Insurance Customer Churn Predictions - Scott Lee

This project examines a set of customer data for an insurance company, in an attempt to predict customer churn (if a given customer will renew their policy).
This r markdown file starts my development of the assignment in full.
Create logical flow of code from the other r markdown files as appropriate.
Contents:
1. Comprehensive EDA
2. Describe your choice of model based off of EDA
3. Develop 2 types of models (e.g. logistic regression and KNN)
4. Evaluate models using selected performance measures (at least 2)
Pick model based off of abover performance meaures to be main model.
5. Use selected model ,identify and discuss the key factors (variable importance) of the selected model
6. Make suggestions/provide commercial insights to marketing based off of these findings Assume a non data science audience.

Load libraries that will be used throughout project

```
library(car) # for qq plot
```

```
## Warning: package 'car' was built under R version 4.1.1
```

```
## Loading required package: carData
```

```
## Warning: package 'carData' was built under R version 4.1.1
```

```
library(caret) # for some functions e.g. findCorrelation()
```

```
## Warning: package 'caret' was built under R version 4.1.1
```

```
## Loading required package: ggplot2
```

```
## Warning: package 'ggplot2' was built under R version 4.1.1
```

```
## Loading required package: lattice
```

```
## Warning: package 'lattice' was built under R version 4.1.1
```

```
library(class)
```

```
## Warning: package 'class' was built under R version 4.1.1
```

```
library(ISLR)
```

```
## Warning: package 'ISLR' was built under R version 4.1.1
```

```
library(tidyverse)
```

```
## ── Attaching packages ─────────────────────────────────────── tidyverse 1.3.1 ──
```

```
## ✓ tibble  3.1.6     ✓ dplyr   1.0.8
## ✓ tidyr   1.2.0     ✓ stringr 1.4.0
## ✓ readr   2.1.2     ✓ forcats 0.5.1
## ✓ purrr   0.3.4
```

```
## Warning: package 'tibble' was built under R version 4.1.1
```

```
## Warning: package 'tidyr' was built under R version 4.1.1
```

```
## Warning: package 'readr' was built under R version 4.1.1
```

```
## Warning: package 'dplyr' was built under R version 4.1.1
```

```
## ── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## x purrr::lift()   masks caret::lift()
## x dplyr::recode() masks car::recode()
## x purrr::some()   masks car::some()
```

1. EDA

In this section of the code we will conduct a preliminary exploritory data analysis (EDA)

```
#Load in training data and inspect
train_data <- read.csv("trainSet.csv")
head(train_data)
```

```
##    feature_0  feature_1  feature_2   feature_3   feature_4  feature_5
## 1  1.5127910 -0.2434605  0.1434182  2.01858846  0.07622994 -0.4114531
## 2 -1.5007763 -0.2125875  1.2248391 -0.15984112 -0.56935064 -0.4114531
## 3  0.9477471  0.5812426 -0.3372133  0.77987360 -0.56935064 -0.4114531
## 4 -0.8415585 -0.2217837  0.5038918 -0.37729577  0.39902023 -0.4114531
## 5 -0.5590365 -0.5922597 -1.1783185 -0.41612696 -0.56935064 -0.4114531
## 6  0.9477471 -0.4654833 -0.5775291  0.08867847 -0.24656035  2.2551433
##    feature_6 feature_7 feature_8 feature_9 feature_10 feature_11 feature_12
## 1 -0.2519404         1         1         1          0          1          0
## 2 -0.2519404         8         2         1          0          0          0
## 3 -0.2519404         0         2         1          0          0          0
## 4 -0.2519404         9         1         1          0          0          0
## 5 -0.2519404         1         2         1          0          1          0
## 6  2.3528870         4         2         2          1          1          0
##   feature_13 feature_14 feature_15 labels
## 1          0          0          3      0
## 2          0          8          3      0
## 3          2          8          3      0
## 4          0          1          3      0
## 5          0          8          3      0
## 6          0          8          0      0
```

```
tail(train_data)
```

```
##          feature_0  feature_1  feature_2  feature_3   feature_4  feature_5
## 27121 -1.500776305  0.9146052 -0.2170554 -0.1831398  0.07622994 -0.4114531
## 27122 -1.029906417  0.1165055 -0.2170554 -0.2375035  0.07622994 -0.4114531
## 27123  1.136095070  1.6571993 -1.1783185 -0.8898674 -0.56935064 -0.4114531
## 27124 -0.559036528 -0.3344373 -1.2984763 -0.9053999 -0.56935064 -0.4114531
## 27125 -0.464862551 -0.3892861 -0.4573712 -0.7578414  0.72181052 -0.4114531
## 27126  0.006007338 -0.3764771 -0.3372133 -0.4122438 -0.24656035 -0.4114531
##        feature_6 feature_7 feature_8 feature_9 feature_10 feature_11 feature_12
## 27121 -0.2519404         8         2         1          0          0          0
## 27122 -0.2519404         1         1         1          0          1          0
## 27123 -0.2519404         4         1         2          0          0          0
## 27124 -0.2519404         1         1         1          0          0          0
## 27125 -0.2519404         2         1         0          0          1          0
## 27126 -0.2519404         3         1         0          0          0          0
##       feature_13 feature_14 feature_15 labels
## 27121          0          1          3      1
## 27122          0          8          3      0
## 27123          2          6          3      0
## 27124          2          6          3      0
## 27125          2          8          3      0
## 27126          0          1          3      0
```

```
dim(train_data)
```

```
## [1] 27126    17
```

```
str(train_data)
```

```
## '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 ...
```

str() has told us the data types of each variable. Given that labels is the response variable, we know: feature_0 to feature 6 are all continuous variables Feature_7 to feature_15 are all discrete, however some appear to be ordinal, taking on different values. E.g. Feature_7, 9, and 14 (possible 15 as well but will need to explore further to determine)

```
#Complete the same for the test data just to check the format and everything is the s
ame, however will not conduct and analysis on this as that would be deteremental the
 integrity of the model.
test_data <- read.csv("testSet.csv")
head(test_data)
```

```
##    feature_0  feature_1  feature_2  feature_3   feature_4  feature_5  feature_6
## 1 -1.0299064 -0.2224406  0.5038918 -0.9053999  0.07622994 -0.4114531 -0.2519404
## 2 -0.8415585 -0.3564425 -1.5387921 -0.1054775  2.98134255 -0.4114531 -0.2519404
## 3 -1.5007763 -0.2648089 -1.4186342  0.3993280 -0.56935064 -0.4114531 -0.2519404
## 4 -0.2765146 -0.4024236  1.8256285 -0.8976337  6.20924545 -0.4114531 -0.2519404
## 5 -0.9357324 -0.2677648  1.5853127  0.0537304 -0.56935064 -0.4114531 -0.2519404
## 6 -1.5949503 -0.4221298 -1.1783185  0.5119384 -0.24656035 -0.4114531 -0.2519404
##   feature_7 feature_8 feature_9 feature_10 feature_11 feature_12 feature_13
## 1         0         1         1          0          1          0          2
## 2         0         1         1          0          1          0          2
## 3         7         2         1          0          1          1          2
## 4         0         1         1          0          1          0          0
## 5         0         2         1          0          1          0          0
## 6         7         2         1          0          1          1          2
##   feature_14 feature_15 labels
## 1          6          3      0
## 2          6          3      0
## 3          6          3      0
## 4          5          3      0
## 5          4          3      0
## 6          8          3      0
```

```
tail(test_data)
```

```
##         feature_0  feature_1  feature_2   feature_3  feature_4  feature_5
## 6777  1.795312914 -0.4434781  0.9845233 -0.20643855 -0.5693506 -0.4114531
## 6778 -0.935732439  0.2219335  0.5038918 -0.42777631 -0.5693506 -0.4114531
## 6779 -1.406602327 -0.4598999  0.7442076  0.71774370 -0.2465603 -0.4114531
## 6780 -0.464862551 -0.3754918 -0.5775291  0.04596417 -0.2465603  3.0641107
## 6781  0.006007338  1.0614162 -1.4186342  0.52358776  2.6585523 -0.4114531
## 6782  0.476877226  0.8610701 -0.3372133 -0.70347774 -0.2465603 -0.4114531
##       feature_6 feature_7 feature_8 feature_9 feature_10 feature_11 feature_12
## 6777 -0.2519404         9         2         1          0          0          0
## 6778 -0.2519404         7         2         1          0          1          0
## 6779 -0.2519404         4         2         1          0          0          0
## 6780  0.1821975         1         2         1          0          1          1
## 6781 -0.2519404         4         2         2          0          1          0
## 6782 -0.2519404         4         1         2          0          0          0
##      feature_13 feature_14 feature_15 labels
## 6777          0          5          3      0
## 6778          2          8          3      0
## 6779          0          5          3      0
## 6780          0          8          0      0
## 6781          2          6          3      0
## 6782          0          1          3      0
```

```
dim(test_data)
```

```
## [1] 6782   17
```

```
str(test_data)
```

```
## '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 ...
```

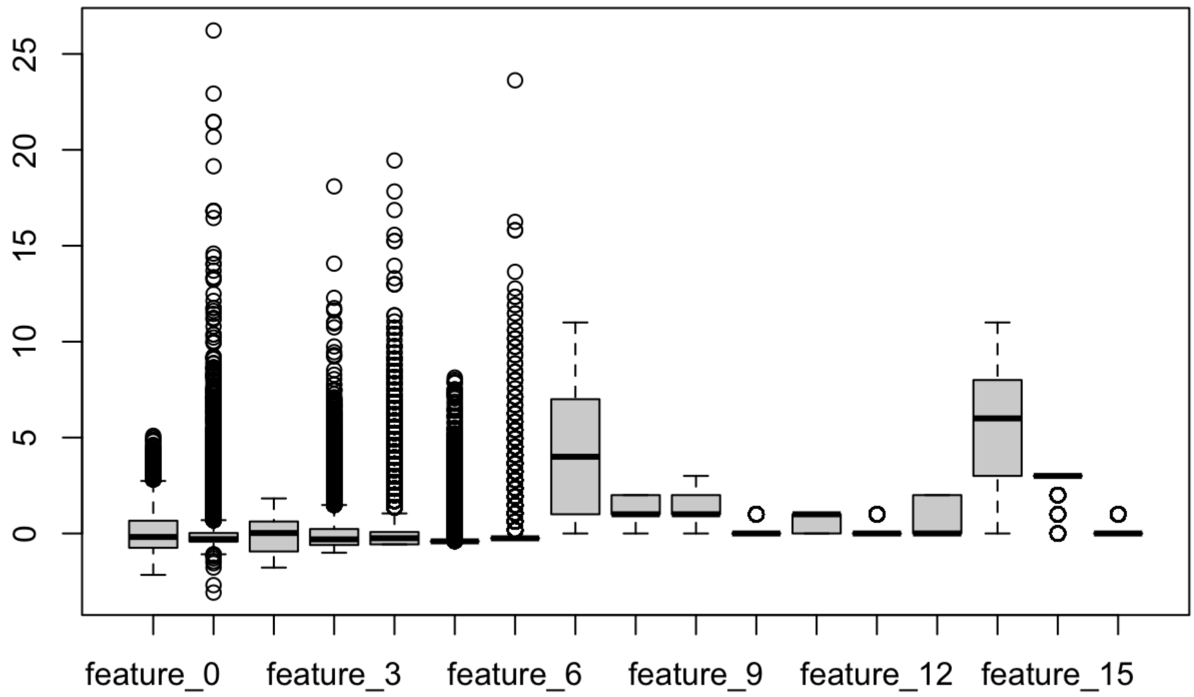
```
#Everything look ok. Will revisit test data at model implementation.
```

Univeriate Analysis

```
summary(train_data)
```

```
##    feature_0           feature_1           feature_2
##  Min.   :-2.159994   Min.   :-3.081149   Min.   :-1.779108
##  1st Qu.:-0.747384   1st Qu.:-0.422458   1st Qu.:-0.938003
##  Median :-0.182341   Median :-0.296996   Median : 0.023260
##  Mean   :-0.004908   Mean   : 0.001337   Mean   : 0.003681
##  3rd Qu.: 0.665225   3rd Qu.: 0.023886   3rd Qu.: 0.624050
##  Max.   : 5.091402   Max.   :26.222907   Max.   : 1.825629
##    feature_3           feature_4           feature_5
##  Min.   :-1.002478   Min.   :-0.569351   Min.   :-0.411453
##  1st Qu.:-0.602517   1st Qu.:-0.569351   1st Qu.:-0.411453
##  Median :-0.307400   Median :-0.246560   Median :-0.411453
##  Mean   :-0.002433   Mean   :-0.000047   Mean   :-0.002946
##  3rd Qu.: 0.232354   3rd Qu.: 0.076230   3rd Qu.:-0.411453
##  Max.   :18.094700   Max.   :19.443647   Max.   : 8.127648
##    feature_6           feature_7        feature_8      feature_9
##  Min.   :-0.251940   Min.   : 0.000   Min.   :0.00   Min.   :0.000
##  1st Qu.:-0.251940   1st Qu.: 1.000   1st Qu.:1.00   1st Qu.:1.000
##  Median :-0.251940   Median : 4.000   Median :1.00   Median :1.000
##  Mean   :-0.009104   Mean   : 4.336   Mean   :1.17   Mean   :1.226
##  3rd Qu.:-0.251940   3rd Qu.: 7.000   3rd Qu.:2.00   3rd Qu.:2.000
##  Max.   :23.625644   Max.   :11.000   Max.   :2.00   Max.   :3.000
##    feature_10        feature_11       feature_12      feature_13
##  Min.   :0.00000   Min.   :0.0000   Min.   :0.000   Min.   :0.0000
##  1st Qu.:0.00000   1st Qu.:0.0000   1st Qu.:0.000   1st Qu.:0.0000
##  Median :0.00000   Median :1.0000   Median :0.000   Median :0.0000
##  Mean   :0.01788   Mean   :0.5522   Mean   :0.159   Mean   :0.6365
##  3rd Qu.:0.00000   3rd Qu.:1.0000   3rd Qu.:0.000   3rd Qu.:2.0000
##  Max.   :1.00000   Max.   :1.0000   Max.   :1.000   Max.   :2.0000
##    feature_14       feature_15        labels
##  Min.   : 0.000   Min.   :0.000   Min.   :0.0000
##  1st Qu.: 3.000   1st Qu.:3.000   1st Qu.:0.0000
##  Median : 6.000   Median :3.000   Median :0.0000
##  Mean   : 5.513   Mean   :2.562   Mean   :0.1173
##  3rd Qu.: 8.000   3rd Qu.:3.000   3rd Qu.:0.0000
##  Max.   :11.000   Max.   :3.000   Max.   :1.0000
```

```
boxplot(train_data)
```



Looking at the boxplot we can see that a lot of the continuous variables have significant portions of outliers (e.g. features 0,1,3,4). This would impact a later model, and so it may make sense to remove some of these outerlers via applying a IQR factor (ref: https://www.r-bloggers.com/2020/01/how-to-remove-outliers-in-r/ (https://www.r-bloggers.com/2020/01/how-to-remove-outliers-in-r/)).

Summary stats also show that we may not be dealing with data on the same scale, so we may need to normalize at some point.

Features 7 to 15 seem to be categorical looking at the structure of the data. The below codeblock generates tables to inspect the frequency of discrete variable data.

```
#Apply table)to get a understanding of frequency of frequency of discrete variables)
table(train_data$feature_7)
```

```
##
##    0    1    2    3    4    5    6    7    8    9   10   11
## 3085 5854  898  756 5729 1329  939 2454  565 4531  809  177
```

```
table(train_data$feature_8)
```

```
##
##     0     1     2
##  3049 16413  7664
```

```
table(train_data$feature_9)
```

```
## 
##     0     1     2     3 
##  4133 13853  8028  1112
```

```
table(train_data$feature_10)
```

```
## 
##     0     1 
## 26641   485
```

```
table(train_data$feature_11)
```

```
## 
##     0     1 
## 12147 14979
```

```
table(train_data$feature_12)
```

```
## 
##     0     1 
## 22814  4312
```

```
table(train_data$feature_13)
```

```
## 
##     0     1     2 
## 17617  1751  7758
```

```
table(train_data$feature_14)
```

```
## 
##    0    1    2    3    4    5    6    7    8    9   10   11 
## 1758 3744  117 1612  885 4161 3205  289 8170 2406  436  343
```

```
table(train_data$feature_15)
```

```
## 
##     0     1     2     3 
##  2925  1091   915 22195
```

Now we'll remove outlines.

