Credit Scoring

Loading Data

```
rm(list = ls())
Credit_Train <- read.csv("CreditTraining.csv")
Credit_Test <- read.csv("CreditTesting.csv", sep=";")

glimpse(Credit_Train)
```

```
## Observations: 5,380
## Variables: 19
## $ Id_Customer         <int> 7440, 573, 9194, 3016, 6524, 3858, 2189, 9...
## $ Y                   <int> 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, ...
## $ Customer_Type       <fct> Non Existing Client, Existing Client, Non ...
## $ BirthDate           <fct> 07/08/1977, 13/06/1974, 07/11/1973, 08/07/...
## $ Customer_Open_Date  <fct> 13/02/2012, 04/02/2009, 03/04/2012, 25/08/...
## $ P_Client            <fct> NP_Client, P_Client, NP_Client, NP_Client,...
## $ Educational_Level   <fct> University, University, University, Univer...
## $ Marital_Status      <fct> Married, Married, Married, Married, Marrie...
## $ Number_Of_Dependant <int> 3, 0, 2, 3, 2, 0, 0, 0, 0, 4, 0, 0, 0, 0, ...
## $ Years_At_Residence  <int> 1, 12, 10, 3, 1, 28, 10, 15, 0, 35, 10, 10...
## $ Net_Annual_Income   <fct> "36", "18", "36", "36", "36", "60", "36", ...
## $ Years_At_Business   <int> 1, 2, 1, 1, 1, 2, 1, 1, 3, 2, 3, 2, 4, 1, ...
## $ Prod_Sub_Category   <fct> C, C, C, C, C, C, C, C, P, C, C, C, C, C, ...
## $ Prod_Decision_Date  <fct> 14/02/2012, 30/06/2011, 04/04/2012, 07/09/...
## $ Source              <fct> Sales, Sales, Sales, Sales, Sales, Sales, ...
## $ Type_Of_Residence   <fct> Owned, Parents, Owned, New rent, Owned, Ol...
## $ Nb_Of_Products      <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, ...
## $ Prod_Closed_Date    <fct> , , , 31/12/2012, , , , 16/04/2013, , 18/1...
## $ Prod_Category       <fct> B, G, B, L, D, C, B, B, E, L, B, B, G, B, ...
```

```
Credit_Train$is_train = 1
Credit_Test$is_train = 0
df <- rbind(Credit_Train, Credit_Test )
```

The dataset *df* contains both the train (5380 customers) and test (1345 customers) datasets concatenated. We included a variable is_train in order to be able to recover both datasets at the end in the modeling part. Here we are in a classical supervised classification problem, where we will try to predict Y, which takes 0 if the credit is issued, and 1 otherwise. For this we will use variables related to the client and the product that he is purchasing.

Variables cleaning

Fixing types

We start by converting Y to a factor, fixing the dates types, and converting Net_Annual_Income to numerical.

```r
# Convert columns to factor
df$Id_Customer <- factor(df$Id_Customer)
df$Y <- factor(df$Y, levels = c(1,0))

# Convert columns to date
df$BirthDate <- as.Date(as.character(df$BirthDate),
                                 format = "%d/%m/%Y",
                                 origin="1970-01-01")

df$Customer_Open_Date <- as.Date(as.character(df$Customer_Open_Date),
                                          format = "%d/%m/%Y",
                                          origin="1970-01-01")

df$Prod_Decision_Date <- as.Date(as.character(df$Prod_Decision_Date),
                                                format = "%d/%m/%Y",
                                                origin="1970-01-01")

df$Prod_Closed_Date <- as.Date(as.character(df$Prod_Closed_Date),
                               format = "%d/%m/%Y",
                               origin="1970-01-01")

# Convert Net_Annual_Income to numeric
df$Net_Annual_Income <- as.numeric(sub(",","", df$Net_Annual_Income))
```

Dealing with dates

Here we simply transform the BirthDate into a variable Age in order to have a useful numerical variable. There are 3 more variables containing successive dates: *Customer_Open_Date*, which is the first date when the client requested the product, *Prod_Decision_Date*, which is the date when the bank took the decision to grant him the credit or not, and finally *Prod_Closed_Date*, which corresponds to the date the bank closes the product (it is not offered anymore). We will compute the difference between $Customer\_Open\_Date - Prod\_Decision\_Date$, because a client which is likely to be eligible for a credit may receive a decision quicker for instance. By having a glimpse of *df*, we can see that *Prod_Closed_Date* seems to contain a lot of missing values, so we won't do anything with it.

```r
# New variables using dates
df$Age <- age_calc(df$BirthDate, units = "years")
df$Opening_to_Decision <- as.numeric(df$Prod_Decision_Date - df$Customer_Open_Date)

# Dropping useless ones
df$BirthDate = NULL
df$Customer_Open_Date = NULL
df$Prod_Decision_Date = NULL
df$Prod_Closed_Date = NULL
```

Dealing with factor labels

This part will be useful in the encoding and modelisations steps later on, because the modalities of these qualitative variables will become columns and so they will be correctly named.

```r
df$Customer_Type <- factor(df$Customer_Type,
                           levels = c("Non Existing Client", "Existing Client"),
                           labels = c("Non_Existing_Client", "Existing_Client"))

df$Educational_Level <- factor(df$Educational_Level,
                               levels = c("Secondary or Less", "Diploma",
                                          "University", "Master/PhD"),
                               labels = c("Secondary_or_Less", "Diploma",
                                          "University", "Master_PhD"))

df$Type_Of_Residence <- factor(df$Type_Of_Residence,
                               levels = c("Owned", "Parents", "New rent",
                                          "Old rent", "Company"),
                               labels = c("Owned", "Parents", "New_rent",
                                          "Old_rent", "Company"))
```

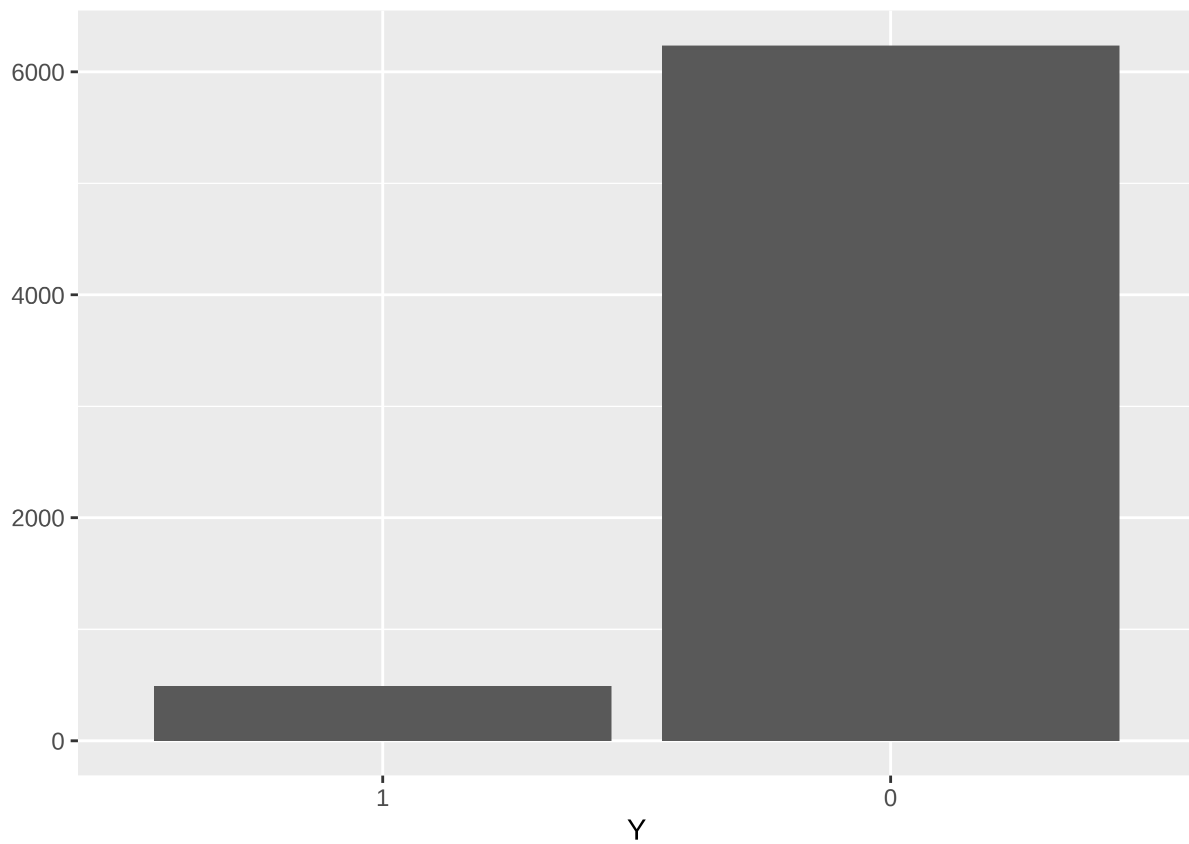
Summary Statistics

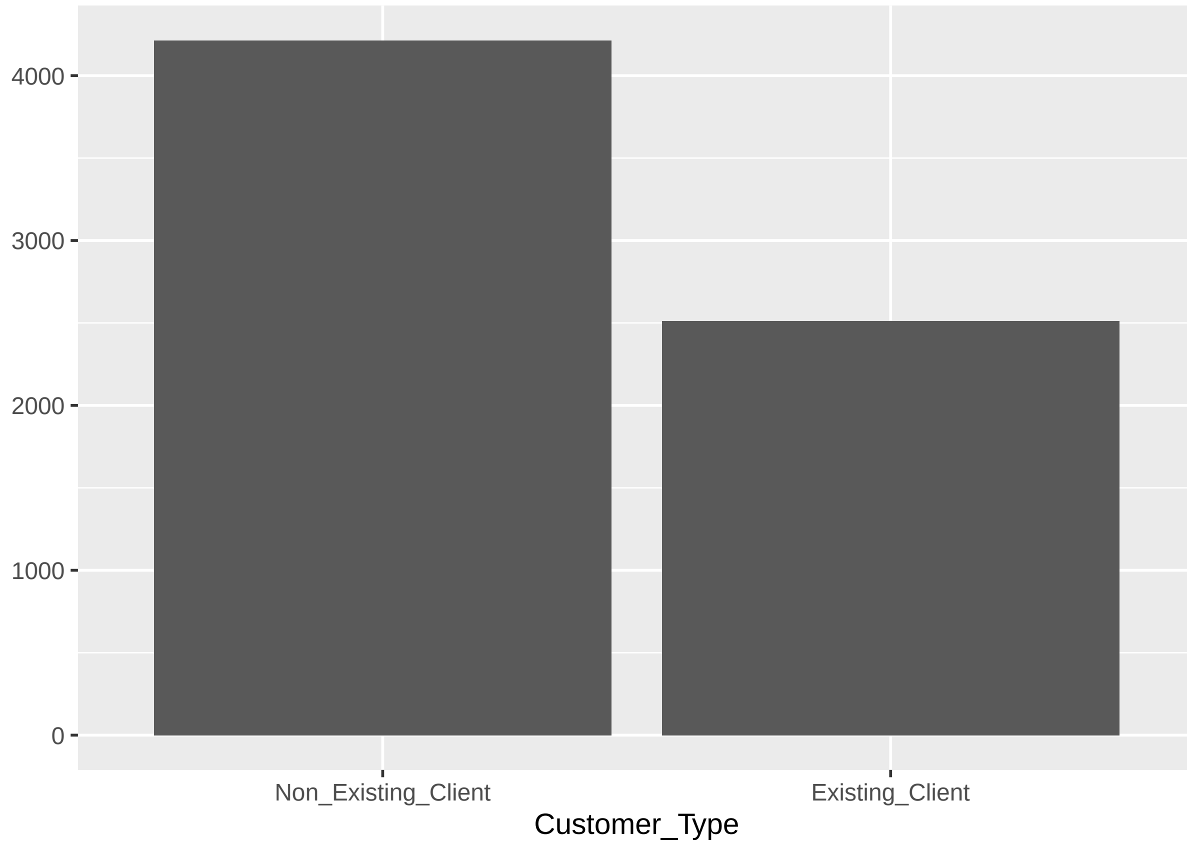
```r
# Barplot for every numerical variable
varnames = names(df)
varnames = varnames[varnames != "Id_Customer"]
varnames = varnames[varnames != "is_train"]

summary(select(df, varnames))
```

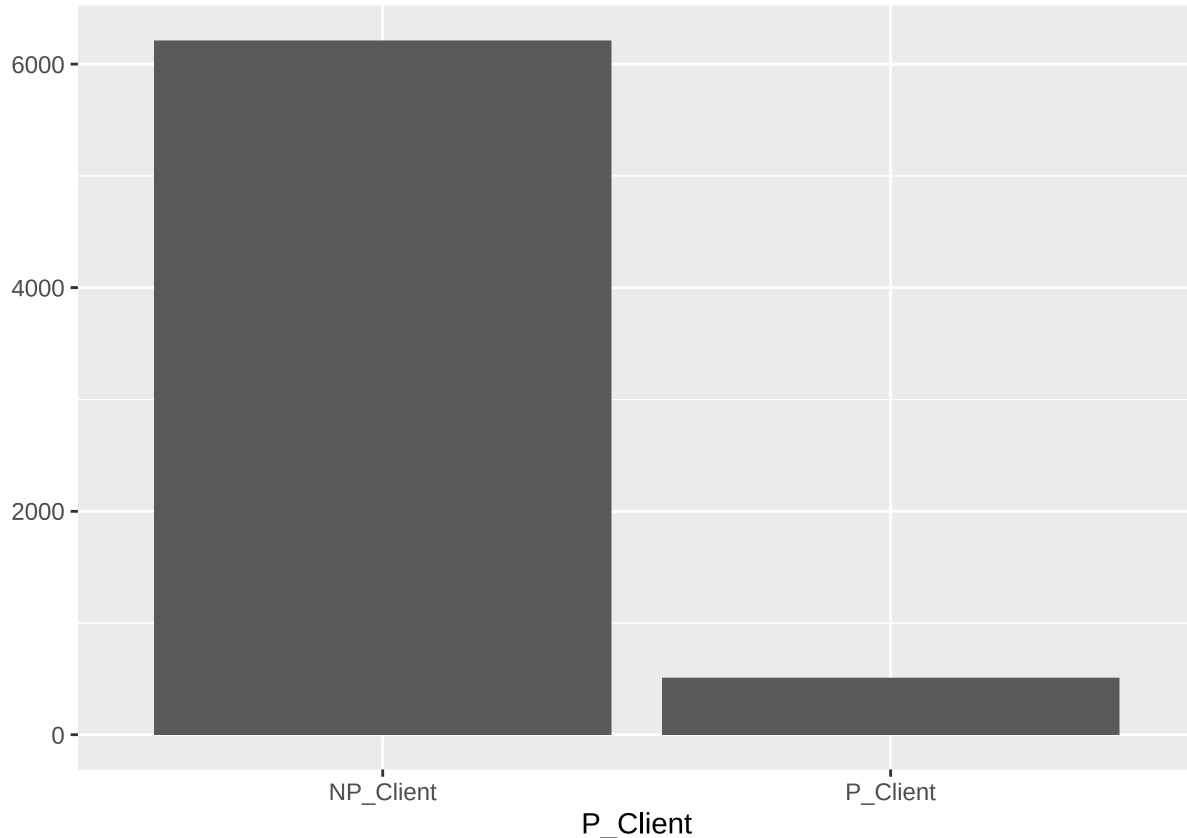
```
##  Y                    Customer_Type       P_Client
##  1: 490   Non_Existing_Client:4214   NP_Client:6213
##  0:6235   Existing_Client    :2511   P_Client : 512
##
##
##
##
##
##          Educational_Level   Marital_Status Number_Of_Dependant
##  Secondary_or_Less:  26    Divorced :  78   Min.   : 0.000
##  Diploma          :  75    Married  :5268   1st Qu.: 0.000
##  University       :5981    Separated:   2   Median : 0.000
##  Master_PhD       : 643    Single   :1293   Mean   : 1.052
##                            Widowed  :  84   3rd Qu.: 2.000
##                                             Max.   :20.000
##                                             NA's   :2
##  Years_At_Residence Net_Annual_Income Years_At_Business Prod_Sub_Category
##  Min.   : 0.00      Min.   :     1    Min.   : 0.000    C:5783
##  1st Qu.: 4.00      1st Qu.:    21    1st Qu.: 1.000    G: 805
##  Median :10.00      Median :    36    Median : 1.000    P: 137
##  Mean   :12.56      Mean   :  2729    Mean   : 4.266
##  3rd Qu.:17.00      3rd Qu.:    50    3rd Qu.: 4.000
##  Max.   :73.00      Max.   :717792    Max.   :98.000
##                     NA's   :3         NA's   :4
##     Source     Type_Of_Residence Nb_Of_Products  Prod_Category
##  Branch:1576   Owned   :5986     Min.   :1.000   B      :3979
```

