# Customer Churn Prediction

November 14, 2021

## 1 Customer Churn Prediction

### 1.1 FIT5149 Assignment1

**Student Name: Madeleine Chalmers**

**Student ID: 31222846** Date: 26 October 2021

### 1.2 Table of Contents

### 1.3 1. Introduction

The aim here is to build a statistical model which will accurately predict when an insurance company is about to lose their customer. There are sixteen anonymized features in the dataset and one response variable (named 'labels') which indicates churn (1) or no churn (0) for each observation. Two datasets are provided for training (trainSet.csv) and testing (testSet.csv) the models.

Data Source: https://www.kaggle.com/mukulsingh/insurance-churn-prediction

Load the required libraries:

```
[1]: library(psych)
library(ggplot2)
library(reshape2)
library(OneR)
library(RColorBrewer)
library(corrplot)
library(gridExtra)
library(tree)
library(rpart)
library(iml)
library(boot)
library(MASS)
library(glmnet)
library(ROCR)
library(caret)
```

```r
library(stats)
library(vcd)
library(car)
```

```

Attaching package: 'ggplot2'


The following objects are masked from 'package:psych':

    %+%, alpha


Warning message:
"package 'reshape2' was built under R version 3.6.3"
Warning message:
"package 'OneR' was built under R version 3.6.3"
corrplot 0.90 loaded

Warning message:
"package 'gridExtra' was built under R version 3.6.3"
Warning message:
"package 'iml' was built under R version 3.6.3"

Attaching package: 'boot'


The following object is masked from 'package:psych':

    logit


Warning message:
"package 'glmnet' was built under R version 3.6.3"
Loading required package: Matrix

Loaded glmnet 4.1-1

Warning message:
"package 'ROCR' was built under R version 3.6.3"
Loading required package: lattice


Attaching package: 'lattice'


The following object is masked from 'package:boot':
    melanoma


Loading required package: grid

Loading required package: carData

Warning message:
"package 'carData' was built under R version 3.6.3"

Attaching package: 'car'


The following object is masked from 'package:boot':

    logit


The following object is masked from 'package:psych':

    logit
```

package installations:

[2]:
```
#install.packages("psych")
#install.packages("reshape2")
#install.packages("OneR")
#install.packages("ggplot2")
#install.packages("tidyverse")
#install.packages("corrplot")
#install.packages("gridExtra")
#install.packages("glmnet")
#install.packages("tree")
#install.packages("rpart")
#install.packages("iml")
#install.packages("ROCR")
#install.packages("car")
#install.packages("vcd")
```

Read in the test and train datasets that were provided.

[3]:
```
# load test dataset
test <- read.csv(file = 'testSet.csv',stringsAsFactors = TRUE)

# load train dataset
```

```
train <-  read.csv(file = 'trainSet.csv',stringsAsFactors = TRUE)
```

### 1.4 2. Exploatory Data Analysis

**Using Exploratory Data Analysis (EDA) we will:**

1. Explore individual variables, their characteristics such as spread, variance, distribution etc.
2. Identify important features which have a relationship with our label, and how they interact.
3. Explore the qualities of the relationships to identify which models will give strongest results, and how we can transform the data to aid with learning.
4. Identify features with no relationship and exclude them from our models, for improved efficiency and reduced over fitting.
5. Identify features which strongly correlate with one another, to eliminate duplicate information.
6. Build effective models which are supported by our exploration.

First look at the data

[4]:
```
# show the first rows
head(train)
```

A data.frame: 6 × 17

| | feature_0 <dbl> | feature_1 <dbl> | feature_2 <dbl> | feature_3 <dbl> | feature_4 <dbl> | feature_5 <dbl> | feature <dbl> |
|---|---|---|---|---|---|---|---|
| 1 | 1.5127910 | -0.2434605 | 0.1434182 | 2.01858846 | 0.07622994 | -0.4114531 | -0.2519 |
| 2 | -1.5007763 | -0.2125875 | 1.2248391 | -0.15984112 | -0.56935064 | -0.4114531 | -0.2519 |
| 3 | 0.9477471 | 0.5812426 | -0.3372133 | 0.77987360 | -0.56935064 | -0.4114531 | -0.2519 |
| 4 | -0.8415585 | -0.2217837 | 0.5038918 | -0.37729577 | 0.39902023 | -0.4114531 | -0.2519 |
| 5 | -0.5590365 | -0.5922597 | -1.1783185 | -0.41612696 | -0.56935064 | -0.4114531 | -0.2519 |
| 6 | 0.9477471 | -0.4654833 | -0.5775291 | 0.08867847 | -0.24656035 | 2.2551433 | 2.3528 |

Check the data types.

[5]:
```
# show training the data structure
str(train)
```

```
'data.frame':   27126 obs. of  17 variables:
 $ feature_0 : num  1.513 -1.501 0.948 -0.842 -0.559 …
 $ feature_1 : num  -0.243 -0.213 0.581 -0.222 -0.592 …
 $ feature_2 : num  0.143 1.225 -0.337 0.504 -1.178 …
 $ feature_3 : num  2.019 -0.16 0.78 -0.377 -0.416 …
 $ feature_4 : num  0.0762 -0.5694 -0.5694 0.399 -0.5694 …
 $ feature_5 : num  -0.411 -0.411 -0.411 -0.411 -0.411 …
 $ feature_6 : num  -0.252 -0.252 -0.252 -0.252 -0.252 …
 $ feature_7 : int  1 8 0 9 1 4 7 6 6 9 …
 $ feature_8 : int  1 2 2 1 2 2 1 1 1 1 …
 $ feature_9 : int  1 1 1 1 1 2 1 2 1 3 …
 $ feature_10: int  0 0 0 0 0 1 0 0 0 0 …
 $ feature_11: int  1 0 0 0 1 1 0 1 1 1 …
 $ feature_12: int  0 0 0 0 0 0 0 1 1 0 …
 $ feature_13: int  0 0 2 0 0 0 0 2 0 0 …
 $ feature_14: int  0 8 8 1 8 8 1 6 5 9 …
 $ feature_15: int  3 3 3 3 3 0 3 3 3 3 …
 $ labels    : int  0 0 0 0 0 0 0 0 0 0 …
```

We want to be certain that the two test and train datasets are compatible with one another. Do they have the same data types? The same features in each?

```
[6]: # Show the testing data structure
     str(test)
```

```
'data.frame':	6782 obs. of  17 variables:
 $ feature_0 : num  -1.03 -0.842 -1.501 -0.277 -0.936 …
 $ feature_1 : num  -0.222 -0.356 -0.265 -0.402 -0.268 …
 $ feature_2 : num  0.504 -1.539 -1.419 1.826 1.585 …
 $ feature_3 : num  -0.9054 -0.1055 0.3993 -0.8976 0.0537 …
 $ feature_4 : num  0.0762 2.9813 -0.5694 6.2092 -0.5694 …
 $ feature_5 : num  -0.411 -0.411 -0.411 -0.411 -0.411 …
 $ feature_6 : num  -0.252 -0.252 -0.252 -0.252 -0.252 …
 $ feature_7 : int  0 0 7 0 0 7 5 9 4 1 …
 $ feature_8 : int  1 1 2 1 2 2 1 1 1 1 …
 $ feature_9 : int  1 1 1 1 1 1 1 1 2 1 …
 $ feature_10: int  0 0 0 0 0 0 0 0 0 0 …
 $ feature_11: int  1 1 1 1 1 1 0 1 1 1 …
 $ feature_12: int  0 0 1 0 0 1 0 0 0 1 …
 $ feature_13: int  2 2 2 0 0 2 0 2 0 2 …
 $ feature_14: int  6 6 6 5 4 8 1 8 3 8 …
 $ feature_15: int  3 3 3 3 3 3 3 3 3 3 …
 $ labels    : int  0 0 0 0 0 0 0 0 0 0 …
```

We know that our response variable 'labels' is a factor. It appears that feature 7-15 probably are as well, but we want to try and understand for certain which are numeric, and which are categoric, so we might apply the right analysis techniques and comparisons.

Below we show the number of unique values in each feature:

```
[7]: #Show the unique values in the features we think could be qualitative
     print(apply(train,2,function(x) length(unique(x))))
```

```
 feature_0  feature_1  feature_2  feature_3  feature_4  feature_5  feature_6
        77       5901         31       1403         46        499         35
 feature_7  feature_8  feature_9 feature_10 feature_11 feature_12 feature_13
        12          3          4          2          2          2          3
feature_14 feature_15     labels
        12          4          2
```

Since the feature names in this dataset have been de-identified, it's logical to assume that categoric data could have also been numerically encoded, this would prevent analysts being able to infer the information in those features. If this is the case, we would expect to see them represented by a somewhat small set of integer values. From what we have seen so far, it is certainly possible that our integer data seen above could be categoric, as there are a limited number of values in features

7-15. Next, we check what the possible values are. If they are a set of numeric integers from a condensed range, that will further our case for assuming they are categoric.

Interestingly, it also appears that more than just our integer data is discrete, for example, feature 2 despite being numeric, only takes 31 unique values. By plotting them on a number line, we can see the exact nature of features 1-6. We'll take a closer look after assessing the categoric nature of features 7-14.

```
[8]: #Show the unique values in the features we think could be qualitative
     apply(train[,8:17],2,function(x) sort(unique(x)))
```

**$feature_7** 1. 0 2. 1 3. 2 4. 3 5. 4 6. 5 7. 6 8. 7 9. 8 10. 9 11. 10 12. 11

**$feature_8** 1. 0 2. 1 3. 2

**$feature_9** 1. 0 2. 1 3. 2 4. 3

**$feature_10** 1. 0 2. 1

**$feature_11** 1. 0 2. 1

**$feature_12** 1. 0 2. 1

**$feature_13** 1. 0 2. 1 3. 2

**$feature_14** 1. 0 2. 1 3. 2 4. 3 5. 4 6. 5 7. 6 8. 7 9. 8 10. 9 11. 10 12. 11

**$feature_15** 1. 0 2. 1 3. 2 4. 3

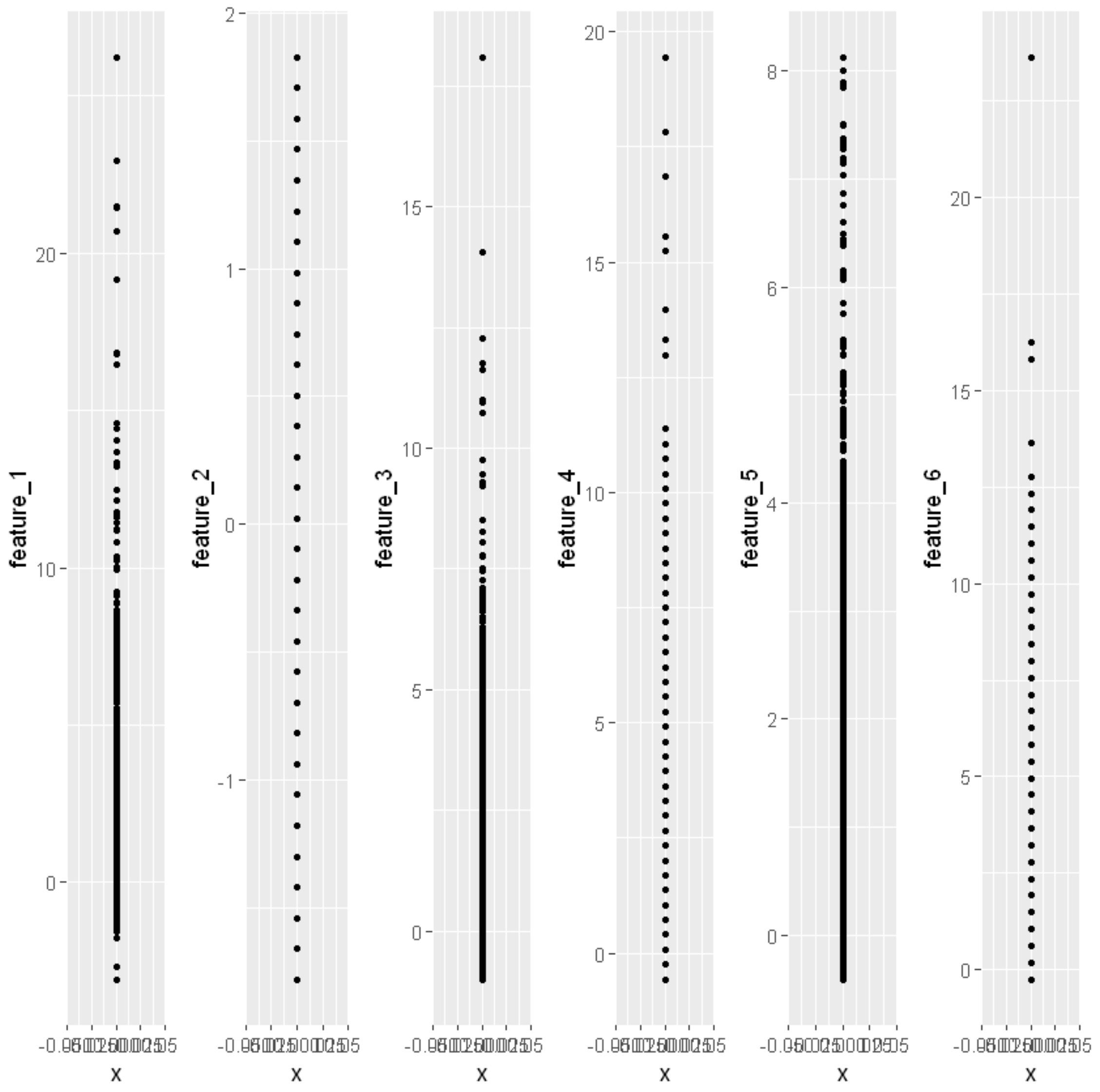
**$labels** 1. 0 2. 1

This is as close as we can get to evidence that these variables are categoric for now, since there is really no wrong way to explore data, it will sometimes be more convenient for us to assume they are categoric. However, we will not change them to factors until we are certain that our models benefit from it.

It doesn't really serve any purpose to know whether features 0-6 are discrete or not, since it is unlikely for categories to be assigned a decimal value. But, for the sake of knowing, we'll check whether they really are discrete.

```
[9]: p0 <- ggplot(train, aes(x=0, y=feature_0)) +
       geom_point(size = 1)

     p1 <- ggplot(train, aes(x=0, y=feature_1)) +
       geom_point(size = 1)

     p2 <- ggplot(train, aes(x=0, y=feature_2)) +
       geom_point(size = 1)

     p3 <- ggplot(train, aes(x=0, y=feature_3)) +
       geom_point(size = 1)

     p4 <- ggplot(train, aes(x=0, y=feature_4)) +
       geom_point(size = 1)
```

```
p5 <- ggplot(train, aes(x=0, y=feature_5)) +
  geom_point(size = 1)

p6 <- ggplot(train, aes(x=0, y=feature_6)) +
  geom_point(size = 1)

grid.arrange(p1, p2, p3, p4, p5, p6, ncol = 6)
```



Yes, it appears that feature 2, 4 and 6 at least are discrete (I'm sure all measurement data is discrete on some level when you look at the lowest resolution though). Knowing all this, we shall proceed with the assumption that features 0-6 are continuous numeric variables and often we will separately explore the categoric nature of features 7-15.

It's also unknown whether the integer variables are a mix of both ordinal and nominal values. For the purpose of exploration, we'll keep them as integers (so we can test correlations etc.) and then we'll change any to factor if we find significant motivations to do so.

Below we change the response variable to factor in both test and train data sets, as we do know that the labels are categoric.

```
[10]: #change label and features 7-15 data type to factor in train dataset
      #train[,8:17] <- data.frame(lapply(train[,8:17], as.factor))
      train$labels <- as.factor(train$labels)

      #change label and features 7-15 data type to factor in test dataset
      #test[,8:17] <- data.frame(lapply(test[,8:17], as.factor))
      test$labels <- as.factor(test$labels)

      #always check your work - looks good.
      str(train)
```

```
'data.frame':   27126 obs. of  17 variables:
 $ feature_0 : num  1.513 -1.501 0.948 -0.842 -0.559 …
 $ feature_1 : num  -0.243 -0.213 0.581 -0.222 -0.592 …
 $ feature_2 : num  0.143 1.225 -0.337 0.504 -1.178 …
 $ feature_3 : num  2.019 -0.16 0.78 -0.377 -0.416 …
 $ feature_4 : num  0.0762 -0.5694 -0.5694 0.399 -0.5694 …
 $ feature_5 : num  -0.411 -0.411 -0.411 -0.411 -0.411 …
 $ feature_6 : num  -0.252 -0.252 -0.252 -0.252 -0.252 …
 $ feature_7 : int  1 8 0 9 1 4 7 6 6 9 …
 $ feature_8 : int  1 2 2 1 2 2 1 1 1 1 …
 $ feature_9 : int  1 1 1 1 1 2 1 2 1 3 …
 $ feature_10: int  0 0 0 0 0 1 0 0 0 0 …
 $ feature_11: int  1 0 0 0 1 1 0 1 1 1 …
 $ feature_12: int  0 0 0 0 0 0 0 1 1 0 …
 $ feature_13: int  0 0 2 0 0 0 0 2 0 0 …
 $ feature_14: int  0 8 8 1 8 8 1 6 5 9 …
 $ feature_15: int  3 3 3 3 3 0 3 3 3 3 …
 $ labels    : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 …
```

What proportion of the data is test versus train?

```
[11]: # get the dimentions of each dataset and return the proportion of data in the␣
       ↪training dataset
      dim(test)[1] / (dim(test)[1] + dim(train)[1])
```

0.200011796626165

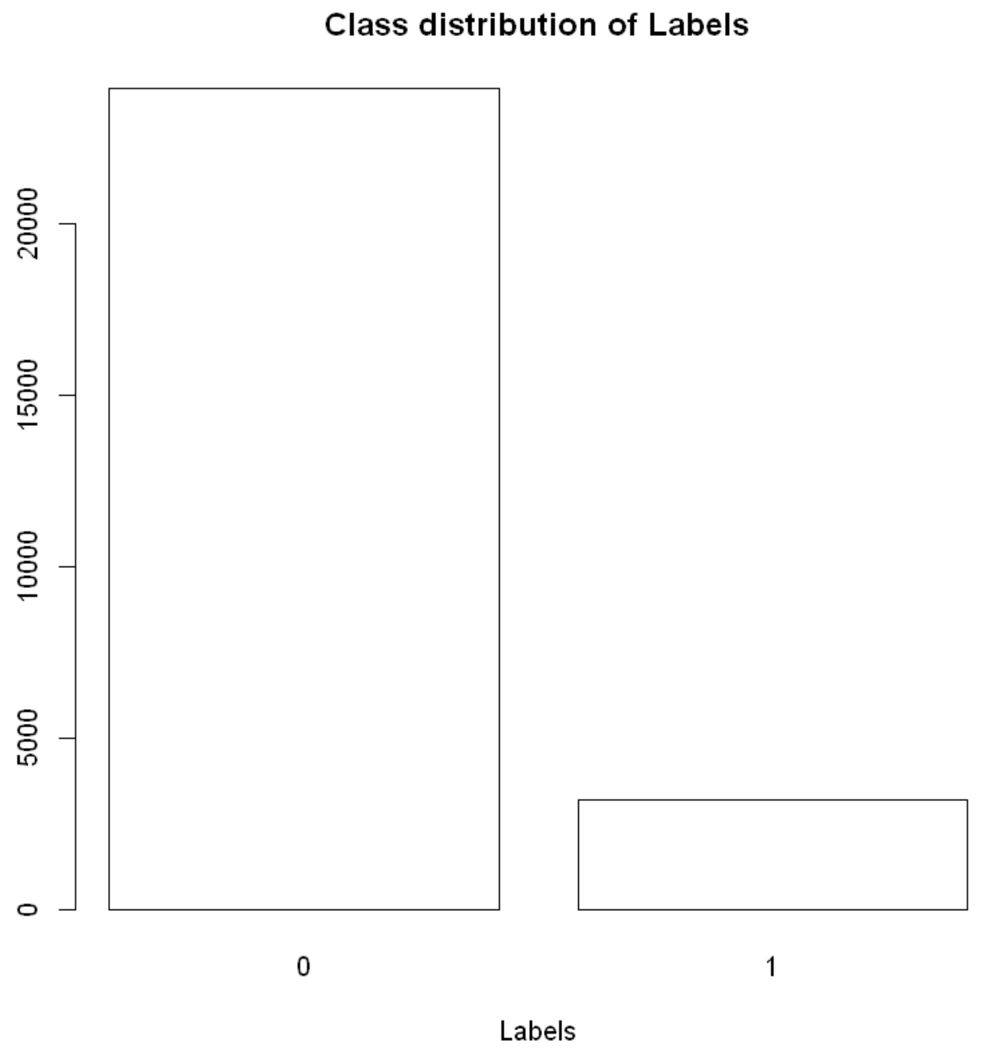
The test dataset makes up one fifth of our total data.