Investigate the impact of removing outlines based off of IQR method

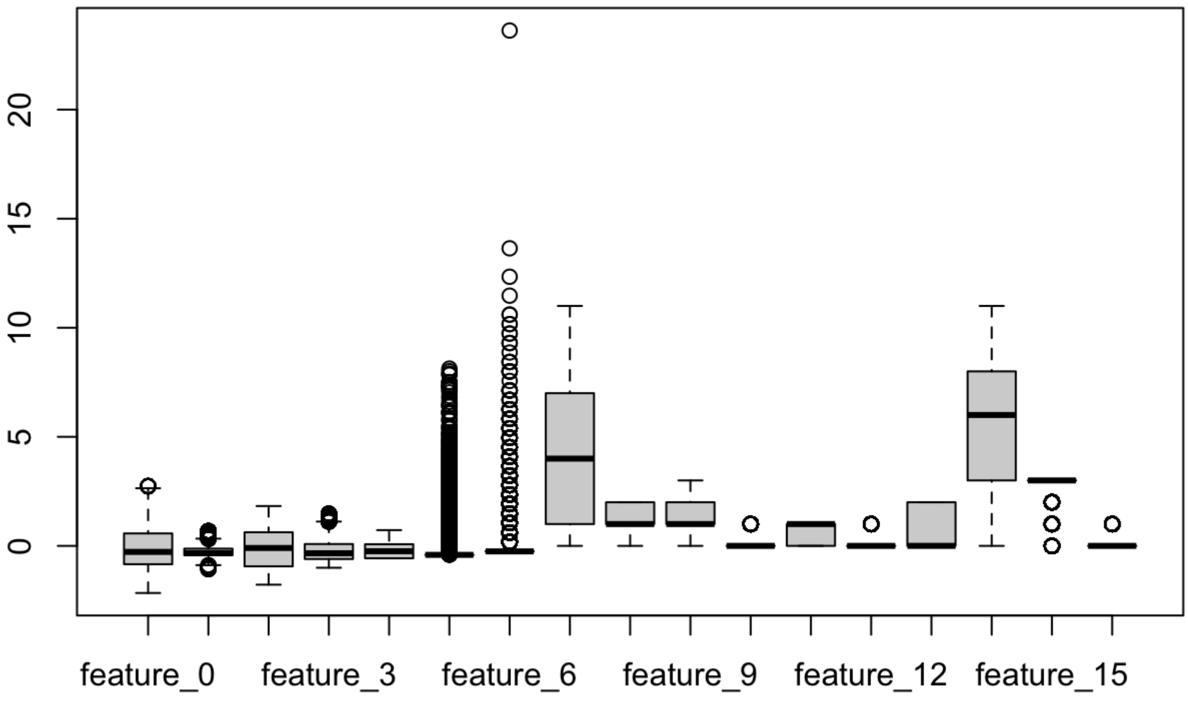
From looking at the boxplots in more detail, Features 0, 1, 3, 4 look to have significant outliers. Therefore IQR method will be applied to exclude the outliers. Lower end outlier is defined as Q1 - 1.5(IQR), an upper outlier is defined as Q3 + 1.5(IQR)

In the model development I apply the IQR removal of outliers to several features. However as I experimented further, I found that feature_4 was the opnly one that positively impacted the accuracy of the final model with the removal of outliers. My final model reflects this (and is detailed in the final section of this assignment).

```
#Feature_0
Q_0 <- quantile(train_data$feature_0, probs=c(0.25, 0.75), na.rm = FALSE)
iqr_0 <- IQR(train_data$feature_0)
up_0 <- Q_0[2]+1.5*iqr_0
low_0 <- Q_0[1] - 1.5*iqr_0

#extract the outliers data
train_data <- subset(train_data, train_data$feature_0 > low_0 & train_data$feature_0
 < up_0)

#Feature 1
Q_1 <- quantile(train_data$feature_1, probs=c(0.25, 0.75), na.rm = FALSE)
iqr_1 <- IQR(train_data$feature_1)
up_1 <- Q_1[2]+1.5*iqr_1
low_1 <- Q_1[1] - 1.5*iqr_1

#extract the outliers data
train_data <- subset(train_data, train_data$feature_1 > low_1 & train_data$feature_1
 < up_1)

#Feature 3
Q_3 <- quantile(train_data$feature_3, probs=c(0.25, 0.75), na.rm = FALSE)
iqr_3 <- IQR(train_data$feature_3)
up_3 <- Q_3[2]+1.5*iqr_3
low_3 <- Q_3[1] - 1.5*iqr_3

#extract the outliers data
train_data <- subset(train_data, train_data$feature_3 > low_3 & train_data$feature_3
 < up_3)

#Feature 4
Q_4 <- quantile(train_data$feature_4, probs=c(0.25, 0.75), na.rm = FALSE)
iqr_4 <- IQR(train_data$feature_4)
up_4 <- Q_4[2]+1.5*iqr_4
low_4 <- Q_4[1] - 1.5*iqr_4

#extract the outliers data
train_data <- subset(train_data, train_data$feature_4 > low_4 & train_data$feature_4
 < up_4)
```

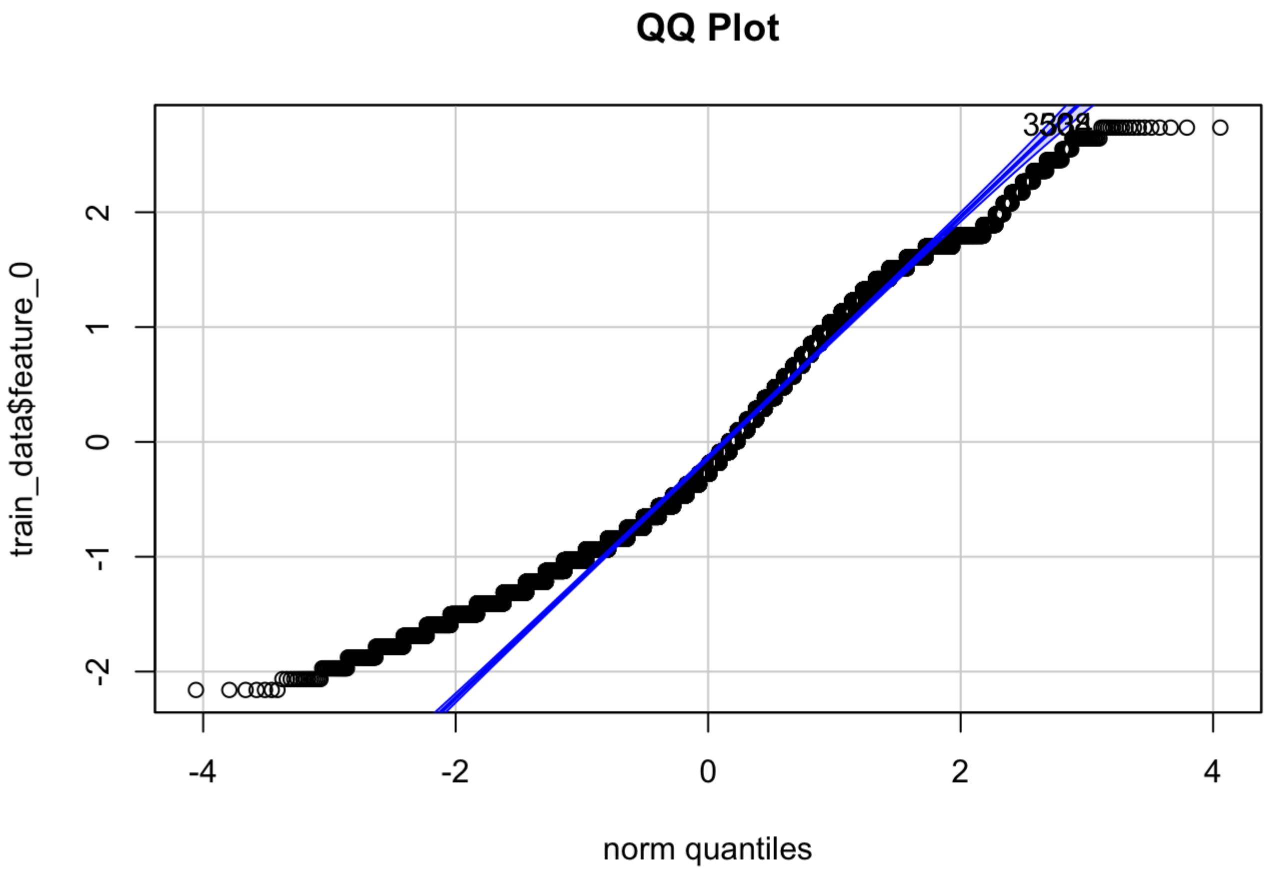
Applying a boxplot again, the removal of the outliser can be seen.

```
boxplot(train_data)
```



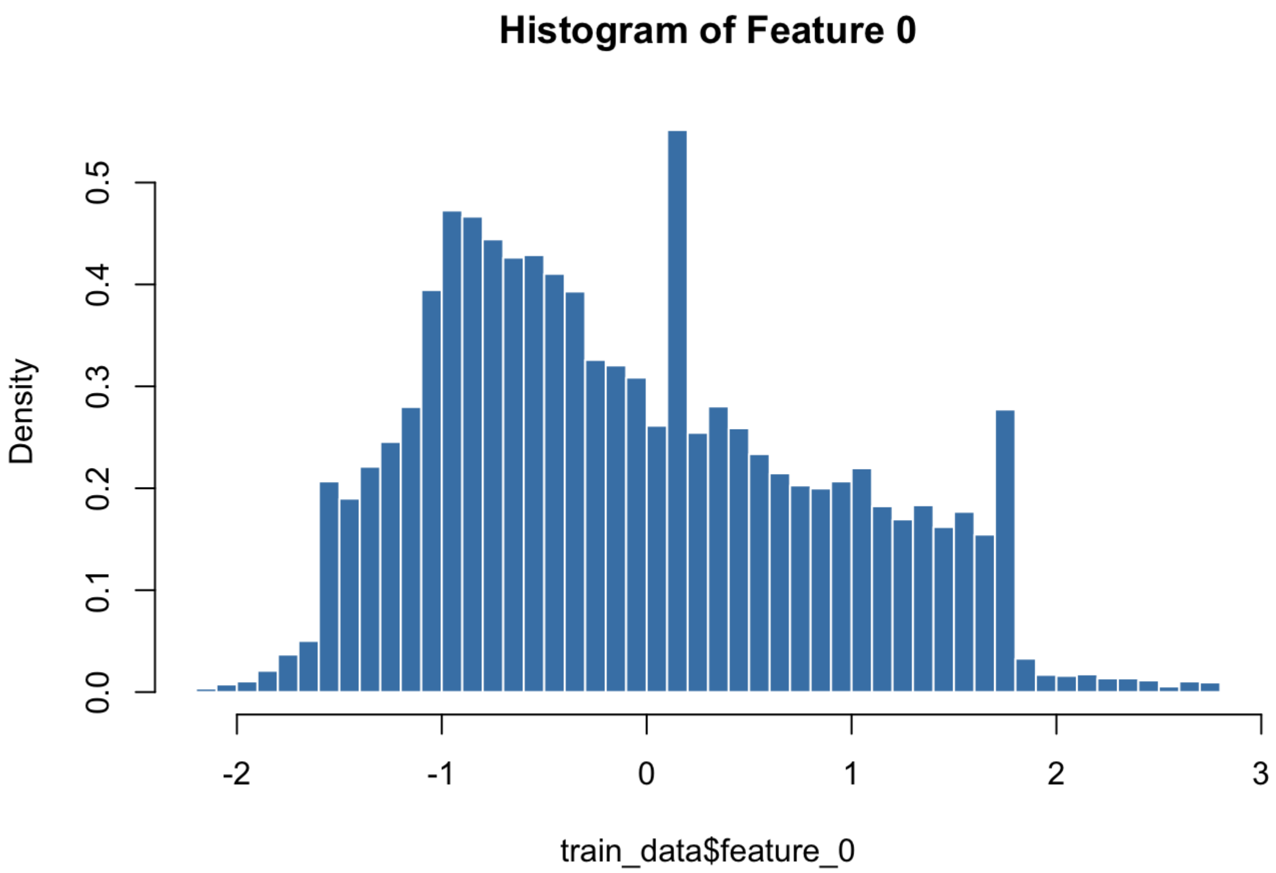
also want to see the distrubtions of the data to see if any patterns/trends appear. Applying hist() and qq plots()

```
qqPlot(train_data$feature_0, main = "QQ Plot")
```

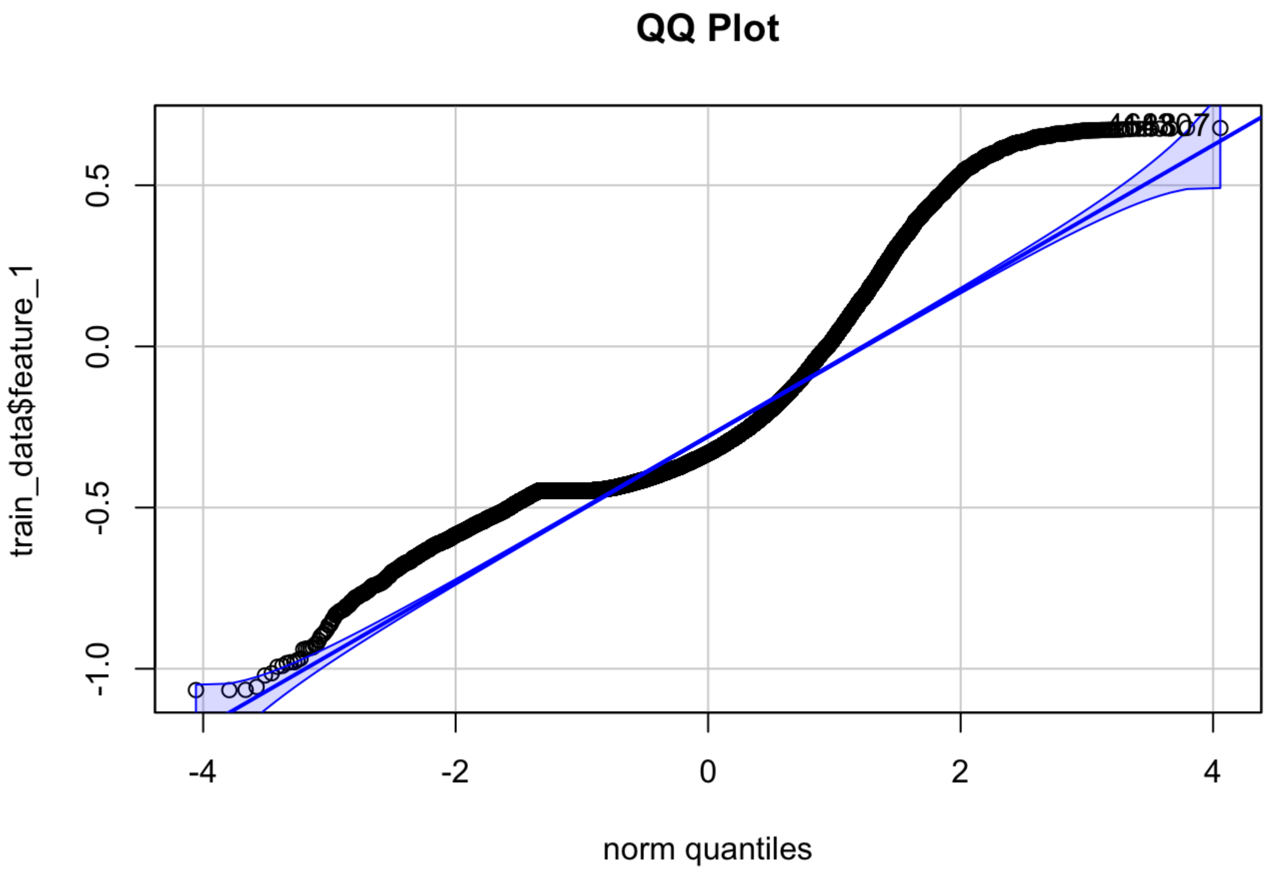


```
## [1]  508 3334
```

```
hist(train_data$feature_0, n = 40, freq  =FALSE, main = "Histogram of Feature 0", bor
der = "white", col = "steelblue")
```

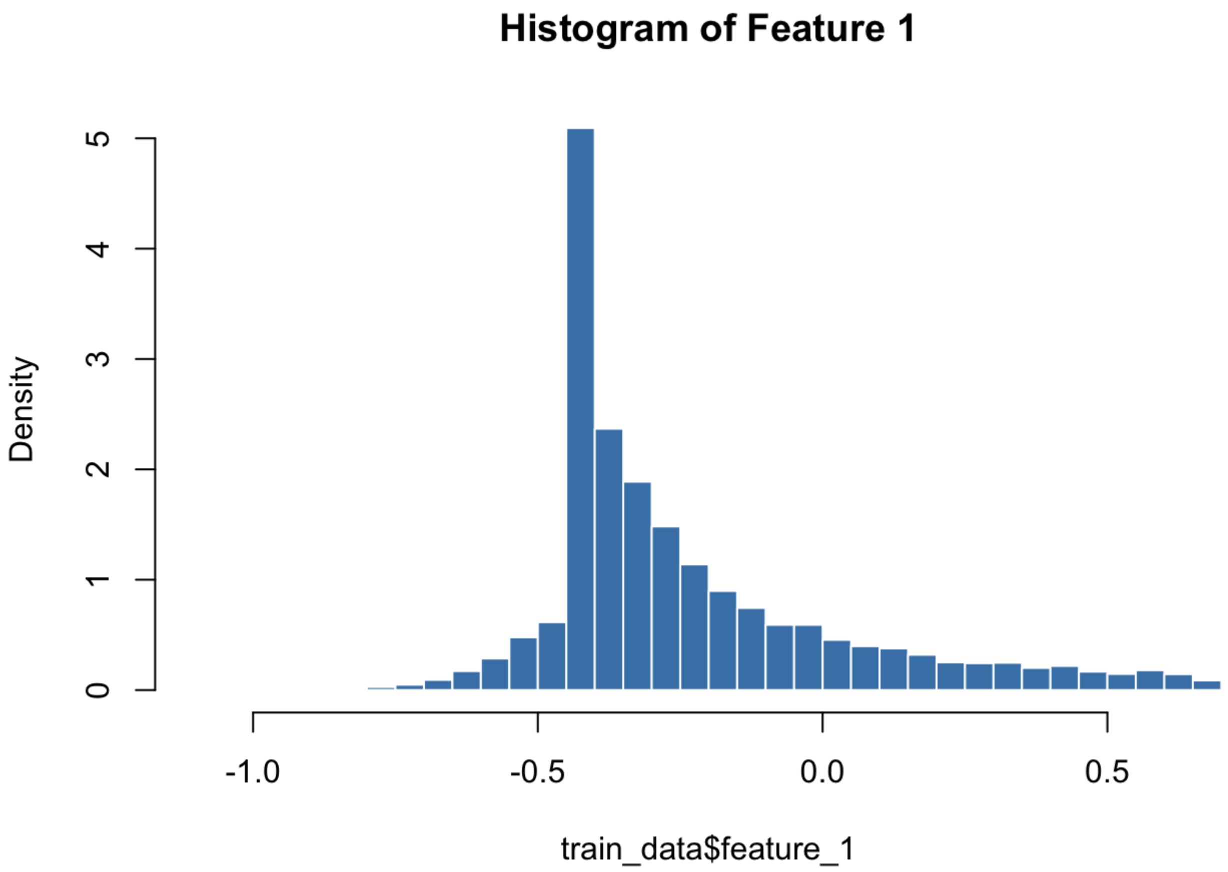



```
qqPlot(train_data$feature_1, main = "QQ Plot")
```

