Insurance Customer Churn Prediction Model (Part 1)

Introduction

A churn model, also known as a customer attrition model, is a predictive model used by businesses to identify customers who are likely to discontinue using the products or services. It helps businesses take proactive measures to retain customers and mitigate the negative impact of churn.

Here I demonstrate the steps of building up a churn prediction model using a synthetic insurance customer dataset that mimic real-world dirty data. The train data contains a binary target variable (0: staying, 1: churn) and 100 de-identified features for 40K customers. After building a binary classifier, I will generate churn predictions for 10K customers in the test data.

As a modeling strategy, I will consider 5 candidate feature sets that are chosen using logistic regression with LASSO penalty. For each candidate feature sets, I will apply logistic regression, multilayer perceptron, random forest, and decision tree models, searching for an optimal model showing the best prediction performance in the validation data in terms of AUC measures.

Load packages

```
library(tidyverse)
library(tictoc)
library(pheatmap)
library(moments)
library(impute)
library(pROC)
library(caret)
library(sparklyr)
library(glmnet)
library(tensorflow)
library(keras)
```

Step 1 - Clean and prepare data

1-1. Glimpse of raw data

```
## Load raw csv files
raw_train <- read_csv(file.path('churnPrediction_train.csv'))
raw_test <- read_csv(file.path('churnPrediction_test.csv'))
head(raw_train)
```

```
## # A tibble: 6 x 101
##       y    x1    x2 x3         x4      x5     x6 x7        x8    x9    x10   x11
##   <dbl> <dbl> <dbl> <chr>   <dbl>   <dbl>  <dbl> <chr>  <dbl> <dbl>  <dbl> <dbl>
```

```
## 1     0 0.165  18.1 Wed     1.08  -1.34   -1.58  0.00~  0.221 1.82   1.17  110.
## 2     1 2.44   18.4 Friday  1.48   0.921  -0.760 0.00~  1.19  3.51   1.42   84.1
## 3     1 4.43   19.2 Thursd~ 0.146  0.366   0.710 -8e-~  0.952 0.783 -1.25   95.4
## 4     0 3.93   19.9 Tuesday 1.76  -0.252  -0.827 -0.0~ -0.521 1.83   2.22   96.4
## 5     0 2.87   22.2 Sunday  3.41   0.0832  1.38  0.01~ -0.733 2.15  -0.275  90.8
## 6     0 1.93   19.5 Saturd~ 3.70   2.30   -0.331 0.00~  3.09  0.473  1.94   90.5
## # ... with 89 more variables: x12 <dbl>, x13 <dbl>, x14 <dbl>, x15 <dbl>,
## #   x16 <dbl>, x17 <dbl>, x18 <dbl>, x19 <chr>, x20 <dbl>, x21 <dbl>,
## #   x22 <dbl>, x23 <dbl>, x24 <chr>, x25 <dbl>, x26 <dbl>, x27 <dbl>,
## #   x28 <dbl>, x29 <dbl>, x30 <dbl>, x31 <chr>, x32 <dbl>, x33 <chr>,
## #   x34 <dbl>, x35 <dbl>, x36 <dbl>, x37 <dbl>, x38 <dbl>, x39 <chr>,
## #   x40 <dbl>, x41 <dbl>, x42 <dbl>, x43 <dbl>, x44 <dbl>, x45 <dbl>,
## #   x46 <dbl>, x47 <dbl>, x48 <dbl>, x49 <dbl>, x50 <dbl>, x51 <dbl>, ...
```

1-2. Sanity check and basic data cleaning for train data

- For simplicity, data cleaning outcomes are suppressed.
- Here, detailed steps include:
  - Check variable types, and fix any incorrect types if necessary
  - Remove features having >80% of missing values
  - Remove features with only one value
  - Sanity check for factor levels, and fix any incorrect levels if necessary
  - Check the number of NAs in columns and rows
  - Check the number of unique elements of features

```r
# Check variable types
tmp <- raw_train %>% select(where(is.double))
str(tmp)
tmp2 <- raw_train %>% select(where(is.character))
str(tmp2)            # Need to correct x7, x19

# Check if types of all the features are either character or double
(ncol(tmp) + ncol(tmp2) == ncol(raw_train))       # TRUE

# Correct x7 (%-scale to numeric) and x19 (remove '$' at the front and convert
# to numeric)
raw_train_clean <- raw_train
new_x7 <- as.numeric(sub("%", "", raw_train$x7)) / 100
head(new_x7)
raw_train_clean$x7 <- new_x7
new_x19 <- as.numeric(sub("\\$", "", raw_train$x19))
head(new_x19)
raw_train_clean$x19 <- new_x19

# Remove features having >80% of missing values
(numNAs <- apply(is.na(raw_train_clean), 2, sum) / nrow(raw_train_clean))
sort(numNAs, decreasing=T)[1:20]
# 3 features with >80% missing: x44, x57, x30
indic <- which(numNAs > 0.8)
colnames_tooManyNA <- colnames(raw_train_clean)[indic]
colnames_tooManyNA
raw_train_clean <- raw_train_clean %>% select(-any_of(colnames_tooManyNA))
```