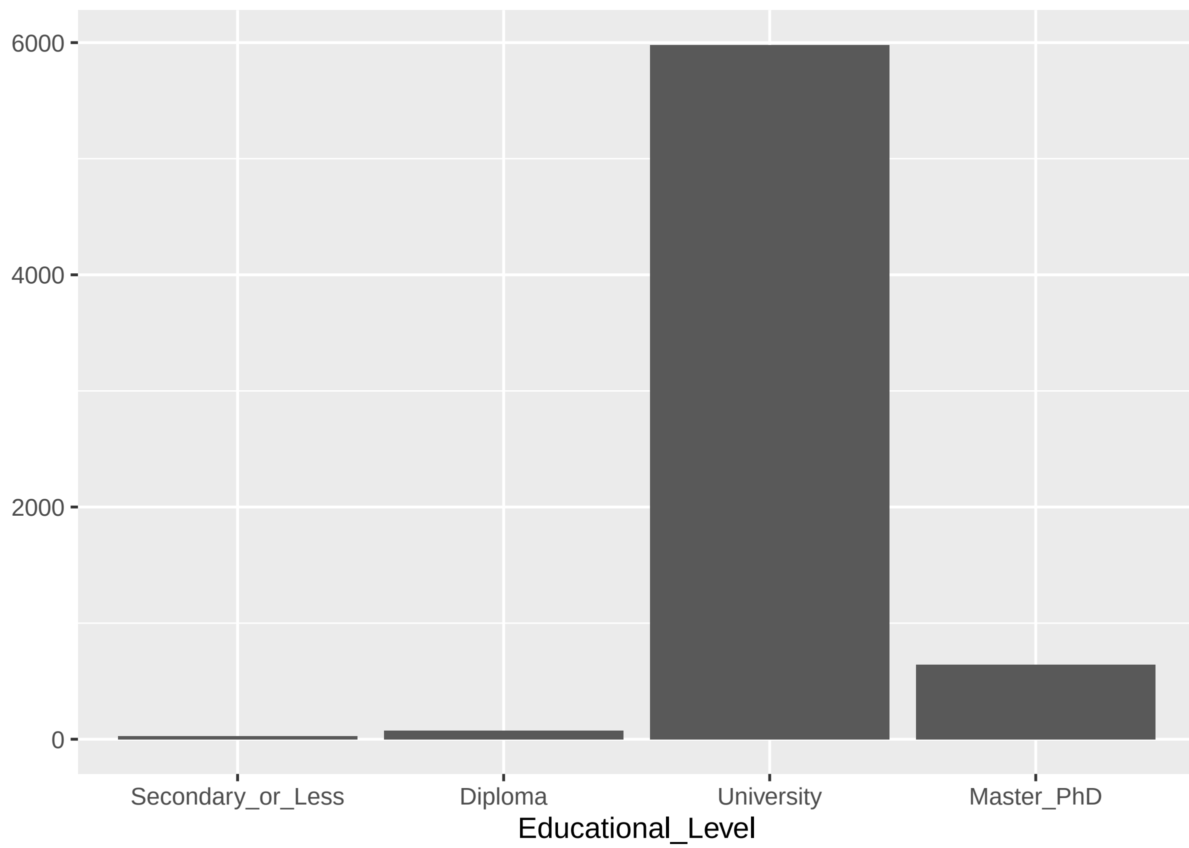
```
##  Sales :5149   Parents : 230     1st Qu.:1.000   D       : 835
##                New_rent: 105     Median :1.000   C       : 665
##                Old_rent: 399     Mean   :1.087   K       : 323
##                Company :   5     3rd Qu.:1.000   L       : 291
##                                  Max.   :3.000   G       : 250
##                                                  (Other): 382
##       Age        Opening_to_Decision
##  Min.   :28.81   Min.   :    0.0
##  1st Qu.:38.46   1st Qu.:    2.0
##  Median :46.23   Median :    7.0
##  Mean   :47.73   Mean   :  449.1
##  3rd Qu.:56.21   3rd Qu.:  295.0
##  Max.   :82.04   Max.   :11488.0
##
```

```
for (varname in varnames) {
  print(qplot(data = df, get(varname), xlab = varname))
}
```

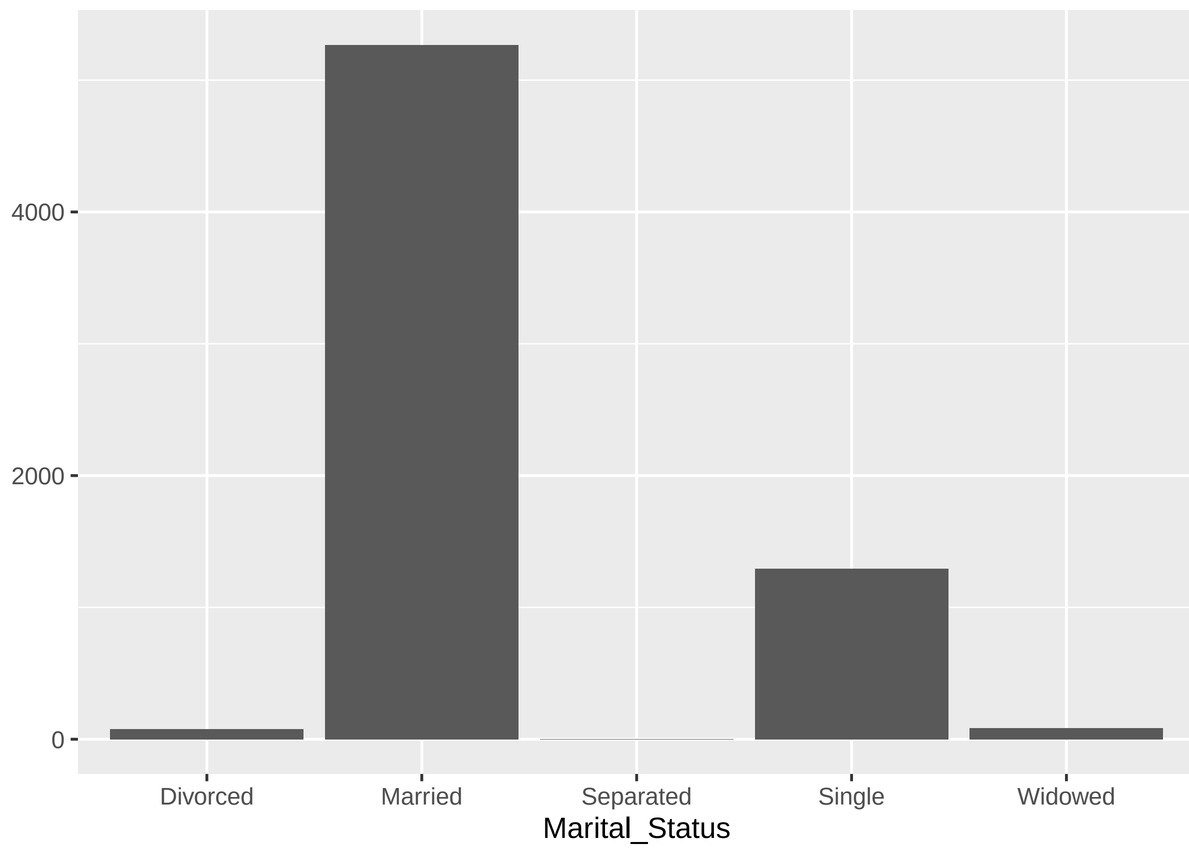



4000
3000
2000
1000
0
Non_Existing_Client
Existing_Client
Customer_Type

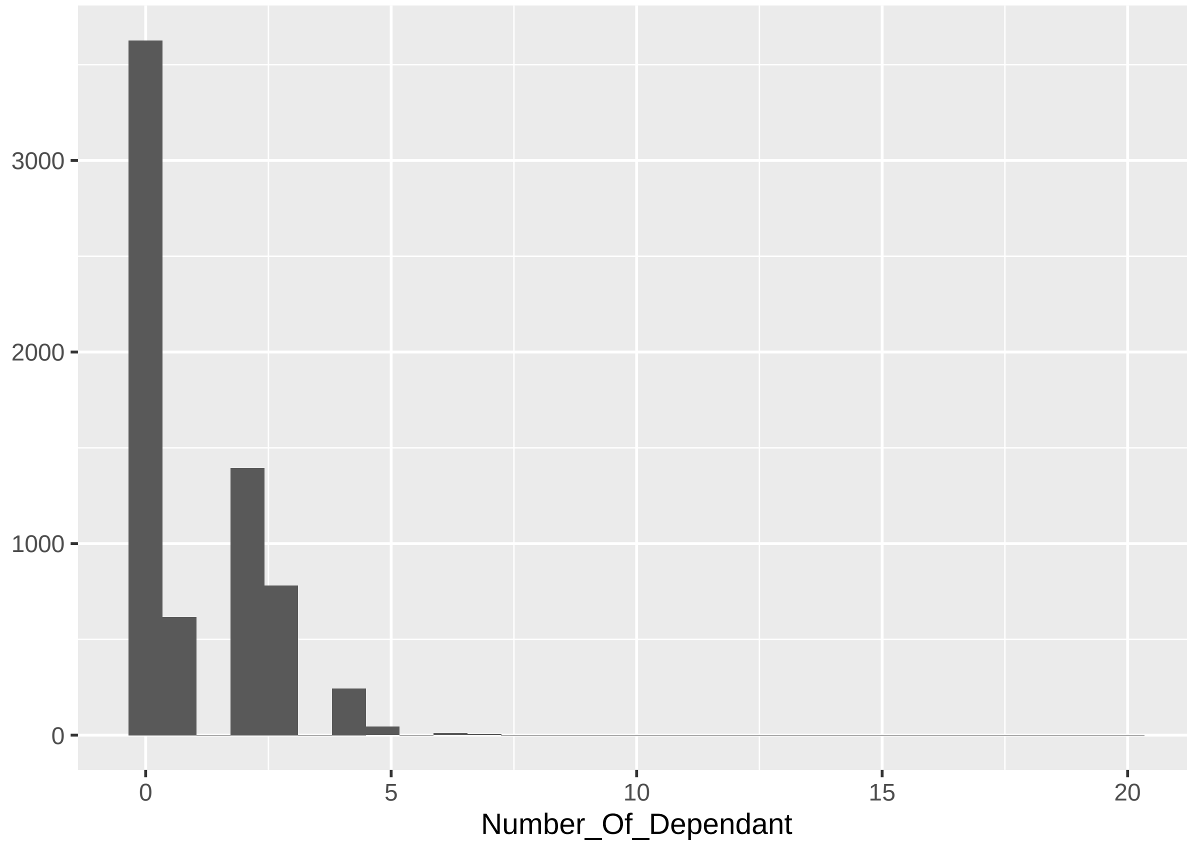

6000
4000
2000
0
NP_Client
P_Client
P_Client



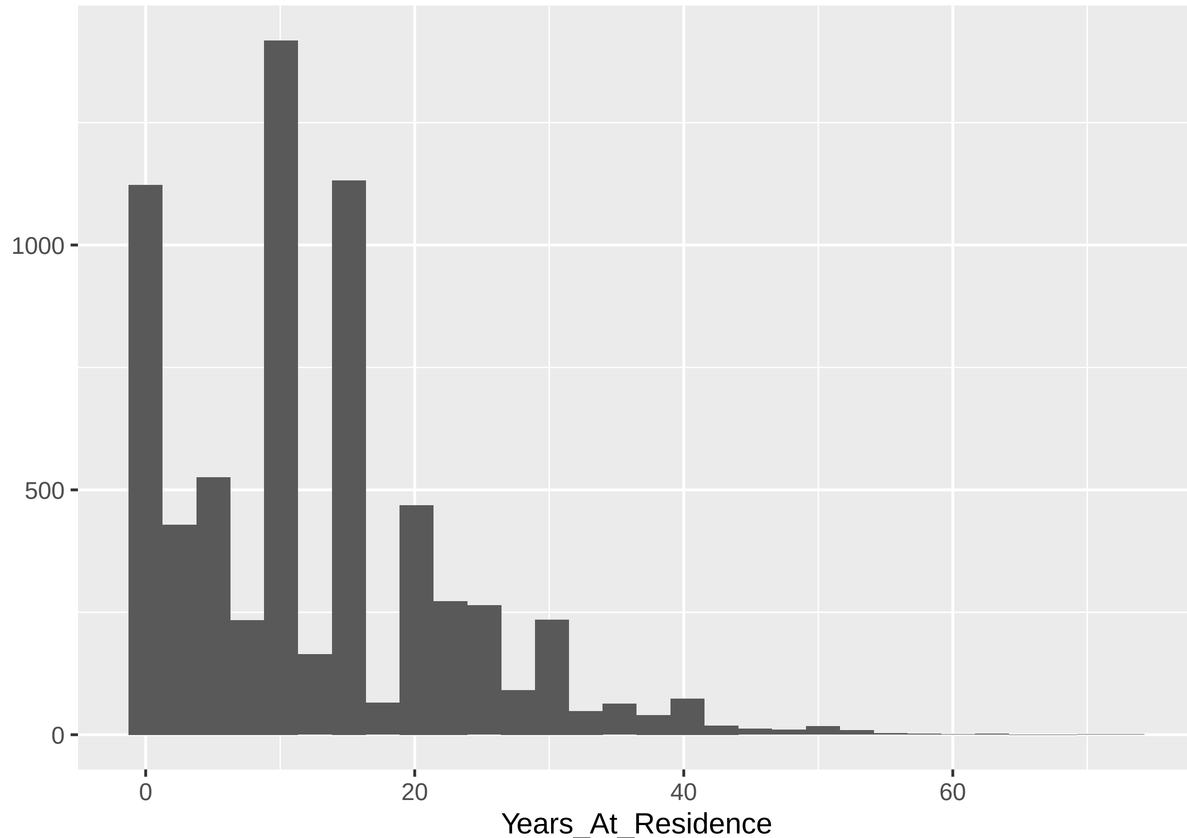
6000
4000
2000
0
Secondary_or_Less
Diploma
University
Master_PhD
Educational_Level