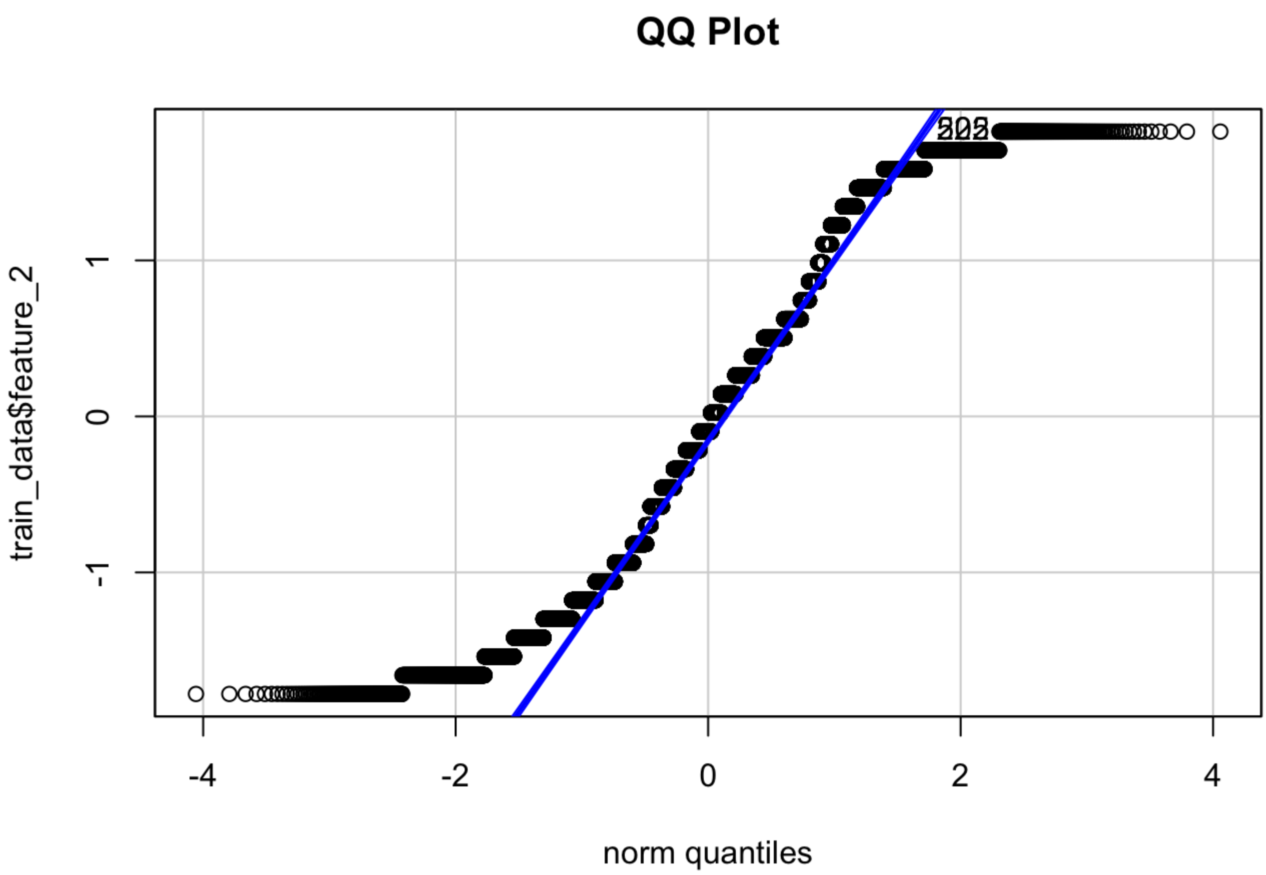



```
## [1]  4688 14307
```

```
hist(train_data$feature_1, n = 50, freq  =FALSE, main = "Histogram of Feature 1", bor
der = "white", col = "steelblue")
```

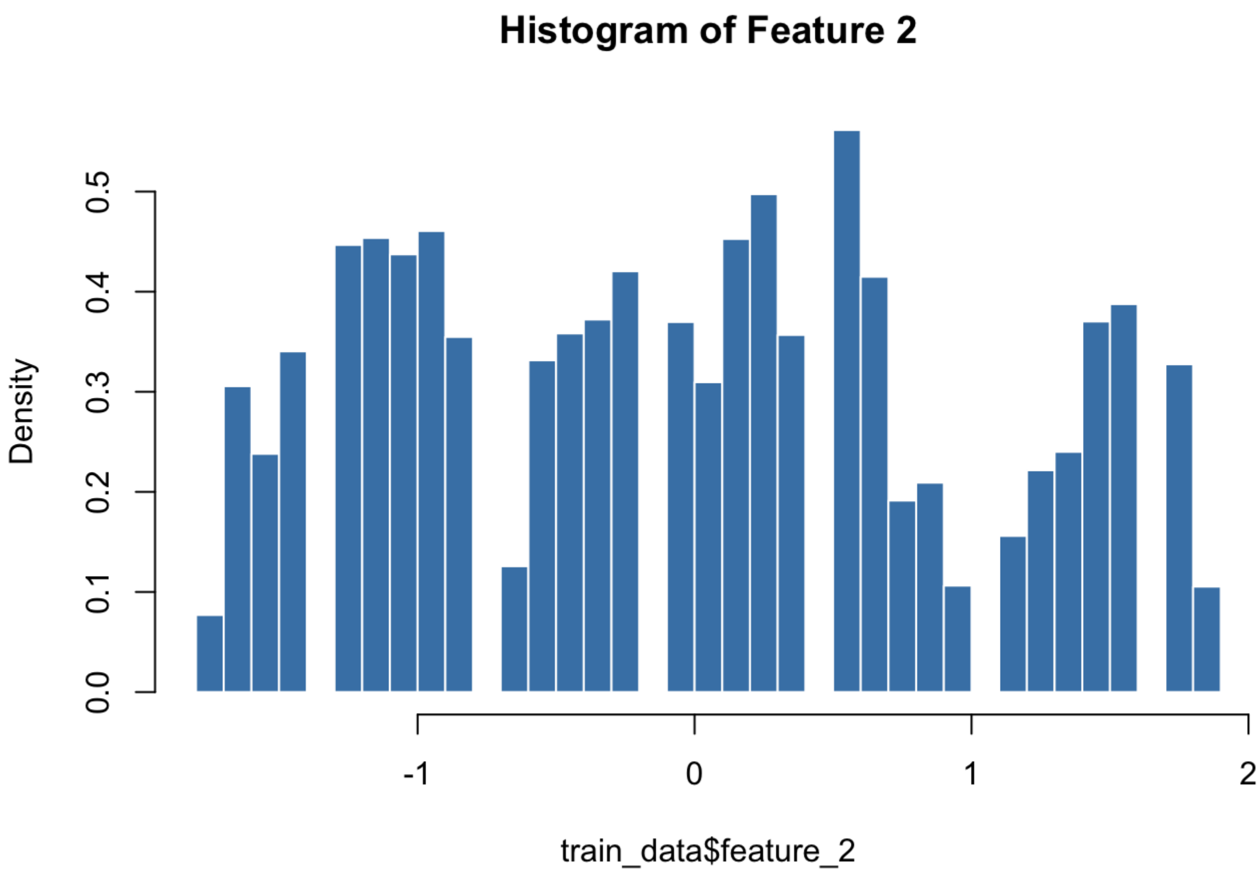



```
qqPlot(train_data$feature_2, main = "QQ Plot")
```



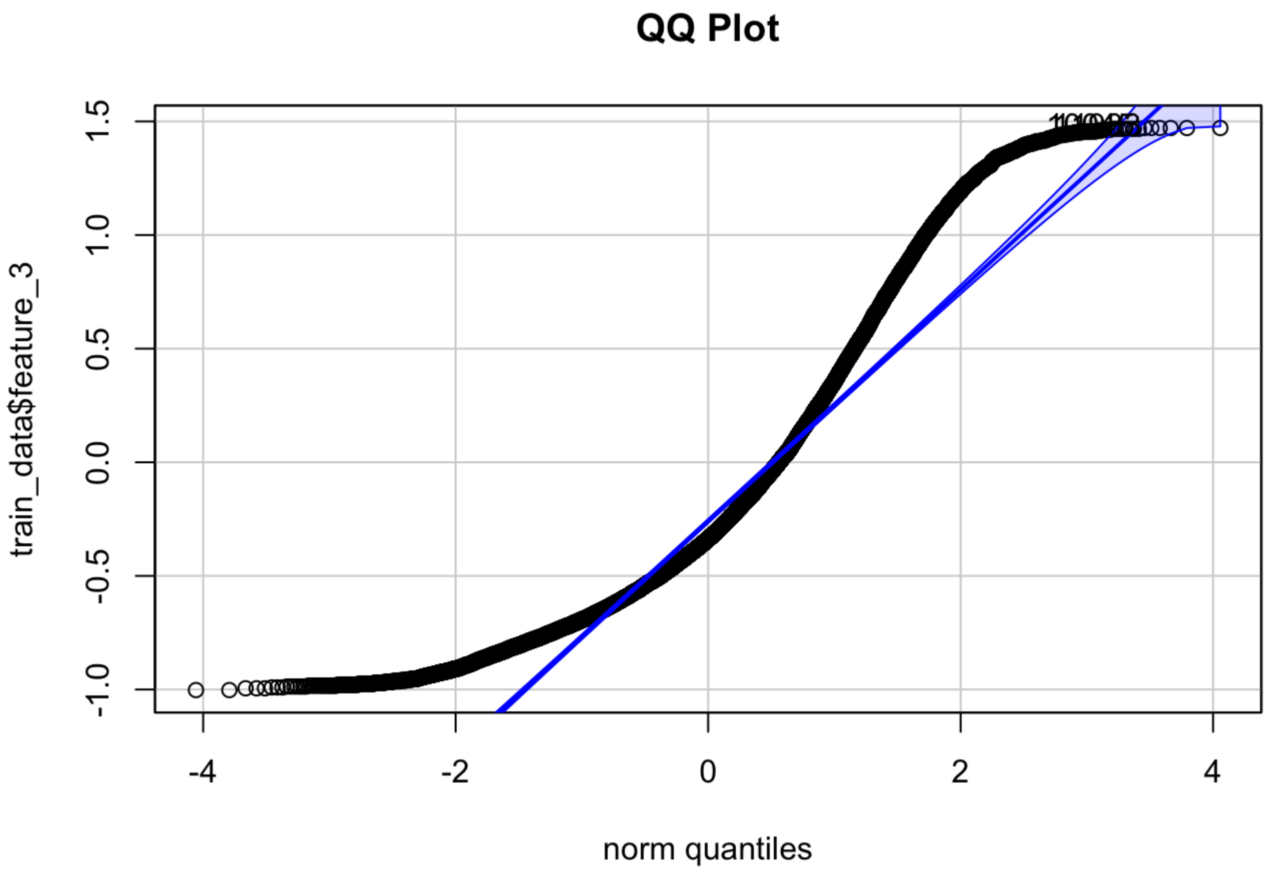
```
## [1] 222 505
```

```
hist(train_data$feature_2, n = 50, freq  =FALSE, main = "Histogram of Feature 2", bor
der = "white", col = "steelblue")
```



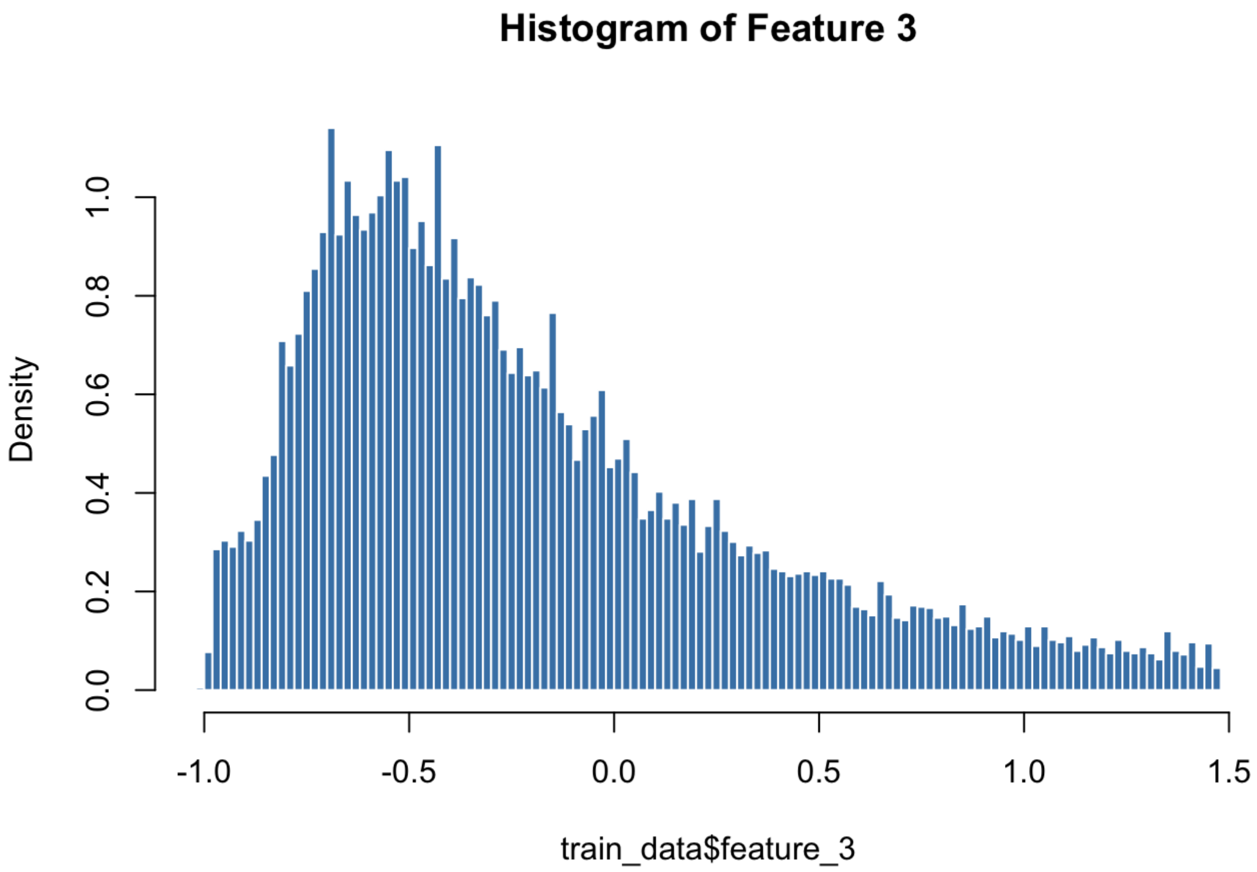
Looking at the above feature_2 plot, it now appears that this features has a discrete number of values as opposed to being truely continouous. This will impact how this data is handled, and will also mean we don't want to remove outliers.

```
qqPlot(train_data$feature_3, main = "QQ Plot")
```



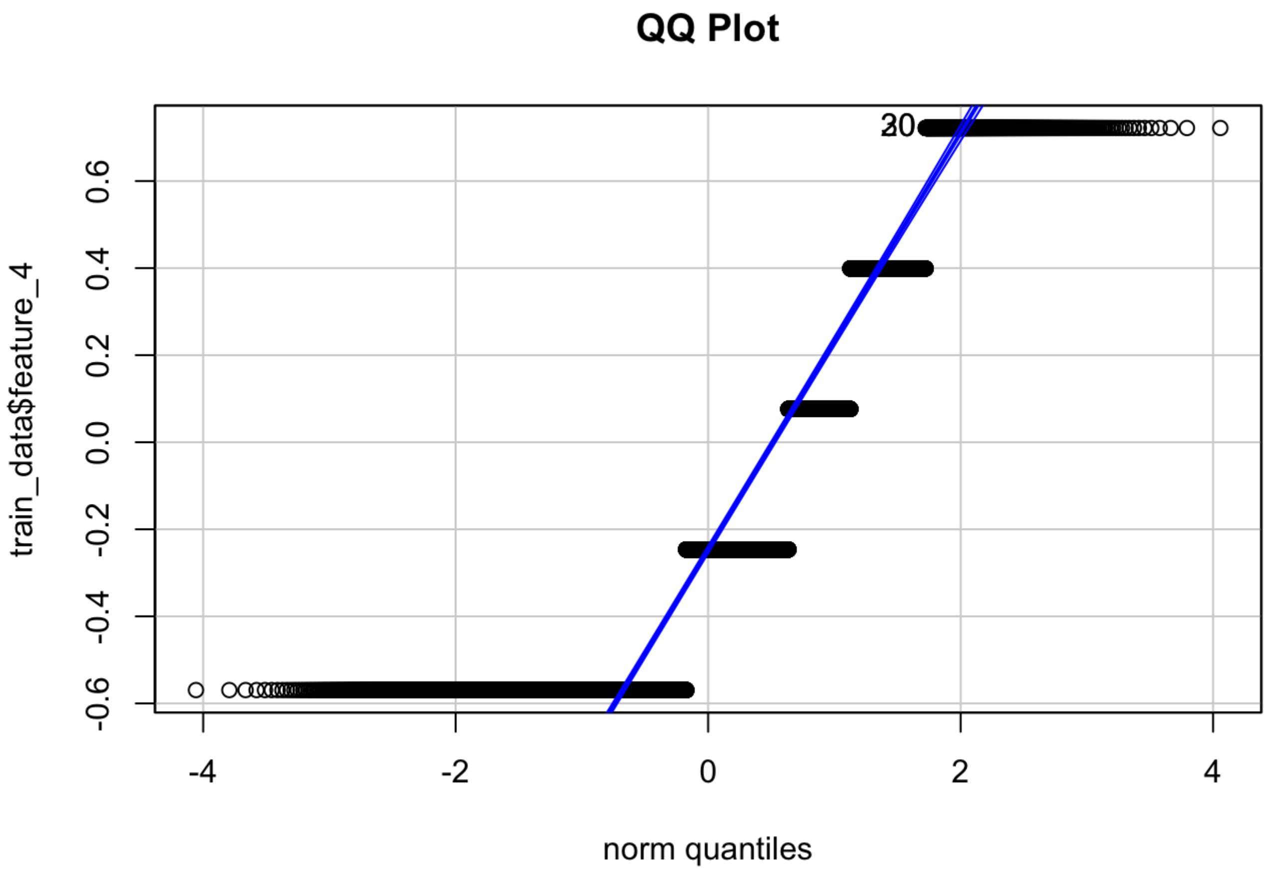

```
## [1] 10045 11933
```

```
#Change bin frequency for better insight into distribution.
hist(train_data$feature_3, n = 100, freq  =FALSE, main = "Histogram of Feature 3", bo
rder = "white", col = "steelblue")
```