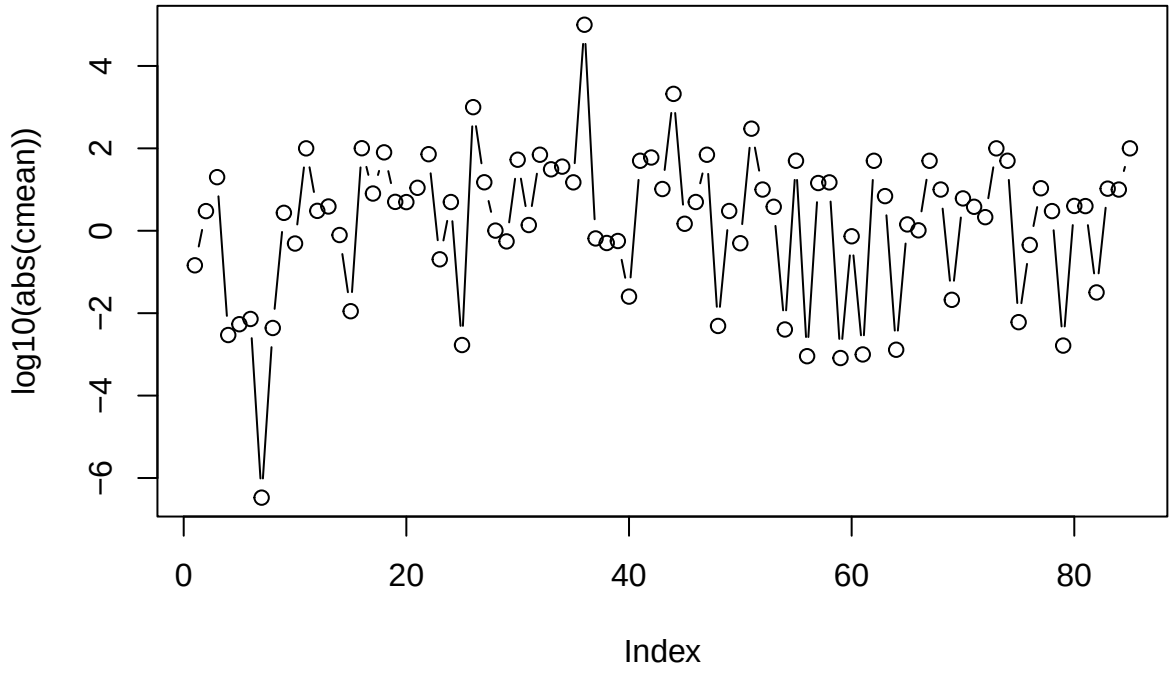
```r
# Remove features with only one value
(numUniqueEle <- apply(raw_train_clean, 2, function(x) length(unique(na.omit(x)))) )
colnames_onlyOneValue <- colnames(raw_train_clean)[numUniqueEle == 1]
colnames_onlyOneValue
raw_train_clean %>% select(all_of(colnames_onlyOneValue))
# x39 = '5-10 miles', x99 = yes
raw_train_clean <- raw_train_clean %>% select(-any_of(colnames_onlyOneValue))

# A list of removed features from the beginning that will be applied to the test data
(colnames_toBeRemoved <- c(colnames_tooManyNA, colnames_onlyOneValue))

# Sanity check for factor levels
tmp3 <- raw_train_clean %>% select(where(is.character))
sapply(tmp3, table, useNA = 'ifany')

# Two observations here.
# (1) x3 (day) levels are mixed with abbreviations.
# (2) 3 features (gender, state, manufacturer) have
# significant portions of NAs. For those, I assign an 'Missing' level.
# The others are fine.

# (1) Correct x3 (day) factor levels
raw_train_clean <- raw_train_clean %>%
  mutate(x3 = fct_recode(x3, Monday = 'Mon', Tuesday = 'Tue', Wednesday = 'Wed',
                         Thursday = 'Thur', Friday = 'Fri', Saturday = 'Sat',
                         Sunday = 'Sun'))
table(raw_train_clean$x3)

# (2) Assign a 'Missing' label to NAs in the 3 features (gender, state, manufacturer)
(colnames_FactorWithNA <- colnames(tmp3)[apply(is.na(tmp3), 2, any)])
raw_train_clean2 <- raw_train_clean %>%
  mutate(across(colnames_FactorWithNA, function(x) fct_explicit_na(x, na_level='Missing')))

# Check cleaned factors
tmp4 <- raw_train_clean2 %>% select(-where(is.numeric))
sapply(tmp4, table, useNA = 'ifany')

# Check out after cleaning
# Number of NAs in columns
print(raw_train_clean2, width=Inf)
(numNAs <- apply(is.na(raw_train_clean2), 2, sum) / nrow(raw_train_clean2))
sort(numNAs, decreasing=T)[1:20]
# Maximum percentage of NAs is ~44% which seems fine.

# Number of NAs in rows to check if there are data points with too many missing values
numNAs_row <- apply(is.na(raw_train_clean2), 1, sum) / ncol(raw_train_clean2)
sort(numNAs_row, decreasing=T)[1:20]
# Maximum percentage of NAs is ~17% which seems fine.

# Number of unique elements of features
(numUniqueEle <- apply(raw_train_clean2, 2, function(x) length(unique(na.omit(x)))) )
sort(numUniqueEle)
```

```r
# Here x59, x79, x98 are found to be binary, but annotated as double.

# Make them categorical
raw_train_clean2 <- raw_train_clean2 %>% mutate(across(c('x59', 'x79', 'x98'), as.factor))

# Sanity check for factor levels for c('x59', 'x79', 'x98')
tmp5 <- raw_train_clean2 %>% select(all_of(c('x59', 'x79', 'x98')))
sapply(tmp5, table, useNA = 'ifany')

# Assign a 'Missing' label to NAs in x79
raw_train_clean2 <- raw_train_clean2 %>%
  mutate(x79 = fct_explicit_na(x79, na_level='Missing'))

# Check again cleaned factors
tmp6 <- raw_train_clean2 %>% select(-where(is.numeric))
sapply(tmp6, table, useNA = 'ifany')
```