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

## Warning: Removed 2 rows containing non-finite values (stat_bin).
```

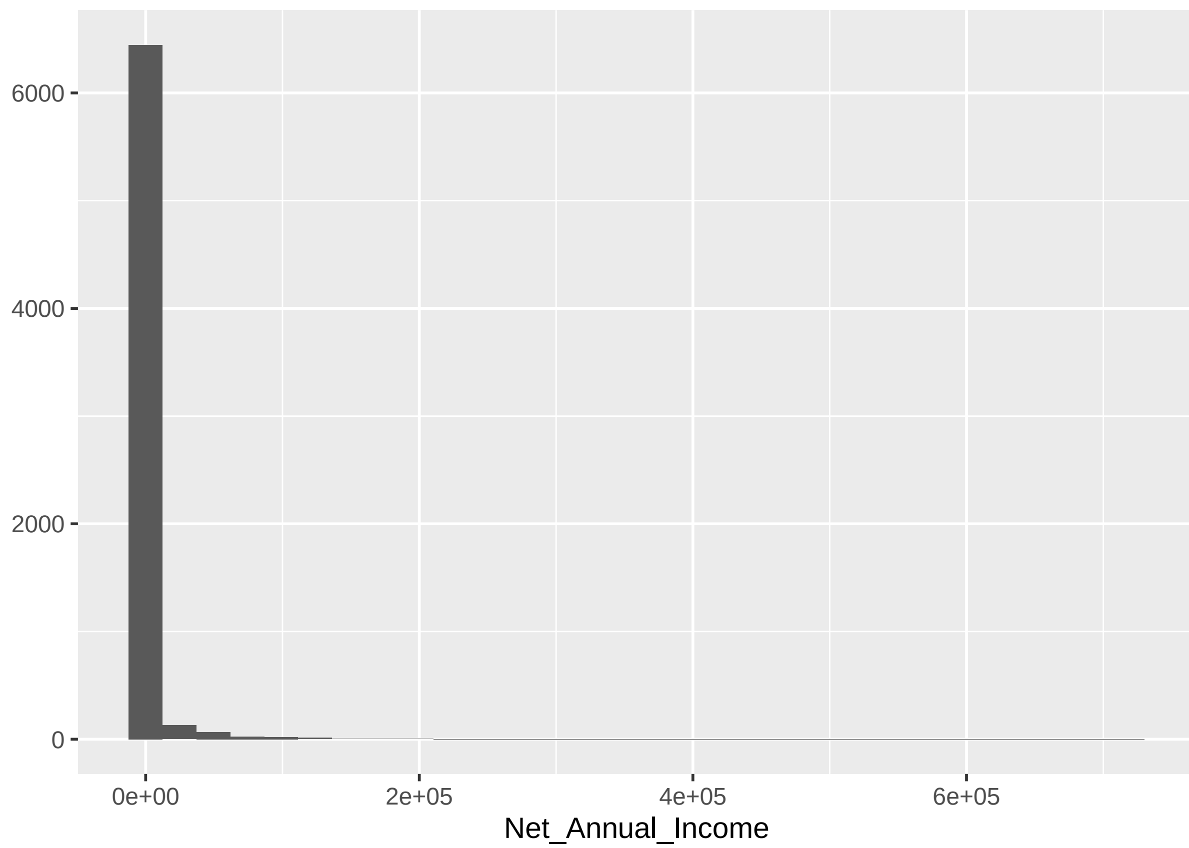



```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

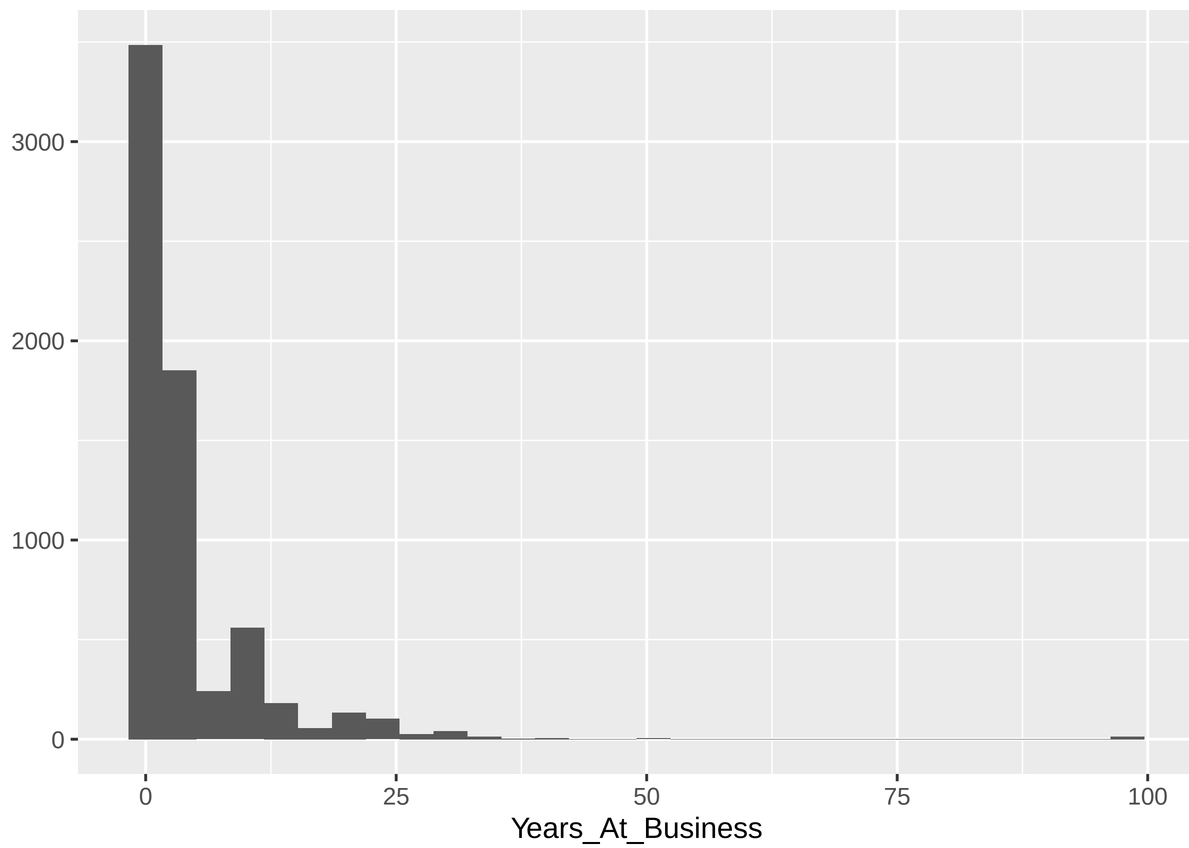


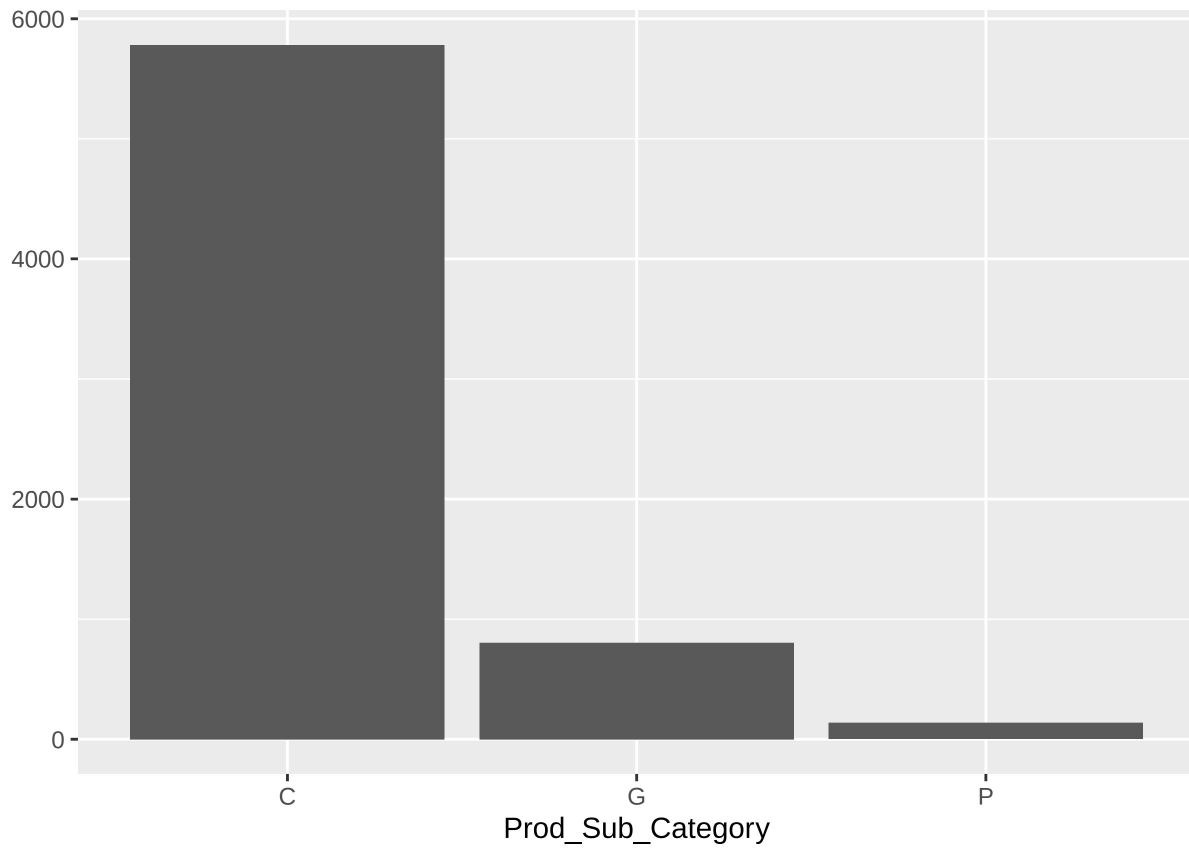

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

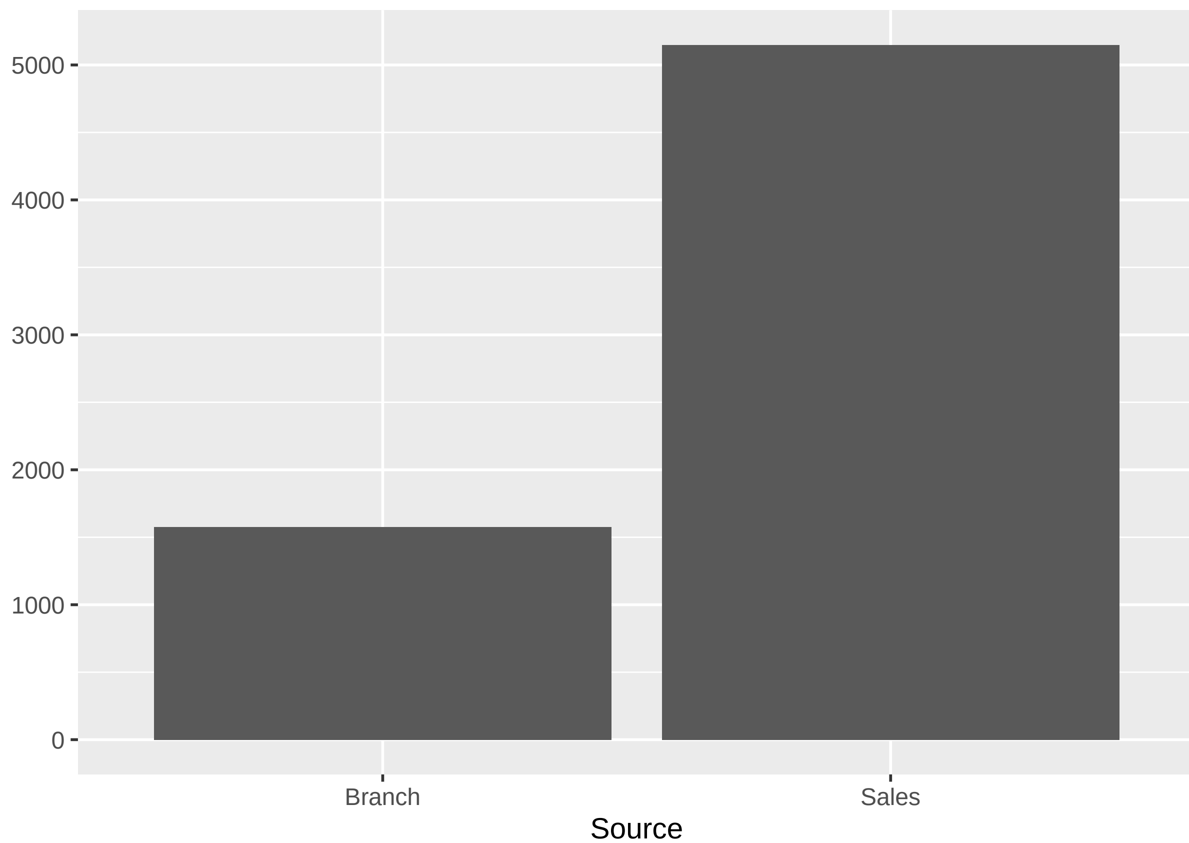
```
## Warning: Removed 3 rows containing non-finite values (stat_bin).
```

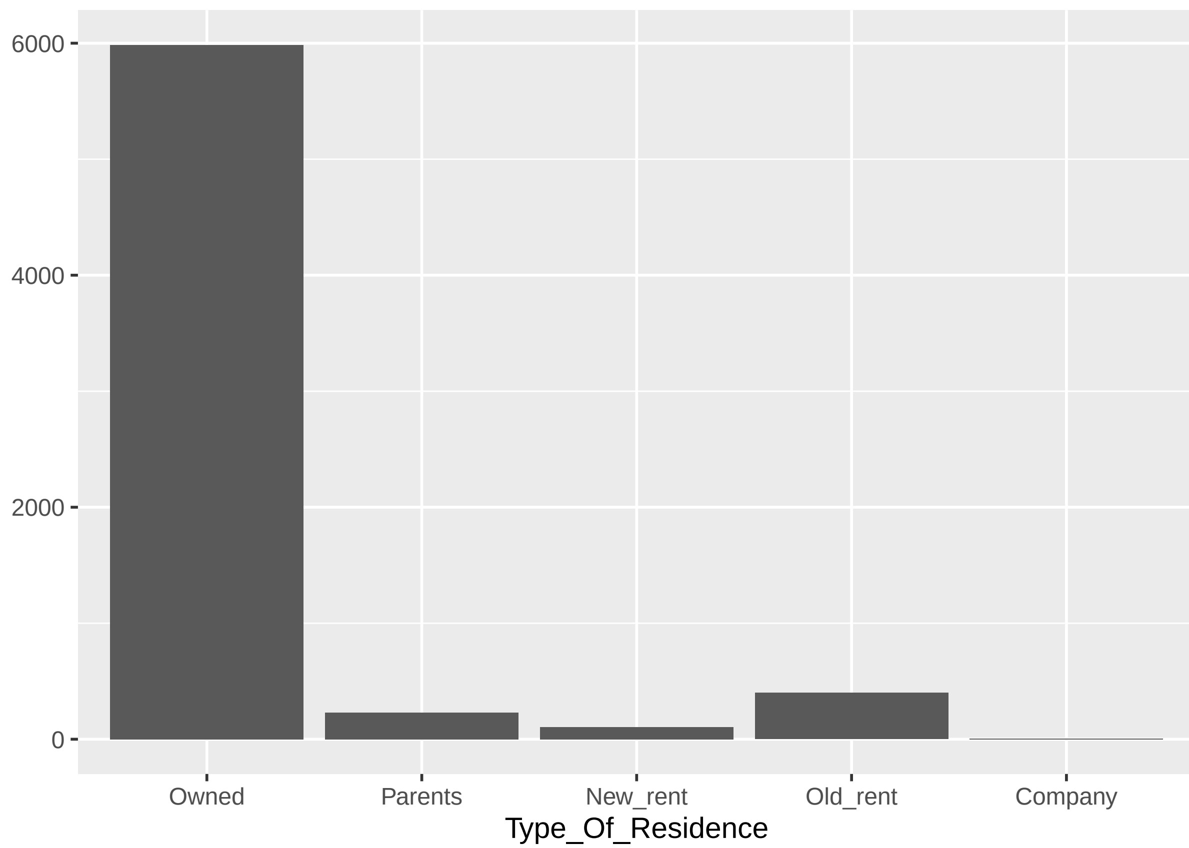



```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.

## Warning: Removed 4 rows containing non-finite values (stat_bin).
```

