Insurance Customer Churn Prediction Model (Part 3)

Load packages and previous output

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

# Load step 1 & 2 output
load('dat_train_cleaned_imputed.rdata')
load('glmLASSO_selectedFeatSets_includeSquared.rdata')
load('lr_models_AUC_Acc.rdata')
load('MLP_models_AUC_Acc.rdata')
mlp1_model_k2 <- load_model_hdf5('mlp1_model_k2.h5')
```

Step 3 - Generate churn predictions for test data

3-1. Apply the same data cleaning done to train data for test data

- Raw test data with 100 features for 10K customers

```
raw_test <- read_csv(file.path('churnPrediction_test.csv'))
head(raw_test)
```

```
## # A tibble: 6 x 100
##      x1    x2 x3        x4     x5     x6 x7         x8    x9    x10   x11    x12
##   <dbl> <dbl> <chr>  <dbl>  <dbl>  <dbl> <chr>   <dbl> <dbl>  <dbl> <dbl>  <dbl>
## 1  4.75  20.5 Wedn~  2.30  -1.82  -0.752 0.00~ -3.24   0.588 -0.261 101.   -0.812
## 2  1.15  19.3 Fri    1.86  -0.774 -1.46  0.00~  0.443  0.522 -1.09  105.    8.81
## 3  4.99  18.8 Satu~  1.04  -1.55   2.63  -5e-~ -1.17   5.74   0.223 102.    7.83
## 4  3.71  18.4 Tues~ -0.170 -2.40  -0.785 -0.0~ -2.66   1.55   0.210  82.7   0.437
## 5  3.80  20.2 Mond~  2.09  -0.733 -0.703 0.01~  0.0564 2.88  -0.458  75.0   8.03
## 6  1.45  17.2 Sun    0.999  0.882 -1.70  0.00~ -0.372  1.27  -1.43   78.4   3.12
## # ... with 88 more variables: 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>, x52 <dbl>, ...
```

```r
tmp <- raw_test %>% select(where(is.double))
str(tmp)
tmp2 <- raw_test %>% 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_test))       # TRUE

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

# Remove the features removed from the train data at the beginning
# due to having too many missing values or only one value
(numNAs <- apply(is.na(raw_test_clean), 2, sum) / nrow(raw_test_clean))
sort(numNAs, decreasing=T)[1:20]
# Also in test data, 3 features with >80% missing: x44, x57, x30

(numUniqueEle <- apply(raw_test_clean, 2, function(x) length(unique(na.omit(x)))) )
(colnames_onlyOneValue <- colnames(raw_test_clean)[numUniqueEle == 1])
# Also in test data, 'x39' and 'x99' have only one value.

colnames_toBeRemoved
# [1] "x30" "x44" "x57" "x39" "x99"
raw_test_clean <- raw_test_clean %>% select(-any_of(colnames_toBeRemoved))

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

# Also in test data, 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_test_clean <- raw_test_clean %>%
  mutate(x3 = fct_recode(x3, Monday = 'Mon', Tuesday = 'Tue', Wednesday = 'Wed',
                         Thursday = 'Thur', Friday = 'Fri', Saturday = 'Sat',
                         Sunday = 'Sun'))
table(raw_test_clean$x3)