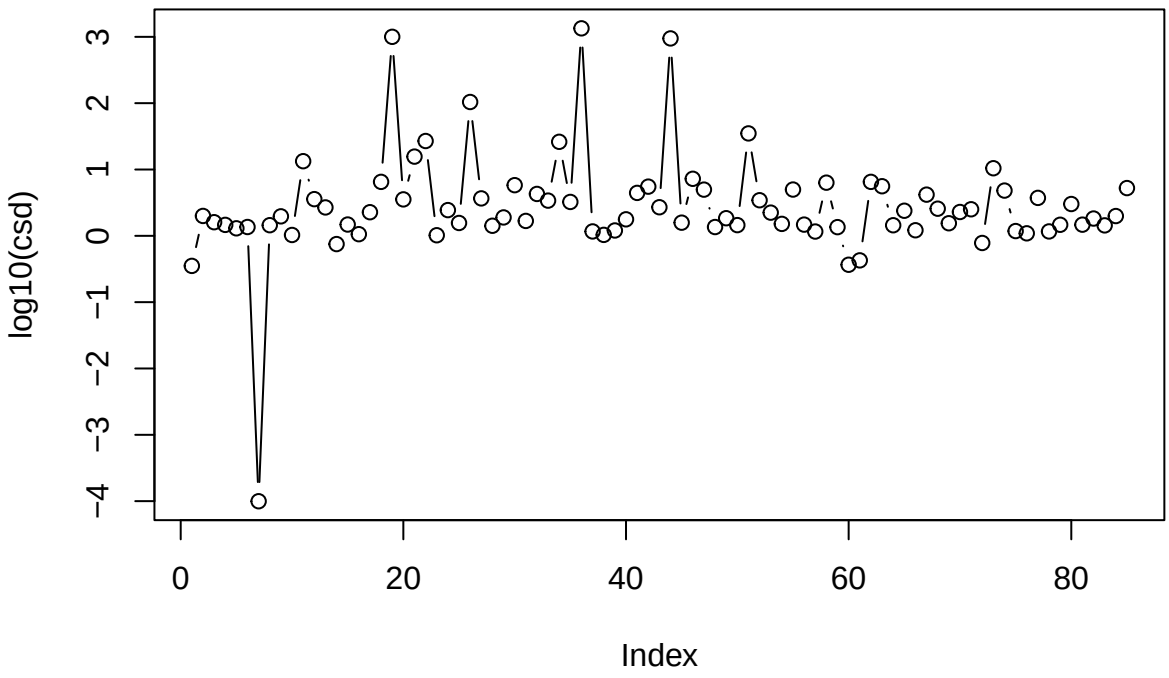
- After cleaning, it has 95 features + 1 binary response with n=40K.

```r
# Check scales of continuous features
df_numeric <- raw_train_clean2 %>% select(where(is.numeric))
summary(df_numeric)

# Moments of numeric features
cmean <- sapply(df_numeric, function(x) mean(x, na.rm=T))
csd <- sapply(df_numeric, function(x) sd(x, na.rm=T))
cmax <- sapply(df_numeric, function(x) max(x, na.rm=T))
cmin <- sapply(df_numeric, function(x) min(x, na.rm=T))

plot(log10(abs(cmean)), type='b')
```

Check distributions of numeric features and consider the necessity of feature transformation




```
plot(log10(csd), type='b')
```

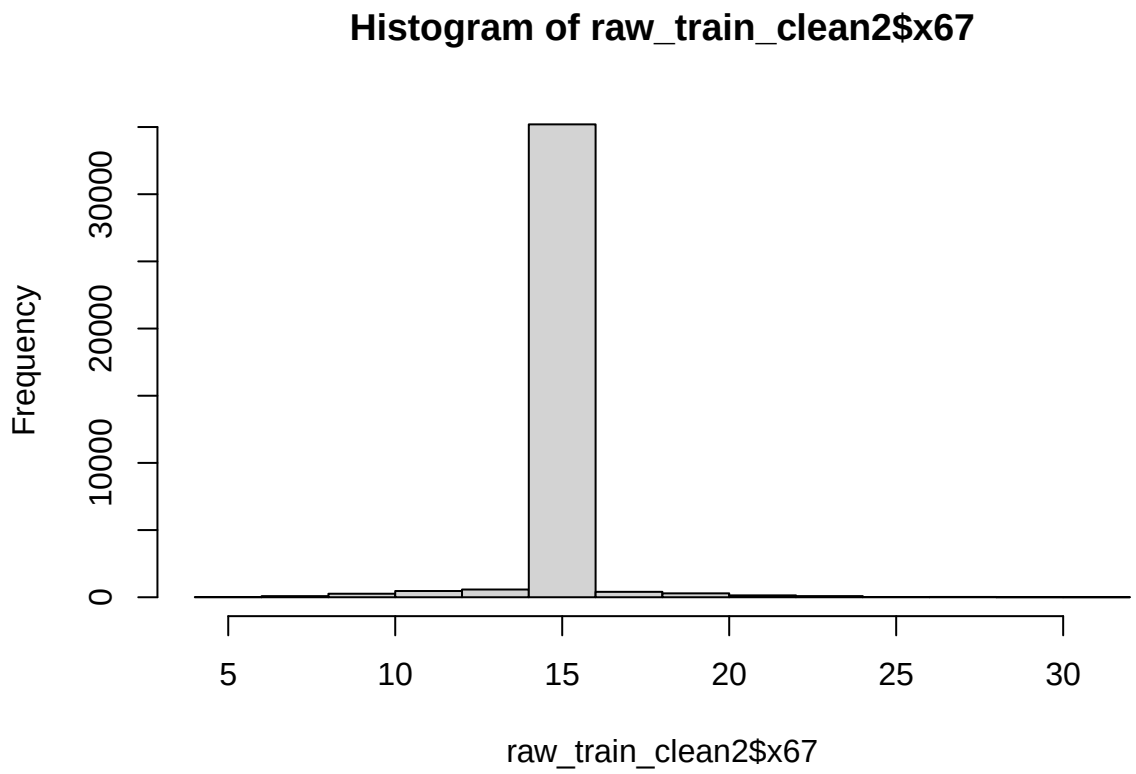


- Plots of column means and standard deviations show that the feature scales are highly heterogeneous.
- Due to this, a scaled dataset will be fed into the next imputation, regression, and other prediction steps for computational stability.