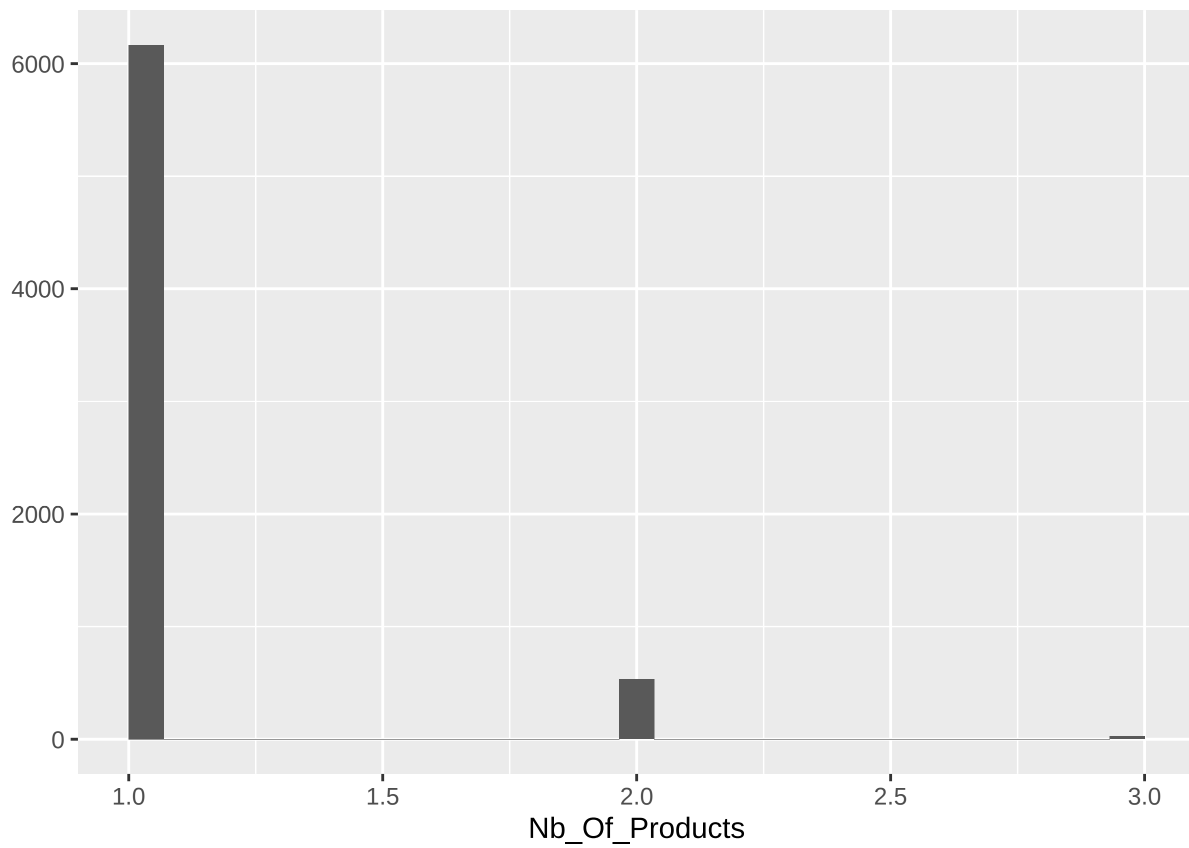

3000
2000
1000
0
0
25
50
75
100
Years_At_Business

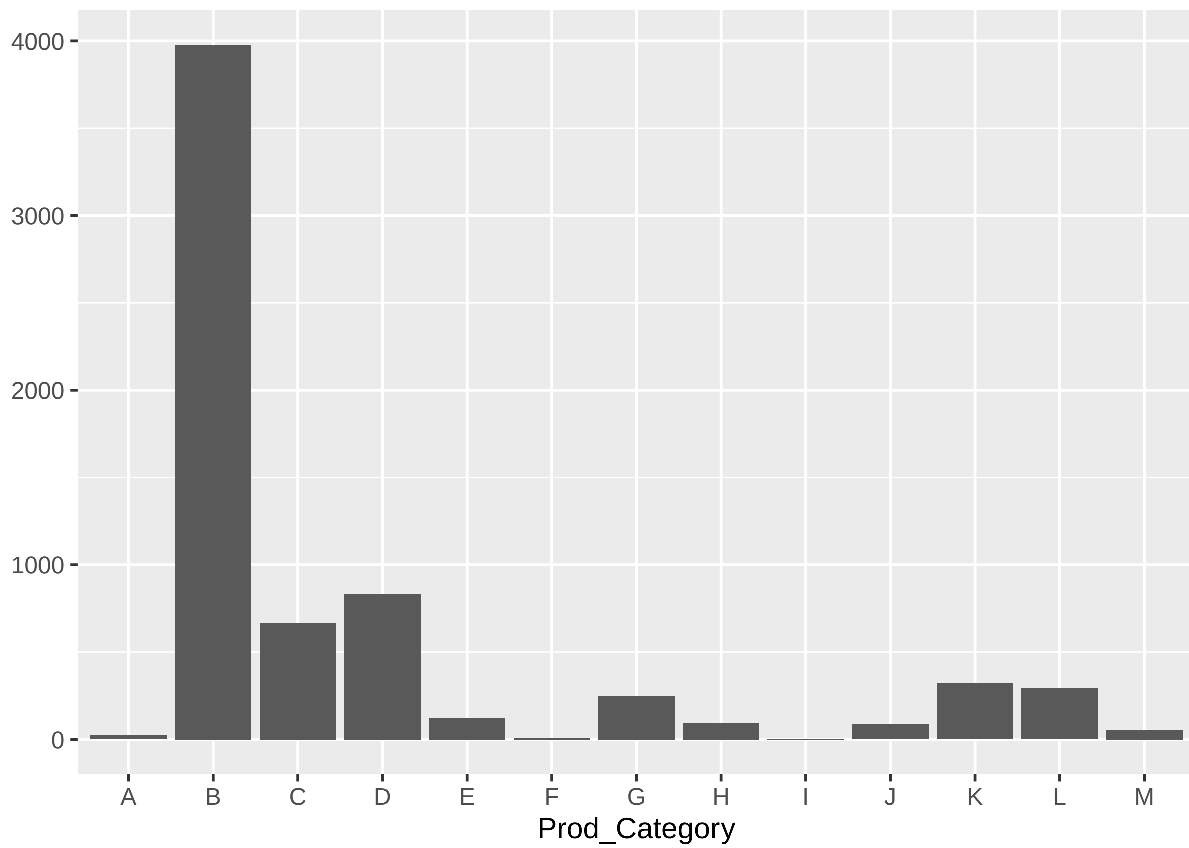

6000
4000
2000
0
C
G
P
Prod_Sub_Category


5000
4000
3000
2000
1000
0
Branch
Sales
Source



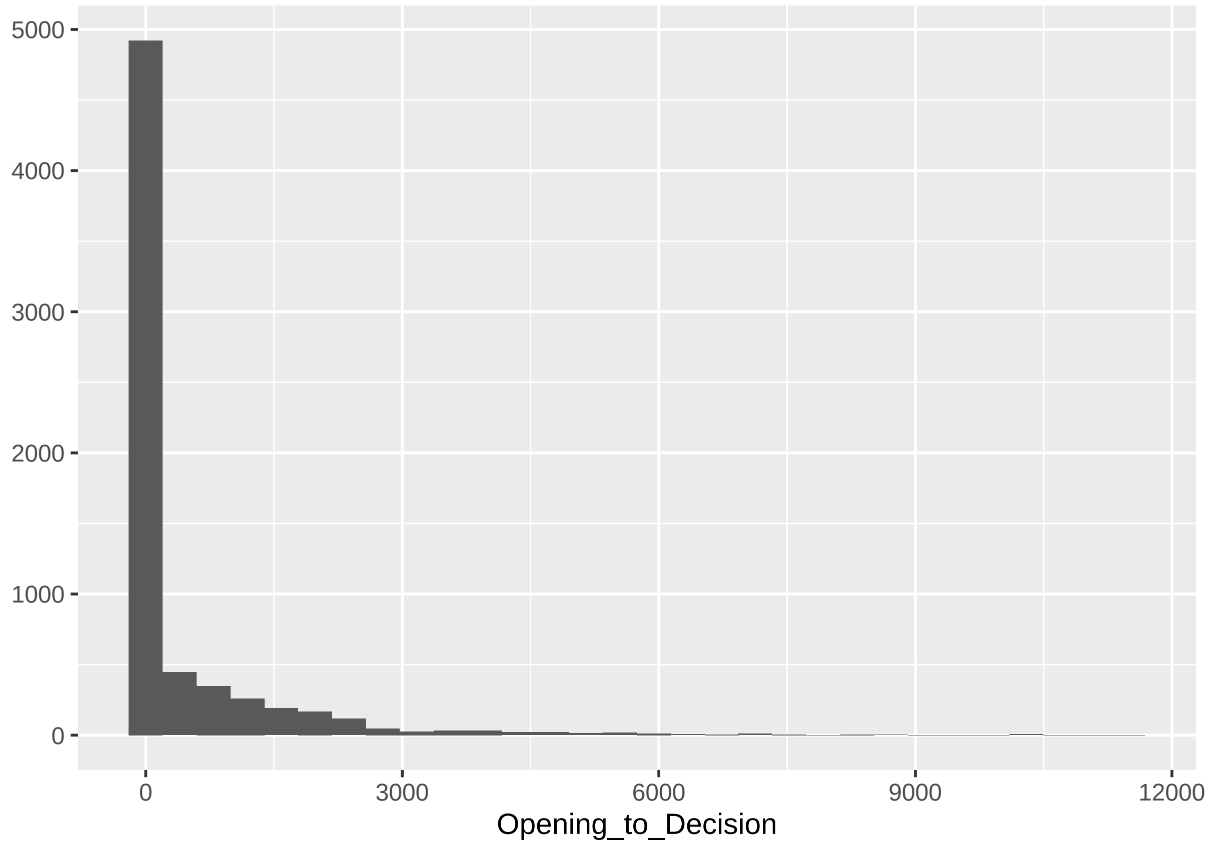

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```


6000
4000
2000
0
1.0
1.5
2.0
2.5
3.0
Nb_Of_Products




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


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



Observations: - Among numerical variables, Age, Net_Annual_Income and Opening_to_Decision are continuous (the rest are discrete).

- The dataset is not well balanced regarding the target variable
- Opening_to_decision: very right skewed with a left peak
- Outliers: number of dependant has max at 20? / years at business has max at 98 ?? and net annual income has a lot of very small (min at 1) and very large values (max at 717792).
- NAN : ** Number_Of_Dependant has 2 NA ** Net_Annual_Income has 3 NA ** Years_At_Business has 4 NA

Dealing with the NANs

```
list_na <- apply(df, 2, anyNA)
which(list_na == TRUE)
```

```
## Number_Of_Dependant   Net_Annual_Income   Years_At_Business
##                   7                   9                  10
```

```
col_indices = which(names(df) %in% names(which(list_na==TRUE)))
```

```r
# Median for numerical variables
medians <- apply(select(df, col_indices),
                        2,
                        median,
                        na.rm = TRUE)
medians
```

```
## Number_Of_Dependant   Net_Annual_Income   Years_At_Business
##                   0                  36                   1
```

```r
# Replace in the numeric variable
df <- df %>% mutate(Number_Of_Dependant_2  = ifelse(is.na(Number_Of_Dependant),
                                                    medians[1],
                                                    Number_Of_Dependant),
                    Net_Annual_Income_2= ifelse(is.na(Net_Annual_Income),
                                                medians[2],
                                                Net_Annual_Income),
                    Years_At_Business_2= ifelse(is.na(Years_At_Business),
                                                medians[3],
                                                Years_At_Business))

# Drop columns
df2 <- df[,-col_indices]
```

Net Annual Income

```r
# We turn it into a qualitative variable
quantiles = quantile(df2$Net_Annual_Income_2, seq(.05, 1-0.05, 0.1))
New_var = 0 * df2$Net_Annual_Income_2
for (i in 1:length(quantiles)) {
  New_var = New_var + (df2$Net_Annual_Income_2 >= quantiles[i])
}
New_var = factor(New_var, ordered=FALSE, levels=c(0:length(quantiles)))
df2$Net_Annual_Income_2 = New_var
```

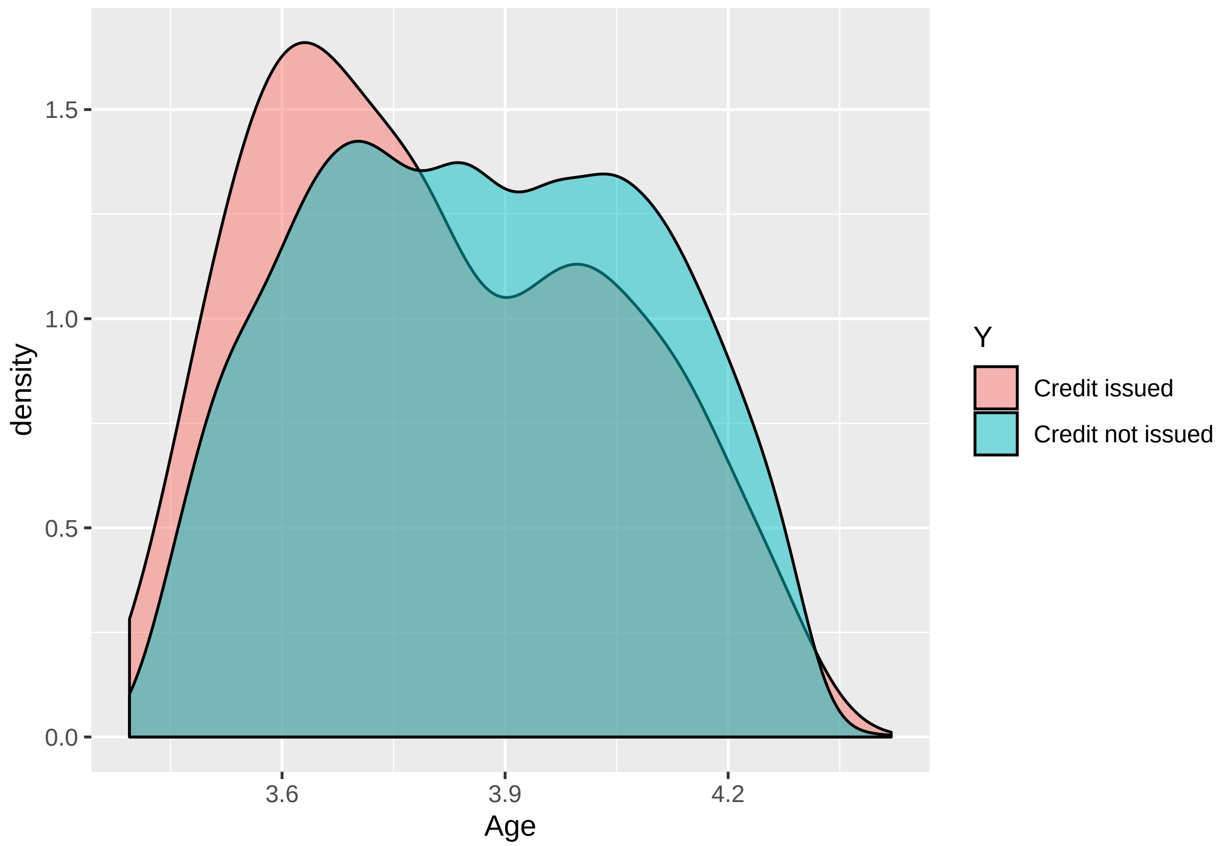
Log+1-transforming the numerical variables with large variance

