UNIVERSITÀ DEGLI STUDI DI PADOVA
Dipartimento di Matematica

"Diabetes Prediction"
An Analysis of Classification Models

Mohammad Matin Parvanian
Sadaf jamali
Bahador Mirzazadeh

May 2023

Contents

1 INTRODUCTION

Diabetes is a chronic metabolic disease that affects millions of people worldwide, with high prevalence rates in both developed and developing countries. Early detection and prediction of diabetes can help prevent and manage the disease, reducing the risk of complications and improving the quality of life for those affected. In recent years statistical learning techniques have emerged as powerful tools for predicting and diagnosing diabetes based on clinical and demographic features.

In this paper, we aim to conduct a systematic analysis of several classification models for diabetes prediction, including, logistic regression, LDA, QDA, Naive Bayes,and KNN, can be used to build predictive models. Also, we describe our model coefficients. We evaluate the performance of these models on the available dataset, using various performance metrics and feature selection techniques. Additionally, we provide insights into the most informative features of diabetes prediction.

2 PREPRATION OF THE DATASET

2.1 Data Collection

In this project, we used the Diabetes dataset also known as the Pima Indians Diabetes dataset. This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. The objective of the dataset is to diagnostically predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset. The dataset contains information about 768 female patients of Pima Indian heritage, aged 21 years and above. Each data point in the dataset represents a patient and consists of several features or attributes, along with a binary outcome variable indicating whether the patient has diabetes (1) or does not have diabetes (0).

As the first steps we imported some libraries and the Diabetes dataset.

```r
options(warn = -1)
library(class)
library(MASS)
library(e1071)
library(DescTools)
library(ggplot2)
library(reshape, logical.return = FALSE , warn.conflicts = FALSE)
library(naniar, logical.return = FALSE , warn.conflicts = FALSE)
library(knitr, logical.return = FALSE , warn.conflicts = FALSE)
library(dplyr, logical.return = FALSE , warn.conflicts = FALSE)
library(corrplot, logical.return = FALSE , warn.conflicts = FALSE , quietly = TRUE)
```

```
## corrplot 0.92 loaded
```

```r
library(pROC, logical.return = FALSE , warn.conflicts = FALSE , quietly = TRUE)
```

```
## Type 'citation("pROC")' for a citation.
```

```
#### IMPORTATION ####
setwd('E:\\2023-2024A\\Statistical Learning\\data')
Diabetes = read.csv("diabetes.csv")
attach(Diabetes, warn.conflicts = FALSE)
```

In this section we will present some useful basic information about our dataset.

```
#### An overview of dataset ####
str(Diabetes)
```

```
## '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 ...
```

The features in the dataset include:

- Pregnancies: Number of times pregnant.
- Glucose: Plasma glucose concentration.
- BloodPressure: Diastolic blood pressure (mm Hg).
- SkinThickness: Triceps skinfold thickness (mm).
- Insulin: 2-Hour serum insulin (mu U/ml).
- BMI: Body mass index (weight in kg/(height in m)^2).
- DiabetesPedigreeFunction: Diabetes pedigree function, which represents the genetic influence of diabetes based on family history.
- Age: Age of instances in years.

```
#### check the dimension of Diabetes dataset ####
dim(Diabetes)
```

```
## [1] 768   9
```

As you can see we have 768 instances with 9 variables.

2.2 Preprocessing

```
#### Number and Percentage of missing values in Diabetes Dataset ####
miss_var_summary = miss_var_summary(Diabetes)
miss_var_summary
```

```
## # A tibble: 9 x 3
##   variable                 n_miss pct_miss
##   <chr>                     <int>    <dbl>
## 1 Pregnancies                   0        0
## 2 Glucose                       0        0
## 3 BloodPressure                 0        0
## 4 SkinThickness                 0        0
## 5 Insulin                       0        0
## 6 BMI                           0        0
## 7 DiabetesPedigreeFunction      0        0
## 8 Age                           0        0
## 9 Outcome                       0        0
```

As you can see we don't have any missing values in our dataset.But is this true? Let's go deeper through our dataset. There are some variables with zero values like Glucose, BloodPressure, SkinThickness, Insulin, and BMI. These variables cannot take zero values since it is not defined for the range of them. So in this dataset missing values are considered as zero.

The number of zeros for each variable is shown below.

```
#### Number zero values in Diabetes variables ####
for (i in c(2:6)) {
  num_zeros = length(which(Diabetes[,i] == 0))
  print(paste("Column", names(Diabetes)[i], "has", num_zeros, "zeros."))
}
```

```
## [1] "Column Glucose has 5 zeros."
## [1] "Column BloodPressure has 35 zeros."
## [1] "Column SkinThickness has 227 zeros."
## [1] "Column Insulin has 374 zeros."
## [1] "Column BMI has 11 zeros."
```

Let's replace these missing values with NA in order to have a better perception of our dataset.

```
#### let's replace the zero values with NA ####
Diabetes[,2:6][Diabetes[,2:6] == 0] = NA
```

Now we can treat our missing values in three different ways. one of them is removing our missing values but since we have a lot of missing values it is not efficient. The second one is replacing them with mean of each column but the mean is sensitive to being heavily influenced by outliers. A single extreme value can skew the mean significantly. The third way is using median.The median, which is the middle value of our variables, is much more resistant to outliers. So since our columns have outliers, using the median to replace missing values might be a better choice.

```
#### let's replace the NA values with median of their columns ####
for (i in c(2, 3, 4, 5, 6)) {
  NA_values = which(is.na(Diabetes[, i]), arr.ind = TRUE)
  Diabetes[, i][NA_values] = median(Diabetes[, i][!is.na(Diabetes[, i])])
}
```

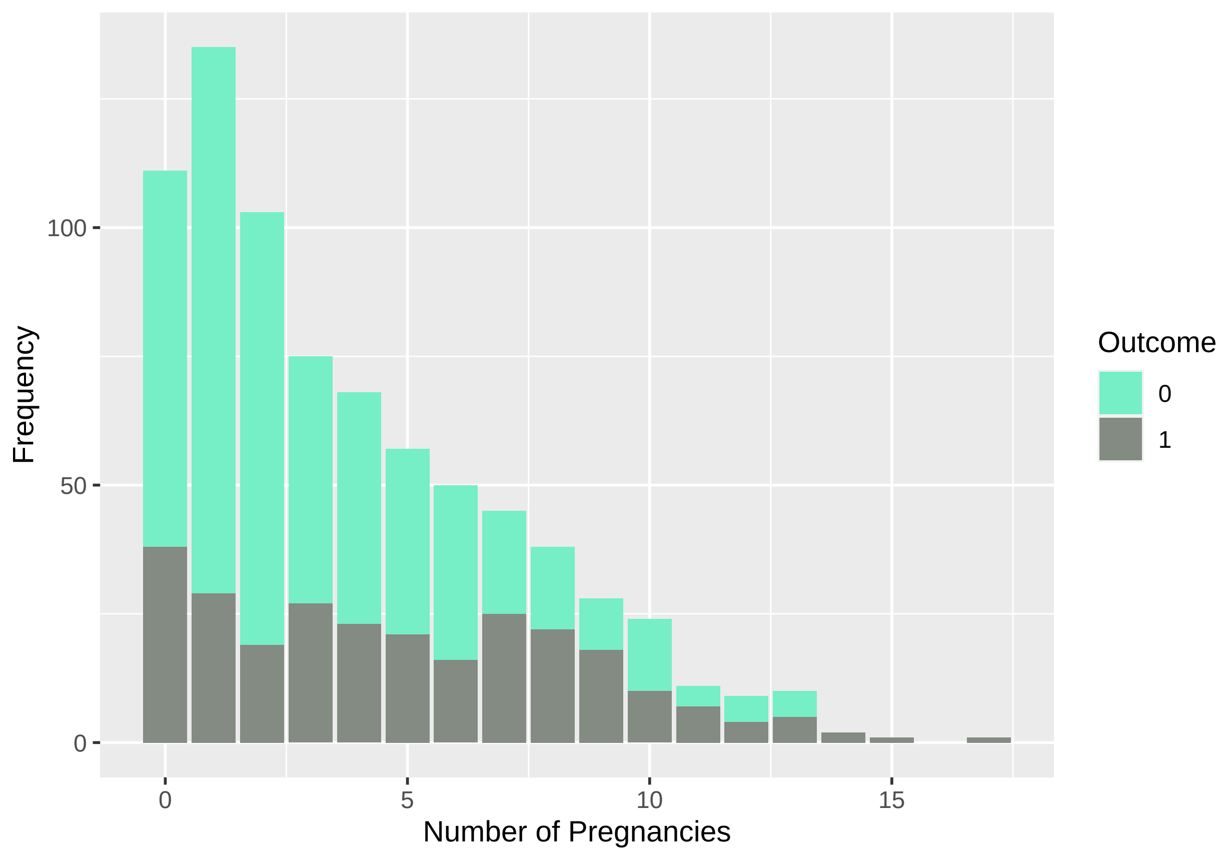
Let's see if our dataset is prepared right now.

```r
#### now let's attach new variables to our dataset ####
attach(Diabetes, warn.conflicts = FALSE)
head(Diabetes)
```

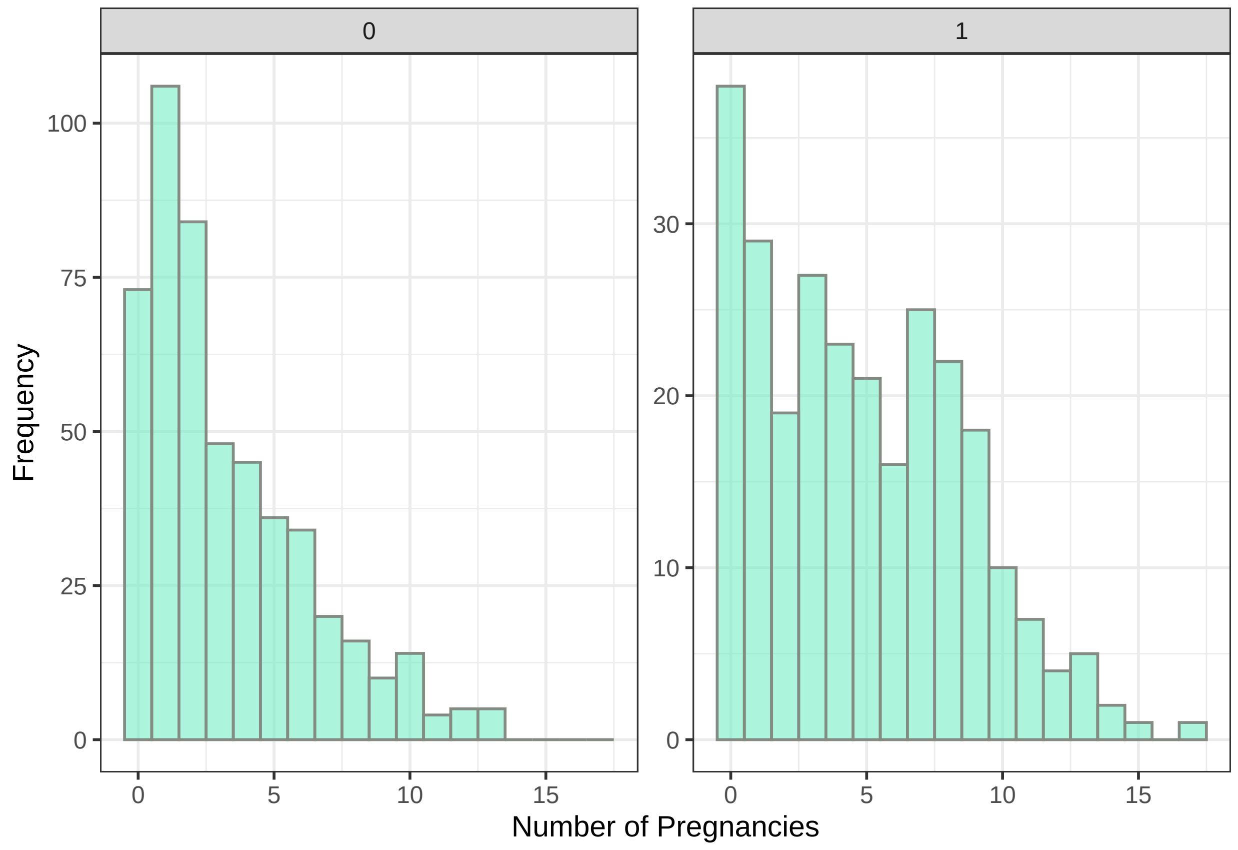
```
##   Pregnancies Glucose BloodPressure SkinThickness Insulin  BMI
## 1           6     148            72            35     125 33.6
## 2           1      85            66            29     125 26.6
## 3           8     183            64            29     125 23.3
## 4           1      89        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

2.3 Exploratory and Data Analysis

```r
ggplot(Diabetes, aes(x = Pregnancies, fill = factor(Outcome))) +
  geom_bar() +
  scale_fill_manual(values = c("#76EEC6", "#838B83"), name = "Outcome") +
  labs(x = "Number of Pregnancies", y = "Frequency")
```



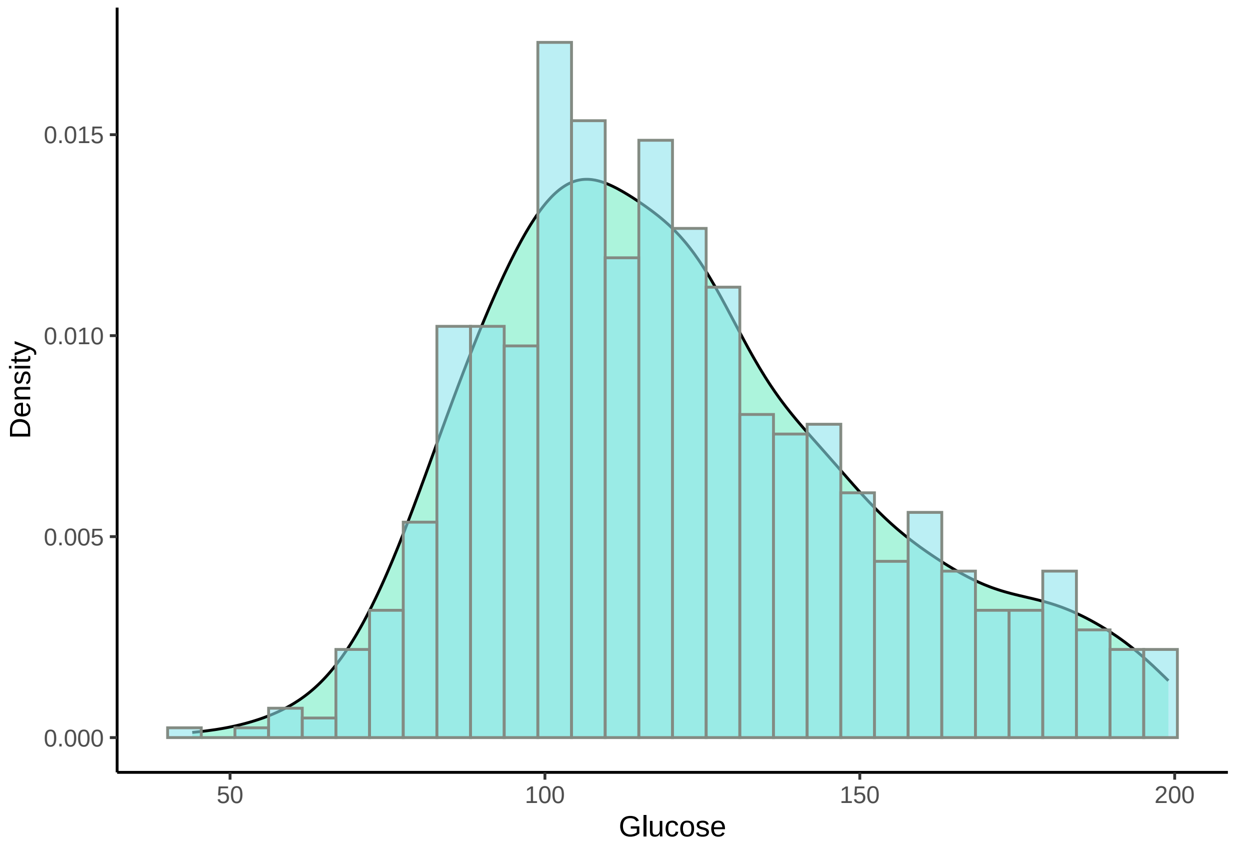
```r
ggplot(Diabetes, aes(x = Pregnancies)) +
  geom_histogram(fill = '#76EEC6', color = '#838B83', alpha = 0.6,
                 binwidth = 1, position = "dodge") +
  facet_wrap(~Outcome, ncol = 2, scales = "free_y") +
  labs(x = "Number of Pregnancies", y = "Frequency") +
  scale_fill_manual(values = c("#76EEC6", "#838B83"), name = "Outcome") +
  theme_bw()
```



From these charts, we can see that the majority of women have less than 5 pregnancies, regardless of their diabetic status. However, among women with 6 or more pregnancies, the proportion of diabetic women is higher than non-diabetic women. This suggests that having a higher number of pregnancies may increase the risk of developing diabetes.

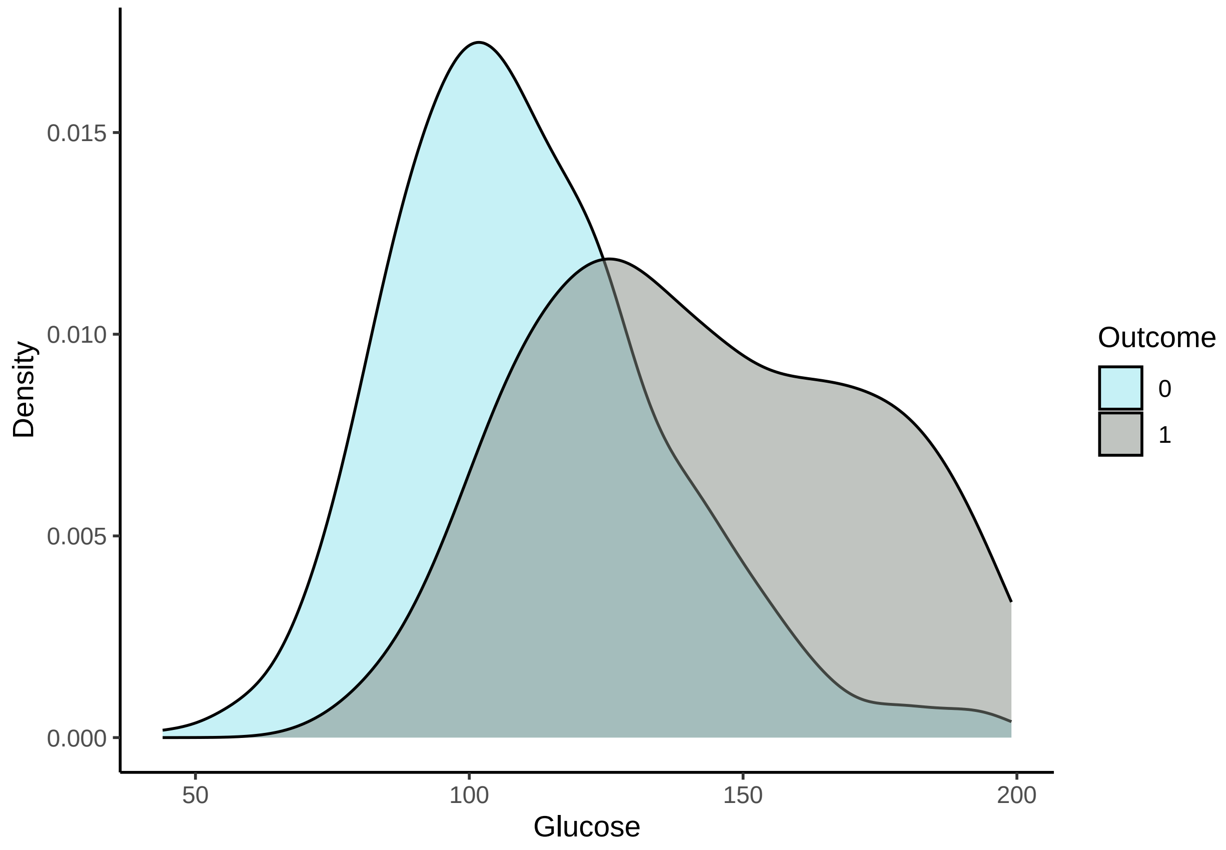
```
ggplot(Diabetes, aes(x = Glucose)) +
  geom_density(fill='#76EEC6', alpha=0.6) +
  geom_histogram(aes(y=..density..), fill='#8EE5EE', color='#838B83', alpha=0.6) +
  labs(x ="Glucose", y="Density") +
  theme_classic()
```