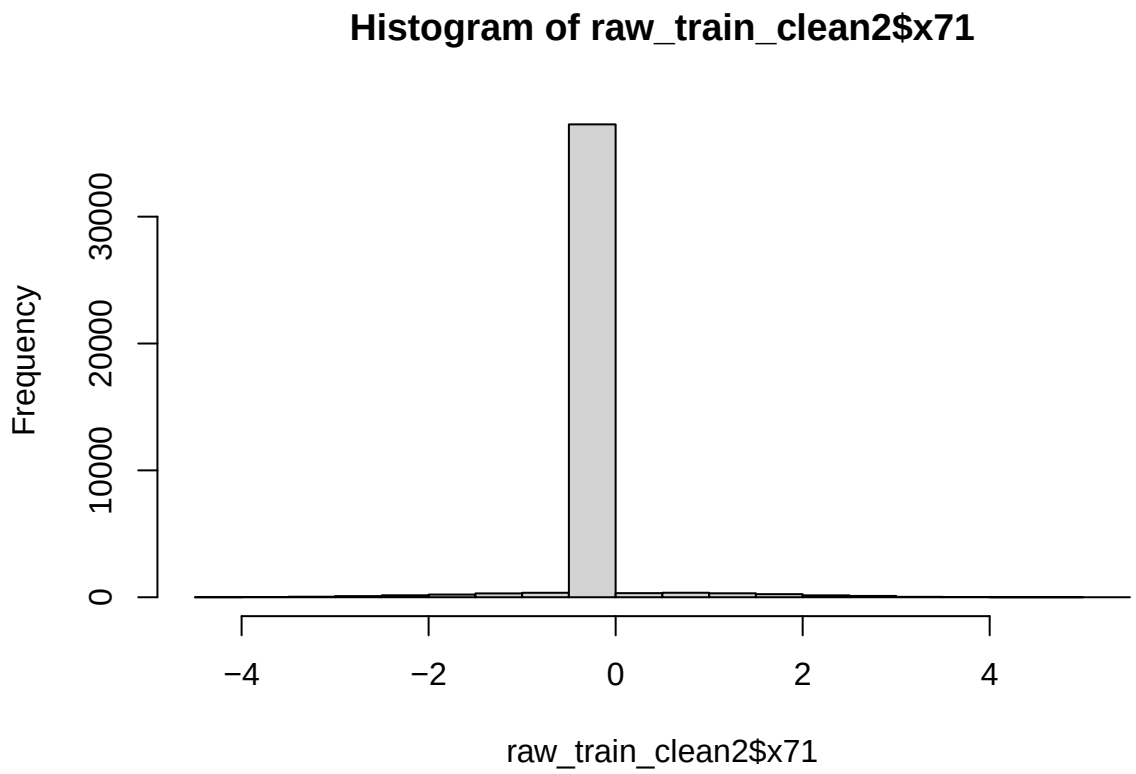
- Now I check if there are features having extremely abnormal distributions, because they might cause computational instability or disrupt robust model fitting.

```
(cols_tooExtremeMax <- colnames(df_numeric)[cmax > cmean + 10 * csd])
# [1] "x21" "x32" "x35" "x67" "x71" "x73" "x75" "x84"
(cols_tooExtremeMin <- colnames(df_numeric)[cmin < cmean - 10 * csd])
# [1] "x71"

# Because there are some features with extreme maxima or minima, I examine kurtosis.
# In fact, those features turned out to have very high kurtosis
# (the highest three kurtosis are 98, 67, 48).

ckurtosis <- sapply(df_numeric, function(x) kurtosis(x, na.rm=T))
sort(ckurtosis, decreasing = T)
(cols_highKurtosis <- colnames(df_numeric)[ckurtosis > 10])
# [1] "x21" "x32" "x35" "x58" "x67" "x71" "x75" "x84"

# Examine those distributions with too high kurtosis.
hist(raw_train_clean2$x58)
```


```
hist(raw_train_clean2$x67)
```




```
hist(raw_train_clean2$x71)
```




```
hist(raw_train_clean2$x84)
```

- Above 4 features seem to be already imputed by mean or median.
- Because most of high kurtosis features have both positive and negative values, I don't consider log-transformation for them.
- Later in logistic regression, I will check if those features are informative or not.
- For now, no feature is transformed.

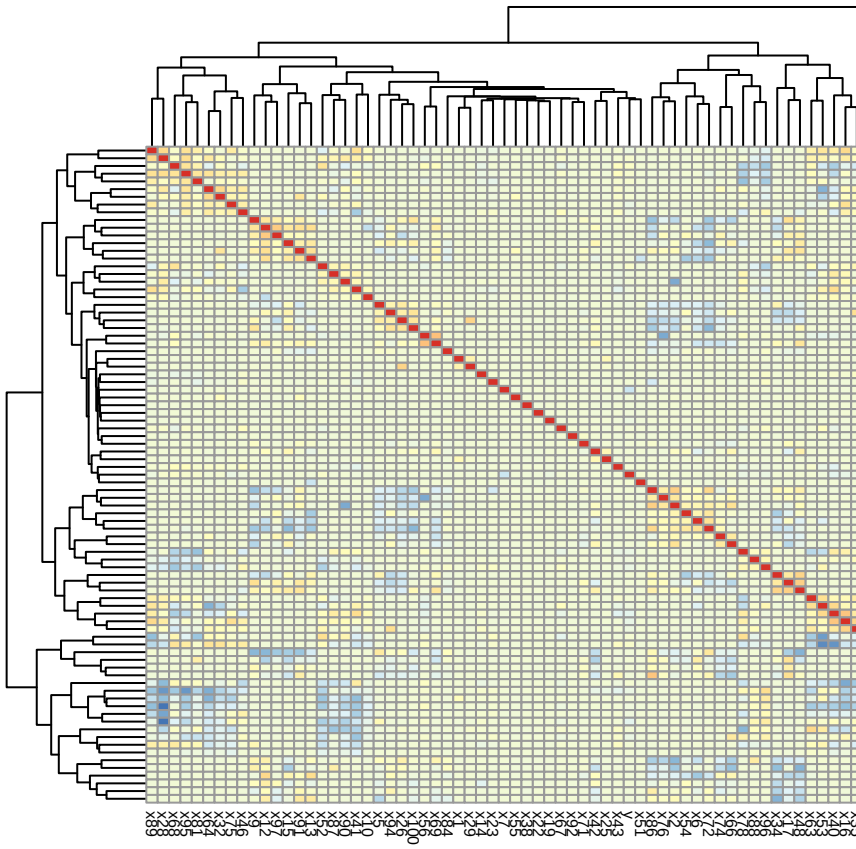
1-3. Impute missing values of numerical features using k-nearest neighbor (knn) method