It brings variables closer to normality and mitigates the effect of outliers

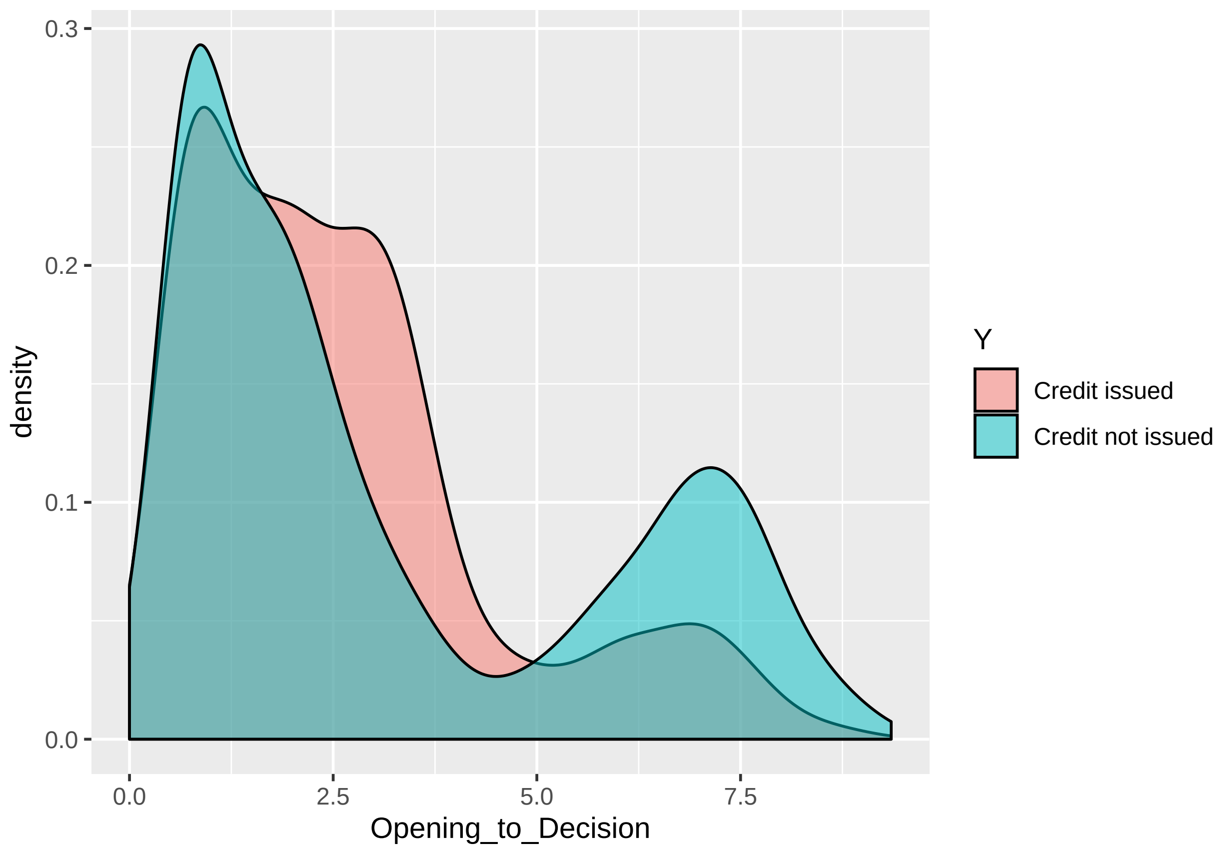
```r
df2$Age = log(df2$Age + 1)
df2$Years_At_Residence = log(df2$Years_At_Residence + 1)
df2$Years_At_Business_2 = log(df2$Years_At_Business_2 + 1)
df2$Number_Of_Dependant_2 = log(df2$Number_Of_Dependant_2 + 1)
df2$Opening_to_Decision = log(df2$Opening_to_Decision + 1)
```

Conditional density plots for continous numerical variables

```r
cond_density_plot <- function(var) {
  ggplot(df2) +
  aes(x = var, fill = Y) +
  xlab(substring(deparse(substitute(var)), 5)) +
  scale_fill_discrete(name = "Y", labels = c("Credit issued", "Credit not issued")) +
  geom_density(alpha = 0.5)
}

cond_density_plot(df2$Age)
```



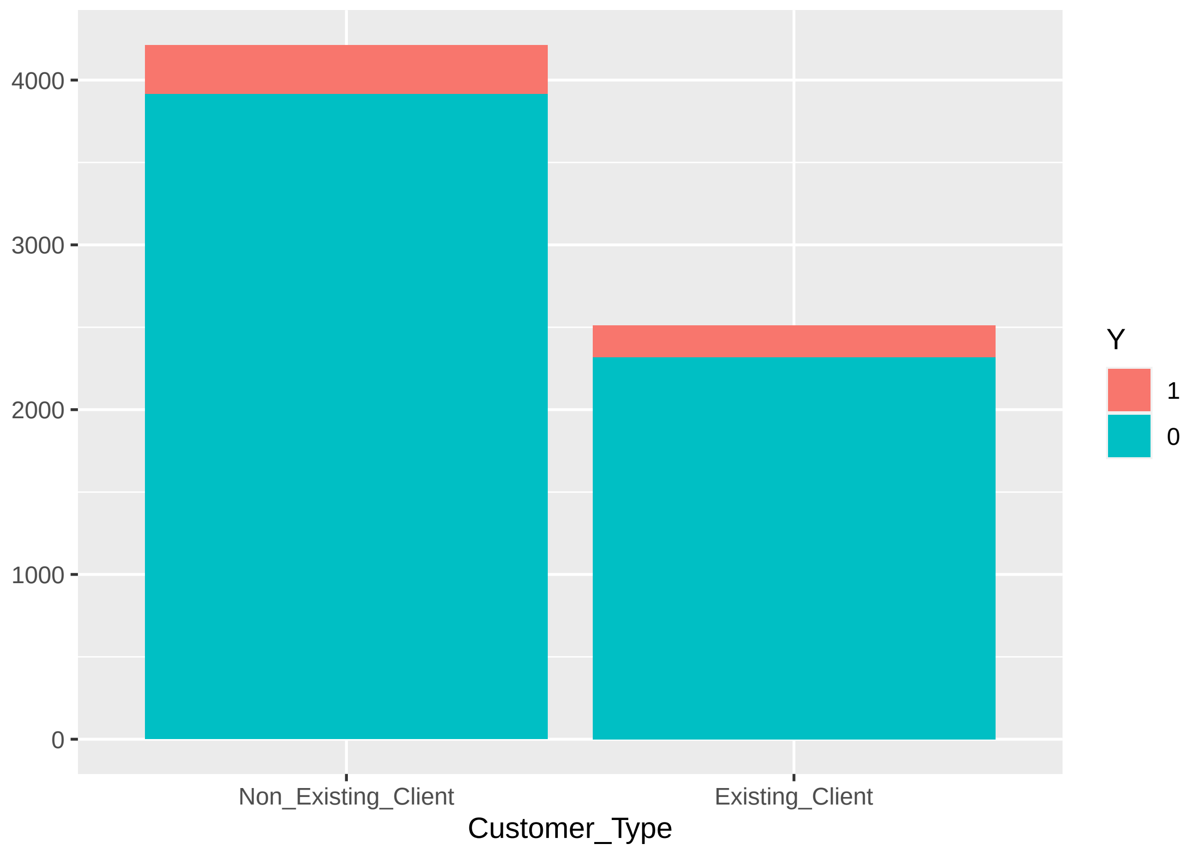

```r
cond_density_plot(df2$Opening_to_Decision)
```

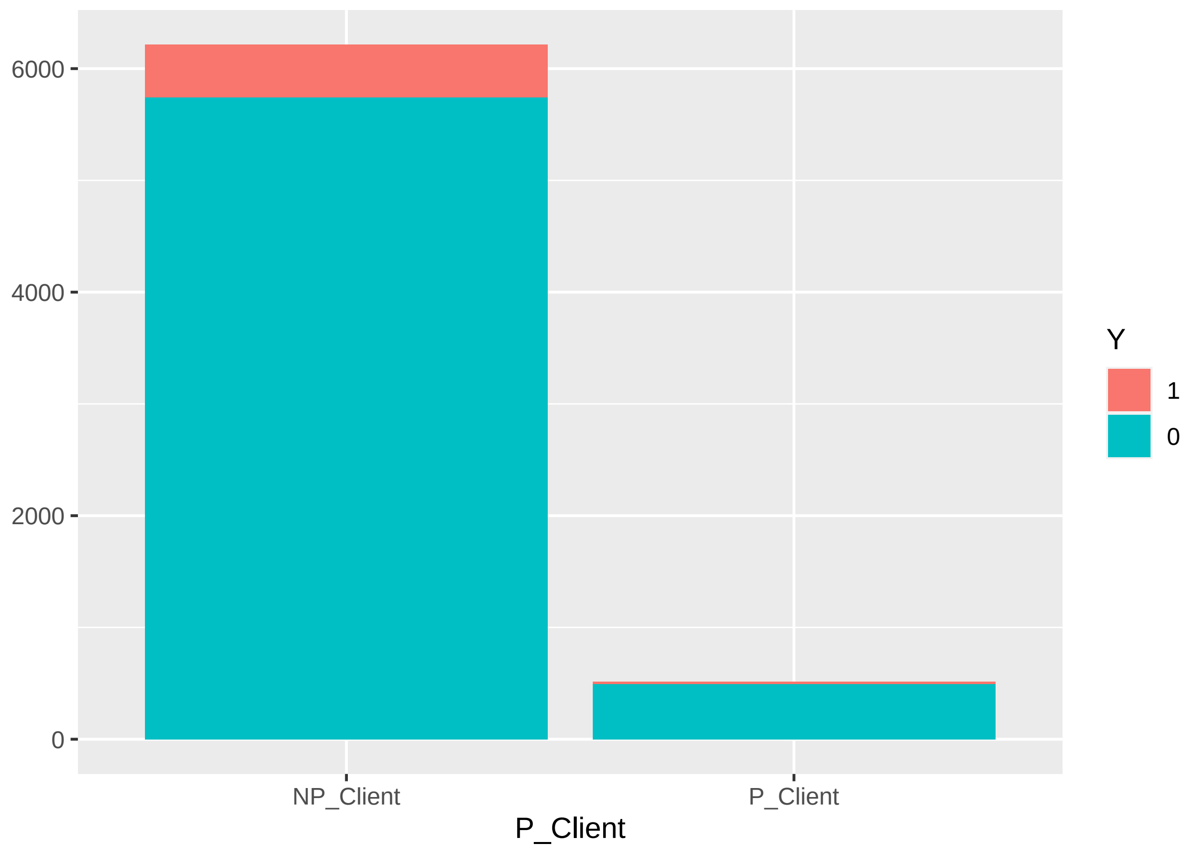


We can see that the conditional distributions look very similar in both cases. As we said, people for which the credit is issued tend to receive a quicker answer from the bank.

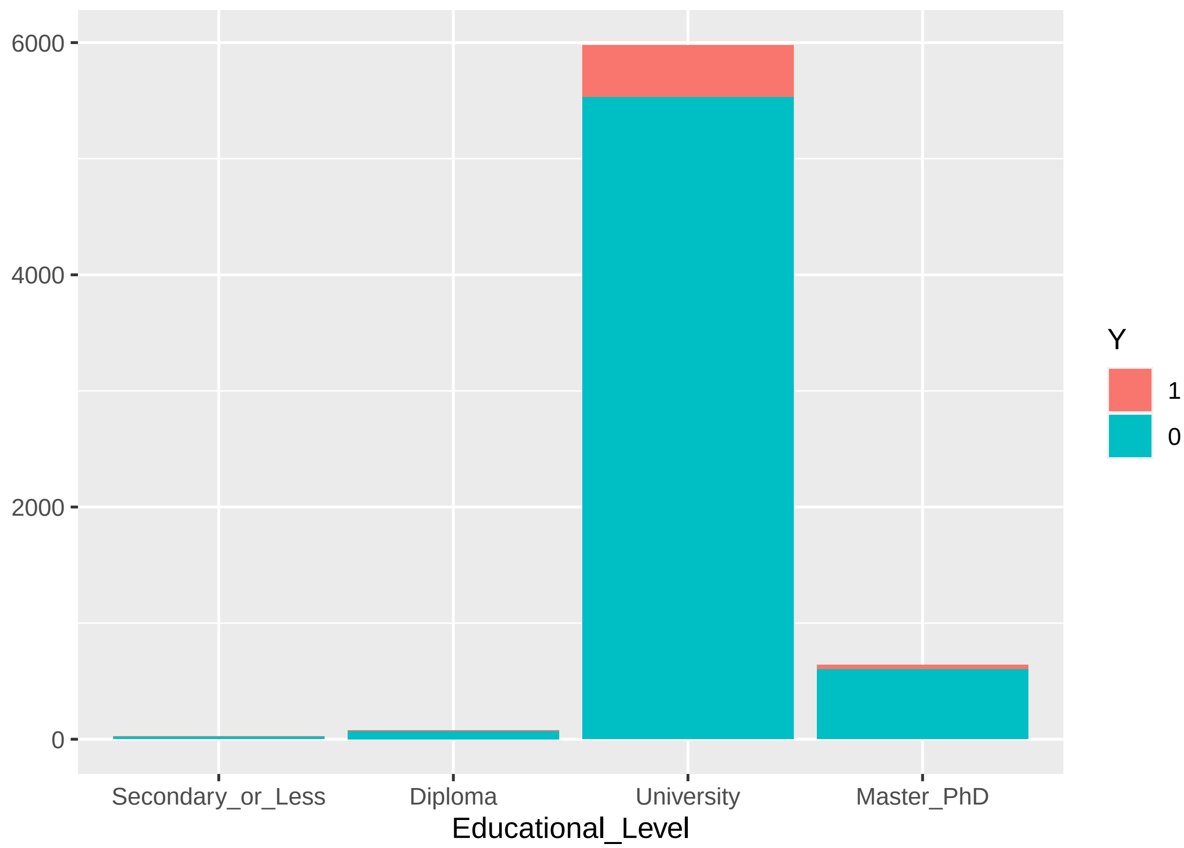
Conditional barplots for the other variables

