telecom_customer_churn_notebook

2023-01-24

1. Loading Data

```
setwd("/Users/lucasquemelli/Documents/repos/telecom_customer_churn")

set.seed(007)
churn_data_raw = read.csv("/Users/lucasquemelli/Documents/repos/telecom_customer_churn/data/churn.csv")
```

```
#churn_data_raw
```

```
#View(churn_data_raw)
```

2. Helper Functions

Installing Packages

If the packages are not in the system, go ahead and install it and its dependencies. Then open it.

```
### First specify the packages of interest
#packages = c("tidyverse", "utils","mice","dplyr",'lime','keras','ggplot2','ggthemes','tidyquant','rsam

## Now load or install & load all
#package.check <- lapply(
#  packages,
#  FUN = function(x) {
#    if (!require(x, character.only = TRUE)) {
#      install.packages(x, dependencies = TRUE)
#      library(x, character.only = TRUE)
#    }
#  }
#)
```

If the following library list does not open, try again and again and again.

```
library("tidyverse")
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
## v ggplot2 3.4.0      v purrr   1.0.1
## v tibble  3.1.8      v dplyr   1.0.10
## v tidyr   1.2.1      v stringr 1.5.0
```

```
## v readr   2.1.3      v forcats 0.5.2
## -- Conflicts ---------------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library("utils")
library("mice")
```

```
##
## Attaching package: 'mice'
##
## The following object is masked from 'package:stats':
##
##     filter
##
## The following objects are masked from 'package:base':
##
##     cbind, rbind
```

```
library("dplyr")
library("lime")
```

```
##
## Attaching package: 'lime'
##
## The following object is masked from 'package:dplyr':
##
##     explain
```

```
library("keras")
library("ggplot2")
library("ggthemes")
library("tidyquant")
```

```
## Loading required package: lubridate
## Loading required package: timechange
##
## Attaching package: 'lubridate'
##
## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
##
## Loading required package: PerformanceAnalytics
## Loading required package: xts
## Loading required package: zoo
##
## Attaching package: 'zoo'
##
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

```
##
##
## Attaching package: 'xts'
##
## The following objects are masked from 'package:dplyr':
##
##     first, last
##
##
## Attaching package: 'PerformanceAnalytics'
##
## The following object is masked from 'package:graphics':
##
##     legend
##
## Loading required package: quantmod
## Loading required package: TTR
## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo
```

```
library("rsample")
library("recipes")
```

```
##
## Attaching package: 'recipes'
##
## The following object is masked from 'package:stringr':
##
##     fixed
##
## The following object is masked from 'package:stats':
##
##     step
```

```
library("yardstick")
```

```
## For binary classification, the first factor level is assumed to be the event.
## Use the argument `event_level = "second"` to alter this as needed.
##
## Attaching package: 'yardstick'
##
## The following object is masked from 'package:keras':
##
##     get_weights
##
## The following object is masked from 'package:readr':
##
##     spec
```

```
library("corrr")
library("readr")
```

```
library("tidyr")
library("phytools")
```

```
## Loading required package: ape
##
## Attaching package: 'ape'
##
## The following object is masked from 'package:rsample':
##
##     complement
##
## Loading required package: maps
##
## Attaching package: 'maps'
##
## The following object is masked from 'package:purrr':
##
##     map
```

```
library("viridis")
```

```
## Loading required package: viridisLite
##
## Attaching package: 'viridis'
##
## The following object is masked from 'package:maps':
##
##     unemp
```

```
#install.packages("tensorflow")
library(tensorflow)
library(reticulate)
#path_to_python <- install_python()
#virtualenv_create("r-reticulate", python = path_to_python)
#install_tensorflow(envname = "r-reticulate")
```

```
#install.packages("keras")
library(keras)
#install_keras(envname = "r-reticulate")
```

```
#install.packages("tibble")
library(tibble)
```

If the following libraries does not open, try again and again and again.

```
#install.packages("MLmetrics")
library(MLmetrics)
```

```
##
## Attaching package: 'MLmetrics'
```

```
## The following object is masked from 'package:base':
##
##     Recall
```

```
library(dplyr)
```

For generating PDF files:

```
#install.packages("tinytex")
#tinytex::install_tinytex()
```

Numerical Variables

```
### Visualizing Numerical  Variables ###
plot_num_var <- function (churn_data_raw){
churn_data_raw %>%
  gather(x, y, -Churn) %>%
  ggplot(aes(x = y, fill = Churn, color = Churn)) +
  facet_wrap(~ x, ncol = 3, scales = "free") +
  geom_density(alpha = 0.5) +
  theme(axis.text.x = element_text(angle = 90, hjust = 1),
        legend.position = "top") +
  scale_color_tableau() +
  scale_fill_tableau()

}
```

Identifying and removing missing values

1. We use the mice package to find out any sort of patterns that exists for the missing data values.