- `impute::impute.knn()` provides fast imputation originally designed for microarray data.

```
indNAs <- (apply(is.na(raw_train_clean2), 2, sum) > 0)
sum(indNAs)                              # 34 numerical features have NAs

# Reorganize numeric features into two sets that have NAs and don't have NAs
df_withNA <- raw_train_clean2 %>% select(where(function(x) any(is.na(x))))
df_numFeaturesWithoutNA <- raw_train_clean2 %>%
  select(-where(function(x) any(is.na(x)))) %>%
  select(where(is.numeric)) %>%
  select(-'y')
df_numFeatures <- bind_cols(df_withNA, df_numFeaturesWithoutNA)

# I choose to feed scaled data for knn imputation because impute.knn() assumes
# homogeneous scales across columns.

df_numFeatures_sc <- scale(df_numFeatures)
```

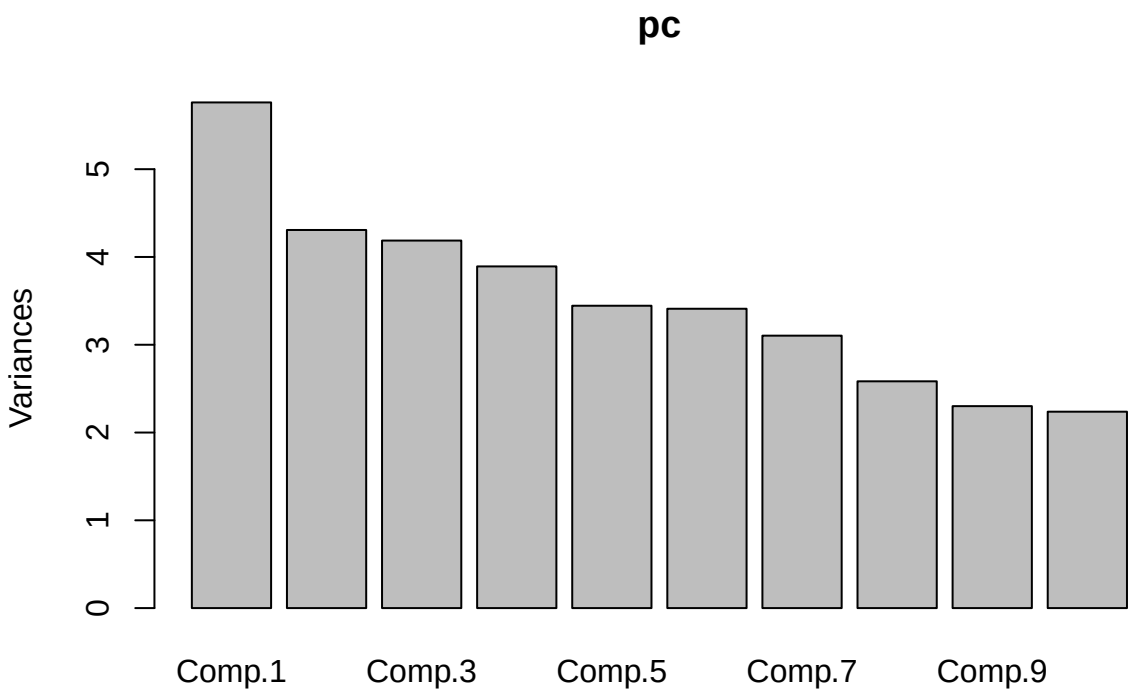
```
# Impute NAs in the scaled numeric dataset using k-nearest neighbor averaging (knn)
tic()
inputmat <- as.matrix(df_numFeatures_sc)
knnout <- impute.knn(inputmat, k=5)
toc(log=T)                                  # 5.8 sec for 40K rows
df_numFeatures_sc_imputed  <- as_tibble(knnout$data)

# Impute NAs also for unscaled data just in case when it is needed.
tic()
inputmat <- as.matrix(df_numFeatures)
knnout <- impute.knn(inputmat, k=5)
toc(log=T)                                  # 5.9 sec for 40K rows
df_numFeatures_imputed  <- as_tibble(knnout$data)

# Reorganize feature orders for easier inspection
df_categorical <- raw_train_clean2 %>% select(-where(is.numeric))
df_y <- raw_train_clean2 %>% select('y')
dat_train_sc <- bind_cols(df_y, df_numFeatures_sc_imputed, df_categorical)
print(dat_train_sc, width=Inf)
dat_train <- bind_cols(df_y, df_numFeatures_imputed, df_categorical)
print(dat_train, width=Inf)
```

```
# A heatmap of correlation matrix shows no strong pattern.
pheatmap(cor(cbind(df_y, df_numFeatures_sc_imputed)),
         cluster_rows = T, cluster_cols = T, fontsize = 6)
```

