	0
3	8	183	64	29	125	23.3	0.672	32	1
4	1	89	66	23	94	28.1	0.167	21	0
5	0	137	40	35	168	43.1	2.288	33	1
6	5	116	74	29	125	25.6	0.201	30	0

histograms

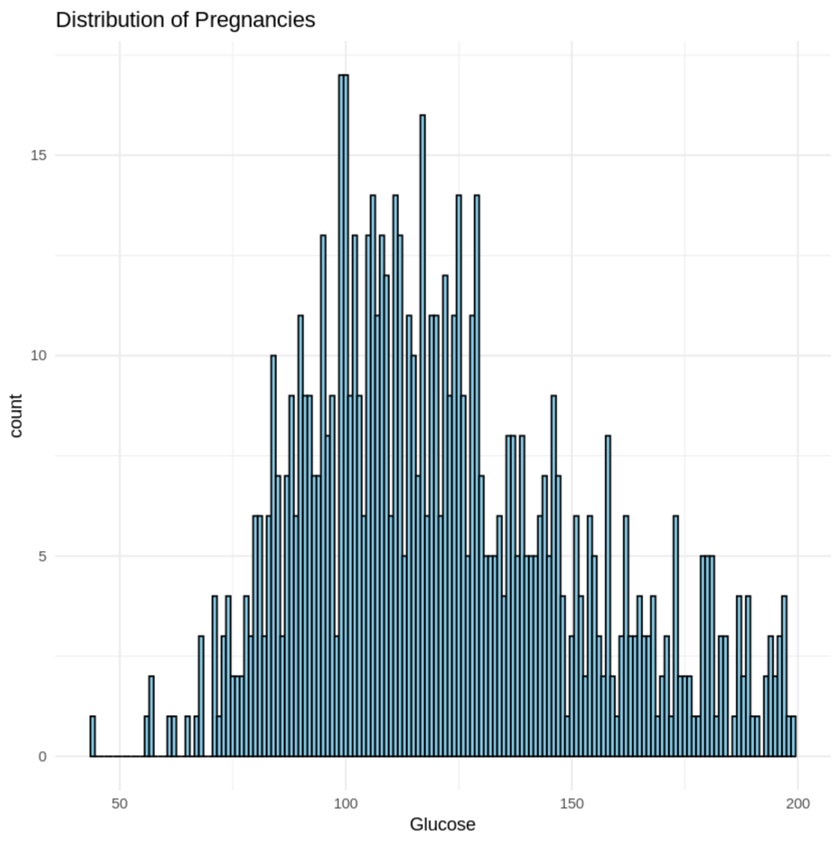
In [18]:

```
ggplot(data = df, aes(x = Pregnancies)) + geom_histogram(binwidth = 1, fill = 'skyblue', color = 'black') +
labs(title = "Distribution of Pregnancies") + theme_minimal()
```



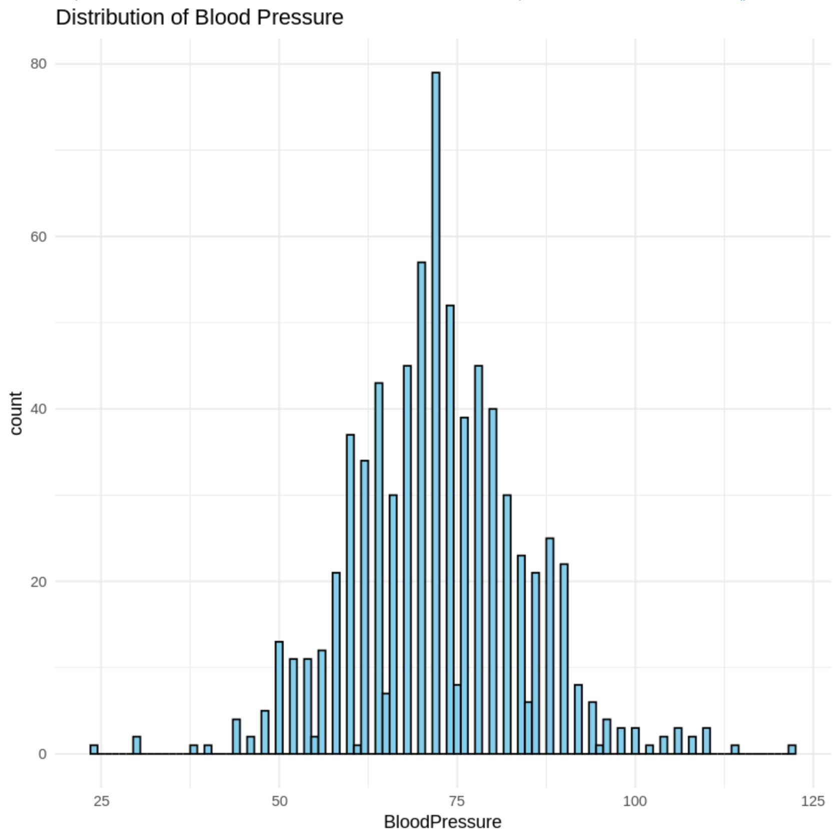
In [19]:

```
ggplot(data = df, aes(x = Glucose)) + geom_histogram(binwidth = 1, fill = 'skyblue', color = 'black') +
labs(title = "Distribution of Pregnancies") + theme_minimal()
```



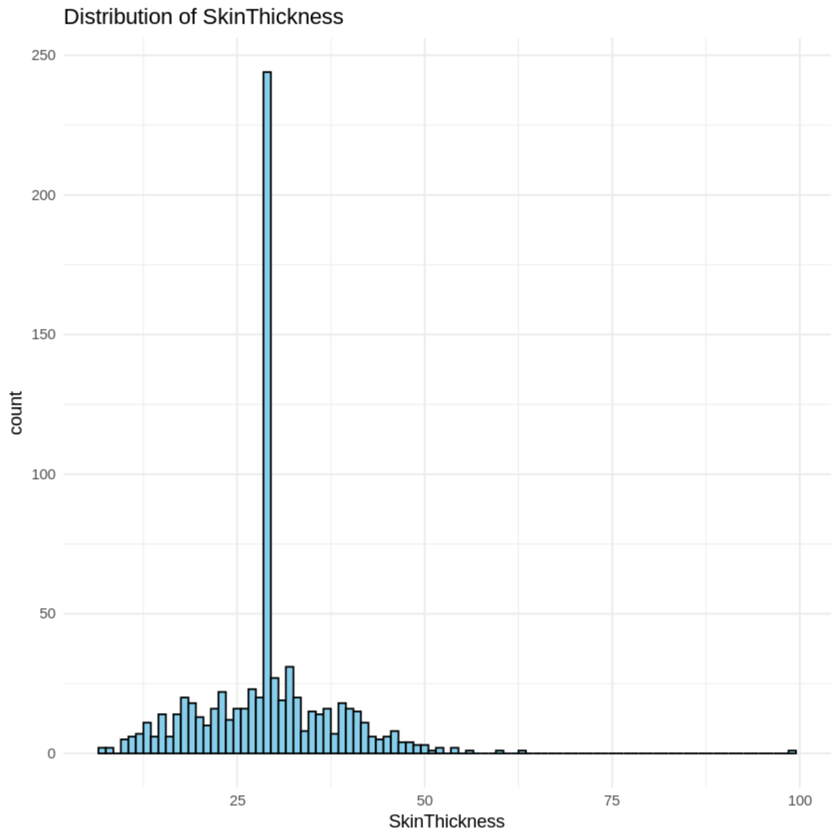
```
In [20]:
```

```
ggplot(data = df, aes(x = BloodPressure)) + geom_histogram(binwidth = 1, fill = 'skyblue', color = 'black')
+ labs(title = "Distribution of Blood Pressure") + theme_minimal()
```