```
## ‘stat_bin()‘ using ‘bins = 30‘. Pick better value with ‘binwidth‘.
```



From the plot, we can see that the glucose levels in the dataset are mostly distributed around the 100-150 mg/dL range. The density plot shows that the probability density of glucose levels is highest at around 120 mg/dL and decreases as we move away from this value. The histogram shows that the majority of the observations fall in the range of 75-200 mg/dL. Additionally, the shape of the distribution appears to be slightly skewed to the right, which suggests that there may be some outliers or a long tail in the higher glucose values.

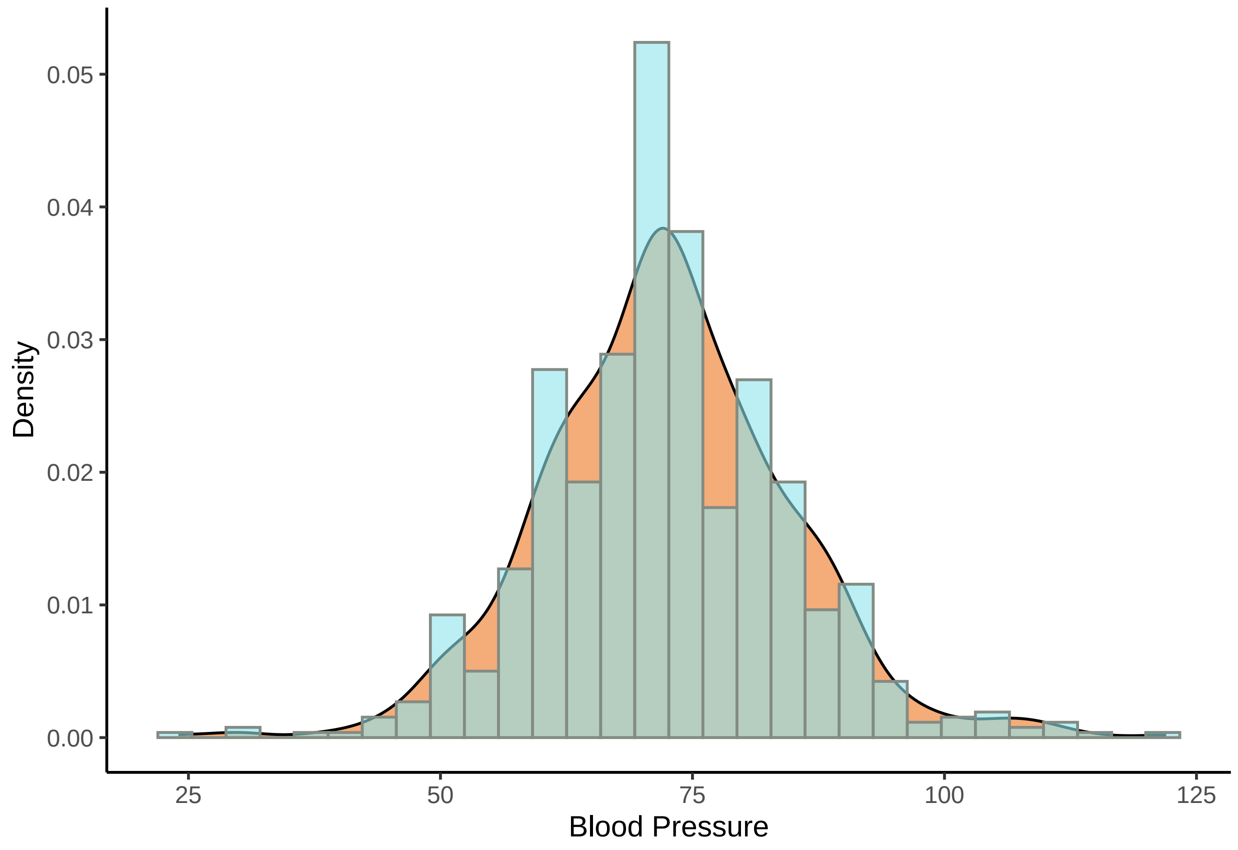
```r
ggplot(Diabetes, aes(x = Glucose, fill = factor(Outcome))) +
  geom_density(alpha = 0.5) +
  scale_fill_manual(values = c("#8EE5EE", "#838B83"), name = "Outcome") +
  labs(x = "Glucose", y = "Density") +
  theme_classic()
```



The chart shows that people with Outcome 1 (diabetes) have a higher density of glucose levels compared to those with Outcome 0 (no diabetes). It also shows that the distribution of glucose levels for Outcome 1 is more spread out than Outcome 0, indicating that there is more variation in glucose levels among people with diabetes.

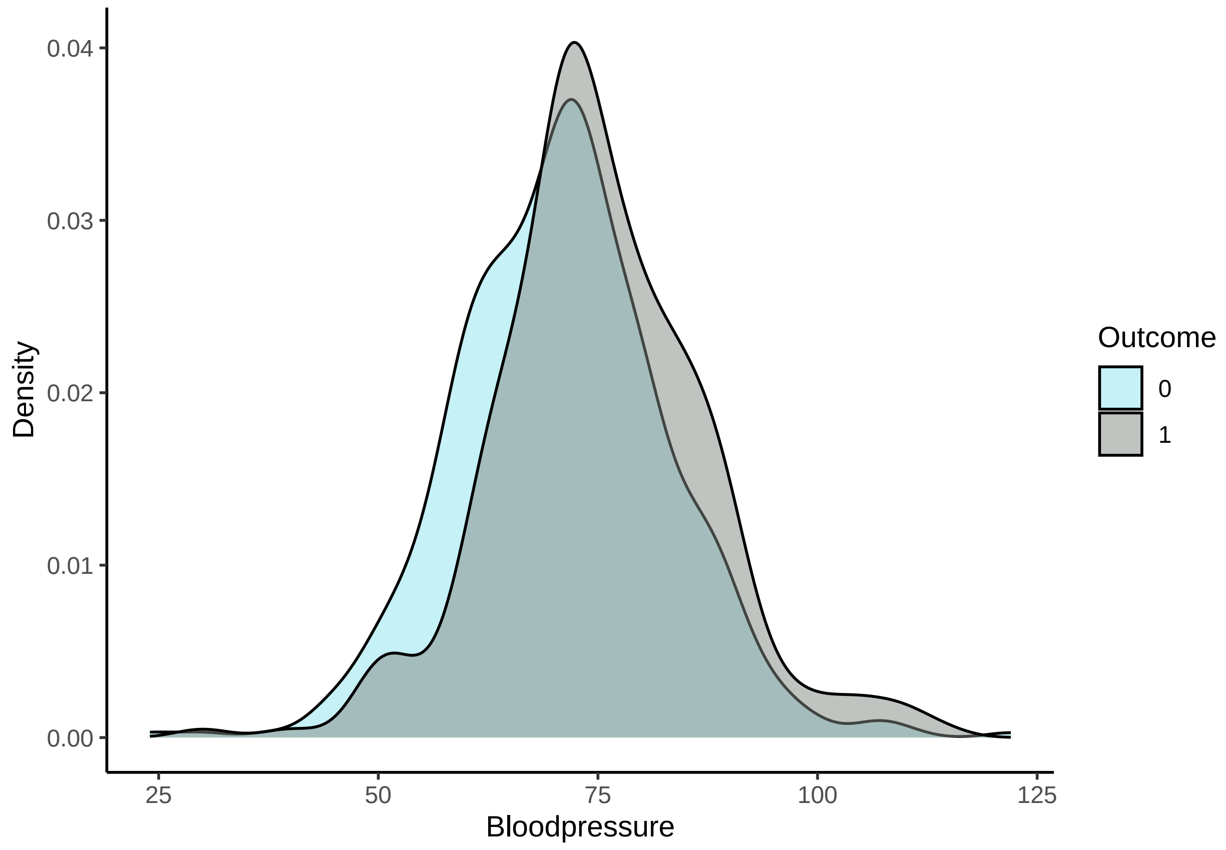
```r
ggplot(Diabetes, aes(x = BloodPressure)) +
  geom_density(fill='#EE7621', alpha=0.6) +
  geom_histogram(aes(y=..density..), fill='#8EE5EE', color='#838B83', alpha=0.6) +
  labs(x ="Blood Pressure", y="Density") +
  theme_classic()
```

```
## ‘stat_bin()‘ using ‘bins = 30‘. Pick better value with ‘binwidth‘.
```



From the plot, we can see that the blood pressure levels in the dataset are mostly distributed around the 60-90 mm Hg range. The density plot shows that the probability density of blood pressure levels is highest at around 70 mm Hg and decreases as we move away from this value. The histogram shows that the majority of the observations fall in the range of 60-100 mm Hg. Additionally, the shape of the distribution appears to be slightly skewed to the right, which suggests that there may be some outliers or a long tail in the higher blood pressure values.

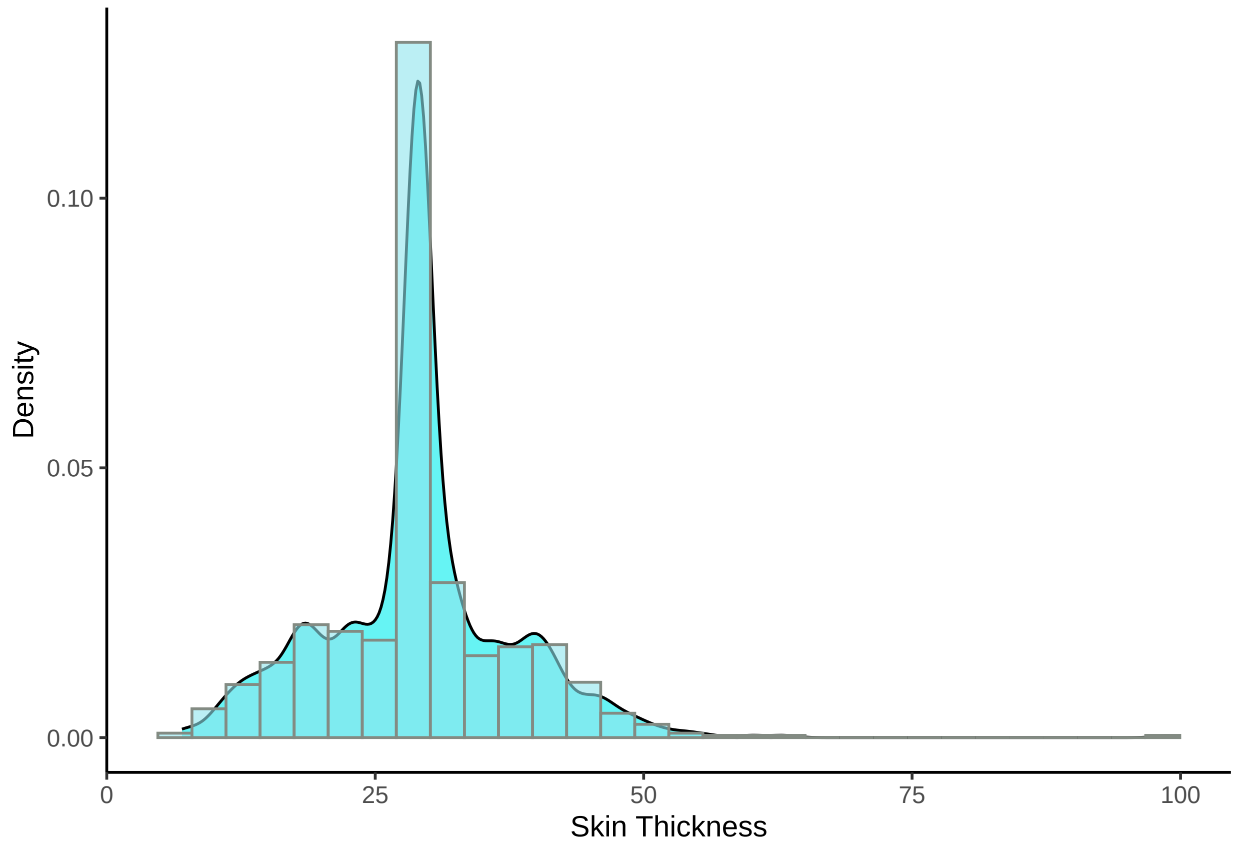
```
ggplot(Diabetes, aes(x = BloodPressure, fill = factor(Outcome))) +
  geom_density(alpha = 0.5) +
  scale_fill_manual(values = c("#8EE5EE", "#838B83"), name = "Outcome") +
  labs(x = "Bloodpressure", y = "Density") +
  theme_classic()
```



The density curve for non-diabetic patients (in blue) is centered around a lower blood pressure value compared to the curve for diabetic patients (in gray), which is more spread out and centered around a higher blood pressure value. We can also see that the density of non-diabetic patients blood pressure values is higher overall, which may indicate that non-diabetic patients are more likely to have blood pressure values clustered around a specific range.Also, we can see that the blood pressure levels in the dataset are mostly concentrated around the 60-80 mmHg range, and the density of blood pressure values for the non-diabetic group (blue) is slightly higher than the density for thediabetic group (gray). The density plot shows that the probability density of blood pressure levels is highest around 70-75 mmHg and decreases as we move away from this value.

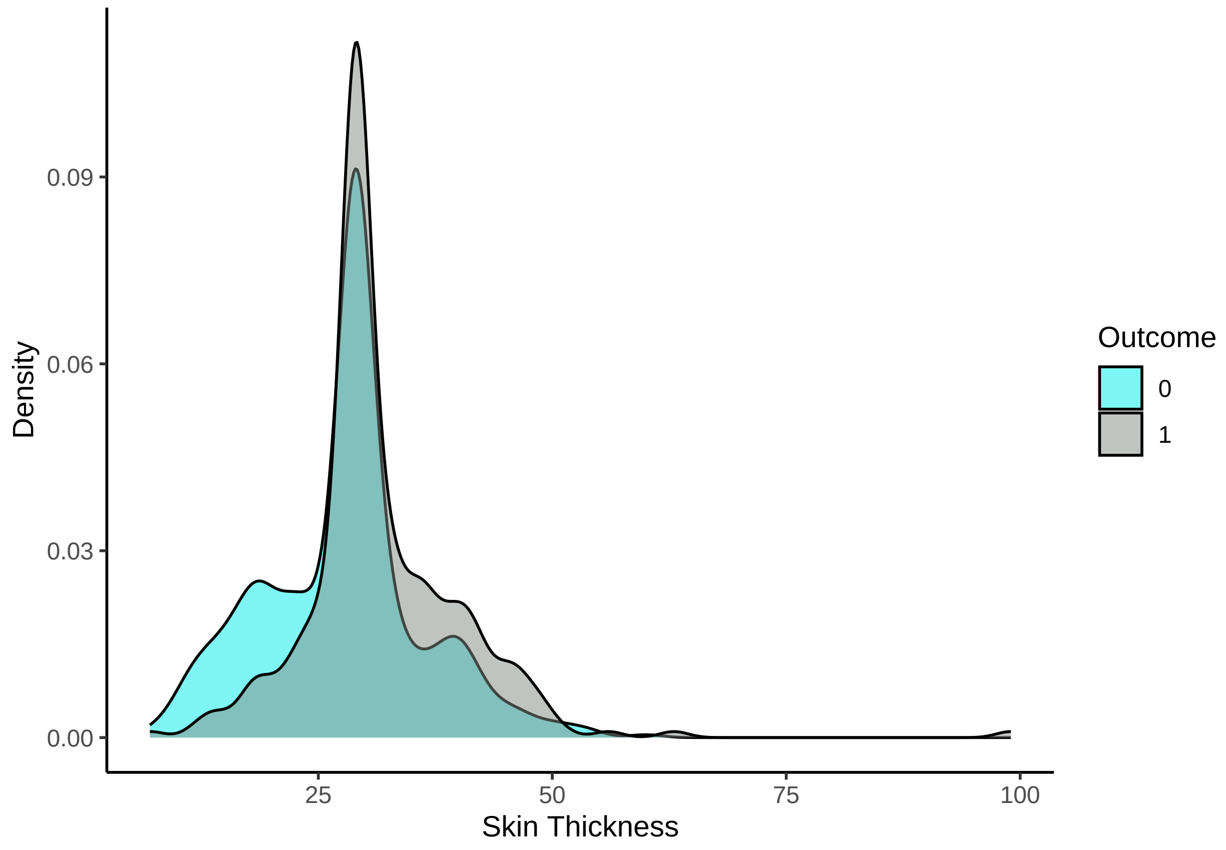
```r
ggplot(Diabetes, aes(x = SkinThickness)) +
  geom_density(fill='#00EEEE', alpha=0.6) +
  geom_histogram(aes(y=..density..), fill='#8EE5EE', color='#838B83', alpha=0.6) +
  labs(x ="Skin Thickness", y="Density") +
  theme_classic()
```

```
## ‘stat_bin()‘ using ‘bins = 30‘. Pick better value with ‘binwidth‘.
```



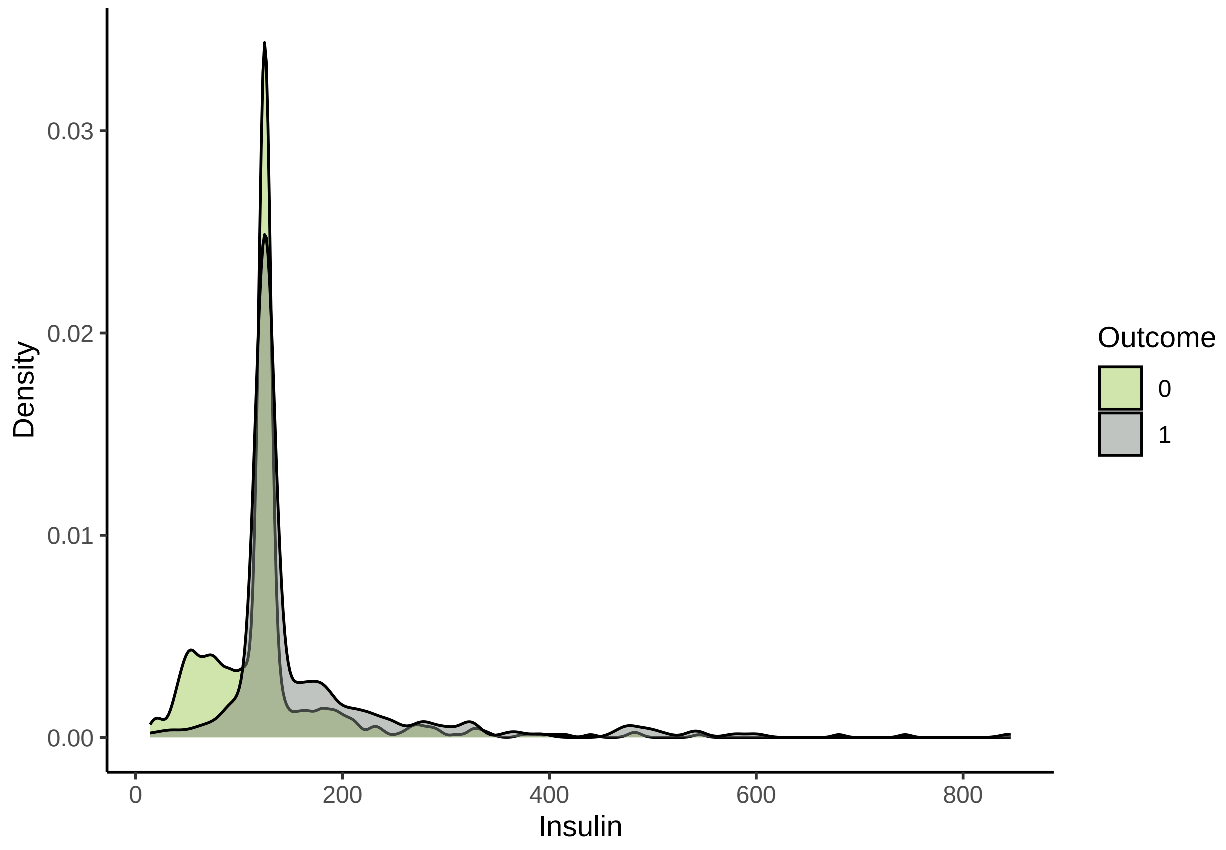
From the chart, we can see that skin thickness is generally concentrated between 20 and 35 mm, with a peak density around 30 mm. The histogram shows that the distribution is slightly skewed to the right, which suggests that there may be some outliers or a long tail in the higher skin thickness values.

