Telco Customer Churn

Focused customer retention programs

A. DATA OVERVIEW

This dataset is used to create a model that can predict when a customer of Telco is likely to leave the company. This is important because it allows us to take steps to prevent the customers from leaving. Keeping customers is easier than acquiring new ones, so it's important to do everything we can to keep them happy and stay using the company product.

This dataset contains information about our customers demographic and their Telco accounts, including their gender, age range, whether they have partners and dependents, how long they've been a customer, their contract type, payment method, paperless billing status, monthly charges, and total charges. This information can help us understand what makes our customers stick with our products and what makes them leave us, and then we can create models to predict whether or not our current customers will leave us in the future.

To better understand our dataset, here is a brief description of each column in the dataset:

- **CustomerID**: Customer ID.
- **gender**: Whether the customer is male or female.
- **SeniorCitizen**: Whether the customer is a senior citizen or not (1,0)
- **Partner**: Whether the customer has a partner or not (Yes, No)
- **Dependents**: Whether the customer has dependents or not (Yes, No)
- **tenure**: Number of months the customer has stayed with the company .
- **PhoneService**: Whether the customer has a phone service or not (Yes, No)
- **MultipleLines**: Whether the customer has multiple lines or not (Yes, No, No phone service)
- **InternetService**: Customer’s internet service provider (DSL, Fiber optic, No)
- **OnlineSecurity**: Whether the customer has online security or not (Yes, No, No internet service)
- **OnlineBackup**: Whether the customer has online backup or not (Yes, No, No internet service)
- **DeviceProtection**: Whether the customer has device protection or not (Yes, No, No internet service)
- **TechSupport**: Whether the customer has tech support or not (Yes, No, No internet service)
- **StreamingTV**: Whether the customer has streaming TV or not (Yes, No, No internet service)

- **StreamingMovies**: Whether the customer has streaming movies or not (Yes, No, No internet service)
- **Contract**: The contract term of the customer (Month-to-month, One year, Two year)
- **PaperlessBilling**: Whether the customer has paperless billing or not (Yes, No)
- **PaymentMethod**: The customer's payment method (Electronic check, Mailed check, Bank transfer (automatic), Credit card (automatic))
- **MonthlyCharges**: The amount charged to the customer monthly.
- **TotalCharges**: The total amount charged to the customer.
- **Churn**: Whether the customer churned or not (Yes or No)

*The churn column is the column that we will predict.

B. EXPLORATORY DATA ANALYSIS (EDA)

Before we can build a model to predict whether the customer will churn or leave the company, we need to understand the data that we have. In this section, we will take a closer look at our data to learn more about it. We will start by finding out how much data we have, what kind of data it is, and how it is structured. Then, we will look for any anomalies. If we find any anomalies, we will need to handle them. Finally, we will use visualization to help us understand the relationships between the different variables in our data.

Step 1: Import all required libraries

```
library(dplyr)
library(ggplot2)
library(plotly)
library(gridExtra)
library(reshape2)
library(ROSE)
library(caTools)
library(randomForest)
library(caret)
```

We use the above code to import libraries that will help us predict customer churn. These libraries provide us with the tools we need to analyze customer data and identify patterns that may indicate a customer is at risk of leaving.

Step 2: Load the dataset

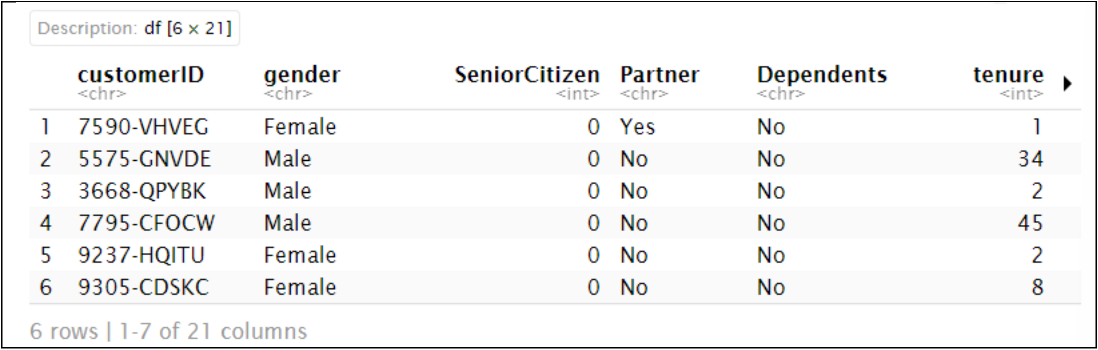
```
df = read.csv('~/pramudya/matkul/data mining/LEC & LAB/LAB
Project/WA_Fn-UseC_-Telco-Customer-Churn.csv')
```

We use the above code to read Telco Customer Churn dataset and load it into R.

Step 3: Explore the data to find general information

```
head(df)
```

We use head(df) to show us the first six rows from our dataset. By reading through the output, we know that the output briefly informs us that **our dataset not only contains numerical data but also categorical data.**



Description: df [6 x 21]

	customerID <chr>	gender <chr>	SeniorCitizen <int>	Partner <chr>	Dependents <chr>	tenure <int>
1	7590-VHVEG	Female	0	Yes	No	1
2	5575-GNVDE	Male	0	No	No	34
3	3668-QPYBK	Male	0	No	No	2
4	7795-CFOCW	Male	0	No	No	45
5	9237-HQITU	Female	0	No	No	2
6	9305-CDSKC	Female	0	No	No	8

6 rows | 1-7 of 21 columns

```
dim(df)
```

We use the above code to get the dimensions of **df**. The function returns a tuple of two numbers which is the number of rows and columns.

```
[1] 7043    21
```

In our case, the data frame has **7043 rows and 21 columns**.

```
str(df)
```

We use the above code to get a summary of the structure of our data frame, including the names of the variables, the data types of the variables, and the first few values of each variable.

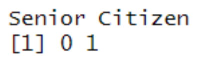
```
'data.frame':	7043 obs. of  21 variables:
 $ customerID      : chr  "7590-VHVEG" "5575-GNVDE" "3668-QPYBK" "7795-CFOCW" ...
 $ gender          : chr  "Female" "Male" "Male" "Male" ...
 $ SeniorCitizen   : int  0 0 0 0 0 0 0 0 0 0 ...
 $ Partner         : chr  "Yes" "No" "No" "No" ...
 $ Dependents      : chr  "No" "No" "No" "No" ...
 $ tenure          : int  1 34 2 45 2 8 22 10 28 62 ...
 $ PhoneService    : chr  "No" "Yes" "Yes" "No" ...
 $ MultipleLines   : chr  "No phone service" "No" "No" "No phone service" ...
 $ InternetService : chr  "DSL" "DSL" "DSL" "DSL" ...
 $ OnlineSecurity  : chr  "No" "Yes" "Yes" "Yes" ...
 $ OnlineBackup    : chr  "Yes" "No" "Yes" "No" ...
 $ DeviceProtection: chr  "No" "Yes" "No" "Yes" ...
 $ TechSupport     : chr  "No" "No" "No" "Yes" ...
 $ StreamingTV     : chr  "No" "No" "No" "No" ...
 $ StreamingMovies : chr  "No" "No" "No" "No" ...
 $ Contract        : chr  "Month-to-month" "One year" "Month-to-month" "One year"
...
 $ PaperlessBilling: chr  "Yes" "No" "Yes" "No" ...
 $ PaymentMethod   : chr  "Electronic check" "Mailed check" "Mailed check" "Bank
transfer (automatic)" ...
 $ MonthlyCharges  : num  29.9 57 53.9 42.3 70.7 ...
 $ TotalCharges    : num  29.9 1889.5 108.2 1840.8 151.7 ...
 $ Churn           : chr  "No" "No" "Yes" "No" ...
```

By looking at the output, we can see that most of our data is categorical and only the columns SeniorCitizen, tenure, MonthlyCharges, and TotalCharges are numerical. However, if we look at the description of each variable in Kaggle, we can see that SeniorCitizen variable is a variable that shows whether the customer is a senior citizen or not, so maybe **we can change the value into “yes” or “no” like the rest of the categorical variables.**

Step 4: Handle the data type for some variables

```
cat('Senior Citizen\n')
unique(df$SeniorCitizen)
```

We use the above code to see all unique values in variable SeniorCitizen.



```
Senior Citizen
[1] 0 1
```