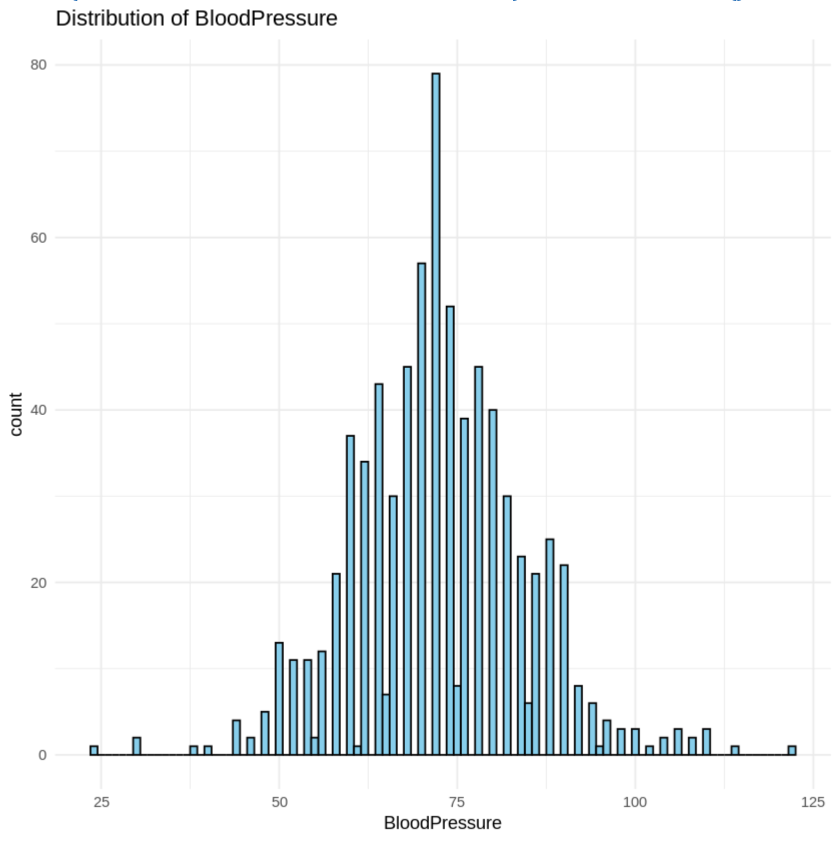
```
In [21]:
```

```
ggplot(data = df, aes(x = SkinThickness)) + geom_histogram(binwidth = 1, fill = 'skyblue', color = 'black')
+ labs(title = "Distribution of SkinThickness") + theme_minimal()
```



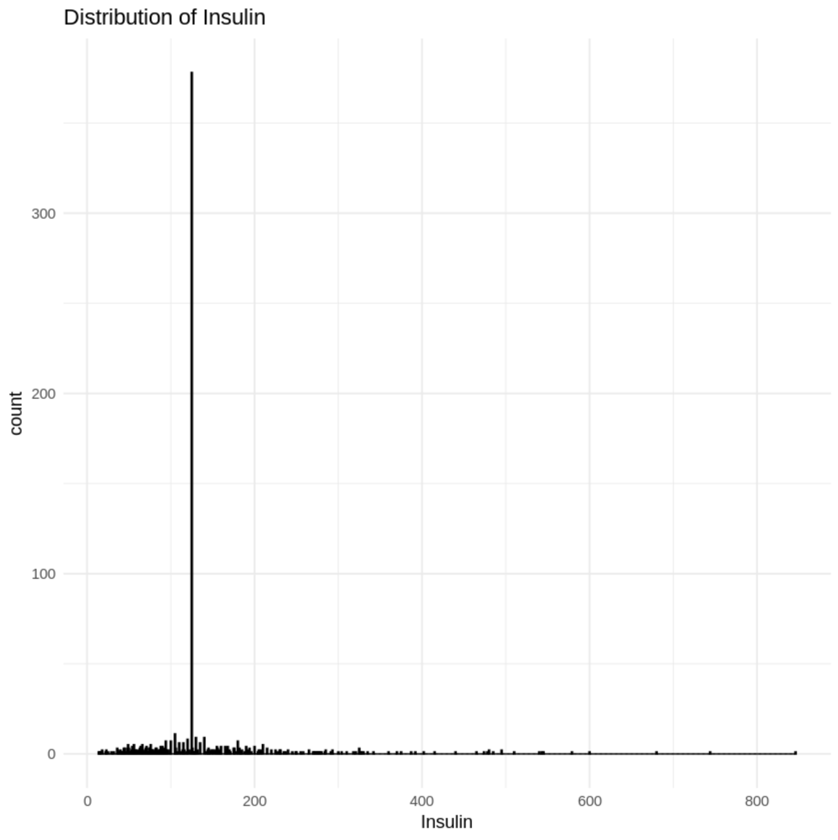
```
In [23]:
```

```
ggplot(data = df, aes(x = BloodPressure)) + geom_histogram(binwidth = 1, fill = 'skyblue', color = 'black')
+ labs(title = "Distribution of BloodPressure") + theme_minimal()
```