```r
ggplot(Diabetes, aes(x = SkinThickness, fill = factor(Outcome))) +
  geom_density(alpha = 0.5) +
  scale_fill_manual(values = c("#00EEEE", "#838B83"), name = "Outcome") +
  labs(x = "Skin Thickness", y = "Density") +
  theme_classic()
```



We can also see that the density curves for the two outcome variables overlap considerably, which suggests that skin thickness may not be a strong predictor of the outcome. However, we can see that there is a slightly higher density of skin thickness values around 30 mm for the outcome variable 0 (non-diabetic) than for the outcome variable 1 (diabetic).

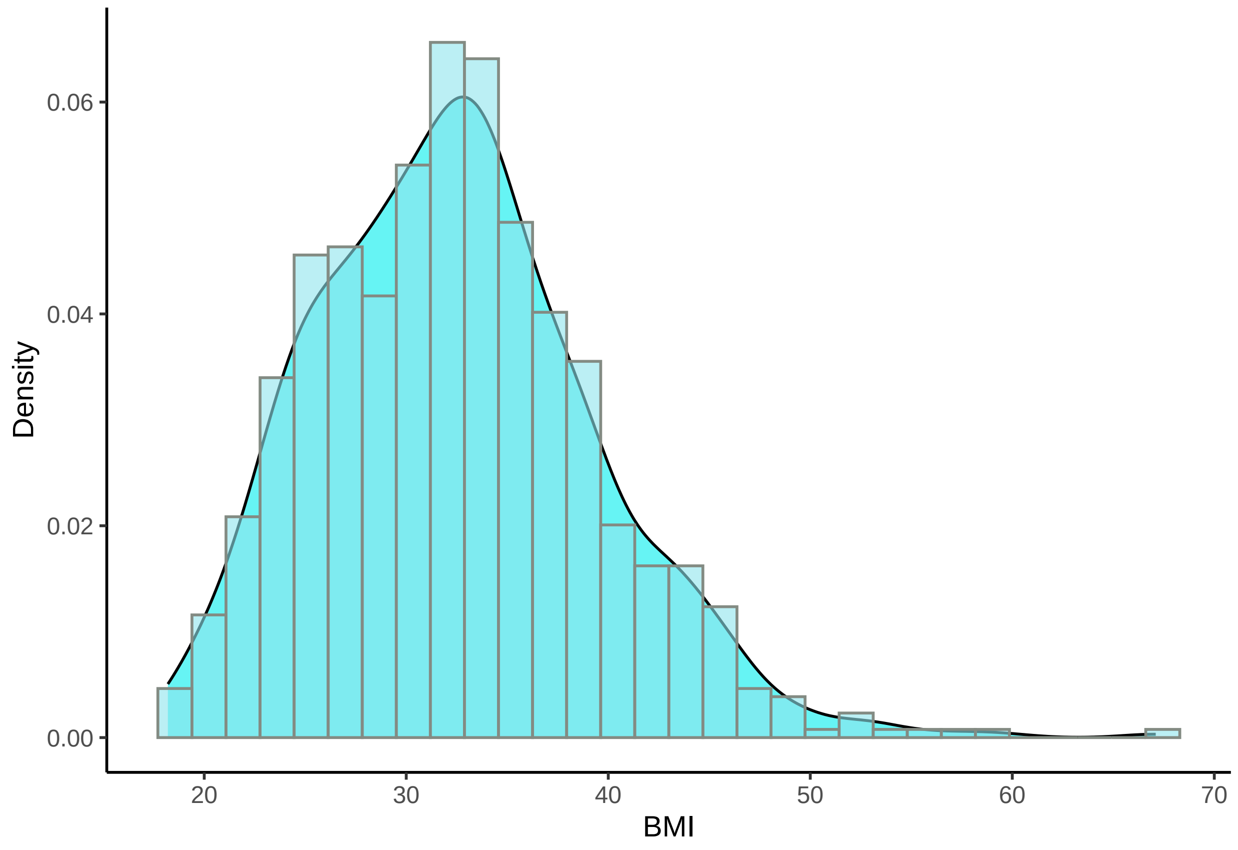
```r
ggplot(Diabetes, aes(x = Insulin, fill = factor(Outcome))) +
  geom_density(alpha = 0.5) +
  scale_fill_manual(values = c("#A2CD5A", "#838B83"), name = "Outcome") +
  labs(x = "Insulin", y = "Density") +
  theme_classic()
```



From the plot, we can see that the distribution of insulin levels in the Diabetes dataset differs between outcomes 0 and 1. The density plot shows that individuals with an outcome of 0 have a slightly higher probability density of insulin levels around 100-150 mIU/mL, whereas those with an outcome of 1 have a broader distribution with a peak at around 150 MIU/mL. The histogram indicates that the majority of individuals in the dataset have insulin levels in the range of 0-400 MIU/mL, with a long tail towards higher values. Overall, this plot suggests that insulin levels may be a useful predictor for diabetes outcomes, as individuals with different outcomes have distinct distributions of insulin levels.

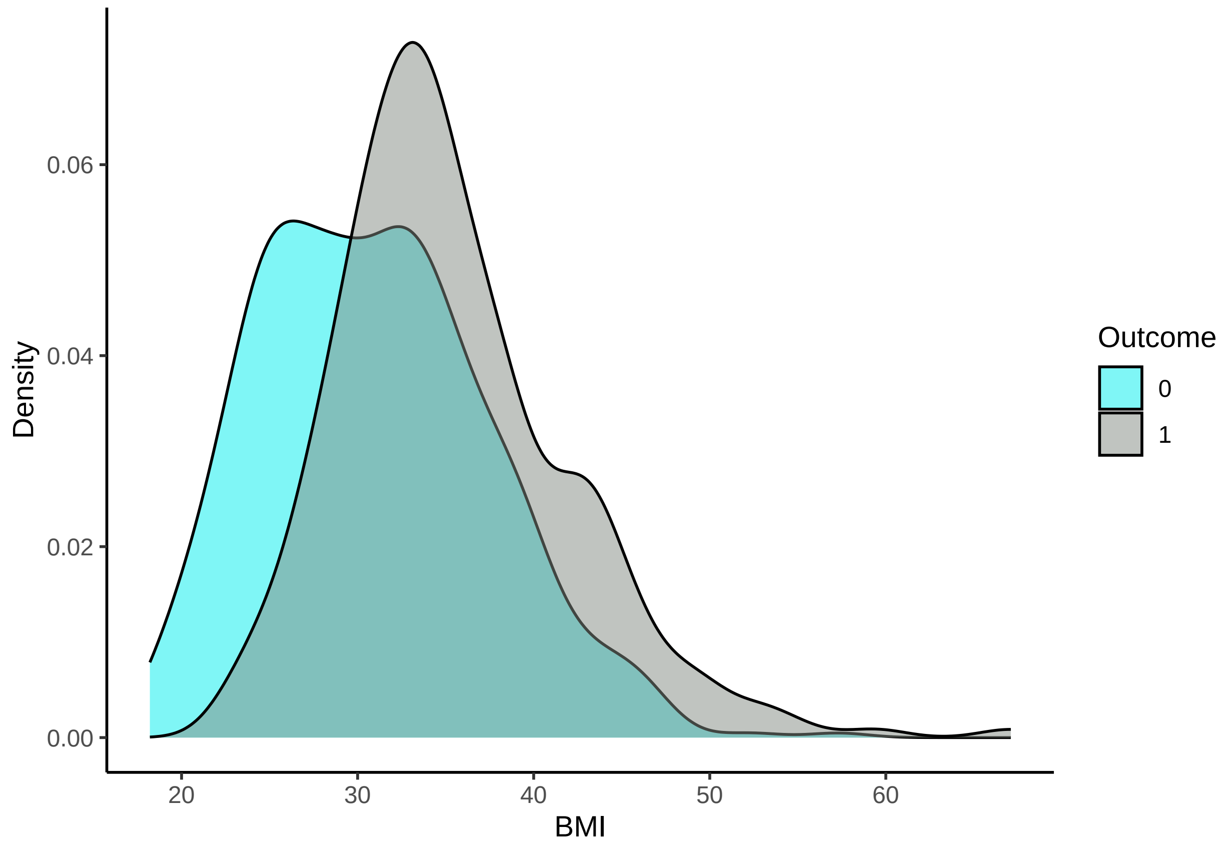
```
ggplot(Diabetes, aes(x = BMI)) +
  geom_density(fill='#00EEEE', alpha=0.6) +
  geom_histogram(aes(y=..density..), fill='#8EE5EE', color='#838B83', alpha=0.6) +
  labs(x ="BMI", y="Density") +
  theme_classic()
```

```
## ‘stat_bin()‘ using ‘bins = 30‘. Pick better value with ‘binwidth‘.
```



we can see that the BMI values in the dataset are mostly distributed around the 20-40 range. The density plot shows that the probability density of BMI values is highest around 30 and decreases as we move away from this value. The histogram shows that the majority of the observations fall in the range of 20-50. Additionally, the shape of the distribution appears to be slightly skewed to the right, which suggests that there may be some outliers or a long tail in the higher BMI values.

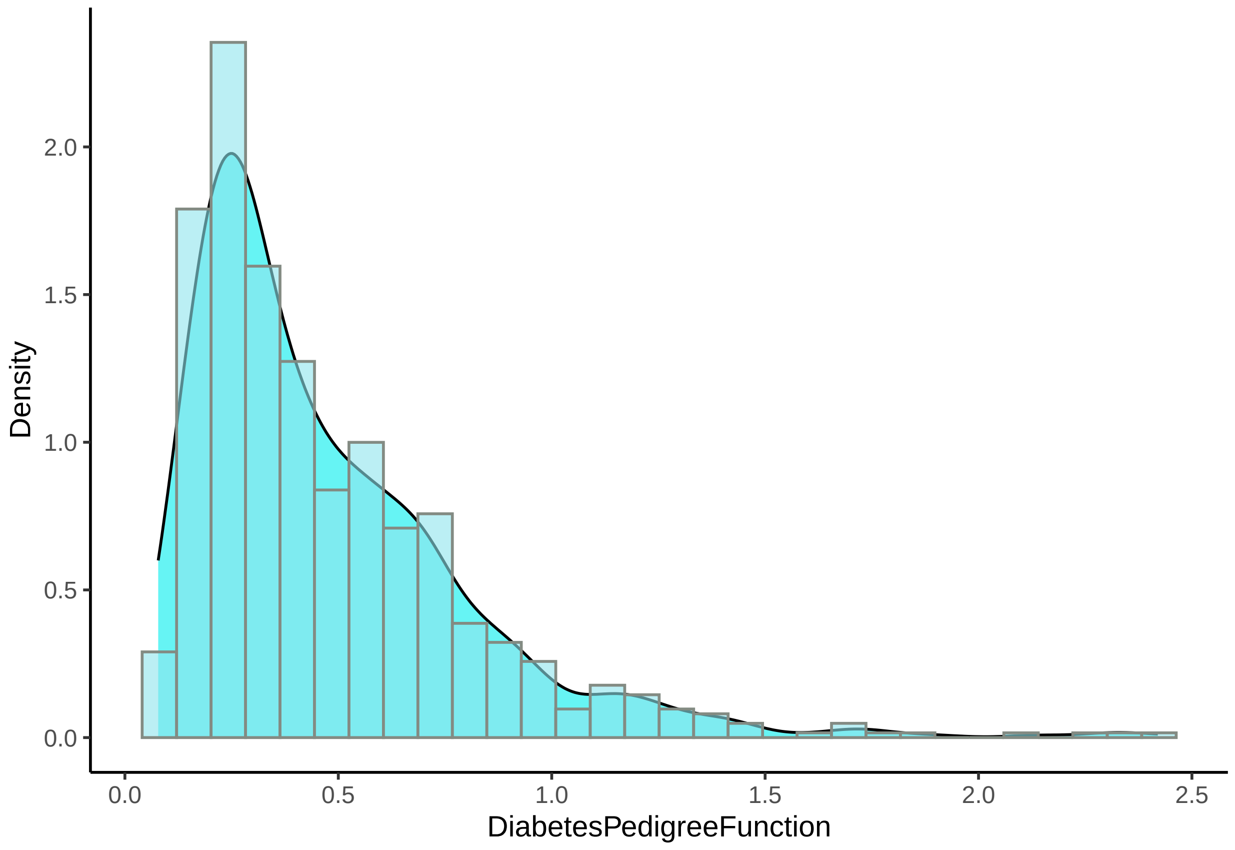
```
ggplot(Diabetes, aes(x = BMI, fill = factor(Outcome))) +
  geom_density(alpha = 0.5) +
  scale_fill_manual(values = c("#00EEEE", "#838B83"), name = "Outcome") +
  labs(x = "BMI", y = "Density") +
  theme_classic()
```



The density plot suggests that the BMI distribution is bimodal, with one peak around 25 and another around 35. Additionally, the density of the BMI values is higher for Outcome 1 (positive diabetes diagnosis) than for Outcome 0 (negative diabetes diagnosis) in the higher BMI range, indicating that individuals with higher BMIs are more likely to be diagnosed with diabetes. The shape of the distribution also appears to be slightly skewed to the right, which suggests that there may be some outliers or a long tail in the higher BMI values. Overall, this plot suggests that BMI is an important predictor of diabetes diagnosis, with higher BMI values associated with a greater likelihood of being diagnosed with diabetes.

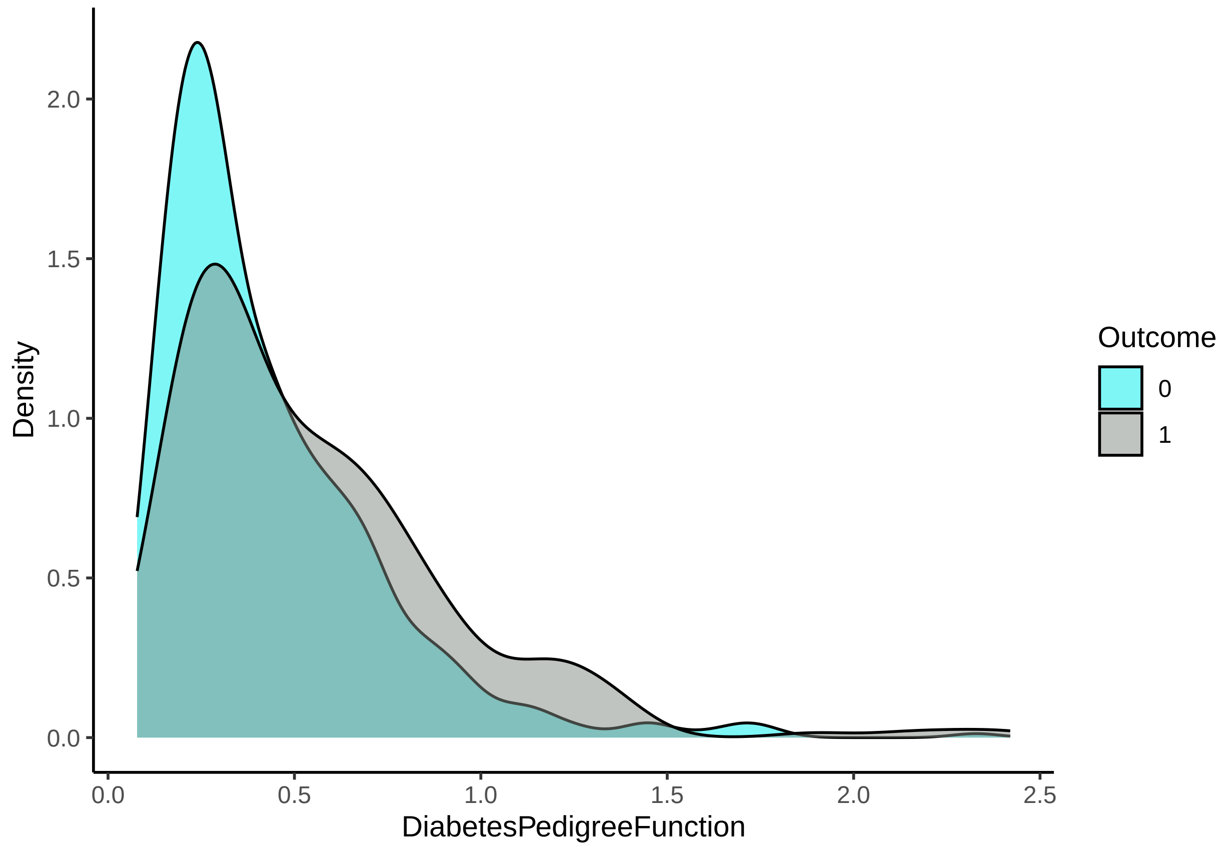
```
ggplot(Diabetes, aes(x = DiabetesPedigreeFunction)) +
  geom_density(fill='#00EEEE', alpha=0.6) +
  geom_histogram(aes(y=..density..), fill='#8EE5EE', color='#838B83', alpha=0.6) +
  labs(x ="DiabetesPedigreeFunction", y="Density") +
  theme_classic()
```

```
## ‘stat_bin()‘ using ‘bins = 30‘. Pick better value with ‘binwidth‘.
```



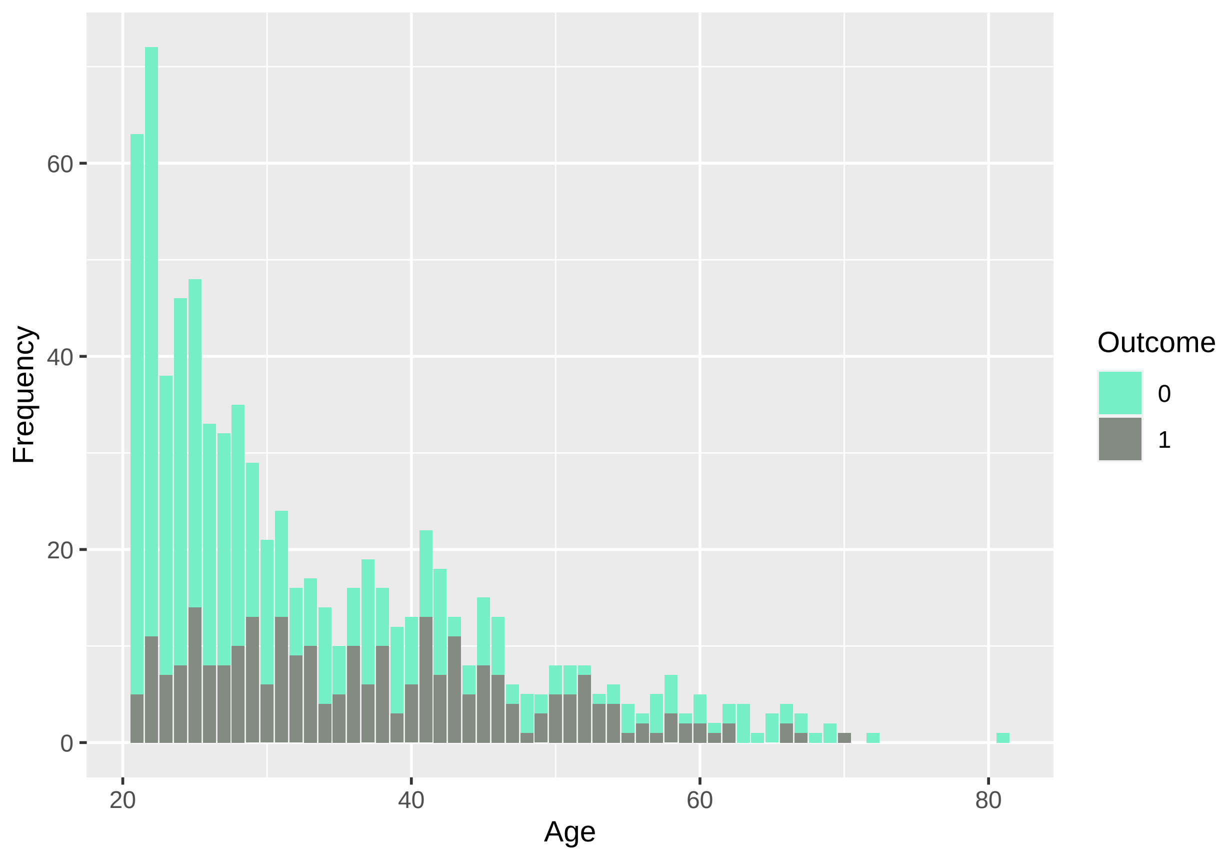
From this plot, we can observe that the DiabetesPedigreeFunction variable is approximately normally distributed with a slight positive skew. The majority of the observations lie between 0 and 1, and there is a small bump in the densityplot around the value of 0.4. The Diabetes Pedigree Function is a genetic score that estimates the risk of developing diabetes based on the family history of the patient. The bimodal shape of the density plot might indicate the presence of two sub-populations with different levels of genetic predisposition to diabetes. However,further analysis is required to confirm this interpretation.

```
ggplot(Diabetes, aes(x = DiabetesPedigreeFunction, fill = factor(Outcome))) +
  geom_density(alpha = 0.5) +
  scale_fill_manual(values = c("#00EEEE", "#838B83"), name = "Outcome") +
  labs(x = "DiabetesPedigreeFunction", y = "Density") +
  theme_classic()
```



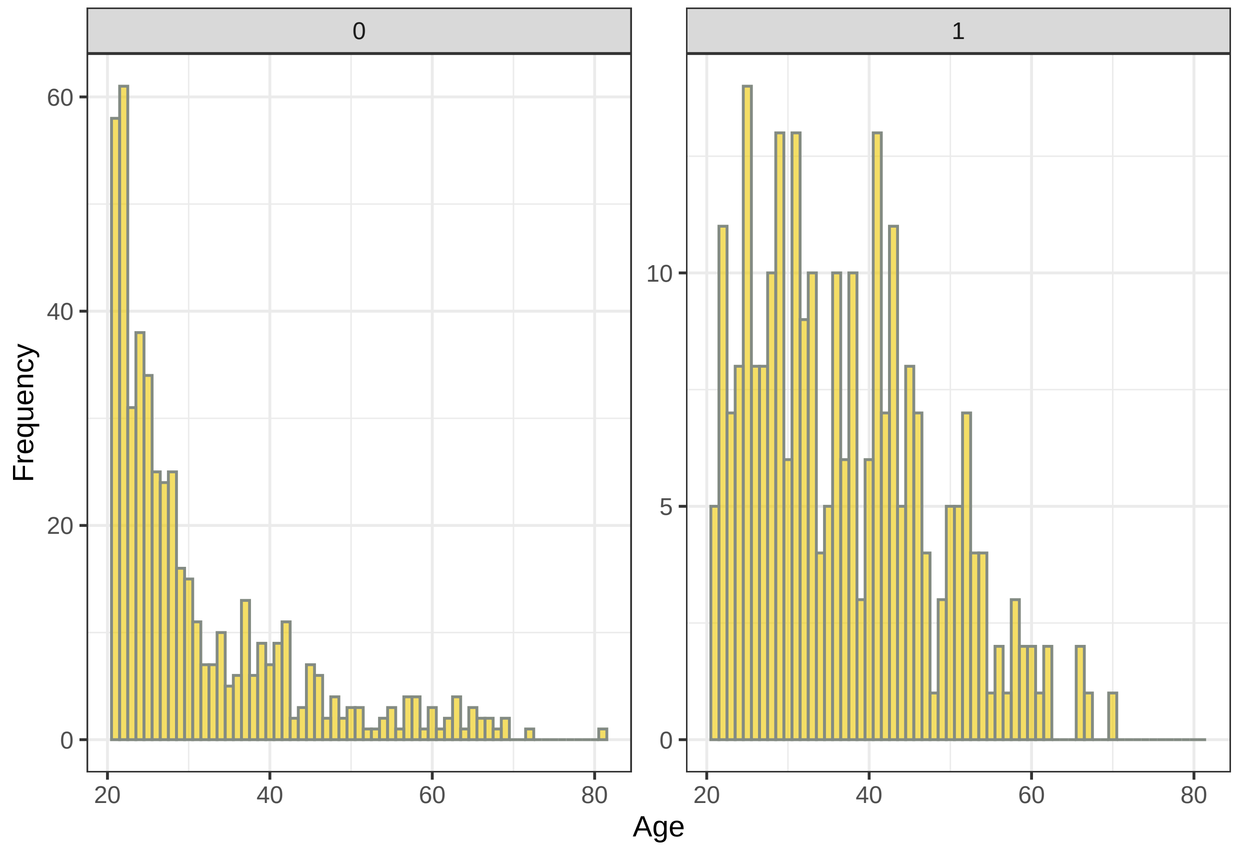
By looking at this plot, we can see that the DiabetesPedigreeFunction values for Outcome 0(Non-Diabetes) are mostly concentrated around 0 to 1, while for Outcome 1(Diabetes),the values are more widely spread and skewed towards the right, with a long tail. This plot can be useful in identifying the differences in the distribution of DiabetesPedigreeFunction between people with and without diabetes.we can see that the distribution of DiabetesPedigreeFunction is shifted towards higher values in the group with diabetes.