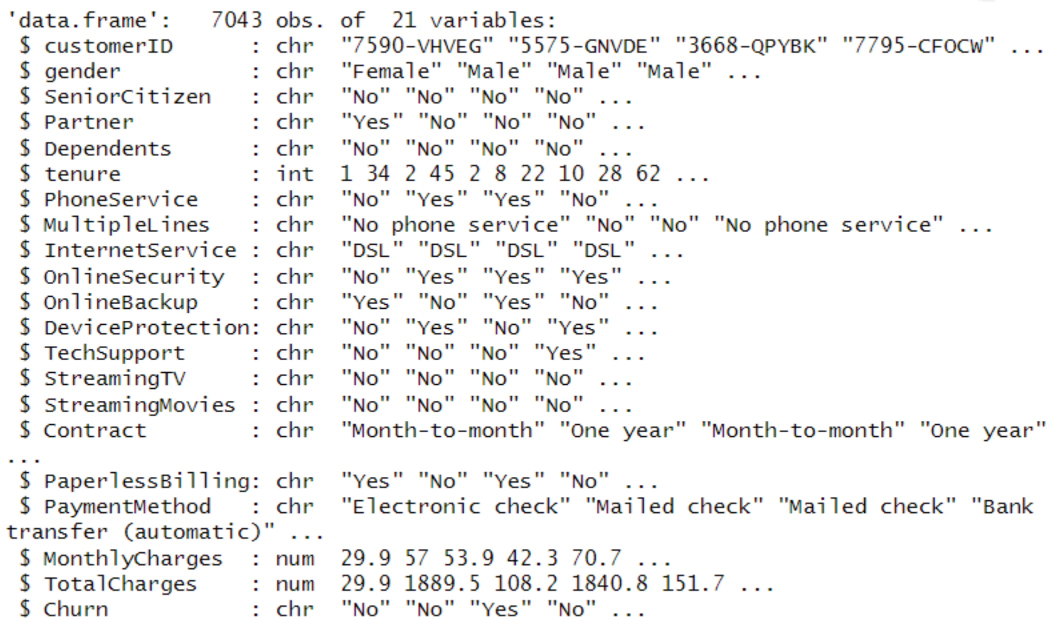
The output reveals that **senior citizen only has two unique values which is 0 and 1.**

```
df$SeniorCitizen <- ifelse(df$SeniorCitizen == 1, "Yes", "No")
```

We use the above code to **change the value of the SeniorCitizen variable from a number to a word.** If the value is 1, we change it to "Yes". If the value is 0, we change it to "No". This will make us easier to understand the data and to analyze it.

```
str(df)
```

We use the above code to get a summary of the current structure of our data frame.

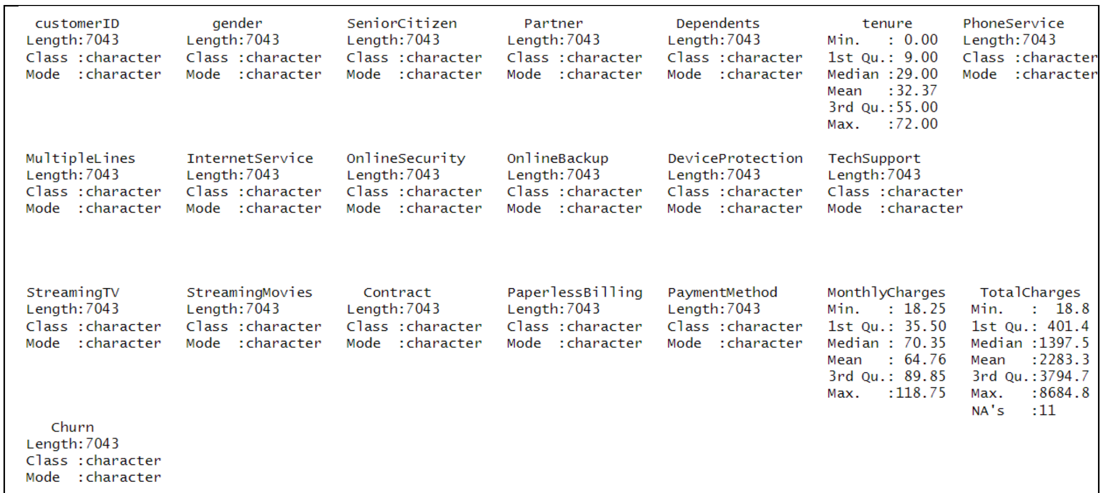


```
'data.frame':	7043 obs. of  21 variables:
 $ customerID      : chr  "7590-VHVEG" "5575-GNVDE" "3668-QPYBK" "7795-CFOCW" ...
 $ gender          : chr  "Female" "Male" "Male" "Male" ...
 $ SeniorCitizen   : chr  "No" "No" "No" "No" ...
 $ Partner         : chr  "Yes" "No" "No" "No" ...
 $ Dependents      : chr  "No" "No" "No" "No" ...
 $ tenure          : int  1 34 2 45 2 8 22 10 28 62 ...
 $ PhoneService    : chr  "No" "Yes" "Yes" "No" ...
 $ MultipleLines   : chr  "No phone service" "No" "No" "No phone service" ...
 $ InternetService : chr  "DSL" "DSL" "DSL" "DSL" ...
 $ OnlineSecurity  : chr  "No" "Yes" "Yes" "Yes" ...
 $ OnlineBackup    : chr  "Yes" "No" "Yes" "No" ...
 $ DeviceProtection: chr  "No" "Yes" "No" "Yes" ...
 $ TechSupport     : chr  "No" "No" "No" "Yes" ...
 $ StreamingTV     : chr  "No" "No" "No" "No" ...
 $ StreamingMovies : chr  "No" "No" "No" "No" ...
 $ Contract        : chr  "Month-to-month" "One year" "Month-to-month" "One year"
...
 $ PaperlessBilling: chr  "Yes" "No" "Yes" "No" ...
 $ PaymentMethod   : chr  "Electronic check" "Mailed check" "Mailed check" "Bank
transfer (automatic)" ...
 $ MonthlyCharges  : num  29.9 57 53.9 42.3 70.7 ...
 $ TotalCharges    : num  29.9 1889.5 108.2 1840.8 151.7 ...
 $ Churn           : chr  "No" "No" "Yes" "No" ...
```

As we can see now the **SeniorCitizen has become a character variable with values “Yes” or “No”.**

```
summary(df)
```

We use the above code to show us the summary of our dataframe. This function will return the min, 1st quantile, median, mean, 3rd quantile, max, and missing values (if any) for numerical variable. Howefer this function will return only the length, class, and mode for categorical variable.



```
  customerID           gender          SeniorCitizen        Partner           Dependents            tenure       PhoneService
 Length:7043        Length:7043        Length:7043        Length:7043        Length:7043        Min.   : 0.00   Length:7043
 Class :character   Class :character   Class :character   Class :character   Class :character   1st Qu.: 9.00   Class :character
 Mode  :character   Mode  :character   Mode  :character   Mode  :character   Mode  :character   Median :29.00   Mode  :character
                                                                                                Mean   :32.37
                                                                                                3rd Qu.:55.00
                                                                                                Max.   :72.00

 MultipleLines      InternetService    OnlineSecurity     OnlineBackup       DeviceProtection   TechSupport
 Length:7043        Length:7043        Length:7043        Length:7043        Length:7043        Length:7043
 Class :character   Class :character   Class :character   Class :character   Class :character   Class :character
 Mode  :character   Mode  :character   Mode  :character   Mode  :character   Mode  :character   Mode  :character



 StreamingTV        StreamingMovies      Contract         PaperlessBilling   PaymentMethod      MonthlyCharges    TotalCharges
 Length:7043        Length:7043        Length:7043        Length:7043        Length:7043        Min.   : 18.25   Min.   :  18.8
 Class :character   Class :character   Class :character   Class :character   Class :character   1st Qu.: 35.50   1st Qu.: 401.4
 Mode  :character   Mode  :character   Mode  :character   Mode  :character   Mode  :character   Median : 70.35   Median :1397.5
                                                                                                Mean   : 64.76   Mean   :2283.3
                                                                                                3rd Qu.: 89.85   3rd Qu.:3794.7
                                                                                                Max.   :118.75   Max.   :8684.8
                                                                                                                 NA's   :11
    Churn
 Length:7043
 Class :character
 Mode  :character
```