### 1.4.1 2.1. Univariate Analysis

How is the response variable distributed? Is it imbalanced? Is it biased?

```
[12]: # produce a bar chart showing how many observations in our label data have␣
       ↪their class equal to 1 or 0.
      barplot(table(train$labels), xlab = "Labels", main = "Class distribution of␣
       ↪Labels", col = "white")

      # imbalance as a ratio
      pc <- round(table(train$labels)[2] / length(train$labels) * 100, 2)
      cat('Percentage rate of churn:', pc, '%')
```

```
Percentage rate of churn: 11.73 %
```



There is no **detectable** bias in the label data, as we do not know any true information about real churn rates compared to the rates observed in this dataset. However, the class distribution in the label is very imbalanced (regardless of bias), this will affect some learning algorithms and/or model assessment metrics later and will need to be factored in.

Detailed descriptive statistics for continuous data:

[13]:
```
# return summary statistics for the train dataset
round(describe(train[,1:16]), 3)
```

A psych: 16 × 13

| | vars<br>&lt;dbl&gt; | n<br>&lt;dbl&gt; | mean<br>&lt;dbl&gt; | sd<br>&lt;dbl&gt; | median<br>&lt;dbl&gt; | trimmed<br>&lt;dbl&gt; | mad<br>&lt;dbl&gt; | min<br>&lt;dbl&gt; | max<br>&lt;dbl |
|---|---|---|---|---|---|---|---|---|---|
| feature_0 | 1 | 27126 | -0.005 | 1.000 | -0.182 | -0.068 | 0.977 | -2.160 | 5.091 |
| feature_1 | 2 | 27126 | 0.001 | 0.986 | -0.297 | -0.193 | 0.223 | -3.081 | 26.22 |
| feature_2 | 3 | 27126 | 0.004 | 1.000 | 0.023 | -0.010 | 1.247 | -1.779 | 1.826 |
| feature_3 | 4 | 27126 | -0.002 | 0.997 | -0.307 | -0.186 | 0.530 | -1.002 | 18.09 |
| feature_4 | 5 | 27126 | 0.000 | 1.009 | -0.247 | -0.210 | 0.479 | -0.569 | 19.44 |
| feature_5 | 6 | 27126 | -0.003 | 0.997 | -0.411 | -0.285 | 0.000 | -0.411 | 8.128 |
| feature_6 | 7 | 27126 | -0.009 | 0.795 | -0.252 | -0.197 | 0.000 | -0.252 | 23.62 |
| feature_7 | 8 | 27126 | 4.336 | 3.273 | 4.000 | 4.242 | 4.448 | 0.000 | 11.00 |
| feature_8 | 9 | 27126 | 1.170 | 0.605 | 1.000 | 1.213 | 0.000 | 0.000 | 2.000 |
| feature_9 | 10 | 27126 | 1.226 | 0.749 | 1.000 | 1.231 | 0.000 | 0.000 | 3.000 |
| feature_10 | 11 | 27126 | 0.018 | 0.133 | 0.000 | 0.000 | 0.000 | 0.000 | 1.000 |
| feature_11 | 12 | 27126 | 0.552 | 0.497 | 1.000 | 0.565 | 0.000 | 0.000 | 1.000 |
| feature_12 | 13 | 27126 | 0.159 | 0.366 | 0.000 | 0.074 | 0.000 | 0.000 | 1.000 |
| feature_13 | 14 | 27126 | 0.637 | 0.896 | 0.000 | 0.546 | 0.000 | 0.000 | 2.000 |
| feature_14 | 15 | 27126 | 5.513 | 3.004 | 6.000 | 5.671 | 2.965 | 0.000 | 11.00 |
| feature_15 | 16 | 27126 | 2.562 | 0.987 | 3.000 | 2.828 | 0.000 | 0.000 | 3.000 |

Some initital findings from summary statistics: - **Missing Data:** There is no missing data in any features - **Mean:** The data all appear to have relatively similar means, ranging between values of -1 and 6 - **Median:** The range of median values is slightly wider than the mean, but not by much. - **Standard Deviation** Features 7 & 14 have the highest variance, this indicates they could be very useful features. On the flip side, features with the lowest variance are 12, 11, and 10, we might concider filtering those out of our models during feature selection. - **Skew:** Feature 1 & 6 have highest skew, in the positive direction. - **Range:** Features 1 & 6 have the widest range. - **Kurtosis:** Features 1, 4 & 6 have relatively high kurtosis, making their tails very short. Conversely, quite a few variables have longer tails, measuring no lesser than -1.06 though. - **Scaling:** there has been no feature scaling yet (e.g. min-max scaling), we may need to do this for some types of statistical model - **Expected Data Characteristics:** Given that we don't have any particular knowledge about each feature, we cannot comment on the expected or unexpected characteristics of each variable. Only that our response variable has two classes, 0 and 1 which represent churn and no churn as expected.

Descriptive statistics for discrete data:

[14]:
```
summary(data.frame(lapply(test[,8:17], as.factor)), maxsum = 12)

# table and prop.table information is way to extensive to interpret here
#table(test[,8:16])
```

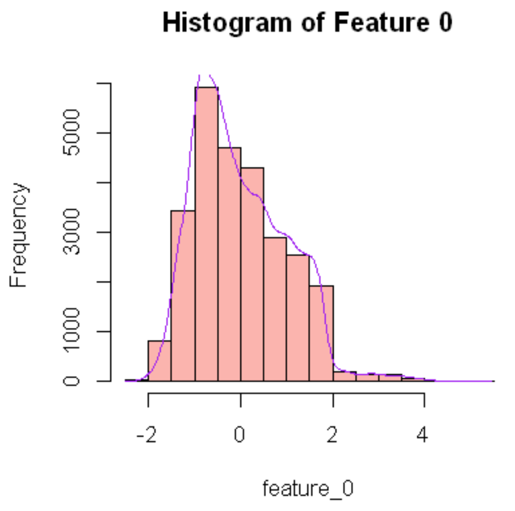
```
 feature_7 feature_8 feature_9 feature_10 feature_11 feature_12 feature_13
```

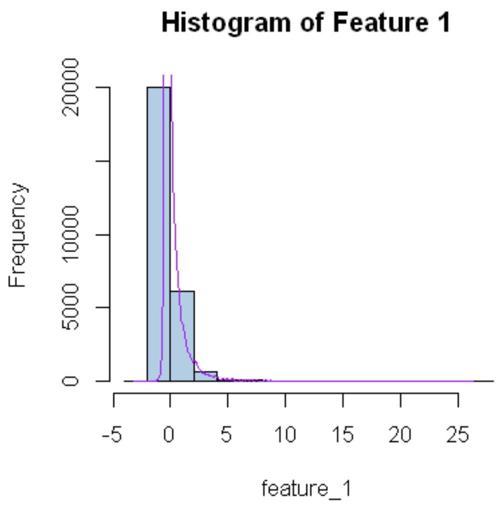
```
0 : 806   0: 788    0:1014    0:6652     0:2925     0:5680     0:4361
1 :1432   1:4021    1:3527    1: 130     1:3857     1:1102     1: 428
2 : 220   2:1973    2:1946                                     2:1993
3 : 182             3: 295
4 :1408
5 : 354
6 : 225
7 : 640
8 : 135
9 :1160
10: 179
11:  41
feature_14 feature_15 labels
0 : 435    0: 736     0:5997
1 : 928    1: 274     1: 785
2 :  33    2: 211
3 : 401    3:5561
4 : 186
5 :1026
6 : 816
7 :  62
8 :2103
9 : 599
10: 109
11:  84
```

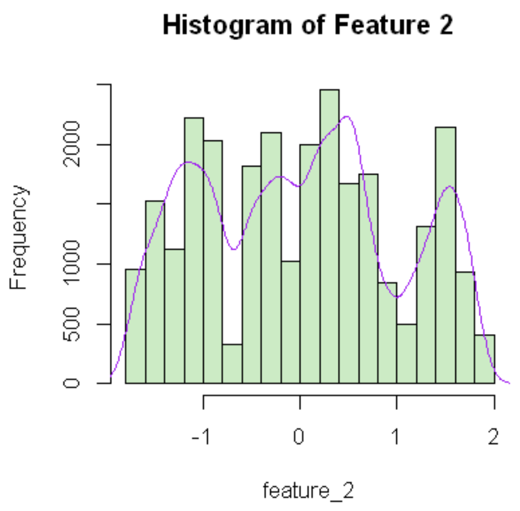
Qualitative data ranges from 2 to 12 categories. None are evenly distributed, while many have a clear preference towards one class over the others, such as feature_13 which has 17,617 in class 0 and less than 10,000 observations in the other two classes (1 & 2) combined.

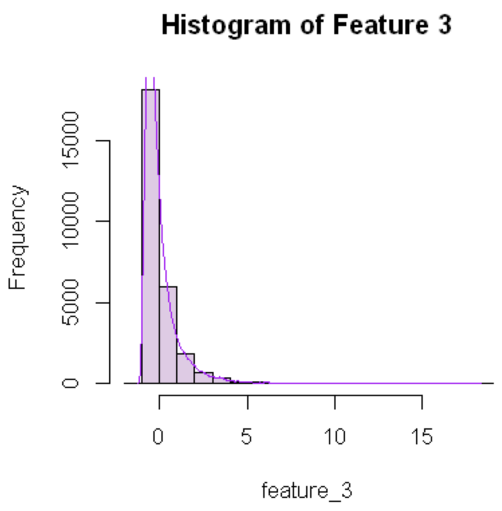
Histograms and Density Plots for our continuous data:

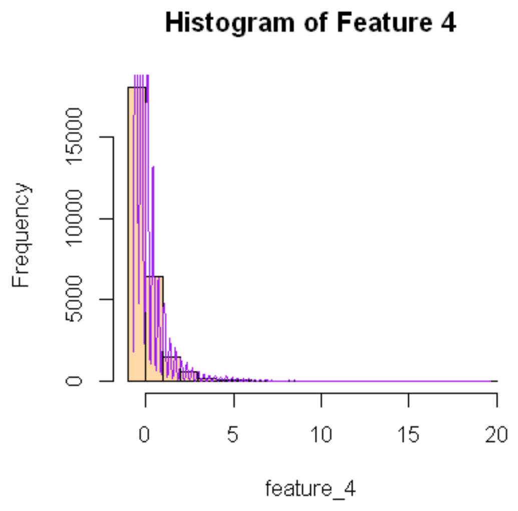
[15]:
```r
#increase scale size of jupyter plots for some very detailed ones momentarily.
options(jupyter.plot_scale=1)

# Histogram for features 0 to 6
par(mfrow = c(2,2))
for (i in 1:7 ) {
    h <- hist(train[,i], plot=FALSE)
    plot(h, main=paste("Histogram of Feature", i-1), xlab = names(train[i]),␣
 ↪col = brewer.pal(8, name = "Pastel1")[i])
    d <- density(train[,i])
    lines(x = d$x, y = d$y * length(train[,i]) * diff(h$breaks)[1], col =␣
 ↪"purple", lwd = 1.5)
}
```

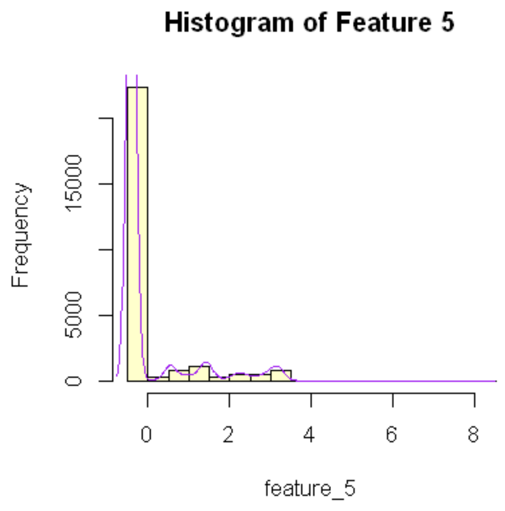

Histogram of Feature 0
Frequency
0
1000
3000
5000
-2
0
2
4
feature_0

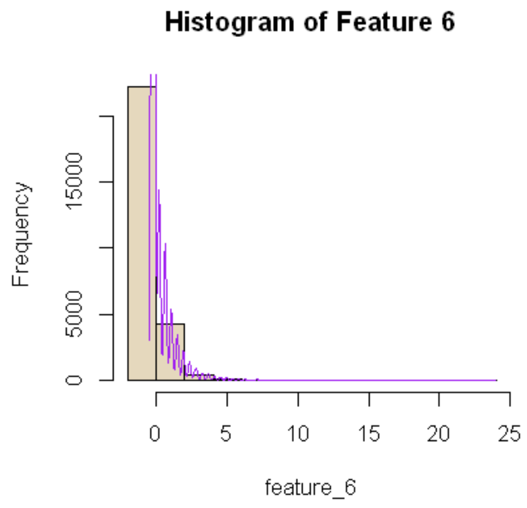

Histogram of Feature 1
Frequency
0
5000
10000
20000
-5
0
5
10
15
20
25
feature_1


Histogram of Feature 2
Frequency
0
500
1000
2000
-1
0
1
2
feature_2


Histogram of Feature 3
Frequency
0
5000
10000
15000
0
5
10
15
feature_3



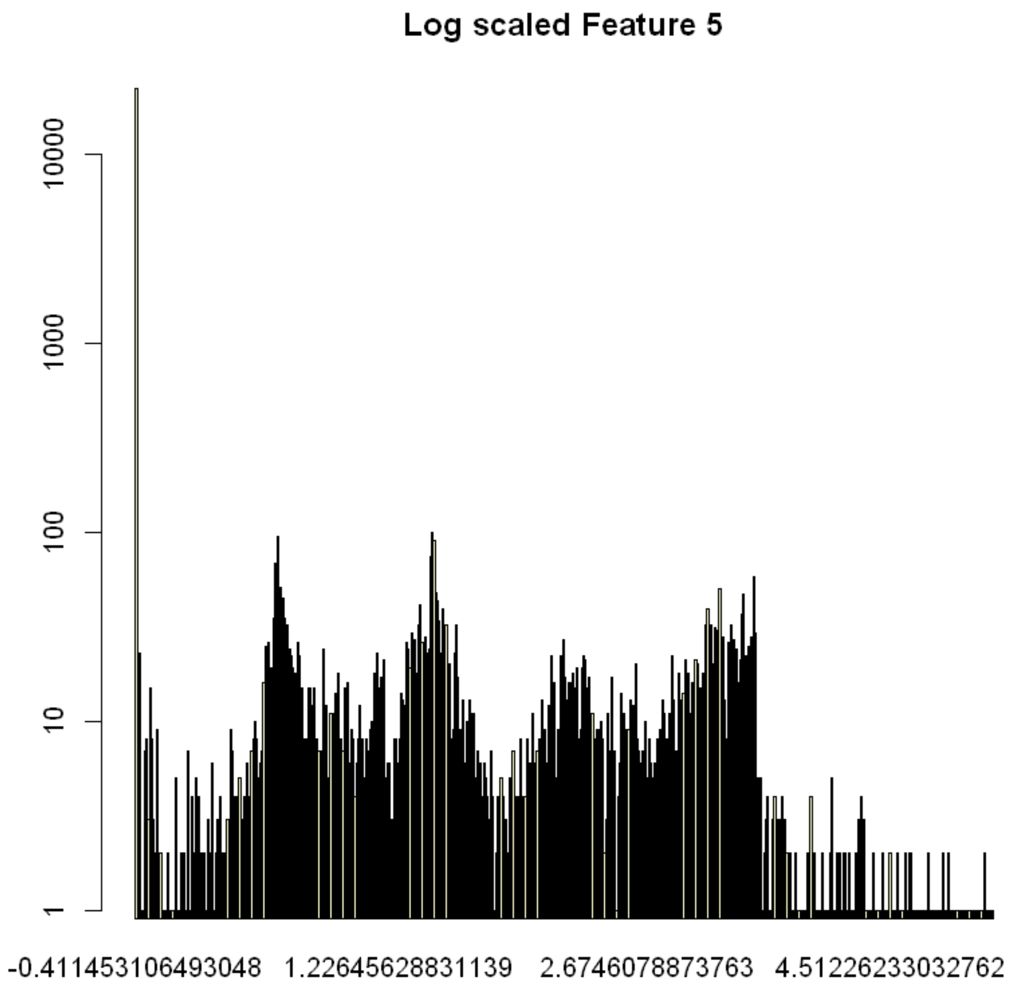




The histograms seem to tell us that the data here is a mix of left skewed (feature 0), exponential (features 1, 3, 4 & 6) and uniformly (2?) distributed features are seen here. But the density plots say a whole lot more. Most of our exponential data is just very highly right skewed unimodal. Our questionably uniform feature (2) could alternatively be multimodal. There are also some interesting artefacts on features 4 & 6, some kind of decaying harmonic oscillation. Feature 5 appears to have some kind of normal distribution with a right tail, could be right skewed unimodal, or it could be something else, it might help to take a closer look.

Below we use log scaling to look more closely at the tail end of the data in feature 5.

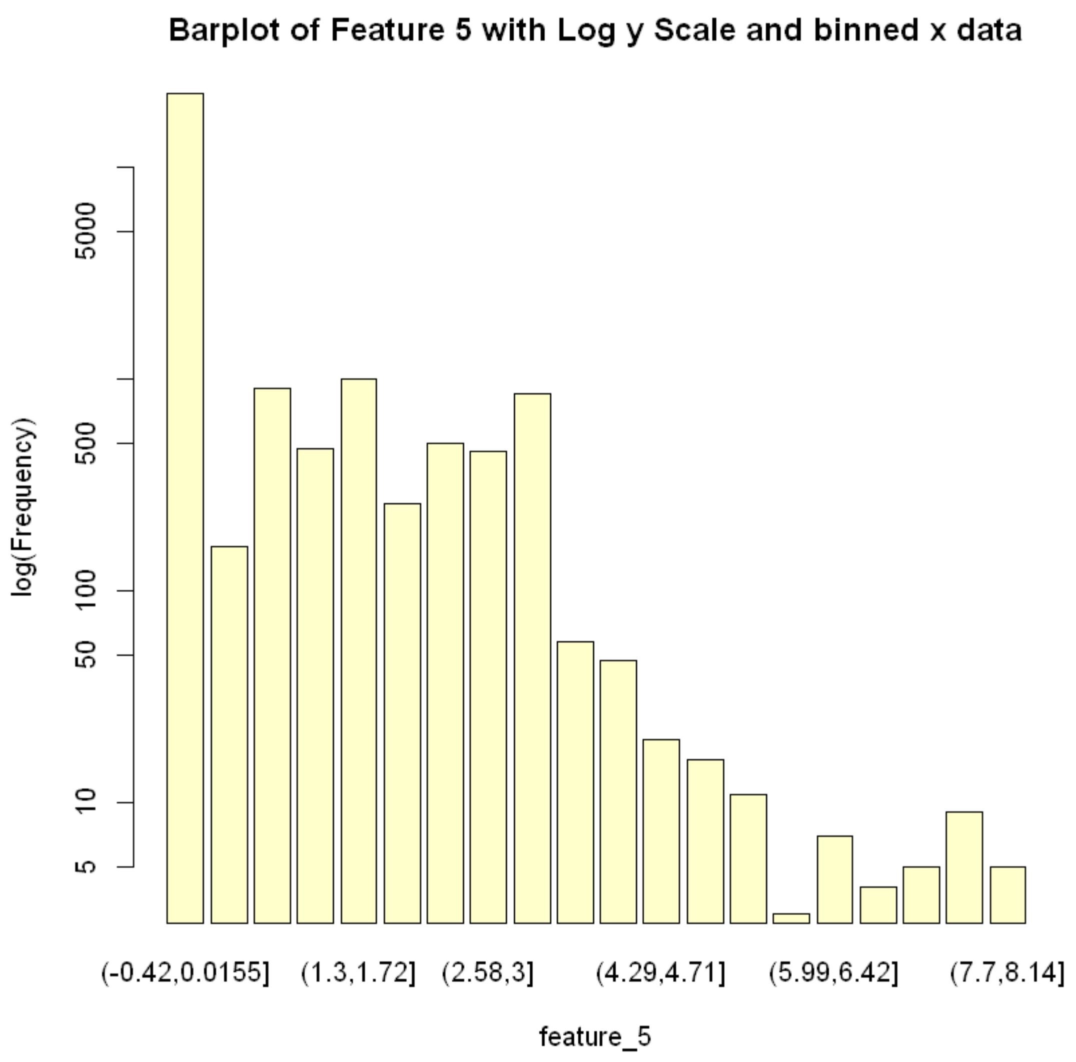
[16]:
```
barplot(table(train[,6]), log="y", xlab = names(train[,6]), main = paste("Log␣
↪scaled Feature", 5), col = brewer.pal(8, name = "Pastel1")[6])
```



This is definitely not the best chart option for the job, but it does have an in-built log scale. It shows that the data could be multimodal.