```
### Pre-Processing data
missing_data_snapshot <- function (churn_data_raw){
  churn_data <- churn_data_raw %>%
    select(-customerID)
  mice::md.pattern(churn_data, plot = FALSE)
}

### Dropping those missing values
preprocess_data <- function (churn_data_raw){
  churn_data <- churn_data_raw %>%
    select(-customerID)
  mice::md.pattern(churn_data, plot = FALSE)
  churn_data <- churn_data %>%
    drop_na()
  return(churn_data)
}
```

Training and test datasets split

```
### Creating train data
train_data <- function(churn_data_raw, train_proportion){
  churn_data_tbl <- churn_data_raw %>%
    select(-customerID) %>%
    drop_na() %>%
    select(Churn, everything())
  glimpse(churn_data_tbl)

  # Split test/training sets
  train_test_split <- initial_split(churn_data_tbl, prop = train_proportion)
  train_test_split
  return(train_test_split)
}
```

Categorical Variables

```
### Visualizing Categorical Variables ###
plot_cat_var<- function (churn_data_raw){

  churn_data_raw %>%
    select(-customerID) %>%
    select_if(is.character) %>%
    select(Churn, everything()) %>%
    gather(x, y, gender:PaymentMethod) %>% #select columns from gender to PaymentMethod
    count(Churn, x, y) %>%
    ggplot(aes(x = y, y = n, fill = Churn, color = Churn)) +
    facet_wrap(~ x, ncol = 4, scales = "free") +
    geom_bar(stat = "identity", alpha = 0.5) +
    theme(axis.text.x = element_text(angle = 9, hjust = 1),
          legend.position = "top") +
    scale_color_tableau() +
    scale_fill_tableau()
}
```

Creating Recipe

1. We selected the target variable and the dataset using recipe function.
2. Tenure is a continuous variable. We used step_discretize function in order to make the variable tenure assume discrete values.
3. We have made log transformation into the TotalCharges variable due to its broad range.
4. We made dummies in order to convert categorical variables into binaries 1 and 0, then transforming them to columns. In step_dummy, we selected the categorical variables using all_nominal() and excluded the target variable using -all_outcomes().
5. We created the mean center using step_center function. This step is to normalize our data.

6. We used step_scale function in order to scale our data. Since we have variables in different scales, it is important to scale them in order to have a better performance training the model.

7. Finally, we used prep function to estimate the required parameters from the training set.

```
# Create recipe for performing some pre-processsing steps on the data
create_recipe <- function (train_tbl){
rec_obj <- recipe(Churn ~ ., data = train_tbl) %>%
  step_discretize(tenure, options = list(cuts = 6)) %>%
  step_log(TotalCharges) %>%
  step_dummy(all_nominal(), -all_outcomes()) %>%
  step_center(all_predictors(), -all_outcomes()) %>%
  step_scale(all_predictors(), -all_outcomes()) %>%
  prep(data = train_tbl)
  return(rec_obj)
}
```

3. Exploratory Data Analysis

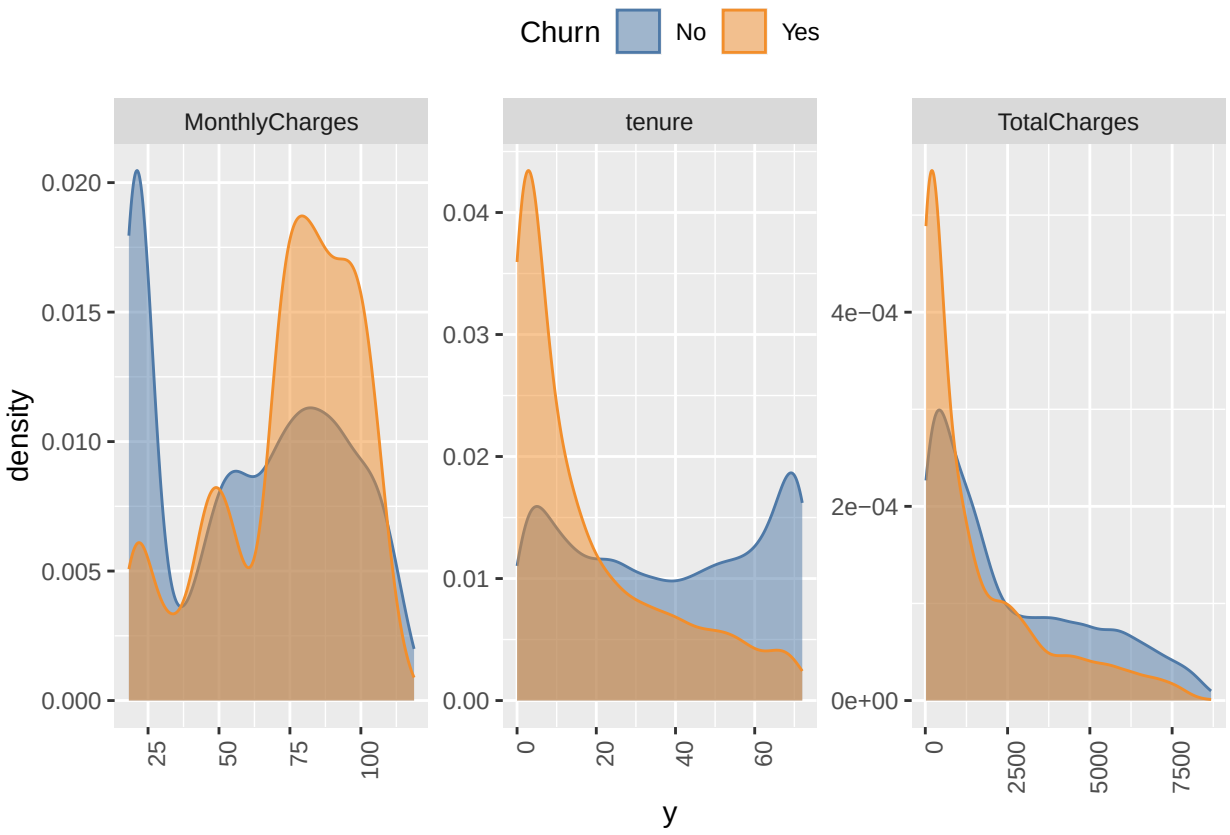
3.1. Numerical variables bivariate analysis

Below, the density plots of the numerical variables.

- Here we select only the numerical variables. We input the column name of each numerical variable.
- Then, we use the function defined in the Helper Function section to plot density plots with the numerical variables selected previously.