As we can see most of our dataset is a categorical variable so it is hard to understand the summary of our dataset. Since all of our categorical variable except customerID is only contains values like "Yes" or "No" **we can change all that variable into a factor instead.** Another reason why we have to convert the categorical data to factor is because later when we create the predictive models, it cannot directly use categorical variables, so they must first be converted into factors.

```
df$gender <- factor(df$gender)
df$SeniorCitizen <- factor(df$SeniorCitizen)
df$Partner <- factor(df$Partner)
df$Dependents <- factor(df$Dependents)
df$PhoneService <- factor(df$PhoneService)
df$MultipleLines <- factor(df$MultipleLines)
df$InternetService <- factor(df$InternetService)
df$OnlineSecurity <- factor(df$OnlineSecurity)
df$OnlineBackup <- factor(df$OnlineBackup)
df$DeviceProtection <- factor(df$DeviceProtection)
df$TechSupport <- factor(df$TechSupport)
df$StreamingTV <- factor(df$StreamingTV)
df$StreamingMovies <- factor(df$StreamingMovies)
df$Contract <- factor(df$Contract)
df$PaperlessBilling <- factor(df$PaperlessBilling)
df$PaymentMethod <- factor(df$PaymentMethod)
df$Churn <- factor(df$Churn)
```

We use the above code to **convert variable** *gender*, *SeniorCitizen*, *Partner*, *Dependents*, *PhoneService*, *MultipleLiner*, *InternetService*, *OnlineSecurity*, *OnlineBackup*, *DeviceProtection*, *TechSupport*, *StreamingTV, StreamingMovies*, *Contract*, *PaperlessBilling*, *PaymentMethod*, and *Churn* **into a factor**.

```
summary(df)
```

We use the code above to display a summary of our current dataframe. Since we have converted a categorical variable into a factor, we can now see the number of levels of each factor variable. For example, we can see that there are fewer senior citizens than non-senior citizens.

```
  customerID           gender    SeniorCitizen Partner    Dependents     tenure      PhoneService          MultipleLines
 Length:7043        Female:3488   No :5901      No :3641   No :4933   Min.   : 0.00   No : 682     No              :3390
 Class :character   Male  :3555   Yes:1142      Yes:3402   Yes:2110   1st Qu.: 9.00   Yes:6361     No phone service: 682
 Mode  :character                                                     Median :29.00                Yes             :2971
                                                                      Mean   :32.37
                                                                      3rd Qu.:55.00
                                                                      Max.   :72.00

    InternetService             OnlineSecurity              OnlineBackup             DeviceProtection              TechSupport
 DSL        :2421   No                 :3498   No                 :3088   No                 :3095   No                 :3473
 Fiber optic:3096   No internet service:1526   No internet service:1526   No internet service:1526   No internet service:1526
 No         :1526   Yes                :2019   Yes                :2429   Yes                :2422   Yes                :2044



             StreamingTV              StreamingMovies           Contract    PaperlessBilling                   PaymentMethod
 No                 :2810   No                 :2785   Month-to-month:3875   No :2872         Bank transfer (automatic):1544
 No internet service:1526   No internet service:1526   One year      :1473   Yes:4171         Credit card (automatic)  :1522
 Yes                :2707   Yes                :2732   Two year      :1695                    Electronic check         :2365
                                                                                              Mailed check             :1612



 MonthlyCharges    TotalCharges     Churn
 Min.   : 18.25   Min.   :  18.8   No :5174
 1st Qu.: 35.50   1st Qu.: 401.4   Yes:1869
 Median : 70.35   Median :1397.5
 Mean   : 64.76   Mean   :2283.3
 3rd Qu.: 89.85   3rd Qu.:3794.7
 Max.   :118.75   Max.   :8684.8
                  NA's   :11
```

In addition, we can also see that **the only variable that has missing values is the TotalCharges variable.**

Step 5: Look for missing values and deal with them if any

```
colSums(is.na(df))
```

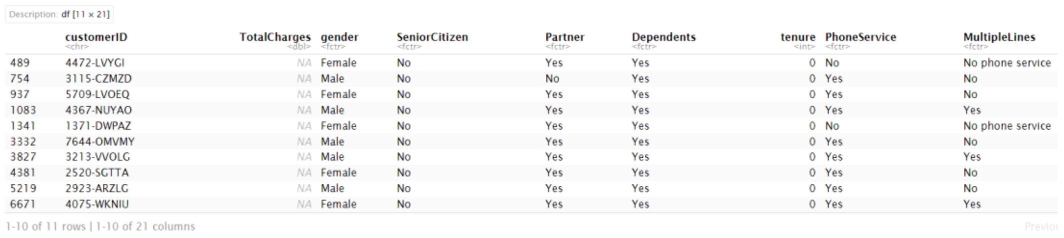
To make sure that the only variable with missing values is TotalCharges, we will use the code above. The code will count and display the number of missing values for each column in our data frame.

```
      customerID           gender    SeniorCitizen          Partner       Dependents           tenure     PhoneService 
               0                0                0                0                0                0                0 
   MultipleLines  InternetService   OnlineSecurity     OnlineBackup DeviceProtection      TechSupport      StreamingTV 
               0                0                0                0                0                0                0 
 StreamingMovies         Contract PaperlessBilling    PaymentMethod   MonthlyCharges     TotalCharges            Churn 
               0                0                0                0                0               11                0 
```

By looking at the output, it turns out to be correct, **only the TotalCharges variable has missing values** and it has 11 missing values.

```
df_na <- df[is.na(df$TotalCharges), ]
relocate(df_na, customerID, TotalCharges)
```

To decide how to handle the missing values, first I want to look at each row that has missing values. By using the code above, we will display every row that has missing values and move the TotalCharges column next to the Customer ID so that it will be easier to analyze.


Description: df [11 x 21]

	customerID <chr>	TotalCharges <dbl>	gender <fctr>	SeniorCitizen <fctr>	Partner <fctr>	Dependents <fctr>	tenure <int>	PhoneService <fctr>	MultipleLines <fctr>
489	4472-LVYGI	NA	Female	No	Yes	Yes	0	No	No phone service
754	3115-CZMZD	NA	Male	No	No	Yes	0	Yes	No
937	5709-LVOEQ	NA	Female	No	Yes	Yes	0	Yes	No
1083	4367-NUYAO	NA	Male	No	Yes	Yes	0	Yes	Yes
1341	1371-DWPAZ	NA	Female	No	Yes	Yes	0	No	No phone service
3332	7644-OMVMY	NA	Male	No	Yes	Yes	0	Yes	No
3827	3213-VVOLG	NA	Male	No	Yes	Yes	0	Yes	Yes
4381	2520-SGTTA	NA	Female	No	Yes	Yes	0	Yes	No
5219	2923-ARZLG	NA	Male	No	Yes	Yes	0	Yes	No
6671	4075-WKNIU	NA	Female	No	Yes	Yes	0	Yes	Yes

1-10 of 11 rows | 1-10 of 21 columns

When we look at the data, we can see that all of the rows where TotalCharges is missing also have tenure equal to 0. This means that all of the customers who have missing TotalCharges values have been customers for less than 1 month. We know that charges are counted at the end of the month, so this tells us that **the missing**

TotalCharges values are caused by the fact that these customers have not been customers for long enough to have any charges yet.

```
df$TotalCharges[is.na(df$TotalCharges)] = 0
colSums(is.na(df))
```

Because the missing TotalCharges values are caused by the fact that these customers have not been customers for long enough to have any charges yet, we used the above code to change all the missing values into 0 and then the code will display the number of missing values for each column in our data frame.

```
      customerID           gender    SeniorCitizen          Partner       Dependents           tenure     PhoneService 
               0                0                0                0                0                0                0 
   MultipleLines  InternetService   OnlineSecurity     OnlineBackup DeviceProtection      TechSupport      StreamingTV 
               0                0                0                0                0                0                0 
 StreamingMovies         Contract PaperlessBilling    PaymentMethod   MonthlyCharges     TotalCharges            Churn 
               0                0                0                0                0                0                0 
```

When we look at the output, we can see that the dataframe no longer contains any missing values. This means that **we were successful in handling the missing values**.

Step 6: Look for duplicated data and deal with them if any

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

we use the code above to count the number of duplicated rows in the dataframe. The duplicated() function will returns a logical vector (TRUE/FALSE) indicating whether each row in a dataframe is a duplicate of a previous row and then the sum() function will calculate the sum of the TRUE values in the vector.