It could be improved with binning:

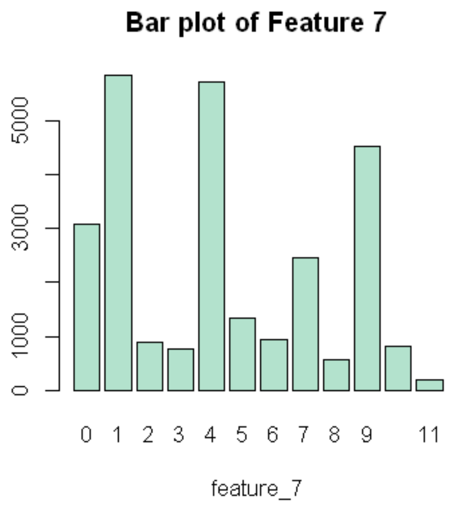
```
[17]: barplot(table(bin(train[,6], nbins = 20)), log = "y", xlab = "feature_5", ylab␣
      ↪= "log(Frequency)", main = "Barplot of Feature 5 with Log y Scale and binned␣
      ↪x data", col = brewer.pal(8, name = "Pastel1")[6])
```

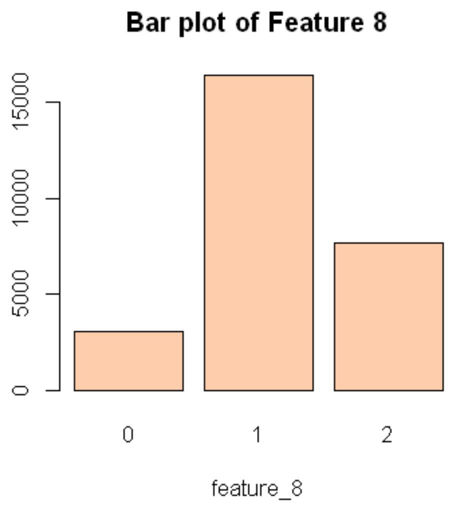


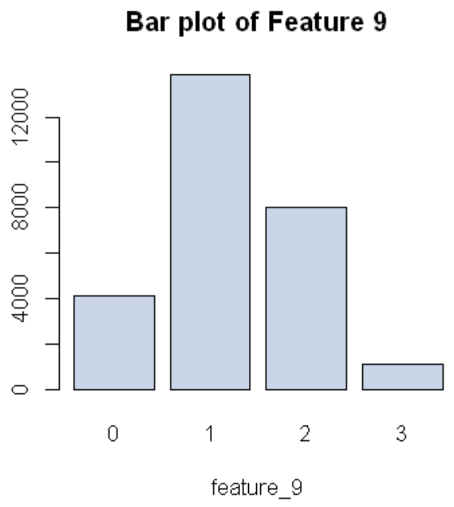
This is quite unusual, not an easily identifiable distribution.

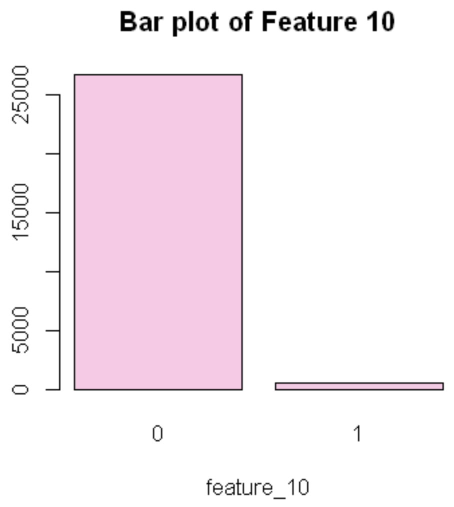
Moving on. Bar chart exploration for our (assumed) qualitative data:

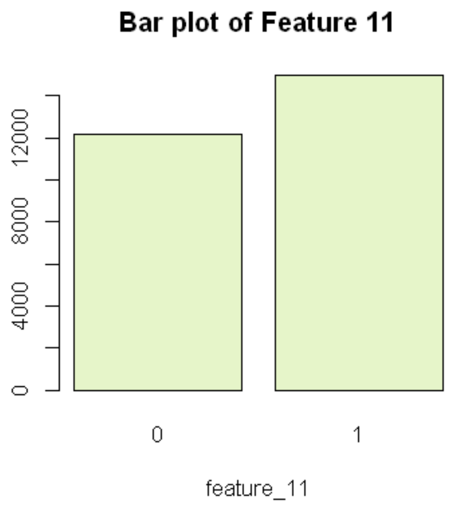
```
[18]: # Bar chart for features 7 to 15
      par(mfrow = c(2,2))
      for (i in 8:16 ) {
              barplot(table(train[,i]), xlab = names(train[i]), main = paste("Bar␣
       ↪plot of Feature", i-1), col = brewer.pal(8, name = "Pastel2")[i-7])
      }
```

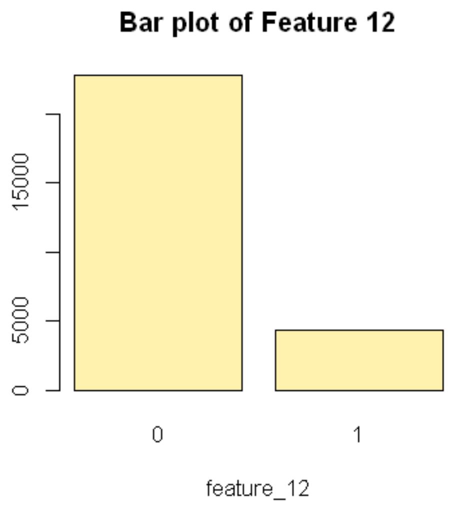

Bar plot of Feature 7
0
1000
3000
5000
0 1 2 3 4 5 6 7 8 9 11
feature_7

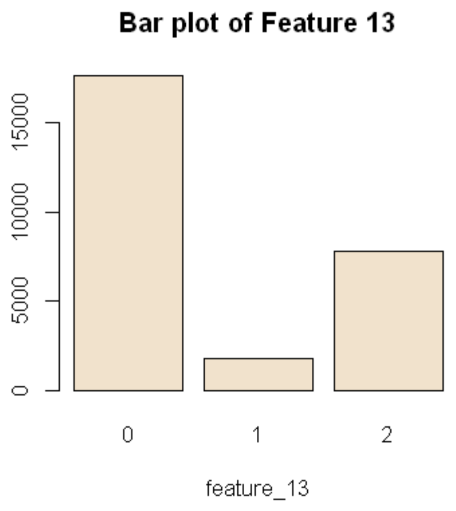

Bar plot of Feature 8
0
5000
10000
15000
0
1
2
feature_8

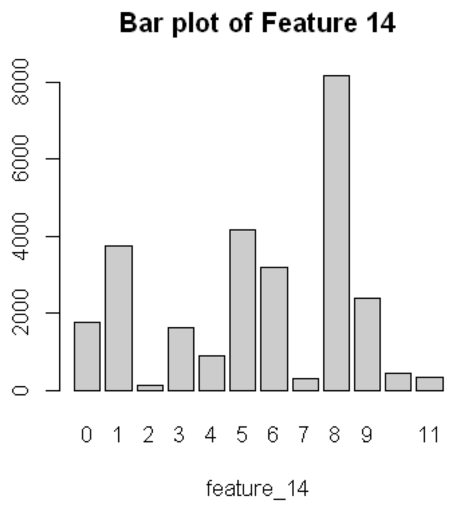

Bar plot of Feature 9
0
4000
8000
12000
0
1
2
3
feature_9

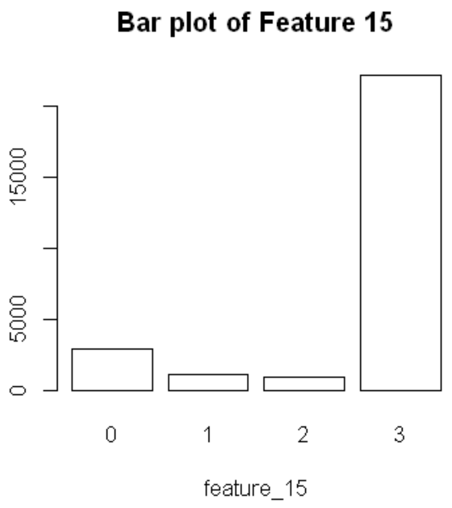

Bar plot of Feature 10
0
5000
15000
25000
0
1
feature_10


Bar plot of Feature 11
0
4000
8000
12000
0
1
feature_11


Bar plot of Feature 12
0
5000
15000
0
1
feature_12


Bar plot of Feature 13
0
5000
10000
15000
0
1
2
feature_13


Bar plot of Feature 14
0
2000
4000
6000
8000
0 1 2 3 4 5 6 7 8 9 11
feature_14



As found earlier, features 10, 11, 12 and the label are binomial. Features 10 & 12 are primarily distributed at zero. Feature 11 on the other hand is a little more evenly distributed.

The rest of our features are multinomial with little patterning to them.

### 1.4.2 2.2. Summary of Univariate Analysis

### 1.4.3 Quantitative vs Qualitative Variables:

We have identified that features 0-6 are continuous, of those 2,4 & 6 appear to be discrete. As discussed earlier, we continue with the assumption that features 7-15 are categoric. It is possible that their (features 7-15) distributions and/or relationships with other variables could reveal the possibility of a rank or logical order, therefore their erratic distributions (notably in feature 7, 14 & 15) could be an indication that they are nominal (have no order or rank), we cannot really know for certain which features are ordinal or nominal though. This information is important to establish early so that we may choose appropriate visual exploration methods and determine whether dummy encoding is needed for specific learning algorithms later on.

### 1.4.4 Distributions:

Other notable observations include that feature 0, 1, 3, 4 & 6 are all skewed (although the histogram would have you believe otherwise). Feature 5 looks interesting, it could be skewed unimodal, bimodal or something else entirely. Meanwhile feature 2 is fairly uniformly distributed (possibly 7 and 11 too). Many of the integer features (7 and above) show a clear imbalance where one class is preference (particularly 10, 12 and 15). Features 8 and 9 show some degree of normal distribution however, if it is categoric this is probably not the case. It's hard to determine what information 7 & 14 have to give us, they are discrete with no clear distribution meaning that they could be nominal variables with no clear rank or order, these would be prime candidates for tree methods or dummy encoding for other learning methods.

### 1.4.5 Labels class imbalance

The response variable is not evenly distributed. The majority class (0 / No Churn) is very dominant here and so we expect that some learning algorithms will significantly favour this class for positive prediction unless we compensate. Resampling or shifting probability thresholds may be necessary so as not to introduce bias in our learning/modelling algorithms.

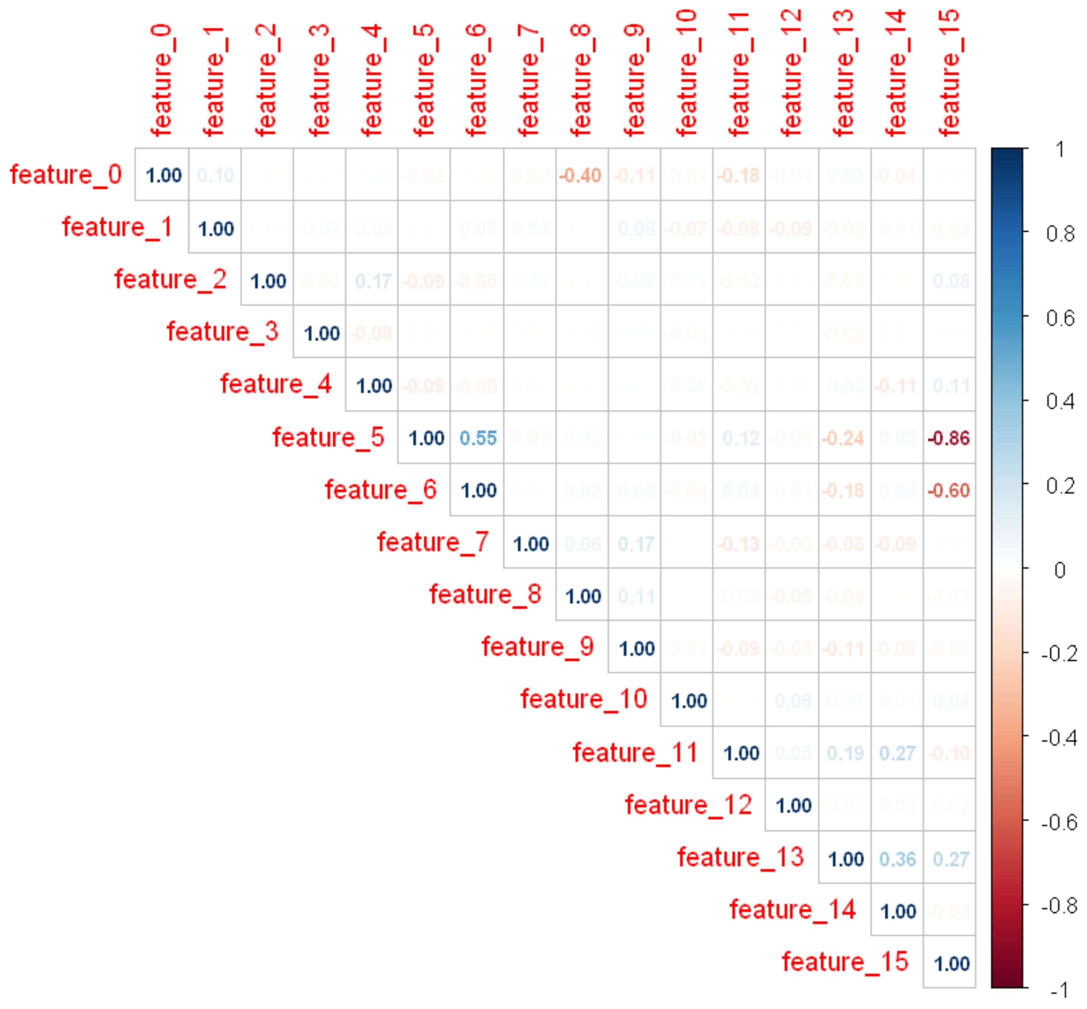
### 1.4.6 Other notes

Feature scaling has not been completed on this dataset. Some learning algorithms might need us to perform this. There is no missing data in any features and since we don't have any particular knowledge about each feature, we cannot comment more generally on the expected or unexpected characteristics of each variable, such as the range of values and outliers.

### 1.4.7 2.3. Bivariate Analysis

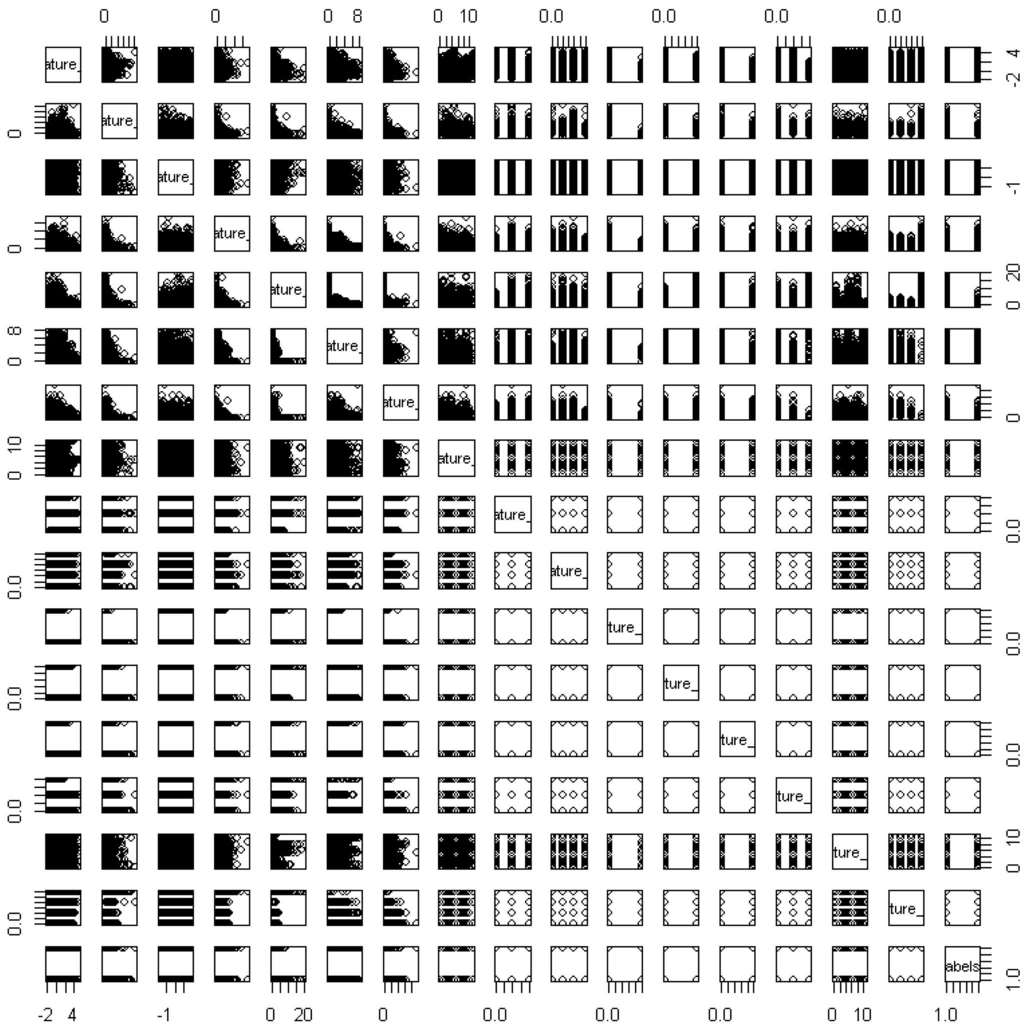
Checking for highly colinear features.

[19]:
```r
# exclude our label from correlation data since it is a factor
cor <- cor(train[-17])

# plot correlations
corrplot(cor, method = 'number', type = 'upper',number.cex=0.75)
```



Feature 15 and 5 are highly negatively correlated, 15 and 6 are somewhat similar too. Features 5 & 6 are positively correlated with each other, which probably explains their similar relationship with 15. Feature 15 or 5 & 6 should be removed. Feature 0 & 8 are also somewhat negatively correlated. Otherwise, correlation between other variables are almost zero. Next we take a quick peek at the pair wise relationships in our data, as that will give us a quick idea of whether there are any important non-linear relationships.

[20]:
```
pairs(train)
```



There are no other descernable realtionships other than those we already established.

```
[21]: cor(train[,-17], as.numeric(train[,17]))
```

A matrix: 16 × 1 of type dbl

| | |
|---|---|
| feature_0 | 0.01550229 |
| feature_1 | 0.05064085 |
| feature_2 | -0.02975611 |
| feature_3 | 0.39008974 |
| feature_4 | -0.07615220 |
| feature_5 | 0.11057683 |
| feature_6 | 0.11904202 |
| feature_7 | 0.03671097 |
| feature_8 | 0.04254066 |
| feature_9 | 0.06883864 |
| feature_10 | -0.02411465 |
| feature_11 | -0.13572276 |
| feature_12 | -0.06511450 |
| feature_13 | -0.14820666 |
| feature_14 | -0.02532398 |
| feature_15 | -0.08062353 |

It appears that feature 3 (cor = 0.39) will be our most important predictor. It stands out above all other features with more than twice the correlation coefficient of any other.

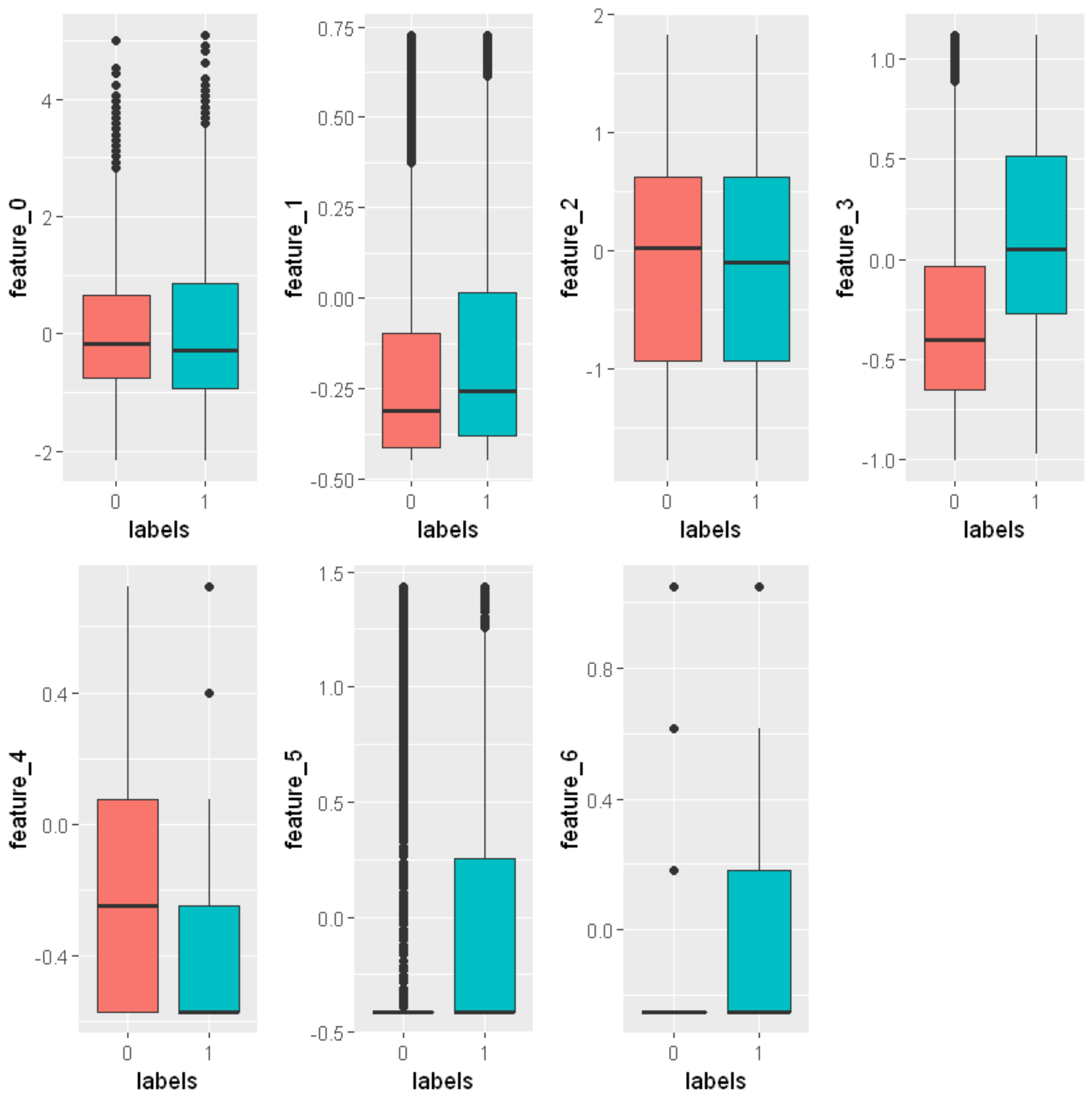
### 1.4.8 Comparing continuous variables against the categoric label:

Side by side box plots are ideal for comparing the continuous variables against our categoric response variable (labels). Many outliers have been deliberately removed with boxplots rescaled, as it was difficult to see the main quantiles of the distributions.