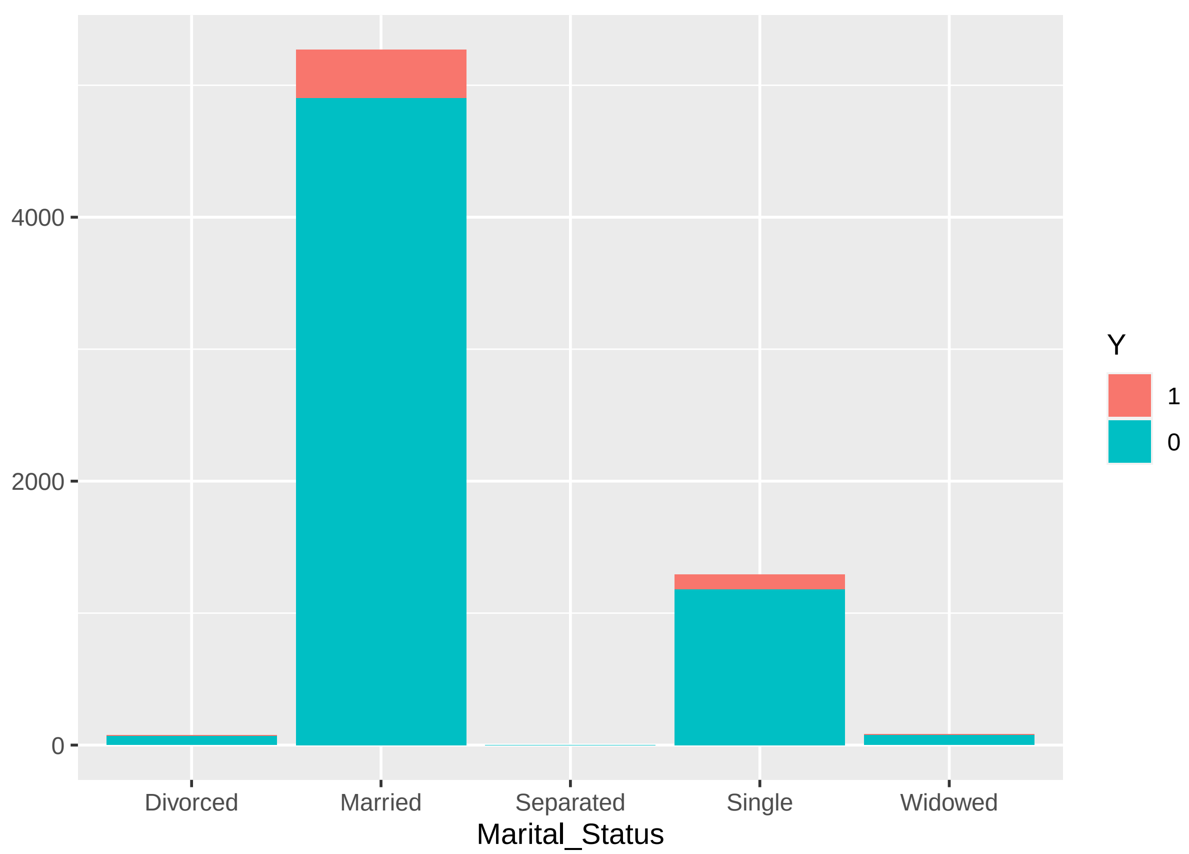
```
to_remove <- c("Age", "Opening_to_Decision", "Id_Customer", "Y", "is_train")
namesdf2 <- names(df2)[! names(df2) %in% to_remove]
for (varname in namesdf2) {
  print(qplot(data = df2, get(varname), fill = Y, xlab = varname))
}
```


4000
3000
2000
1000
0
Non_Existing_Client
Existing_Client
Customer_Type
Y
1
0

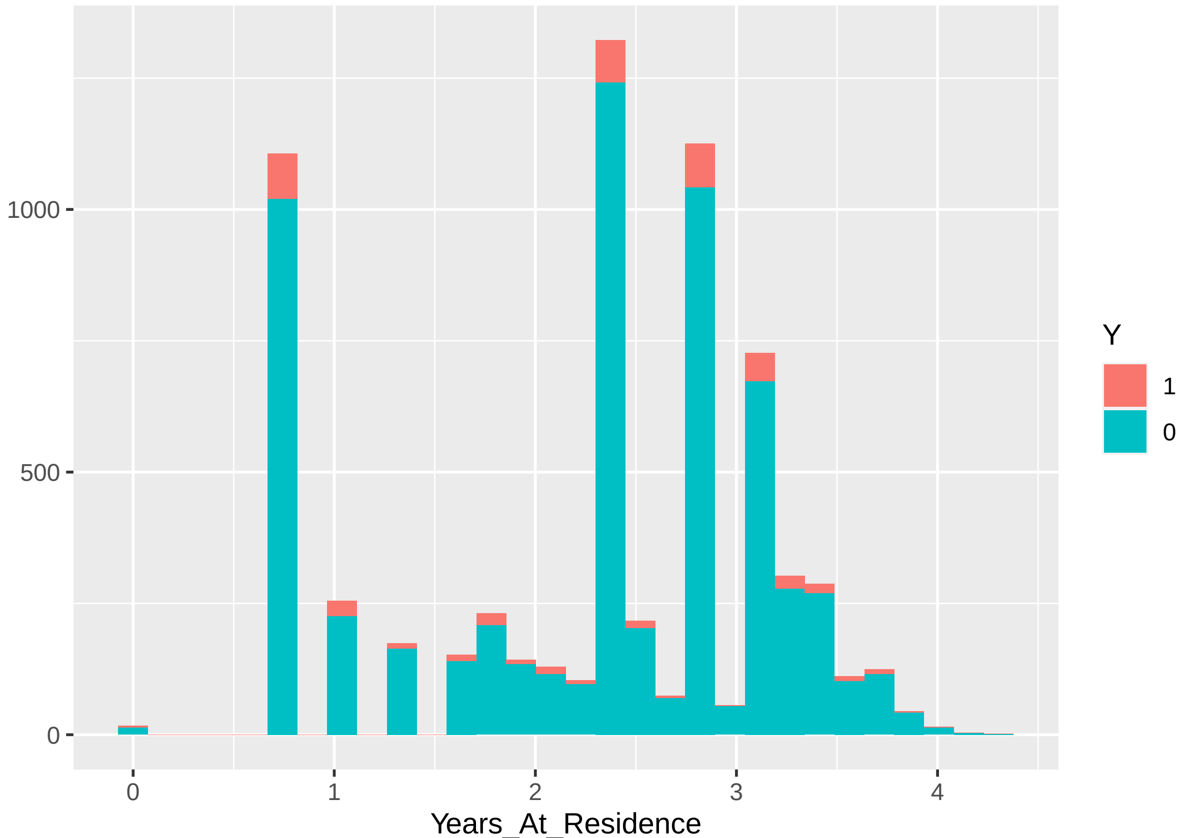


6000
4000
2000
0
NP_Client
P_Client
P_Client
Y
1
0

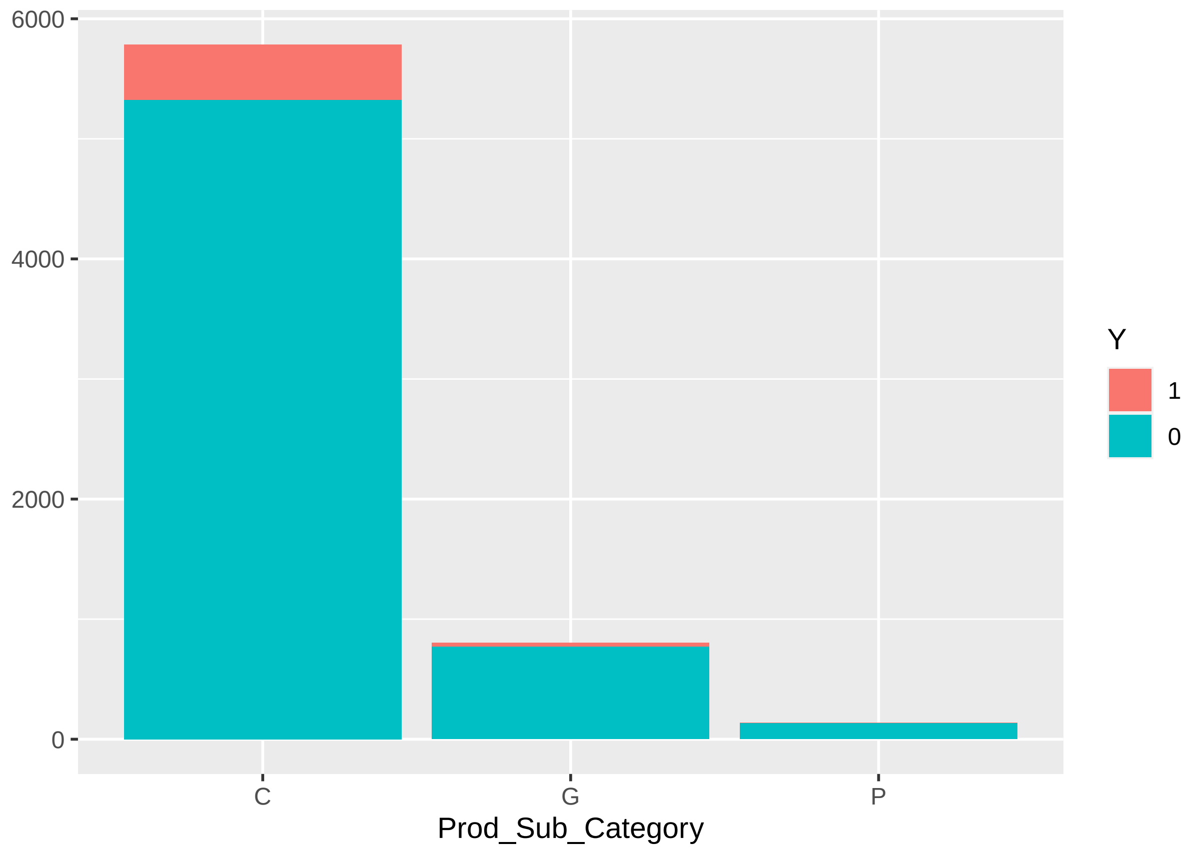

6000
4000
2000
0
Secondary_or_Less
Diploma
University
Master_PhD
Educational_Level
Y
1
0