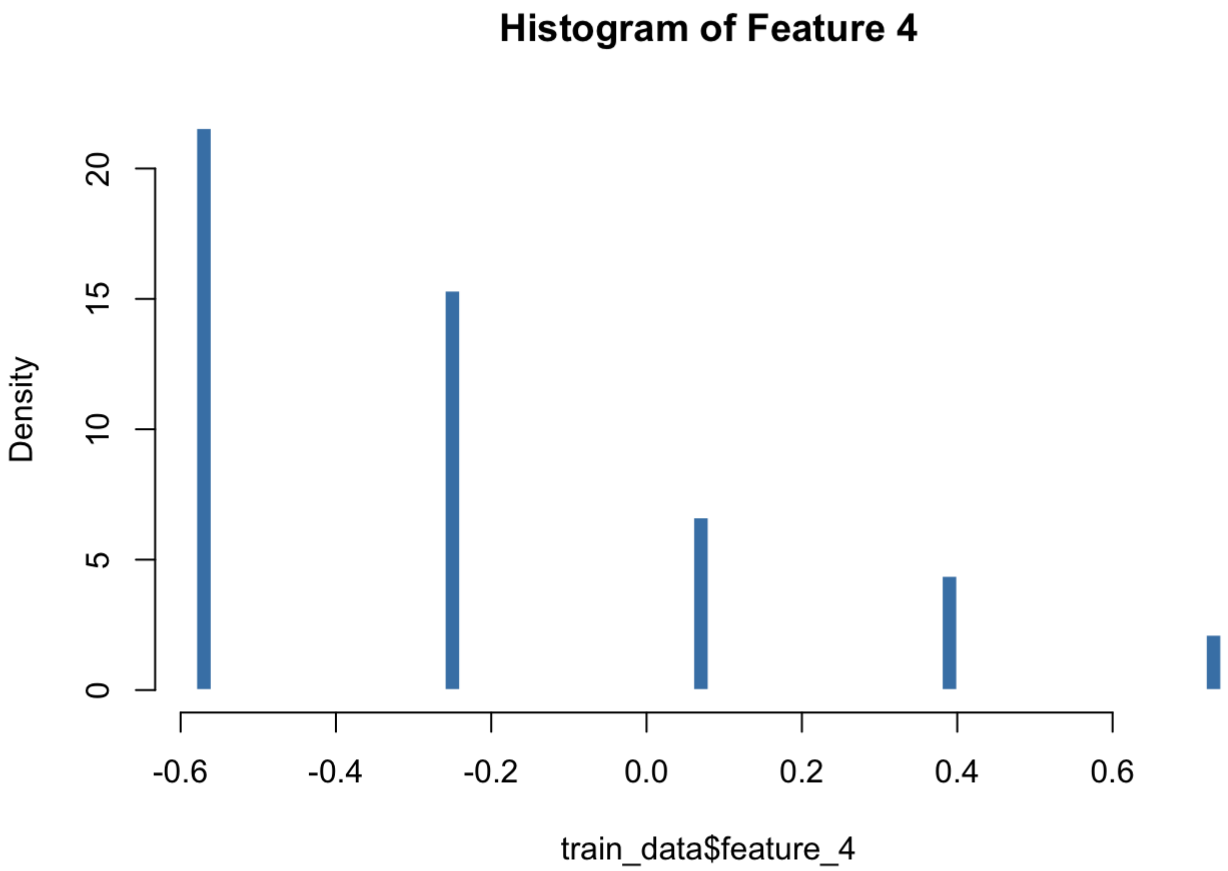
Feature 3 appears to follow a right skewed distibution

```
qqPlot(train_data$feature_4, main = "QQ Plot")
```



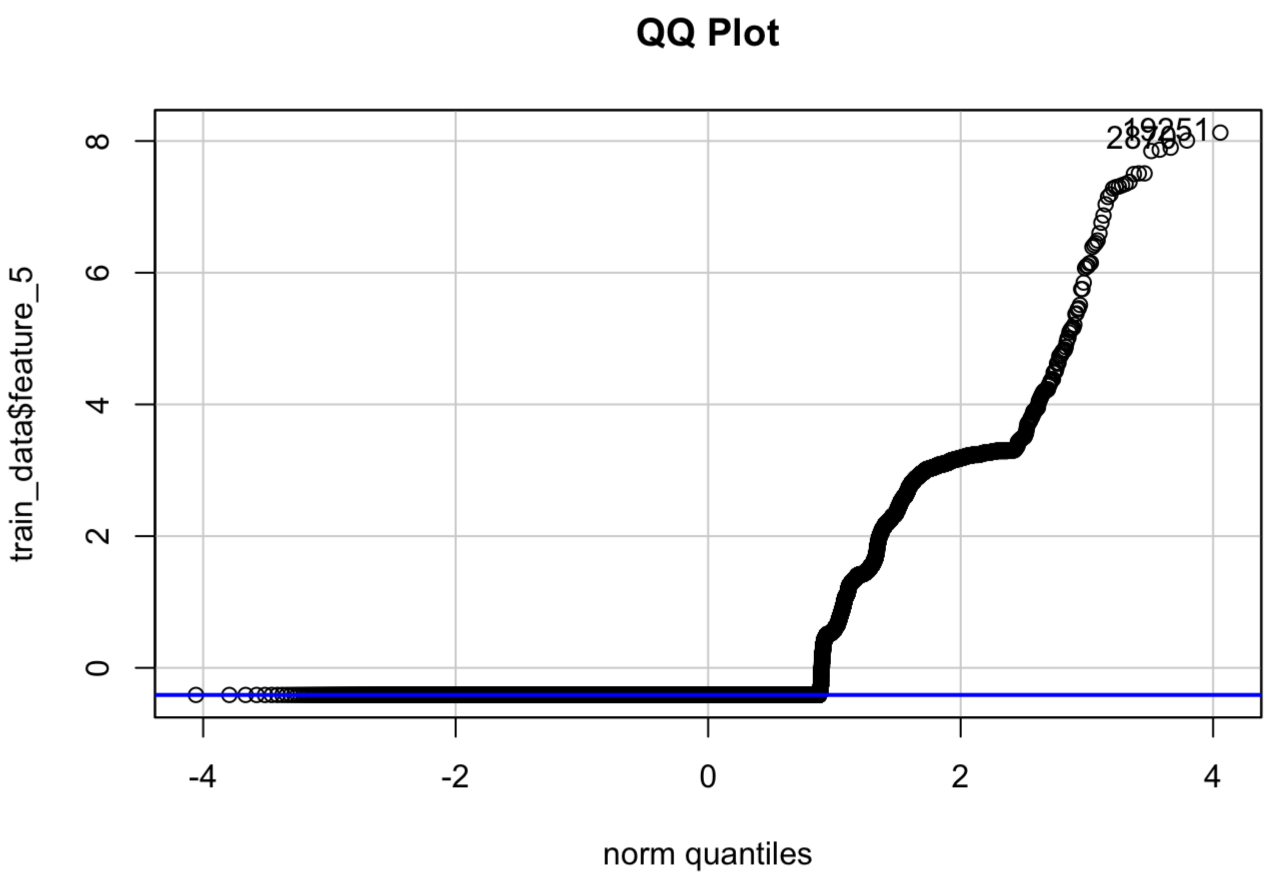

```
## [1] 20 30
```

```
hist(train_data$feature_4, n = 50, freq  =FALSE, main = "Histogram of Feature 4", bor
der = "white", col = "steelblue")
```



Appears to be descrete after clearing up the outliers.

```
qqPlot(train_data$feature_5, main = "QQ Plot")
```



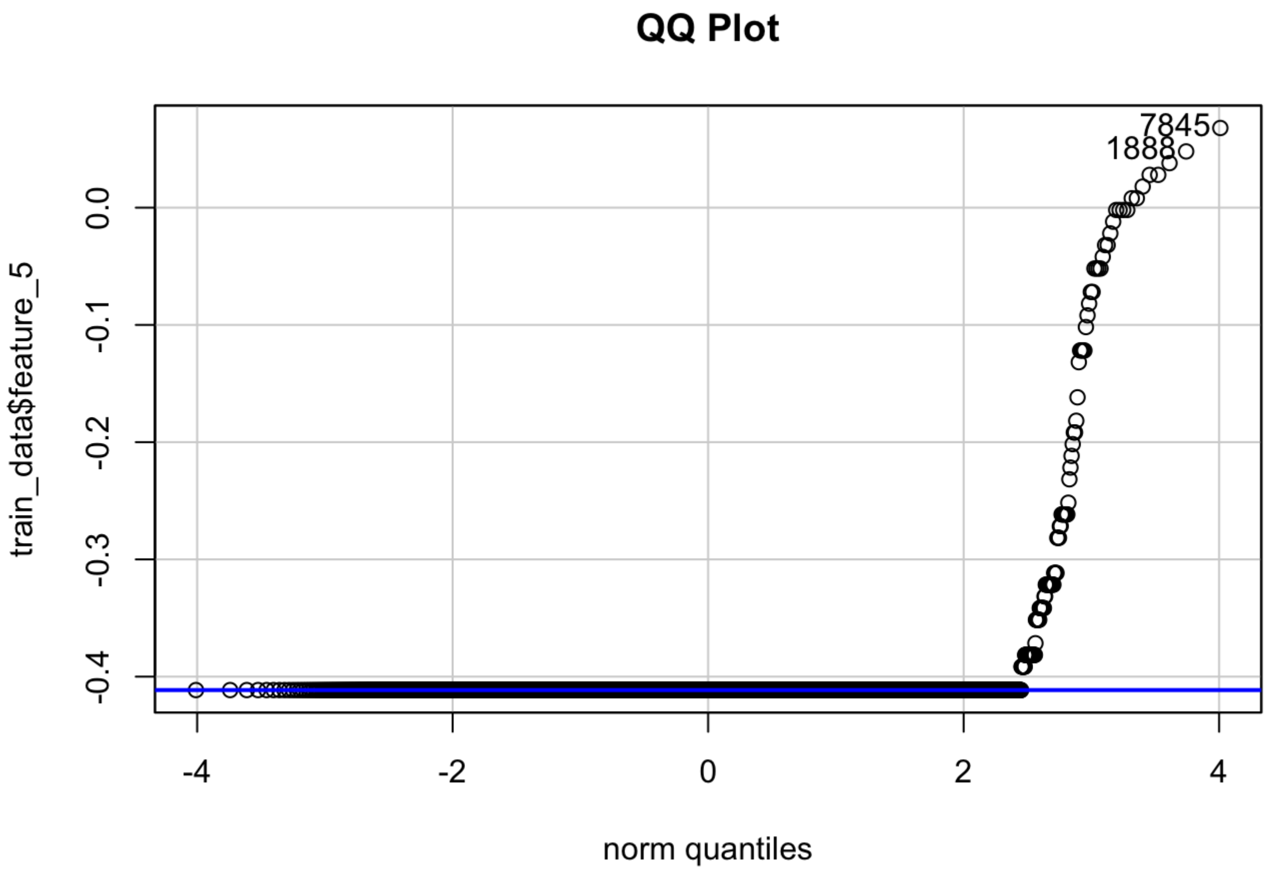
```
## [1] 19251  2870
```

```
hist(train_data$feature_5, n = 50, freq  =FALSE, main = "Histogram of Feature 5", bor
der = "white", col = "steelblue")
```

The histogram explains the appearence of a significant number of outliers on the original boxplot. There is a heavy concentration of data towards a single value. Given this and the qq plot, I will remove the outliers as I suspect they will negatively influence any future models.

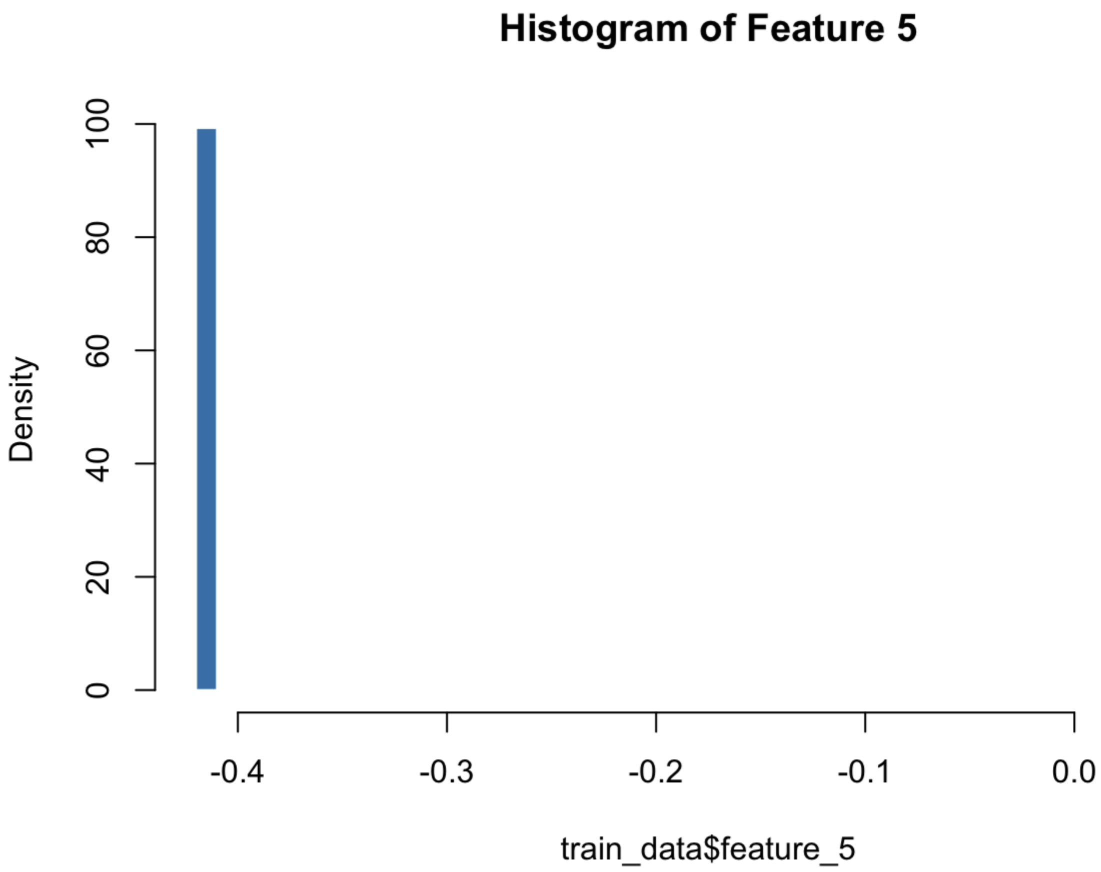
```
#Feature 5
Q_5 <- quantile(train_data$feature_5, probs=c(0.25, 0.75), na.rm = FALSE)
iqr_5 <- IQR(train_data$feature_5)
up_5 <- Q_4[2]+1.5*iqr_5
low_5 <- Q_4[1] - 1.5*iqr_5

#extract the outliers data
train_data <- subset(train_data, train_data$feature_5 > low_5 & train_data$feature_5
 < up_5)
```

```
qqPlot(train_data$feature_5, main = "QQ Plot")
```

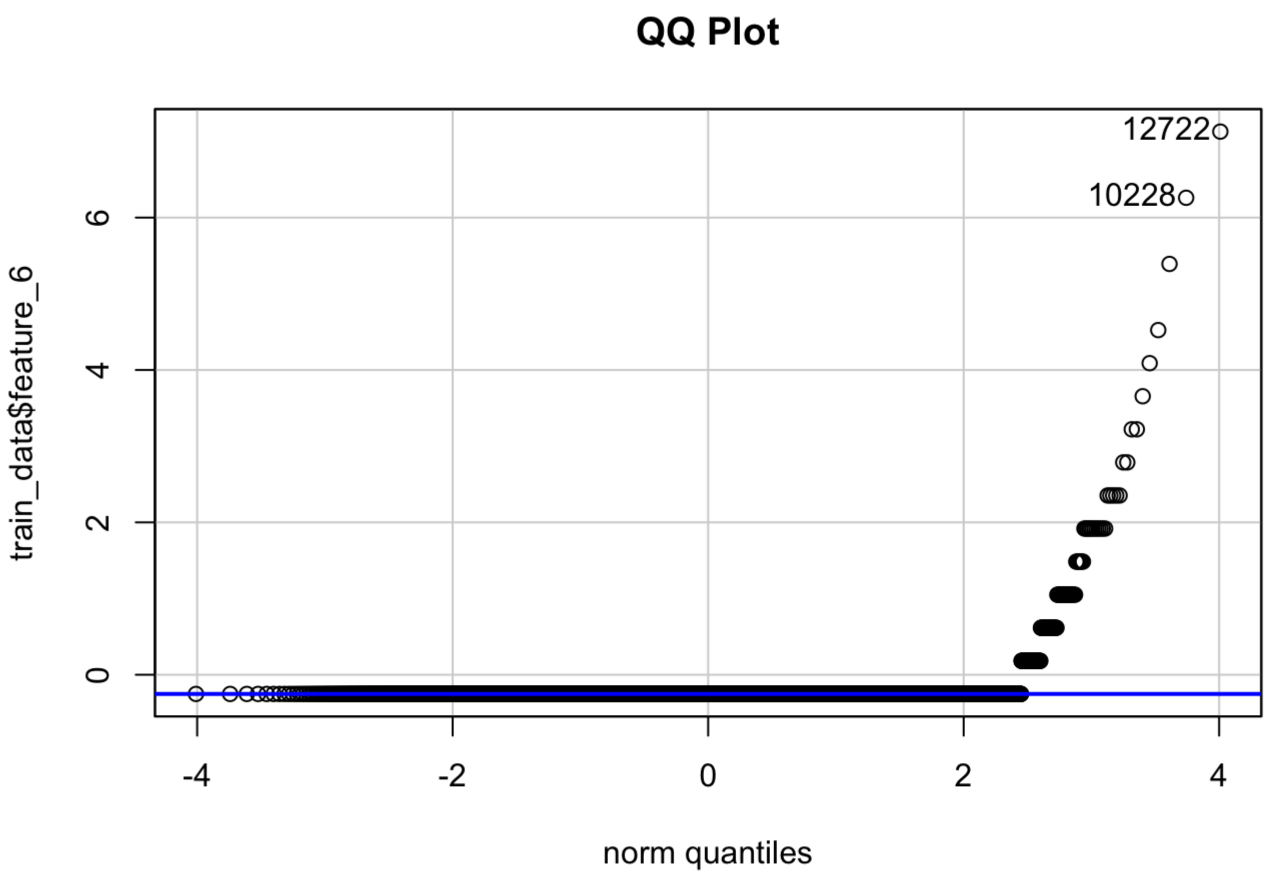



```
## [1] 7845 1888
```

```
hist(train_data$feature_5, n = 50, freq  =FALSE, main = "Histogram of Feature 5", bor
der = "white", col = "steelblue")
```