[22]:
```
p1 <- ggplot(train, aes(x=factor(labels), y=train[,1], fill=factor(labels))) +
          geom_boxplot()+
          #scale_y_continuous(limits = c(quantile(train[,1], c(0.1, 0.9))[1],␣
 ↪quantile(train[,1], c(0.1, 0.9))[2]))+
          labs(x = "labels",y = names(train)[1]) +
          guides(fill="none")

p2 <- ggplot(train, aes(x=factor(labels), y=train[,2], fill=factor(labels))) +
          geom_boxplot()+
          scale_y_continuous(limits = c(quantile(train[,2],c(0.1, 0.9))[1],␣
 ↪quantile(train[,2], c(0.1, 0.9))[2]))+
          labs(x = "labels",y = names(train)[2]) +
          guides(fill="none")

p3 <- ggplot(train, aes(x=factor(labels), y=train[,3], fill=factor(labels))) +
          geom_boxplot()+
          #scale_y_continuous(limits = c(quantile(train[,3], c(0.1, 0.9))[1],␣
 ↪quantile(train[,3], c(0.1, 0.9))[2]))+
          labs(x = "labels",y = names(train)[3]) +
          guides(fill="none")

p4 <- ggplot(train, aes(x=factor(labels), y=train[,4], fill=factor(labels))) +
```

```r
          geom_boxplot()+
          scale_y_continuous(limits = c(quantile(train[,4], c(0, 0.9))[1],␣
 ↪quantile(train[,4], c(0, 0.9))[2]))+
          labs(x = "labels",y = names(train)[4]) +
          guides(fill="none")

p5 <- ggplot(train, aes(x=factor(labels), y=train[,5], fill=factor(labels))) +
          geom_boxplot()+
          scale_y_continuous(limits = c(quantile(train[,5], c(0, 0.9))[1],␣
 ↪quantile(train[,5], c(0, 0.9))[2]))+
          labs(x = "labels",y = names(train)[5]) +
          guides(fill="none")

p6 <- ggplot(train, aes(x=factor(labels), y=train[,6], fill=factor(labels))) +
          geom_boxplot()+
          scale_y_continuous(limits = c(quantile(train[,6], c(0.1, 0.9))[1],␣
 ↪quantile(train[,6], c(0.1, 0.9))[2]))+
          labs(x = "labels",y = names(train)[6]) +
          guides(fill="none")

p7 <- ggplot(train, aes(x=factor(labels), y=train[,7], fill=factor(labels))) +
          geom_boxplot()+
          scale_y_continuous(limits = c(quantile(train[,7], c(0, 0.95))[1],␣
 ↪quantile(train[,7], c(0, 0.95))[2]))+
          labs(x = "labels",y = names(train)[7]) +
          guides(fill="none")


grid.arrange(p1, p2, p3, p4, p5, p6, p7, ncol = 4)
```

```
Warning message:
"Removed 4955 rows containing non-finite values (stat_boxplot)."
Warning message:
"Removed 2705 rows containing non-finite values (stat_boxplot)."
Warning message:
"Removed 2602 rows containing non-finite values (stat_boxplot)."
Warning message:
"Removed 2691 rows containing non-finite values (stat_boxplot)."
Warning message:
"Removed 1314 rows containing non-finite values (stat_boxplot)."
```

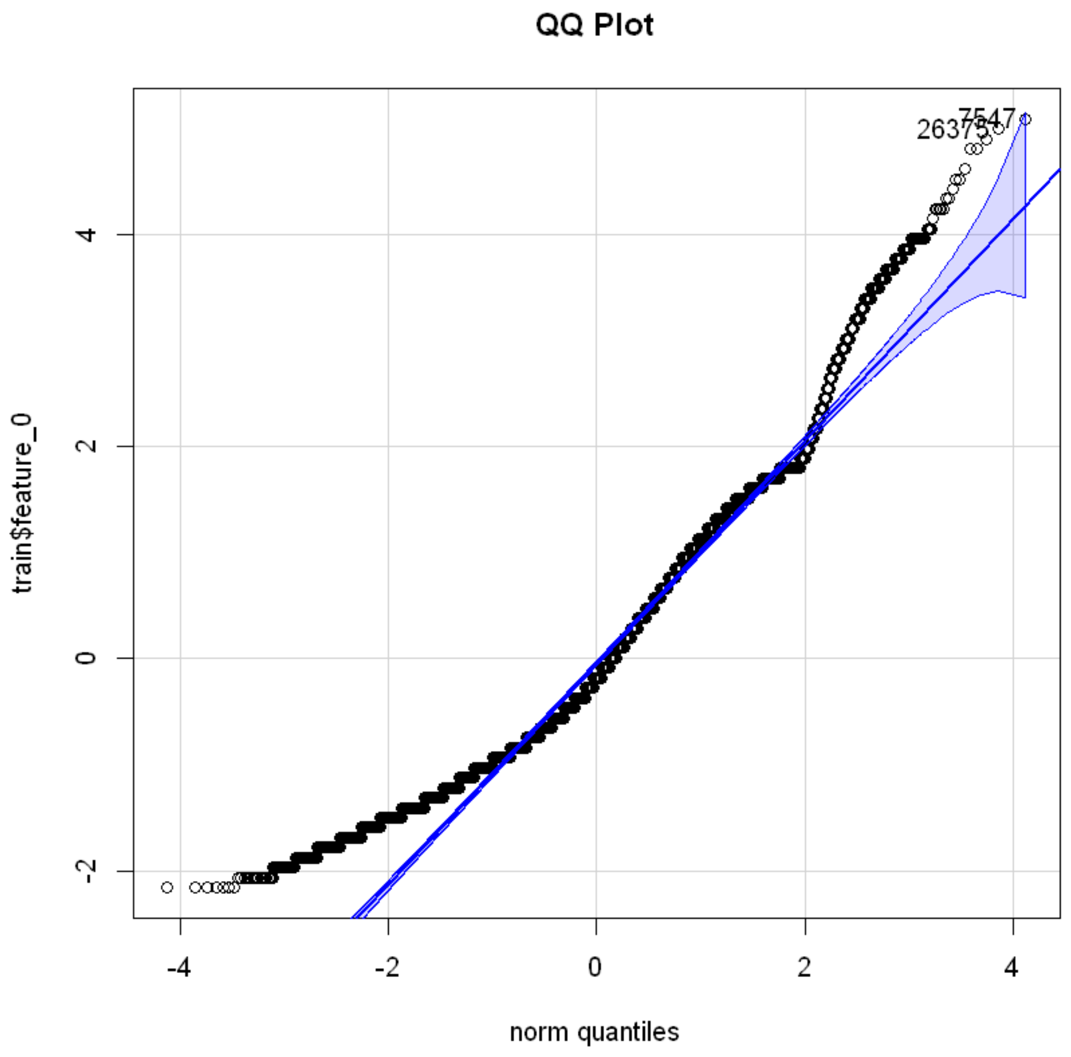


These box plots reveal that features 1 & 3 exhibit higher mean for churn (1) compared to no churn (0). Feature 0 has a wider distribution for churn than no churn. Feature 2 has a slightly lower mean for churn but the same overall spread. feature 5 & 6 appear to be very similar, with the same mean but greater spread for churn. Lastly, feature 4 has lower mean and spread for churn. These are explored in increased detail below with density plots. There are a great many potential outliers sitting outside the third quartile in all except feature 3.

We could compute a t-test to determine whether the means are significantly different, however this requires the assumption that our data is normally distributed, which it isn't. Even our most normal variable - feature_0 doesn't meet this criteria, as seen in the QQ plot below.

[23]:
```
qqPlot(train$feature_0, main="QQ Plot", distribution="norm", envelope=.99)
```

1. 7547 2. 26375

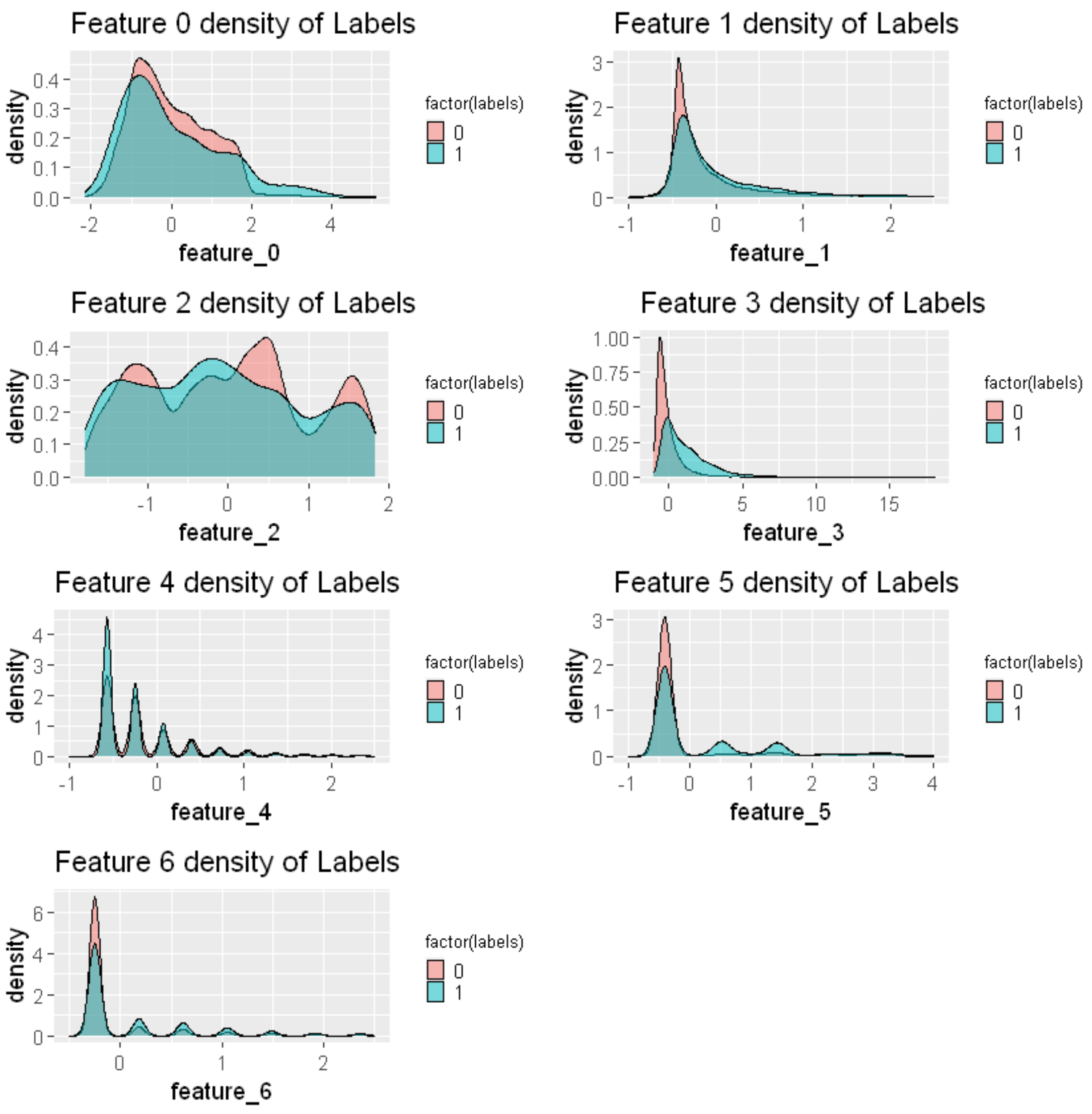


Exploring density plots:

```
[24]: p1 <- ggplot(train) +
        geom_density(aes(x = feature_0, fill = factor(labels)), alpha=.5) +
        labs(x = 'feature_0') +
        ggtitle("Feature 0 density of Labels")+
        theme(legend.key.size = unit(0.3, 'cm'), legend.title = element_text(size=8))

      p2 <- ggplot(train) +
        geom_density(aes(x = feature_1, fill = factor(labels)), alpha=.5) +
        labs(x = 'feature_1') +
        ggtitle("Feature 1 density of Labels")+
        theme(legend.key.size = unit(0.3, 'cm'), legend.title = element_text(size=8))␣
      ↪+
        xlim(c(-1,2.5))

      p3 <- ggplot(train) +
        geom_density(aes(x = feature_2, fill = factor(labels)), alpha=.5) +
        labs(x = 'feature_2') +
        ggtitle("Feature 2 density of Labels")+
        theme(legend.key.size = unit(0.3, 'cm'), legend.title = element_text(size=8))

      p4 <- ggplot(train) +
        geom_density(aes(x = feature_3, fill = factor(labels)), alpha=.5) +
        labs(x = 'feature_3') +
        ggtitle("Feature 3 density of Labels")+
        theme(legend.key.size = unit(0.3, 'cm'), legend.title = element_text(size=8))

      p5 <- ggplot(train) +
        geom_density(aes(x = feature_4, fill = factor(labels)), alpha=.5) +
        labs(x = 'feature_4') +
        ggtitle("Feature 4 density of Labels")+
        theme(legend.key.size = unit(0.3, 'cm'), legend.title = element_text(size=8))␣
      ↪+
        xlim(c(-1,2.5))

      p6 <- ggplot(train) +
        geom_density(aes(x = feature_5, fill = factor(labels)), alpha=.5) +
        labs(x = 'feature_5') +
        ggtitle("Feature 5 density of Labels")+
        theme(legend.key.size = unit(0.3, 'cm'), legend.title = element_text(size=8))␣
      ↪+
        xlim(c(-1,4))

      p7 <- ggplot(train) +
        geom_density(aes(x = feature_6, fill = factor(labels)), alpha=.5) +
        labs(x = 'feature_6') +
        ggtitle("Feature 6 density of Labels")+
        theme(legend.key.size = unit(0.3, 'cm'), legend.title = element_text(size=8))␣
      ↪+
        xlim(c(-0.5,2.5))

      grid.arrange(p1, p2, p3, p4, p5, p6, p7, ncol = 2)
```

```
Warning message:
"Removed 619 rows containing non-finite values (stat_density)."
Warning message:
"Removed 726 rows containing non-finite values (stat_density)."
Warning message:
```

```
"Removed 111 rows containing non-finite values (stat_density)."
Warning message:
"Removed 447 rows containing non-finite values (stat_density)."
```



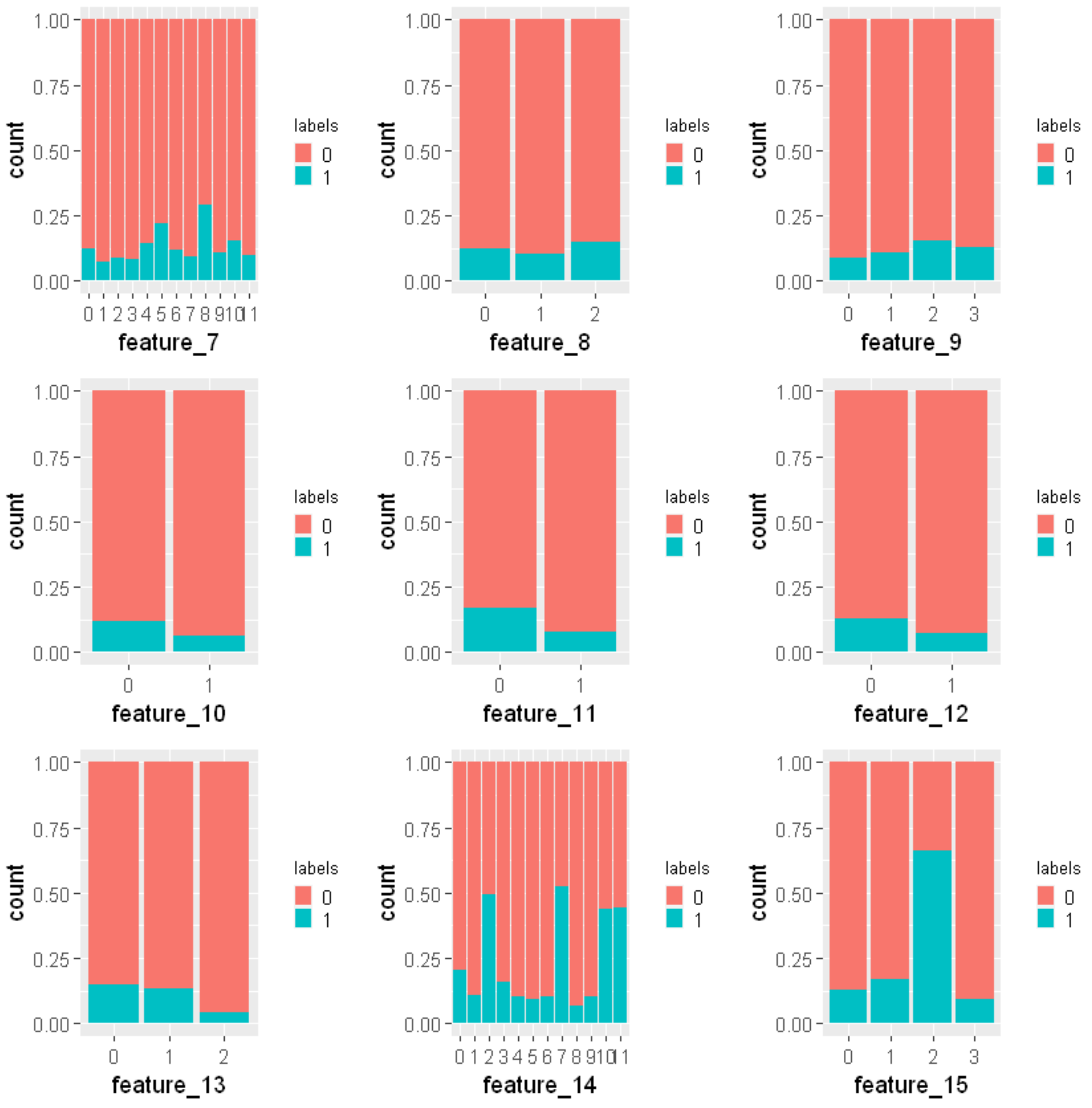
These comparisons for features 0, 1 & 3 are explained well enough by their boxplots. The additional detail in features 2, 4,5 & 6 is interesting. Features 4, 5 and 6 appear to have similar distributions for churn versus no churn, however they differ in the magnitude of their peaks - feature 4's high peak colours are the inverse of 5 & 6. Feature 2 is different, it has churn and no churn peaks in different places. It now appears more multimodal than uniformly distributed.

The difference in mean and spread for each label class in features 1 & 3 remains convincing as a strong predictors.

### 1.4.9 Comparing two categoric variables

Bar charts are used to compare features 7-15 against their labels. Since there is substantial imbalance between the churn and no churn rates, we use a percentage scale for comparisons.