```
### Selecting Numerical variables to visualize
num_variables <- c("Churn", "MonthlyCharges", "tenure", "TotalCharges")
num_var_data  <-  churn_data_raw[num_variables]

### Bi-variate analysis

#Visualize the numerical data
plot_num_var(num_var_data)
```

```
## Warning: Removed 11 rows containing non-finite values (‘stat_density()‘).
```



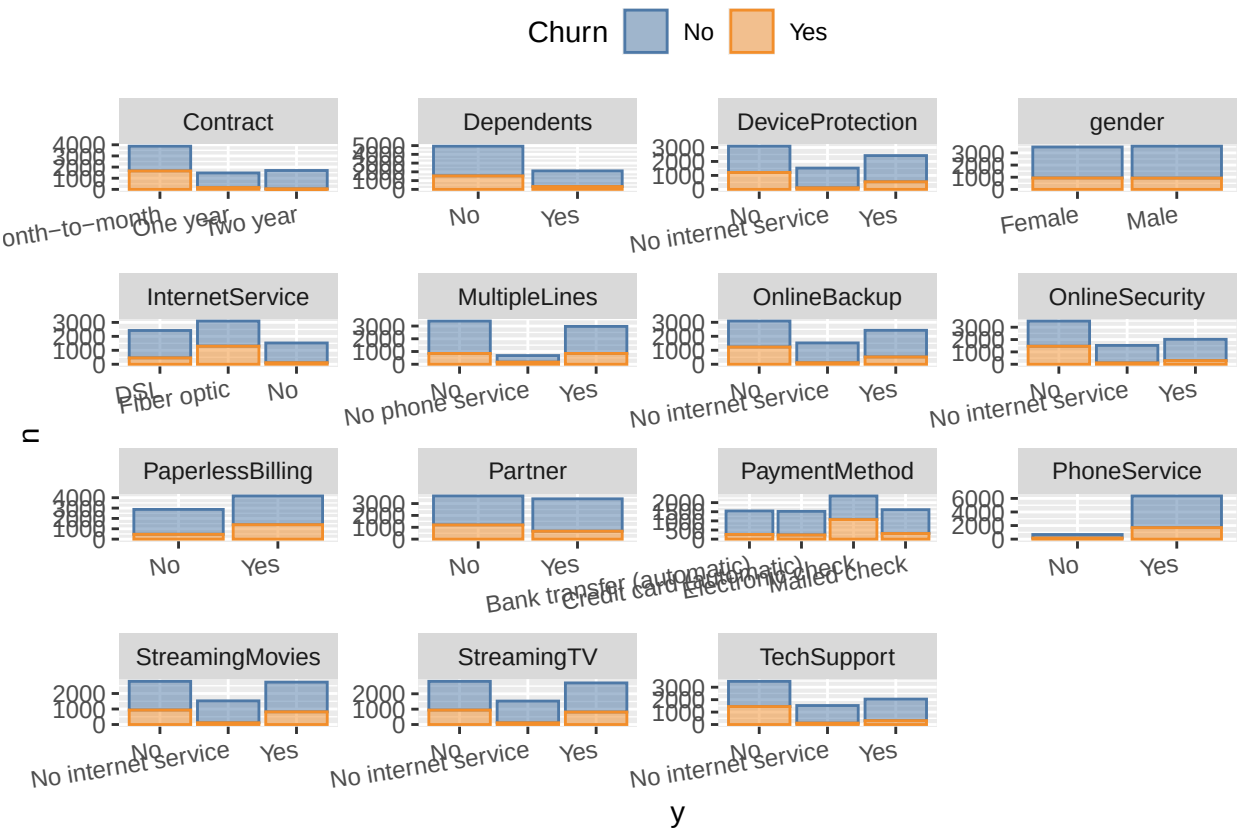
- For higher tenure (months the user has stayed with the company), the less is the churn.
- As the monthly charges (the amount that is charged to the customer every month) increases, the churn also increases.

3.2. Categorical variables bivariate analysis

Below, the stacked histograms of the categorical variables.

- We select only the variables which are of char type.
- Then, we use the function defined in the Helper Functions section to plot the stacked histograms below.