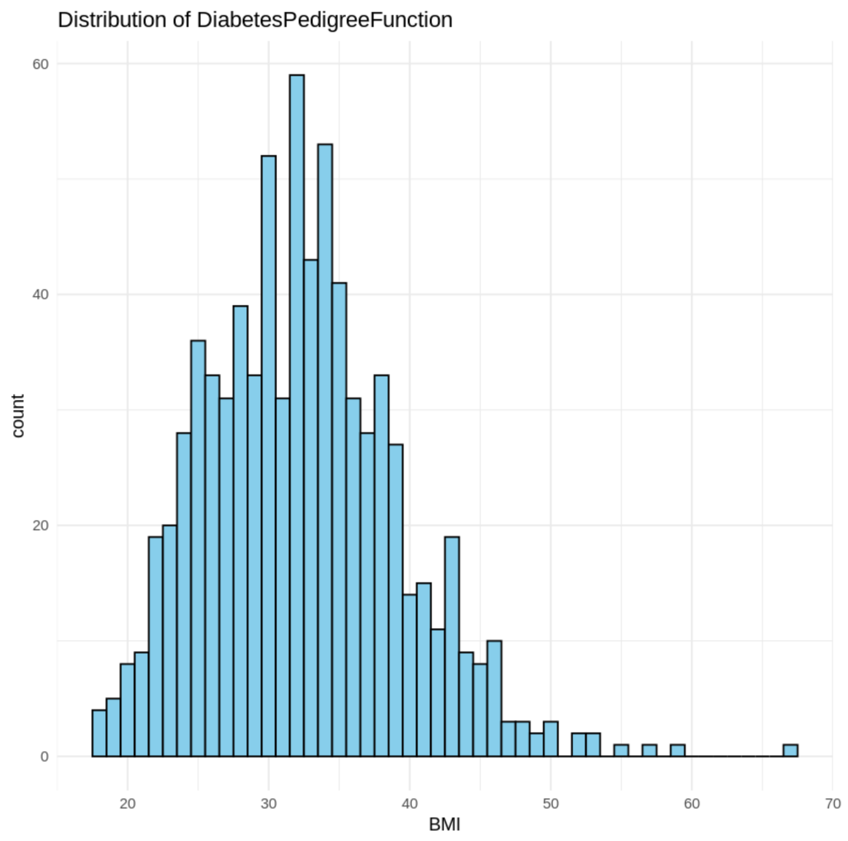
In [24]:

```
ggplot(data = df, aes(x = Insulin)) + geom_histogram(binwidth = 1, fill = 'skyblue', color = 'black') +
labs(title = "Distribution of Insulin") + theme_minimal()
```



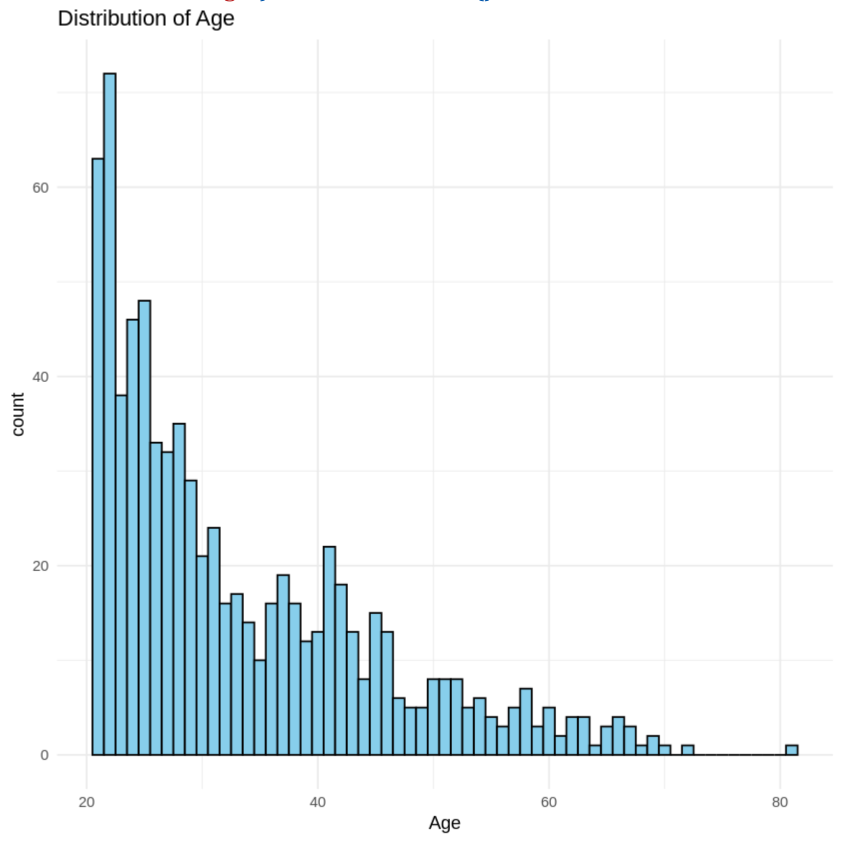
In [25]:

```
ggplot(data = df, aes(x = BMI)) + geom_histogram(binwidth = 1, fill = 'skyblue', color = 'black') + labs(title
= 'Distribution of DiabetesPedigreeFunction') + theme_minimal()
```



In [26]:

```
ggplot(data = df, aes(x = Age)) + geom_histogram(binwidth = 1, fill = 'skyblue', color = 'black') + labs(title
= "Distribution of Age") + theme_minimal()
```



In [ ]:

Task 3: Machine Learning Models (if applicable)

Using Machine Learning model to predict whether a person has type-2 diabetes or not based on input 8 features

In [2]:

```
import pandas as pd
import numpy as np
df = pd.read_csv('/content/diabetes.csv')
df.head()
```

Out[2]:

	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

In [3]:

```
### filling out missing values
df.isna().sum()
```

Out[3]:

	0
Pregnancies	0
Glucose	0
BloodPressure	0
SkinThickness	0

	0
Insulin	0
BMI	0
DiabetesPedigreeFunction	0
Age	0
Outcome	0

dtype: int64

In [4]:

```
df[df['Glucose'] == 0]
```

Out[4]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
75	1	0	48	20	0	24.7	0.140	22	0
182	1	0	74	20	23	27.7	0.299	21	0
342	1	0	68	35	0	32.0	0.389	22	0
349	5	0	80	32	0	41.0	0.346	37	1
502	6	0	68	41	0	39.0	0.727	41	1

In [5]:

```
df[df['BloodPressure'] == 0]
```

Out[5]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
7	10	115	0	0	0	35.3	0.134	29	0

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
15	7	100	0	0	0	30.0	0.484	32	1
49	7	105	0	0	0	0.0	0.305	24	0
60	2	84	0	0	0	0.0	0.304	21	0
78	0	131	0	0	0	43.2	0.270	26	1
81	2	74	0	0	0	0.0	0.102	22	0
172	2	87	0	23	0	28.9	0.773	25	0
193	11	135	0	0	0	52.3	0.578	40	1
222	7	119	0	0	0	25.2	0.209	37	0
261	3	141	0	0	0	30.0	0.761	27	1
266	0	138	0	0	0	36.3	0.933	25	1
269	2	146	0	0	0	27.5	0.240	28	1
300	0	167	0	0	0	32.3	0.839	30	1
332	1	180	0	0	0	43.3	0.282	41	1
336	0	117	0	0	0	33.8	0.932	44	0