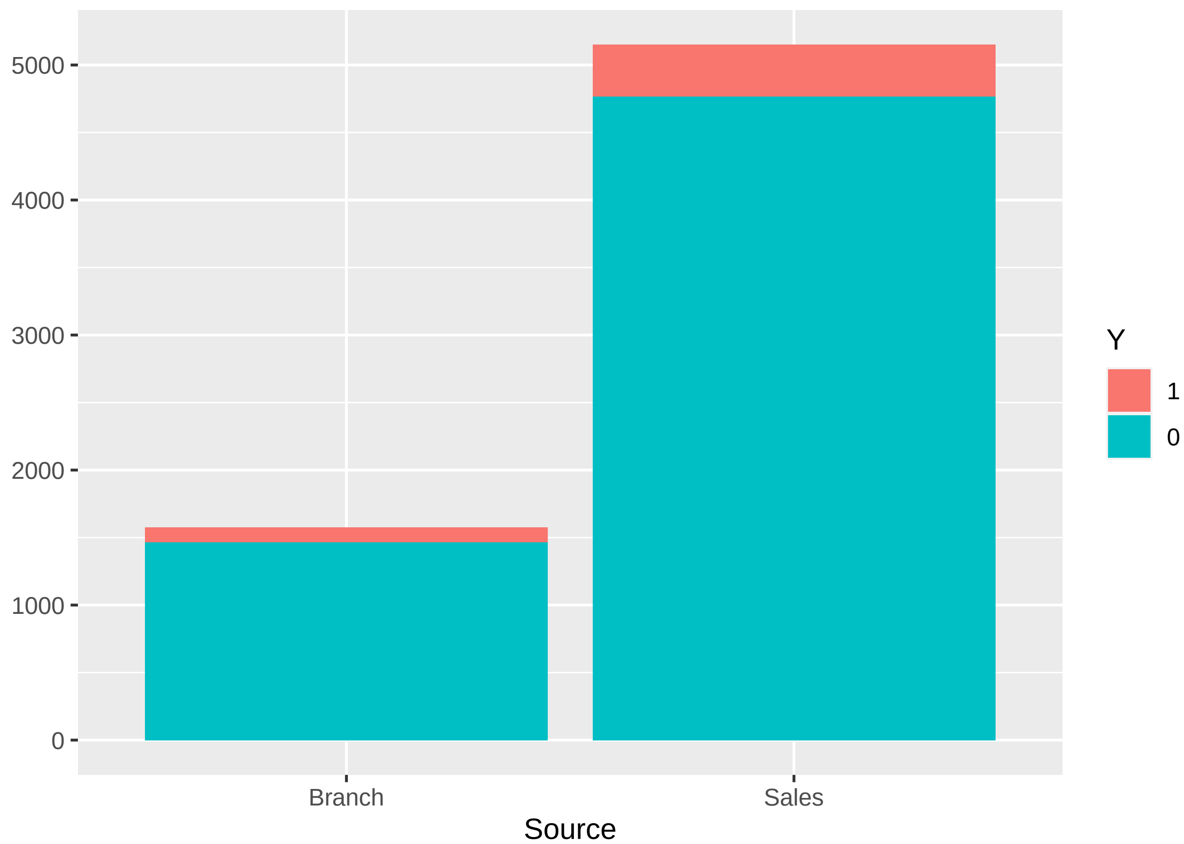



```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

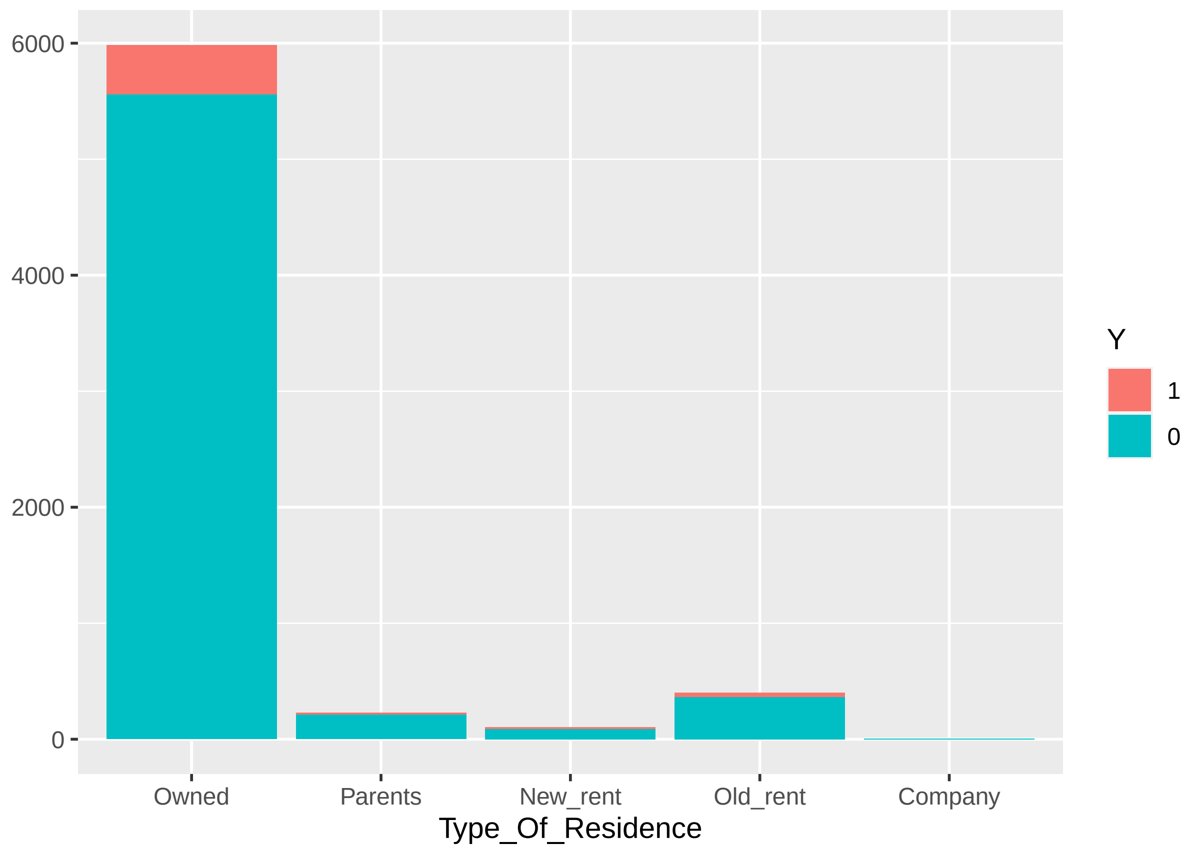


1000
500
0
0
1
2
3
4
Years_At_Residence
Y
1
0

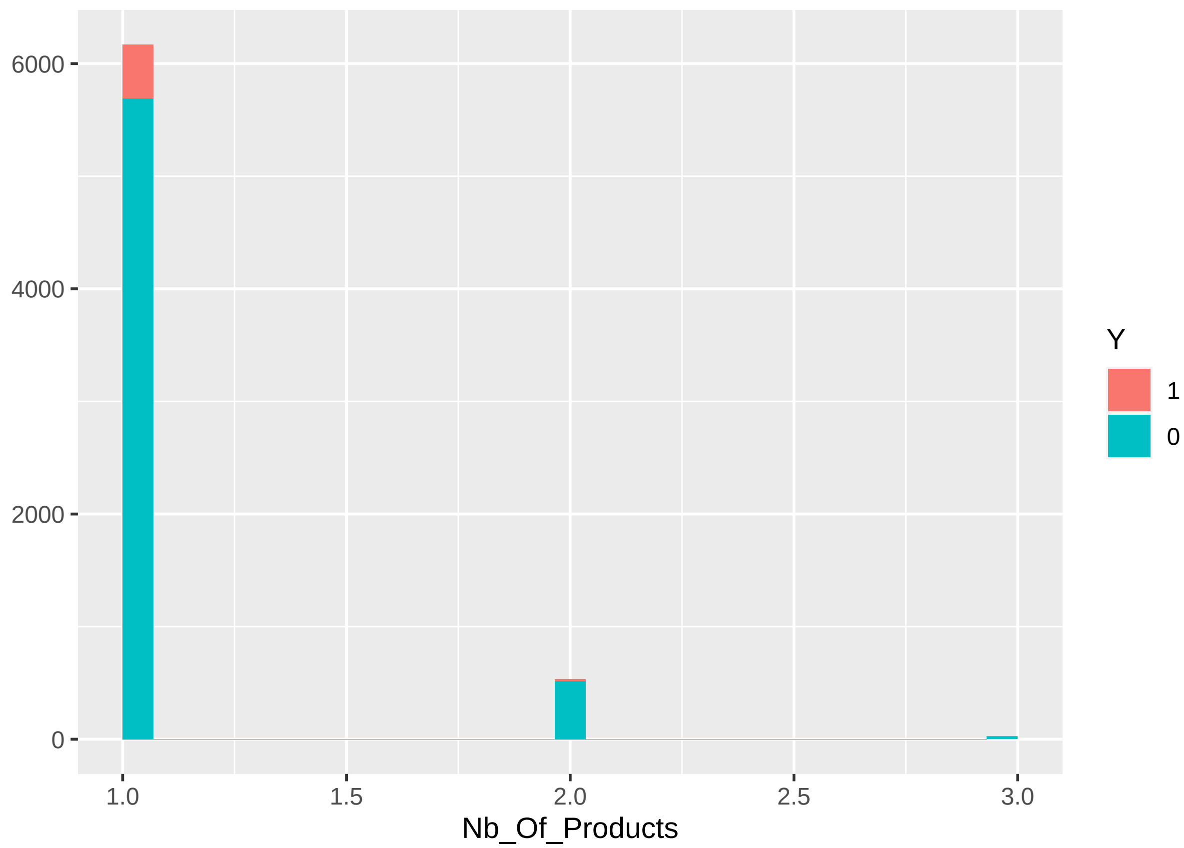

6000
4000
2000
0
C
G
P
Prod_Sub_Category
Y
1
0

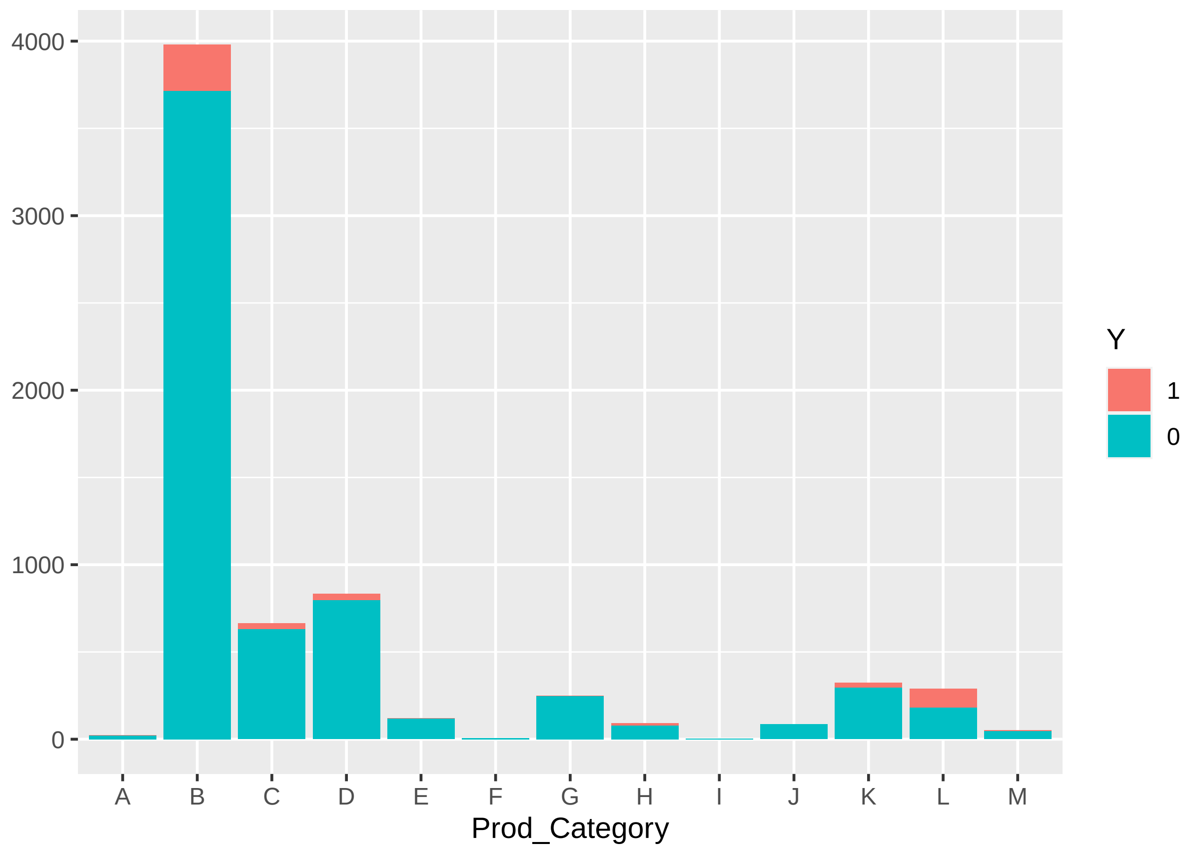


5000
4000
3000
2000
1000
0
Branch
Sales
Source
Y
1
0

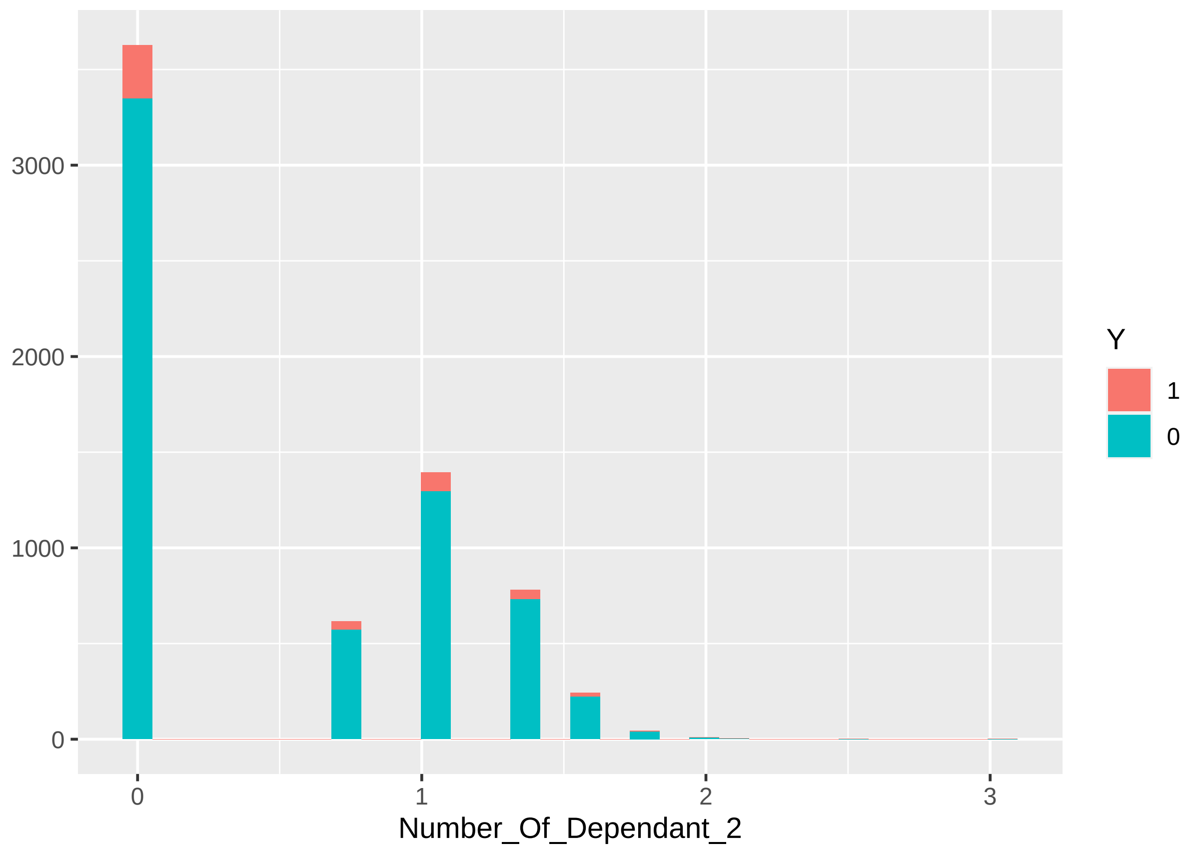


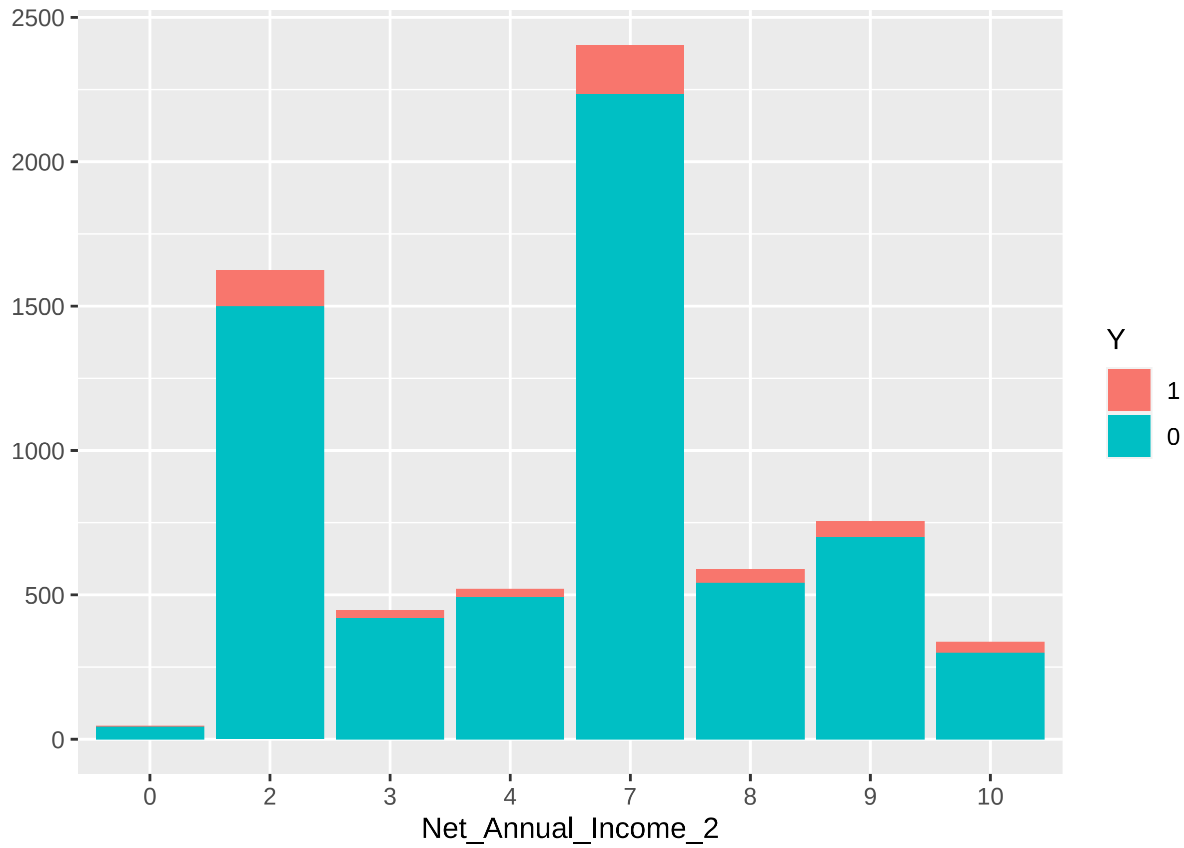

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```


6000
4000
2000
0
1.0
1.5
2.0
2.5
3.0
Nb_Of_Products
Y
1
0

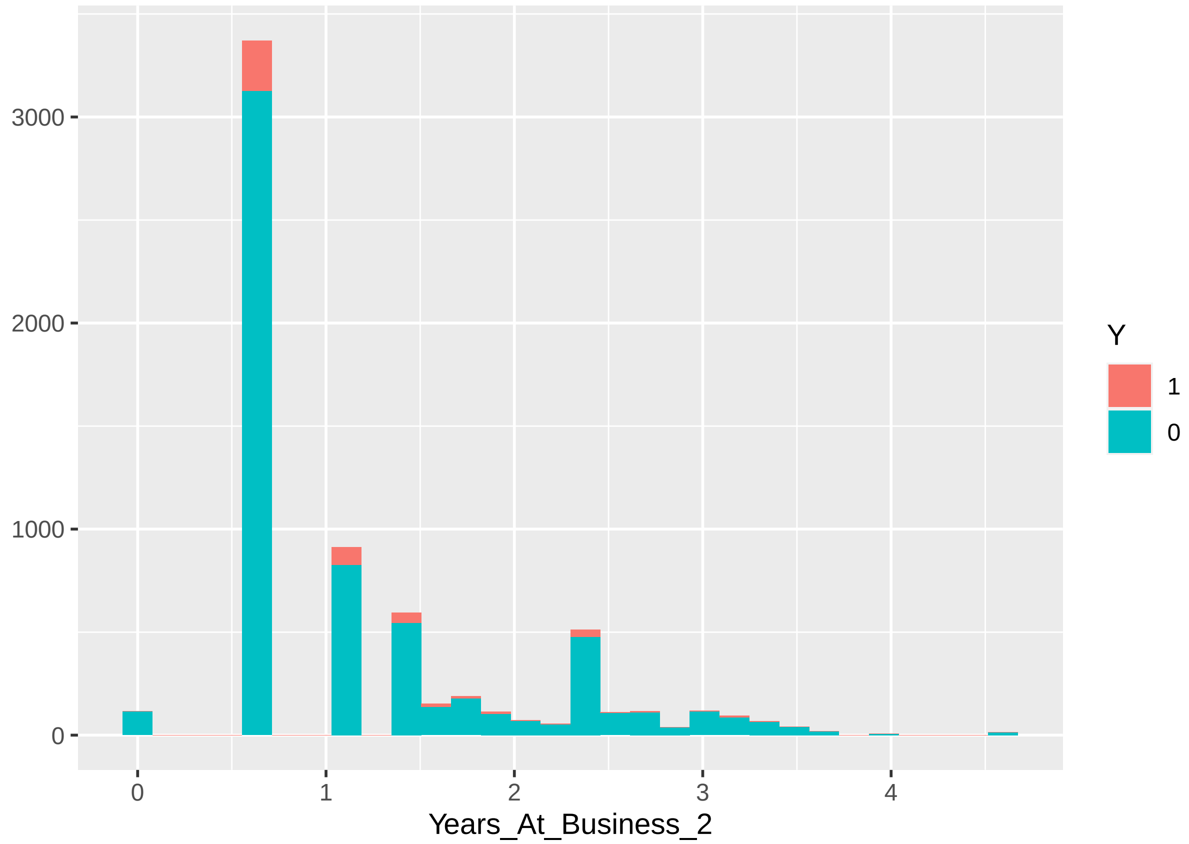