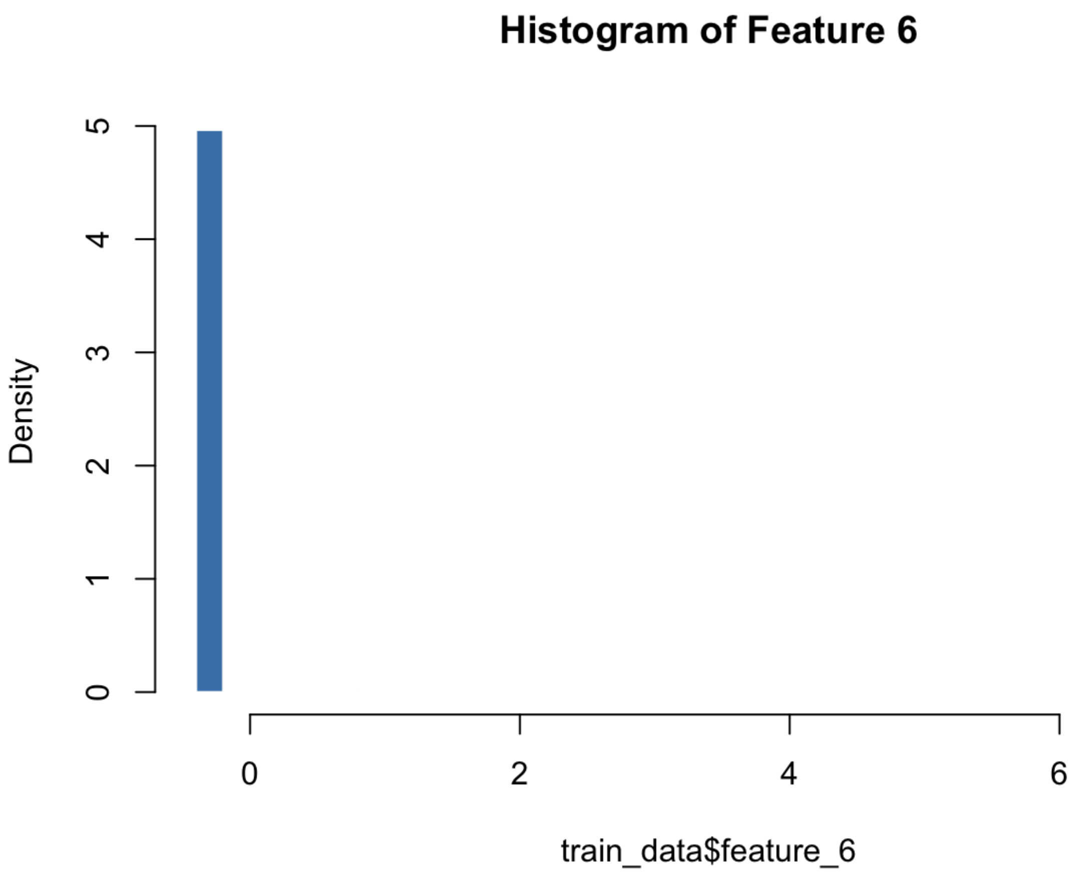



```
qqPlot(train_data$feature_6, main = "QQ Plot")
```



```
## [1] 12722 10228
```

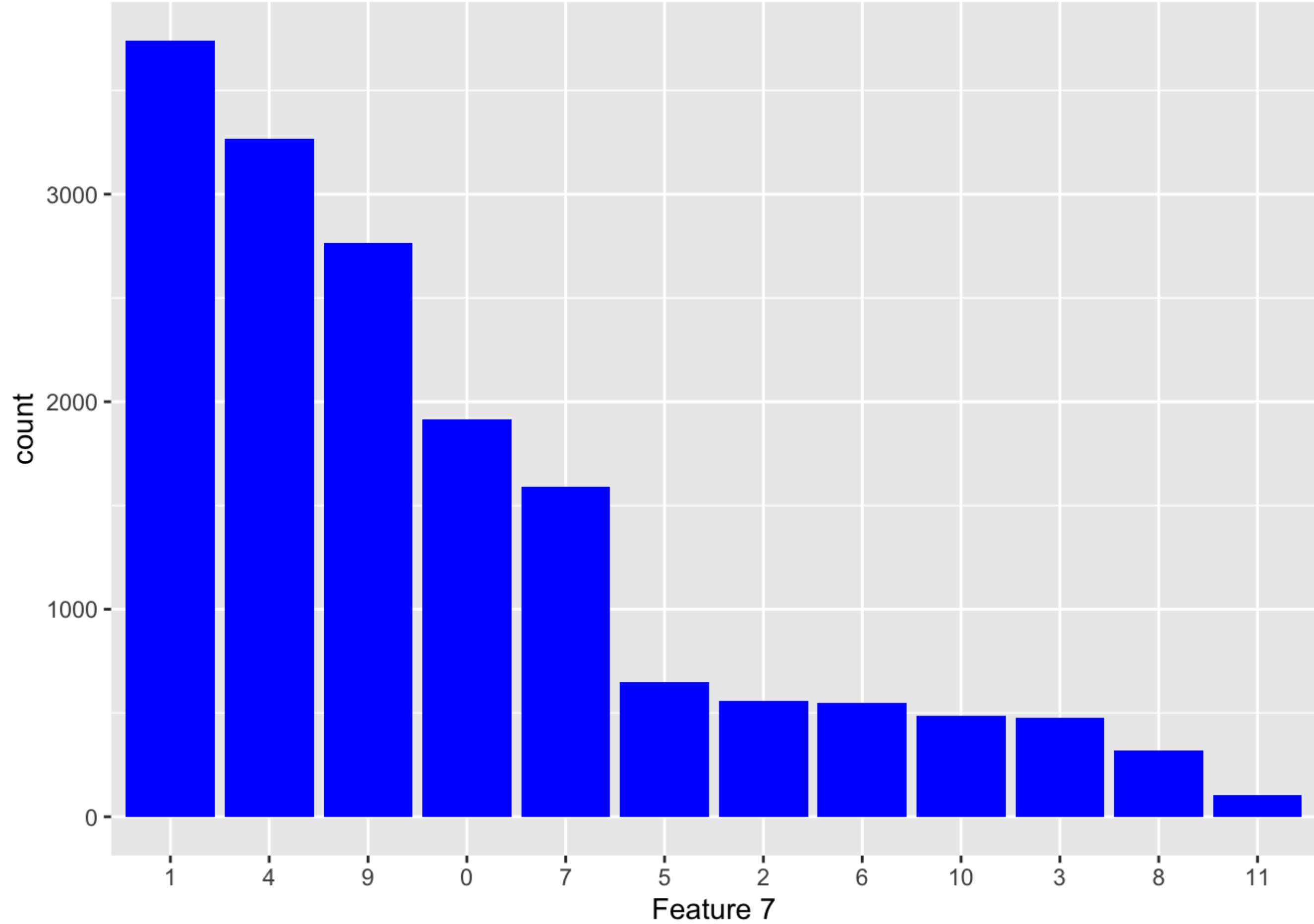
```
hist(train_data$feature_6, n = 50, freq  =FALSE, main = "Histogram of Feature 6", bor
der = "white", col = "steelblue")
```



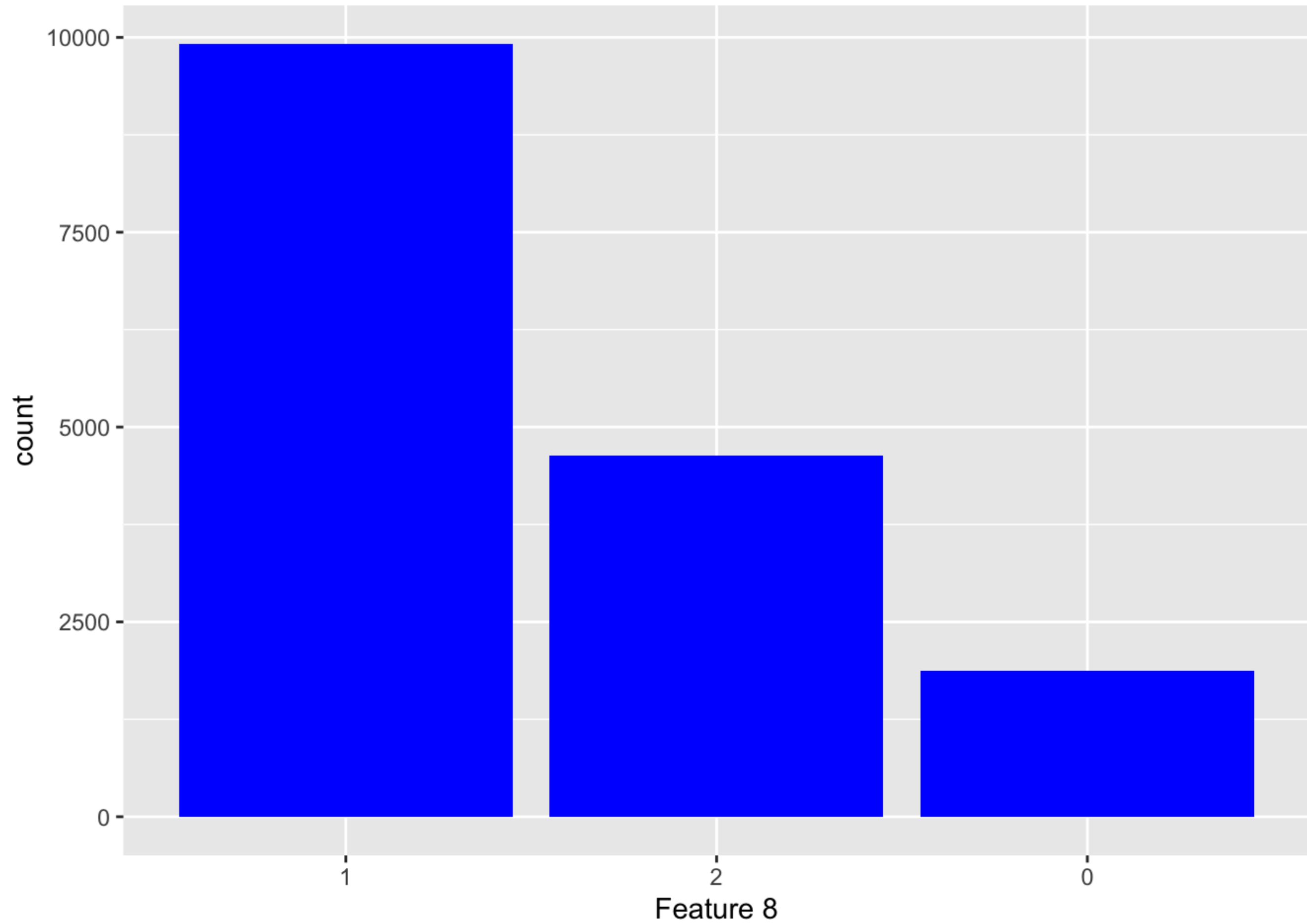
Up until this point, we have looked at continuous variable distributions, now we will examine some of the categorical variables to assess their distributions.

Ref: https://www.r-bloggers.com/2021/08/how-to-plot-categorical-data-in-r-quick-guide/ (https://www.r-bloggers.com/2021/08/how-to-plot-categorical-data-in-r-quick-guide/)

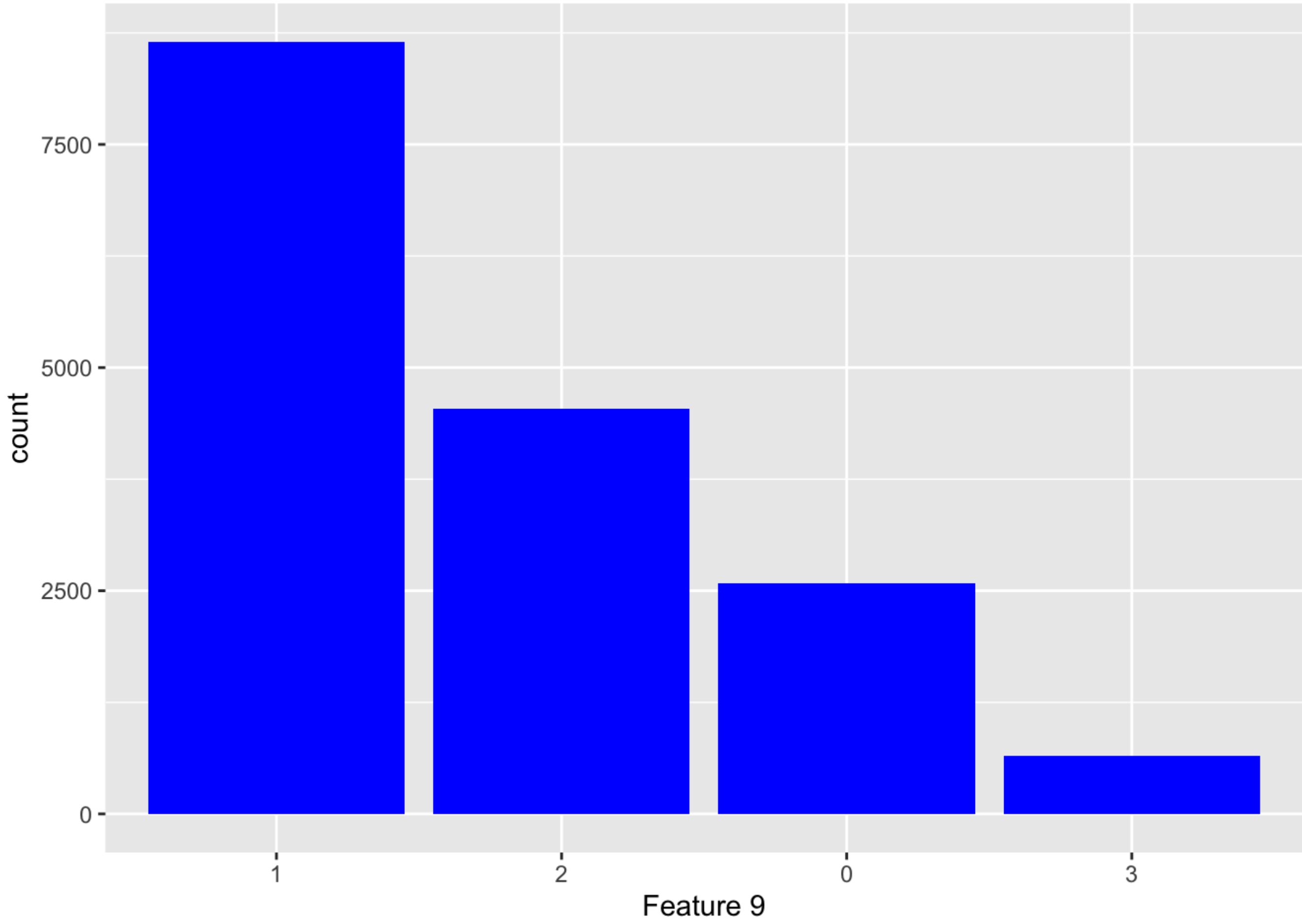
```
ggplot(train_data, aes(x=reorder(feature_7,feature_7, function(x)-length(x)))) +
geom_bar(fill='blue') +  labs(main = "Feature 7",x='Feature 7')
```



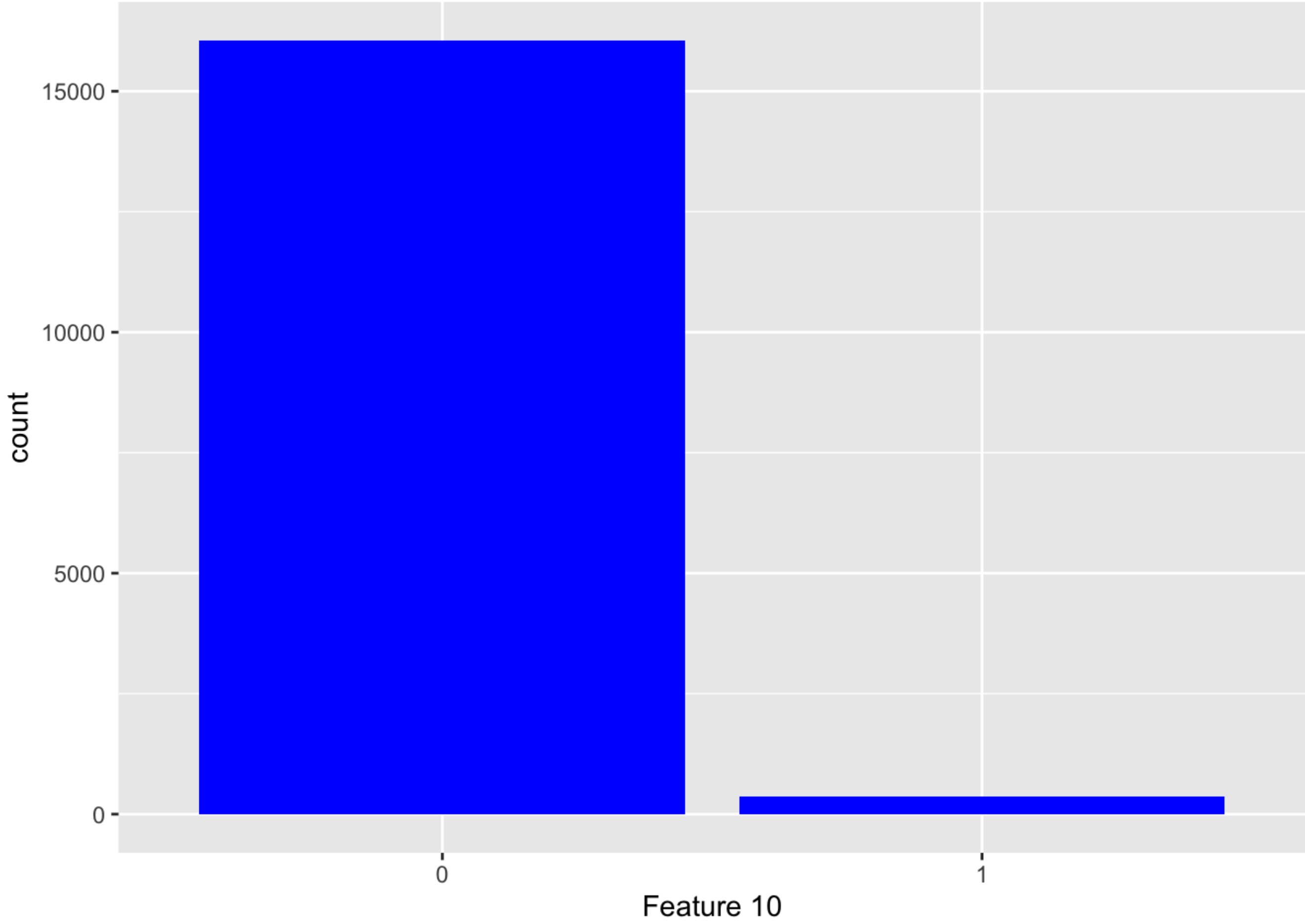

```
ggplot(train_data, aes(x=reorder(feature_8,feature_8, function(x)-length(x)))) +
geom_bar(fill='blue') +  labs(main = "Feature 8",x='Feature 8')
```