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
347	3	116	0	0	0	23.5	0.187	23	0
357	13	129	0	30	0	39.9	0.569	44	1
426	0	94	0	0	0	0.0	0.256	25	0
430	2	99	0	0	0	22.2	0.108	23	0
435	0	141	0	0	0	42.4	0.205	29	1
453	2	119	0	0	0	19.6	0.832	72	0
468	8	120	0	0	0	30.0	0.183	38	1
484	0	145	0	0	0	44.2	0.630	31	1
494	3	80	0	0	0	0.0	0.174	22	0
522	6	114	0	0	0	0.0	0.189	26	0
533	6	91	0	0	0	29.8	0.501	31	0
535	4	132	0	0	0	32.9	0.302	23	1
589	0	73	0	0	0	21.1	0.342	25	0

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
601	6	96	0	0	0	23.7	0.190	28	0
604	4	183	0	0	0	28.4	0.212	36	1
619	0	119	0	0	0	32.4	0.141	24	1
643	4	90	0	0	0	28.0	0.610	31	0
697	0	99	0	0	0	25.0	0.253	22	0
703	2	129	0	0	0	38.5	0.304	41	0
706	10	115	0	0	0	0.0	0.261	30	1

In [6]:

```
df[df['SkinThickness'] == 0]
```

Out[6]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
2	8	183	64	0	0	23.3	0.672	32	1
5	5	116	74	0	0	25.6	0.201	30	0
7	10	115	0	0	0	35.3	0.134	29	0
9	8	125	96	0	0	0.0	0.232	54	1
10	4	110	92	0	0	37.6	0.191	30	0

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
...	...	...	...	...	...	...	...	...	...
757	0	123	72	0	0	36.3	0.258	52	1
758	1	106	76	0	0	37.5	0.197	26	0
759	6	190	92	0	0	35.5	0.278	66	1
762	9	89	62	0	0	22.5	0.142	33	0
766	1	126	60	0	0	30.1	0.349	47	1

227 rows × 9 columns

In [7]:

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

Out[7]:

	count
SkinThickness	
0	227
32	31
30	27
27	23
23	22
18	20

	count
SkinThickness	
33	20
28	20
31	19
39	18
19	18
29	17
25	16
40	16
22	16
37	16
26	16
41	15
35	15
36	14
15	14
17	14
20	13

	count
SkinThickness	
24	12
42	11
13	11
21	10
46	8
34	8
12	7
38	7
16	6
11	6
45	6
14	6
43	6
44	5
10	5
47	4
48	4

	count
SkinThickness	
49	3
50	3
54	2
8	2
52	2
7	2
60	1
51	1
56	1
63	1
99	1

dtype: int64

In [8]:

```
df[df['Insulin'] == 0]
```

Out[8]:

	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

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
2	8	183	64	0	0	23.3	0.672	32	1
5	5	116	74	0	0	25.6	0.201	30	0
7	10	115	0	0	0	35.3	0.134	29	0
...	...	...	...	...	...	...	...	...	...
761	9	170	74	31	0	44.0	0.403	43	1
762	9	89	62	0	0	22.5	0.142	33	0
764	2	122	70	27	0	36.8	0.340	27	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

374 rows × 9 columns

In [9]:

```
df[df.BMI == 0]
```

Out[9]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
9	8	125	96	0	0	0.0	0.232	54	1
49	7	105	0	0	0	0.0	0.305	24	0
60	2	84	0	0	0	0.0	0.304	21	0

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
81	2	74	0	0	0	0.0	0.102	22	0
145	0	102	75	23	0	0.0	0.572	21	0
371	0	118	64	23	89	0.0	1.731	21	0
426	0	94	0	0	0	0.0	0.256	25	0
494	3	80	0	0	0	0.0	0.174	22	0
522	6	114	0	0	0	0.0	0.189	26	0
684	5	136	82	0	0	0.0	0.640	69	0
706	10	115	0	0	0	0.0	0.261	30	1

In [10]:

```
df['BMI'].fillna(df['BMI'].mean(), inplace = True)
```

```
<ipython-input-10-af84c3c91cd3>:1: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.


  df['BMI'].fillna(df['BMI'].mean(), inplace = True)
```

In [11]:

```
df['SkinThickness'].fillna(df['SkinThickness'].mean(), inplace = True)
```

```
<ipython-input-11-a88431808108>:1: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.


  df['SkinThickness'].fillna(df['SkinThickness'].mean(), inplace = True)
```

In [12]:

```
df['BloodPressure'].fillna(df['BloodPressure'].mean(), inplace = True)
```

```
<ipython-input-12-7f0f99ddb78f>:1: FutureWarning: A value is trying to be set on a copy of a DataFrame
or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate ob
ject on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace
=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.


  df['BloodPressure'].fillna(df['BloodPressure'].mean(), inplace = True)
```

In [13]:

```
df['Glucose'].fillna(df['Glucose'].mean(), inplace = True)
```

```
<ipython-input-13-af81b4c27021>:1: FutureWarning: A value is trying to be set on a copy of a DataFrame
or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate ob
ject on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace
=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.


  df['Glucose'].fillna(df['Glucose'].mean(), inplace = True)
```

Train Test Splitting

In [14]:

```
from sklearn.model_selection import train_test_split , GridSearchCV, RandomizedSearchCV
X_train, X_test, y_train, y_test = train_test_split(df.drop('Outcome', axis = 1), df['Outcome'], test_size = 0.2,
random_state = 42)
```

In [15]:

```
## standard Scaling
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [16]:

```
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from xgboost import XGBClassifier
```

In [17]:

```
models = {
  'SVC' : SVC(),
  'RandomForest' : RandomForestClassifier(),
  'LogisticRegression' : LogisticRegression(),
  'XGBoost' : XGBClassifier()
}
```

In [18]:

```
model_hyperparams = {
  'SVC': {
    'kernel': ['linear', 'rbf', 'poly'],
    'C': [0.1, 1, 10]
    },
  'RandomForest': {
   'n_estimators': np.arange(100, 1000, 100),
  'max_depth': np.arange(10, 32),
  'min_samples_split': np.arange(2, 20),
  'min_samples_leaf': np.arange(1, 20),
  'max_features': ['sqrt', 'log2', None],
  'bootstrap': [True, False]
  },
  'LogisticRegression': {
    'C': [0.1, 1, 10],
    'penalty': ['l1', 'l2']
  },
  'XGBoost': {
    'n_estimators': [100, 500],
    'learning_rate': [0.01, 0.1, 0.2]
  }

}
```

Randomized SearchCV to find best fit of hyperparameters for each

In [19]:

```
def evaluate_models(models, model_hyperparams, X_train, X_test, y_train, y_test):
  best_models = {}
  for model_name, model in models.items():
    params = model_hyperparams[model_name]
    grid = RandomizedSearchCV(model, params, cv = 5)
    grid.fit(X_train, y_train)
    best_models[model_name] = grid.best_estimator_
    print(f"Best parameters for {model_name}: {grid.best_params_}")
    print(f"Best score for {model_name}: {grid.best_score_}")
  return best_models
```

In [20]:

```
best_models = evaluate_models(models, model_hyperparams, X_train_scaled, X_test_scaled, y_train,
y_test)
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/model_selection/_search.py:317: UserWarning: The to
tal space of parameters 9 is smaller than n_iter=10. Running 9 iterations. For exhaustive searches, use Gri
dSearchCV.
  warnings.warn(
Best parameters for SVC: {'kernel': 'rbf', 'C': 1}
Best score for SVC: 0.7687458349993335
Best parameters for RandomForest: {'n_estimators': np.int64(100), 'min_samples_split': np.int64(13), 'mi
n_samples_leaf': np.int64(2), 'max_features': 'sqrt', 'max_depth': np.int64(13), 'bootstrap': True}
Best score for RandomForest: 0.783406637345062
Best parameters for LogisticRegression: {'penalty': 'l2', 'C': 10}
Best score for LogisticRegression: 0.7655071304811408
/usr/local/lib/python3.11/dist-packages/sklearn/model_selection/_search.py:317: UserWarning: The to
tal space of parameters 6 is smaller than n_iter=10. Running 6 iterations. For exhaustive searches, use Gri
dSearchCV.
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/model_selection/_validation.py:528: FitFailedWarning
:
15 fits failed out of a total of 30.
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
15 fits failed with the following error:
Traceback (most recent call last):
  File "/usr/local/lib/python3.11/dist-packages/sklearn/model_selection/_validation.py", line 866, in _fit
_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.11/dist-packages/sklearn/base.py", line 1389, in wrapper
    return fit_method(estimator, *args, **kwargs)
```

```
        ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py", line 1193, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
             ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py", line 63, in _check_sol
ver
    raise ValueError(
ValueError: Solver lbfgs supports only 'l2' or None penalties, got l1 penalty.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/model_selection/_search.py:1108: UserWarning: One
or more of the test scores are non-finite: [       nan 0.76389444        nan 0.76062908        nan 0.76550713]
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/model_selection/_search.py:317: UserWarning: The to
tal space of parameters 6 is smaller than n_iter=10. Running 6 iterations. For exhaustive searches, use Gri
dSearchCV.
  warnings.warn(
Best parameters for XGBoost: {'n_estimators': 100, 'learning_rate': 0.2}
Best score for XGBoost: 0.7589230974276956
```

In [21]:

```
from sklearn.metrics import classification_report
model = RandomForestClassifier(max_depth=np.int64(26), max_features='log2',
                      min_samples_leaf=np.int64(4),
                      min_samples_split=np.int64(2),
                      n_estimators=np.int64(700))
```

In [22]:

```
model.fit(X_train_scaled, y_train)
y_pred = model.predict(X_test_scaled)
```

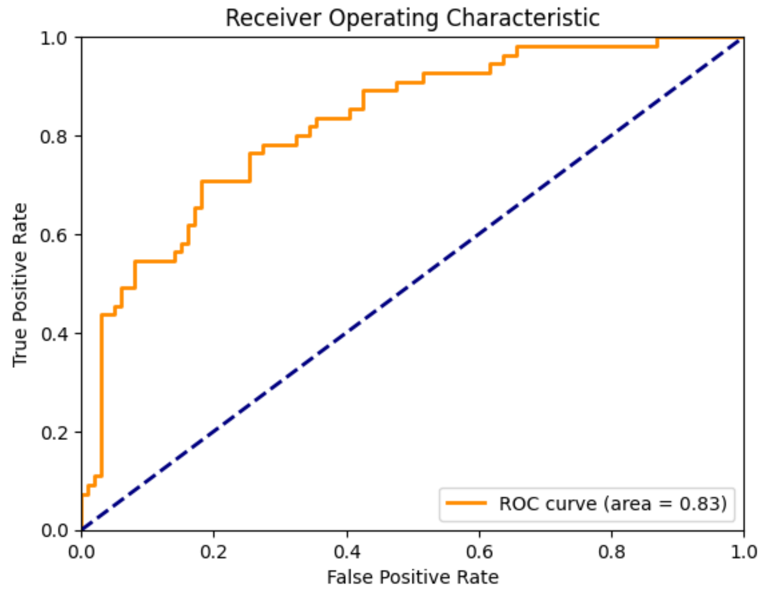
In [23]:

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.84      0.82      0.83        99
           1       0.68      0.71      0.70        55

    accuracy                           0.78       154
   macro avg       0.76      0.76      0.76       154
weighted avg       0.78      0.78      0.78       154
```

In [27]:

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


y_pred_proba = model.predict_proba(X_test_scaled)[:, 1]

# Calculate the ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)

# Calculate the AUC
roc_auc = auc(fpr, tpr)

# Plot the ROC curve
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()
```

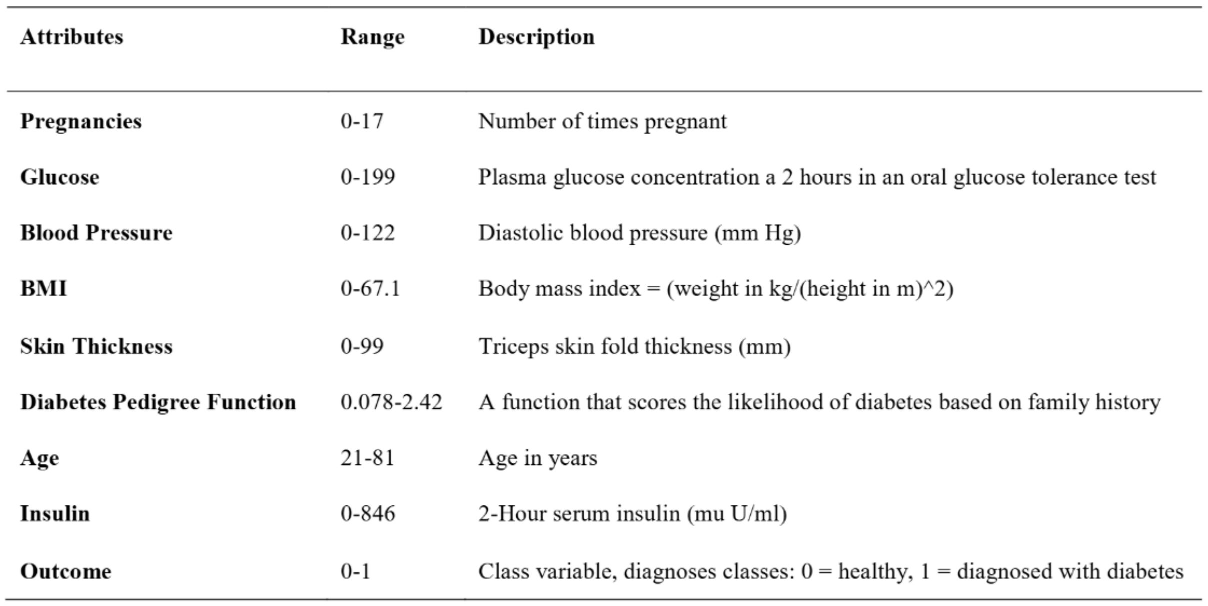