```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```


3000
2000
1000
0
0
1
2
3
Number_Of_Dependant_2
Y
1
0




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



Creating dummies (One-hot encoding)

```
one_hot_encode <- function(var, dataframe) {
  # Encoding
  new_df <- cbind(dataframe, as.data.frame(dummy_cols(var))[,-1])

  # Removing last modality
  new_df[length(new_df)] <- NULL

  # Naming correctly
  var_name = substring(deparse(substitute(var)), 5)
  nb_levels <- length(levels(var))
  levels <- levels(var)[1:nb_levels-1]
  for (i in 1:length(levels)) {
    new_name = paste(var_name, levels[i], sep="_")
    levels[i] <- new_name
  }
  for (i in 1:length(levels)) {
    colnames(new_df)[i + length(dataframe)] <- levels[i]
  }

  # Removing initial variable
  new_df = new_df[, !(names(new_df) == var_name)]
```

```r
  return(new_df)
}

vars_to_encode = c("Customer_Type", "P_Client", "Educational_Level",
                   "Marital_Status", "Prod_Sub_Category", "Source",
                   "Type_Of_Residence")
df2 <- one_hot_encode(df2$Customer_Type, df2)
df2 <- one_hot_encode(df2$P_Client, df2)
df2 <- one_hot_encode(df2$Educational_Level, df2)
df2 <- one_hot_encode(df2$Marital_Status, df2)
df2 <- one_hot_encode(df2$Prod_Sub_Category, df2)
df2 <- one_hot_encode(df2$Source, df2)
df2 <- one_hot_encode(df2$Type_Of_Residence, df2)
df2 <- one_hot_encode(df2$Prod_Category, df2)
```

Preparation for modelling

```r
# Removing Id_customer
df2$Id_Customer <- NULL

# train and test sets
train = df2[df2$is_train == 1,]
train$is_train <- NULL

test = df2[df2$is_train == 0,]
test$is_train <- NULL

true_Y_test <- test$Y
test$Y <- NULL
```

Re sampling

```r
trainSplit <- SMOTE(Y ~ ., data = train, perc.over = 100, perc.under = 200)
# perc.over (under-sampling): what percentage of extra cases from the minority class are generated, bas
# perc.under (over-sampling): what percentage of cases from the majority class are selected (ex: 100 wi
table(train$Y)
```

```
##
##    1    0
##  393 4987
```

```r
table(trainSplit$Y)
```

```
##
##   1   0
## 786 786
```

Modelling

We impose 10 Cross-validations and fix a Seed to compare models without being exposed to randomness

```
V <- 10
T <- 4
TrControl <- trainControl(method = "repeatedcv",
                          number = V,
                          repeats = T)
set.seed(345)
```

```
Errs_folds <- function(Model, Name) {
  return(data.frame(Model$resample, model = Name))
}

Err_train <- function(errs_folds, Model, Name) {
  errs <- Errs_folds(Model, Name)
  err_train <- data.frame(mAccuracy = mean(errs$Accuracy, na.rm = TRUE),
                          mKappa = mean(errs$Kappa, na.rm = TRUE))
  return(err_train)
}

Err_test <- function(Model, Name) {
  err_test <- data.frame(t(postResample(predict(Model, newdata = test),
                                        true_Y_test)),
                         model = Name)
  return(err_test)
}
```

```
CaretLearnAndDisplay <- function(Name, Formula, Method) {
  Model <- train(as.formula(Formula),
                 data = trainSplit,
                 method = Method,
                 trControl = TrControl)
  print(Model)
  errs_folds <- Errs_folds(Model, Name)
  print(errs_folds)
  print(Err_train(errs_folds, Model, Name))
  print(Err_test(Model, Name))
}
```

Trying various models

Logistic model

```
CaretLearnAndDisplay("Logistic", "Y ~ .", "glm")
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type
## == : prediction from a rank-deficient fit may be misleading
```

```
## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type
## == : prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type
## == : prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type
## == : prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type
## == : prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type
## == : prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type
## == : prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type
## == : prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type
## == : prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type
## == : prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type
## == : prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type
## == : prediction from a rank-deficient fit may be misleading

## Warning: glm.fit: algorithm did not converge

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type
## == : prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type
## == : prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type
## == : prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type
## == : prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type
## == : prediction from a rank-deficient fit may be misleading
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type
## == : prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type
## == : prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type
## == : prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type
## == : prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type
## == : prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type
## == : prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type
## == : prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type
## == : prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type
## == : prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type
## == : prediction from a rank-deficient fit may be misleading

## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type
## == : prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type
## == : prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type
## == : prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type
## == : prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type
## == : prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type
## == : prediction from a rank-deficient fit may be misleading
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type
## == : prediction from a rank-deficient fit may be misleading

## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type
## == : prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type
## == : prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type
## == : prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type
## == : prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type
## == : prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type
## == : prediction from a rank-deficient fit may be misleading

## Generalized Linear Model
##
## 1572 samples
##   35 predictor
##    2 classes: '1', '0'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 4 times)
## Summary of sample sizes: 1415, 1415, 1414, 1415, 1415, 1414, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.7162987  0.4326355
##
##     Accuracy     Kappa    Resample    model
## 1  0.7070064 0.4138938 Fold01.Rep1 Logistic
## 2  0.6878981 0.3760240 Fold02.Rep1 Logistic
## 3  0.7341772 0.4683544 Fold03.Rep1 Logistic
## 4  0.7006369 0.4012982 Fold04.Rep1 Logistic
## 5  0.6878981 0.3755175 Fold05.Rep1 Logistic
## 6  0.7658228 0.5316456 Fold06.Rep1 Logistic
## 7  0.6751592 0.3508715 Fold07.Rep1 Logistic
## 8  0.6858974 0.3717949 Fold08.Rep1 Logistic
## 9  0.7468354 0.4936709 Fold09.Rep1 Logistic
## 10 0.8025478 0.6049192 Fold10.Rep1 Logistic
## 11 0.7051282 0.4102564 Fold01.Rep2 Logistic
## 12 0.7006369 0.4012010 Fold02.Rep2 Logistic
## 13 0.7564103 0.5128205 Fold03.Rep2 Logistic
```

```
## 14 0.7179487 0.4358974 Fold04.Rep2 Logistic
## 15 0.6815287 0.3627212 Fold05.Rep2 Logistic
## 16 0.6898734 0.3797468 Fold06.Rep2 Logistic
## 17 0.7151899 0.4303797 Fold07.Rep2 Logistic
## 18 0.6772152 0.3544304 Fold08.Rep2 Logistic
## 19 0.7594937 0.5189873 Fold09.Rep2 Logistic
## 20 0.7215190 0.4430380 Fold10.Rep2 Logistic
## 21 0.7628205 0.5256410 Fold01.Rep3 Logistic
## 22 0.6687898 0.3377677 Fold02.Rep3 Logistic
## 23 0.6898734 0.3797468 Fold03.Rep3 Logistic
## 24 0.6751592 0.3502394 Fold04.Rep3 Logistic
## 25 0.7324841 0.4649464 Fold05.Rep3 Logistic
## 26 0.6518987 0.3037975 Fold06.Rep3 Logistic
## 27 0.7468354 0.4936709 Fold07.Rep3 Logistic
## 28 0.7070064 0.4141791 Fold08.Rep3 Logistic
## 29 0.7579618 0.5161395 Fold09.Rep3 Logistic
## 30 0.7579618 0.5160610 Fold10.Rep3 Logistic
## 31 0.7006369 0.4017835 Fold01.Rep4 Logistic
## 32 0.7500000 0.5000000 Fold02.Rep4 Logistic
## 33 0.7388535 0.4778129 Fold03.Rep4 Logistic
## 34 0.7197452 0.4398313 Fold04.Rep4 Logistic
## 35 0.7468354 0.4936709 Fold05.Rep4 Logistic
## 36 0.7435897 0.4871795 Fold06.Rep4 Logistic
## 37 0.7151899 0.4303797 Fold07.Rep4 Logistic
## 38 0.7215190 0.4430380 Fold08.Rep4 Logistic
## 39 0.6624204 0.3249777 Fold09.Rep4 Logistic
## 40 0.6835443 0.3670886 Fold10.Rep4 Logistic
##   mAccuracy    mKappa
## 1 0.7162987 0.4326355
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type
## == : prediction from a rank-deficient fit may be misleading
```

```
##    Accuracy      Kappa    model
## 1 0.7078067 0.09482515 Logistic
```

Simple tree (CART)

```r
CaretLearnAndDisplay("Tree", "Y ~ .", "treebag")
```

```
## Bagged CART
##
## 1572 samples
##   35 predictor
##    2 classes: '1', '0'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 4 times)
## Summary of sample sizes: 1414, 1414, 1416, 1416, 1414, 1415, ...
## Resampling results:
##
```

```
##   Accuracy   Kappa
##   0.7760899  0.5521779
##
##     Accuracy     Kappa    Resample model
## 1  0.7784810 0.5569620 Fold01.Rep1  Tree
## 2  0.7658228 0.5316456 Fold02.Rep1  Tree
## 3  0.8269231 0.6538462 Fold03.Rep1  Tree
## 4  0.7500000 0.5000000 Fold04.Rep1  Tree
## 5  0.8164557 0.6329114 Fold05.Rep1  Tree
## 6  0.7388535 0.4773041 Fold06.Rep1  Tree
## 7  0.7692308 0.5384615 Fold07.Rep1  Tree
## 8  0.7594937 0.5189873 Fold08.Rep1  Tree
## 9  0.7405063 0.4810127 Fold09.Rep1  Tree
## 10 0.7961783 0.5926707 Fold10.Rep1  Tree
## 11 0.7500000 0.5000000 Fold01.Rep2  Tree
## 12 0.7911392 0.5822785 Fold02.Rep2  Tree
## 13 0.7834395 0.5672827 Fold03.Rep2  Tree
## 14 0.7452229 0.4907558 Fold04.Rep2  Tree
## 15 0.7770701 0.5537962 Fold05.Rep2  Tree
## 16 0.7658228 0.5316456 Fold06.Rep2  Tree
## 17 0.8354430 0.6708861 Fold07.Rep2  Tree
## 18 0.7643312 0.5294451 Fold08.Rep2  Tree
## 19 0.8089172 0.6175085 Fold09.Rep2  Tree
## 20 0.7834395 0.5666504 Fold10.Rep2  Tree
## 21 0.8205128 0.6410256 Fold01.Rep3  Tree
## 22 0.7784810 0.5569620 Fold02.Rep3  Tree
## 23 0.7452229 0.4903425 Fold03.Rep3  Tree
## 24 0.7324841 0.4658999 Fold04.Rep3  Tree
## 25 0.7515924 0.5024783 Fold05.Rep3  Tree
## 26 0.7770701 0.5545197 Fold06.Rep3  Tree
## 27 0.7531646 0.5063291 Fold07.Rep3  Tree
## 28 0.7341772 0.4683544 Fold08.Rep3  Tree
## 29 0.7961783 0.5918765 Fold09.Rep3  Tree
## 30 0.7898089 0.5793619 Fold10.Rep3  Tree
## 31 0.7948718 0.5897436 Fold01.Rep4  Tree
## 32 0.7088608 0.4177215 Fold02.Rep4  Tree
## 33 0.8227848 0.6455696 Fold03.Rep4  Tree
## 34 0.8089172 0.6176327 Fold04.Rep4  Tree
## 35 0.8164557 0.6329114 Fold05.Rep4  Tree
## 36 0.7834395 0.5672126 Fold06.Rep4  Tree
## 37 0.7834395 0.5667208 Fold07.Rep4  Tree
## 38 0.7898089 0.5791569 Fold08.Rep4  Tree
## 39 0.7133758 0.4267748 Fold09.Rep4  Tree
## 40 0.7961783 0.5924724 Fold10.Rep4  Tree
##   mAccuracy    mKappa
## 1 0.7760899 0.5521779
##    Accuracy     Kappa model
## 1 0.7568773 0.1141389  Tree
```

XGBoost

```r
labels <- data.matrix(trainSplit$Y)
xgb <- xgboost(data = data.matrix(trainSplit[,-1]),
               label = labels,
               eta = 0.01,
               max_depth = 15,
               nround=25,
               subsample = 0.5,
               colsample_bytree = 0.5,
               eval_metric = "error",
               objective = "binary:logistic",
               nthread = 4,
               verbose = FALSE
)

y_pred <- predict(xgb, data.matrix(test)) # This outputs a vector of probabilites

# Confusion matrix
cm = table(true_Y_test, as.numeric(y_pred > 0.5))
cm
```

```
##
## true_Y_test    0    1
##           1   62   35
##           0 1106  142
```

```r
# Metrics
precision = (cm[1,2]) / (cm[1,2] + cm[2,2])
recall = (cm[1,2]) / (cm[1,2] + cm[0,0])
f1_score = 2 * (precision*recall)/(precision+recall)
f1_score
```

```
## <0 x 0 matrix>
```

Support Vector Machine

```r
classifier = svm(formula = Y ~ .,
                 data = trainSplit,
                 type = 'C-classification',
                 kernel = 'linear')
```

```
## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'Prod_Category_F' constant. Cannot scale data.
```

```r
y_pred = predict(classifier, newdata = test)
```

Evaluation metrics

```r
cm = table(true_Y_test, y_pred)
cm
```

```
##            y_pred
## true_Y_test   1   0
##           1  53  44
##           0 390 858
```

```r
err <- mean(as.numeric(y_pred) != true_Y_test)
print(paste("False Positives + False Negatives=", err))
```

```
## [1] "False Positives + False Negatives= 0.960594795539033"
```

Overall, XGBoost is by far the best technique by far! Support Vector Machine performed awfully bad. It improved the simple Logistic Model and the CART model and we got an accuracy of 88%.