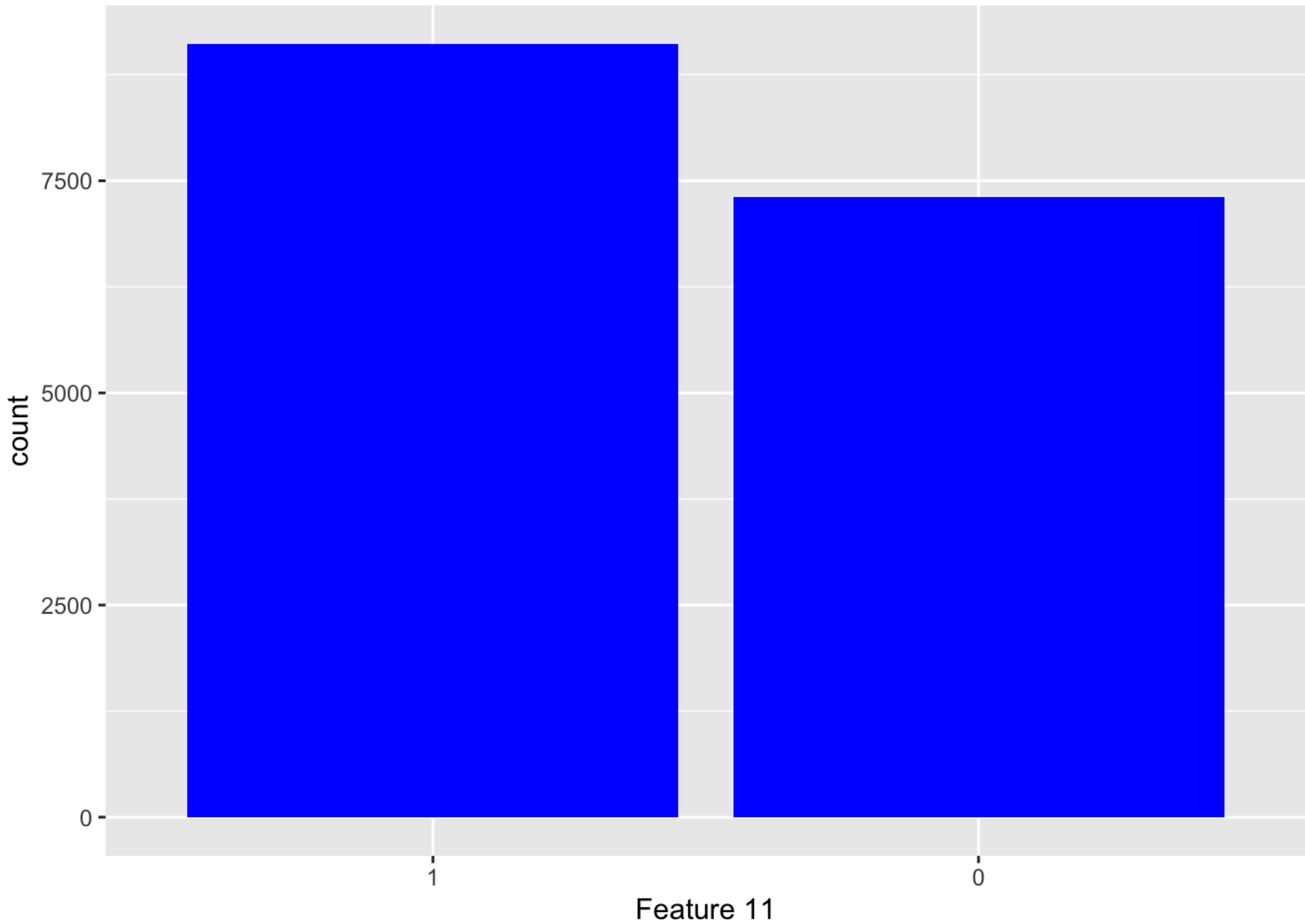
```
ggplot(train_data, aes(x=reorder(feature_9,feature_9, function(x)-length(x)))) +
geom_bar(fill='blue') +  labs(main = "Feature 9",x='Feature 9')
```



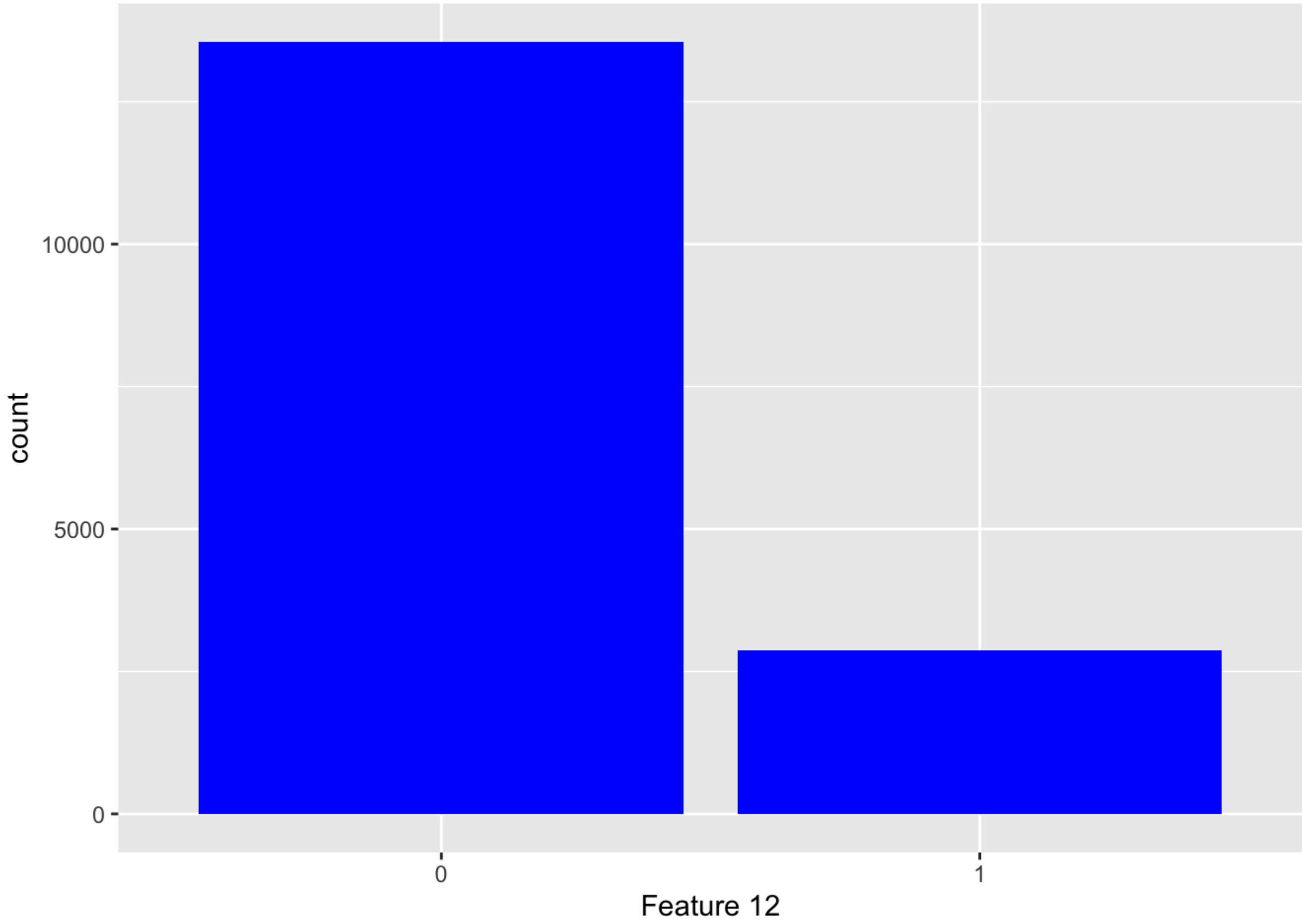

```
ggplot(train_data, aes(x=reorder(feature_10,feature_10, function(x)-length(x)))) +
geom_bar(fill='blue') +  labs(main = "Feature 10",x='Feature 10')
```



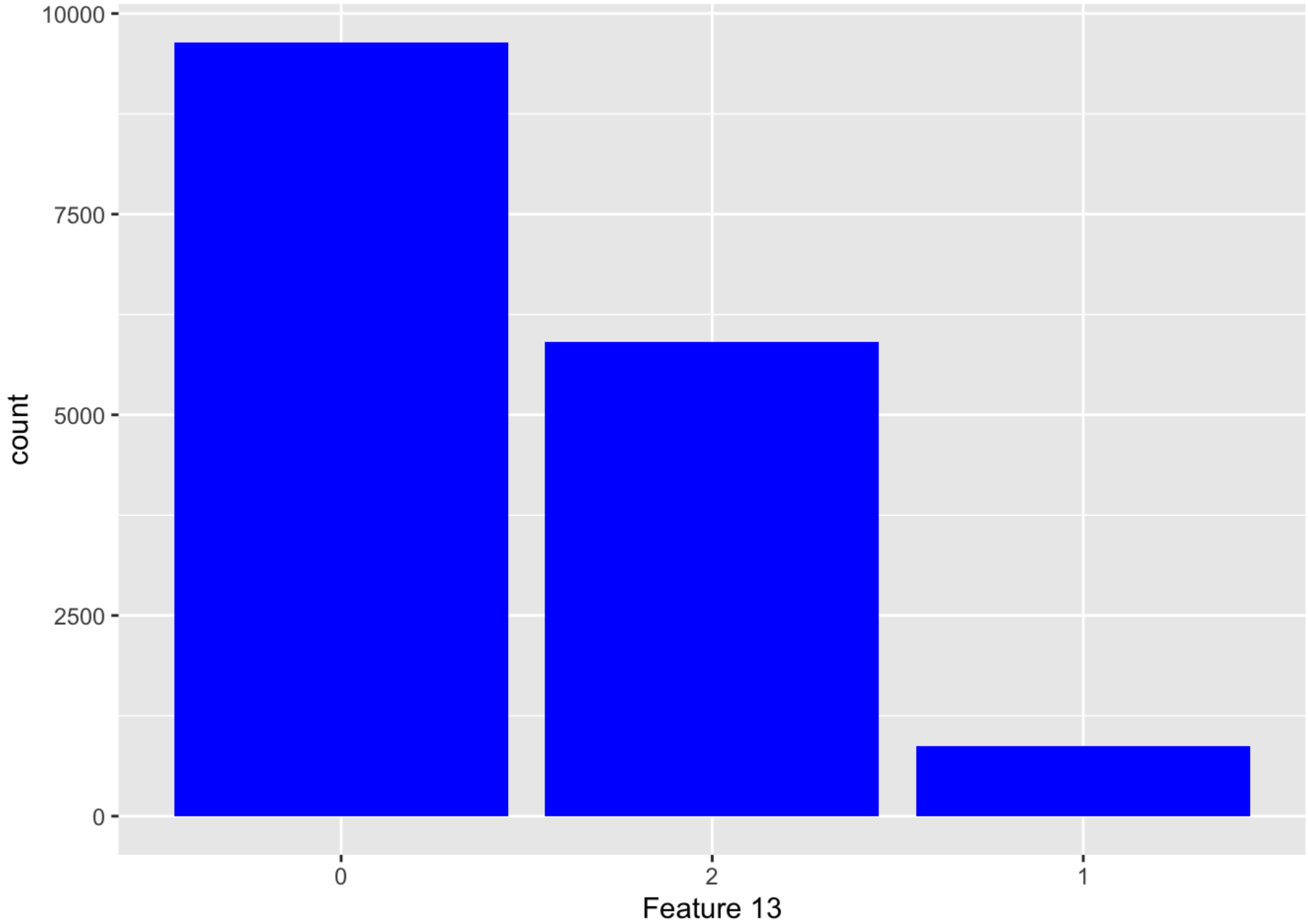

```
ggplot(train_data, aes(x=reorder(feature_11,feature_11, function(x)-length(x)))) +
geom_bar(fill='blue') +  labs(main = "Feature 11",x='Feature 11')
```



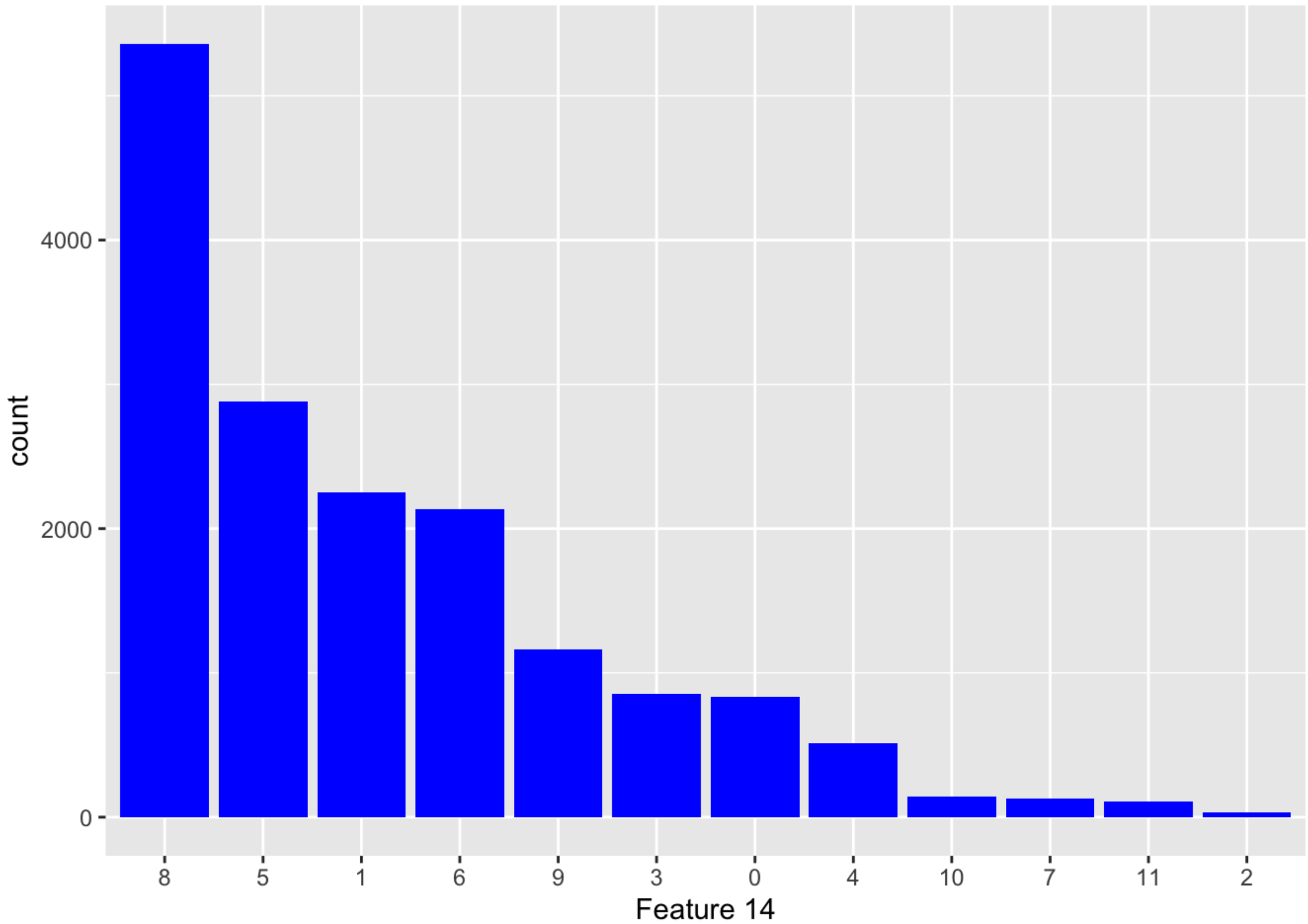
```
ggplot(train_data, aes(x=reorder(feature_12,feature_12, function(x)-length(x)))) +
geom_bar(fill='blue') +  labs(main = "Feature 12",x='Feature 12')
```




```
ggplot(train_data, aes(x=reorder(feature_13,feature_13, function(x)-length(x)))) +
geom_bar(fill='blue') +  labs(main = "Feature 13",x='Feature 13')
```