```
In [ ]:
```

Task 4: Critical Analysis

The paper presents several visual elements to communicate the dataset characteristics and model performance metrics.

Table 1: PIMA Indian Dataset Attributes Description



Attributes	Range	Description
Pregnancies	0-17	Number of times pregnant
Glucose	0-199	Plasma glucose concentration a 2 hours in an oral glucose tolerance test
Blood Pressure	0-122	Diastolic blood pressure (mm Hg)
BMI	0-67.1	Body mass index = (weight in kg/(height in m)^2)
Skin Thickness	0-99	Triceps skin fold thickness (mm)
Diabetes Pedigree Function	0.078-2.42	A function that scores the likelihood of diabetes based on family history
Age	21-81	Age in years
Insulin	0-846	2-Hour serum insulin (mu U/ml)
Outcome	0-1	Class variable, diagnoses classes: 0 = healthy, 1 = diagnosed with diabetes

This table provides essential information about the dataset's attributes, including:

- Range: The minimum and maximum values for each attribute
- Description: A brief explanation of what each attribute represents

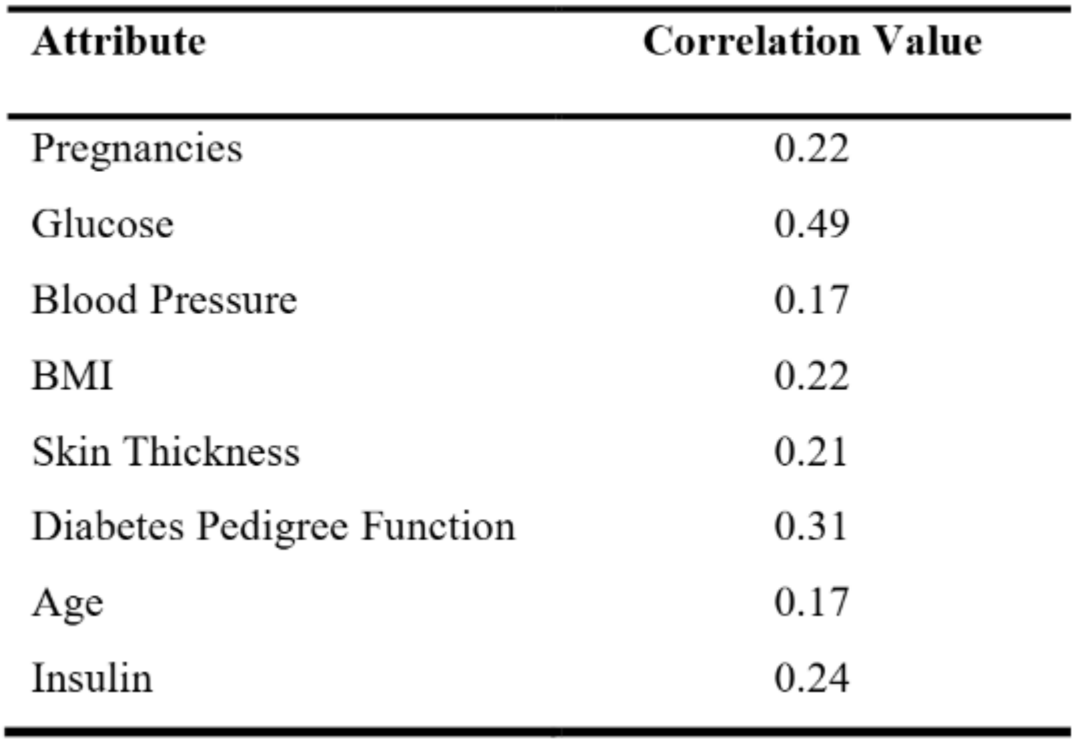
The table effectively documents all nine attributes, including eight predictor variables and one target variable (Outcome). It shows the diversity of medical measurements used for prediction, from basic demographic information like age to specific clinical measurements like glucose levels and insulin.

The table serves as a foundational reference point for understanding the dataset, showing reasonable ranges for each attribute (e.g., pregnancies ranging from 0-17, ages from 21-81). The clear descriptions help readers understand medical terminology like "Diabetes Pedigree Function."

Suggested improvements:

- Include the units of measurement for all attributes consistently (mm Hg is included for blood pressure but units aren't specified for other measurements)
- Add statistical information like mean and standard deviation to provide a better sense of the distribution
- Include information about missing values or zeros in the original dataset since this becomes important later

Table 2: Correlation with Outcome (Target)



Attribute	Correlation Value
Pregnancies	0.22
Glucose	0.49
Blood Pressure	0.17
BMI	0.22
Skin Thickness	0.21
Diabetes Pedigree Function	0.31
Age	0.17
Insulin	0.24

This table quantifies the relationship between each predictor variable and the target outcome (diabetes diagnosis) using Pearson correlation coefficients. The key insights include:

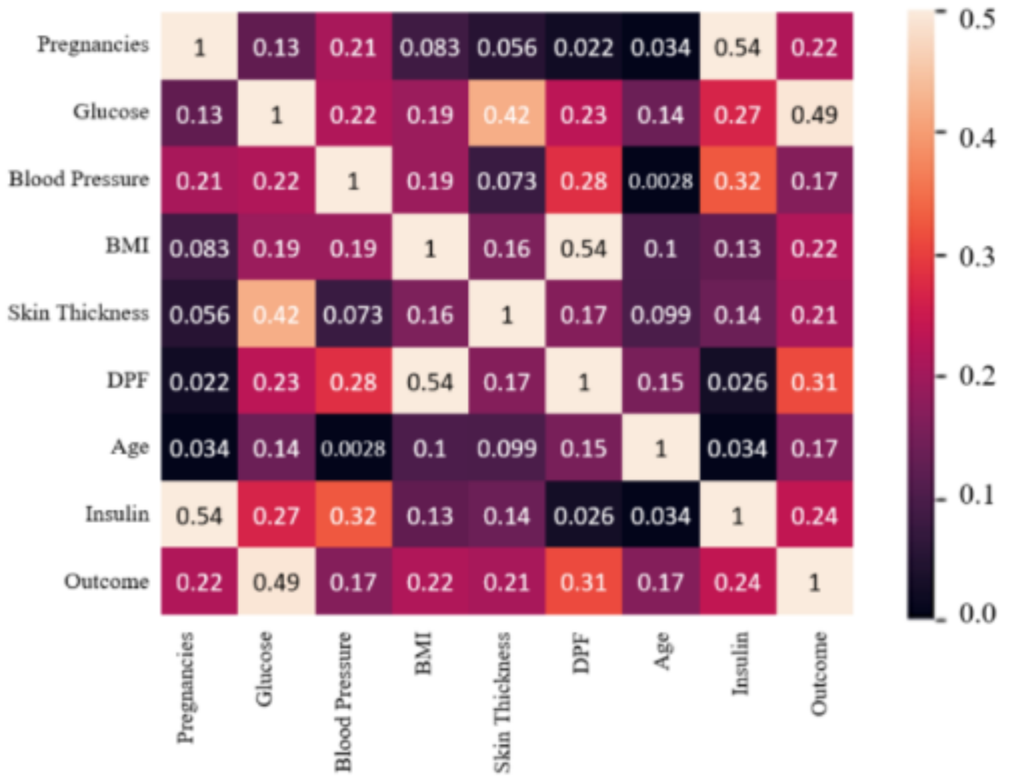
- Glucose shows the strongest correlation (0.49) with diabetes diagnosis
- Diabetes Pedigree Function has the second highest correlation (0.31)
- Other variables show moderate to weak correlations (0.17-0.24)

These correlation values help readers understand which attributes are most predictive of diabetes, informing both the machine learning approach and clinical understanding.

Suggested improvements:

- Include p-values to indicate statistical significance of each correlation