```
ggplot(Diabetes, aes(x = Age, fill = factor(Outcome))) +
  geom_bar() +
  scale_fill_manual(values = c("#76EEC6", "#838B83"), name = "Outcome") +
  labs(x = "Age", y = "Frequency")
```



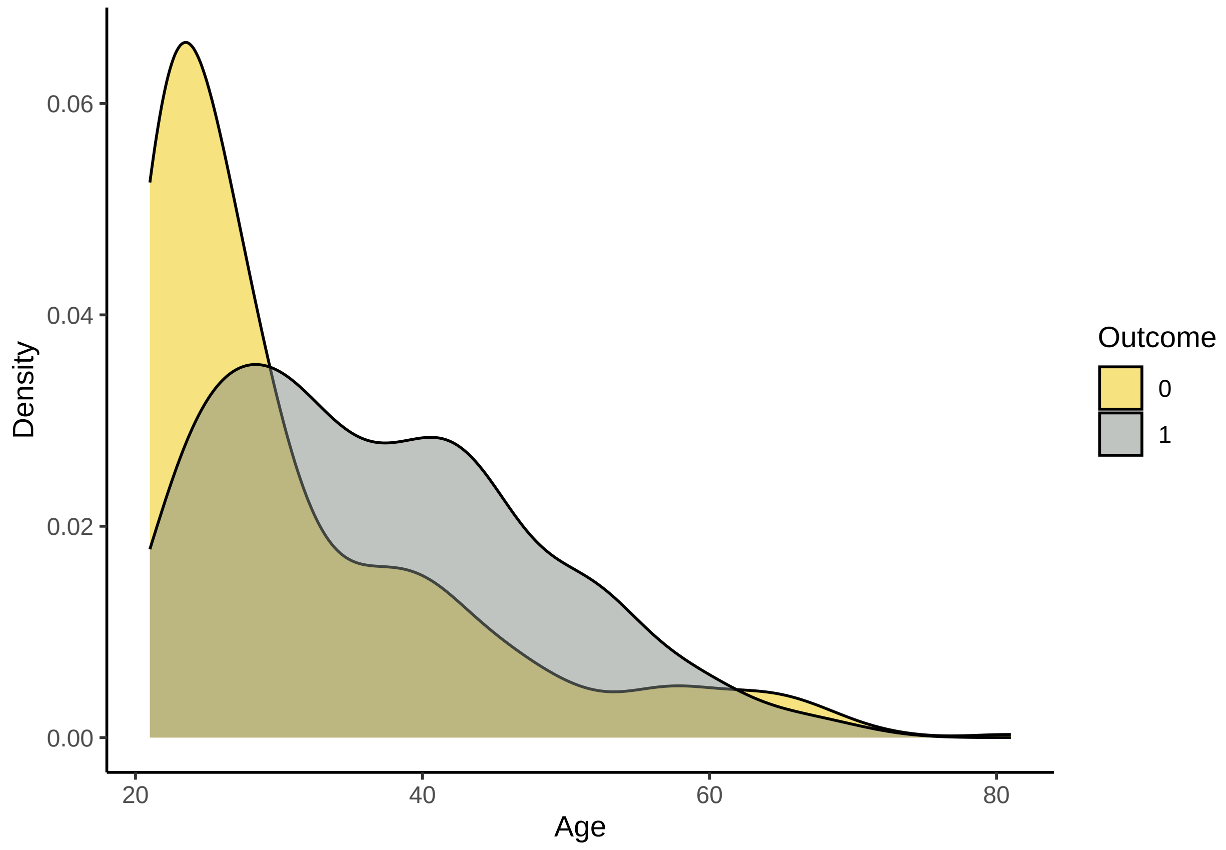
By looking at the plot, we can see that the frequency of individuals with diabetes increases with age, peaking in the age group of 50-59 years, and then gradually decreasing in older age groups. We can also see that the frequency of individuals without diabetes remains relatively stable across different age groups.

```
ggplot(Diabetes, aes(x = Age)) +
  geom_histogram(fill = '#EEC900', color = '#838B83', alpha = 0.6,
                 binwidth = 1, position = "dodge") +
  facet_wrap(~Outcome, ncol = 2, scales = "free_y") +
  labs(x = "Age", y = "Frequency") +
  scale_fill_manual(values = c("#EEC900", "#838B83"), name = "Outcome") +
  theme_bw()
```



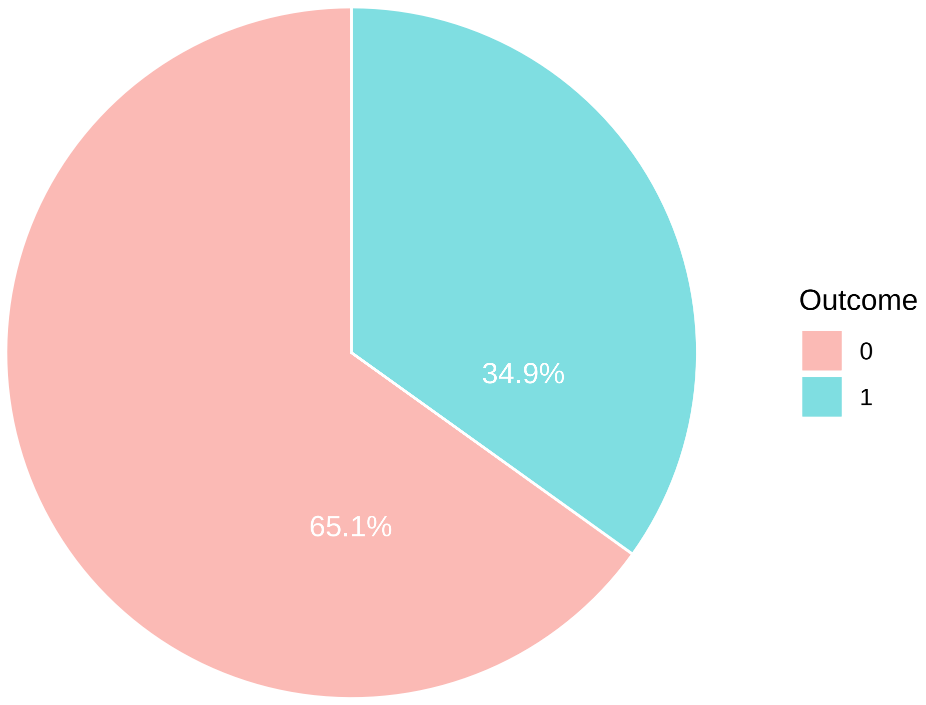
The histogram plot shows the distribution of age for people with and without diabetes in the dataset. The plot suggests that people with diabetes tend to be older than those without diabetes, with a peak in the 50-60 age range. This is consistent with the well-known fact that diabetes is more common in older age groups.The plot also shows that the age distribution for people without diabetes is roughly normal, with a peak in the 25-30 age range. This is consistent with the general population distribution of age.The differences in age distribution between people with and without diabetes are visually clear in the plot, which highlights the utility of this type of visualization for identifying potential risk factors for a given condition or disease.

```
ggplot(Diabetes, aes(x = Age, fill = factor(Outcome))) +
  geom_density(alpha = 0.5) +
  scale_fill_manual(values = c("#EEC900", "#838B83"), name = "Outcome") +
  labs(x = "Age", y = "Density") +
  theme_classic()
```



This plot shows the density distribution of age for individuals with and without diabetes. The plot suggests that the distribution of age for people with diabetes is slightly shifted to the right compared to those without diabetes. It also shows that the density of people with diabetes is higher in the age range of 40-60 years old, while the density of people without diabetes is higher in the age range of 20-30 years old.

```
Diabetes %>%
  group_by(Outcome) %>%
  summarise(n = n()) %>%
  mutate(Percentage = round(n/sum(n)*100, 1)) %>%
  ggplot(aes(x="", y=n, fill = factor(Outcome))) +
  geom_bar(width = 1, color = "white", alpha = 0.5, stat = "identity") +
  coord_polar("y", start=0) +
  labs(fill ="Outcome", x="", y="") +
  theme_void() +
  geom_text(aes(y = n/1.3, label = paste0(Percentage, "%")), color = "white", size = 4)
```



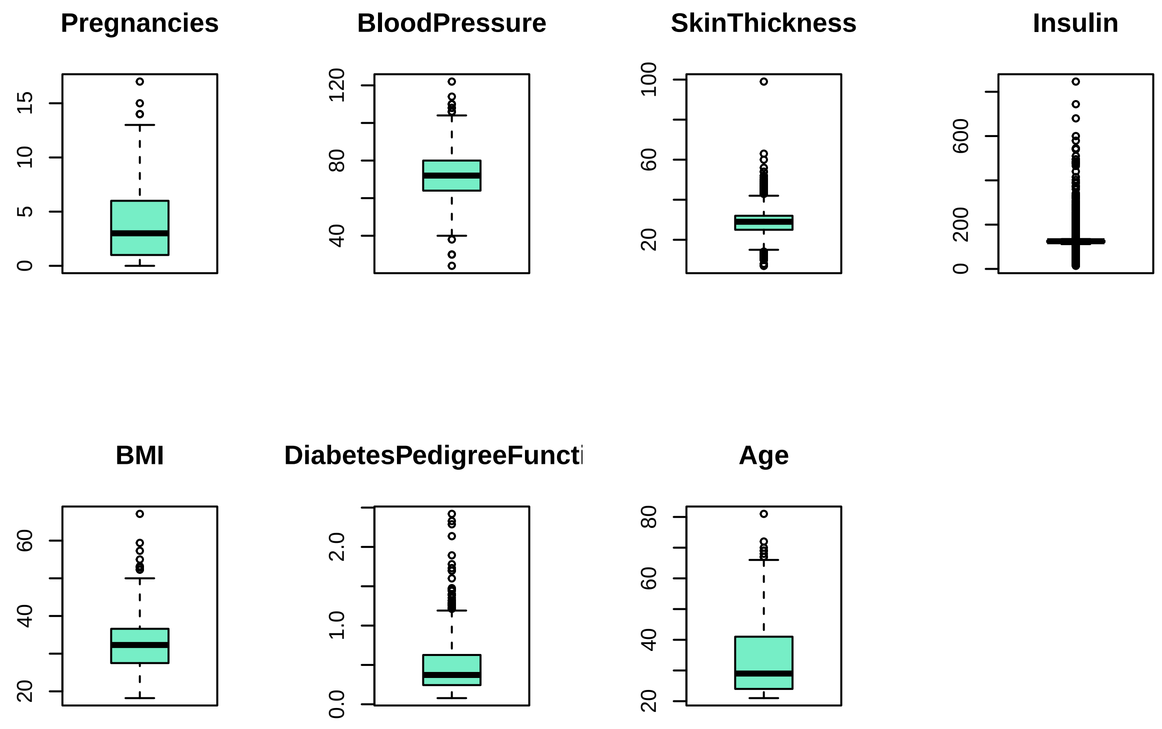
The plot displays the distribution of the Outcome variable in the Diabetes dataset. The Outcome variable consists of two levels: 0 indicating no diabetes and 1 indicating diabetes. According to the plot, out of the total 768 observations, 500 (65.1%) do not have diabetes, while 268 (34.9%) have diabetes. As evident, the number of non-diabetic instances surpasses the diabetic ones, indicating an imbalanced response variable. To address this issue, we can employ various methods such as collecting more data, under-sampling, or applying class weighting to the diabetic instances.

Now let's check out outliers. As you can see the all of our independent variables have outliers except Glucose. We can conclude these information from boxplots too.

```
#### Let's check outliers for Diabetes Dataset ####
check_outlier = function(x) {
  q1 = quantile(x, 0.25)
  q3 = quantile(x, 0.75)
  iqr = q3 - q1
  lower = q1 - 1.5 * iqr
  upper = q3 + 1.5 * iqr
  outliers = x[x < lower | x > upper]
  if (length(outliers) > 0) {
    return(paste(':', TRUE,"| Number of outliers:", length(outliers)))
  } else {
    return(paste(':', FALSE,"| Number of outliers:", length(outliers)))
  }
}

for (col in names(Diabetes[-9])) {
  print(paste(col, check_outlier(Diabetes[[col]])))
}
```

```
## [1] "Pregnancies : TRUE | Number of outliers: 4"
## [1] "Glucose : FALSE | Number of outliers: 0"
## [1] "BloodPressure : TRUE | Number of outliers: 14"
## [1] "SkinThickness : TRUE | Number of outliers: 87"
## [1] "Insulin : TRUE | Number of outliers: 346"
## [1] "BMI : TRUE | Number of outliers: 8"
## [1] "DiabetesPedigreeFunction : TRUE | Number of outliers: 29"
## [1] "Age : TRUE | Number of outliers: 9"
```

```
par(mfrow=c(2,4))
for (i in c(1 , 3:8)) {
  boxplot(Diabetes[,i], main=names(Diabetes)[i] , col = '#76EEC6')
}
```



We recognized which variables have missing values. Now it's time to handle these missing values.We can consider different approaches for this issue such as log transformation, Winsorize, and IQR. We considered log transformation for Age variable, winsorize approach for Pregnancies and DiabetesPedigreeFunction, and IQR for rest of the other variables.

```
####Using Log-transformtion for Age variable####
Diabetes$Age = log(Diabetes$Age)
```

We Considered winsorized approach for Pregnancies and DiabetesPedigreeFunction. We used other approaches but they have some problems.When we applied Log-Transformation and IQR methods some values of these two variables changed to negetaive.

```r
####using Winsorize approach to handle outliers for Pregnancies and DiabetesPedigreeFunction####

for (i in c(1 , 7)){
  pctiles = quantile(Diabetes[,i], probs = c(0.25, 0.75), na.rm = TRUE)
  Diabetes[,i] = Diabetes[,i]
  Diabetes[,i][Diabetes[,i] < pctiles[1]] = pctiles[1]
  Diabetes[,i][Diabetes[,i] > pctiles[2]] = pctiles[2]
}
```

Let's apply IQR method for the rest of variables.First we detect outliers. In this method we replaced the values which are bigger than third quantile with it's third quantile plus 1.5 times the inter quantile (the difference between first and third quantile). Also we replaced the values which are lower than first quantile with first quantile minus 1.5 times the inter quantile.

```r
####using IQR method to replace outliers####
quantiles = c(0.25, 0.75)
```

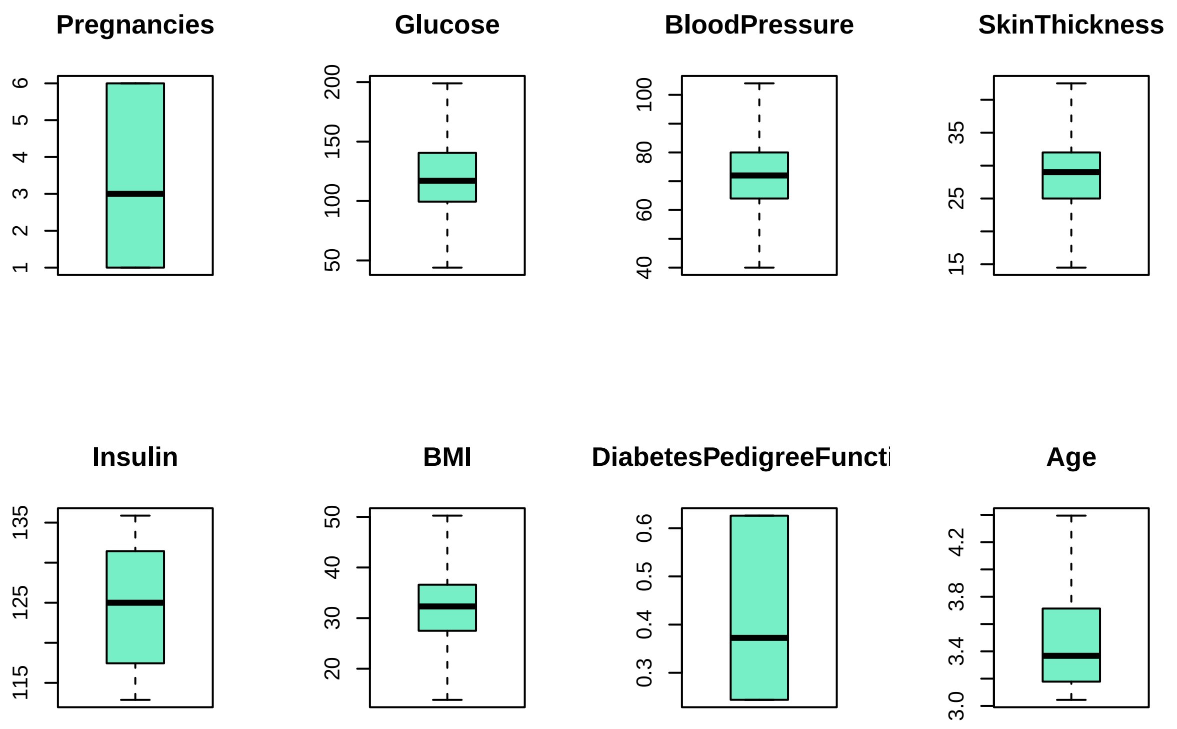
```r
# lets check which instances are outliers logically.
check_outlier2 = function(x) {
  q1 = quantile(x, 0.25)
  q3 = quantile(x, 0.75)
  iqr = q3 - q1
  lower = q1 - 1.5 * iqr
  upper = q3 + 1.5 * iqr
  outliers = x[x < lower | x > upper]
  if (length(outliers) > 0) {
    return (TRUE)
  } else {
    return(FALSE)
  }
}

for (col in c("Insulin","BloodPressure", "SkinThickness", "BMI")) {
  if (check_outlier2(Diabetes[[col]]) == TRUE){
    col_quantiles = quantile(Diabetes[[col]], quantiles)
    iqr = diff(col_quantiles)

    Diabetes[[col]][Diabetes[[col]] > col_quantiles[2]] = col_quantiles[2] + 1.5 * iqr
    Diabetes[[col]][Diabetes[[col]] < col_quantiles[1]] = col_quantiles[1] - 1.5 * iqr
  }
}

attach(Diabetes)
```

```
## The following objects are masked from Diabetes (pos = 3):
##
##     Age, BloodPressure, BMI, DiabetesPedigreeFunction, Glucose,
##     Insulin, Outcome, Pregnancies, SkinThickness
## The following objects are masked from Diabetes (pos = 4):
##
##     Age, BloodPressure, BMI, DiabetesPedigreeFunction, Glucose,
##     Insulin, Outcome, Pregnancies, SkinThickness
```

```
head(Diabetes)
```

```
##   Pregnancies Glucose BloodPressure SkinThickness Insulin   BMI
## 1           6     148            72          42.5 125.000 33.60
## 2           1      85            66          29.0 125.000 13.85
## 3           6     183            64          29.0 125.000 13.85
## 4           1      89            66          14.5 112.875 28.10
## 5           1     137            40          42.5 135.875 50.25
## 6           5     116            74          29.0 125.000 13.85
##   DiabetesPedigreeFunction      Age Outcome
## 1                  0.62625 3.912023       1
## 2                  0.35100 3.433987       0
## 3                  0.62625 3.465736       1
## 4                  0.24375 3.044522       0
## 5                  0.62625 3.496508       1
## 6                  0.24375 3.401197       0
```

```
par(mfrow=c(2,4))
for (i in c(1:8)) {
  boxplot(Diabetes[,i], main=names(Diabetes)[i] , col = '#76EEC6')
}
```



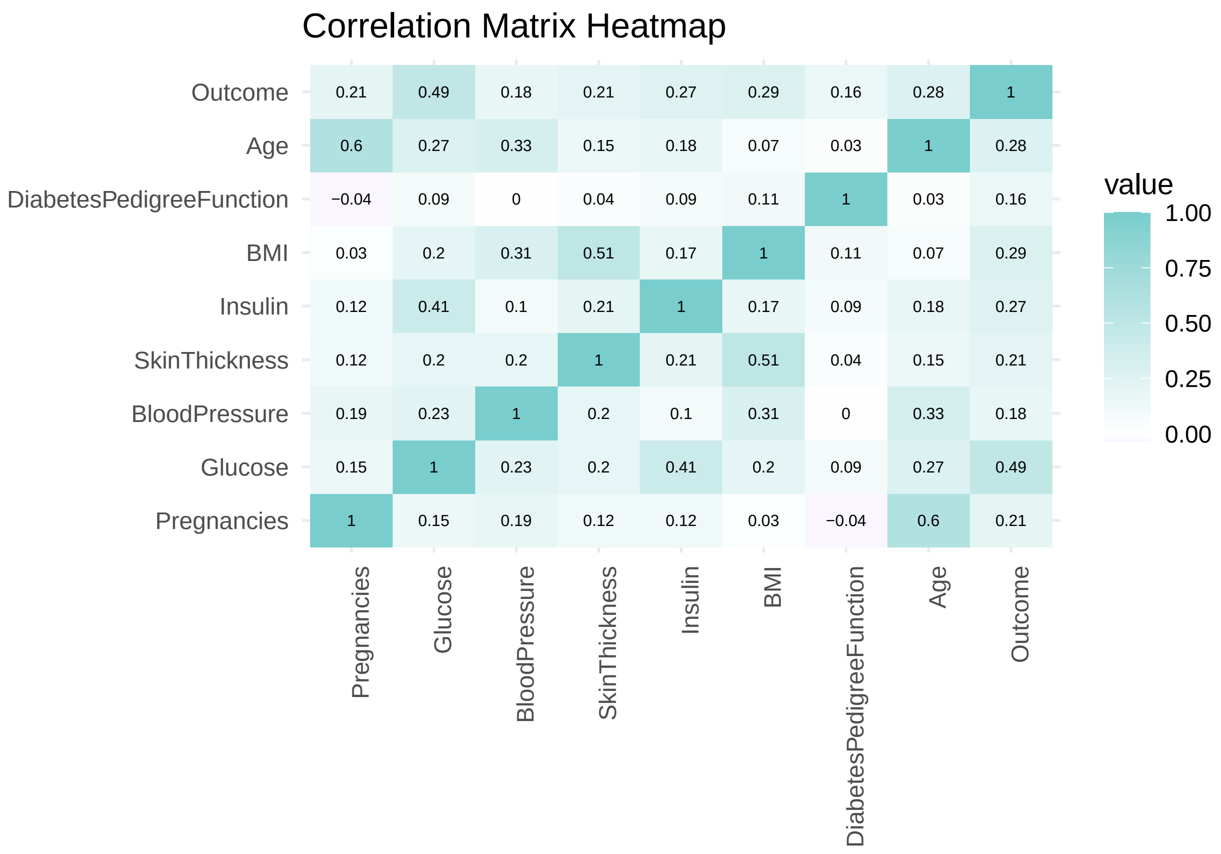
Now we can draw a heatmap to check the relationship between our response and the other variables. As you can see Outcome has the most relationship with Age, Glucose, BMI, and Insulin.