# (2) Assign a 'Missing' label to NAs in the 3 features (gender, state, manufacturer)
```

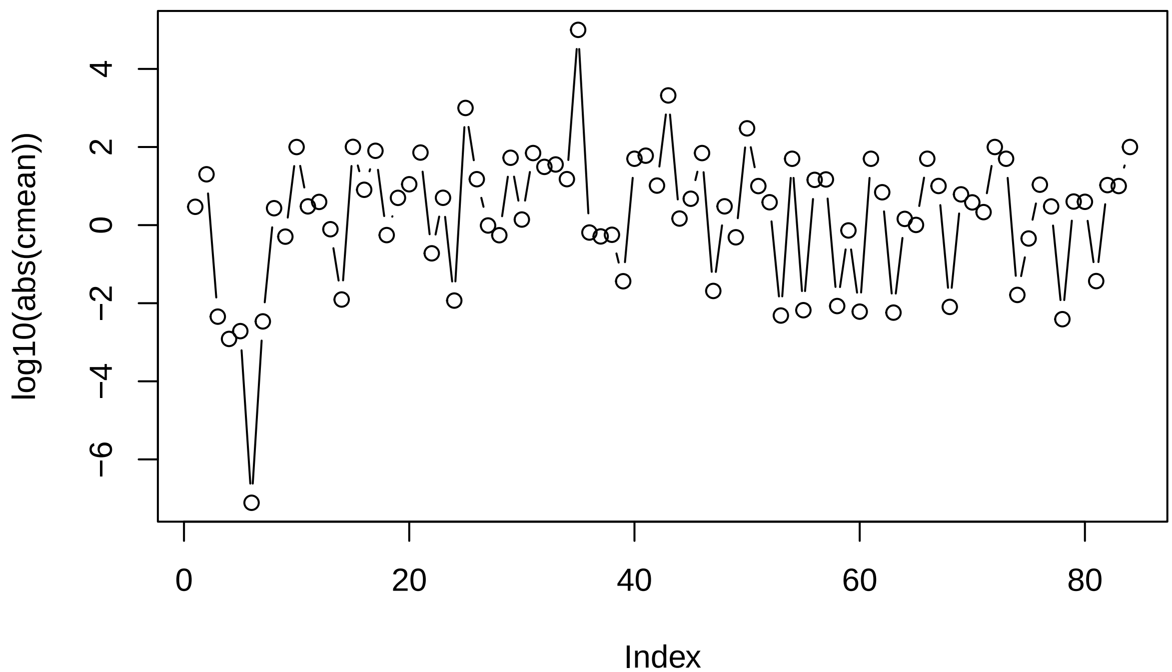
```r
(colnames_FactorWithNA <- colnames(tmp3)[apply(is.na(tmp3), 2, any)])
raw_test_clean2 <- raw_test_clean %>%
  mutate(across(colnames_FactorWithNA, function(x) fct_explicit_na(x, na_level='Missing')))

# Check cleaned factors
tmp4 <- raw_test_clean2 %>% select(-where(is.numeric))
sapply(tmp4, table, useNA = 'ifany')

# Check out after cleaning
# Number of NAs in columns
print(raw_test_clean2, width=Inf)
(numNAs <- apply(is.na(raw_test_clean2), 2, sum) / nrow(raw_test_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_test_clean2), 1, sum) / ncol(raw_test_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_test_clean2, 2, function(x) length(unique(na.omit(x)))) )
sort(numUniqueEle)

# Also in test data, x59, x79, x98 are found to be binary, but annotated as double.

# Make them categorical
raw_test_clean2 <- raw_test_clean2 %>% mutate(across(c('x59', 'x79', 'x98'), as.factor))

# Sanity check for factor levels for c('x59', 'x79', 'x98')
tmp5 <- raw_test_clean2 %>% select(all_of(c('x59', 'x79', 'x98')))
sapply(tmp5, table, useNA = 'ifany')

# Assign a 'Missing' label to NAs in x79
raw_test_clean2 <- raw_test_clean2 %>%
  mutate(x79 = fct_explicit_na(x79, na_level='Missing'))

# Check again cleaned factors
tmp6 <- raw_test_clean2 %>% select(-where(is.numeric))
sapply(tmp6, table, useNA = 'ifany')
```

- After cleaning, cleaned test data has 95 features with n=10K.

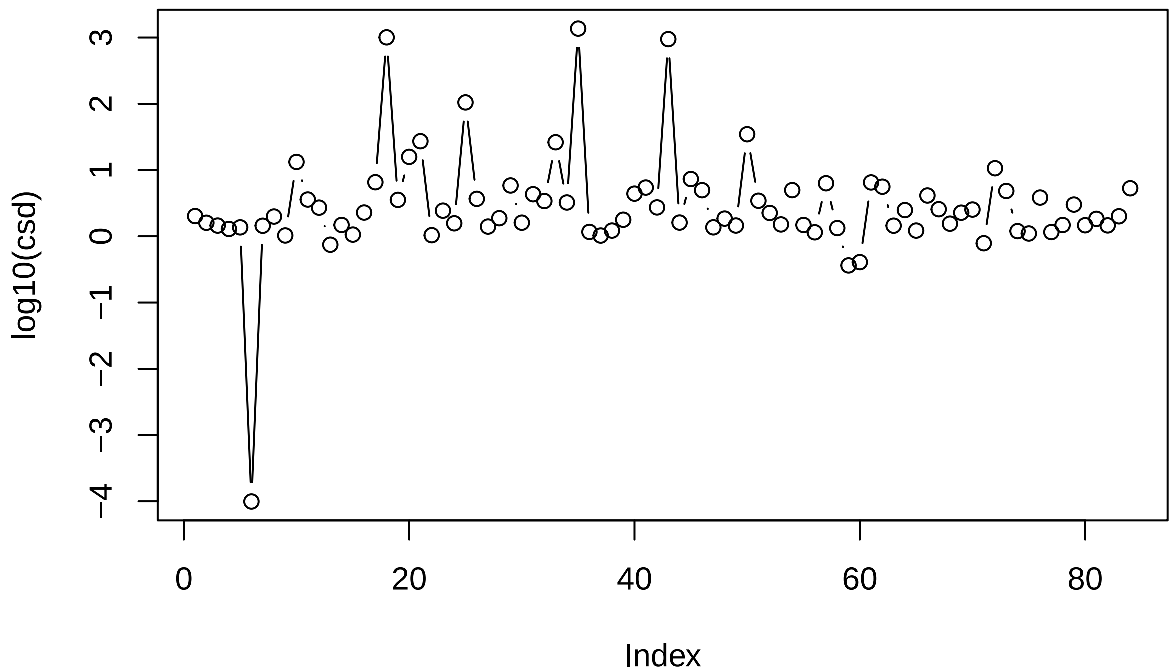
- Check distributions of numeric features just to confirm the test data has the same distribution characteristics