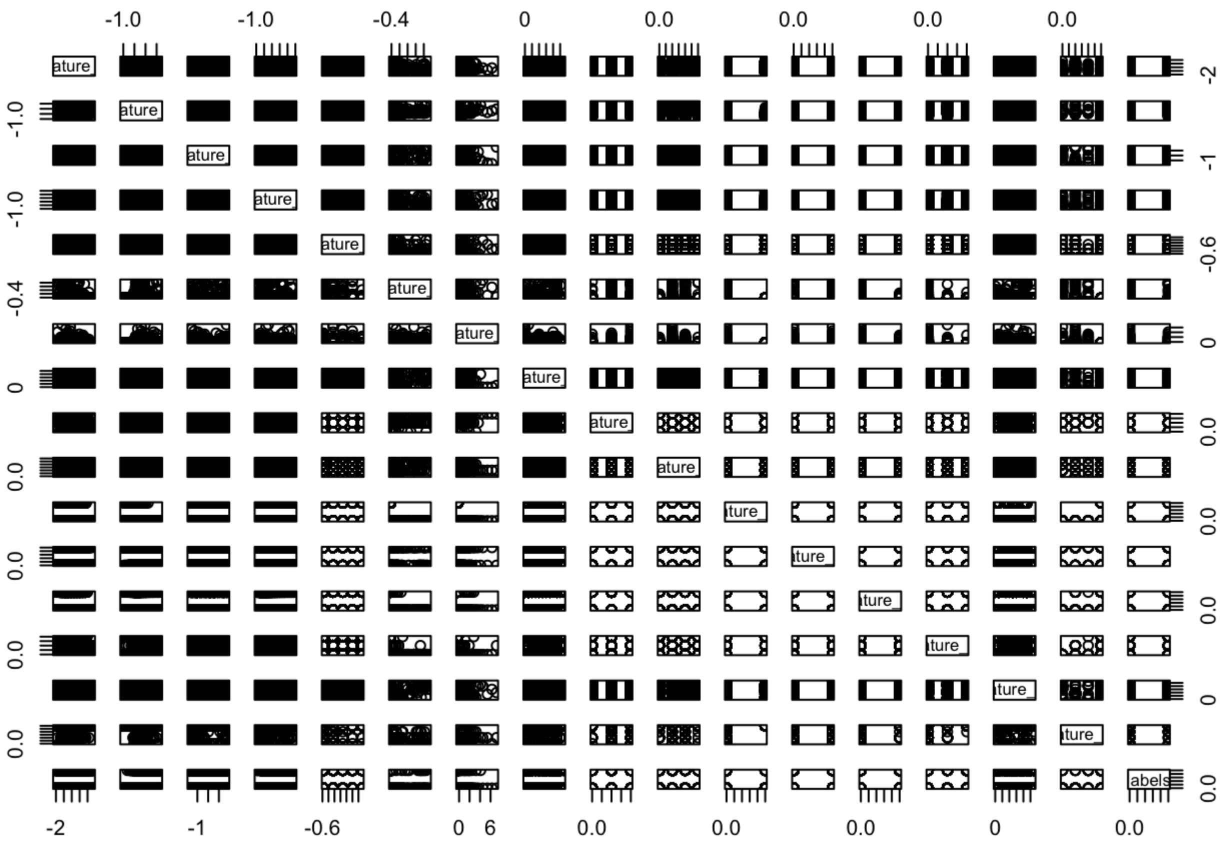

```
ggplot(train_data, aes(x=reorder(feature_14,feature_14, function(x)-length(x)))) +
geom_bar(fill='blue') +  labs(main = "Feature 14",x='Feature 14')
```



Bivariate Analysis

Moving on to bivariate analysis to see if we can find any interesting correlations.

```
#SLOWS DOWN PROCESSING, INCLUDE IN FINAL REPORT
pairs(train_data)
```



No obvious patterns emerging from the above)even when you zoom in).

Appling a broad correlation analysis

```
cor(train_data)
```

```
##              feature_0   feature_1   feature_2   feature_3    feature_4
## feature_0   1.000000000  0.084931596 -0.007528859 -0.049984748  0.0399071174
## feature_1   0.084931596  1.000000000  0.001125249  0.010688353 -0.0197152779
## feature_2  -0.007528859  0.001125249  1.000000000 -0.029311841  0.1009274660
## feature_3  -0.049984748  0.010688353 -0.029311841  1.000000000 -0.0485090900
## feature_4   0.039907117 -0.019715278  0.100927466 -0.048509090  1.0000000000
## feature_5  -0.017788183  0.013810204 -0.003287773  0.009126183 -0.0091341633
## feature_6  -0.007311845  0.013096468 -0.008903406  0.005893111  0.0009348197
## feature_7  -0.025093661  0.014023791  0.013176762 -0.015183758  0.0071391180
## feature_8  -0.413596799  0.015220733 -0.014869038  0.018154353 -0.0353330489
## feature_9  -0.103549147  0.058279845  0.019609135  0.014372184 -0.0086312005
## feature_10 -0.015120179 -0.137745094  0.004141541  0.009121282  0.0160981467
## feature_11 -0.150788450 -0.059610858 -0.001635728  0.023514761 -0.0361207780
## feature_12  0.005619447 -0.093459363 -0.001456557  0.003358218 -0.0168261491
## feature_13  0.035165384 -0.024221863 -0.039573416 -0.027947655 -0.0361244923
## feature_14 -0.059115447 -0.008212084  0.025489744  0.010015950 -0.1338861986
## feature_15  0.009647811 -0.007351650  0.009756738  0.002075373  0.0147531169
## labels     -0.036084789  0.070758809 -0.032384178  0.234298094 -0.0668166160
##              feature_5    feature_6    feature_7     feature_8   feature_9
## feature_0  -0.017788183 -0.0073118451 -0.025093661 -0.4135967987 -0.10354915
## feature_1   0.013810204  0.0130964680  0.014023791  0.0152207328  0.05827984
## feature_2  -0.003287773 -0.0089034060  0.013176762 -0.0148690383  0.01960914
## feature_3   0.009126183  0.0058931108 -0.015183758  0.0181543525  0.01437218
## feature_4  -0.009134163  0.0009348197  0.007139118 -0.0353330489 -0.00863120
## feature_5   1.000000000  0.5899236942  0.013843086  0.0218495394  0.03580781
## feature_6   0.589923694  1.0000000000 -0.004121416  0.0229711745  0.01318580
## feature_7   0.013843086 -0.0041214160  1.000000000  0.0596179807  0.17098678
## feature_8   0.021849539  0.0229711745  0.059617981  1.0000000000  0.11195142
## feature_9   0.035807808  0.0131857993  0.170986779  0.1119514153  1.00000000
## feature_10 -0.009575596 -0.0094255047 -0.008268869 -0.0025711443 -0.01108232
## feature_11 -0.009188161 -0.0087877441 -0.129172874 -0.0215212301 -0.08718968
## feature_12 -0.023890071 -0.0093562103 -0.031558744 -0.0550768680 -0.04572986
## feature_13 -0.047258340 -0.0411743513 -0.090760710 -0.0383094697 -0.12046744
## feature_14  0.005837701 -0.0228243575 -0.111923541 -0.0009942939 -0.07268313
## feature_15 -0.776933976 -0.6875337837 -0.010354468 -0.0205027673 -0.02378696
## labels      0.042777797  0.0313804044  0.032252886  0.0516412709  0.07613300
##             feature_10   feature_11   feature_12  feature_13    feature_14
## feature_0  -0.015120179 -0.150788450  0.005619447  0.03516538 -0.0591154465
## feature_1  -0.137745094 -0.059610858 -0.093459363 -0.02422186 -0.0082120840
## feature_2   0.004141541 -0.001635728 -0.001456557 -0.03957342  0.0254897440
## feature_3   0.009121282  0.023514761  0.003358218 -0.02794765  0.0100159501
## feature_4   0.016098147 -0.036120778 -0.016826149 -0.03612449 -0.1338861986
## feature_5  -0.009575596 -0.009188161 -0.023890071 -0.04725834  0.0058377011
## feature_6  -0.009425505 -0.008787744 -0.009356210 -0.04117435 -0.0228243575
## feature_7  -0.008268869 -0.129172874 -0.031558744 -0.09076071 -0.1119235412
## feature_8  -0.002571144 -0.021521230 -0.055076868 -0.03830947 -0.0009942939
## feature_9  -0.011082322 -0.087189675 -0.045729856 -0.12046744 -0.0726831309
## feature_10  1.000000000 -0.008850956  0.075757226  0.01625859  0.0208275232
## feature_11 -0.008850956  1.000000000  0.033113129  0.24561604  0.3408347146
## feature_12  0.075757226  0.033113129  1.000000000 -0.02049494  0.0355212846
## feature_13  0.016258589  0.245616043 -0.020494937  1.00000000  0.4532474097
## feature_14  0.020827523  0.340834715  0.035521285  0.45324741  1.0000000000
## feature_15  0.012279018  0.007688449  0.021258384  0.05850411  0.0284592742
## labels     -0.010990181 -0.130276294 -0.059789463 -0.13732894 -0.0518606215
##             feature_15      labels
```

```
## feature_0    0.009647811 -0.03608479
## feature_1   -0.007351650  0.07075881
## feature_2    0.009756738 -0.03238418
## feature_3    0.002075373  0.23429809
## feature_4    0.014753117 -0.06681662
## feature_5   -0.776933976  0.04277780
## feature_6   -0.687533784  0.03138040
## feature_7   -0.010354468  0.03225289
## feature_8   -0.020502767  0.05164127
## feature_9   -0.023786961  0.07613300
## feature_10   0.012279018 -0.01099018
## feature_11   0.007688449 -0.13027629
## feature_12   0.021258384 -0.05978946
## feature_13   0.058504110 -0.13732894
## feature_14   0.028459274 -0.05186062
## feature_15   1.000000000 -0.01991393
## labels      -0.019913927  1.00000000
```

Quite hard to read Applying correlation analysis - use cut off of 0.60, otherwise not really useful

```
train_data_cor <- cor(train_data)
high_cor <- findCorrelation(train_data_cor, cutoff = 0.6)
high_cor
```

```
## [1] 16
```

Feature 5 and 15, and 6 and 15 are the only pairs with a resonable correlation after inspcting column 16 (feature 15). -.78 correlation with feature 5 and -0.69 correlation with feature 6.

Further apply covariance to see if applicable. Can see some data is on a different scale so may need to normalise

```
#cov(train_data)
```

#Noramlize the data # Reference: https://www.edureka.co/blog/knn-algorithm-in-r/ (https://www.edureka.co/blog/knn-algorithm-in-r/)

```
normalize <- function(x) {
  return ((x-min(x)) / (max(x) - min(x)))
}
```

```
train_data_norm <- as.data.frame(lapply(train_data[,1:17], normalize))
#train_data_norm

test_data_norm <- as.data.frame(lapply(test_data[,1:17], normalize))
#test_data_norm
```

```
#cov(train_data)
```

Feature 7 and 9, and features 13 and 14 are of interest.

```
table(train_data$feature_7, train_data$feature_9)
```

```
##
##         0    1    2    3
##   0    71 1574  211   59
##   1  1457 2071   50  163
##   2    70  213  246   27
##   3   260  144   58   17
##   4   114  399 2661   92
##   5   226  296   99   30
##   6    45  223  270   11
##   7   141 1342   53   55
##   8    10  169   87   52
##   9    58 1916  694   98
##   10  105  278   97    8
##   11   22   26   16   40
```

```
table(train_data$feature_13, train_data$feature_14)
```

```
##
##        0    1    2    3    4    5    6    7    8    9   10   11
##   0  780 2157   28  761  456 2431  235  121 1467 1036   93   77
##   1   54   66    5   85   52  304   24    8  118  111   30   13
##   2    3   26    2    7    6  149 1878    2 3776   19   23   21
```

2. Describe your choice of model based off of eda.

I want to approach this problem form 2 different perspectives intially. I will first develop a parametric model. However as can be seen in the analysis I swicth between a generalised logistic model and a Linear Discremenant Analysis model after observing some of the eraly results, to improve performance.
There were a couple of reasonable correlations observed in the EDA, so I do want to see how a parametric model works. I also want a apply a nonparametric approach so wil use knn. Particularly because there is quite a mix of discrete and continous variables, so I feel that may distort any underlying equation assumptions (implicit in a parametric approach). Also there are so many variables being considred, so a non parametric approach may be effective. Considiering it is a relatively small data set (not millions of rows of data) I am comfortable to computational requirements of KNN (which is greater than LDA and GLM) will not be prohibitive to employing that type of model.

After I have run both of these I will reassess and see how the different models perform.

##3. Develop 2 types of models (e.g. logistic regression and KNN)

Logistic regression model

```
#Create factors for labels in both test and training set
train_data$labels <- factor(train_data$labels)
table(train_data$labels)
```

```
##
##     0     1
## 15474   950
```

```
test_data$labels <- factor(test_data$labels)
table(test_data$labels)
```

```
##
##    0    1
## 5997  785
```

```
train_data_norm$labels <- factor(train_data_norm$labels)
test_data_norm$labels <- factor(test_data_norm$labels)
```

#Logistic regression model

```
logit <- train(labels ~., data=train_data, method = 'glm', family=binomial(link='logi
t'), preProcess=c('scale', 'center'))
```

#Summary of logit model

```
summary(logit)
```

```
##
## Call:
## NULL
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.6859  -0.3298  -0.2026  -0.1198   3.4032
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -3.629250   0.057735 -62.860  < 2e-16 ***
## feature_0   -0.110641   0.040121  -2.758  0.00582 **
## feature_1    0.199244   0.033466   5.954 2.62e-09 ***
## feature_2   -0.144878   0.036079  -4.016 5.93e-05 ***
## feature_3    0.827869   0.031088  26.630  < 2e-16 ***
## feature_4   -0.341498   0.043366  -7.875 3.41e-15 ***
## feature_5    0.102141   0.040809   2.503  0.01232 *
## feature_6    0.034593   0.032622   1.060  0.28894
## feature_7    0.018370   0.037665   0.488  0.62575
## feature_8    0.055204   0.040264   1.371  0.17036
## feature_9    0.175910   0.037478   4.694 2.68e-06 ***
## feature_10   0.005853   0.041705   0.140  0.88839
## feature_11  -0.533776   0.040937 -13.039  < 2e-16 ***
## feature_12  -0.342975   0.048194  -7.117 1.11e-12 ***
## feature_13  -0.688640   0.053250 -12.932  < 2e-16 ***
## feature_14   0.090793   0.036349   2.498  0.01250 *
## feature_15   0.110395   0.058207   1.897  0.05788 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 7259.0  on 16423  degrees of freedom
## Residual deviance: 5684.3  on 16407  degrees of freedom
## AIC: 5718.3
##
## Number of Fisher Scoring iterations: 7
```

Null deviance is high, showing this model is improving on the null model (good thing) Also the fisher iterations at 7 are showing us that the solution is able be solved.

Confusion Matrix

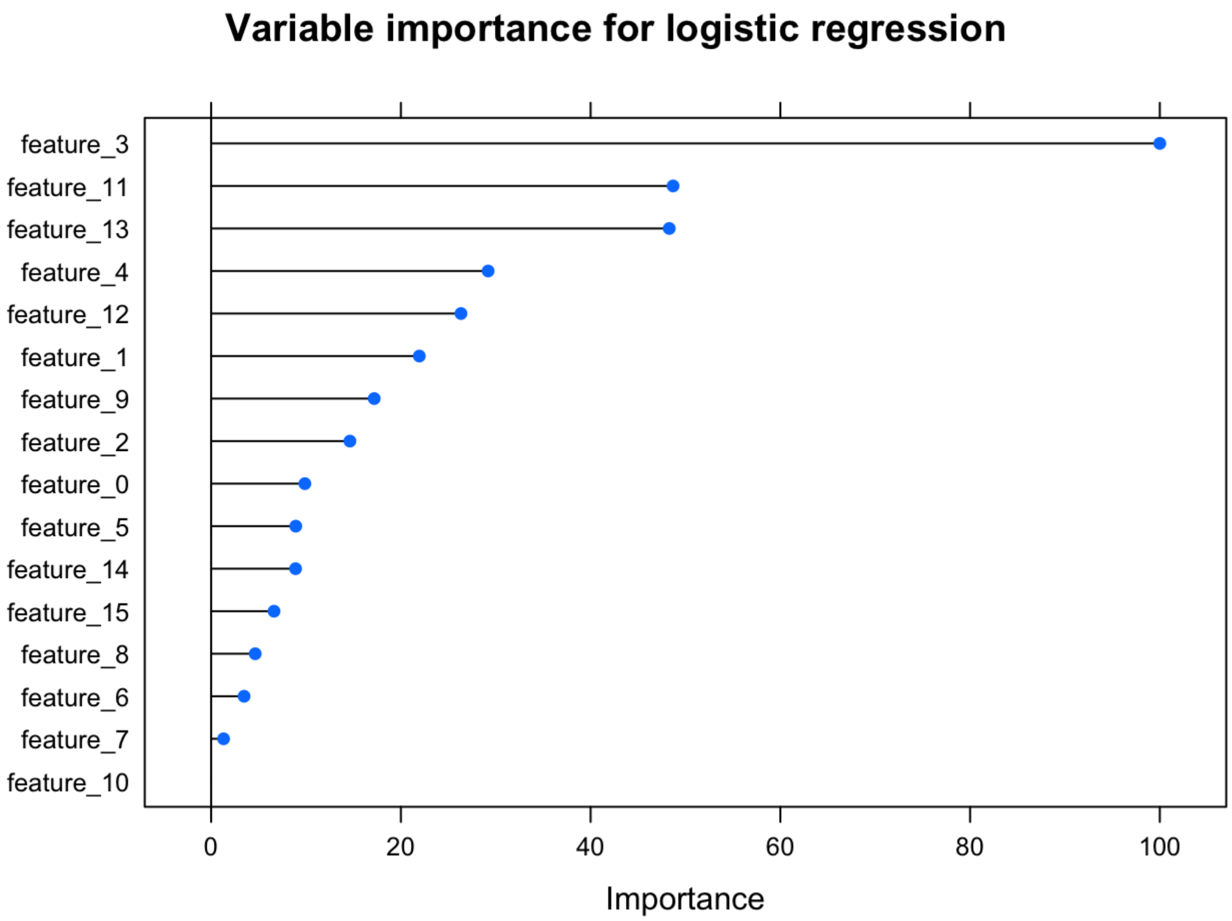
```
confusionMatrix(predict(logit, test_data[,-17]), test_data$labels)
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction    0    1
##          0 4918  345
##          1 1079  440
##                                           
##                Accuracy : 0.79            
##                  95% CI : (0.7801, 0.7997)
##     No Information Rate : 0.8843          
##     P-Value [Acc > NIR] : 1               
##                                           
##                   Kappa : 0.2706          
##                                           
##  Mcnemar's Test P-Value : <2e-16          
##                                           
##             Sensitivity : 0.8201          
##             Specificity : 0.5605          
##          Pos Pred Value : 0.9344          
##          Neg Pred Value : 0.2897          
##              Prevalence : 0.8843          
##          Detection Rate : 0.7252          
##    Detection Prevalence : 0.7760          
##       Balanced Accuracy : 0.6903          
##                                           
##        'Positive' Class : 0               
## 
```

Assessing the above model it is “ok”. 0.79 accuracy so can somewhat predict however a high error rate. False positive results are conceningly high on this (at 43.95%)and I would like to see that reduce significantly before accepting the model. The false negative rate is better (at 17.99%) but I would like to see that reduce further.