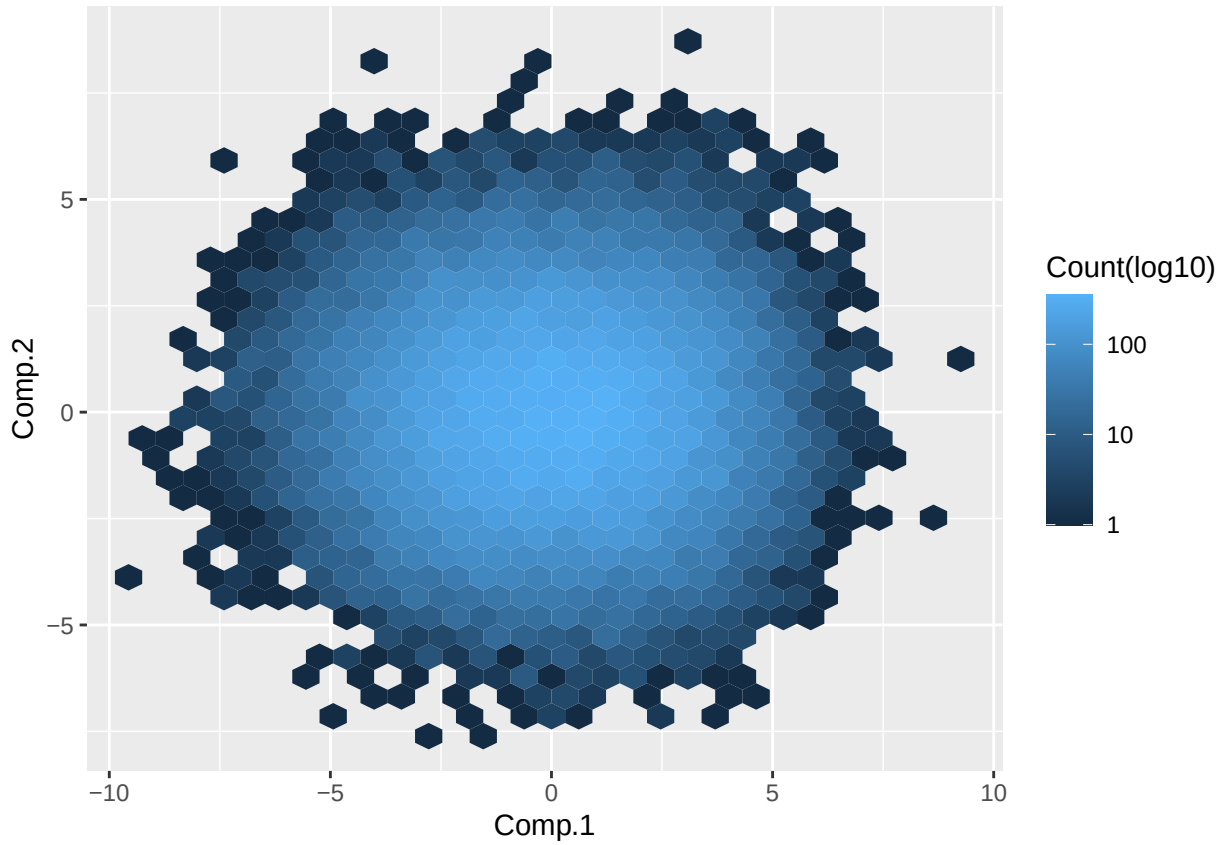


Simple visualization of continuous features

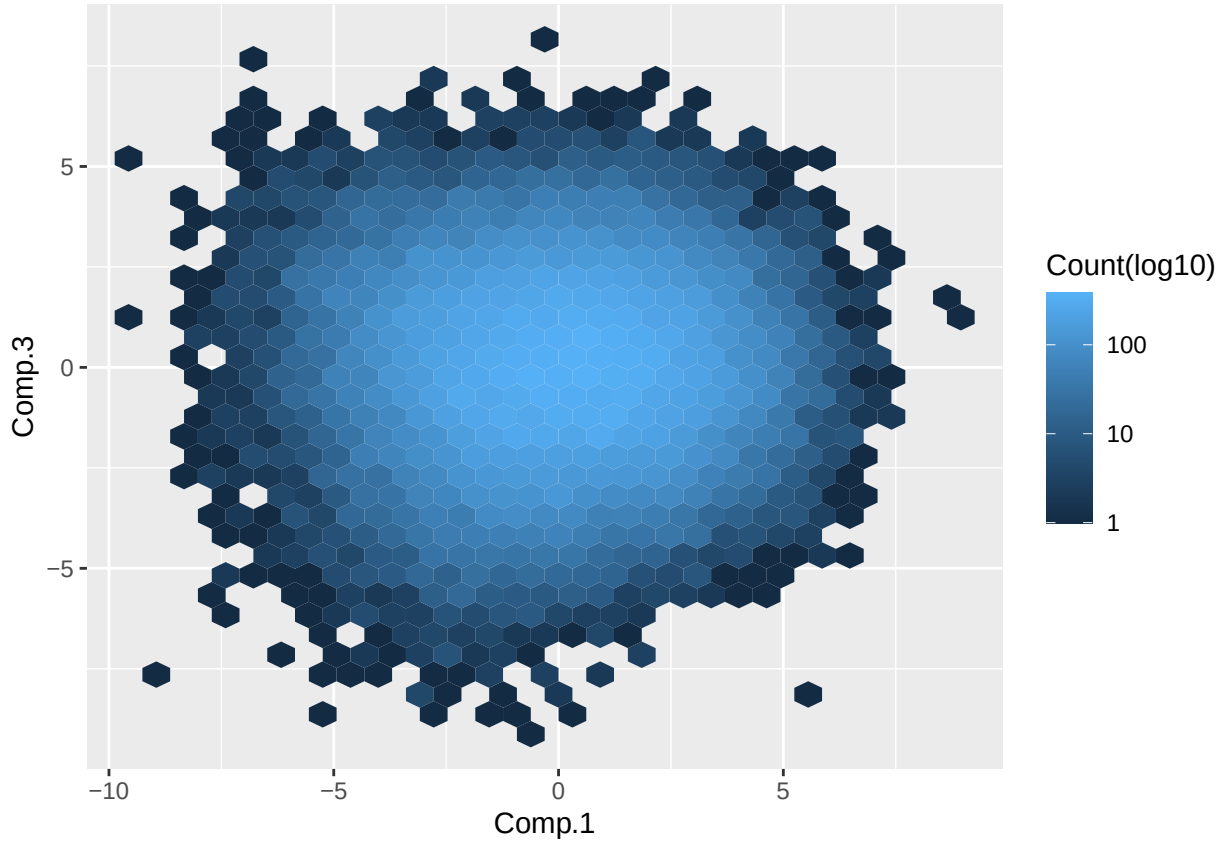
```
# Low-dimensional representation of numeric features using PCA
pc <- princomp(df_numFeatures_sc_imputed)
plot(pc)
```




```
pc3 <- data.frame(pc$scores[, 1:3])
ggplot(data = pc3, aes(x=Comp.1, y=Comp.2)) +
  geom_hex() +
  scale_fill_gradient(name = 'Count(log10)', trans = "log10")
```




```
ggplot(data = pc3, aes(x=Comp.1, y=Comp.3)) +
  geom_hex() +
  scale_fill_gradient(name = 'Count(log10)', trans = "log10")
```