[25]:
```r
p7<- ggplot(train, aes(x = factor(feature_7), fill = labels)) +
  geom_bar(position = "fill") +
  theme(legend.key.size = unit(0.3, 'cm'), legend.title = element_text(size=8))+
  xlab('feature_7')

p8<- ggplot(train, aes(x = factor(feature_8), fill = labels)) +
  geom_bar(position = "fill") +
  theme(legend.key.size = unit(0.3, 'cm'), legend.title = element_text(size=8))+
  xlab('feature_8')

p9<- ggplot(train, aes(x = factor(feature_9), fill = labels)) +
  geom_bar(position = "fill") +
  theme(legend.key.size = unit(0.3, 'cm'), legend.title = element_text(size=8))+
  xlab('feature_9')

p10<- ggplot(train, aes(x = factor(feature_10), fill = labels)) +
  geom_bar(position = "fill") +
  theme(legend.key.size = unit(0.3, 'cm'), legend.title = element_text(size=8))+
  xlab('feature_10')

p11<- ggplot(train, aes(x = factor(feature_11), fill = labels)) +
  geom_bar(position = "fill") +
  theme(legend.key.size = unit(0.3, 'cm'), legend.title = element_text(size=8))+
  xlab('feature_11')

p12<- ggplot(train, aes(x = factor(feature_12), fill = labels)) +
  geom_bar(position = "fill") +
  theme(legend.key.size = unit(0.3, 'cm'), legend.title = element_text(size=8))+
  xlab('feature_12')

p13<- ggplot(train, aes(x = factor(feature_13), fill = labels)) +
  geom_bar(position = "fill") +
  theme(legend.key.size = unit(0.3, 'cm'), legend.title = element_text(size=8))+
  xlab('feature_13')

p14<- ggplot(train, aes(x = factor(feature_14), fill = labels)) +
  geom_bar(position = "fill") +
  theme(legend.key.size = unit(0.3, 'cm'), legend.title = element_text(size=8))+
  xlab('feature_14')

p15<- ggplot(train, aes(x = factor(feature_15), fill = labels)) +
  geom_bar(position = "fill") +
  theme(legend.key.size = unit(0.3, 'cm'), legend.title = element_text(size=8))+
  xlab('feature_15')


grid.arrange(p7, p8, p9, p10, p11, p12, p13, p14, p15, ncol = 3)
```



The overall imbalance in this dataset's labels dictates that churn (1/blue) should be roughly around 12% of the counts. We see substantial deviation from this for some category values in feature 7, 14 and 15. These could be helpful in predicting churn. Because we see no obvious trends in the way the data is split for churn and no churn within these features, this might indicate there is no ordinal relationship between the categories and that dummy encoding would be beneficial in the modelling stage.

Chi-squared test of independence:

[26]:
```
chisq.test(train$labels, train$feature_7, correct=FALSE)
chisq.test(train$labels, train$feature_8, correct=FALSE)
chisq.test(train$labels, train$feature_9, correct=FALSE)
chisq.test(train$labels, train$feature_10, correct=FALSE)
chisq.test(train$labels, train$feature_11, correct=FALSE)
chisq.test(train$labels, train$feature_12, correct=FALSE)
chisq.test(train$labels, train$feature_13, correct=FALSE)
chisq.test(train$labels, train$feature_14, correct=FALSE)
chisq.test(train$labels, train$feature_15, correct=FALSE)
```

```
Pearson's Chi-squared test

data:  train$labels and train$feature_7
X-squared = 500.81, df = 11, p-value < 2.2e-16



Pearson's Chi-squared test

data:  train$labels and train$feature_8
X-squared = 115.64, df = 2, p-value < 2.2e-16



Pearson's Chi-squared test

data:  train$labels and train$feature_9
X-squared = 164.58, df = 3, p-value < 2.2e-16



Pearson's Chi-squared test

data:  train$labels and train$feature_10
X-squared = 15.774, df = 1, p-value = 7.137e-05



Pearson's Chi-squared test

data:  train$labels and train$feature_11
X-squared = 499.68, df = 1, p-value < 2.2e-16



Pearson's Chi-squared test

data:  train$labels and train$feature_12
X-squared = 115.01, df = 1, p-value < 2.2e-16
```

```
Pearson's Chi-squared test

data:  train$labels and train$feature_13
X-squared = 617.63, df = 2, p-value < 2.2e-16



Pearson's Chi-squared test

data:  train$labels and train$feature_14
X-squared = 1788.7, df = 11, p-value < 2.2e-16



Pearson's Chi-squared test

data:  train$labels and train$feature_15
X-squared = 2802.7, df = 3, p-value < 2.2e-16
```

If we have a significance level of 0.05, we can reject the null hypothesis, all tested features are dependent on the labels, therefore there is a relationship between them.

### 1.4.10 2.4. Summary of Bivariate Analysis

In this section correlation was measured between features. Feature 15 and 5 are highly negatively correlated, 15 and 6 are somewhat similar too. Since we feel feature 15 could be a strong categoric predictor, features 5 & 6 should be removed.

Correlation against the label is used to identify that feature 3 could be our most important predictor.

Boxplots of the numeric variables broken down by label confirms again that feature 3 has a lot of promise as a predictor. Additionally feature 1 may be of importance. Density plots reveal little extra detail that is helpful above what has been found so far.

Boxplots also show that there are a great many potential outliers sitting outside the third quartile in all features except feature 3.

The bar charts of our assumed categoric data show that features 7, 14, 15 have categories which deviate from the expected 12% churn rate for some particular categories, also highlighting that they could prove very useful as predictors.

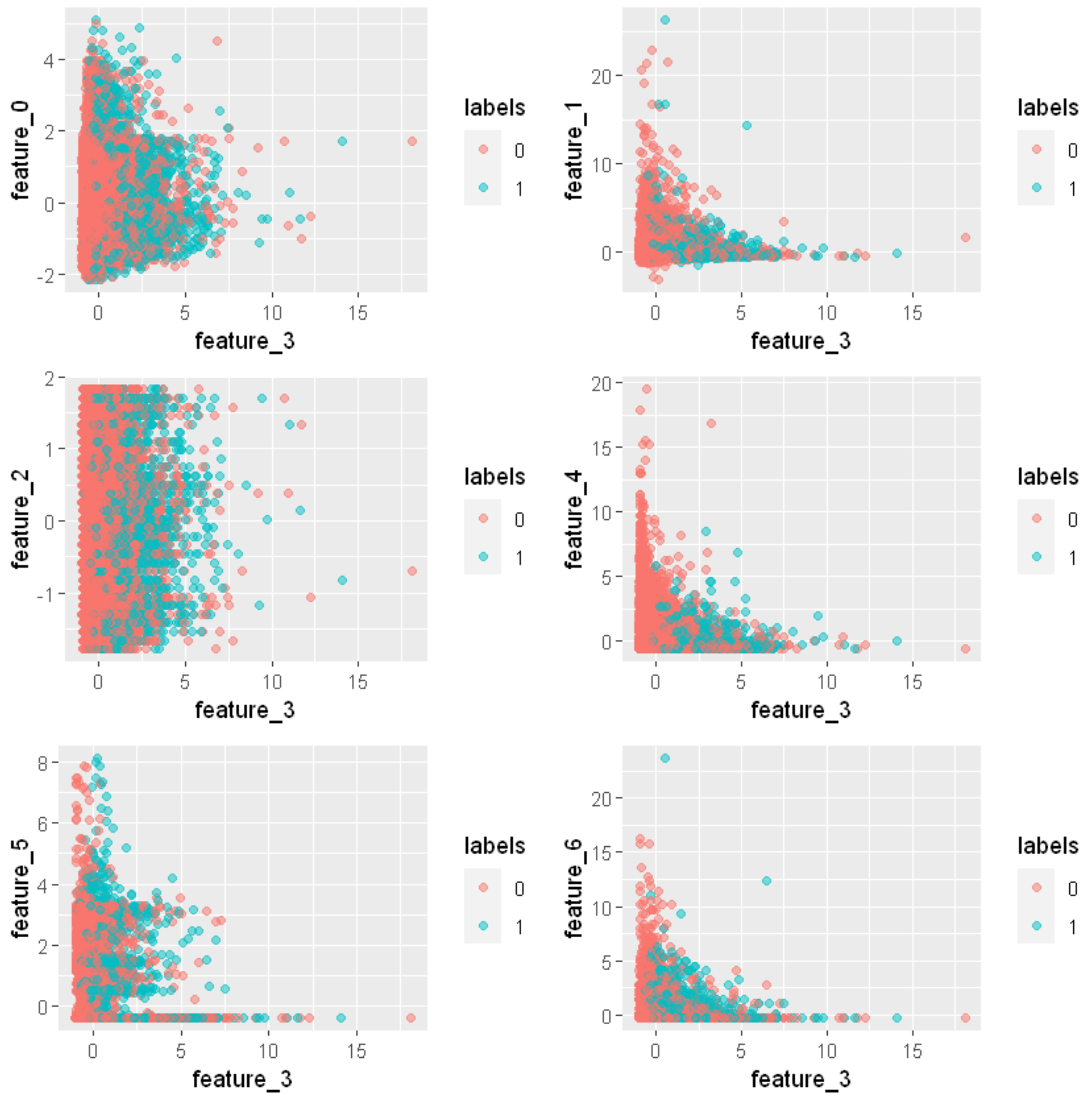
Results from the chi-squared test of independence show evidence that there is a relationship between categoric variables 7-14 and out label. This supports the theory that features 7-15 could be categoric.

### 1.4.11 2.5. Multivariate Analysis

### 1.4.12 Analysing interactions between two continuous variables and the label:

In this section we have explored interactions between all continuous variables with colour separation of the label, only those with interactions of interest have been kept and discussed. If there is no difference of behavior seen between the churn and no churn label, the plot was removed.

[27]:
```
p0 <- ggplot(train, aes(x=feature_3, y=feature_0)) +
        geom_point(aes(colour = labels), alpha=.5)

p1 <- ggplot(train, aes(x=feature_3, y=feature_1)) +
        geom_point(aes(colour = labels), alpha=.5)

p2 <- ggplot(train, aes(x=feature_3, y=feature_2)) +
        geom_point(aes(colour = labels), alpha=.5)

p4 <- ggplot(train, aes(x=feature_3, y=feature_4)) +
        geom_point(aes(colour = labels), alpha=.5)

p5 <- ggplot(train, aes(x=feature_3, y=feature_5)) +
        geom_point(aes(colour = labels), alpha=.5)

p6 <- ggplot(train, aes(x=feature_3, y=feature_6)) +
        geom_point(aes(colour = labels), alpha=.5)

grid.arrange(p0, p1, p2, p4, p5, p6, ncol = 2)
```



During this exploration we only found clear/noticeable separation of the label classes (1/Blue = Churn) in plots that involve feature 3.

Feature 0 x 3: Churn is more commonly found towards the right

Feature 1 x 3: Churn is more commonly found towards the lower right

Feature 2 x 3: Churn is more commonly found towards the right. This appears to be a much more uniform separation than for some other features.

Feature 4 x 3: Churn is more commonly found towards the lower right

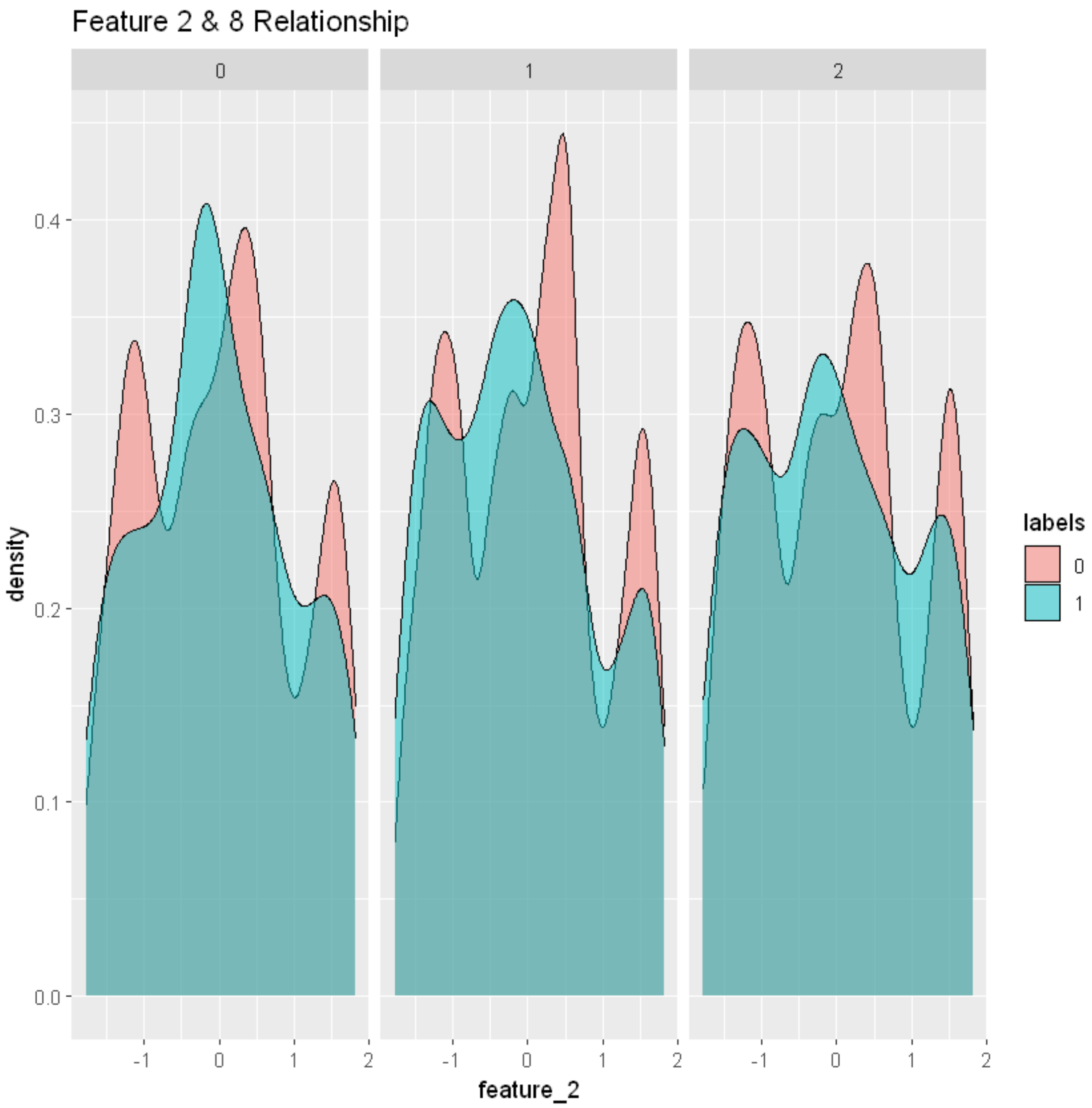
Feature 5 x 3: Churn is more commonly found towards the right

Feature 6 x 3: Churn is more commonly found towards the lower right

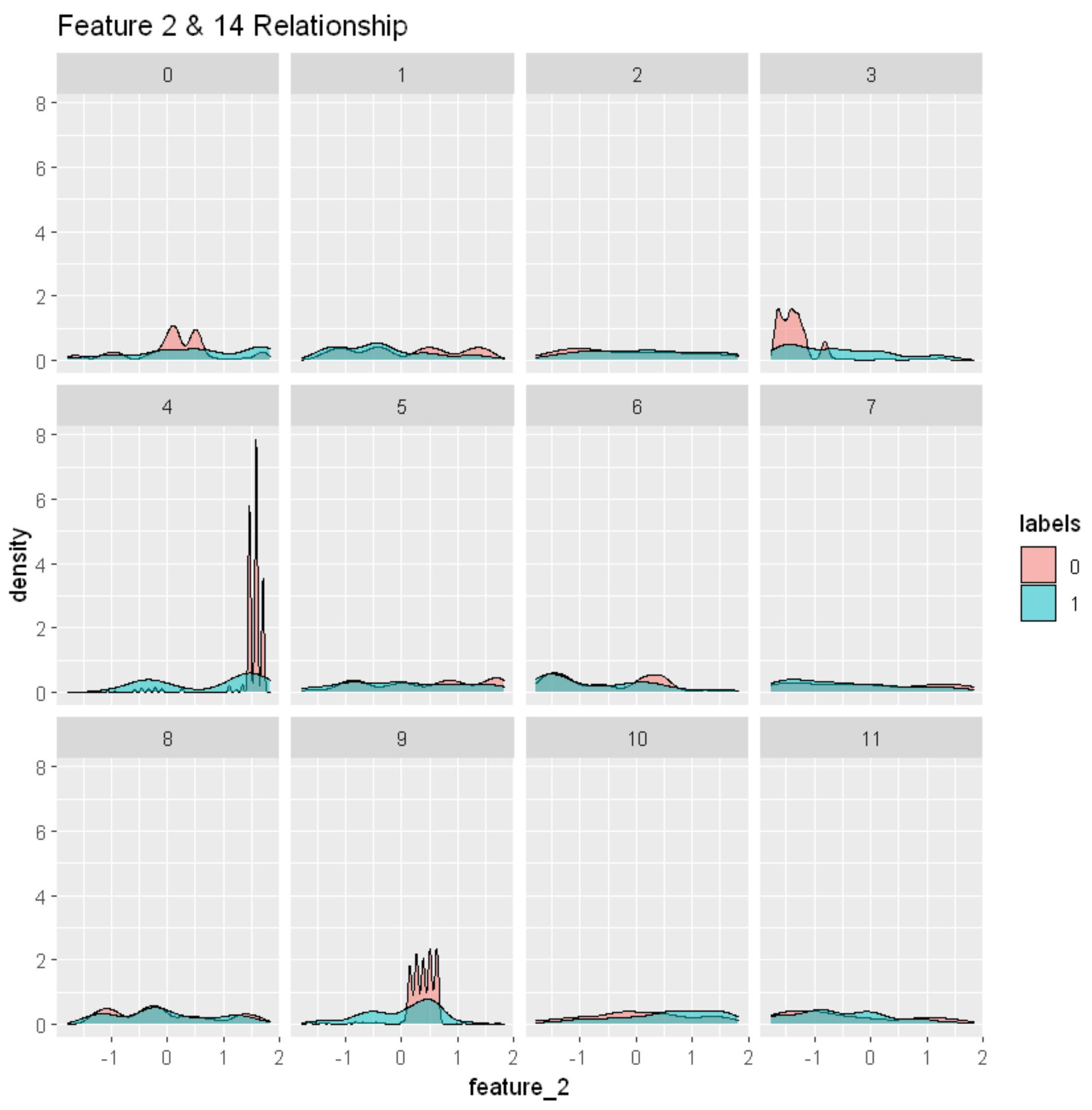
### 1.4.13 Analysing interactions between a continuous and discrete variable with the label:

In this section we have explored interactions between all continuous and discrete variable combinations with colour separation of the label, only those with interactions of interest have been kept and discussed. If there is no difference of behaviour seen between the churn and no churn label, the plot was removed. Also, if there was no new behaviours not already discovered during earlier exploration, the plot was removed.

[28]:
```
ggplot(aes(x=feature_2),data =train) +
    geom_density(aes(fill = labels), alpha=.5) +
    facet_wrap(~feature_8) +
    ggtitle('Feature 2 & 8 Relationship')

ggplot(aes(x=feature_2),data =train) +
    geom_density(aes(fill = labels), alpha=.5) +
    facet_wrap(~feature_14) +
    ggtitle('Feature 2 & 14 Relationship')
```



Feature 2 & 8 Relationship
0
1
2
density
0.0
0.1
0.2
0.3
0.4
-1
0
1
2
feature_2
labels
0
1

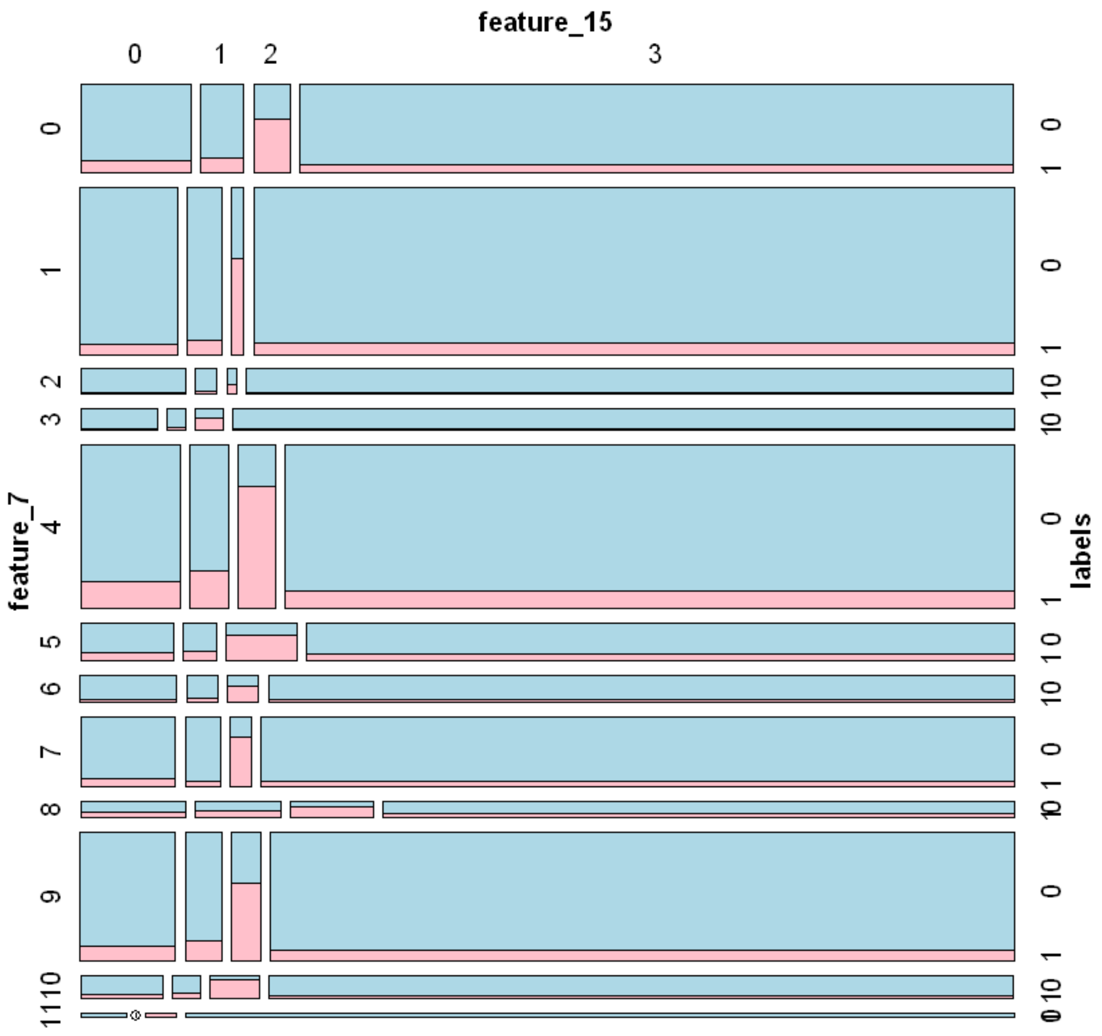


Exploration of feature 2's interactions with other categories was quite interesting. In general, it seems to have four or more peaks that rise and fall (independently for churn and no churn) depending on the feature and category, the interaction between feature 2 & 8 is shown as an example of this. Additionally, the interaction between 2 and 14 is quite unique, it is not clear how to interpret this information currently - perhaps conversion into a discrete form (such as binning into four groups) would make this variable more useful, but this is hard to justify without background knowledge of the data.