```
#### Let's plot a heatmap in order to check the correlations ####
cor_matrix = cor(Diabetes)
```

```
#### Reshape the correlation matrix into a dataframe ####
cor_df = as.data.frame(as.table(cor_matrix))
names(cor_df) = c("Var1", "Var2", "value")
```

```
####Add Pearson correlation coefficient to each cell####
cor_df$corr_coef = round(cor_df$value, 2)
```

```
####Create a heatmap####
ggplot(cor_df, aes(Var1, Var2, fill = value, label = corr_coef)) +
  geom_tile() +
  scale_fill_gradient2(low = "blue", mid = "white", high = "#79CDCD", midpoint = 0) +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  labs(title = "Correlation Matrix Heatmap", x = NULL, y = NULL) +
  geom_text(color = "black" , size = 2)
```



	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
Outcome	0.21	0.49	0.18	0.21	0.27	0.29	0.16	0.28	1
Age	0.6	0.27	0.33	0.15	0.18	0.07	0.03	1	0.28
DiabetesPedigreeFunction	−0.04	0.09	0	0.04	0.09	0.11	1	0.03	0.16
BMI	0.03	0.2	0.31	0.51	0.17	1	0.11	0.07	0.29
Insulin	0.12	0.41	0.1	0.21	1	0.17	0.09	0.18	0.27
SkinThickness	0.12	0.2	0.2	1	0.21	0.51	0.04	0.15	0.21
BloodPressure	0.19	0.23	1	0.2	0.1	0.31	0	0.33	0.18
Glucose	0.15	1	0.23	0.2	0.41	0.2	0.09	0.27	0.49
Pregnancies	1	0.15	0.19	0.12	0.12	0.03	−0.04	0.6	0.21

3 MODELS

In this section, we will fit our models to the Diabetes dataset. As mentioned earlier, we will consider five different algorithms: Logistic Regression, K-Nearest Neighbors (KNN), Naive Bayes, Linear Discriminant Analysis (LDA), and Quadratic Discriminant Analysis (QDA). For each model, we will explore different scenarios by applying feature selection methods and cross-validation techniques. Our goal is to identify the best model that is most suitable for our Diabetes dataset.

Before going through the details of our models there exist some important points that we need to discuss.

Firstly, as we are dealing with a classification problem, each model is accompanied by a confusion matrix. The confusion matrix consists of four different values. Firstly, we have the true positive, which represents the number of instances correctly predicted as having diabetes. Secondly, we have the false positive, which indicates the number of instances predicted as having diabetes but actually being non-diabetic. The third value is the false negative, which signifies the instances that have diabetes but are predicted as non-diabetic. Lastly, we have the true negative, which represents the number of instances correctly predicted as non-diabetic.

Secondly, while it is important to prioritize higher true positives, higher true negatives, and lower false positives, the most crucial aspect for us is minimizing the false negative rate. You may wonder why. In the context of medical diagnosis, a false negative implies that the model fails to identify individuals who actually have diabetes. The consequences of false negatives can be severe, as it could result in delayed or missed treatment, leading to potential health risks and complications for individuals requiring medical attention. Missing cases of diabetes can have long-term health implications, including uncontrolled blood sugar levels, an increased risk of complications, and a reduced quality of life.

Regarding false positives, while they can lead to unnecessary follow-up tests or treatments, they are generally less severe compared to false negatives. In the case of diabetes, a false positive may result in additional medical evaluations or interventions, but it typically does not pose the same immediate health risks as missing a true positive case. However, false positives can still cause anxiety, inconvenience, and potential economic costs associated with unnecessary medical procedures or treatments.

Considering the potential health risks and consequences associated with missing cases of diabetes, minimizing false negatives (FN) is typically of higher importance in diabetes classification. The primary goal is to ensure that individuals with diabetes are correctly identified and receive the necessary care and management.

All in all, the relative importance of false negatives and false positives may vary based on specific circumstances, such as the prevalence of diabetes in the population, the availability of follow-up confirmatory tests, the cost of those tests, and the potential impact of false positives on individuals' well-being. It is important to assess the specific context and consider the trade-offs between false negatives and false positives to determine the optimal approach for diabetes classification.

3.1 Logistic Regression Model

Let's fit our Logistic Regression model with all of our independent variables. We considered 80% of our data as the training dataset, and the remaining data as the test dataset.

```
#### Split Diabetes dataset to train and test ####
set.seed(123)
test_index = sample(nrow(Diabetes), 0.2 * nrow(Diabetes))
train = Diabetes[-test_index, ]
test = Diabetes[test_index, ]
```

```
#### Fit a logistic regression model to train dataset ####
model = glm(Outcome ~ ., data = train, family = "binomial")
```

```r
pred_train = predict(model, newdata = train, type = "response")
pred_class_train = ifelse(pred_train > 0.5, 1, 0)
```

```r
####  Calculating the confusion matrix ####
confusion_matrix_train = table(pred_class_train, train$Outcome)
true_positive_train = confusion_matrix_train[2,2]
false_positive_train = confusion_matrix_train[1,2]
false_negative_train = confusion_matrix_train[2,1]
true_negative_train = confusion_matrix_train[1,1]
```

```r
#### Calculating the performance metrics for train set ####
recall_train = true_positive_train /
  (true_positive_train + false_negative_train)

precision_train = true_positive_train /
  (true_positive_train + false_positive_train)

f1_score_train = 2 * precision_train * recall_train /
  (precision_train + recall_train)

accuracy_train = (true_positive_train + true_negative_train) /
  sum(confusion_matrix_train)
```

```r
#### Test set performance ####
pred_test = predict(model, newdata = test, type = "response")
pred_class_test = ifelse(pred_test > 0.5, 1, 0)
```

```r
#### Calculate confusion matrix for test set ####
confusion_matrix_test = table(pred_class_test, test$Outcome)
true_positive_test = confusion_matrix_test[2,2]
false_positive_test = confusion_matrix_test[1,2]
false_negative_test = confusion_matrix_test[2,1]
true_negative_test = confusion_matrix_test[1,1]
```

```r
#### Calculate performance metrics for test set ####
recall_test = true_positive_test /
  (true_positive_test + false_negative_test)

precision_test = true_positive_test /
  (true_positive_test + false_positive_test)

f1_score_test = 2 * precision_test * recall_test /
  (precision_test + recall_test)

accuracy_test = (true_positive_test + true_negative_test) /
  sum(confusion_matrix_test)
```

```r
#### Output performance metrics ####
results_train = data.frame(
  Metric = c("Recall", "Precision","F1 Score" , "Accuracy"),
  Value = c(recall_train, precision_train,f1_score_train , accuracy_train),
  Set = "Train"
)

results_test = data.frame(
  Metric = c("Recall", "Precision","F1 Score" , "Accuracy"),
  Value = c( recall_test,precision_test, f1_score_test ,accuracy_test),
  Set = "Test"
)
```

```
kable(results_train)
```

Metric	Value	Set
Recall	0.7065868	Train
Precision	0.5645933	Train
F1 Score	0.6276596	Train
Accuracy	0.7723577	Train

```
kable(results_test)
```

Metric	Value	Set
Recall	0.7500000	Test
Precision	0.5084746	Test
F1 Score	0.6060606	Test
Accuracy	0.7450980	Test

Let's discuss the different values in the output. As you can see, we have four different metrics. These four metrics serve as our model selection criteria. Let's take a closer look at them.

Recall $= \frac{TP}{TP+FN}$ Pre cision $= \frac{TP}{TP+FP}$

F1 Score $= 2 \times \frac{Precision \times Recall}{Precision + Recall}$ Accuracy $= \frac{TP+TN}{TP+TN+FP+FN}$

As we mentioned earlier, it is crucial for us to minimize false negatives (FN), but it is also important to have a low number of false positives (FP). The recall formula indicates that a higher recall value implies a lower rate of false negatives, which is particularly important when identifying positive instances. Therefore, this metric holds significant importance for us. Similarly, a higher precision value signifies a lower rate of false positives. Another vital metric is the F1 score, which allows us to consider both precision and recall simultaneously. While we strive for a higher recall value, we also aim for a higher precision. Hence, the F1 score is of great significance to us. Lastly, we have accuracy, which represents the proportion of correctly classified instances (both positive and negative) out of the total number of instances. However, it is important to note that accuracy alone is not a sufficient metric, and it should be considered alongside the aforementioned metrics.

So in the logistic model we have a good value for our recall and accuracy metrics but the value if precision and F1-score is not very satisfiable. Let's see if we can make a better result by applying cross validation technique.

3.1.1 Coefficients Interpretation for Logistic Regression

In this we want to discuss our logistic regression model coefficients.

```
summary(model)
```

```
##
## Call:
## glm(formula = Outcome ~ ., family = "binomial", data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.5180  -0.7162  -0.3652   0.6977   2.5132
##
## Coefficients:
##                           Estimate Std. Error z value Pr(>|z|)
## (Intercept)             -11.788484   2.148732  -5.486 4.11e-08 ***
## Pregnancies               0.190786   0.062758   3.040  0.00237 **
## Glucose                   0.036873   0.004310   8.554  < 2e-16 ***
## BloodPressure            -0.003562   0.005635  -0.632  0.52732
## SkinThickness             0.005284   0.012675   0.417  0.67675
## Insulin                   0.010045   0.014509   0.692  0.48871
## BMI                       0.039715   0.010199   3.894 9.87e-05 ***
## DiabetesPedigreeFunction   2.015679   0.658233   3.062  0.00220 **
## Age                       0.716324   0.415684   1.723  0.08485 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 788.34  on 614  degrees of freedom
## Residual deviance: 564.40  on 606  degrees of freedom
## AIC: 582.4
##
## Number of Fisher Scoring iterations: 5
```

- Pregnancies: The exponentiated value of the coefficient (0.190786) is the odds ratio. So exp(0.190786) is almost 1.21. It suggests that for every additional pregnancy, the odds of having diabetes increase by about 21%.
- Glucose: The exponentiated value of the coefficient (0.036873) represents the odds ratio. After calculating we have exp(0.036873) is almost 1.04. It suggests that for each unit increase in glucose levels, the odds of having diabetes increase by about 4%.
- BloodPressure: The exponentiated value of the coefficient (-0.003562) is the odds ratio so exp(-0.003562) is almost 0.996. It implies that there is almost no change in the odds of having diabetes for a one-unit increase in blood pressure.
- SkinThickness: exp(0.005284) is almost 1.01, indicating a negligible increase in the odds of diabetes.
- Insulin: exp(0.010045) is almost 1.01, suggesting a slight increase in the odds of diabetes.
- BMI: exp(0.039715) is almost 1.04, indicating a 4% increase in the odds of diabetes.
- DiabetesPedigreeFunction: exp(2.015679) is almost 7.5, suggesting a significant increase in the odds of diabetes.

- Age: exp(0.716324) is almost 2.05, indicating a 105% (or 2.05-fold) increase in the odds of diabetes for each unit increase in age.

the null deviance is 788.34, indicating that the initial model with only the intercept explains a certain amount of variability in the data. the residual deviance is 564.40, which is lower than the null deviance. A lower residual deviance suggests that the model with our predictor variables explains more of the variability in the data compared to the null model.

For the null deviance, the degrees of freedom are 614, indicating the number of observations in the dataset minus 1 (due to the intercept-only model).

For the residual deviance, the degrees of freedom are 606, indicating the number of observations minus the number of estimated parameters in the full model.

In your case, the AIC is 582.4.A lower AIC value suggests a better-fitting model.

3.2 Logistic Regression with Cross-Validation

```r
k = 10
CV_train_index = sample(nrow(Diabetes), 0.8 * nrow(Diabetes))
CV_train = Diabetes[CV_train_index, ]
```

```r
#### Split the data into k folds ####
set.seed(123)
folds = cut(seq(1, nrow(CV_train)), breaks = k, labels = FALSE)
```

```r
#### Initialize vectors to store performance metrics ####
CV_accuracy = rep(0, k)
CV_precision = rep(0, k)
CV_recall = rep(0, k)
CV_f1_score = rep(0, k)
```

```r
#### Perform k-fold cross-validation ####
for (i in 1:k) {
  #### Subset the data for the i-th fold ####
  CV_test_indices = which(folds == i)
  CV_test = CV_train[CV_test_indices, ]
  CV_train_subset = CV_train[-CV_test_indices, ]

  #### Fit the logistic regression model on the training subset ####
  CV_model = glm(Outcome ~ ., data = CV_train_subset, family = "binomial")

  #### Make predictions on the test subset ####
  CV_pred = predict(CV_model, newdata = CV_test, type = "response")
  CV_pred_class = ifelse(CV_pred > 0.5, 1, 0)

  #### Calculate confusion matrix and performance metrics ####
  CV_confusion_matrix = table(CV_pred_class, CV_test$Outcome)
  CV_true_positive = CV_confusion_matrix[2,2]
  CV_false_positive = CV_confusion_matrix[1,2]
  CV_false_negative = CV_confusion_matrix[2,1]
  CV_true_negative = CV_confusion_matrix[1,1]
  CV_recall[i] = CV_true_positive /
    (CV_true_positive + CV_false_negative)

  CV_precision[i] = CV_true_positive /
    (CV_true_positive + CV_false_positive)

  CV_f1_score[i] = 2 *  CV_precision[i] * CV_recall[i] /
    ( CV_precision[i] + CV_recall[i])

  CV_accuracy[i] = (CV_true_positive + CV_true_negative) /
    sum(CV_confusion_matrix)
}
```

```
#### Calculate average performance metrics across all folds ####
mean_accuracy = mean(CV_accuracy)
mean_precision = mean(CV_precision)
mean_recall = mean(CV_recall)
mean_f1_score = mean(CV_f1_score)
```

```
#### Output performance metrics ####
CV_results = data.frame(
  Metric = c("CV Recall","CV Precision", "CV F1 Score","CV Accuracy"),
  Value = c(mean_recall,mean_precision, mean_f1_score, mean_accuracy)
)
```

```
kable(CV_results)
```

Metric	Value
CV Recall	0.6694097
CV Precision	0.5526208
CV F1 Score	0.5927402
CV Accuracy	0.7427287

As you can see, the precision and F1-score have increased, but the recall and accuracy have decreased compared to the previous model. This indicates a trade-off between these metrics. Let's investigate whether we can achieve better results by applying the backward feature selection method.

3.2.1 Coefficients Interpretation for Logistic Regression with Cross-Validation

In this we want to discuss our logistic regression model coefficients.

```
summary(CV_model)
```

```
##
## Call:
## glm(formula = Outcome ~ ., family = "binomial", data = CV_train_subset)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.5607  -0.7519  -0.3926   0.7560   2.1464
##
## Coefficients:
##                           Estimate Std. Error z value Pr(>|z|)
## (Intercept)             -11.151862   2.166953  -5.146 2.66e-07 ***
## Pregnancies               0.148833   0.063131   2.358  0.01840 *
## Glucose                   0.033943   0.004425   7.670 1.72e-14 ***
## BloodPressure             0.004011   0.005866   0.684  0.49409
## SkinThickness             0.003505   0.012758   0.275  0.78354
## Insulin                   0.013032   0.014595   0.893  0.37191
## BMI                       0.042593   0.010354   4.114 3.90e-05 ***
## DiabetesPedigreeFunction   2.113587   0.685459   3.083  0.00205 **
## Age                       0.420931   0.425804   0.989  0.32288
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 716.98  on 551  degrees of freedom
## Residual deviance: 529.20  on 543  degrees of freedom
## AIC: 547.2
##
## Number of Fisher Scoring iterations: 5
```

- Pregnancies: The exponential value of the coefficient (0.148833) is the odds ratio. Exponentiating it gives us exp(0.148833) almost 1.16. It suggests that for each additional pregnancy, the odds of having diabetes increase by approximately 16%.
- Glucose: The exponential value of the coefficient (0.033943) represents the odds ratio. Exponentiating it gives us exp(0.033943) almost 1.03. It suggests that for each unit increase in glucose levels, the odds of having diabetes increase by about 3%.
- BloodPressure: The exponential value of the coefficient (0.004011) is the odds ratio. Exponentiating it gives us exp(0.004011) almost 1.00. It indicates that there is almost no change in the odds of having diabetes for a one-unit increase in blood pressure.
- SkinThickness and Insulin: For these variables, the exponential coefficients represent the odds ratio associated with a one-unit increase in each variable: SkinThickness: exp(0.003505) is almost 1.00, indicating a negligible increase in the odds of diabetes. Insulin: exp(0.013032) is almost 1.01, suggesting a slight increase in the odds of diabetes.
- BMI: The exponential value of the coefficient (0.042593) represents the odds ratio. Exponentiating it gives us exp(0.042593) is almost 1.04. It suggests that for each unit increase in BMI, the odds of having diabetes increase by about 4%.
- DiabetesPedigreeFunction: The exponential value of the coefficient (2.113587) is the odds ratio. Exponentiating it gives us exp(2.113587) almost 8.28. It suggests a significant increase in the odds of having diabetes for each unit increase in the DiabetesPedigreeFunction.
- Age: The exponential value of the coefficient (0.420931) represents the odds ratio. Exponentiating it gives us exp(0.420931) almost 1.52. It suggests a 52% increase in the odds of having diabetes for each unit increase in age.

The null deviance which is 716.98 represents the deviance obtained when only the intercept is included in the logistic regression. It measures the total unexplained variability in the Outcome variable by the null model.

The residual deviance is 529.20 which represents the deviance that remains after including the predictor variables in the logistic regression model.

Also in our case the AIC is 547.2.

3.3 Logistic Regression with Backward feature selection method

```
#### Perform backward feature selection using step function ####
backward_model = step(model, direction = "backward")
```