- PC plots show no interesting pattern.

```
mean(df_y$y)
```

Examine churn rates per level of categorical features

```
## [1] 0.145075
```

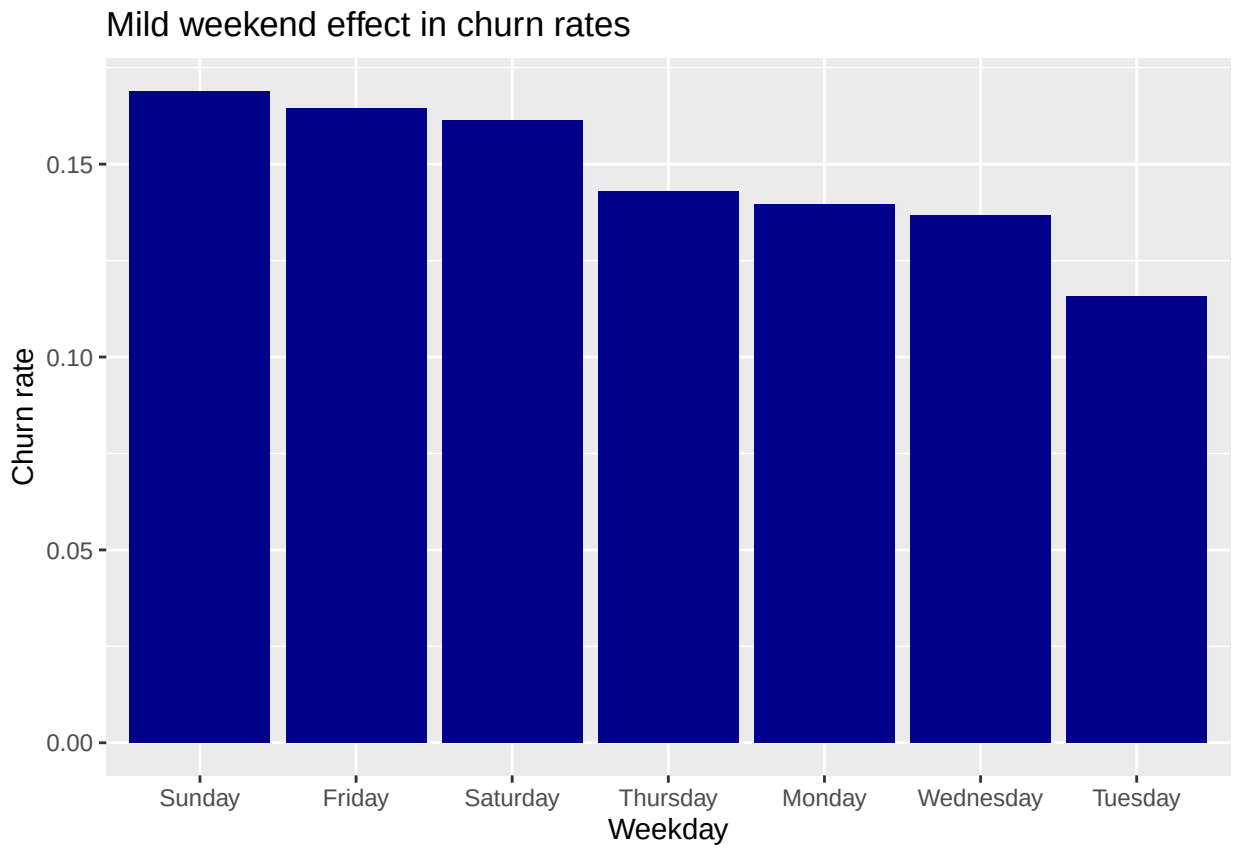
```
df_y_categorical <- bind_cols(df_y, df_categorical)
```

- Churn rates per weekday
  - Fri~Sun churn rates are higher (>0.16), suggesting a mild 'weekend' effect on the insurance customer churn rates.
  - Tuesday (17%) churn rate is the lowest (0.116).

```
dftmp <- df_y_categorical %>% group_by(x3) %>%
  summarize(meanY = mean(y)) %>%
  arrange(desc(meanY))
print(dftmp)
```

```
## # A tibble: 7 x 2
##   x3        meanY
##   <fct>     <dbl>
## 1 Sunday    0.169
## 2 Friday    0.165
## 3 Saturday  0.161
## 4 Thursday  0.143
## 5 Monday    0.140
## 6 Wednesday 0.137
## 7 Tuesday   0.116
```

```
fig_churn_day <- dftmp %>%
  ggplot(aes(x = factor(x3, levels=c(x3)), y = meanY)) +
  geom_col(fill='darkblue') +
  labs(title = 'Mild weekend effect in churn rates',
       x = 'Weekday',
       y = 'Churn rate')
print(fig_churn_day)
```