```
#Visualize the categorical data
cat_var_data=churn_data_raw[,sapply(churn_data_raw,is.character)]
plot_cat_var(cat_var_data)
```



- There are more users who does not have dependents and also a higher number of churn.
- Contracts of long periods have lower number of churn.
- Users with has no online security has a higher number of churn.
- Users with paperless billing has a higher number of churn.
- Users with payment method equals to electronic check has a higher number of churn.
- Users with phone service has a higher number of churn.
- Users with no tech support has a higher number of churn.

4. Data Pre-processing

4.1 Missing values check

1. Firstly, we used the missing_data_snapshot function.
2. Since we have found only 11 rows with missing values for 7032 in total, we chose remove all of them.

```
# Missing values snapshot
missing_data_snapshot(churn_data_raw)
```

```
##      gender SeniorCitizen Partner Dependents tenure PhoneService MultipleLines
## 7032      1             1       1          1      1            1             1
## 11        1             1       1          1      1            1             1
##           0             0       0          0      0            0             0
##      InternetService OnlineSecurity OnlineBackup DeviceProtection TechSupport
## 7032               1              1            1                 1           1
## 11                 1              1            1                 1           1
##                    0              0            0                 0           0
##      StreamingTV StreamingMovies Contract PaperlessBilling PaymentMethod
## 7032           1               1        1                 1             1
## 11             1               1        1                 1             1
##                0               0        0                 0             0
##      MonthlyCharges Churn TotalCharges
## 7032              1     1            1  0
## 11                1     1            0  1
##                   0     0           11 11
```

```
# Missing values removal
x <- preprocess_data(churn_data_raw)
```

- Only TotalCharges has missing values. The number of missing values is 11.

Since the percentage of missing values is very low, we removed all of them.

```
# Percentage of missing values
100*11/7032
```

```
## [1] 0.1564278
```

4.2 Training and test split

We split the dataset into training and test and then created a recipe using the functions in the Helper Functions section.

```
### Creating train train split with 80% data in train
train_test_split <- train_data(churn_data_raw, .8) ### Input data & % of data for training
```

```
## Rows: 7,032
## Columns: 20
## $ Churn            <chr> "No", "No", "Yes", "No", "Yes", "Yes", "No", "No", "Y~
## $ gender           <chr> "Female", "Male", "Male", "Male", "Female", "Female",~
## $ SeniorCitizen    <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,~
## $ Partner          <chr> "Yes", "No", "No", "No", "No", "No", "No", "No", "Yes~
## $ Dependents       <chr> "No", "No", "No", "No", "No", "No", "Yes", "No", "No"~
## $ tenure           <int> 1, 34, 2, 45, 2, 8, 22, 10, 28, 62, 13, 16, 58, 49, 2~
## $ PhoneService     <chr> "No", "Yes", "Yes", "No", "Yes", "Yes", "Yes", "No", ~
## $ MultipleLines    <chr> "No phone service", "No", "No", "No phone service", "~
## $ InternetService  <chr> "DSL", "DSL", "DSL", "DSL", "Fiber optic", "Fiber opt~
## $ OnlineSecurity   <chr> "No", "Yes", "Yes", "Yes", "No", "No", "No", "Yes", "~
## $ OnlineBackup     <chr> "Yes", "No", "Yes", "No", "No", "No", "Yes", "No", "N~
## $ DeviceProtection <chr> "No", "Yes", "No", "Yes", "No", "Yes", "No", "No", "Y~
```

```
## $ TechSupport      <chr> "No", "No", "No", "Yes", "No", "No", "No", "No", "Yes~
## $ StreamingTV      <chr> "No", "No", "No", "No", "No", "Yes", "Yes", "No", "Ye~
## $ StreamingMovies  <chr> "No", "No", "No", "No", "No", "Yes", "No", "No", "Yes~
## $ Contract         <chr> "Month-to-month", "One year", "Month-to-month", "One ~
## $ PaperlessBilling <chr> "Yes", "No", "Yes", "No", "Yes", "Yes", "Yes", "No", ~
## $ PaymentMethod    <chr> "Electronic check", "Mailed check", "Mailed check", "~
## $ MonthlyCharges   <dbl> 29.85, 56.95, 53.85, 42.30, 70.70, 99.65, 89.10, 29.7~
## $ TotalCharges     <dbl> 29.85, 1889.50, 108.15, 1840.75, 151.65, 820.50, 1949~
```