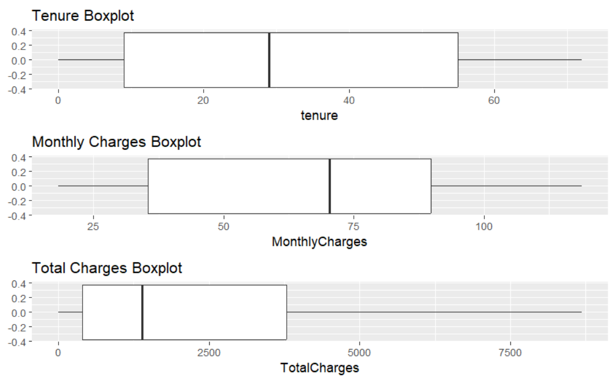
```
[1] 0
```

Because the output is "0", it means that **our dataframe has no duplicated rows.**

Step 7: Look for outliers and deal with them if any

```
plot1 <- ggplot(df, aes(x=tenure)) + geom_boxplot() + ggtitle('Tenure Boxplot')
plot2 <- ggplot(df, aes(x=MonthlyCharges)) + geom_boxplot()  + ggtitle('Monthly Charges Boxplot')
plot3 <- ggplot(df, aes(x=TotalCharges)) + geom_boxplot() + ggtitle('Total Charges Boxplot')
grid.arrange(plot1, plot2, plot3, nrow = 3)
```

We use the above code to create a boxplot for the variable tenure, MonthlyCharges, and TotalCharges. The boxplots can be used to visually identify potential outliers in the data.

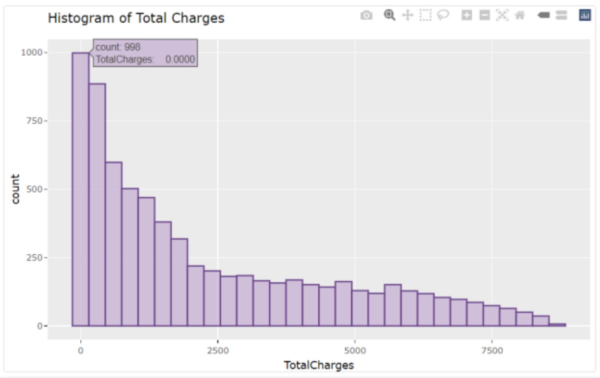


When we look at the plots, we can see that there are no data points that fall outside the whiskers. **This means that the variables tenure, MonthlyCharges, and TotalCharges don’t have any outliers.**

Step 8: create visualizations to gain additional information

```
plot4 <- ggplot(df, aes(x=TotalCharges)) + geom_histogram(bins=30,
fill="#C3ACD0", alpha=0.7, color="#674188") + ggtitle("Histogram of
Total Charges")
ggplotly(plot4)
```

The above code creates a histogram of the TotalCharges variable. This allows us to visualize the distribution of TotalCharges in our dataframe.



By looking at the output, we can see that the distribution of TotalCharges is positively skewed or right-skewed. It means that the majority of the values are concentrated towards $0, and as the values increase, the frequencies gradually decrease. We have 998 customers that has TotalCharges $0. **The high frequency at $0 indicates that certain customers may not have started using Telco product for more than one month.** We will need to analyze the data further to confirm this hypothesis.

```
ZeroTC <- subset(df, TotalCharges == 0)
unique(ZeroTC$tenure)
```

The above code creates a subset of the dataframe called ZeroTC that contains only the rows where the TotalCharges value is 0. We then use the unique() function to display the unique values of tenure in this subset. This allows us to confirm our hypothesis that the high frequency of $0 TotalCharges is due to the fact that some customers have not been using the product for more than 1 month.

```
[1] 0
```

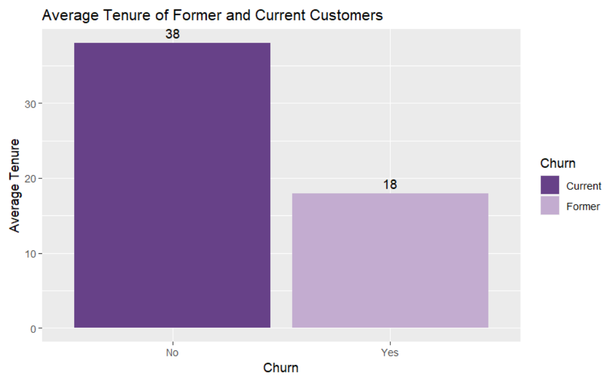
The unique() function shows us that the only unique value of tenure in the subset is 0. This means that all of the customers in the subset have been using the product for less than 1 month. This confirms our hypothesis that **the high frequency of $0 TotalCharges is due to the fact that some customers have not been using the product for very long.**

```
average_tenure <- aggregate(tenure ~ Churn, data = df, FUN = mean)
average_tenure$tenure <- round(average_tenure$tenure)

plot5 <- ggplot(average_tenure, aes(x = Churn, y = tenure, fill =
Churn)) +
  geom_bar(stat = "identity") +
  geom_text(aes(label = tenure), vjust = -0.5) +
  labs(x = "Churn", y = "Average Tenure", title = "Average Tenure of
Former and Current Customers") +
  scale_fill_manual(values = c("#674188", "#C3ACD0"), labels =
c("Current", "Former"))

print(plot5)
```

We use the above code to create and display a bar plot that answers the question, "What is the average number of months current and former customers stayed with the company?"



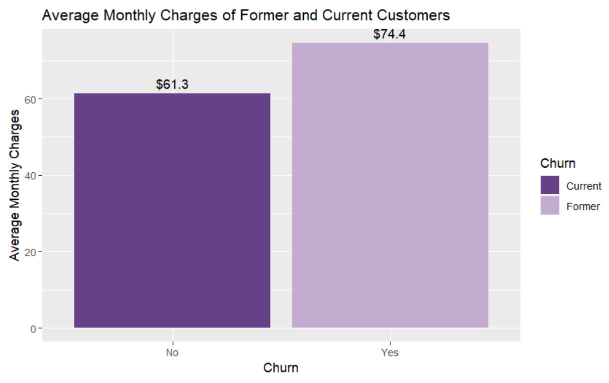
By looking at the output we can see that the average number of months current customers have stayed with the company is 38 and the average number of months former customers have stayed with the company is 18. It means that **mostly the customer that has leave the company, churn after 18 months using the product.**

```
average_MonthlyCharge <- aggregate(MonthlyCharges ~ Churn, data = df,
FUN = mean)

plot6 <- ggplot(average_MonthlyCharge, aes(x = Churn, y =
MonthlyCharges, fill = Churn)) +
  geom_bar(stat = "identity") +
  geom_text(aes(label = paste0('$', round(MonthlyCharges, 1))), vjust =
-0.5) +
  labs(x = "Churn", y = "Average Monthly Charges", title = "Average
Monthly Charges of Former and Current Customers") +
  scale_fill_manual(values = c("#674188", "#C3ACD0"), labels =
c("Current", "Former"))

print(plot6)
```

We use the above code to create and display a bar plot that answers the question, "What is the average monthly charges of former and current customers?"



The output of this code shows that the average monthly charges for former customers is $74.4 and the average monthly charges for current customers is $61.3. This means that former customers tend to have higher monthly charges than current customers. This could be because former customers are more likely to have more expensive plans or services.