```
## Start:  AIC=582.4
## Outcome ~ Pregnancies + Glucose + BloodPressure + SkinThickness +
##     Insulin + BMI + DiabetesPedigreeFunction + Age
##
##                            Df Deviance    AIC
## - SkinThickness             1   564.57 580.57
## - BloodPressure             1   564.80 580.80
## - Insulin                   1   564.88 580.88
## <none>                          564.40 582.40
## - Age                       1   567.35 583.35
## - Pregnancies               1   573.83 589.83
## - DiabetesPedigreeFunction  1   573.90 589.90
## - BMI                       1   580.27 596.27
## - Glucose                   1   657.11 673.11
##
## Step:  AIC=580.57
## Outcome ~ Pregnancies + Glucose + BloodPressure + Insulin + BMI +
##     DiabetesPedigreeFunction + Age
##
##                            Df Deviance    AIC
## - BloodPressure             1   564.95 578.95
## - Insulin                   1   565.06 579.06
## <none>                          564.57 580.57
## - Age                       1   567.57 581.57
## - DiabetesPedigreeFunction  1   574.04 588.04
## - Pregnancies               1   574.10 588.10
## - BMI                       1   586.31 600.31
## - Glucose                   1   657.85 671.85
##
## Step:  AIC=578.95
## Outcome ~ Pregnancies + Glucose + Insulin + BMI + DiabetesPedigreeFunction +
##     Age
##
##                            Df Deviance    AIC
## - Insulin                   1   565.50 577.50
## <none>                          564.95 578.95
## - Age                       1   567.60 579.60
## - DiabetesPedigreeFunction  1   574.46 586.46
## - Pregnancies               1   574.46 586.46
## - BMI                       1   586.87 598.87
## - Glucose                   1   658.28 670.28
##
```

```
## Step:  AIC=577.5
## Outcome ~ Pregnancies + Glucose + BMI + DiabetesPedigreeFunction +
##     Age
##
##                            Df Deviance    AIC
## <none>                          565.50 577.50
## - Age                       1   568.11 578.11
## - Pregnancies               1   575.16 585.16
## - DiabetesPedigreeFunction  1   575.28 585.28
## - BMI                       1   589.22 599.22
## - Glucose                   1   679.63 689.63
```

```
#### Train set performance ####
backward_pred_train = predict(backward_model, newdata = train, type = "response")
backward_pred_class_train = ifelse(backward_pred_train > 0.5, 1, 0)
```

```
#### Calculate confusion matrix for train set ####
backward_confusion_matrix_train = table(backward_pred_class_train, train$Outcome)
backward_true_positive_train = backward_confusion_matrix_train[2,2]
backward_false_positive_train = backward_confusion_matrix_train[1,2]
backward_false_negative_train = backward_confusion_matrix_train[2,1]
backward_true_negative_train = backward_confusion_matrix_train[1,1]
```

```
#### Calculate performance metrics for train set ####
backward_recall_train = backward_true_positive_train /
  (backward_true_positive_train + backward_false_negative_train)

backward_precision_train = backward_true_positive_train /
  (backward_true_positive_train + backward_false_positive_train)

backward_f1_score_train = 2 * backward_precision_train * backward_recall_train /
 (backward_precision_train + backward_recall_train)

backward_accuracy_train = (backward_true_positive_train +backward_true_negative_train)/
  sum(backward_confusion_matrix_train)
```

```
#### Test set performance ####
backward_pred_test = predict(backward_model, newdata = test, type = "response")
backward_pred_class_test = ifelse(backward_pred_test > 0.5, 1, 0)
```

```
#### Calculate confusion matrix for test set ####
backward_confusion_matrix_test = table(backward_pred_class_test, test$Outcome)
backward_true_positive_test = backward_confusion_matrix_test[2,2]
backward_false_positive_test = backward_confusion_matrix_test[1,2]
backward_false_negative_test = backward_confusion_matrix_test[2,1]
backward_true_negative_test = backward_confusion_matrix_test[1,1]
```

```
#### Calculate performance metrics for test set ####
backward_recall_test = backward_true_positive_test /
  (backward_true_positive_test + backward_false_negative_test)

backward_precision_test = backward_true_positive_test /
```

```
  (backward_true_positive_test + backward_false_positive_test)

backward_f1_score_test = 2 * backward_precision_test * backward_recall_test /
  (backward_precision_test + backward_recall_test)

backward_accuracy_test = (backward_true_positive_test + backward_true_negative_test) /
  sum(backward_confusion_matrix_test)
```

```
#### Output performance metrics ####
backward_results_train = data.frame(
  Metric = c("backward_Recall","backward_Precision",
             "backward_F1 Score", "backward_Accuracy"),
  Value = c( backward_recall_train, backward_precision_train,
             backward_f1_score_train, backward_accuracy_train),
  Set = "Train"
)

backward_results_test= data.frame(
  Metric = c("backward_Recall","backward_Precision",
             "backward_F1 Score", "backward_Accuracy"),

  Value = c(backward_recall_test, backward_precision_test,
            backward_f1_score_test, backward_accuracy_test),

  Set = "Test"
)
```

```
kable(backward_results_train)
```

Metric	Value	Set
backward_Recall	0.7195122	Train
backward_Precision	0.5645933	Train
backward_F1 Score	0.6327078	Train
backward_Accuracy	0.7772358	Train

```
kable(backward_results_test)
```

Metric	Value	Set
backward_Recall	0.7317073	Test
backward_Precision	0.5084746	Test
backward_F1 Score	0.6000000	Test
backward_Accuracy	0.7385621	Test

We can see that there is no impressive changes in backward model in comparison to logistic model!!

It's time to check the summary of the backward model.

```r
summary(backward_model)
```

```
##
## Call:
## glm(formula = Outcome ~ Pregnancies + Glucose + BMI + DiabetesPedigreeFunction +
##     Age, family = "binomial", data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.5814  -0.7188  -0.3706   0.6823   2.5274
##
## Coefficients:
##                           Estimate Std. Error z value Pr(>|z|)
## (Intercept)             -10.562913   1.410756  -7.487 7.02e-14 ***
## Pregnancies               0.192942   0.062734   3.076   0.0021 **
## Glucose                   0.037630   0.004021   9.358  < 2e-16 ***
## BMI                       0.040994   0.008650   4.739 2.15e-06 ***
## DiabetesPedigreeFunction   2.039643   0.656826   3.105   0.0019 **
## Age                       0.650746   0.401606   1.620   0.1052
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 788.34  on 614  degrees of freedom
## Residual deviance: 565.50  on 609  degrees of freedom
## AIC: 577.5
##
## Number of Fisher Scoring iterations: 5
```

- Pregnancies: The exponential value of the coefficient (0.192942) is the odds ratio. Exponentiating it gives us exp(0.192942) almost 1.213. It suggests that for each additional pregnancy, the odds of having diabetes increase by approximately 21.3%.
- Glucose: The exponential value of the coefficient (0.037630) represents the odds ratio. Exponentiating it gives us exp(0.037630) almost 1.038. It suggests that for each unit increase in glucose levels, the odds of having diabetes increase by about 3.8%.
- BMI: The exponential value of the coefficient (0.040994) represents the odds ratio. Exponentiating it gives us exp(0.040994) almost 1.042. It suggests that for each unit increase in BMI, the odds of having diabetes increase by about 4.2%.
- DiabetesPedigreeFunction: The exponential value of the coefficient (2.039643) is the odds ratio. Exponentiating it gives us exp(2.039643) almost 7.686. It suggests a significant increase in the odds of having diabetes for each unit increase in the DiabetesPedigreeFunction.
- Age: The exponential value of the coefficient (0.650746) represents the odds ratio. Exponentiating it gives us exp(0.650746) almost 1.92. It suggests a 92% increase in the odds of having diabetes for each unit increase in age.

The null deviance is 788.34 which represents the total unexplained variability in the Outcome variable by the null model. Moreover, the residual deviance is 565.50 which represents the deviance that remains after including the predictor variables in the logistic regression model. Besides, the AIC is 577.5.

Now let's check which of these 3 models are the best one.

```r
#### A data frame with metrics for three available logistic regression models ####
Logistic_Regression_results = data.frame(
  Metric = c('Recall', 'Precision', 'F1-Score', 'Accuracy', 'AIC'),
  Logistic_Regression = c(0.75, 0.50, 0.60, 0.74,582.4),
  Cross_Validation = c(0.68, 0.58, 0.62, 0.76,547.2),
  Backward_Feature_Selection = c(0.73, 0.50, 0.60, 0.73,577.5)
)

kable(Logistic_Regression_results, format = "markdown")
```

Metric	Logistic_Regression	Cross_Validation	Backward_Feature_Selection
Recall	0.75	0.68	0.73
Precision	0.50	0.58	0.50
F1-Score	0.60	0.62	0.60
Accuracy	0.74	0.76	0.73
AIC	582.40	547.20	577.50

As mentioned previously, it is crucial to consider the trade-off between recall and precision metrics. Comparing the logistic regression model with the logistic regression model with backward feature selection, the logistic regression model outperforms in terms of performance. Now, the question arises: which is the better model between the logistic regression model and the logistic regression model with cross-validation?

In the logistic regression model, the recall metric demonstrates a better result, whereas in the model with cross-validation, the precision metric performs better. Although, our primary concern is maximizing recall, which signifies minimizing false negatives, but we select the logistic regression with cross-validation model as the superior model among the three options since it holds good balance between all of the metrics.Also we can see that the logistic regresion model with cross-validation has lower AIC which means that it is a better model since this model provides a better balance between model complexity and goodness-of-fit compared to the other two models.

In the confusion matrix below, you can see that the logistic model has performed better in classification compared to the other two models. Our model has predicted 10 instances as false negatives (FN), indicating that 10 instances actually had Diabetes but were predicted as Non-Diabetic by our model.

```r
#### A table with the confusion matrix of all three available logistic models ####
Confusion_Matrix = data.frame(class = c('Negative' , 'Positive'),
                              'LR-Negative' = c(84 , 10),
                              'LR-Positive' = c(29 , 30),
                              'CV-Negative' = c(35 , 4) ,
                              'CV-Positive' = c(11 , 12),
                              'BW-Negative' = c(83 , 11) ,
                              'BW-Positive' = c(29 , 30)
                              )

kable(Confusion_Matrix, format = "markdown")
```

class	LR.Negative	LR.Positive	CV.Negative	CV.Positive	BW.Negative	BW.Positive
Negative	84	29	35	11	83	29
Positive	10	30	4	12	11	30

We can reduce the number of false negatives (FN) by adjusting the threshold.

```
#### changing threshold ####
pred_test = predict(model, newdata = test, type = "response")
pred_class_test = ifelse(pred_test > 0.6, 1, 0)

confusion_matrix_test = table(pred_class_test, test$Outcome)
true_positive_test = confusion_matrix_test[2,2]
false_positive_test = confusion_matrix_test[1,2]
false_negative_test = confusion_matrix_test[2,1]
true_negative_test = confusion_matrix_test[1,1]

recall_test = true_positive_test /
  (true_positive_test + false_negative_test)

precision_test = true_positive_test /
  (true_positive_test + false_positive_test)

f1_score_test = 2 * precision_test * recall_test /
  (precision_test + recall_test)

accuracy_test = (true_positive_test + true_negative_test) /
  sum(confusion_matrix_test)

results_test = data.frame(
  Metric = c("Recall", "Precision","F1 Score" , "Accuracy"),
  Value = c( recall_test,precision_test, f1_score_test ,accuracy_test),
  Set = "Test"
)
kable(results_test)
```

Metric	Value	Set
Recall	0.7647059	Test
Precision	0.4406780	Test
F1 Score	0.5591398	Test
Accuracy	0.7320261	Test

```
kable(confusion_matrix_test)
```

	0	1
0	86	33
1	8	26

It is evident that the model's performance, in terms of the recall metric, has improved, resulting in a reduction in the number of false negatives. However, there has been a decrease in the performance of other metrics, such as precision and F1-score. Despite this, the recall has not changed significantly. Therefore, we have decided to continue using the standard threshold.

3.4 Logistic Regression ROC curve

```r
roc_data = data.frame(prob = backward_pred_test, label = test$Outcome)
CV_roc_data = data.frame(prob = CV_pred, label = CV_test$Outcome)
LR_roc_data = data.frame(prob = pred_test, label = test$Outcome)

#### Sort the data frame by the predicted probabilities in descending order ####
roc_data = roc_data[order(-roc_data$prob), ]
CV_roc_data = CV_roc_data[order(-CV_roc_data$prob), ]
LR_roc_data = LR_roc_data[order(-LR_roc_data$prob), ]
```

```r
#### Calculate the true positive rate and the false positive rate ####
tpr = cumsum(roc_data$label) / sum(roc_data$label)
fpr = cumsum(!roc_data$label) / sum(!roc_data$label)

CV_tpr = cumsum(CV_roc_data$label) / sum(CV_roc_data$label)
CV_fpr = cumsum(!CV_roc_data$label) / sum(!CV_roc_data$label)

LR_tpr = cumsum(LR_roc_data$label) / sum(LR_roc_data$label)
LR_fpr = cumsum(!LR_roc_data$label) / sum(!LR_roc_data$label)
```

```r
#### Plot the ROC curve ####
par(mfrow = c(1,1))
plot(fpr, tpr, type = "l", main = "ROC Curve", xlab = "False Positive Rate",
     ylab = "True Positive Rate" , col = '#76EEC6')
lines(CV_fpr, CV_tpr, type = "l", main = "ROC Curve", xlab = "False Positive Rate",
      ylab = "True Positive Rate" , col = 'blue')
lines(LR_fpr, LR_tpr, type = "l", main = "ROC Curve", xlab = "False Positive Rate",
      ylab = "True Positive Rate" , col = 'red')
#### Add a diagonal line representing random guessing ####
abline(0, 1, col = "gray", lty = 2)

Backward_auc_roc = round(auc(fpr, tpr), 3)
```

```
## Setting levels: control = 0, case = 0.0106382978723404

## Setting direction: controls < cases
```

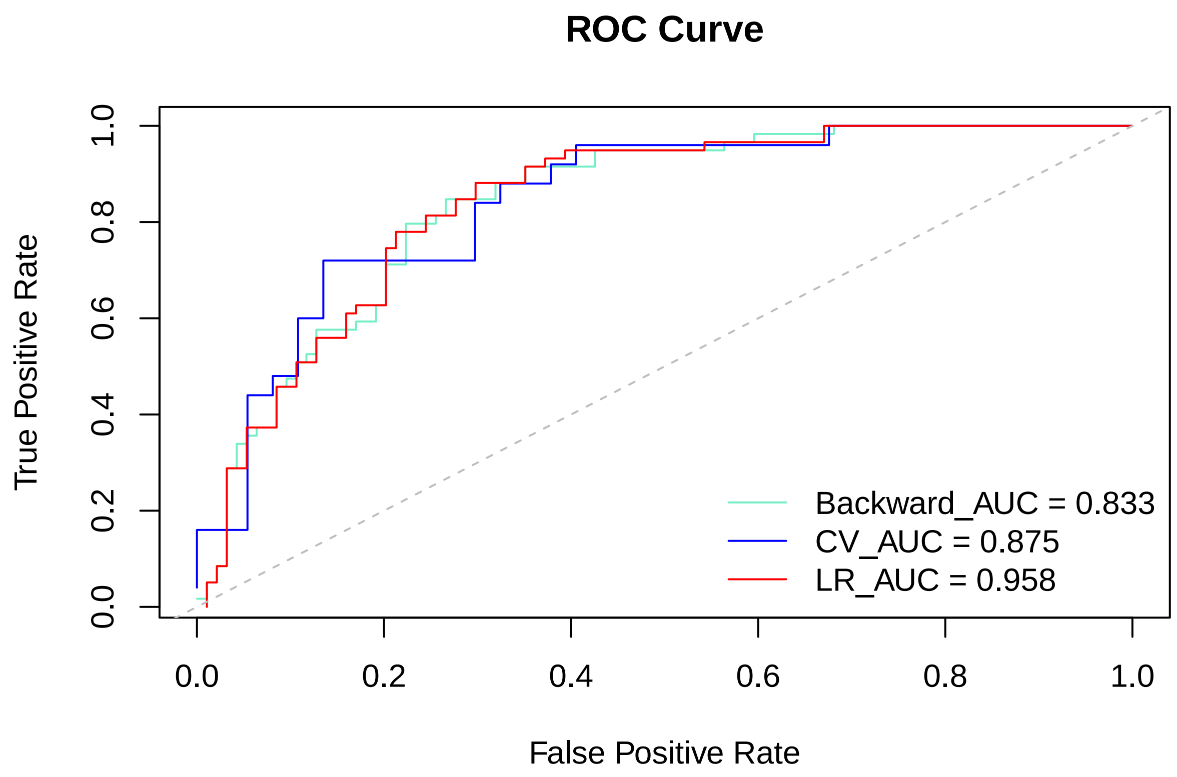
```r
CV_auc_roc = round(auc(CV_fpr, CV_tpr), 3)
```

```
## Setting levels: control = 0, case = 0.027027027027027
## Setting direction: controls < cases
```

```r
LR_auc_roc = round(auc(LR_fpr, LR_tpr), 3)
```

```
## Setting levels: control = 0.0106382978723404, case = 0.0212765957446809
## Setting direction: controls < cases
```

```r
legend_labels = c(paste("Backward_AUC =", Backward_auc_roc),
                   paste("CV_AUC =", CV_auc_roc),
                   paste("LR_AUC =", LR_auc_roc))

legend("bottomright",legend = legend_labels, bty = "n",
       col = c('#76EEC6', 'blue', 'red'),lty = 1)
```



The AUC values indicate the overall performance of each model in terms of their ability to discriminate between positive and negative instances. A higher AUC suggests better predictive performance and a greater ability to correctly classify instances. Therefore, based on the AUC values, the logistic regression model appears to have the best performance among the three models in terms of its ability to distinguish between the classes.

3.5 KNN Model

In the code below, we utilized the KNN (K-Nearest Neighbors) method. However, the results indicate that the recall and precision values are not satisfactory.

```
#### Fit a KNN model ####
k = 5
KNN_model = knn(train[, -9], test[, -9], train[, 9], k)
```

```
#### Calculate confusion matrix for test set ####
KNN_confusion_matrix = table(KNN_model, test$Outcome)
KNN_true_positive = KNN_confusion_matrix[2,2]
KNN_false_positive = KNN_confusion_matrix[1,2]
KNN_false_negative = KNN_confusion_matrix[2,1]
KNN_true_negative = KNN_confusion_matrix[1,1]
```

```
#### Calculate performance metrics for test set ####
KNN_accuracy = (KNN_true_positive + KNN_true_negative) /
  sum(KNN_confusion_matrix)
```

```r
KNN_precision = KNN_true_positive /
  (KNN_true_positive + KNN_false_positive)

KNN_recall = KNN_true_positive /
  (KNN_true_positive + KNN_false_negative)

KNN_f1_score = 2 * KNN_precision * KNN_recall /
  (KNN_precision + KNN_recall)
```

```r
#### Output performance metrics ####
KNN_results = data.frame(
  Metric = c("Recall", "Precision", "F1 Score", "Accuracy"),
  Value = c(KNN_recall, KNN_precision, KNN_f1_score, KNN_accuracy)
)

kable(KNN_results)
```

Metric	Value
Recall	0.6304348
Precision	0.4915254
F1 Score	0.5523810
Accuracy	0.6928105

Let's check if we can improve the results using cross validation.

3.6 KNN with Cross-Validation

```r
#### Split the data into training and test sets ####
k = 10
KNN_folds = cut(seq(1, nrow(Diabetes)), breaks = k, labels = FALSE)
KNN_CV_recall <- numeric(k)
KNN_CV_recall = numeric(k)
KNN_CV_precision = numeric(k)
KNN_CV_f1_score = numeric(k)
KNN_CV_accuracy = numeric(k)
```

```r
for (i in 1:k) {
  ##### Create training and test sets for this fold ####
  KNN_test_indices = which(KNN_folds == i, arr.ind = TRUE)
  KNN_test = Diabetes[KNN_test_indices, ]
  KNN_train = Diabetes[-KNN_test_indices, ]

  #### Fit a KNN model ####
  KNN_CV_model = knn(KNN_train[, -9], KNN_test[, -9], KNN_train[, 9], k = 5)

  #### Calculate confusion matrix for test set ####
  KNN_CV_confusion_matrix = table(KNN_CV_model, KNN_test$Outcome)
  KNN_CV_true_positive = KNN_CV_confusion_matrix[2,2]
  KNN_CV_false_positive = KNN_CV_confusion_matrix[1,2]
  KNN_CV_false_negative = KNN_CV_confusion_matrix[2,1]
  KNN_CV_true_negative = KNN_CV_confusion_matrix[1,1]

  #### Calculate performance metrics for test set ####
  KNN_CV_accuracy[i] = (KNN_CV_true_positive + KNN_CV_true_negative) /
    sum(KNN_CV_confusion_matrix)

  KNN_CV_precision[i] = KNN_CV_true_positive /
    (KNN_CV_true_positive + KNN_CV_false_positive)

  KNN_CV_recall[i] = KNN_CV_true_positive /
    (KNN_CV_true_positive + KNN_CV_false_negative)

  KNN_CV_f1_score[i] = 2 * KNN_CV_precision[i] * KNN_CV_recall[i] /
    (KNN_CV_precision[i] + KNN_CV_recall[i])
}
```

```
#### Calculate average performance metrics across all folds ####
KNN_mean_accuracy = mean(KNN_CV_accuracy)
KNN_mean_precision = mean(KNN_CV_precision)
KNN_mean_recall = mean(KNN_CV_recall)
KNN_mean_f1_score = mean(KNN_CV_f1_score)
```

```
#### Output performance metrics ####
KNN_CV_results = data.frame(
  Metric = c("CV Recall","CV Precision", "CV F1 Score","CV Accuracy"),
  Value = c(KNN_mean_recall, KNN_mean_precision, KNN_mean_f1_score, KNN_mean_accuracy)
)
```

```
kable(KNN_CV_results)
```

Metric	Value
CV Recall	0.6033597
CV Precision	0.5740493
CV F1 Score	0.5821436
CV Accuracy	0.7199590

It is evident that there is a significant improvement in precision, F1-score, and accuracy. However, it is worth noting that the recall has decreased compared to the previous model.

Now we can check which of these two models are better.