```
# Partition train and test dataset
train_tbl <- training(train_test_split)
test_tbl  <- testing(train_test_split)

### Creating Recipe Object
rec_obj<- create_recipe(train_tbl)
rec_obj
```

```
## Recipe
##
## Inputs:
##
##       role #variables
##    outcome          1
##  predictor         19
##
## Training data contained 5625 data points and no missing data.
##
## Operations:
##
## Discretize numeric variables from tenure [trained]
## Log transformation on TotalCharges [trained]
## Dummy variables from gender, Partner, Dependents, tenure, PhoneService, Multip... [trained]
## Centering for SeniorCitizen, MonthlyCharges, TotalCharges, ge... [trained]
## Scaling for SeniorCitizen, MonthlyCharges, TotalCharges, ge... [trained]
```

We baked the recipe by creating our relevant test and training data for the predictor variables, based on the transformations made on the recipe object.

1. We used the bake function to the with train_tbl the same the is found in the recipe object. Then, we excluded the target variable.
2. Sequentially, we also used bake for the test dataset.
3. glimpse() is like a transposed version of print() : columns run down the page, and data runs across.

```
### Baking the recipes
# Predictors
x_train_tbl <- bake(rec_obj, new_data = as.data.frame(train_tbl)) %>% select(-Churn)
x_test_tbl  <- bake(rec_obj, new_data = test_tbl) %>% select(-Churn)
glimpse(x_train_tbl)
```

```
## Rows: 5,625
## Columns: 34
```

```
## $ SeniorCitizen                          <dbl> -0.4366856, -0.4366856, -0.43668~
## $ MonthlyCharges                         <dbl> 1.10773580, 0.54279506, -1.48102~
## $ TotalCharges                           <dbl> 1.01434490, 0.60325854, -1.44858~
## $ gender_Male                            <dbl> -1.0108140, 0.9891258, 0.9891258~
## $ Partner_Yes                            <dbl> 1.0374105, -0.9637672, -0.963767~
## $ Dependents_Yes                         <dbl> 1.5387795, -0.6497501, -0.649750~
## $ tenure_bin2                            <dbl> -0.4401433, -0.4401433, 2.271583~
## $ tenure_bin3                            <dbl> -0.4541677, -0.4541677, -0.45416~
## $ tenure_bin4                            <dbl> -0.4478891, 2.2322985, -0.447889~
## $ tenure_bin5                            <dbl> 2.2209609, -0.4501755, -0.450175~
## $ tenure_bin6                            <dbl> -0.4337966, -0.4337966, -0.43379~
## $ PhoneService_Yes                       <dbl> 0.3275124, 0.3275124, 0.3275124,~
## $ MultipleLines_No.phone.service         <dbl> -0.3275124, -0.3275124, -0.32751~
## $ MultipleLines_Yes                      <dbl> 1.1684136, 1.1684136, -0.8557091~
## $ InternetService_Fiber.optic            <dbl> 1.1239915, 1.1239915, -0.8895283~
## $ InternetService_No                     <dbl> -0.5273226, -0.5273226, 1.896035~
## $ OnlineSecurity_No.internet.service     <dbl> -0.5273226, -0.5273226, 1.896035~
## $ OnlineSecurity_Yes                     <dbl> 1.5701483, -0.6367693, -0.636769~
## $ OnlineBackup_No.internet.service       <dbl> -0.5273226, -0.5273226, 1.896035~
## $ OnlineBackup_Yes                       <dbl> 1.3764704, 1.3764704, -0.7263667~
## $ DeviceProtection_No.internet.service   <dbl> -0.5273226, -0.5273226, 1.896035~
## $ DeviceProtection_Yes                   <dbl> 1.384080, -0.722373, -0.722373, ~
## $ TechSupport_No.internet.service        <dbl> -0.5273226, -0.5273226, 1.896035~
## $ TechSupport_Yes                        <dbl> -0.6375966, -0.6375966, -0.63759~
## $ StreamingTV_No.internet.service        <dbl> -0.5273226, -0.5273226, 1.896035~
## $ StreamingTV_Yes                        <dbl> -0.7862121, -0.7862121, -0.78621~
## $ StreamingMovies_No.internet.service    <dbl> -0.5273226, -0.5273226, 1.896035~
## $ StreamingMovies_Yes                    <dbl> 1.2452890, -0.8028837, -0.802883~
## $ Contract_One.year                      <dbl> -0.5127017, -0.5127017, -0.51270~
## $ Contract_Two.year                      <dbl> -0.5629968, -0.5629968, -0.56299~
## $ PaperlessBilling_Yes                   <dbl> -1.2084578, 0.8273539, -1.208457~
## $ PaymentMethod_Credit.card..automatic. <dbl> -0.5237419, -0.5237419, -0.52374~
## $ PaymentMethod_Electronic.check         <dbl> -0.7132752, 1.4017342, -0.713275~
## $ PaymentMethod_Mailed.check             <dbl> -0.5421657, -0.5421657, 1.844126~
```

We also prepared the target variable for training and test datasets. For the both datasets, what we did means: if the variable Churn is equal to "Yes", then attribute to it the valor 1, else 0.