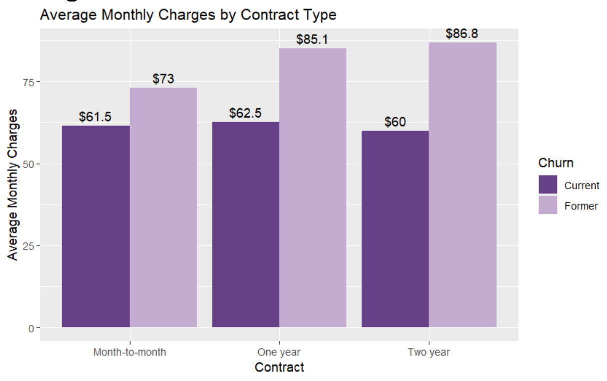
The fact that former customers tend to have higher monthly charges suggests that **high monthly charges may be a factor in customer churn.** This is because customers who find it difficult to pay their monthly charges may be more likely to cancel their service. The company could address this issue by offering more affordable plans or services.

```
average_charges <- aggregate(MonthlyCharges ~ Churn + Contract, data =
df, FUN = mean)

plot7 <- ggplot(average_charges, aes(x = Contract, y = MonthlyCharges,
fill = Churn)) +
  geom_bar(stat = "identity", position = "dodge") +
  geom_text(aes(label = paste0('$', round(MonthlyCharges, 1))), position
= position_dodge(width = 0.9), vjust = -0.5) +
  labs(x = "Contract", y = "Average Monthly Charges", title = "Average
Monthly Charges by Contract Type") +
  scale_fill_manual(values = c("#674188", "#C3ACD0"), labels =
c("Current", "Former"))

print(plot7)
```

We use the above code to create a bar plot that compares the average monthly charges based on the combination of contract type and churn status. This bar plot can help us understand the average monthly charges for different contract types and the impact of charges on churn.



Average monthly charges for current customers:
month-to-month contract = $61.5
one-year contract = $62.5
two-year contract = $60
Average monthly charges for former customers:
month-to-month contract = $73
one-year contract = $85.1
two-year contract = $86.8

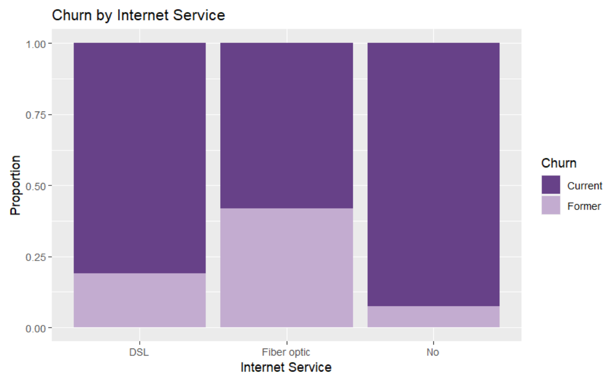
Based on the output above we can conclude that:

- The average monthly charges for current customers are relatively consistent across different contract types.

- Former customers who had longer contract durations tended to have higher average charges compared to those with shorter durations.
- There is a positive correlation between average monthly charges and churn rate.
- Charges may be a factor in customer churn.

```
plot8 <- ggplot(df, aes(x = InternetService, fill = Churn)) +
  geom_bar(position = "fill") +
  labs(x = "Internet Service", y = "Proportion", fill = "Churn") +
  ggtitle("Churn by Internet Service") + scale_fill_manual(values =
c("#674188", "#C3ACD0"), labels = c("Current", "Former"))
print(plot8)
```

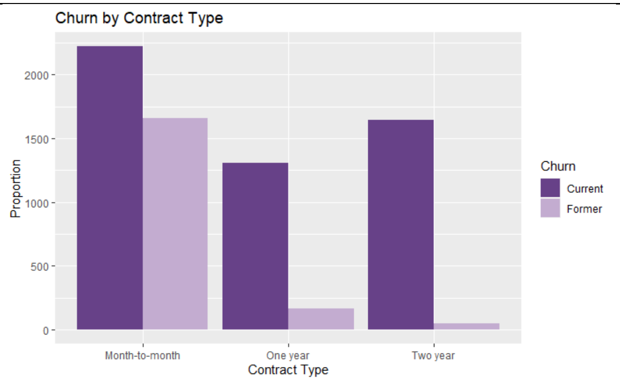
We use the code above to create and display a stacked bar plot that visualizes the proportion of current and former customers based on their internet service. This plot can help us understand the impact of internet service on churn, or in other words, identify if certain types of internet services are associated with higher or lower churn rates.



As we can see, the plot shows that most former customers chose fiber optic as their internet service. This means that **fiber optic is associated with a higher churn rate**. This could be because fiber optic is more expensive and more difficult to install and maintain than other internet services.

```
plot9 <- ggplot(df, aes(x = Contract, fill = Churn)) +
  geom_bar(position = "dodge") +
  labs(x = "Contract Type", y = "Proportion", fill = "Churn") +
  ggtitle("Churn by Contract Type") + scale_fill_manual(values =
c("#674188", "#C3ACD0"), labels = c("Current", "Former"))
print(plot9)
```

The code above used to create and display a grouped bar plot that visualizes the proportion of current and former customer based on the contract tyoe. By creating the plot we can gain information regarding the impact of contract type on churn.



The most popular contract type is month-to-month. However, most customers who churn also choose month-to-month contracts. This means that **customers who choose month-to-month contracts are more likely to churn** than customers who choose one-year or two-year contracts.

One possible explanation for this is that we tend to choose month-to-month contracts because we want to try the product first before committing to a longer contract. If we are not satisfied with the product, we can choose not to continue. To confirm whether our hypothesis is correct, we need to further analyse the data.

```
mtm <- subset(df, df$Contract == "Month-to-month" & df$Churn == "Yes" )
min(mtm$tenure)
```

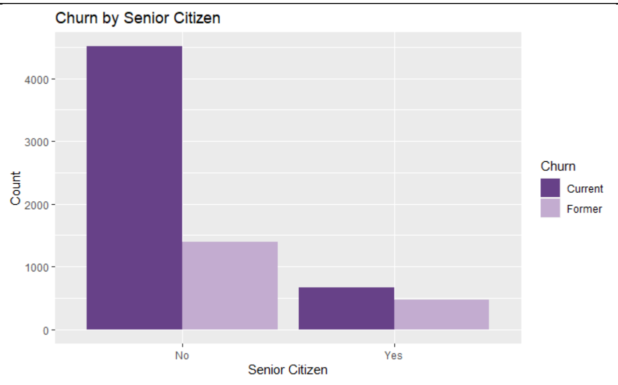
We use the code above to create a subset of the data called mtm that contains only the rows where the contract type is month-to-month and the churn status is yes. We then use the min() function to display the minimum value of tenure in this subset. This allows us to confirm our hypothesis that customers who churn are most likely to choose month-to-month contracts because they want to try the product first before committing to a longer contract.

```
[1] 1
```

The min() function shows us that the minimum value of tenure in the subset is 1 month. This means that **our hypothesis is correct. Customers who choose month-to-month contracts are more likely to churn because they only want to try the product for a short period of time.** If they are not satisfied with the product, they can cancel their subscription without penalty.

```
plot10 <- ggplot(df, aes(x = SeniorCitizen, fill = Churn)) +
  geom_bar(position = "dodge") +
  labs(x = "Senior Citizen", y = "Count", fill = "Churn") +
  ggtitle("Churn by Senior Citizen")+ scale_fill_manual(values =
c("#674188", "#C3ACD0"), labels = c("Current", "Former"))
print(plot10)
```

We use the code above to create a group bar plot that visualizes the total comparison between former and current customers based on their senior citizen status. This graph can help us understand whether there are more current customers who are senior citizens than there are current customers who are not senior citizens.



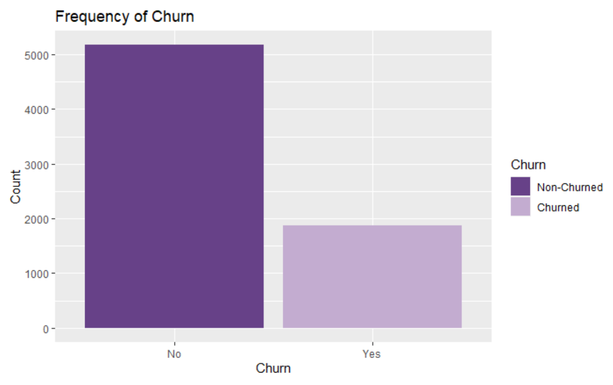
The output shows that there are more current customers who are not senior citizens than there are current customers who are senior citizens. This means that the majority of our customers are not senior citizens.

Step 9: checks if the data is imbalanced and handles it if it is.

```
plot12 <- ggplot(df, aes(x = Churn, fill = Churn)) +
  geom_bar(position = "dodge") +
  labs(x = "Churn", y = "Count", fill = "Churn") +
  ggtitle("Frequency of Churn") +
  scale_fill_manual(values = c("#674188", "#C3ACD0"), labels = c("Non-
Churned", "Churned"))

print(plot12)
```

The code above creates a bar plot that shows the frequency of each value in the Churn variable. This graph can help us understand whether our dataset is imbalanced or not. A dataset is called imbalanced if there are more instances of one class than another.