```r
# Check scales of continuous features
df_numeric <- raw_test_clean2 %>% select(where(is.numeric))
summary(df_numeric)
```

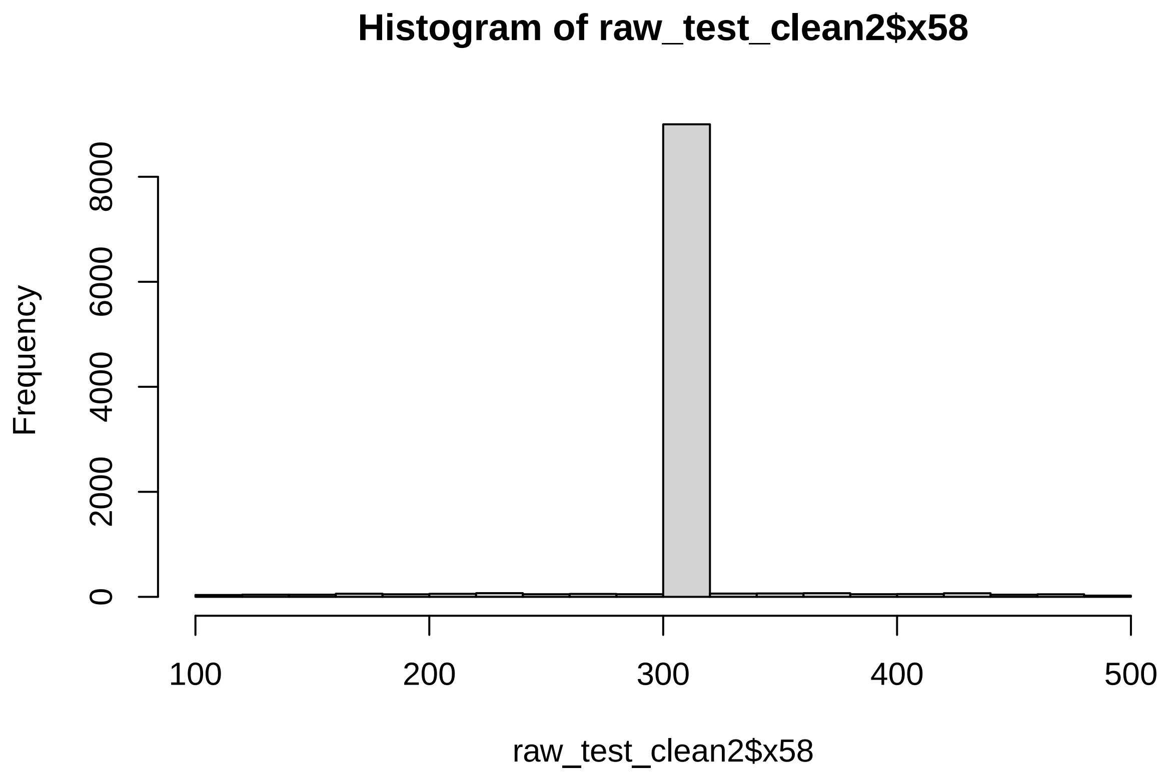
```r
# Moments of numeric features
cmean <- sapply(df_numeric, function(x) mean(x, na.rm=T))
csd <- sapply(df_numeric, function(x) sd(x, na.rm=T))
cmax <- sapply(df_numeric, function(x) max(x, na.rm=T))
cmin <- sapply(df_numeric, function(x) min(x, na.rm=T))