```
# Response variables for training and testing sets
y_train_vec <- ifelse(pull(train_tbl, Churn) == "Yes", 1, 0)
y_test_vec  <- ifelse(pull(test_tbl, Churn) == "Yes", 1, 0)
glimpse(y_train_vec)
```

```
##  num [1:5625] 0 0 0 0 1 1 0 0 0 0 ...
```

5. Machine Learning Modelling

5.1. Training the model

We trained a neural network MLP (Multi-Layer Perceptron), where:

- We created (1) the first layer, (2) two hidden layers and (3) the last layer.

- For the first layer, the number of neurons is the next 2^n the is higher than the number of the train set (x_train_tbl) dimensions. As the number of dimensions for x_train_tbl is 20, the next 2^n that is higher than 20 is 2^5 = 32.
- The following hidden layers have the number of neurons equals to the half of the previous layer.
- The number of neurons for the last layer is the number of dimensions for the response variable. In this case, it is 1.
- kernel_initializer is the initialization schemes that create the layer's weights. This initializer generates tensors with a uniform distribution.
- The rectified linear activation function or ReLU is a linear function that will output the input directly if it is positive, otherwise, it will output zero. It has become the default activation function for many types of neural networks because a model that uses it is easier to train and often achieves better performance. It is ideal for the first layers.
- We used sigmoid for the last layer since it is a classification problem and it is a binary-class output type.
- Dropout consists in randomly setting a fraction rate of input units to a valor at each update during training time, which helps prevent overfitting. The rate is a float between 0 and 1; it is the fraction of the input units to drop.
- Finally, the optimizer is adamax (ideal for classification problems) and we are looking at binary_accuracy since we have a binary-class output type.

```
### Creating sequential model using Keras
dev_keras_model<- function(x_train_tbl){
  model_keras <- keras_model_sequential()

  model_keras %>%
    layer_dense(units = 32, kernel_initializer = "uniform", activation = "relu",
                input_shape = ncol(x_train_tbl)) %>%
    layer_dropout(rate = 0.2) %>%

    layer_dense(units = 16, kernel_initializer = "uniform", activation = "relu") %>%
    layer_dropout(rate = 0.2) %>%

    layer_dense(units = 8, kernel_initializer = "uniform", activation = "relu") %>%
    layer_dropout(rate = 0.2) %>%

    layer_dense(units = 1,
                kernel_initializer = "uniform", activation = "sigmoid") %>%

    compile(
      optimizer = 'adamax',
      loss      = 'binary_crossentropy',
      metrics   = c("binary_accuracy", "mse")
    )

  return(model_keras)

}
```

```
#######  Building Keras Model#######
model_keras <- dev_keras_model(x_train_tbl)

### Viewing the model layers
model_keras
```

```
## Model: "sequential"
## _________________________________________________________________________
##  Layer (type)                       Output Shape                    Param #
## =========================================================================
##  dense_3 (Dense)                    (None, 32)                      1120
##  dropout_2 (Dropout)                (None, 32)                      0
##  dense_2 (Dense)                    (None, 16)                      528
##  dropout_1 (Dropout)                (None, 16)                      0
##  dense_1 (Dense)                    (None, 8)                       136
##  dropout (Dropout)                  (None, 8)                       0
##  dense (Dense)                      (None, 1)                       9
## =========================================================================
## Total params: 1,793
## Trainable params: 1,793
## Non-trainable params: 0
## _________________________________________________________________________
```

5.2. Model Fitting

`validation_split()` takes a single random sample (without replacement) of the original data set to be used for analysis. All other data points are added to the assessment set (to be used as the validation set). This is a manner of using different portions of our dataset in order to achieve a good variability during fitting (similar to Cross-Validation).

```
fit_model <- function(x_train_tbl, y_train_vec){
  history <- fit(
    object           = model_keras,
    x                = as.matrix(x_train_tbl),
    y                = y_train_vec,
    batch_size       = 32,
    epochs           = 35,
    validation_split = 0.30

  )

  return(history)
}
```

- The loss is the error of the model.
- The binnary_accuracy is how accurate is the model. Accuracy is calculated by the ratio of number of correct predicted values over total number of samples.
- MSE: mean squared error.
- val_loss: error in the validation dataset. The same is for val_binary_accuracy and val_mse.

- Epochs is the number of iterations. For each iteration, we may see the model performance. After a number o iterations, we may see the performance getting stable.