- Churn rates per gender are similar.

```
df_y_categorical %>% group_by(x24) %>%
  summarize(meanY = mean(y))
```

```
## # A tibble: 3 x 2
```

```
##   x24     meanY
##   <fct>   <dbl>
## 1 female  0.146
## 2 male    0.144
## 3 Missing 0.145
```

- State-wide churn rates are quite varying.

```
dftmp <- df_y_categorical %>% group_by(x33) %>%
  summarize(meanY = mean(y)) %>%
  arrange(desc(meanY))
print(dftmp, n=Inf)
```

```
## # A tibble: 52 x 2
##    x33             meanY
##    <fct>           <dbl>
##  1 Alaska          0.312
##  2 Idaho           0.303
##  3 Oregon          0.277
##  4 Washington      0.26
##  5 Montana         0.231
##  6 Kansas          0.220
##  7 North Carolina  0.211
##  8 Illinois        0.190
##  9 Georgia         0.188
## 10 Iowa            0.187
## 11 Vermont         0.185
## 12 Florida         0.183
## 13 Missouri        0.177
## 14 Ohio            0.176
## 15 South Carolina  0.173
## 16 Indiana         0.166
## 17 Kentucky        0.165
## 18 Nebraska        0.155
## 19 South Dakota    0.153
## 20 Maine           0.150
## 21 New Hampshire   0.147
## 22 Hawaii          0.142
## 23 Missing         0.142
## 24 Rhode Island    0.138
## 25 Connecticut     0.135
## 26 DC              0.134
## 27 Nevada          0.134
## 28 New York        0.134
## 29 Mississippi     0.133
## 30 Massachusetts   0.130
## 31 Texas           0.127
## 32 Wisconsin       0.126
## 33 West Virginia   0.125
## 34 New Mexico      0.120
## 35 Minnesota       0.119
## 36 Arizona         0.117
## 37 Oklahoma        0.116
```

```
## 38 Arkansas       0.116
## 39 Pennsylvania   0.114
## 40 Alabama        0.114
## 41 California     0.112
## 42 Maryland       0.110
## 43 Louisiana      0.110
## 44 New Jersey     0.109
## 45 Michigan       0.109
## 46 Tennessee      0.109
## 47 Colorado       0.104
## 48 North Dakota   0.0994
## 49 Utah           0.0973
## 50 Virginia       0.0936
## 51 Wyoming        0.0794
## 52 Delaware       0.0508
```

- Churn rates per competitor are similar, except that 'farmers' is lower and 'progressive' is higher than the average.

```
df_y_categorical %>% group_by(x65) %>%
  summarize(meanY = mean(y)) %>%
  arrange(desc(meanY))
```

```
## # A tibble: 5 x 2
##   x65         meanY
##   <chr>       <dbl>
## 1 progressive 0.151
## 2 esurance    0.146
## 3 geico       0.144
## 4 allstate    0.144
## 5 farmers     0.135
```

- Churn rates per manufacturer are similar

```
df_y_categorical %>% group_by(x77) %>%
  summarize(meanY = mean(y)) %>%
  arrange(desc(meanY))
```

```
## # A tibble: 8 x 2
##   x77       meanY
##   <fct>     <dbl>
## 1 nissan    0.151
## 2 toyota    0.150
## 3 buick     0.149
## 4 ford      0.146
## 5 subaru    0.146
## 6 chevrolet 0.143
## 7 Missing   0.143
## 8 mercedes  0.139
```

- x31_yes group (~15% of total) has a very low churn rate (0.08).

```
df_y_categorical %>% group_by(x31) %>%
  summarize(meanY = mean(y)) %>%
  arrange(desc(meanY))
```

```
## # A tibble: 2 x 2
##   x31    meanY
##   <chr>  <dbl>
## 1 no    0.157
## 2 yes   0.0771
```

- Mar & Sep show slightly higher churn rates.

```
df_y_categorical %>% group_by(x60) %>%
  summarize(meanY = mean(y)) %>%
  arrange(desc(meanY))
```

```
## # A tibble: 12 x 2
##    x60       meanY
##    <chr>     <dbl>
##  1 March     0.159
##  2 September 0.156
##  3 November  0.153
##  4 October   0.153
##  5 January   0.150
##  6 February  0.147
##  7 August    0.145
##  8 April     0.144
##  9 July      0.143
## 10 December  0.141
## 11 June      0.138
## 12 May       0.123
```

- x93_yes group (~11% of total) has a very low churn rate (0.07).

```
df_y_categorical %>% group_by(x93) %>%
  summarize(meanY = mean(y)) %>%
  arrange(desc(meanY))
```

```
## # A tibble: 2 x 2
##   x93    meanY
##   <chr>  <dbl>
## 1 no    0.155
## 2 yes   0.0650
```

- The rest categorical features are little informative for the response.

```
df_y_categorical %>% group_by(x59) %>%
  summarize(meanY = mean(y))
```

```
## # A tibble: 2 x 2
##   x59   meanY
##   <fct> <dbl>
## 1 0     0.145
## 2 1     0.144
```

```
# Mean resonse per x79
df_y_categorical %>% group_by(x79) %>%
  summarize(meanY = mean(y))
```

```
## # A tibble: 3 x 2
##   x79     meanY
##   <fct>   <dbl>
## 1 0       0.147
## 2 1       0.145
## 3 Missing 0.148
```

```
# Mean resonse per x98
df_y_categorical %>% group_by(x98) %>%
  summarize(meanY = mean(y))
```

```
## # A tibble: 2 x 2
##   x98   meanY
##   <fct> <dbl>
## 1 0     0.149
## 2 1     0.141
```

```
# Save
save(dat_train_sc, dat_train, df_y, df_numFeatures_sc_imputed,
     df_numFeatures_imputed, df_categorical, colnames_toBeRemoved,
     file = 'dat_train_cleaned_imputed.rdata')
```