As we can see, the plot shows that there are more customers who doesn't churn than the customers who churn. This means that our dataset is imbalanced, and we have to handle this problem if we want to create a good predictive model.

```
table(df$Churn)

df_balance <- ovun.sample(Churn ~ ., data = df, method = "over",N =
10348)$data
```

An imbalance dataset can be a problem for us because the predictive model can be biased towards the majority class.

We use the code above to show us the number of each value in churn variable then we oversample the minority class. In our case the minority is customer who churn since the number of customers who churn is more smaller than the number of customer who doesn't churn.

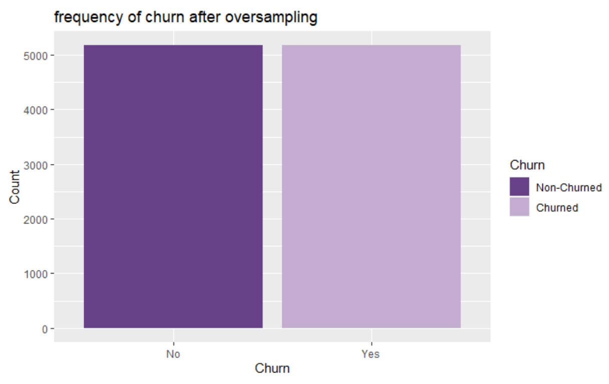
```
  No  Yes
5174 1869
```

By using oversample, we can add more data to our dataframe where the customer churn status is "Yes" so that the number of customer who does and doesn't churn is equal. 10348 means that we will oversample the minority class till the sum of each class is equal to 10348. We choose 10348 because the number of majority class is 5174 so we have to add more minority class till the sum of each class become (5174x2) 10348 and the number of each value in churn variable become equals.

```
plot13 <- ggplot(df_balance, aes(x = Churn, fill = Churn)) +
  geom_bar(position = "dodge") +
  labs(x = "Churn", y = "Count", fill = "Churn") +
  ggtitle("frequency of churn after oversampling") +
  scale_fill_manual(values = c("#674188", "#C3ACD0"), labels = c("Non-
Churned", "Churned"))

print(plot13)
```

The code above creates a bar plot that shows the frequency of each value in the Churn variable. This graph can help us understand whether our dataset is imbalanced or not. A dataset is called imbalanced if there are more instances of one class than another.



As we can see, the plot shows **that our dataset is no longer imbalanced** because the number of customers who churn and customer who doesn't churn is already equal.

C. PREDICTIVE MODEL

Once we have cleaned up our data and removed any anomalies, we can start building a model to predict customer churn. We will **use a random forest model to predict categorical outcomes**. The random forest model will be trained on our data, and it will learn to identify the features that are most predictive of customer churn. Once the model is trained, we can use it to predict whether a new customer is likely to churn. This information can be used to take steps to prevent customers from leaving our company.

We choose random forest as the algorithm for our model because it is well suited for predicting categorical outcomes and can help us reduce the variance of the model since it made up of multiple decision trees.

Step 1: Split the dataset

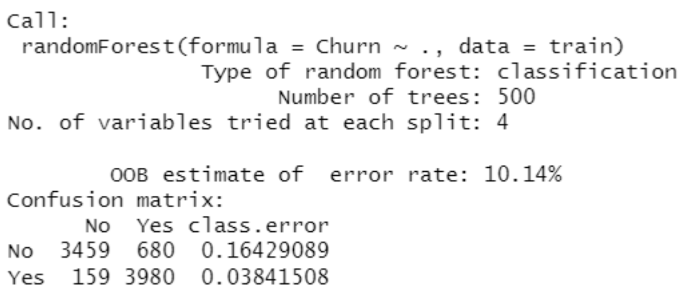
```
set.seed(1)
sample <- sample.split(df_balance$Churn, SplitRatio = 0.8)
train = subset(df_balance, sample == TRUE)
test = subset(df_balance, sample == FALSE)
```

The code above is **used to split our data into two sets - a training set and a testing set, with 80% of the data allocated for training and 20% for testing**. Setting a seed in the code ensures that the same random split will be used whenever the code is run in the future, which helps with reproducibility.

Step 2: Create the predictive model

```
library(randomForest)
model <- randomForest(Churn ~ .,   data=train)
print(model)
```

We use the code above to builds a random forest model. The formula Churn ~ . to specify that the variable Churn is the target variable to be predicted, and the "." indicates that all other variables in the train dataframe are used as predictors. After that we use print(model) to prints the details of the trained random forest model. It displays information such as the number of trees in the forest and the number of input variables.



```
Call:
 randomForest(formula = Churn ~ ., data = train)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 4

        OOB estimate of  error rate: 10.14%
Confusion matrix:
      No  Yes class.error
No  3459  680  0.16429089
Yes  159 3980  0.03841508
```

The output reveals that the model is a classification random forest that consists of 500 decision trees, which is the default value. At each split, the model considers 4 randomly selected variables as potential predictors. The OOB estimate the error rate of the model, lower indicates that the model has a better perfomance. This output also shows the confusion matrix.

The model predicted 3459 instances correctly as "No" and 680 instances incorrectly as "Yes". The **class error rate for "No" is 16.43%.** The model also predicted 3459 instances correctly as "Yes" and 159 instances incorrectly as "No". **The class error rate for "Yes" is 3.84%.**

Step 3: Predict the test set

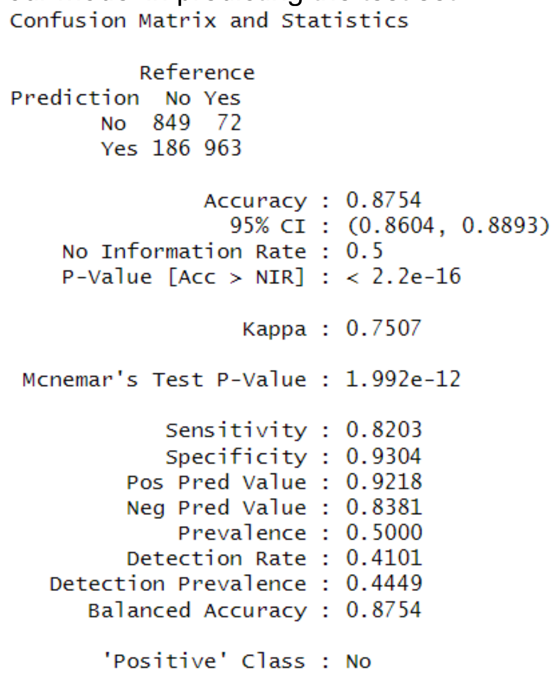
```
predictions <- predict(model, newdata = test)
```

We use the code above to predict the class of churn variable for the test set.

Step 4: evaluating the model performance

```
eval <- confusionMatrix(data=predictions, reference = test$Churn)
eval
```

The code above is used to create a confusion matrix, which allows us to evaluate the performance of our model in predicting the test set.



```
Confusion Matrix and Statistics

          Reference
Prediction  No Yes
       No  849  72
       Yes 186 963
                                          
               Accuracy : 0.8754          
                 95% CI : (0.8604, 0.8893)
    No Information Rate : 0.5             
    P-Value [Acc > NIR] : < 2.2e-16       
                                          
                  Kappa : 0.7507          
                                          
 Mcnemar's Test P-Value : 1.992e-12       
                                          
            Sensitivity : 0.8203          
            Specificity : 0.9304          
         Pos Pred Value : 0.9218          
         Neg Pred Value : 0.8381          
             Prevalence : 0.5000          
         Detection Rate : 0.4101          
   Detection Prevalence : 0.4449          
      Balanced Accuracy : 0.8754          
                                          
       'Positive' Class : No              
```

Based on the output we know that:

1. The model correctly predicted "No" for 849 data points that were actually "No." It also correctly predicted "Yes" for 963 data points that were actually "Yes." However, the model incorrectly predicted "No" for 72 data points that were actually "Yes," and it incorrectly predicted "Yes" for 186 data points that were actually "No."
2. The accuracy of the model is 0.8754, indicating that the model correctly predicts the churn status for 87.54% of the instances in the test set.
3. 95% Confidence Interval (CI) is 0.8604, 0.8893 this interval gives a range within which the accuracy is likely to fall.
4. NIR or No Information Rate 0.5 means that the model performs significantly better than simply predicting the majority class.
5. Kappa value 0.7507 means there is an agreement that the prediction are not simply due to chance.
6. sensitivity (Recall) measures the proportion of actual positive instances that are correctly classified as positive while specificity measures the proportion of actual negative instances that are correctly classified as negative. Value 0.8203 for sensitivity and 0.9304 for specificity means that our model good enough at predicting the churn rate.
7. PPV or precision indicates the proportion of predicted positive instances that are actually positive while NPV represents the proportion of predicted negative instances that are truly negative. Value of 0.9218 for PPV and 0.8381 for NPV means that our model good enough at predicting the churn rate.
8. The prevalence of 0.5 indicates that half of the instances in the test set are positive. The detection rate of 0.4101 indicates that the model correctly

identified 41.01% of the positive instances. This information tells us that the model is not very good at identifying positive instances.