```
KNN_results = data.frame(Metric = c('Recall' , 'Precission' , 'F1-Score' , 'Accuracy'),
                         KNN = c( 0.63 , 0.49 ,0.55 ,0.69),
                         KNN_Cross_Validation = c(0.60 , 0.57 , 0.58 , 0.71))


kable(KNN_results)
```

Metric	KNN	KNN_Cross_Validation
Recall	0.63	0.60
Precission	0.49	0.57
F1-Score	0.55	0.58
Accuracy	0.69	0.71

```
Confusion_Matrix_KNN = data.frame(class = c('Non-Diabetic' , 'Diabetic'),
                                  'KNN-Non-Diabetic' = c(757 , 17),
                                   'KNN-Diabetic' = c(30 , 29),
                                  'KNN_CV-Non-Diabetic' = c(38 , 8) ,
                                  'KNN_CV-Diabetic' = c(13 , 18))

kable(Confusion_Matrix_KNN)
```

class	KNN.Non.Diabetic	KNN.Diabetic	KNN_CV.Non.Diabetic	KNN_CV.Diabetic
Non-Diabetic	757	30	38	13
Diabetic	17	29	8	18

As evident from the results, both models have shown inferior performance compared to the logistic regression models. However, after careful evaluation, we have decided to choose the KNN model with cross-validation due to its superior performance. Therefore, our selected models include one logistic regression with cross validation model and one KNN model with cross-validation.

Now let's check the Naive Bayes model.

3.7 Naive Bayes Model

```
#### Fit Naive Bayes model ####
nb_model = naiveBayes(Outcome ~ ., data = train)

#### Train set performance ####
nb_pred_train = predict(nb_model, newdata = train, type = 'class')
```

```
#### Calculate confusion matrix for train set ####
nb_confusion_matrix_train = table(nb_pred_train, train$Outcome)
nb_true_positive_train = nb_confusion_matrix_train[2,2]
nb_false_positive_train = nb_confusion_matrix_train[1,2]
nb_false_negative_train = nb_confusion_matrix_train[2,1]
nb_true_negative_train = nb_confusion_matrix_train[1,1]
```

Now we have defined different values for our confusion matrix. now it's time to define different metrics.

```
#### Calculate performance metrics for train set ####
nb_recall_train = nb_true_positive_train /
  (nb_true_positive_train + nb_false_negative_train)

nb_precision_train = nb_true_positive_train /
  (nb_true_positive_train + nb_false_positive_train)
```

```r
nb_f1_score_train = 2 * nb_precision_train * nb_recall_train /
  (nb_precision_train + nb_recall_train)

nb_accuracy_train = (nb_true_positive_train + nb_true_negative_train) /
  sum(nb_confusion_matrix_train)
```

```r
# Test set performance
nb_pred_test = predict(nb_model, newdata = test , type = 'class')
```

```r
#### Calculate confusion matrix for test set ####
nb_confusion_matrix_test = table(nb_pred_test, test$Outcome)
nb_true_positive_test = nb_confusion_matrix_test[2,2]
nb_false_positive_test = nb_confusion_matrix_test[1,2]
nb_false_negative_test = nb_confusion_matrix_test[2,1]
nb_true_negative_test = nb_confusion_matrix_test[1,1]
```

```r
#### Calculate performance metrics for test set ####
nb_recall_test = nb_true_positive_test /
  (nb_true_positive_test + nb_false_negative_test)

nb_precision_test = nb_true_positive_test /
  (nb_true_positive_test + nb_false_positive_test)

nb_f1_score_test = 2 * nb_precision_test * nb_recall_test /
  (nb_precision_test + nb_recall_test)

nb_accuracy_test = (nb_true_positive_test + nb_true_negative_test) /
  sum(nb_confusion_matrix_test)
```

```r
#### Output performance metrics ####
nb_results_train = data.frame(
  Metric = c("Recall", "Precision","F1 Score" , "Accuracy"),
  Value = c(nb_recall_train,nb_precision_train,nb_f1_score_train,nb_accuracy_train),
  Set = "Train"
)

nb_results_test = data.frame(
  Metric = c("Recall", "Precision","F1 Score" , "Accuracy"),
  Value = c( nb_recall_test, nb_precision_test, nb_f1_score_test , nb_accuracy_test),
  Set = "Test"
)
```

```r
kable(nb_results_train)
```

Metric	Value	Set
Recall	0.6476190	Train
Precision	0.6507177	Train
F1 Score	0.6491647	Train
Accuracy	0.7609756	Train

```r
kable(nb_results_test)
```

Metric	Value	Set
Recall	0.6481481	Test
Precision	0.5932203	Test
F1 Score	0.6194690	Test
Accuracy	0.7189542	Test

The model achieved a recall of 0.6481481, indicating that it correctly identified 64.81% of the positive cases in the test set.The precision value is 0.5932203, indicating that out of the cases predicted as positive by the model, 59.32% were actually true positive cases.The F1 score, which is a balanced measure of precision and recall, is 0.6194690. It takes into account both the precision and recall values, providing an overall assessment of the model's performance.The accuracy of the model is 0.7189542, suggesting that it correctly classified 71.89% of the cases in the test set.

3.8 Naive Bayes with cross-validation

```r
#### Define the number of folds for cross-validation ####
k = 10

nb_CV_test_index = sample(nrow(Diabetes), 0.2 * nrow(Diabetes))
nb_CV_train = Diabetes[-nb_CV_test_index, ]
nb_CV_test = Diabetes[nb_CV_test_index, ]

#### Split the data into k folds ####
set.seed(123)
nb_folds = cut(seq(1, nrow(nb_CV_train)), breaks = k, labels = FALSE)

#### Initialize vectors to store performance metrics ####
nb_CV_accuracy = rep(0, k)
nb_CV_precision = rep(0, k)
nb_CV_recall = rep(0, k)
nb_CV_f1_score = rep(0, k)
```

```r
#### Perform k-fold cross-validation####
for (i in 1:k) {
  #### Subset the data for the i-th fold ####
  nb_CV_test_indices = which(nb_folds == i)
  nb_CV_test = nb_CV_train[nb_CV_test_indices, ]
  nb_train_subset = nb_CV_train[-nb_CV_test_indices, ]

  #### Fit the logistic regression model on the training subset ####
  nb_CV_model = naiveBayes(Outcome ~ ., data = nb_train_subset, family = "binomial")

  #### Make predictions on the test subset ####
  nb_CV_pred = predict(nb_CV_model, newdata = nb_CV_test, type = "class")

  #### Calculate confusion matrix and performance metrics ####
  nb_CV_confusion_matrix = table(nb_CV_pred, nb_CV_test$Outcome)
  nb_CV_true_positive = nb_CV_confusion_matrix[2,2]
  nb_CV_false_positive = nb_CV_confusion_matrix[1,2]
  nb_CV_false_negative = nb_CV_confusion_matrix[2,1]
  nb_CV_true_negative = nb_CV_confusion_matrix[1,1]


  nb_CV_recall[i] = nb_CV_true_positive /
    (nb_CV_true_positive + nb_CV_false_negative)

  nb_CV_precision[i] = nb_CV_true_positive /
    (nb_CV_true_positive + nb_CV_false_positive)

  nb_CV_f1_score[i] = 2 *  nb_CV_precision[i] * nb_CV_recall[i] /
    ( nb_CV_precision[i] + nb_CV_recall[i])

  nb_CV_accuracy[i] = (nb_CV_true_positive + nb_CV_true_negative) /
    sum(nb_CV_confusion_matrix)
}
```

```
#### Calculate average performance metrics across all folds ####
mean_accuracy = mean(nb_CV_accuracy)
mean_precision = mean(nb_CV_precision)
mean_recall = mean(nb_CV_recall)
mean_f1_score = mean(nb_CV_f1_score)
```

```
#### Output performance metrics ####
nb_CV_results = data.frame(
  Metric = c("nb_CV Recall","nb_CV Precision", "nb_CV F1 Score","nb_CV Accuracy"),
  Value = c(mean_recall,mean_precision, mean_f1_score, mean_accuracy)
)

kable(nb_CV_results)
```

Metric	Value
nb_CV Recall	0.6418409
nb_CV Precision	0.6523671
nb_CV F1 Score	0.6431722
nb_CV Accuracy	0.7591750

The model achieved a recall of 0.6418409, indicating that it correctly identified 64.18% of the positive cases.The precision value is 0.6523671, suggesting that out of the cases predicted as positive by the model, 65.24% were actually true positive cases.The F1 score, which is a balanced measure of precision and recall, is 0.6431722. It takes into account both the precision and recall values, providing an overall assessment of the model's performance.The accuracy of the model is 0.7591750, indicating that it correctly classified 75.92% of the cases.

```
Naive_Bayes_results = data.frame(Metric = c('Recall' , 'Precission' , 'F1-Score' , 'Accuracy'),
                                         Naive_Bayes = c( 0.64 , 0.59 ,0.61 ,0.71),
                                         NB_Cross_Validation = c(0.64 , 0.65 , 0.64 , 0.75))



kable(Naive_Bayes_results)
```

Metric	Naive_Bayes	NB_Cross_Validation
Recall	0.64	0.64
Precission	0.59	0.65
F1-Score	0.61	0.64
Accuracy	0.71	0.75

Both models have the same recall value of 0.64, indicating that they correctly identified the same proportion of positive cases.The Naive Bayes model without cross-validation has a precision of 0.59, while the Naive Bayes model with cross-validation has a higher precision of 0.65. This suggests that the Naive Bayes model with cross-validation performs better in terms of correctly predicting positive cases.The F1-score for the Naive Bayes model without cross-validation is 0.61, whereas the Naive Bayes model with cross-validation has an F1-score of 0.64. This indicates that the Naive Bayes model with cross-validation achieves a better balance between precision and recall. Finally, the Naive Bayes model without cross-validation has an accuracy of 0.71, while the Naive Bayes model with cross-validation has a higher accuracy of 0.75. This means that the Naive Bayes model with cross-validation has a higher overall accuracy in classifying the data. So we select the Naive Bayes model with cross validation. So far we have selected Logistic regression model with cross validation, KNN with cross Validation, and Naive Bayes model with cross validation.

The relevant confusion matrix for each model is shown below.

```r
Confusion_Matrix_nb = data.frame(class = c('Non-Diabetic' , 'Diabetic'),
                                 'NB-Non-Diabetic' = c(75 , 19) ,
                                 'NB-Diabetic' = c(24 , 35),
                                 'NB_CV-Non-Diabetic' = c(35 , 4) ,
                                 'NB_CV-Diabetic' = c(9 , 14))

kable(Confusion_Matrix_nb)
```

class	NB.Non.Diabetic	NB.Diabetic	NB_CV.Non.Diabetic	NB_CV.Diabetic
Non-Diabetic	75	24	35	9
Diabetic	19	35	4	14

3.9 Naive Bayes ROC Curve.

```r
NB_CV_roc_data = data.frame(prob = nb_CV_pred, label = nb_CV_test$Outcome)
NB_roc_data = data.frame(prob = nb_pred_test, label = test$Outcome)

#### Sort the data frame by the predicted probabilities in descending order ####
NB_CV_roc_data = CV_roc_data[order(- NB_CV_roc_data$prob), ]
NB_roc_data = LR_roc_data[order(- NB_roc_data$prob), ]

#### Calculate the true positive rate and the false positive rate  ####
NB_tpr = cumsum(NB_roc_data$label) / sum(NB_roc_data$label)
NB_fpr = cumsum(!NB_roc_data$label) / sum(!NB_roc_data$label)


NB_CV_tpr = cumsum(NB_CV_roc_data$label) / sum(NB_CV_roc_data$label)
NB_CV_fpr = cumsum(!NB_CV_roc_data$label) / sum(!NB_CV_roc_data$label)
```

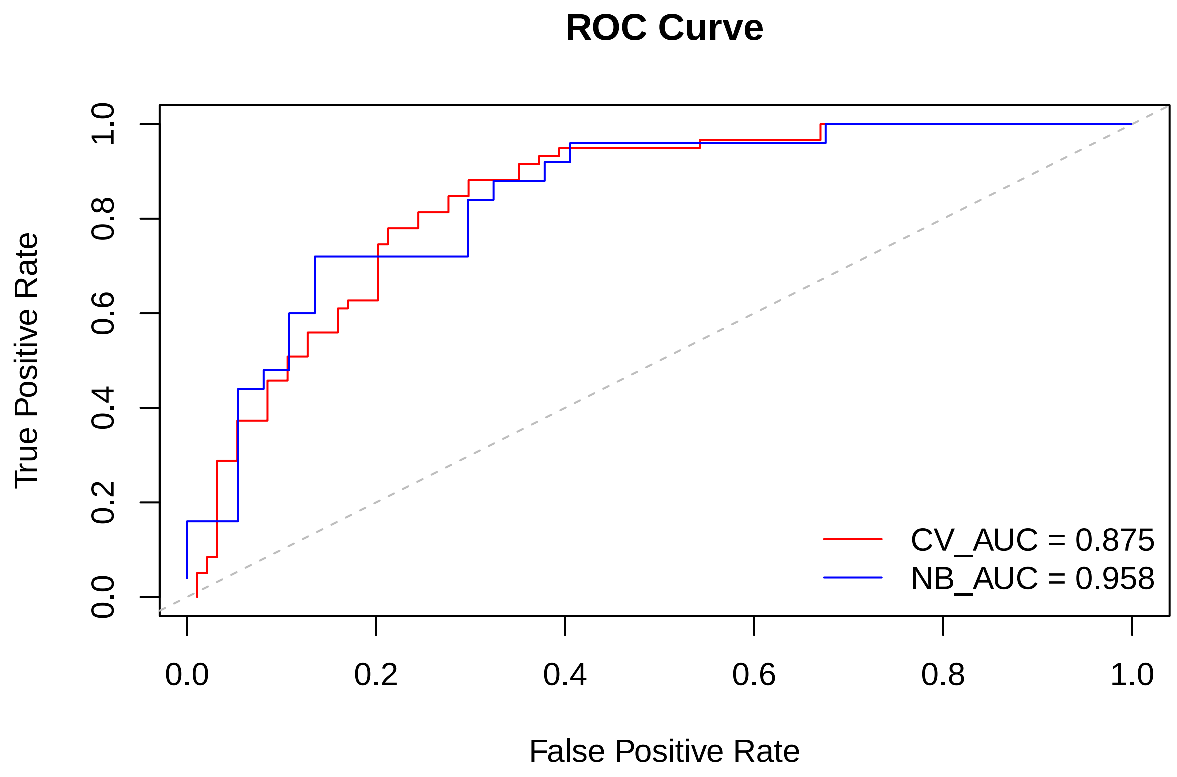
```r
#### Plot the ROC curve ####
par(mfrow = c(1,1))
plot(NB_fpr, NB_tpr, type = "l", main = "ROC Curve", xlab = "False Positive Rate",
     ylab = "True Positive Rate" , col = 'red')

lines(NB_CV_fpr, NB_CV_tpr, type = "l", main = "ROC Curve", xlab = "False Positive Rate",
      ylab = "True Positive Rate" , col = 'blue')

# Add a diagonal line representing random guessing
abline(0, 1, col = "gray", lty = 2)
NB_CV_auc_roc = round(auc(NB_CV_fpr, NB_CV_tpr), 3)
```

```
## Setting levels: control = 0, case = 0.027027027027027

## Setting direction: controls < cases
```

```r
NB_auc_roc = round(auc(NB_fpr, NB_tpr), 3)
```

```
## Setting levels: control = 0.0106382978723404, case = 0.0212765957446809
## Setting direction: controls < cases
```

```r
legend_labels = c(paste("CV_AUC =", NB_CV_auc_roc),
                  paste("NB_AUC =", NB_auc_roc))

legend("bottomright", legend = legend_labels, bty = "n", col = c('red', 'blue'), lty = 1)
```



AS you can see, The AUC of Naive Bayes is higher than Naive Bayes with cross Validation. CV_AUC (Cross-Validation Area Under the Curve) has a value of 0.875. In this case, the Cross-Validation model has an AUC of 0.875, which suggests that it has good discriminative ability in distinguishing between the positive and negative classes.Also, NB_AUC (Naive Bayes Area Under the Curve) has a value of 0.958, which suggests it has even better discriminative ability compared to the Cross-Validation model.

3.10 LDA

```
#### Fit LDA model ####
LDA_model = lda(Outcome ~ ., data = train)

#### Make predictions on test set ####
LDA_pred_train = predict( LDA_model, newdata = train)$class
LDA_pred_test = predict( LDA_model, newdata = test)$class
```

```
#### Calculate confusion matrix for train set ####
LDA_confusion_matrix_train = table( LDA_pred_train, train$Outcome)
LDA_true_positive_train = LDA_confusion_matrix_train[2,2]
LDA_false_positive_train = LDA_confusion_matrix_train[1,2]
LDA_false_negative_train = LDA_confusion_matrix_train[2,1]
LDA_true_negative_train = LDA_confusion_matrix_train[1,1]
```

```
#### Calculate confusion matrix for test set ####
LDA_confusion_matrix_test = table( LDA_pred_test, test$Outcome)
LDA_true_positive_test = LDA_confusion_matrix_test[2,2]
LDA_false_positive_test = LDA_confusion_matrix_test[1,2]
LDA_false_negative_test = LDA_confusion_matrix_test[2,1]
LDA_true_negative_test = LDA_confusion_matrix_test[1,1]
```

```
#### Calculate performance metrics for train set ####
LDA_accuracy_train = (LDA_true_positive_train + LDA_true_negative_train) /
  sum(LDA_confusion_matrix_train)

LDA_precision_train = LDA_true_positive_train /
  (LDA_true_positive_train + LDA_false_positive_train)

LDA_recall_train = LDA_true_positive_train /
  (LDA_true_positive_train + LDA_false_negative_train)

LDA_f1_score_train = 2 * LDA_precision_train * LDA_recall_train /
  (LDA_precision_train + LDA_recall_train)
```

```
#### Calculate performance metrics for train set ####
LDA_accuracy_test = (LDA_true_positive_test + LDA_true_negative_test) /
  sum(LDA_confusion_matrix_test)

LDA_precision_test = LDA_true_positive_test /
  (LDA_true_positive_test + LDA_false_positive_test)

LDA_recall_test = LDA_true_positive_test /
  (LDA_true_positive_test + LDA_false_negative_test)
```

```r
LDA_f1_score_test = 2 * LDA_precision_test * LDA_recall_test /
  (LDA_precision_test + LDA_recall_test)
```

```r
#### Output performance metrics ####
LDA_results_train = data.frame(
  Metric = c("Recall","Precision", "F1 Score","Accuracy"),
  Value = c(LDA_recall_train,LDA_precision_train, LDA_f1_score_train, LDA_accuracy_train),
  set = 'train'
)

LDA_results_test = data.frame(
  Metric = c("Recall", "Precision","F1 Score","Accuracy"),
  Value = c(LDA_recall_test, LDA_precision_test,LDA_f1_score_test, LDA_accuracy_test),
  set = 'test'
)
```

```r
kable(LDA_results_train)
```

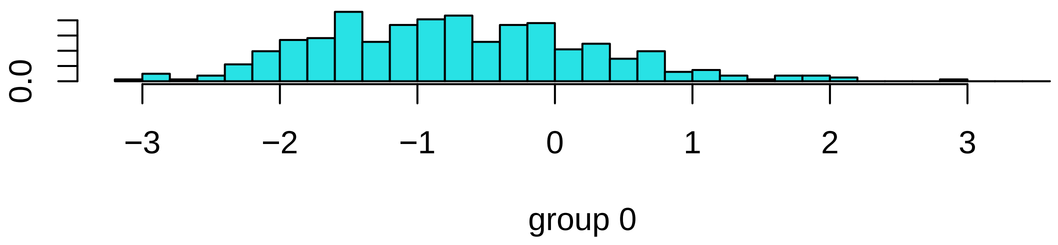
Metric	Value	set
Recall	0.7065868	train
Precision	0.5645933	train
F1 Score	0.6276596	train
Accuracy	0.7723577	train

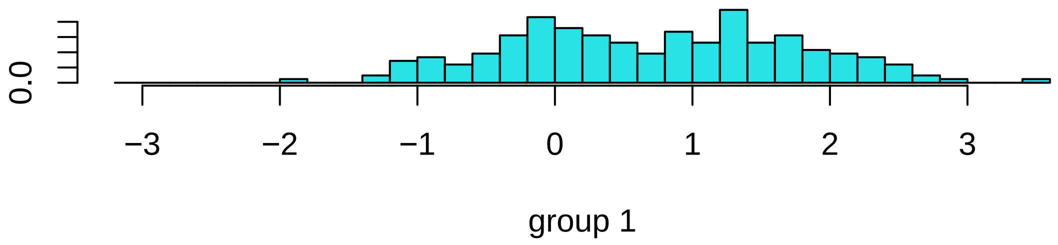
```r
kable(LDA_results_test)
```

Metric	Value	set
Recall	0.7560976	test
Precision	0.5254237	test
F1 Score	0.6200000	test
Accuracy	0.7516340	test

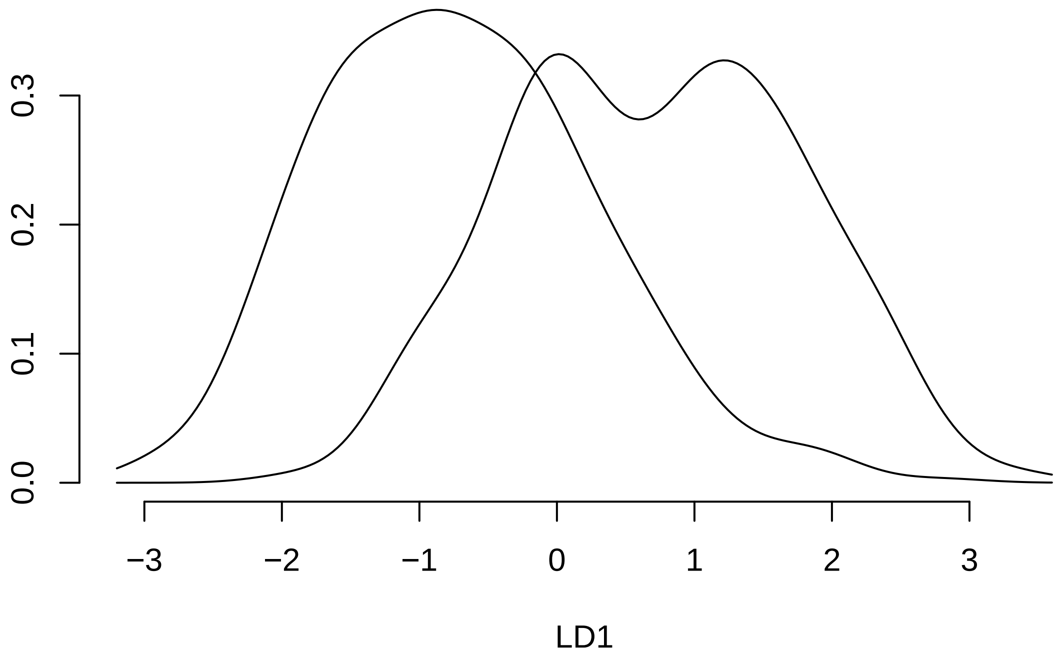
In summary, the model has relatively high recall, indicating that it can identify a good proportion of positive cases. However, the precision is relatively low, suggesting that there are a significant number of false positives. The F1 score takes into account both precision and recall and provides a balanced measure of the model's performance. The accuracy value indicates the overall correctness of the model's predictions.