```
### Model Fitting & Performance
fit_model(x_train_tbl, y_train_vec)
```

```
##
## Final epoch (plot to see history):
##                 loss: 0.4241
##     binary_accuracy: 0.8128
##                  mse: 0.1381
##             val_loss: 0.4458
## val_binary_accuracy: 0.798
##              val_mse: 0.145
```

A binary accuracy of 81% is a very good performance for a classification model. Since the validation binary accuracy is close to this value (79%), we may assume that there is no over and underfitting.

Furthermore, associated with the results above, we may see the stabilization of the model performance after the epoch number 10.

5.3. Model Performance

To use probability, we stablish a threshold as follows

probability > threshold -> 1

probability < threshold <- 0

To Machine Learning, this threshold is commonly 0.5%. Thus, we are assuming threshold of 0.5.

- If your model does multi-class classification (e.g. if it uses a `softmax` last-layer activation):

```
model %\>% predict(x) %\>% k_argmax()
```

- if your model does binary classification (e.g. if it uses a `sigmoid` last-layer activation):

```
model %\>% predict(x) %\>% \‘\>\‘(0.5) %\>% k_cast("int32")
```

```
### Predictions of the model
predict_estimates <- function(model_keras, x_test_tbl, y_test_vec){
  # Predicted Class
  yhat_keras_class_vec <- predict(object = model_keras, x = as.matrix(x_test_tbl)) %>% `>`(0.5) %>% k_ca
  as.vector()
  #yhat_keras_class_vec <- model_keras %>% predict(x = as.matrix(x_test_tbl)) %>% `>`(0.5) %>% k_cast("
    #as.vector()

  # Predicted Class Probability
  yhat_keras_prob_vec  <- predict(object = model_keras, x = as.matrix(x_test_tbl)) %>% `>`(0.5) %>% k_ca
  as.vector()

  # Format test data and predictions for yardstick metrics
  estimates_keras_tbl <- tibble(
```

```
    truth      = factor(y_test_vec, levels = c(0, 1)),
    estimate   = factor(yhat_keras_class_vec, levels = c(0, 1)),
    class_prob = yhat_keras_prob_vec

  )

  estimates_keras_tbl
}
```

```
### Predictions from model on test data
estimates_keras_tbl <- predict_estimates(model_keras, x_test_tbl, y_test_vec)
```

```
### Judging model benchmark metrics
# Confusion Table
estimates_keras_tbl %>% conf_mat(truth, estimate)
```

```
##            Truth
## Prediction   0   1
##          0 939 177
##          1 106 185
```

The model was able to predict 919 true negative (no) and 205 true positive (yes) values. These values are in the main diagonal.

```
# AUC/ROC
estimates_keras_tbl %>% roc_auc(truth, class_prob)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc binary         0.295
```

Recall shows us how our positive values (churn) are corrected predicted.

```
# Recall for positive label
yhat_keras_class_vec <- predict(object = model_keras, x = as.matrix(x_test_tbl)) %>% `>`(0.5) %>% k_cas
  as.vector()

Recall(y_test_vec, ifelse(yhat_keras_class_vec > .5, 1, 0), positive = 1)
```

```
## [1] 0.5110497
```

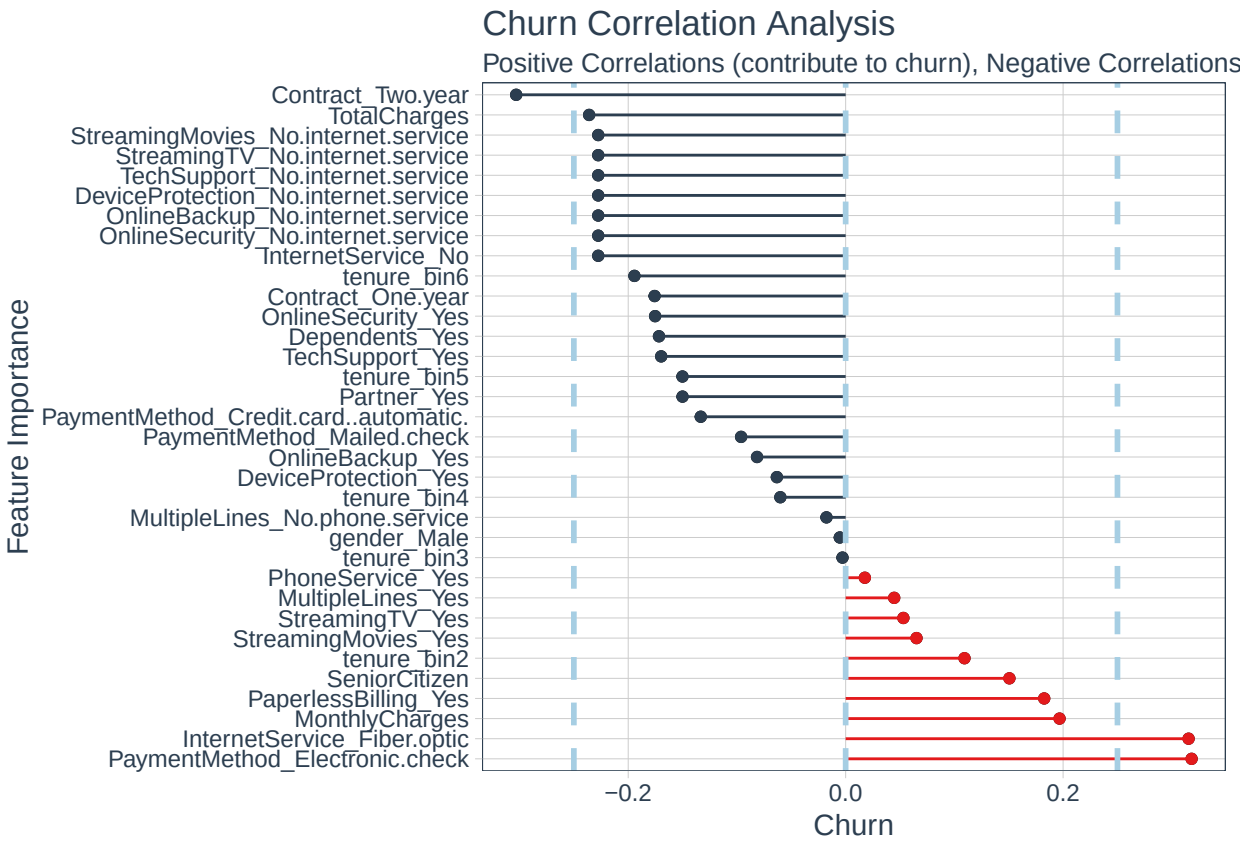
```
#F1-Score
estimates_keras_tbl %>% f_meas(truth, estimate, beta = 1)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 f_meas  binary         0.869
```