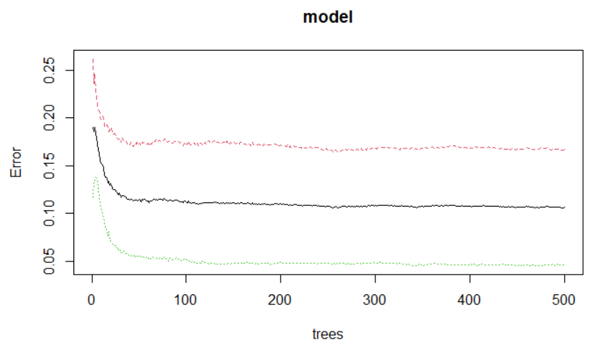
9. The balanced accuracy of 0.8754 tells us that the model is performing well overall. It can correctly identify both positive and negative cases with a high degree of accuracy.

Conclusion:
In a simple word **our model is performing well and can correctly predicts the churn status for 87.54% of the instances in the test set.**

Step 5: visualize the change of error rate

```
plot(model)
```

We use the code above to display a plot that can visualize the change of error as the number of trees increase.



The error rate is a measure of how often the model makes a mistake. A low error rate means that the model is accurate, while a high error rate means that the model is inaccurate. By looking at the output, we can see that the error rate become more stable as the number of trees increases. This means that **the model becomes more accurate to predict the churn status as the number of trees increases**.

Step 6: display the variable importance in our model

```
importance(model)
```

The code above is used to calculate and display the variable importance in our random forest model. By examining the variable importance, we can gain insights regarding which predictors (independent variable) have the most influence on the model to predict the the target (Churn).

```
                 MeanDecreaseGini
customerID              368.62209
gender                   64.53378
SeniorCitizen            47.71107
Partner                  55.96877
Dependents               51.23247
tenure                  479.50192
PhoneService             14.30378
MultipleLines            64.17278
InternetService         132.16843
OnlineSecurity          188.44549
OnlineBackup             78.77382
DeviceProtection         64.32064
TechSupport             134.04500
StreamingTV              59.95266
StreamingMovies          58.87584
Contract                386.10364
PaperlessBilling         63.24944
PaymentMethod           178.37152
MonthlyCharges          417.82632
TotalCharges            484.56409
```

Mean Decrease Gini is a measure of how much each variable contributes to the model's ability to predict customer churn. This means that Variables with higher MeanDecreaseGini values are generally more important in predicting the target variable (Churn).

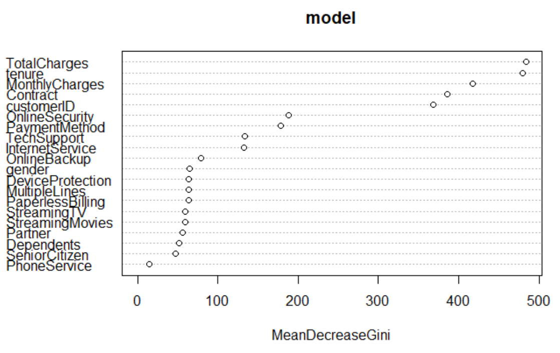
Based on the output we can see that **TotalCharges, has the highest MeanDecreaseGini with a value of 484.56409 indicating that TotalCharges is a crucial predictor if we want to predict churn.** This makes sense, because high TotalCharges can make it difficult to pay. Imagine if the customer suddenly loses their job or having financial difficulties, they may find it difficult to continue paying their bill so they choose to leave the product or switching to the product from our competitor with more lower prices.

Other than TotalCharges, tenure, Contract, OnlineSecurity, TechSupport, InternetService, and MonthlyCharges alsho show high MeanDecreaseGini indicating that **those variable lays a significant role as predictors to predict churn**.

Step 7: plot the meanDecreaseGini for each predictors

```
varImpPlot(model)
```

The code above is used to simply plot the meanDecreaseGini for each predictors. By creating this plot we can easier find the variable with the greatest meanDecreaseGini.



Based on the output we can see that **the top 5 predictors to predict churn are TotalCharges, tenure, MonthlyCharges, contract, and customerID**. On the other hand, **Phone services has the lowest meanDecreaseGini.** This means that phone services is not a very important predictor of churn. This may be because phone services are a relatively standard feature that most customers have.

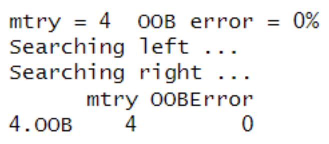
Based on this information I would like to tune the model and exclude the PhoneService predictor from the model. By excluding this variable, I hope we can simplify the model and potentially improve its interpretability. Additionally, removing less important predictors can reduce the risk of overfitting and improve the model's generalization performance.

Step 8: use the tuneRF () function for finding optimal parameter

```
tuneParam <- tuneRF(train,train$Churn,stepFactor = 1.2, improve = 0.01,
trace=T, plot= T)
tuneParam
```

we use the code above to find the best parameter of the random forest model using the tuneRF() function. The function performs an iterative search for the optimal value of the mtry parameter, which controls the number of randomly selected predictor variables used at each split in the random forest. It evaluates the model's out-of-bag (OOB) error rate for different values of mtry and selects the value that minimizes the error rate.

By executing this code we can find the optimal value of the "mtyr" parameter. The "mtry" parameter in Random Forest refers to the number of randomly selected predictors considered at each split when constructing each tree in the forest.



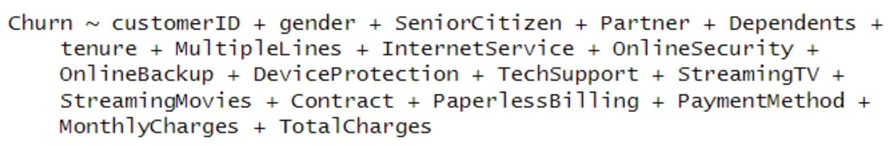
```
mtry = 4  OOB error = 0%
Searching left ...
Searching right ...
      mtry OOBError
4.OOB    4        0
```

The output showed that the model achieved a perfect OOB error of 0% when "mtry" was set to 4. This means that the model was able to accurately predict the target variable for all of the out-of-bag samples. Out-of-bag samples are data points that were not used to train the model.

Step 9: create a new formula that exclude PhoneService Variable

```
n <- names(train)
excluded_vars <- c("Churn", "PhoneService")
included_vars <- n[!(n %in% excluded_vars)]
f <- as.formula(paste("Churn ~", paste(included_vars, collapse = " +
")))
f
```

The code above is used to create a new formula for the model. We don't want to include Churn or PhoneService in the model because PhoneService is the least important variable and Churn is the variable we are trying to predict. We create the formula so we don't have to type all the variables in one by one.



```
Churn ~ customerID + gender + SeniorCitizen + Partner + Dependents +
    tenure + MultipleLines + InternetService + OnlineSecurity +
    OnlineBackup + DeviceProtection + TechSupport + StreamingTV +
    StreamingMovies + Contract + PaperlessBilling + PaymentMethod +
    MonthlyCharges + TotalCharges
```

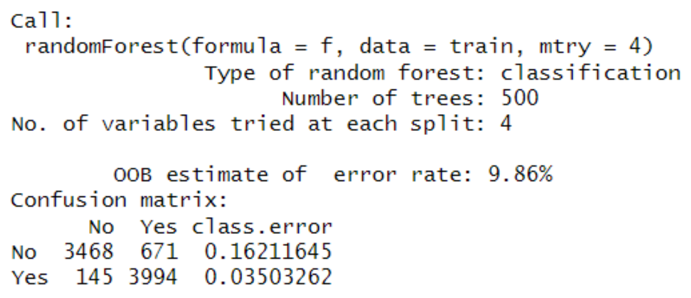
The tilde symbol (~) separates the target variable from the predictor variables. The target variable is the variable that we are trying to predict, and the predictor variables are the variables that we are using to make the prediction. In this case, we are trying to predict whether a customer will churn, and we are using the predictor variables on the right side of the tilde symbol to make the prediction.