### 1.4.14 Analysing interactions between two categoric variables with the label:

Mosaic plots were used to explore category interactions.

[29]:
```
mosaic( ~ feature_7 + feature_15 + labels, data = train,
        highlighting = "labels", highlighting_fill = c("lightblue", "pink"))
```



No new discoveries were made using this method, it only confirmed what we already knew, that features 14 & 15 have categories with much higher churn rates than others.

#### 1.4.15 2.6. Summary of Multivariate Analysis

Numeric variable interactions again confirmed feature 3's significance. Numeric and categorical interactions showed unusual peaks in feature 2 that appear to move independently for churn and no churn groups. As discussed earlier, this behavior seems reminiscent of categorical behavior. It's possible that conversion into a discrete form (such as binning into four groups) would make this variable more useful, but this is hard to justify without background knowledge of the data.

No new discoveries were made using mosaic plots to compare categoric interactions, it only confirmed what we already knew, that features 14 & 15 have some categories with much higher churn rates than others.

### 1.4.16 2.3. More

Build a linear model with all variables:

```
[30]: fit.glm = glm(labels ~ ., data = train, family = binomial)
      summary(fit.glm)
```

```
Call:
glm(formula = labels ~ ., family = binomial, data = train)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-5.5843  -0.4504  -0.2904  -0.1688   3.1445

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept) -2.985656   0.129631 -23.032  < 2e-16 ***
feature_0    0.057227   0.023064   2.481  0.01309 *
feature_1    0.055672   0.017962   3.099  0.00194 **
feature_2   -0.048750   0.022234  -2.193  0.02834 *
feature_3    1.005097   0.020166  49.842  < 2e-16 ***
feature_4   -0.433253   0.040982 -10.572  < 2e-16 ***
feature_5    0.374963   0.032498  11.538  < 2e-16 ***
feature_6    0.263095   0.024147  10.895  < 2e-16 ***
feature_7    0.006500   0.006922   0.939  0.34771
feature_8    0.191809   0.039996   4.796 1.62e-06 ***
feature_9    0.206859   0.029721   6.960 3.40e-12 ***
feature_10  -0.315873   0.208964  -1.512  0.13063
feature_11  -1.004934   0.048833 -20.579  < 2e-16 ***
feature_12  -0.711614   0.073431  -9.691  < 2e-16 ***
feature_13  -0.626692   0.034716 -18.052  < 2e-16 ***
feature_14   0.034472   0.007175   4.804 1.55e-06 ***
feature_15   0.265232   0.037600   7.054 1.74e-12 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 19613  on 27125  degrees of freedom
Residual deviance: 14573  on 27109  degrees of freedom
AIC: 14607

Number of Fisher Scoring iterations: 6
```

Many variables appear to be statistically significant to this linear model. Only 10 & 7 aren't.

[31]:
```
with(fit.glm, pchisq(null.deviance - deviance, df.null - df.residual, lower.
↪tail = FALSE))
```

```
0
```

In this chi-squared test, the p value is zero, it tells us that our model fits much better than an empty model.

As we have seen earlier, there are few variables which correlate strongly with our response variable, only feature_3 stood out initially. However, the takeaway in this section is that we have more than enough information to create a linear model.

#### 1.4.17 2.4. EDA Summary

#### 1.4.18 Univariate

- Data types were difficult to determine given the anonymized data. They are all numeric, however we have completed some exploration of features 7-14 as if they could be numerically encoded categories (for anonymity).
- Features 7 & 14 have the highest variance, this indicates they could be very useful features. On the flip side, features with the lowest variance are 6 & 8-10, we may exclude these from our model on the basis of filter method of feature selection.
- Many features are highly skewed.
- There are substantial amounts of data outside the third quartile of our box plots indicating many possible outliers.
- Label class imbalance was identified which will impact modelling and scoring in the next section.
- No scale normalization has been completed, this might need to be implemented depending on the models

#### 1.4.19 Bivariate

- Multicollinearity was identified between feature 15 with features 5 & 6. Since 15 has been singled out as a potentially strong predictor, 5 & 6 are to be removed.
- Feature 3 correlates substantially higher than others with the label, this could be a very important predictor. On the other hand, 1-4, 7-10, 12, 14 & 15 scored lowest on label correlation

#### 1.4.20 Multivariate

- Variable interactions were trialed, no substantial new findings were made. Feature 3 interacts with 1, 4 and 6, the churn category is more bunched together than no churn in these cases. No conclusions were drawn over feature 2's interaction with other features, although it was interesting.

#### 1.4.21 Chi-Squared Test

We have plenty of variables which are statistically significant in testing with chi-squared on a generalized linear model.

Feature_3 again proved it's strong significance along with 5, 11 and 13. Features 14 & 15 are still significant, although they rate less highly than we gave them credit for earlier, if we experiment with factors during model development, we might find we can increase their significant that way.

#### 1.4.22 Choice of models and feature filtering

We've seen that there is clearly more than enough information to create a linear model. In the following section we will eliminate features 10 & 7 as their p-values shows there is no statistically significant relationship with the label. we also found multicollinearity between features 5, 6 & 15, therefore features 5 & 6 are to be eliminated.

Based on what we have learned during EDA, our label classes appear to be separable with linear models. In the following section we'll explore Logistic Regression and Linear Discriminant Analysis for modelling.

## 1.5 3. Model Development

## 1.6 3.1 Model 1 - GLM

## 1.7 Feature subset selection

We'll use the step function to select the best subset of variables for our model fit.

[32]:
```
# all features aside from those we determined should be removed earlier - 5, 6,␣
↪7 & 10
fit1.glm <- glm(formula = labels ~ feature_0 + feature_1 + feature_2 +␣
↪feature_3 +
    feature_4 + feature_8 + feature_9 + feature_11 + feature_12 + feature_13 +
    feature_14 + feature_15, family = binomial, data = train)
```

[33]:
```
step1 <- step(fit1.glm)
```

```
Start:  AIC=14864.95
labels ~ feature_0 + feature_1 + feature_2 + feature_3 + feature_4 +
    feature_8 + feature_9 + feature_11 + feature_12 + feature_13 +
    feature_14 + feature_15

             Df Deviance   AIC
<none>            14839 14865
- feature_0   1    14845 14869
- feature_2   1    14847 14871
- feature_1   1    14848 14872
- feature_8   1    14865 14889
- feature_14  1    14866 14890
- feature_9   1    14886 14910
- feature_15  1    14924 14948
- feature_12  1    14954 14978
- feature_4   1    14971 14995
- feature_13  1    15247 15271
- feature_11  1     15268 15292
- feature_3   1     18039 18063
```

AIC is reduced from 17943 to 14731 with these features which is good, it seems that features 0, 1 & 2 are contributing very little, in fact it appears that the first eight features are doing the heavy lifting in this model.

[34]:
```
fit.step1.glm <- glm(formula = labels ~ feature_0 + feature_1 + feature_2 +␣
↪feature_3 + feature_4 +
    feature_8 + feature_9 + feature_11 + feature_12 + feature_13 +
    feature_14 + feature_15, family = binomial, data = train)
```

Function to evaluate our models:

[35]:
```
# we're going to evaluate a few models, so we'll make a function to reuse
# this will only make predictions from the train/validation data / glm models␣
↪(not glmnet)
evaluate <- function(model, true_labels) {
    probs <- predict(model, type = "response")
    pred.glm <- rep("0", length(probs))
    pred.glm[probs > 0.5] <- "1"
    #table(pred.glm, true_labels)
    std_confusion_mat <- confusionMatrix(as.factor(pred.glm), true_labels,␣
↪positive="1")
    std_confusion_mat
}
```

Evaluation:

[36]:
```
evaluate(fit.step1.glm, train$labels)
```

```
Confusion Matrix and Statistics

          Reference
Prediction     0     1
         0 23497  2570
         1   447   612

               Accuracy : 0.8888
                 95% CI : (0.885, 0.8925)
    No Information Rate : 0.8827
    P-Value [Acc > NIR] : 0.0008884

                  Kappa : 0.2443

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.19233
            Specificity : 0.98133
         Pos Pred Value : 0.57790
         Neg Pred Value : 0.90141
             Prevalence : 0.11730
         Detection Rate : 0.02256
   Detection Prevalence : 0.03904
      Balanced Accuracy : 0.58683

       'Positive' Class : 1
```

Our model appears to be performing well at 89% accuracy, however a minimum of 88% accuracy can be expected given the imbalance in the class labels. Looking at the balanced accuracy tells us that maybe it's not as good as we think.

We think there might be something to gain from setting features 7-9 & 13-15 as factors (7-15 were our hypothetical categoric variables, but 10, 11 & 12 aren't required to be set as factors because they are already binary). We'll add them as new features to our glm model and see whether the factor or numeric variable comes out on top after applying the step function.

[37]:
```
fit.factors.glm = glm(labels ~ feature_0 + feature_1 + feature_2 + feature_3 +␣
 ↪feature_4 + feature_11 + feature_12 + feature_13 +
                          as.factor(feature_8)+as.factor(feature_9)+as.
 ↪factor(feature_14)+as.factor(feature_15), data = train, family = binomial)
```

[38]:
```
step2 <- step(fit.factors.glm)
```

```
Start:  AIC=13121.83
labels ~ feature_0 + feature_1 + feature_2 + feature_3 + feature_4 +
    feature_11 + feature_12 + feature_13 + as.factor(feature_8) +
    as.factor(feature_9) + as.factor(feature_14) + as.factor(feature_15)

                        Df Deviance   AIC
- feature_0              1    13068 13122
<none>                        13066 13122
- feature_1              1    13069 13123
- feature_2              1    13070 13124
- as.factor(feature_9)   3    13102 13152
- as.factor(feature_8)   2    13102 13154
- feature_12             1    13106 13160
- feature_4              1    13124 13178
- feature_11             1    13258 13312
- feature_13             1    13347 13401
- as.factor(feature_14) 11    13723 13757
- as.factor(feature_15)  3    13991 14041
- feature_3              1    16346 16400

Step:  AIC=13121.73
labels ~ feature_1 + feature_2 + feature_3 + feature_4 + feature_11 +
    feature_12 + feature_13 + as.factor(feature_8) + as.factor(feature_9) +
    as.factor(feature_14) + as.factor(feature_15)

                        Df Deviance   AIC
<none>                        13068 13122
- feature_1              1    13071 13123
- feature_2              1    13072 13124
- as.factor(feature_9)   3    13102 13150
- as.factor(feature_8)   2    13103 13153
- feature_12             1    13109 13161
- feature_4              1    13126 13178
- feature_11             1    13271 13323
- feature_13             1    13347 13399
- as.factor(feature_14) 11    13729 13761
- as.factor(feature_15)  3    13999 14047
- feature_3              1    16350 16402
```

We've substantially lowered the AIC again from 14865 in our last model to 13122 here. Feature 0 has been dropped as it does not contribute to lowering the AIC by the looks.

The best model and features produced by the step function is:

[39]:
```
fit.step2.glm <- glm(formula = labels ~ feature_1 + feature_2 + feature_3 +␣
↪feature_4 + feature_11 +
   feature_12 + feature_13 + as.factor(feature_8) + as.factor(feature_9) + as.
↪factor(feature_14) +
   as.factor(feature_15), family = binomial, data = train)
```

[40]:
```
anova(fit.glm,fit.step1.glm, fit.step2.glm, test = "Chisq")
```

A anova: 3 × 5

| | Resid. Df <dbl> | Resid. Dev <dbl> | Df <dbl> | Deviance <dbl> | Pr(>Chi) <dbl> |
|---|---|---|---|---|---|
| 1 | 27109 | 14573.44 | NA | NA | NA |
| 2 | 27113 | 14838.95 | -4 | -265.5108 | 2.960573e-56 |
| 3 | 27099 | 13067.73 | 14 | 1771.2221 | 0.000000e+00 |

This is much better, the p-value indicates this model is substantially different from the previous two models.

[41]:
```
evaluate(fit.step2.glm, train$labels)
```

```
Confusion Matrix and Statistics

          Reference
Prediction     0     1
         0 23357  2063
         1   587  1119


               Accuracy : 0.9023
                 95% CI : (0.8987, 0.9058)
```

```
    No Information Rate : 0.8827
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.4095

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.35167
            Specificity : 0.97548
         Pos Pred Value : 0.65592
         Neg Pred Value : 0.91884
             Prevalence : 0.11730
         Detection Rate : 0.04125
   Detection Prevalence : 0.06289
      Balanced Accuracy : 0.66358

       'Positive' Class : 1
```

Our accuracy, sensitivity and specificity are all much improved.

Try a log transform of our most exponential looking skewed distributions:

[42]:
```
fit.log.glm = glm(labels ~ log(feature_1+4) + feature_2 + +log(feature_3+2) +␣
↪log(feature_4+1) + feature_11 +
   feature_12 + feature_13 + as.factor(feature_8) + as.factor(feature_9) + as.
↪factor(feature_14) +
   as.factor(feature_15), data = train, family = binomial)
```

[43]:
```
step3 <- step(fit.log.glm)
```

```
Start:  AIC=12580.01
labels ~ log(feature_1 + 4) + feature_2 + +log(feature_3 + 2) +
    log(feature_4 + 1) + feature_11 + feature_12 + feature_13 +
    as.factor(feature_8) + as.factor(feature_9) + as.factor(feature_14) +
    as.factor(feature_15)

                        Df Deviance   AIC
<none>                        12526 12580
- feature_2              1    12532 12584
- log(feature_1 + 4)     1    12534 12586
- as.factor(feature_9)   3    12563 12611
- as.factor(feature_8)   2    12562 12612
- feature_12             1    12563 12615
- log(feature_4 + 1)     1    12592 12644
- feature_11             1    12718 12770
- feature_13             1    12784 12836
- as.factor(feature_14) 11    13185 13217
- as.factor(feature_15)  3    13438 13486
- log(feature_3 + 2)     1    16334 16386
```

The AIC is once again decreased, so our model is still improving, 13122 down to 12580. Feature 1 & 12 are really not contributing to this model they should be removed.

Best model from step3:

[44]:
```
fit.step3.glm <- glm(formula = labels ~ log(feature_3 + 2) +
    log(feature_4 + 1) + feature_11 + feature_13 + as.factor(feature_8) +
    as.factor(feature_9) + as.factor(feature_14) +
    as.factor(feature_15), family = binomial, data = train)
```

We cannot compare these models with anova, as the log component makes them no longer from the same subset model. Our model is quite complex though and we want to reduce the features further. Although it would go against the AIC rating to say that features 1, 2 & 12 should be removed, they are contributing very little value to our model and there will be some uncertainty in those ratings (with more time we would quantify which features have high enough uncertainty and low enough contribution that we could remove those features), but for today, it is probable that features 1, 2 and 12 are not contributing anything significantly valuable to this model other than extra complexity so we will remove them.

[45]:
```
evaluate(fit.step3.glm, train$labels)
```

```
Confusion Matrix and Statistics

          Reference
Prediction     0     1
         0 23323  2006
         1   621  1176

               Accuracy : 0.9032
                 95% CI : (0.8996, 0.9067)
    No Information Rate : 0.8827
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.4236

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.36958
            Specificity : 0.97406
         Pos Pred Value : 0.65442
         Neg Pred Value : 0.92080
             Prevalence : 0.11730
         Detection Rate : 0.04335
   Detection Prevalence : 0.06625
      Balanced Accuracy : 0.67182
```

```
'Positive' Class : 1
```

While the addition of the logarithmic transformation has left the accuracy and specificity mostly unchanged, this has given us a much needed boost of 2.3% extra sensitivity.

## 1.8 Model 1 Cross Validation

We're going to swap from glm to cv.glmnet here so that cross validation is simpler to implement. The function cv.glm uses k=10 folds by default which suits us well, it also optimises lambda and alpha for us without requiring us to search that space, and it will scale our features which will make out model coefficients more interpretable. Unfortunately, it doesn't have the functionality to create a stratified k-fold which would compensate for our imbalanced label classes, this would be an ideal addition if we had extra time.

As we mentioned earlier, the imbalance in our label classes (churn / no churn) is going to bias the model predictions towards 'no churn' if we only measure accuracy. However, the cv.glmnet function allows us to choose from the following parameters to optimise lambda with, and this gives us a chance to undo some of the effects of imbalance. - Mean square error (mse) - Binomial deviance (deviance) - Misclassification Error (class) - Area under the ROC Curve (auc) - Mean absolute error (mae) - Harrel's concordance (C) - Not suitable for our model

ROC evaluation of a model gives equal weight to correct classification of both classes, which is why it is the best metric for this problem. Therefore, the AUC parameter will be used to optimise and assess the models from here on.

Create a model matrix which will handle the dummy variable conversion for factors.