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



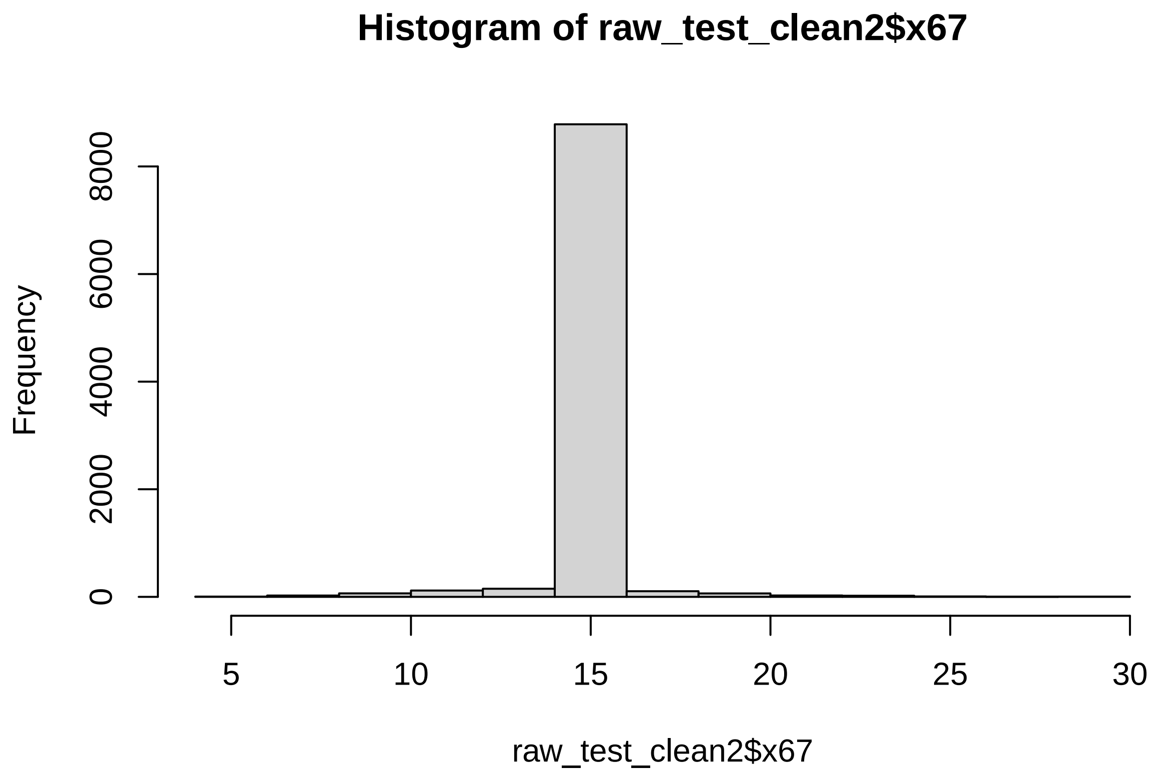

```r
plot(log10(csd), type='b')
```



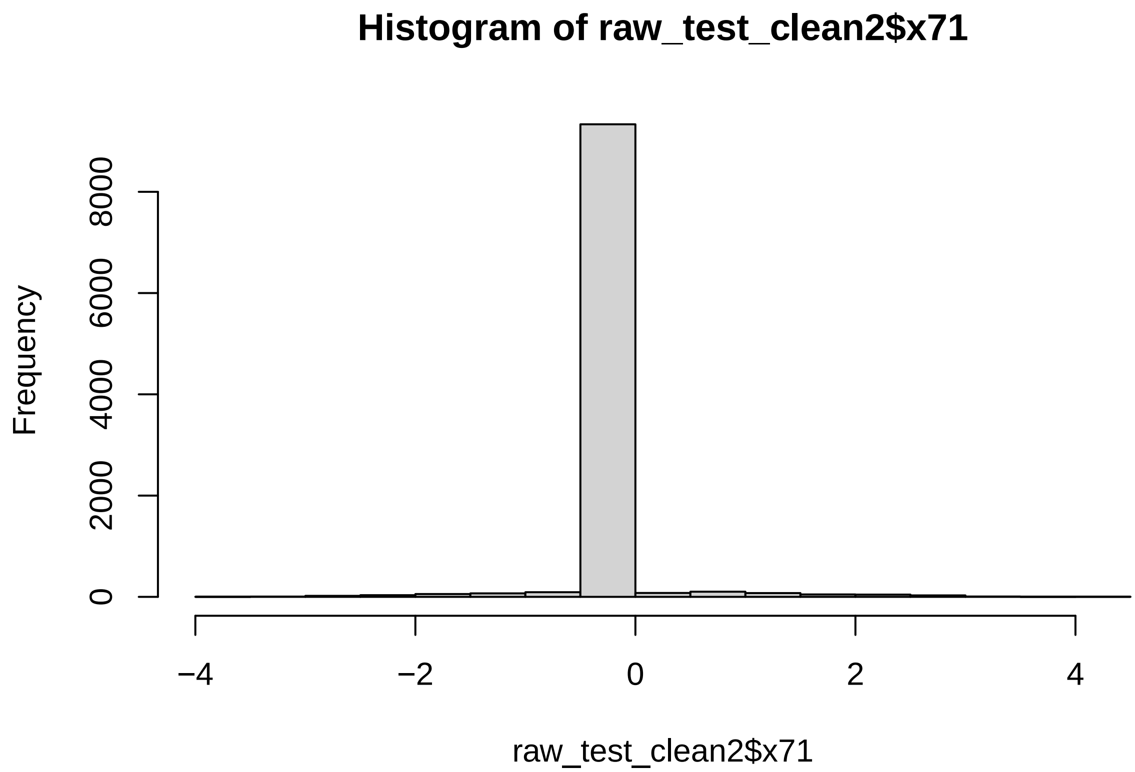

```
# Checking similar features as in train data just to confirm
(cols_tooExtremeMax <- colnames(df_numeric)[cmax > cmean + 10 * csd])
# [1] "x32" "x35" "x67" "x71" "x75" "x84"
(cols_tooExtremeMin <- colnames(df_numeric)[cmin < cmean - 10 * csd])
# character(0)

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" "x58" "x67" "x71" "x75" "x84"

# Examine those distributions with too high kurtosis.
hist(raw_test_clean2$x58)
```




```
hist(raw_test_clean2$x67)
```




```
hist(raw_test_clean2$x71)
```



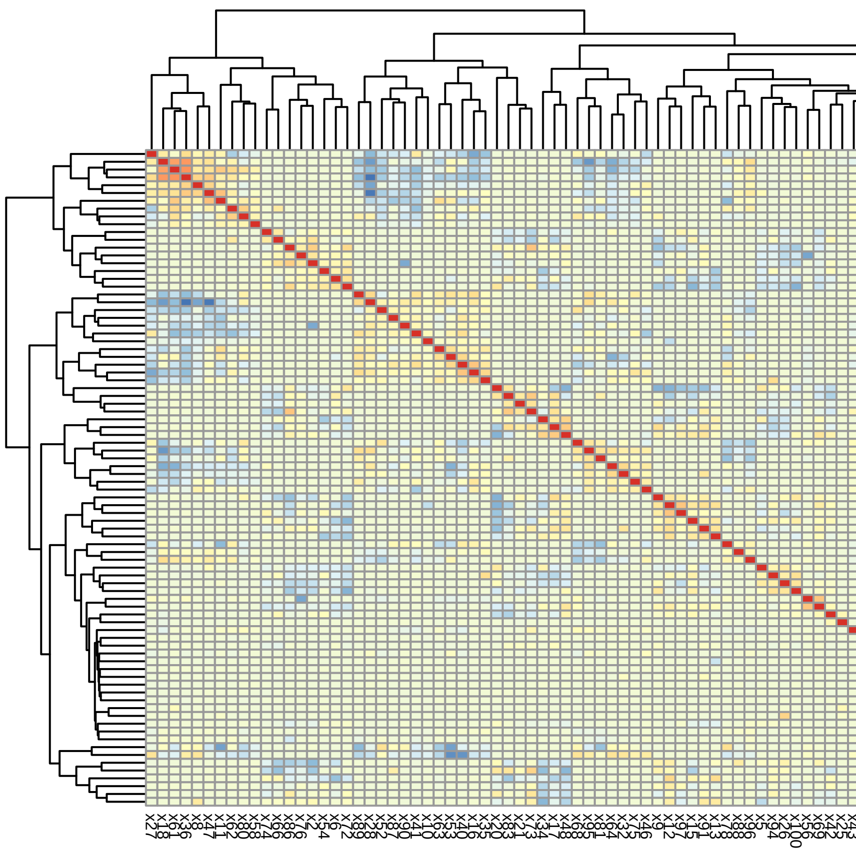

```
hist(raw_test_clean2$x84)
```

3-2. Impute missing values of numerical features using k-nearest neighbor (knn) method as in train data

```
indNAs <- (apply(is.na(raw_test_clean2), 2, sum) > 0)
sum(indNAs)
# Also in test data, 34 numerical features have NAs

# Reorganize numeric features into two sets that have NAs and don't have NAs
df_withNA <- raw_test_clean2 %>% select(where(function(x) any(is.na(x))))
df_numFeaturesWithoutNA <- raw_test_clean2 %>%
  select(-where(function(x) any(is.na(x)))) %>%
  select(where(is.numeric))
df_numFeatures <- bind_cols(df_withNA, df_numFeaturesWithoutNA)

# As in train data, feed scaled data for knn imputation because impute.knn() below
# 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)                                  # 2.4 sec for 10K rows
```