```r
#### LDA model plots ####
plot(LDA_model)
```





```
plot(LDA_model, type="density")
```



We can see that the LDA model has classify our Output variable in such an acceptable way.

3.11 LDA with cross-validation

```
#### Set the number of folds for cross-validation ####
k = 10

#### Split the data into k folds ####
set.seed(123)
LDA_folds = cut(seq(1, nrow(train)), breaks = k, labels = FALSE)

LDA_CV_test_index = sample(nrow(Diabetes), 0.2 * nrow(Diabetes))
LDA_CV_train = Diabetes[-LDA_CV_test_index, ]
LDA_CV_test = Diabetes[LDA_CV_test_index, ]

LDA_CV_accuracy_test = rep(0, k)
LDA_CV_precision_test = rep(0, k)
LDA_CV_recall_test = rep(0, k)
LDA_CV_f1_score_test = rep(0, k)
```

```
#### Perform k-fold cross-validation ####
for (i in 1:k) {
  #### Subset the data for the i-th fold ####
  LDA_test_indices = which(LDA_folds == i)
  LDA_train_subset = LDA_CV_train[-LDA_test_indices, ]
  LDA_test_subset = LDA_CV_train[LDA_test_indices, ]

  #### Fit the LDA model on the training subset ####
  LDA_CV_model = lda(Outcome ~ ., data = LDA_train_subset)


  #### Make predictions on the test subset ####
  LDA_CV_pred_test = predict(LDA_CV_model, newdata = LDA_test_subset)$class

  #### Calculate confusion matrix and performance metrics for test subset ####
  LDA_CV_confusion_matrix_test = table(LDA_CV_pred_test, LDA_test_subset$Outcome)
  LDA_CV_true_positive_test = LDA_CV_confusion_matrix_test[2, 2]
  LDA_CV_false_positive_test = LDA_CV_confusion_matrix_test[1, 2]
  LDA_CV_false_negative_test = LDA_CV_confusion_matrix_test[2, 1]
  LDA_CV_true_negative_test = LDA_CV_confusion_matrix_test[1, 1]


  LDA_CV_accuracy_test[i] = (LDA_CV_true_positive_test + LDA_CV_true_negative_test) /
    sum(LDA_CV_confusion_matrix_test)

  LDA_CV_precision_test[i] = LDA_CV_true_positive_test /
    (LDA_CV_true_positive_test + LDA_CV_false_positive_test)

  LDA_CV_recall_test[i] = LDA_CV_true_positive_test /
    (LDA_CV_true_positive_test + LDA_CV_false_negative_test)

  LDA_CV_f1_score_test[i] = 2 * LDA_CV_precision_test[i] * LDA_CV_recall_test[i] /
    (LDA_CV_precision_test[i] + LDA_CV_recall_test[i])
}
```

```
#### Calculate average performance metrics across all folds ####
LDA_CV_mean_accuracy = mean(LDA_CV_accuracy_test)
LDA_CV_mean_precision = mean(LDA_CV_precision_test)
LDA_CV_mean_recall = mean(LDA_CV_recall_test)
LDA_CV_mean_f1_score = mean(LDA_CV_f1_score_test)

#### Output performance metrics ####
LDA_CV_results = data.frame(
  Metric = c("CV Recall","CV Precision", "CV F1 Score","CV Accuracy"),
  Value = c(LDA_CV_mean_recall,LDA_CV_mean_precision,LDA_CV_mean_f1_score,LDA_CV_mean_accuracy),
  Set = "Test"
)
```

```
kable(LDA_CV_results)
```

Metric	Value	Set
CV Recall	0.6876245	Test
CV Precision	0.5696065	Test
CV F1 Score	0.6190659	Test
CV Accuracy	0.7690375	Test

The model shows a relatively good recall, capturing a significant proportion of positive cases. However, the precision is relatively lower, suggesting the presence of false positives. The F1 score combines both precision and recall into a single value and provides a balanced measure of the model's performance. The accuracy value represents the overall correctness of the model's predictions on the test set.

Now let's check which of these 2 models are the best one!

```r
LDA_results = data.frame(Metric = c('Recall' , 'Precission' , 'F1-Score' , 'Accuracy'),
                                       LDA = c( 0.75 , 0.52 ,0.62 ,0.75  ),
                                       LDA_Cross_Validation = c(0.68 , 0.56 , 0.61 , 0.76))


kable(LDA_results)
```

Metric	LDA	LDA_Cross_Validation
Recall	0.75	0.68
Precission	0.52	0.56
F1-Score	0.62	0.61
Accuracy	0.75	0.76

As you can see the LDA model with cross-validation generally performs slightly worse than the LDA model across all metrics. So we can consider the LDA model as our selection from these two models.So far we have selected Logistic regression model with cross validation, KNN with cross Validation,Naive Bayes model with cross validation, and LDA model.

```r
Confusion_Matrix_LDA = data.frame(class = c('Non-Diabetic' , 'Diabetic'),
                                  'LDA-Non-Diabetic' = c(84 , 10) ,
                                  'LDA-Diabetic' = c(28 , 31),
                                  'LDA_CV-Non-Diabetic' = c(36 , 4) ,
                                  'LDA_CV-Diabetic' = c(8 , 14))

kable(Confusion_Matrix_LDA)
```

class	LDA.Non.Diabetic	LDA.Diabetic	LDA_CV.Non.Diabetic	LDA_CV.Diabetic
Non-Diabetic	84	28	36	8
Diabetic	10	31	4	14

We can conclude the same results as before by the outputs of the confusion matrix.

3.12 QDA Model

```r
#### Fit QDA model ####
QDA_model = qda(Outcome ~ ., data = train)

#### Make predictions on test set ####
QDA_pred_train = predict(QDA_model, newdata = train)$class
QDA_pred_test = predict(QDA_model, newdata = test)$class
```

```r
#### Calculate confusion matrix for train data ####
QDA_confusion_matrix_train = table( QDA_pred_train, train$Outcome)
QDA_true_positive_train = QDA_confusion_matrix_train[2,2]
QDA_false_positive_train = QDA_confusion_matrix_train[1,2]
QDA_false_negative_train = QDA_confusion_matrix_train[2,1]
QDA_true_negative_train = QDA_confusion_matrix_train[1,1]
```

```r
#### Calculate confusion matrix for test data ####
QDA_confusion_matrix_test = table( QDA_pred_test, test$Outcome)
QDA_true_positive_test = QDA_confusion_matrix_test[2,2]
QDA_false_positive_test = QDA_confusion_matrix_test[1,2]
QDA_false_negative_test = QDA_confusion_matrix_test[2,1]
QDA_true_negative_test = QDA_confusion_matrix_test[1,1]
```

```r
#### Calculate performance metrics for train set ####
QDA_accuracy_train = (QDA_true_positive_train + QDA_true_negative_train) /
sum(QDA_confusion_matrix_train)

QDA_precision_train = LDA_true_positive_train /
(QDA_true_positive_train + QDA_false_positive_train)

QDA_recall_train = QDA_true_positive_train /
(QDA_true_positive_train + QDA_false_negative_train)

QDA_f1_score_train = 2 * QDA_precision_train * QDA_recall_train /
(QDA_precision_train + QDA_recall_train)
```

```r
#### Calculate performance metrics for train set ####
QDA_accuracy_test = (QDA_true_positive_test + QDA_true_negative_test) /
  sum(QDA_confusion_matrix_test)

QDA_precision_test = QDA_true_positive_test /
  (QDA_true_positive_test + QDA_false_positive_test)

QDA_recall_test = QDA_true_positive_test /
  (QDA_true_positive_test + QDA_false_negative_test)

QDA_f1_score_test = 2 * QDA_precision_test * QDA_recall_test /
  (QDA_precision_test + QDA_recall_test)
```

```r
#### Output performance metrics ####
QDA_results_train = data.frame(
  Metric = c("Recall","Precision", "F1 Score","Accuracy"),
  Value = c(QDA_recall_train,QDA_precision_train,QDA_f1_score_train,QDA_accuracy_train),
  set = 'train'
)

QDA_results_test = data.frame(
  Metric = c("Recall", "Precision","F1 Score","Accuracy"),
  Value = c(QDA_recall_test,QDA_precision_test,QDA_f1_score_test,QDA_accuracy_test),
  set = 'test'
)
```

```r
kable(QDA_results_train)
```

Metric	Value	set
Recall	0.7225434	train
Precision	0.5645933	train
F1 Score	0.6338770	train
Accuracy	0.7853659	train

```r
kable(QDA_results_test)
```

Metric	Value	set
Recall	0.7209302	test
Precision	0.5254237	test
F1 Score	0.6078431	test
Accuracy	0.7385621	test

There are some good results for recall and accuracy but there is room for improvement in precision, and the overall F1 score. Let's see if we can improve it by applying cross-validation.

3.13 QDA with cross-validation.

```r
# Set the number of folds for cross-validation
k = 10

# Split the data into k folds
set.seed(123)
QDA_folds = cut(seq(1, nrow(train)), breaks = k, labels = FALSE)

QDA_CV_test_index = sample(nrow(Diabetes), 0.2 * nrow(Diabetes))
QDA_CV_train = Diabetes[-QDA_CV_test_index, ]
QDA_CV_test = Diabetes[QDA_CV_test_index, ]

QDA_CV_accuracy_test = rep(0, k)
QDA_CV_precision_test = rep(0, k)
QDA_CV_recall_test = rep(0, k)
QDA_CV_f1_score_test = rep(0, k)
```

```r
#### Perform k-fold cross-validation ####
for (i in 1:k) {
  #### Subset the data for the i-th fold ####
  QDA_test_indices = which(QDA_folds == i)
  QDA_train_subset = QDA_CV_train[-QDA_test_indices, ]
  QDA_test_subset = QDA_CV_train[QDA_test_indices, ]

  #### Fit the LDA model on the training subset ####
  QDA_CV_model = qda(Outcome ~ ., data = QDA_train_subset)
  #### Make predictions on the test subset ####
  QDA_CV_pred_test = predict(QDA_CV_model, newdata = QDA_test_subset)$class

  #### Calculate confusion matrix and performance metrics for test subset ####
  QDA_CV_confusion_matrix_test = table(QDA_CV_pred_test, QDA_test_subset$Outcome)
  QDA_CV_true_positive_test = QDA_CV_confusion_matrix_test[2, 2]
  QDA_CV_false_positive_test = QDA_CV_confusion_matrix_test[1, 2]
  QDA_CV_false_negative_test = QDA_CV_confusion_matrix_test[2, 1]
  QDA_CV_true_negative_test = QDA_CV_confusion_matrix_test[1, 1]


  QDA_CV_accuracy_test[i] = (QDA_CV_true_positive_test + QDA_CV_true_negative_test) /
    sum(QDA_CV_confusion_matrix_test)

  QDA_CV_precision_test[i] = QDA_CV_true_positive_test /
    (QDA_CV_true_positive_test + QDA_CV_false_positive_test)

  QDA_CV_recall_test[i] = QDA_CV_true_positive_test /
    (QDA_CV_true_positive_test + QDA_CV_false_negative_test)

  QDA_CV_f1_score_test[i] = 2 * QDA_CV_precision_test[i] * QDA_CV_recall_test[i] /
    (QDA_CV_precision_test[i] + QDA_CV_recall_test[i])
}
```

```
#### Calculate average performance metrics across all folds ####
QDA_CV_mean_accuracy = mean(QDA_CV_accuracy_test)
QDA_CV_mean_precision = mean(QDA_CV_precision_test)
QDA_CV_mean_recall = mean(QDA_CV_recall_test)
QDA_CV_mean_f1_score = mean(QDA_CV_f1_score_test)
```

```
#### Output performance metrics ####
QDA_CV_results = data.frame(
  Metric = c("CV Recall","CV Precision", "CV F1 Score","CV Accuracy"),
  Value = c(QDA_CV_mean_recall,QDA_CV_mean_precision,
            QDA_CV_mean_f1_score, QDA_CV_mean_accuracy),
  Set = "Test"
)
```

```
kable(QDA_CV_results)
```

Metric	Value	Set
CV Recall	0.6593632	Test
CV Precision	0.5422734	Test
CV F1 Score	0.5920100	Test
CV Accuracy	0.7511370	Test

As you can see , the model demonstrates moderate performance in terms of accuracy, recall, precision, and the overall F1 score.

Now let's compare these results for two available QDA models.

```
QDA_results = data.frame(Metric = c('Recall' , 'Precission' , 'F1-Score' , 'Accuracy'),
                         QDA = c( 0.72 , 0.52 ,0.60 ,0.73),
                         QDA_Cross_Validation = c(0.65 , 0.54 , 0.59 , 0.75))


kable(QDA_results)
```

Metric	QDA	QDA_Cross_Validation
Recall	0.72	0.65
Precission	0.52	0.54
F1-Score	0.60	0.59
Accuracy	0.73	0.75

The QDA model's performance is similar in both the non-cross-validated and cross-validated scenarios, with slightly lower values in the cross-validated results. As we said we really do care about having higher recall, so among these two models we choose QDA model. So far we have selected Logistic regression model with cross validation, KNN with cross Validation, Naive Bayes model with cross validation, LDA model, and QDA.

```
Confusion_Matrix_QDA = data.frame(class = c('Non-Diabetic' , 'Diabetic'),
                                  'QDA-Non-Diabetic' = c(82 , 12) ,
                                  'QDA-Diabetic' = c(28 , 31),
                                  'QDA_CV-Non-Diabetic' = c(35 , 5) ,
                                  'CV-Diabetic' = c(9 , 13))

kable(Confusion_Matrix_QDA)
```

class	QDA.Non.Diabetic	QDA.Diabetic	QDA_CV.Non.Diabetic	CV.Diabetic
Non-Diabetic	82	28	35	9
Diabetic	12	31	5	13

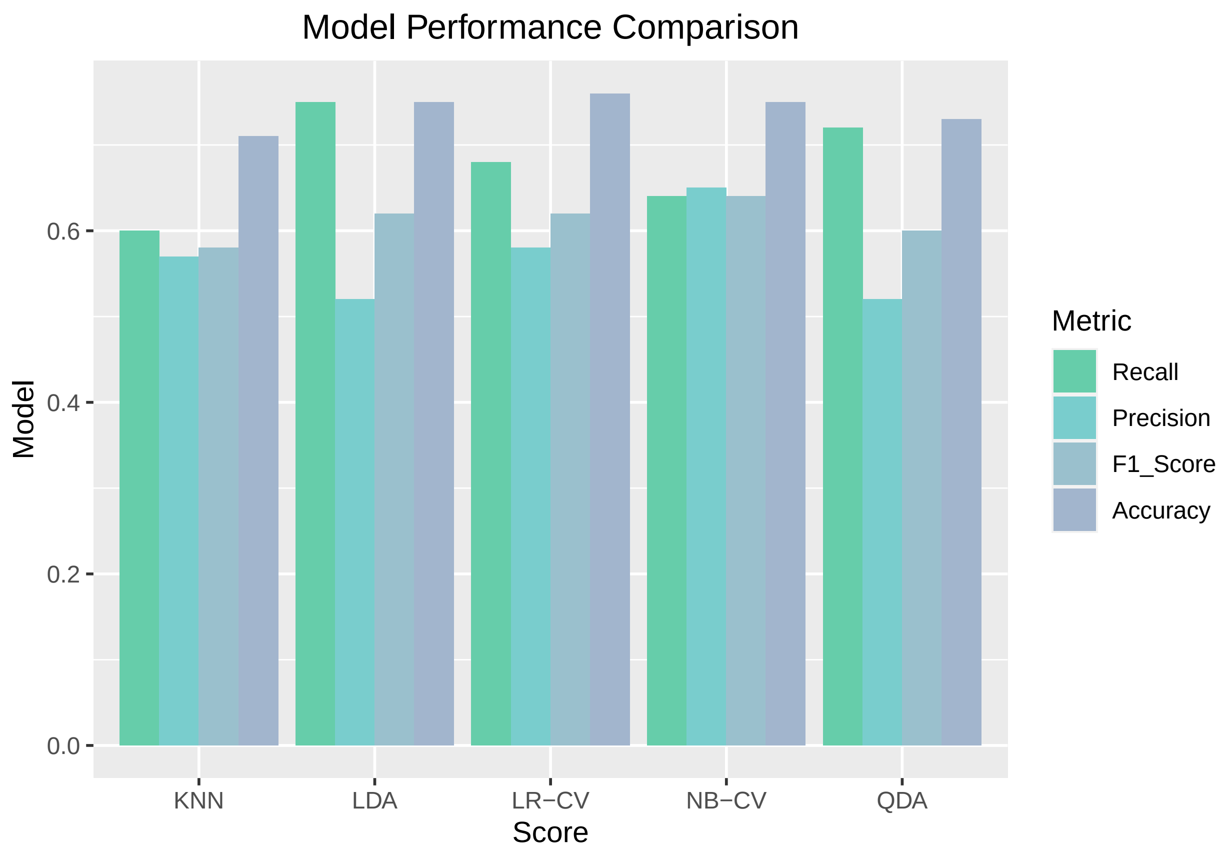
We can conclude the same result from confusion matrix as available metric.

4 COMPARING DIFFERENT MODELS

```
results_df = data.frame(
  Model = c('LR-CV', 'KNN', 'NB-CV', 'LDA' , 'QDA'),
  Recall = c(0.68, 0.60, 0.64, 0.75, 0.72),
  Precision = c(0.58, 0.57, 0.65, 0.52, 0.52),
  F1_Score = c(0.62, 0.58, 0.64, 0.62, 0.6),
  Accuracy = c(0.76, 0.71, 0.75, 0.75, 0.73)
)
```

```
#### Reshape data into long format ####
results_long = reshape2::melt(results_df, id.vars = "Model", variable.name = "Metric")
my_colors = c("#66CDAA", "#79CDCD", "#9AC0CD", "#A2B5CD")
#### Plot the results ####
```

```
ggplot(results_long, aes(x = Model, y = value, fill = Metric)) +
  geom_bar(stat = "identity", position = "dodge") +
  labs(y = "Model", x = "Score", fill = "Metric") +
  ggtitle("Model Performance Comparison") +
  theme(plot.title = element_text(hjust = 0.5))+
  scale_fill_manual(values = my_colors)
```



5 CONCLUSION

Okay, now that we have fitted different models, it's time to choose the best model among all the available options. We have chosen the LDA model as our preferred choice. This model demonstrates a good recall and accuracy value, along with satisfactory precision and F1-score. However, it's worth noting that the choice of the best model depends on various factors, including the specific requirements and priorities of the problem at hand.

While we focused on the models mentioned above, it's important to mention that there are other classification architectures available, such as random forest, SVM, decision tree, Ada boosting, and more. These models may have different strengths and weaknesses and could potentially yield different results for the given problem.

In the context of diabetes prediction, where the consequences of false negatives can be significant, prioritizing high recall is crucial. However, it's also essential to consider other factors such as computational complexity, interpretability of the model, and potential trade-offs between different evaluation metrics.

Therefore, it's advisable to perform further analysis and experimentation, considering additional models and possibly exploring ensemble methods or model combination approaches to improve the predictive performance.