[46]:
```
#Converting dataset to a model matrix, this will handle our factors for us
train_mat <- model.matrix(labels ~ log(feature_3 + 2) + log(feature_4 + 1) +␣
↪feature_11 + feature_13 +
    as.factor(feature_8) + as.factor(feature_9) + as.factor(feature_14) + as.
↪factor(feature_15), data=train)



test_mat <- model.matrix(labels ~ log(feature_3 + 2) + log(feature_4 + 1) +␣
↪feature_11 + feature_13 +
    as.factor(feature_8) + as.factor(feature_9) + as.factor(feature_14) + as.
↪factor(feature_15), data=test)
```

Fit a cross validated model and optimise lambda and alpha

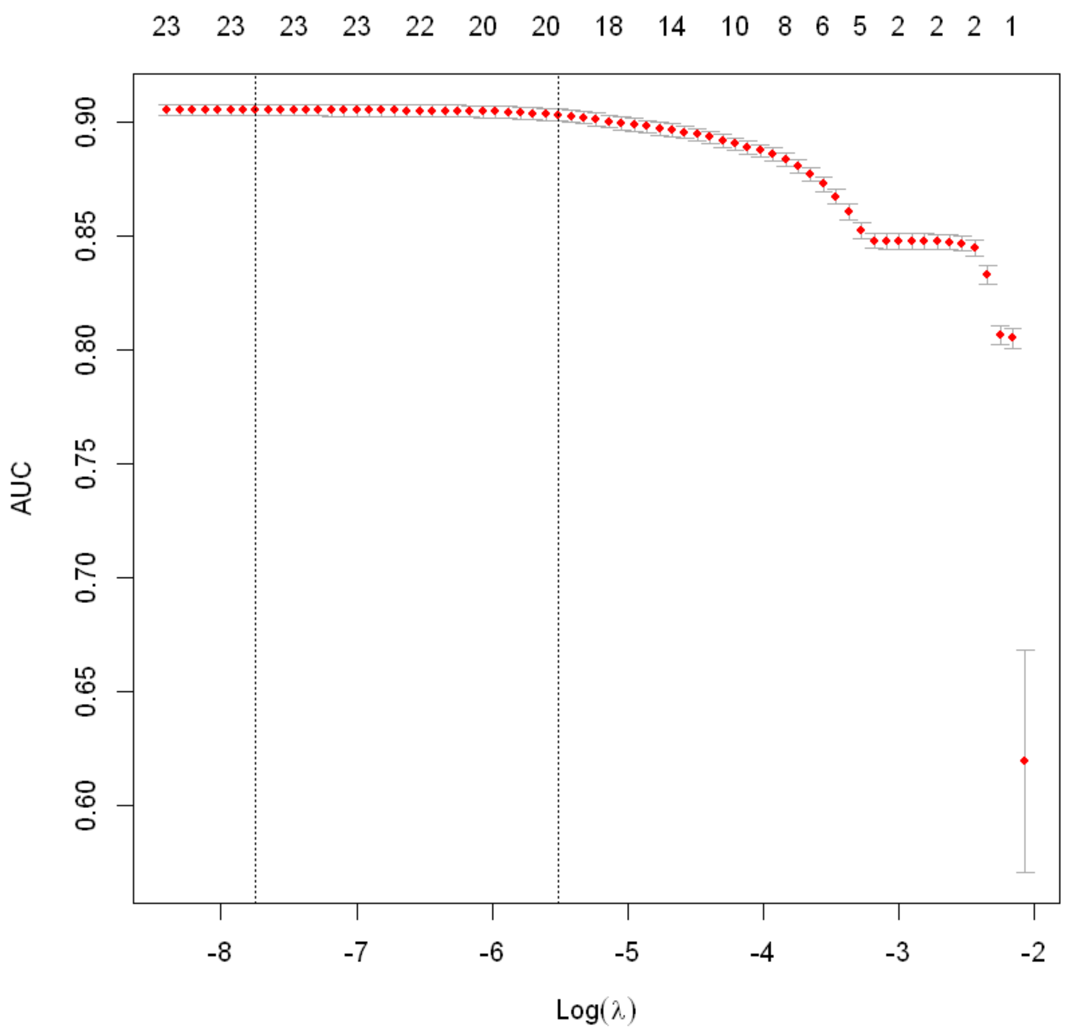
[47]:
```
glm.cvfit = cv.glmnet(train_mat,train$labels, family = "binomial", type.measure␣
↪= "auc", standardize = T)
```

[48]:
```
#return the best lambda
cat("Min lambda on standardized set :",glm.cvfit$lambda.min)
```

```
Min lambda on standardized set : 0.0004335026
```

MSE Plot as a function of log(lambda)

```
[49]: plot(glm.cvfit)
```



```
[74]: cat("As we seein the plot, AUC is maximised at log(lambda) = ",round(log(glm.
       ↪cvfit$lambda.min),2))
```

```
As we seein the plot, AUC is maximised at log(lambda) =  -7.74
```

Below, the coefficients of our model show us that some dummy variables have been eliminated as expected.

```
[51]: coef(glm.cvfit)
```

```
25 x 1 sparse Matrix of class "dgCMatrix"
                                   1
(Intercept)             -3.867844303
(Intercept)              .
log(feature_3 + 2)       3.029270680
log(feature_4 + 1)      -0.271009581
feature_11              -0.629447365
feature_13              -0.489412459
as.factor(feature_8)1   -0.140082525
as.factor(feature_8)2    0.086977859
as.factor(feature_9)1    .
as.factor(feature_9)2    0.198550047
as.factor(feature_9)3    .
as.factor(feature_14)1  -0.202469002
as.factor(feature_14)2   0.867771611
as.factor(feature_14)3   .
as.factor(feature_14)4  -0.492728214
as.factor(feature_14)5  -0.374035036
as.factor(feature_14)6   0.189233245
as.factor(feature_14)7   1.955595396
as.factor(feature_14)8  -0.148487368
as.factor(feature_14)9  -0.283684575
as.factor(feature_14)10  1.230766711
as.factor(feature_14)11  0.928522137
as.factor(feature_15)1   0.003639561
as.factor(feature_15)2   2.305161762
as.factor(feature_15)3  -0.203016418
```

Evaluate the result:

[52]:
```
std_predict <- predict(glm.cvfit, newx = train_mat, s = "lambda.min",type =␣
 ↪"class")
std_probability <- predict(glm.cvfit, newx = train_mat, s = "lambda.min", type␣
 ↪= "response")


std_confusion_mat <- confusionMatrix(as.factor(std_predict), as.
 ↪factor(train$labels), positive="1")
std_confusion_mat
```

```
Confusion Matrix and Statistics

          Reference
Prediction     0     1
         0 23327  2021
         1   617  1161
```

```
               Accuracy : 0.9028
                 95% CI : (0.8992, 0.9063)
    No Information Rate : 0.8827
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.4193

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.36486
            Specificity : 0.97423
         Pos Pred Value : 0.65298
         Neg Pred Value : 0.92027
             Prevalence : 0.11730
         Detection Rate : 0.04280
   Detection Prevalence : 0.06555
      Balanced Accuracy : 0.66955

       'Positive' Class : 1
```

Our results show that the model is 90.3% accurate however, our Sensitivity (True Positive Rate) is quite low at 0.36, meaning we're only detecting 36% of our churn cases. We probably want to shift our threshold so that we can detect as many as possible.

Assume that the insurance company has specified that they want to detect 90% of churn cases, no matter the cost of accidentally chasing down false positives. By shifting the probability threshold, we can increase the sensitivity to 90%.

[53]:
```r
# threshold for adjusting sensitivity.
t <- 0.078

# convert probabilities to predictions
predict_binary <- ifelse(std_probability > t, 1, 0)
CM <- confusionMatrix(as.factor(predict_binary), as.factor(train$labels),␣
 ↪positive="1")
CM
```

```
Confusion Matrix and Statistics

          Reference
Prediction     0     1
         0 18066   313
         1  5878  2869

               Accuracy : 0.7718
                 95% CI : (0.7667, 0.7768)
    No Information Rate : 0.8827
    P-Value [Acc > NIR] : 1

                  Kappa : 0.3732

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.9016
            Specificity : 0.7545
         Pos Pred Value : 0.3280
         Neg Pred Value : 0.9830
             Prevalence : 0.1173
         Detection Rate : 0.1058
   Detection Prevalence : 0.3225
      Balanced Accuracy : 0.8281

       'Positive' Class : 1
```

The cost of setting the churn threshold so high is that our accuracy has dropped to 77% (although balanced accuracy is much improved at 82%) and the specificity to 75%, meaning that 25% of customers who do not plan to leave are predicted to do so. This could amount to a huge amount of work for a marketing team to reach out to so many customers, in which case you might refine that threshold a bit more depending on the preventative measures you wish to implement.

### 1.9 3.2 Model 2 - LDA

Fit the LDA model to all features:

```
[54]: fit.lda = lda(labels ~ ., data = train)
      fit.lda
```

```
Call:
lda(labels ~ ., data = train)

Prior probabilities of groups:
        0         1
0.8826956 0.1173044

Group means:
    feature_0   feature_1   feature_2  feature_3   feature_4   feature_5
0 -0.01055885 -0.01687312  0.01452347 -0.1441421  0.02796778 -0.04315315
1  0.03761570  0.13836133 -0.07790341  1.0639063 -0.21085256  0.29960662
    feature_6 feature_7 feature_8 feature_9  feature_10 feature_11 feature_12
0 -0.04359262  4.292516  1.160750  1.206774 0.019044437  0.5768042 0.16764116
1  0.25041530  4.665933  1.240729  1.367065 0.009113765  0.3670647 0.09365179
  feature_13 feature_14 feature_15
0  0.6849733   5.541221   2.591338
1  0.2721559   5.304840   2.344123
```

```
Coefficients of linear discriminants:
                    LD1
feature_0   0.040904031
feature_1   0.052893029
feature_2  -0.030310505
feature_3   0.943193304
feature_4  -0.077309868
feature_5   0.346286763
feature_6   0.275782235
feature_7   0.004894059
feature_8   0.125733950
feature_9   0.132972701
feature_10 -0.169616378
feature_11 -0.629516904
feature_12 -0.345676579
feature_13 -0.287418092
feature_14  0.034465294
feature_15  0.283324726
```

[55]:
```
pred.lda = predict(fit.lda, train)
confusionMatrix(pred.lda$class, train$labels, positive="1")
```

```
Confusion Matrix and Statistics

          Reference
Prediction     0     1
         0 23227  2256
         1   717   926

               Accuracy : 0.8904
                 95% CI : (0.8866, 0.8941)
    No Information Rate : 0.8827
    P-Value [Acc > NIR] : 3.593e-05

                  Kappa : 0.3303

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.29101
            Specificity : 0.97006
         Pos Pred Value : 0.56360
         Neg Pred Value : 0.91147
             Prevalence : 0.11730
         Detection Rate : 0.03414
   Detection Prevalence : 0.06057
      Balanced Accuracy : 0.63053
```

```
        'Positive' Class : 1
```

As we found with logistic regression, the lda model trained on all features makes a biased prediction towards the majority label class. As they are both linear models, we suspect the same features that were chosen by the step function will perform well on an LDA model. The features from step3 are used below:

```
[56]: fit.step3.lda <- lda(formula = labels ~ log(feature_3 + 2) + log(feature_4 + 1)␣
      ↪+ feature_11 + feature_13 + as.factor(feature_8) +
          as.factor(feature_9) + as.factor(feature_14) + as.factor(feature_15),␣
      ↪family = binomial, data = train)

      fit.step3.lda
```

```
Call:
lda(labels ~ log(feature_3 + 2) + log(feature_4 + 1) + feature_11 +
    feature_13 + as.factor(feature_8) + as.factor(feature_9) +
    as.factor(feature_14) + as.factor(feature_15), data = train,
    family = binomial)

Prior probabilities of groups:
        0         1
0.8826956 0.1173044

Group means:
  log(feature_3 + 2) log(feature_4 + 1) feature_11 feature_13
0          0.5539135         -0.2236684  0.5768042  0.6849733
1          1.0176981         -0.4038181  0.3670647  0.2721559
  as.factor(feature_8)1 as.factor(feature_8)2 as.factor(feature_9)1
0             0.6159790             0.2723856             0.5178333
1             0.5229415             0.3588938             0.4569453
  as.factor(feature_9)2 as.factor(feature_9)3 as.factor(feature_14)1
0             0.2838289            0.04042766              0.1396174
1             0.3871779            0.04525456              0.1260214
  as.factor(feature_14)2 as.factor(feature_14)3 as.factor(feature_14)4
0            0.002464083             0.05663214             0.03324424
1            0.018227530             0.08045255             0.02796983
  as.factor(feature_14)5 as.factor(feature_14)6 as.factor(feature_14)7
0              0.1574507              0.1200718            0.005763448
1              0.1228787              0.1037084            0.047454431
  as.factor(feature_14)8 as.factor(feature_14)9 as.factor(feature_14)10
0              0.3177414             0.09041931              0.01023221
1              0.1766185             0.07573853              0.06002514
  as.factor(feature_14)11 as.factor(feature_15)1 as.factor(feature_15)2
0             0.007976946             0.03788005              0.01290511
```

```
1              0.047768699              0.05782527              0.19044626
  as.factor(feature_15)3
0               0.8425493
1               0.6351351

Coefficients of linear discriminants:
                                LD1
log(feature_3 + 2)       2.12655462
log(feature_4 + 1)      -0.04783205
feature_11              -0.37579447
feature_13              -0.25044702
as.factor(feature_8)1   -0.10036163
as.factor(feature_8)2    0.05885736
as.factor(feature_9)1    0.03885763
as.factor(feature_9)2    0.19611312
as.factor(feature_9)3    0.09511386
as.factor(feature_14)1  -0.56380858
as.factor(feature_14)2   1.00160806
as.factor(feature_14)3  -0.24368863
as.factor(feature_14)4  -0.76788010
as.factor(feature_14)5  -0.56026676
as.factor(feature_14)6  -0.06283886
as.factor(feature_14)7   1.87839387
as.factor(feature_14)8  -0.35078391
as.factor(feature_14)9  -0.53373256
as.factor(feature_14)10  1.08874182
as.factor(feature_14)11  0.89922259
as.factor(feature_15)1   0.19243804
as.factor(feature_15)2   2.94802511
as.factor(feature_15)3  -0.03345371
```

[57]:
```
pred.lda = predict(fit.step3.lda, train)
confusionMatrix(pred.lda$class, train$labels, positive="1")
```

```
Confusion Matrix and Statistics

          Reference
Prediction     0     1
         0 23097  1824
         1   847  1358

               Accuracy : 0.9015
                 95% CI : (0.8979, 0.9051)
    No Information Rate : 0.8827
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.4515
```

```
Mcnemar's Test P-Value : < 2.2e-16

           Sensitivity : 0.42678
           Specificity : 0.96463
        Pos Pred Value : 0.61587
        Neg Pred Value : 0.92681
            Prevalence : 0.11730
        Detection Rate : 0.05006
  Detection Prevalence : 0.08129
     Balanced Accuracy : 0.69570

      'Positive' Class : 1
```

The sensitivity of this model is better than the logit model (while it had unchanged probability threshold), but the down side of this model is that we cannot adjust the probabilities and choose to preference the minority label class.

## 1.10 4. Model Scoring on Test Data

Below we score the model on our test data using accuracy, sensitivity and specificity, as well as boostrap standard error for uncertainty.

## 1.11 Model 1 - GLM

Below our best outcome for a general linear model is shown at two probability thresholds.

[58]:
```
std_predict <- predict(glm.cvfit, newx = test_mat, s = "lambda.min",type =␣
 ↪"class")
std_probability <- predict(glm.cvfit, newx = test_mat, s = "lambda.min", type =␣
 ↪"response")

# threshold at 0.5
cat("Performance metrics for predictions with default probability threshold at␣
 ↪0.5:")
confusionMatrix(as.factor(std_predict), as.factor(test$labels), positive="1")

# threshold for adjusting sensitivity.
t <- 0.076

# convert probabilities to predictions
predict_binary <- ifelse(std_probability > t, 1, 0)

cat("Performance metrics for predictions with adjusted probability threshold at␣
 ↪0.084, to increase senseitivity to 90%:")
confusionMatrix(as.factor(predict_binary), as.factor(test$labels), positive="1")
```

Performance metrics for predictions with default probability threshold at 0.5:

```
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 5837  499
         1  160  286

               Accuracy : 0.9028
                 95% CI : (0.8955, 0.9098)
    No Information Rate : 0.8843
    P-Value [Acc > NIR] : 5.381e-07

                  Kappa : 0.4157

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.36433
            Specificity : 0.97332
         Pos Pred Value : 0.64126
         Neg Pred Value : 0.92124
             Prevalence : 0.11575
         Detection Rate : 0.04217
   Detection Prevalence : 0.06576
      Balanced Accuracy : 0.66883

       'Positive' Class : 1
```

Performance metrics for predictions with adjusted probability threshold at 0.084, to increase senseitivity to 90%:

```
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 4479   77
         1 1518  708

               Accuracy : 0.7648
                 95% CI : (0.7545, 0.7749)
    No Information Rate : 0.8843
    P-Value [Acc > NIR] : 1

                  Kappa : 0.3609

 Mcnemar's Test P-Value : <2e-16
```

```
            Sensitivity : 0.9019
            Specificity : 0.7469
         Pos Pred Value : 0.3181
         Neg Pred Value : 0.9831
             Prevalence : 0.1157
         Detection Rate : 0.1044
   Detection Prevalence : 0.3282
      Balanced Accuracy : 0.8244

       'Positive' Class : 1
```

### 1.11.1 GLM Bootstrap error:

[59]:
```
# boot function to recalculate the model based on each sample of the dataset
boot.fn <- function(data, index) {
    fit <- glm(formula = labels ~ log(feature_3 + 2) + log(feature_4 + 1) +␣
 ↪feature_11 + feature_13 + as.factor(feature_8) +
    as.factor(feature_9) + as.factor(feature_14) + as.factor(feature_15),␣
 ↪family = binomial, data = train, subset = index)
    return (coef(fit))
}
```

[60]:
```
#  set the index for a sample
index <- sample(dim(train)[1], dim(train)[1] / 2)
```

Below we use the bootstrap function to estimate the standard error of our feature coefficients. It's a bit difficult to interpret because there are so many dummy variables, we'll provide an index of them below.

[61]:
```
set.seed(1234)
myBootstrap<-boot(train, boot.fn, 1000)
myBootstrap
```

```
ORDINARY NONPARAMETRIC BOOTSTRAP


Call:
boot(data = train, statistic = boot.fn, R = 1000)


Bootstrap Statistics :
        original        bias    std. error
t1*  -3.82206420  0.0001550054  0.15157035
t2*   3.30726851  0.0066620298  0.06526583
t3*  -0.35474907  0.0010815289  0.04865484
```

```
t4*  -0.78698635 -0.0080512107  0.06151141
t5*  -0.65896554 -0.0011652724  0.04845240
t6*  -0.21439437 -0.0018220369  0.07483212
t7*   0.09375067 -0.0025629204  0.07922197
t8*   0.17455802  0.0004604736  0.07803962
t9*   0.44677077 -0.0001494480  0.08439055
t10*  0.27081944 -0.0030043907  0.14176933
t11* -0.83460618 -0.0042316895  0.11662458
t12*  0.78935311 -0.0131068744  0.29648866
t13* -0.36244694 -0.0050007475  0.12216189
t14* -1.33573113 -0.0083085196  0.16332429
t15* -0.99772682 -0.0043098425  0.10886985
t16*  0.10249986 -0.0031451141  0.13447534
t17*  1.74516117 -0.0033918293  0.17544373
t18* -0.58491139 -0.0011354831  0.09612413
t19* -0.95367508 -0.0021605571  0.11672563
t20*  1.00058460 -0.0024484466  0.17452304
t21*  0.68201633 -0.0042938348  0.17432429
t22*  0.19170911 -0.0028642456  0.12725411
t23*  2.49166885  0.0124260667  0.11099802
t24* -0.13320378 -0.0006781575  0.08004287
```

Features in t12 has the highest uncertainty, this corresponds to feature feature_14 category_2.

Features in t2, t3 t4, t5 have the lowest uncertainty, these respectively correspond to: - feature_3 - feature_4 - feature_11 - feature_13

See below to help us understand which row in the above statistics corresponds to which variable:

[62]:
```
tibble::rowid_to_column(as.data.frame(colnames(train_mat)), "ID")
```

A data.frame: 24 × 2

| ID | colnames(train_mat) |
|---|---|
| <int> | <fct> |
| 1 | (Intercept) |
| 2 | log(feature_3 + 2) |
| 3 | log(feature_4 + 1) |
| 4 | feature_11 |
| 5 | feature_13 |
| 6 | as.factor(feature_8)1 |
| 7 | as.factor(feature_8)2 |
| 8 | as.factor(feature_9)1 |
| 9 | as.factor(feature_9)2 |
| 10 | as.factor(feature_9)3 |
| 11 | as.factor(feature_14)1 |
| 12 | as.factor(feature_14)2 |
| 13 | as.factor(feature_14)3 |
| 14 | as.factor(feature_14)4 |
| 15 | as.factor(feature_14)5 |
| 16 | as.factor(feature_14)6 |
| 17 | as.factor(feature_14)7 |
| 18 | as.factor(feature_14)8 |
| 19 | as.factor(feature_14)9 |
| 20 | as.factor(feature_14)10 |
| 21 | as.factor(feature_14)11 |
| 22 | as.factor(feature_15)1 |
| 23 | as.factor(feature_15)2 |
| 24 | as.factor(feature_15)3 |

## 1.12 Model 2 - LDA

[63]:
```
pred.lda = predict(fit.step3.lda, test)
confusionMatrix(pred.lda$class, test$labels, positive="1")
```

```
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 5782  454
         1  215  331

               Accuracy : 0.9014
                 95% CI : (0.894, 0.9084)
    No Information Rate : 0.8843
    P-Value [Acc > NIR] : 3.753e-06

                  Kappa : 0.4446

 Mcnemar's Test P-Value : < 2.2e-16
```

```
         Sensitivity : 0.42166
         Specificity : 0.96415
      Pos Pred Value : 0.60623
      Neg Pred Value : 0.92720
          Prevalence : 0.11575
      Detection Rate : 0.04881
Detection Prevalence : 0.08051
   Balanced Accuracy : 0.69290

    'Positive' Class : 1
```

This model has a higher sensitivity than the final glm model with default probability threshold. The accuracy is a smidge lower but this is not a material difference. However, the fact that the glm model gives us the ability to shift the bar in order to manipulate the sensitivity score is an unquestionable advantage in a problem such as this. A marketing team might be able to devise a staged response to customers based on their risk level, those with the greatest probability of leaving may warrant greater intervention measures.

### 1.12.1 LDA Bootstrap error

Redefine the boot function for lda.

[64]:
```
boot.fn <- function(data, index) {
    fit <- lda(formula = labels ~ log(feature_3 + 2) + log(feature_4 + 1) +␣
 ↪feature_11 + feature_13 + as.factor(feature_8) +
    as.factor(feature_9) + as.factor(feature_14) + as.factor(feature_15),␣
 ↪family = binomial, data = train, subset = index)
    return (coef(fit))
}
```

Estimate bootstrap error:

[65]:
```
myBootstrap<-boot(train, boot.fn, 1000)
myBootstrap
```

```
ORDINARY NONPARAMETRIC BOOTSTRAP


Call:
boot(data = train, statistic = boot.fn, R = 1000)


Bootstrap Statistics :
        original          bias    std. error
t1*   2.12655462 -0.0030945375  0.03587594
t2*  -0.04783205  0.0003051808  0.01781525
t3*  -0.37579447  0.0014629635  0.02797130
```

```
t4*  -0.25044702  0.0004037174  0.01975007
t5*  -0.10036163  0.0010686944  0.03599078
t6*   0.05885736  0.0009478020  0.03958033
t7*   0.03885763 -0.0008607981  0.02953705
t8*   0.19611312 -0.0002622491  0.03510479
t9*   0.09511386 -0.0029170547  0.06186013
t10* -0.56380858 -0.0024444100  0.07238720
t11*  1.00160806  0.0084858498  0.29210337
t12* -0.24368863 -0.0034461956  0.08267755
t13* -0.76788010  0.0012698246  0.08515351
t14* -0.56026676 -0.0013078624  0.06680245
t15* -0.06283886 -0.0005732603  0.07668322
t16*  1.87839387  0.0073209980  0.19332453
t17* -0.35078391 -0.0015655579  0.06503386
t18* -0.53373256 -0.0011389856  0.06940129
t19*  1.08874182  0.0069921209  0.15843164
t20*  0.89922259  0.0035119118  0.17319399
t21*  0.19243804  0.0005583861  0.07758724
t22*  2.94802511 -0.0006413273  0.10472908
t23* -0.03345371 -0.0000948979  0.04687941
```

There's one less value in this as we no longer have an intercept for lda, so the feature names are offset by one from before.