- Sort the attributes by correlation strength for easier interpretation
- Add visual indicators (like + or -) to clarify positive versus negative correlations

Figure 1: Heat Map of Correlation Values



This heat map visualizes the correlation matrix between all variables, using colour intensity to represent correlation strength. The figure shows:

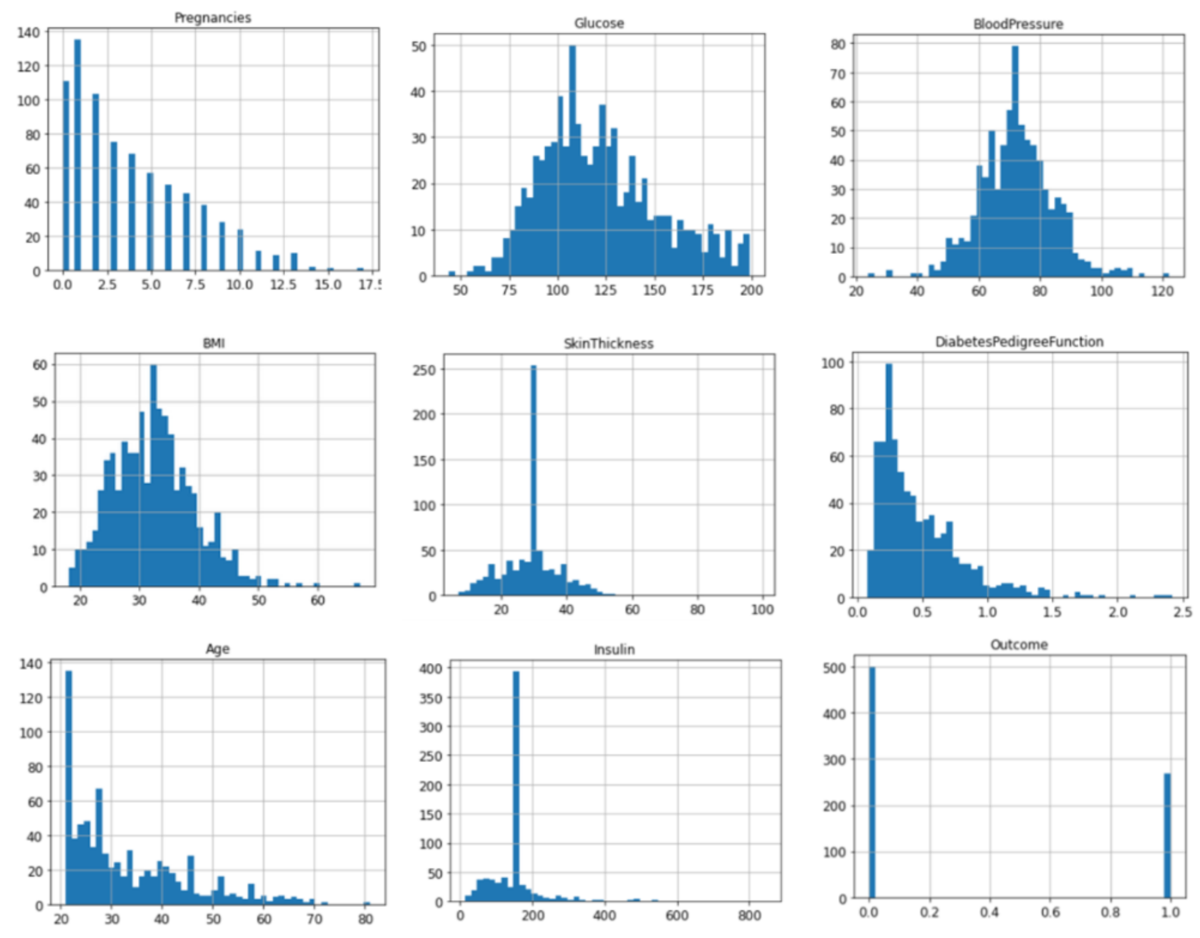
- The diagonal represents perfect self-correlation (1.0)
- Glucose and outcome have the strongest visible positive correlation (matching the 0.49 value from Table 2)
- Age and pregnancy appear to have a moderate positive correlation with each other
- Some attributes show minimal correlation with others (lighter squares)

The heat map effectively condenses a complex matrix of relationships into a visual format, revealing potential multicollinearity issues and reinforcing which variables most strongly predict diabetes.

Suggested improvements:

- Add a colour scale bar to quantify the correlation values
- Label the axes more clearly (the y-axis labels are difficult to read)
- Use a divergent colour scheme (e.g., blue-white-red) to better distinguish positive from negative correlations
- Annotate the strongest correlations with numerical values directly on the heat map

Figure 2: Histograms of the Different Attributes



This figure presents the distribution of each attribute through histograms, showing:

- Most attributes display non-normal distributions
- Some attributes (Glucose, Blood Pressure, BMI) approximate bell curves
- Others (Pregnancies, Insulin, Skin Thickness) show right-skewed distributions
- Age shows multiple peaks, suggesting potential cohort effects
- Outcome shows the class imbalance (more non-diabetic than diabetic cases)

These histograms are crucial for understanding the data distribution and identifying potential preprocessing needs, such as normalization or handling skewed variables.

Suggested improvements:

- Add kernel density plots to smooth the histograms
- Use consistent bin sizes or normalization across attributes
- Add reference lines for mean/median values
- For the Outcome histogram, use descriptive labels ("Diabetic"/"Non-diabetic") rather than just 0/1
- Split histograms by outcome class to show distribution differences between diabetic/non-diabetic groups

Figures 3-8: Confusion Matrices for Different Models

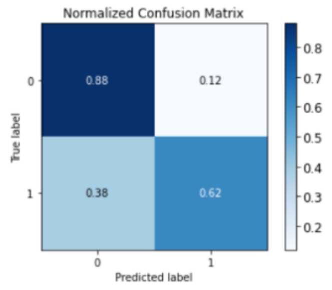


Figure 3: Logistic Regression Confusion Matrix

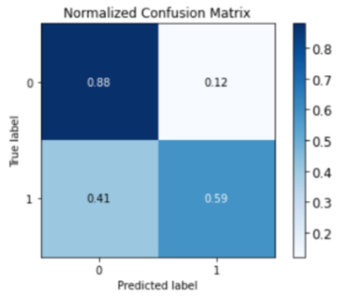


Figure 4: Linear Discriminant Analysis Confusion Matrix

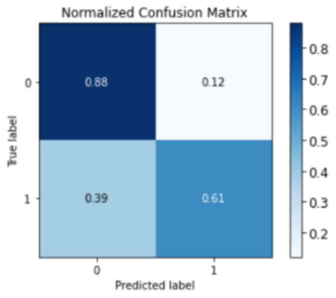


Figure 5: Linear SVM Confusion Matrix

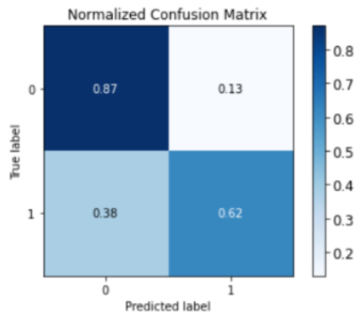


Figure 6: Polynomial SVM Confusion Matrix

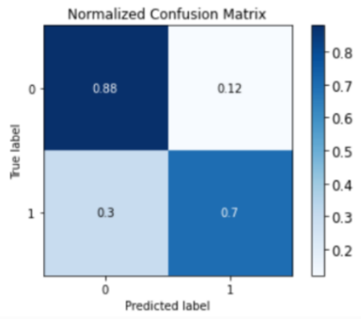


Figure 7: Random Forest Classifier Confusion Matrix

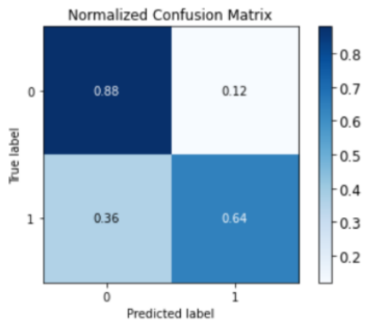


Figure 8: Voting Classifier Confusion Matrix

These six figures display confusion matrices for each classification model tested, showing:

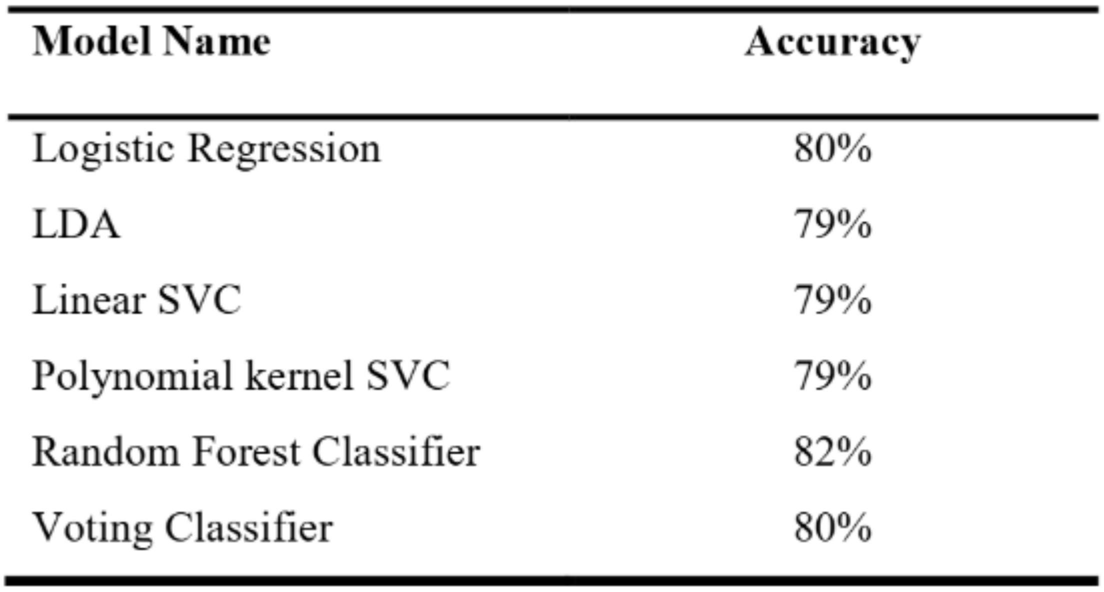
- True positives, false positives, true negatives, and false negatives
- Raw counts rather than percentages
- Similar performance patterns across most models
- Random Forest (Figure 7) shows the highest true positive rate

The confusion matrices provide a deeper understanding of model performance beyond just accuracy, revealing where each model makes errors. This is particularly important for medical diagnostics where false negatives (missing actual diabetes cases) may be more concerning than false positives.

Suggested improvements:

- Add colour intensity or shading to emphasize the diagonal (correct predictions)
- Include derived metrics like precision, recall, and F1-score in each figure
- Standardize the format across all confusion matrices
- Add row and column percentages to better understand error distributions
- Consider combining all matrices into a single figure for easier comparison

Table 3: Accuracy of Trained Models



Model Name	Accuracy
Logistic Regression	80%
LDA	79%
Linear SVC	79%
Polynomial kernel SVC	79%
Random Forest Classifier	82%
Voting Classifier	80%

This final table summarizes the accuracy scores of all six classification models, showing:

- Random Forest Classifier achieved the highest accuracy (82%)
- Logistic Regression and Voting Classifier tied for second place (80%)
- Three models (LDA, Linear SVC, Polynomial kernel SVC) tied at 79%

The table clearly ranks model performance and supports the paper's conclusion that Random Forest was the most effective classifier for this dataset.

Suggested improvements:

- Include additional performance metrics (precision, recall, F1-score, ROC-AUC)
- Add confidence intervals for accuracy scores
- Include computational complexity or training time to assess efficiency
- Highlight the best performer visually
- Add a baseline accuracy (e.g., from always predicting the majority class)

Overall Assessment and Recommendations

The visual elements in this paper effectively communicate both the dataset characteristics and model performance. The authors use appropriate visualization techniques for each purpose: tables for structured information, a heat map for correlation analysis, histograms for distribution analysis, and confusion matrices for performance evaluation.

Recommendations:

1. Integrated visualizations that combine multiple metrics (e.g., ROC curves comparing all models)

2. Feature importance plots for the Random Forest model to show which attributes contribute most to predictions
3. Learning curves to assess how model performance changes with training set size
4. Calibration plots to evaluate how well the predicted probabilities match actual outcomes
5. Visualizations of misclassified instances to understand systematic errors

These visualizations would strengthen the analysis by providing deeper insights into model behaviour and potential areas for improvement in diabetes prediction.