```
### Correlation Analysis ###
coorelation_analysis <- function (x_train_tbl,y_train_vec){


  # Feature correlations to Churn
  corrr_analysis <- x_train_tbl %>%
    mutate(Churn = y_train_vec) %>%
    correlate() %>%
    focus(Churn) %>%
    mutate(feature = colnames(x_train_tbl)) %>%
    arrange(abs(Churn)) %>%
    mutate(feature = as_factor(feature))
  corrr_analysis

  # Correlation visualization
  corrr_analysis %>%
    ggplot(aes(x = Churn, y = fct_reorder(feature, desc(Churn)))) +
    geom_point() +
    # Positive Correlations - Contribute to churn
    geom_segment(aes(xend = 0, yend = feature),
                 color = palette_light()[[2]],
                 data = corrr_analysis %>% filter(Churn > 0)) +
    geom_point(color = palette_light()[[2]],
               data = corrr_analysis %>% filter(Churn > 0)) +
    # Negative Correlations - Prevent churn
    geom_segment(aes(xend = 0, yend = feature),
                 color = palette_light()[[1]],
                 data = corrr_analysis %>% filter(Churn < 0)) +
    geom_point(color = palette_light()[[1]],
               data = corrr_analysis %>% filter(Churn < 0)) +
    # Vertical lines
    geom_vline(xintercept = 0, color = palette_light()[[5]], size = 1, linetype = 2) +
    geom_vline(xintercept = -0.25, color = palette_light()[[5]], size = 1, linetype = 2) +
    geom_vline(xintercept = 0.25, color = palette_light()[[5]], size = 1, linetype = 2) +
    # Aesthetics
    theme_tq() +
    labs(title = "Churn Correlation Analysis",
         subtitle = paste("Positive Correlations (contribute to churn),",
                          "Negative Correlations (prevent churn)"),
         y = "Feature Importance")
}
```

```
### Correlation Insights
coorelation_analysis(x_train_tbl,y_train_vec)
```

```
## Correlation computed with
## * Method: 'pearson'
## * Missing treated using: 'pairwise.complete.obs'

## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use `linewidth` instead.
```



Considering a value of correlation to churn equals or higher than the absolute of 0.2.

- Variables from Contract_Two.year until tenure_bin6 contribute very much to the model and users with this characteristics are less likely to churn (negative correlation values <= -0.2).
- The variables PaymentMethod_Electronic.check, InternetService_Fiber.optic and MonthlyCharges also contributes very much to the model - and they are more likely to churn (positive correlation values >= 0.2).

By using these insights, the business team may decide how characteristics must aim for preventing churn.

6. Deployment

```
# save the model in desired location
save_model_hdf5(model_keras, 'user_churn_model.hdf5')
```

```
# Load model
load_model_hdf5("/Users/lucasquemelli/Documents/repos/telecom_customer_churn/user_churn_model.hdf5", cus
```

```
## Model: "sequential"
## _________________________________________________________________________
##  Layer (type)                       Output Shape                    Param #
## =========================================================================
```

```
##  dense_3 (Dense)                  (None, 32)                  1120
##  dropout_2 (Dropout)              (None, 32)                  0
##  dense_2 (Dense)                  (None, 16)                  528
##  dropout_1 (Dropout)              (None, 16)                  0
##  dense_1 (Dense)                  (None, 8)                   136
##  dropout (Dropout)                (None, 8)                   0
##  dense (Dense)                    (None, 1)                   9
## =================================================================
## Total params: 1,793
## Trainable params: 1,793
## Non-trainable params: 0
## _________________________________________________________________
```