Step 10: create a new prediction model using random forest with the optimal parameter that we get from the tuneRF() function.

```
modelTuned <- randomForest(f, data = train, mtry = 4)
print(modelTuned)
```

We use the code above to builds a tuned random forest model. Unlike the previous model, this time we use the **f** formula that we have defined before. In this formula we will predict churn using all the variables we have as the predictors except the Churn and PhoneService variables. In addition, we also define the value of the variable "mtyr" as 4.

After that we use print(model) to prints the details of the trained random forest model. It displays information such as the number of trees in the forest and the number of input variables.



```
Call:
 randomForest(formula = f, data = train, mtry = 4) 
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 4

        OOB estimate of  error rate: 9.86%
Confusion matrix:
      No  Yes class.error
No  3468  671  0.16211645
Yes  145 3994  0.03503262
```

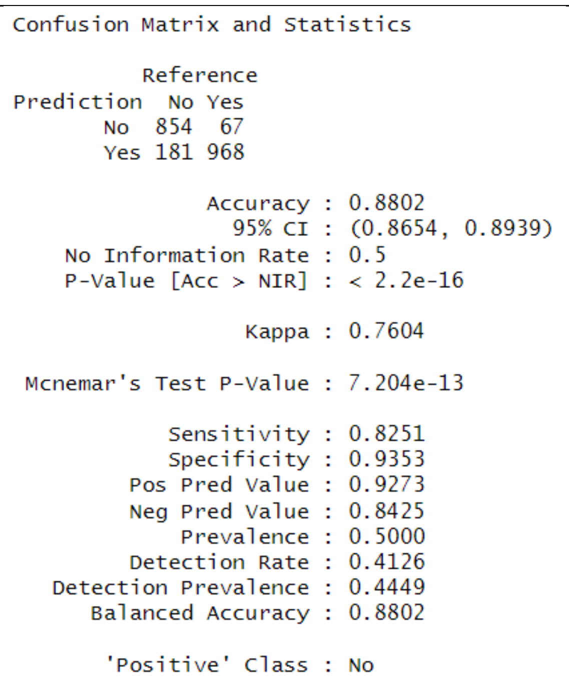
The output reveals that the model is a classification random forest that consists of 500 decision trees, which is the default value. At each split, the model considers 4 randomly selected variables as potential predictors. The OOB estimate the error rate of the model, lower indicates that the model has a better perfomance. This output shows that the OOB of the tuned parameter which is 9.86% is smaller than the previous model that has OOB value of 10.14%

The model predicted 3468 instances correctly as "No" and 671 instances incorrectly as "Yes". The **class error rate for "No" is 16.21%.** The model also predicted 3994 instances correctly as "Yes" and 145 instances incorrectly as "No". **The class error rate for "Yes" is 3.5%.**

Step 11: use modelTuned to predict churn on the test set and display the confusion matrix

```
TunedPredictions <- predict(modelTuned, newdata = test)
evalTuned <- confusionMatrix(data=TunedPredictions, reference =
test$Churn)
evalTuned
```

We use the code above to predict the class of churn variable for the test set using the tuned model and create a confusion matrix, which allows us to evaluate the performance of our model in predicting the test set.



```
Confusion Matrix and Statistics

          Reference
Prediction  No Yes
       No  854  67
       Yes 181 968

               Accuracy : 0.8802
                 95% CI : (0.8654, 0.8939)
    No Information Rate : 0.5
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.7604

 Mcnemar's Test P-Value : 7.204e-13

            Sensitivity : 0.8251
            Specificity : 0.9353
         Pos Pred Value : 0.9273
         Neg Pred Value : 0.8425
             Prevalence : 0.5000
         Detection Rate : 0.4126
   Detection Prevalence : 0.4449
      Balanced Accuracy : 0.8802

       'Positive' Class : No
```

Based on the output we know that:

1. The model correctly predicted "No" for 854 data points that were actually "No." It also correctly predicted "Yes" for 968 data points that were actually "Yes." However, the model incorrectly predicted "No" for 67 data points that were actually "Yes," and it incorrectly predicted "Yes" for 181 data points that were actually "No."
2. The accuracy of the model is 0.8802, indicating that the model correctly predicts the churn status for 88.02% of the instances in the test set.
3. 95% Confidence Interval (CI) is 0.8654, 0.8939 this interval gives a range within which the accuracy is likely to fall.
4. NIR or No Information Rate 0.5 means that the model performs significantly better than simply predicting the majority class.
5. Kappa value 0.7604 means there is an agreement that the prediction are not simply due to chance.
6. sensitivity (Recall) measures the proportion of actual positive instances that are correctly classified as positive while specificity measures the proportion of actual negative instances that are correctly classified as negative. Value 0.8251 for sensitivity and 0.9353 for specificity means that our model good enough at predicting the churn rate.
7. PPV or precision indicates the proportion of predicted positive instances that are actually positive while NPV represents the proportion of predicted negative instances that are truly negative. Value of 0.9273 for PPV and 0.8425 for NPV means that our model good enough at predicting the churn rate.
8. The prevalence of 0.5 indicates that half of the instances in the test set are positive. The detection rate of 0.4126 indicates that the model correctly identified 41.26% of the positive instances. This information tells us that the model is not very good at identifying positive instances.

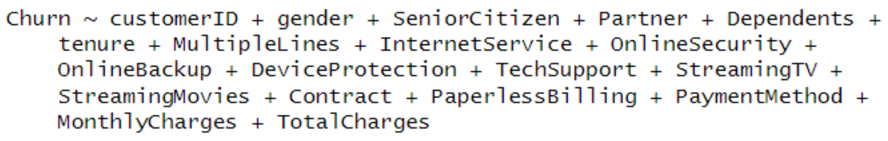
9. The balanced accuracy of 0.8802 tells us that the model is performing well overall. It can correctly identify both positive and negative cases with a high degree of accuracy.

Conclusion:
In a simple word **our model is performing well and can correctly predicts the churn status for 88.02% of the instances in the test set.**

Step 12: display the variable importance in our model

```
n <- names(train)
excluded_vars <- c("Churn", "PhoneService")
included_vars <- n[!(n %in% excluded_vars)]
f <- as.formula(paste("Churn ~", paste(included_vars, collapse = " + 
")))
f
```

The code above is used to create a new formula for the model. We don't want to include Churn or PhoneService in the model because PhoneService is the least important variable and Churn is the variable we are trying to predict. We create the formula so we don't have to type all the variables in one by one.



```
Churn ~ customerID + gender + SeniorCitizen + Partner + Dependents + 
    tenure + MultipleLines + InternetService + OnlineSecurity + 
    OnlineBackup + DeviceProtection + TechSupport + StreamingTV + 
    StreamingMovies + Contract + PaperlessBilling + PaymentMethod + 
    MonthlyCharges + TotalCharges
```

The tilde symbol (~) separates the target variable from the predictor variables. The target variable is the variable that we are trying to predict, and the predictor variables are the variables that we are using to make the prediction. In this case, we are trying to predict whether a customer will churn, and we are using the predictor variables on the right side of the tilde symbol to make the prediction.

D. CONCLUSION

We have created a predictive model using random forest to predict churn with an accuracy of 87.54%. The top 5 predictors to predict churn with this model are TotalCharges, tenure, MonthlyCharges, contract, and customerID. On the other hand, Phone services has the lowest meanDecreaseGini.

The TotalCharges being the most important predictor is make sense because high TotalCharges can make it difficult to pay. Imagine if the customer suddenly loses their job or having financial difficulties, they may find it difficult to continue paying their bill so they choose to leave the product or switching to the product from our competitor with more lower prices.

Based on the information we get, we then created a tuned model that excludes PhoneService because it has the smallest meanDecreaseGini. The tuned model turned out more accurate than the first model, with an accuracy of 88.02%.

E. GITHUB

I created a repository on github for this project and it can be accessed using this link
Github Repo