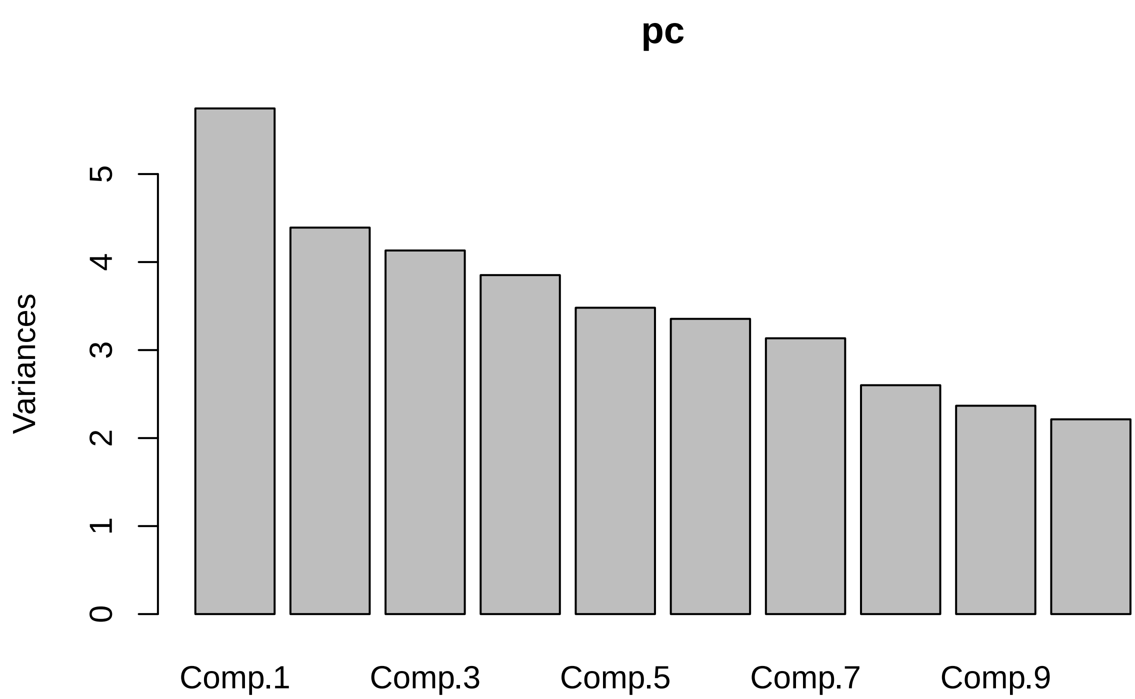
```
df_numFeatures_sc_imputed  <- as_tibble(knnout$data)

# 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)                                    # 1.9 sec for 10K rows
df_numFeatures_imputed  <- as_tibble(knnout$data)

# Reorganize feature orders for easier inspection
df_categorical <- raw_test_clean2 %>% select(-where(is.numeric))
dat_test_sc <- bind_cols(df_numFeatures_sc_imputed, df_categorical)
print(dat_test_sc, width=Inf)
```