Uncertainty in this model is very similar to the logistic regression model, although the LD1 coefficients are produced and scaled differently to the logit model, so perhaps they are not comparable in this way. Our features with highest uncertainty again appear to be in feature 14, our lowest uncertainty is in t4 (feature 4), accompanied by t2, t3 & t4. These correspond to features 4, 11 & 13.

Index of feature names in bootstrap uncertainty table:

[66]:
```
tibble::rowid_to_column(as.data.frame(colnames(train_mat)[-1]), "ID")
```

A data.frame: 23 × 2

| ID | colnames(train_mat)[-1] |
|---|---|
| <int> | <fct> |
| 1 | log(feature_3 + 2) |
| 2 | log(feature_4 + 1) |
| 3 | feature_11 |
| 4 | feature_13 |
| 5 | as.factor(feature_8)1 |
| 6 | as.factor(feature_8)2 |
| 7 | as.factor(feature_9)1 |
| 8 | as.factor(feature_9)2 |
| 9 | as.factor(feature_9)3 |
| 10 | as.factor(feature_14)1 |
| 11 | as.factor(feature_14)2 |
| 12 | as.factor(feature_14)3 |
| 13 | as.factor(feature_14)4 |
| 14 | as.factor(feature_14)5 |
| 15 | as.factor(feature_14)6 |
| 16 | as.factor(feature_14)7 |
| 17 | as.factor(feature_14)8 |
| 18 | as.factor(feature_14)9 |
| 19 | as.factor(feature_14)10 |
| 20 | as.factor(feature_14)11 |
| 21 | as.factor(feature_15)1 |
| 22 | as.factor(feature_15)2 |
| 23 | as.factor(feature_15)3 |

## 2 Best model selection

Below we compare the same glm model at two different probability thresholds against the LDA model.

Bootstrap uncertainty in all models are very similar.

Accuracy of the LDA and GLM (t=0.5) were very similar:
GLM (t=0.5) Accuracy : 0.90
GLM (t=0.076) Accuracy : 0.76
LDA Accuracy : 0.90

Sensitivity is a very imporatnt parameter for this problem, GLM (t=0.076) is the stand out winner:
GLM (t=0.5) Sensitivity : 0.36
GLM (t=0.076) Sensitivity : 0.90
LDA Sensitivity : 0.42

Specificity GLM (t=0.5) is slightly ahead of LDA, while GLM (t=0.076) is poor:
GLM (t=0.5) Specificity : 0.97
GLM (t=0.076) Specificity : 0.75
LDA Specificity : 0.96

GLM (t=0.076) is far better than the other two:
GLM (t=0.5) Balanced Accuracy : 0.67

GLM (t=0.076) Balanced Accuracy : 0.82
LDA Balanced Accuracy : 0.69

The best model we have developed for this problem is the GLM model (glm.cvfit), the lda lacks the flexibility needed to get high enough sensitivity that churn customers won't slip through the gaps.

## 2.1 5. Model Interpretation and Inference

The benefit of using linear models are their simpler interpretability. The feature coefficients allow us to infer the meaning of our models as they describe the change in log odds of churn occurring for one every degree of change in each feature.

The log odds describes the probability of churn occurring, where the relationship is log(p/(1-p)), p=probability. So as the probability of an event occurring increases, so does the logit.

We can also factor in the uncertainty of the log odds from the previous section where calculated bootstrap error estimates on the coefficients.

The model coefficients:

[67]:
```
coef(glm.cvfit)
```

```
25 x 1 sparse Matrix of class "dgCMatrix"
                                   1
(Intercept)             -3.867844303
(Intercept)              .
log(feature_3 + 2)       3.029270680
log(feature_4 + 1)      -0.271009581
feature_11              -0.629447365
feature_13              -0.489412459
as.factor(feature_8)1   -0.140082525
as.factor(feature_8)2    0.086977859
as.factor(feature_9)1    .
as.factor(feature_9)2    0.198550047
as.factor(feature_9)3    .
as.factor(feature_14)1  -0.202469002
as.factor(feature_14)2   0.867771611
as.factor(feature_14)3   .
as.factor(feature_14)4  -0.492728214
as.factor(feature_14)5  -0.374035036
as.factor(feature_14)6   0.189233245
as.factor(feature_14)7   1.955595396
as.factor(feature_14)8  -0.148487368
as.factor(feature_14)9  -0.283684575
as.factor(feature_14)10  1.230766711
as.factor(feature_14)11  0.928522137
as.factor(feature_15)1   0.003639561
as.factor(feature_15)2   2.305161762
as.factor(feature_15)3  -0.203016418
```

The above coefficients tell us that our strongest predictors are feature_3 with a coefficient of 2.98+/-0.07, and feature_15:category_2 with a coefficient of 2.29+/-0.11. They tell us that customers which rate more highly in feature 3 characteristics are more likely to churn, in fact, for every one degree of increase in feature 3 the log odds of the customers likelihood to churn will increase by 2.91 - 3.05. Additionally, customers who are present in category 2 of feature 15 are also more likely to churn, specifically the log odds of them churning is between 2.17 - 2.39 times higher than someone not in that category.

Similarly, customers who are present in feature_14 category_7 and category 10 have a high likelihood of churn.

On the other hand, the lowest coefficients are feature 11 & 13. For every degree increase in a customer's standing within feature 11 & 13, their log odds of churning decreases by between 0.54-0.66 and 0.41-0.51 respectively.

## 2.2 5.1 Variable Importance

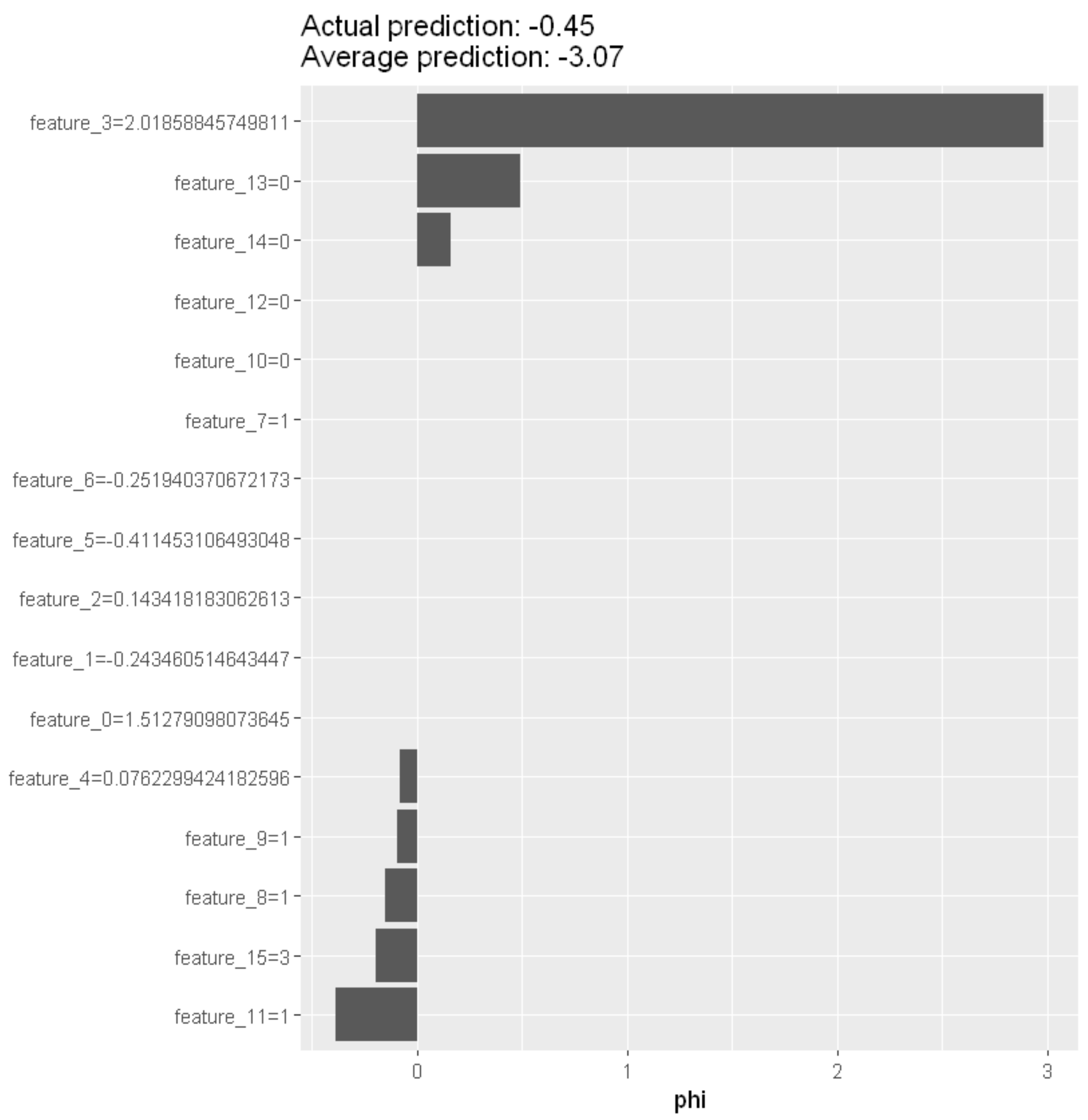
We look at shapely values to compare each variables importance in our model. Shapely values show us feature importance by assessing the increase in error after rearranging the model several times and provide the aggregate global feature importance (Raoniar, 2020). We'll discuss the final model and also compare it to our initial model that included all variables, to see how we did with feature selection.

Calculate shapely values for our model:

[68]:
```
#glm.cvfit can't be used here, so we'll recreate it with glm
fit.glm.final <- glm(labels ~ log(feature_3 + 2) + log(feature_4 + 1) +␣
 ↪feature_11 + feature_13 +
    as.factor(feature_8) + as.factor(feature_9) + as.factor(feature_14) + as.
 ↪factor(feature_15), data = test, family = binomial)



X <- train[-which(names(train) == "labels")] # label data
mod <- Predictor$new(fit.glm.final, data = X) # holding the machine learning␣
 ↪model and the data

# Then we explain the first instance of the dataset with the Shapley method:
x.interest <- X[1, ]
shapley.glm.final <- Shapley$new(mod, x.interest = x.interest)
```

Plot the data:

[69]:
```
plot(shapley.glm.final)
```



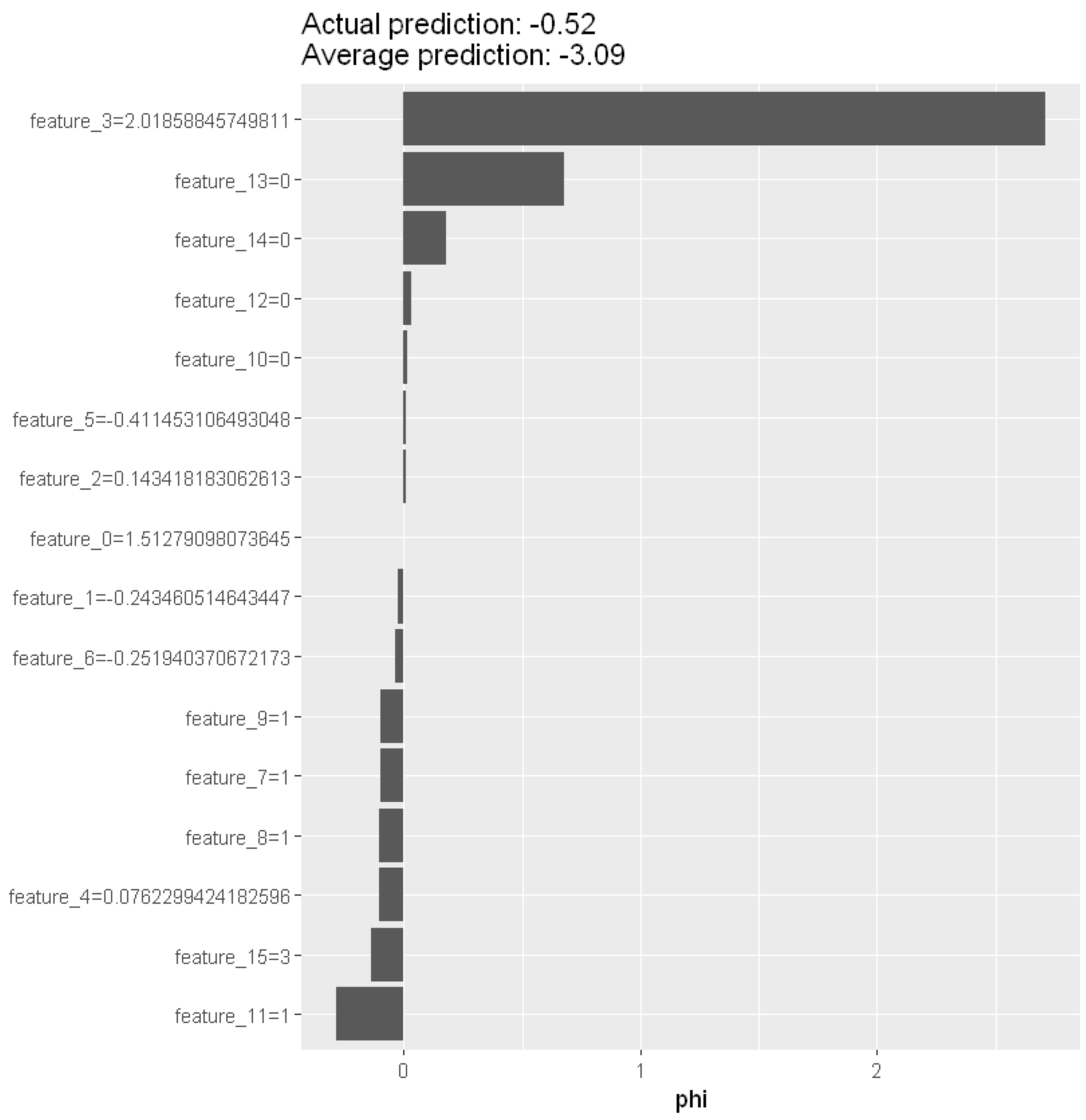
As we have said from the start, feature 3 is our most important, followed by 13, 14 and 11.

Below we regenerate the original glm model, using all features but with dummy variables and log functions as we know they were the best form for those variables.

[70]:
```
#glm.cvfit can't be used here, so we'll recreate it with glm
fit.glm.initial <- glm(labels ~ log(feature_1+4) + feature_2 + log(feature_3 +␣
↪2) + log(feature_4 + 1) + feature_5 + feature_6 +
                       feature_7 + as.factor(feature_8) + as.factor(feature_9) +␣
↪feature_10 + feature_11 + feature_12 +
                       feature_13 +  as.factor(feature_14) + as.
↪factor(feature_15), data = test, family = binomial)
```

```
X <- train[-which(names(train) == "labels")] # label data
mod <- Predictor$new(fit.glm.initial, data = X) # holding the machine learning␣
 ↪model and the data

# Then we explain the first instance of the dataset with the Shapley method:
x.interest <- X[1, ]
shapley.glm.initial <- Shapley$new(mod, x.interest = x.interest)
```

[71]:
```
plot(shapley.glm.initial)
```



It seems our selection of features is in agreeance with the model containing all features. Those with the least importance have all been removed from our final model, such as feature 0, 1 & 10 in our first iteration, followed by further complexity reductions by elimination of 7, 6 12 and 5.

Perhaps for a less complex model, we might have improved things again by only including 3, 13, 14 & 11.

It's possible we misjudged the importance of feature 6, which we eliminated due to it's high multi-collinearity with feature 5 & 15, the later of which was the one retained in the model. Feature 6 is ranked of higher importance than feature 15 and would have reduced the model complexity since feature 15 was split into three dummy variables.

## 2.3 7. Marketing Suggestions

As our model has some inaccuracy when targeting customers who we suspect may leave, marketing teams should approach their customers differently depending on how high their risk of churn is.

If the probability threshold is set at 0.7, we are highly certain that any customers who are predicted to leave are going to do so, and therefore, more aggressive or expensive marketing strategies should consider for that cohort. There will still be ~1% of those predicted to leave who will be caught up unnecessarily in the marketing strategy, but this a small cost for potentially keeping the 20% of customers who are predicted to leave.

A second cohort should be considered for a softer marketing strategy. By reducing the probability threshold to 0.46 we can double the number of customers that are targeted because we expect may want to leave, however now 3% of happy customers will be caught up who may not want to be bothered by a marketing team. You don't want to bother happy customers with extra marketing too much, because then you might make them unhappy.

This can continue with as many cohorts as they want.

We might also suggest to the company that when marketing to new customers, they could have a more reliable customer base if they target the parameters we discussed in the inference section, such as if they have characteristics that align with low values of feature 3 they will be less likely to leave immediately. This below can be used to test the thresholds suggested above:

This below can be used to test the thresholds suggested above:

[72]:
```
std_probability <- predict(glm.cvfit, newx = test_mat, s = "lambda.min", type =␣
 ↪"response")

# threshold for adjusting sensitivity.
#t <- 0.7
t <- 0.46

# convert probabilities to predictions
predict_binary <- ifelse(std_probability > t, 1, 0)

cat("Performance metrics for predictions with adjusted probability threshold at␣
 ↪0.084, to increase senseitivity to 90%:")
confusionMatrix(as.factor(predict_binary), as.factor(test$labels), positive="1")
```

```
Performance metrics for predictions with adjusted probability threshold at
0.084, to increase senseitivity to 90%:
```

```
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 5814  471
         1  183  314

               Accuracy : 0.9036
                 95% CI : (0.8963, 0.9105)
    No Information Rate : 0.8843
    P-Value [Acc > NIR] : 1.917e-07

                  Kappa : 0.4396

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.40000
            Specificity : 0.96948
         Pos Pred Value : 0.63179
         Neg Pred Value : 0.92506
             Prevalence : 0.11575
         Detection Rate : 0.04630
   Detection Prevalence : 0.07328
      Balanced Accuracy : 0.68474

       'Positive' Class : 1
```

## 2.4 8. Conclusion

## 2.5 EDA

Many different graphical and statistical comparisons are made between one, two and three variables. It is immediately clear that feature 3 is significant. We suspect that features 7-15 may be best treated as if they are categoric. The distributions are analysed and significant imbalance is found the response variable.

## 2.6 Model Development

First features are filtered out based on their multicollinearity and their low significance in the initial glm model, this resulted in the removal of features 5, 6, 7 & 10. Following this, the step function was used to iteratively reduce the AIC score and by removing features 0, 1 & 2 for their very low AIC scored contribution (with more time we would have confirmed the AIC uncertainty before assuming their AIC contribution was basically within the noise of the problem). Next it was found that the use of dummy variables on our suspected categoric features and logarithmic transforms on our highly skewed features was beneficial to the AIC as well. At the end, the model was still more complex than it should be for easy interpretation, however it was performing better and further reductions resulted in loss of accuracy or sensitivity.

## 2.7 Model Scoring on Test Data

Models are scored on the test data by comparing their sensitivity, specificity, bootstrap standard error of coefficients and their flexibility. It was found that the logistic regression model's ability to flexibly adjust the probability threshold was a major advantage which would prevent customers from slipping through the gaps caused by low sensitivity.

## 2.8 Model Interpretation and Inference

To interpret the model, the coefficients are compared along with their uncertainty. Feature_3 with a coefficient of 2.98+/-0.07 tells us that customers which rate more highly in feature 3 characteristics are more likely to churn and feature_15:category_2 with a coefficient of 2.29+/-0.11, meaning that customers who are present in category 2 of feature 15 are also more likely to churn.

On the other hand, the lowest coefficients are feature 11 & 13. For every degree increase in a customers standing within feature 11 & 13, their log odds of churning decreases by between 0.54-0.66 and 0.41-0.51 respectively.

## 2.9 Variable Importance

In this section we use shapely values to compare out final model against the importance of all features from our initial model. Our findings here reinforce that good decisions were made with regard to feature selection. Feature 3 is a very strong feature along with 11, 13 & 14. This subset may have made a better simpler model had we continued to reduce the model complexity further. The weakest features, 0, 10 and 2 were all removed along with others.

## 2.10 Marketing Suggestions

Marketing suggestion is made to approach different cohorts of customers with varying degrees of aggressiveness in marketing strategy based on the probability that they customer is going to leave.

## 2.11 9. References

Raoniar, R. (September 26, 2020). Machine Learning Model Explanation using Shapley Values. https://onezero.blog/machine-learning-model-explanation-using-shapley-values/