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

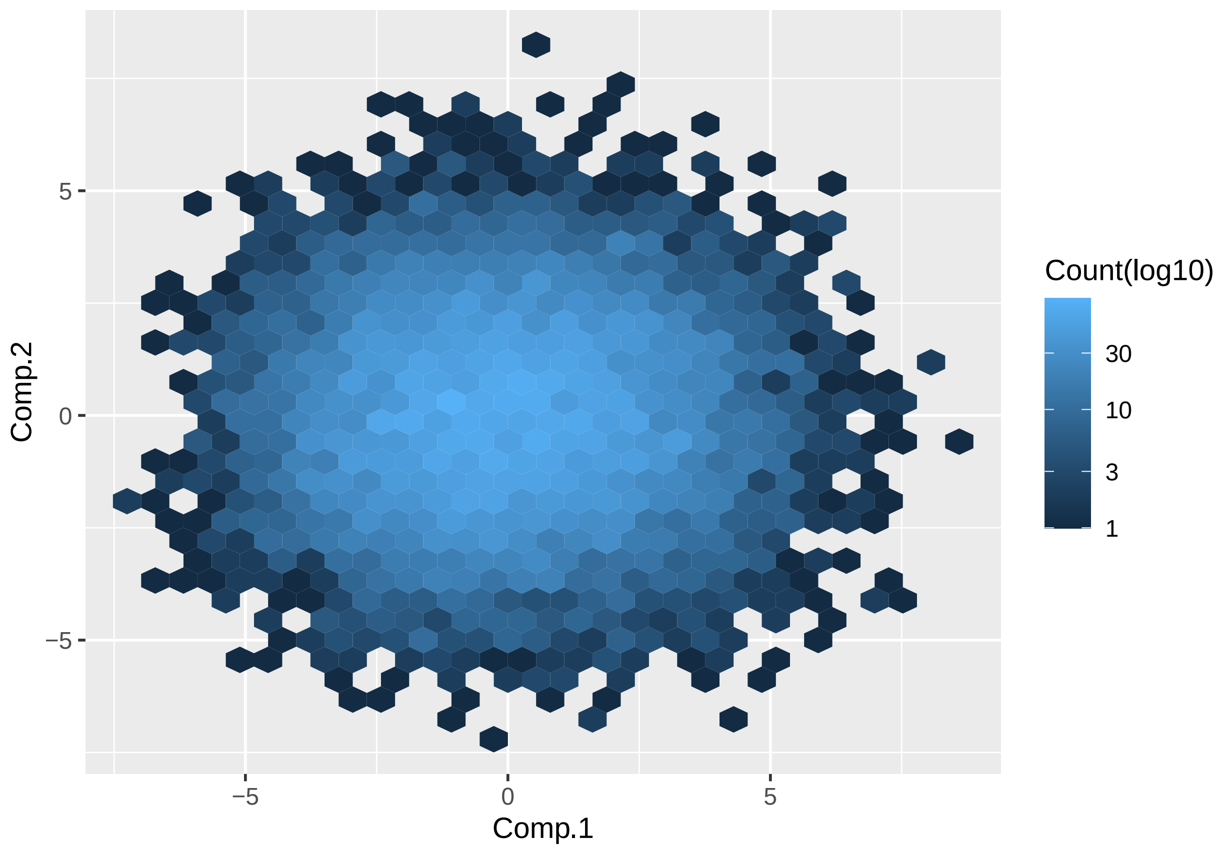


Simple visualization of continuous features

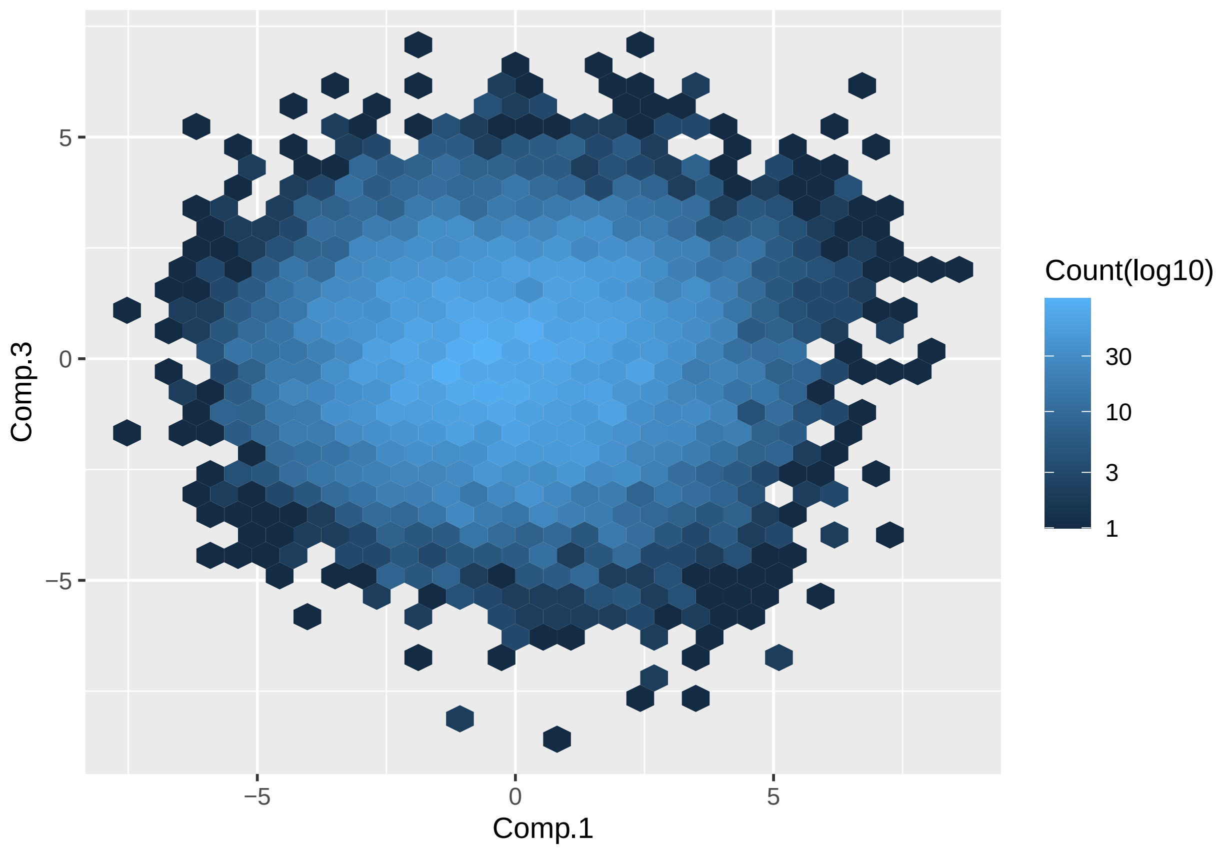
```r
# Low-dimensional representation of numeric features using PCA
pc <- princomp(df_numFeatures_sc_imputed)
plot(pc)
```




```r
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")
```




```r
# PC plots show no interesting pattern.
```

3-3. Include squared continuous features

```r
# Make a new data frame
df_numFeatures_squared <- (as.data.frame(df_numFeatures_sc_imputed))^2
colnames0 <- paste0(colnames(df_numFeatures_squared), "_squared")
colnames(df_numFeatures_squared) <- colnames0
df_numFeatures_squared <- as_tibble(df_numFeatures_squared)
dat_test_sc2 <- bind_cols(df_numFeatures_sc_imputed,
                          df_numFeatures_squared, df_categorical)
```

3-4. Predict test data using the optimal logistic regression (when k = 2) (chosen model: lr_models[[2]])

- Validation AUC was 0.8093, which is the estimate for the test AUC.

```r
lr_AUC[[2]]
```

```
## Area under the curve: 0.8093
```

```r
pred <- predict(lr_models[[2]], newdata = dat_test_sc2, type = 'response')
```

Customer churn prediction for test data using the chosen logistic regression model

3-5. Predict test data using the optimal Multi-layer perceptron (when k = 2) (chosen model: mlp1_models[[2]])

- Validation AUC is 0.7978, which is the estimate for the test AUC.

```r
mlp1_AUC[[2]]
```

```
## Area under the curve: 0.7982
```

```r
pred_formula <- as.formula(paste0('~ ', paste0(sigOrigFeatures[[2]], collapse = ' + ')))
x_test <- model.matrix(pred_formula, data = dat_test_sc2)[, -1]
pred_mlp <- mlp1_model_k2  %>% predict(x_test)
```

Customer churn prediction for test data using the chosen MLP model

Conclusion and Implications

- The project goal is to predict which customers are likely to cancel current insurance based on 100 features in a dataset with 40,000 customer records. I built two predictive models using R. One is a logistic regression model, and the other is a multilayer perceptron (or a neural network) model.
- Through simple exploratory data analysis (EDA), two interesting categorical variables were identified to be useful to tell which customers will be loyal.
  - x31_yes group (~15% of total) has a very low churn rate (0.08).
  - x93_yes group (~11% of total) has a very low churn rate (0.07).
- Through the EDA, a mild weekend effect were found.
  - Fri~Sun churn rates are higher (> 0.160) than overall average 0.145.
- Some of the prediction features were identified to have informative non-linear relations to the target variable.
  - "x4_squared" "x18_squared" "x40_squared" "x87_squared"
  - Especially, 'x4' shows a clear non-linear effect. The higher 'x4' is the lower 'y' is.
- An optimal model was searched for across logistic regression, multilayer perceptron, random forest, and decision tree models together with 5 candidate feature sets that are chosen by a method using logistic regression with LASSO penalty.
- The first selected model is a logistic regression model using 74 extended features, and the second selected model is a multilayer perceptron model using the same feature set.

- Estimates of AUC for the test data are the validation AUC for each model, which is 80.93% for the logistic regression, and 79.78% for the multilayer perceptron.
- The logistic regression model is recommended because it has better interpretability and showed better prediction performances in a validation dataset.