Looking at variable importance

```
plot(varImp(logit, scale = TRUE), main = "Variable importance for logistic regressio
n")
```



Based off of the above variable importance graph, I'll refine the model to include the more important features.

```
train_data_refined <- train_data[, c(4,12,14,5,13,17)]
head(train_data_refined)
```

```
##    feature_3 feature_11 feature_13   feature_4 feature_12 labels
## 2 -0.1598411          0          0 -0.56935064          0      0
## 3  0.7798736          0          2 -0.56935064          0      0
## 4 -0.3772958          0          0  0.39902023          0      0
## 5 -0.4161270          1          0 -0.56935064          0      0
## 8 -0.9403480          1          2  0.07622994          1      0
## 9 -0.3384646          1          0 -0.56935064          1      0
```

```
test_data_refined <- test_data[,c(4,12,14,5,13, 17)]
head(test_data_refined)
```

```
##    feature_3 feature_11 feature_13   feature_4 feature_12 labels
## 1 -0.9053999          1          2  0.07622994          0      0
## 2 -0.1054775          1          2  2.98134255          0      0
## 3  0.3993280          1          2 -0.56935064          1      0
## 4 -0.8976337          1          0  6.20924545          0      0
## 5  0.0537304          1          0 -0.56935064          0      0
## 6  0.5119384          1          2 -0.24656035          1      0
```

#Logistic regression model

```
logit <- train(labels ~., data=train_data_refined, method = 'glm', family=binomial(li
nk='logit'), preProcess=c('scale', 'center'))
```

#Summary of logit model

```
summary(logit)
```

```
##
## Call:
## NULL
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.4849  -0.3409  -0.2119  -0.1250   3.3191
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -3.57600    0.05652 -63.270  < 2e-16 ***
## feature_3    0.83145    0.03067  27.113  < 2e-16 ***
## feature_11  -0.52873    0.03897 -13.566  < 2e-16 ***
## feature_13  -0.69035    0.05116 -13.493  < 2e-16 ***
## feature_4   -0.38871    0.04253  -9.141  < 2e-16 ***
## feature_12  -0.37249    0.04741  -7.856 3.97e-15 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 7259.0  on 16423  degrees of freedom
## Residual deviance: 5803.3  on 16418  degrees of freedom
## AIC: 5815.3
##
## Number of Fisher Scoring iterations: 7
```

As you can see above, according to the z-scores, all of the predictor variables included were significant in influencing the response variable.

Confusion Matrix

```
confusionMatrix(predict(logit, test_data_refined[,-17]), test_data_refined$labels)
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction    0    1
##          0 5854  576
##          1  143  209
##                                           
##                Accuracy : 0.894           
##                  95% CI : (0.8864, 0.9012)
##     No Information Rate : 0.8843          
##     P-Value [Acc > NIR] : 0.005997        
##                                           
##                   Kappa : 0.3188          
##                                           
##  Mcnemar's Test P-Value : < 2.2e-16       
##                                           
##             Sensitivity : 0.9762          
##             Specificity : 0.2662          
##          Pos Pred Value : 0.9104          
##          Neg Pred Value : 0.5937          
##              Prevalence : 0.8843          
##          Detection Rate : 0.8632          
##    Detection Prevalence : 0.9481          
##       Balanced Accuracy : 0.6212          
##                                           
##        'Positive' Class : 0               
## 
```

Clearly a significant improvement as I remove some of the variables that have little statistical significant. (seen in z scores on the summary)

Now I will work to optimise further referring to z scores.

```
test_sample <- c(4,12,14,5,13,2, 17)
train_data_refined <- train_data[, test_sample]
head(train_data_refined)
```

```
##    feature_3 feature_11 feature_13   feature_4 feature_12  feature_1 labels
## 2 -0.1598411          0          0 -0.56935064          0 -0.2125875      0
## 3  0.7798736          0          2 -0.56935064          0  0.5812426      0
## 4 -0.3772958          0          0  0.39902023          0 -0.2217837      0
## 5 -0.4161270          1          0 -0.56935064          0 -0.5922597      0
## 8 -0.9403480          1          2  0.07622994          1  0.0222443      0
## 9 -0.3384646          1          0 -0.56935064          1 -0.3502023      0
```

```
test_data_refined <- test_data[,test_sample]
head(test_data_refined)
```

```
##    feature_3 feature_11 feature_13   feature_4 feature_12  feature_1 labels
## 1 -0.9053999          1          2  0.07622994          0 -0.2224406      0
## 2 -0.1054775          1          2  2.98134255          0 -0.3564425      0
## 3  0.3993280          1          2 -0.56935064          1 -0.2648089      0
## 4 -0.8976337          1          0  6.20924545          0 -0.4024236      0
## 5  0.0537304          1          0 -0.56935064          0 -0.2677648      0
## 6  0.5119384          1          2 -0.24656035          1 -0.4221298      0
```

```
logit <- train(labels ~., data=train_data_refined, method = 'glm', family=binomial(li
nk='logit'), preProcess=c('scale', 'center'))

summary(logit)
```

```
##
## Call:
## NULL
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.6246  -0.3358  -0.2086  -0.1236   3.3369
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -3.59354    0.05692 -63.132  < 2e-16 ***
## feature_3    0.83036    0.03079  26.971  < 2e-16 ***
## feature_11  -0.51670    0.03909 -13.220  < 2e-16 ***
## feature_13  -0.68822    0.05133 -13.409  < 2e-16 ***
## feature_4   -0.38131    0.04261  -8.949  < 2e-16 ***
## feature_12  -0.34878    0.04760  -7.327 2.35e-13 ***
## feature_1    0.20706    0.03257   6.357 2.05e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 7259.0  on 16423  degrees of freedom
## Residual deviance: 5764.8  on 16417  degrees of freedom
## AIC: 5778.8
##
## Number of Fisher Scoring iterations: 7
```

```
confusionMatrix(predict(logit, test_data_refined[,-17]), test_data_refined$labels)
```

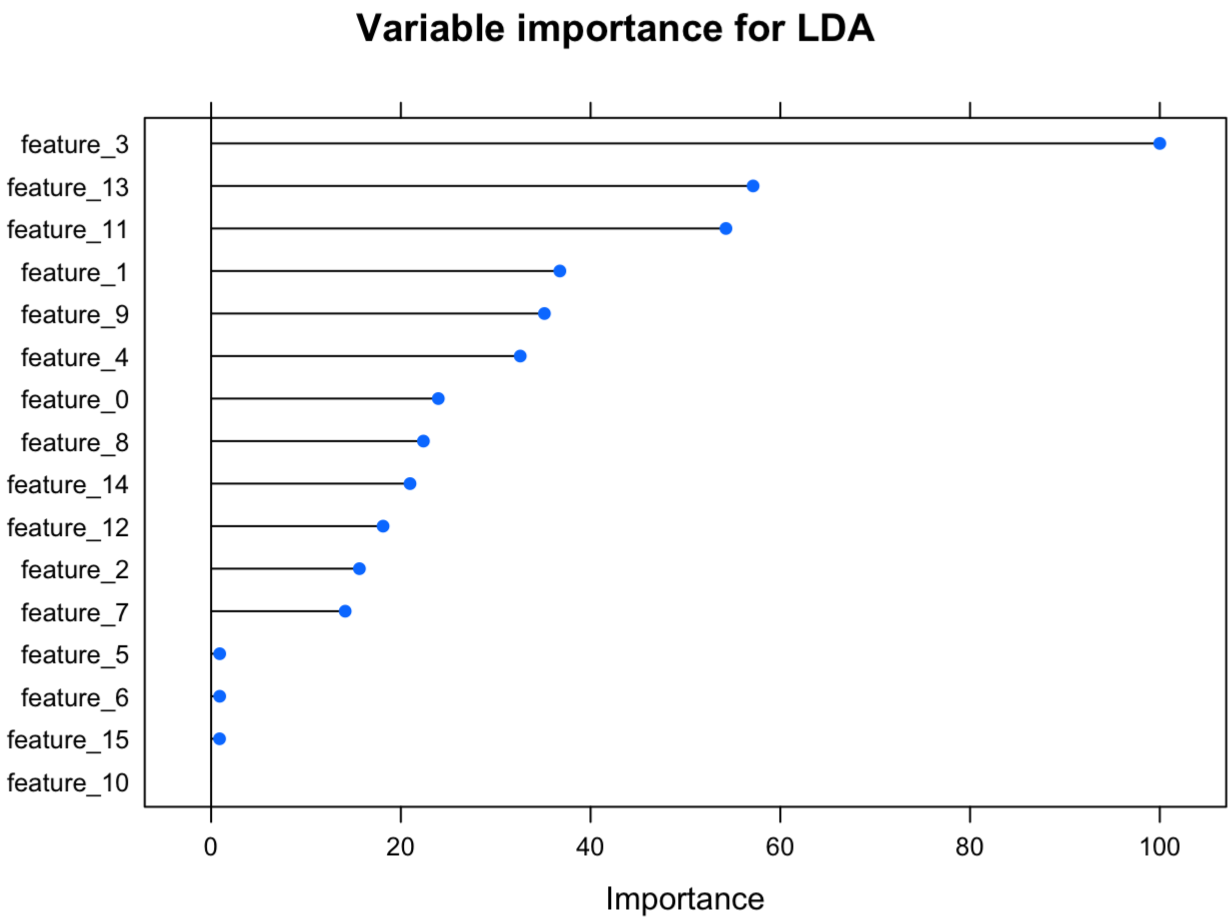
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 5769  539
##          1  228  246
##
##                Accuracy : 0.8869
##                  95% CI : (0.8791, 0.8944)
##     No Information Rate : 0.8843
##     P-Value [Acc > NIR] : 0.2541
##
##                   Kappa : 0.3326
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.9620
##             Specificity : 0.3134
##          Pos Pred Value : 0.9146
##          Neg Pred Value : 0.5190
##              Prevalence : 0.8843
##          Detection Rate : 0.8506
##    Detection Prevalence : 0.9301
##       Balanced Accuracy : 0.6377
##
##        'Positive' Class : 0
##
```

Second model, LDA - lower accuracy, but gives better specificity

```
train_data_refined <- train_data
test_data_refined <- test_data
LDA_original <- train(labels ~., data=train_data_refined, method = "lda", preProcess=
c('scale','center'))

confusionMatrix(test_data_refined$labels, predict(LDA_original, test_data_refined[-17
]))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 4804 1193
##          1  292  493
##
##                Accuracy : 0.781
##                  95% CI : (0.771, 0.7908)
##     No Information Rate : 0.7514
##     P-Value [Acc > NIR] : 5.637e-09
##
##                   Kappa : 0.2863
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9427
##             Specificity : 0.2924
##          Pos Pred Value : 0.8011
##          Neg Pred Value : 0.6280
##              Prevalence : 0.7514
##          Detection Rate : 0.7083
##    Detection Prevalence : 0.8843
##       Balanced Accuracy : 0.6176
##
##        'Positive' Class : 0
##
```

```
plot(varImp(LDA_original, scale = TRUE), main = "Variable importance for LDA")
```



Refine a LDA model

```
test_sample <- c(4,14,12,2,10,5,1,9,15,17)
train_data_refined <- train_data[, test_sample]
test_data_refined <- test_data[, test_sample]
LDA_original <- train(labels ~., data=train_data_refined, method = "lda", preProcess=
c('scale','center'))
confusionMatrix(test_data_refined$labels, predict(LDA_original, test_data_refined[-10
]))
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction    0    1
##          0 5652  345
##          1  489  296
## 
##                Accuracy : 0.877
##                  95% CI : (0.869, 0.8848)
##     No Information Rate : 0.9055
##     P-Value [Acc > NIR] : 1
## 
##                   Kappa : 0.3472
## 
##  Mcnemar's Test P-Value : 7.357e-07
## 
##             Sensitivity : 0.9204
##             Specificity : 0.4618
##          Pos Pred Value : 0.9425
##          Neg Pred Value : 0.3771
##              Prevalence : 0.9055
##          Detection Rate : 0.8334
##    Detection Prevalence : 0.8843
##       Balanced Accuracy : 0.6911
## 
##        'Positive' Class : 0
## 
```

Second machine learning model, k nearest Neighbour

```
#sqrt the number of samples
(sqrt(20124))
```

```
## [1] 141.8591
```

```
head(test_data)
```

```
##    feature_0  feature_1  feature_2  feature_3   feature_4  feature_5  feature_6
## 1 -1.0299064 -0.2224406  0.5038918 -0.9053999  0.07622994 -0.4114531 -0.2519404
## 2 -0.8415585 -0.3564425 -1.5387921 -0.1054775  2.98134255 -0.4114531 -0.2519404
## 3 -1.5007763 -0.2648089 -1.4186342  0.3993280 -0.56935064 -0.4114531 -0.2519404
## 4 -0.2765146 -0.4024236  1.8256285 -0.8976337  6.20924545 -0.4114531 -0.2519404
## 5 -0.9357324 -0.2677648  1.5853127  0.0537304 -0.56935064 -0.4114531 -0.2519404
## 6 -1.5949503 -0.4221298 -1.1783185  0.5119384 -0.24656035 -0.4114531 -0.2519404
##   feature_7 feature_8 feature_9 feature_10 feature_11 feature_12 feature_13
## 1         0         1         1          0          1          0          2
## 2         0         1         1          0          1          0          2
## 3         7         2         1          0          1          1          2
## 4         0         1         1          0          1          0          0
## 5         0         2         1          0          1          0          0
## 6         7         2         1          0          1          1          2
##   feature_14 feature_15 labels
## 1          6          3      0
## 2          6          3      0
## 3          6          3      0
## 4          5          3      0
## 5          4          3      0
## 6          8          3      0
```

```
pred.knn.k5 = knn(train_data[,-17], test_data[,-17], train_data$labels, k = 5)
table(pred.knn.k5, test_data$labels)
```

```
##
## pred.knn.k5    0    1
##           0 5884  686
##           1  113   99
```

```
confusionMatrix(table(pred.knn.k5, test_data$labels))
```

```
## Confusion Matrix and Statistics
## 
##          
## pred.knn.k5    0    1
##           0 5884  686
##           1  113   99
##                                           
##                Accuracy : 0.8822          
##                  95% CI : (0.8743, 0.8898)
##     No Information Rate : 0.8843          
##     P-Value [Acc > NIR] : 0.7101          
##                                           
##                   Kappa : 0.1571          
##                                           
##  Mcnemar's Test P-Value : <2e-16          
##                                           
##             Sensitivity : 0.9812          
##             Specificity : 0.1261          
##          Pos Pred Value : 0.8956          
##          Neg Pred Value : 0.4670          
##              Prevalence : 0.8843          
##          Detection Rate : 0.8676          
##    Detection Prevalence : 0.9687          
##       Balanced Accuracy : 0.5536          
##                                           
##        'Positive' Class : 0               
## 
```

Now lets apply knn but with the most relevant factors as determined before

```
#Subsets to remove irrelevant variables
reduced_variables <- c(2,3,4,5,6,11,14, 15, 16)


#Model
pred.knn.kx = knn(train_data[,reduced_variables], test_data[,reduced_variables], trai
n_data$labels, k = 7)
table(pred.knn.kx, test_data$labels)
```

```
##            
## pred.knn.kx    0    1
##           0 5901  618
##           1   96  167
```

```
confusionMatrix(table(pred.knn.kx, test_data$labels))
```

```
## Confusion Matrix and Statistics
##
##
## pred.knn.kx    0    1
##           0 5901  618
##           1   96  167
##
##                Accuracy : 0.8947
##                  95% CI : (0.8872, 0.9019)
##     No Information Rate : 0.8843
##     P-Value [Acc > NIR] : 0.003393
##
##                   Kappa : 0.2767
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9840
##             Specificity : 0.2127
##          Pos Pred Value : 0.9052
##          Neg Pred Value : 0.6350
##              Prevalence : 0.8843
##          Detection Rate : 0.8701
##    Detection Prevalence : 0.9612
##       Balanced Accuracy : 0.5984
##
##        'Positive' Class : 0
##
```

4. Evaluate models using selected performance measures (at least 2)

Assessing the knn model. I am happy with the accuracy of this mode at 89.46%, using a subset of factors that were shown to have influence in the EDA and logistic modeling. My concern with this model is the specificity at 21.27%. This means this model is doing a good job at predicting accuratley when a customer doesn't churn (98% of the time), however it incorrectly identifies when a customer will churn, often these will actually be customers that stay. Commercially, I think this would pose a probelm for the marketing team, and I do not think it is in the best interest of the marketing team/the client to have a model that has a high degree of accuracy, but a very poor (less than 50 %) True Positive Rate.

After applying both models I would be in favor of using lda as it does a better job at capturing specificity. This is really important as the data is heavily skewed toward customers that don't churn (i.e. most customers do not leave) It still performs relatively well with my final model over 89% (in the final section of this assignment). But imprtantly it enabled me to get a higher True Positive Rate.

5. Use selected model ,identify and discuss the key factors (variable importance) of the selected model

I would suggest using a lda regression model for churn prediction purposes.

Although it was slightly lower in accuracy compared to other models I developed it had a better read on specificity. This is important as churn customers (label = 1), are the subset of interest. If they are incorrectly being classified the majority of the time (as with the other models) it would be detrimental to the marketing team in a real world application.

The initial EDA did not reveal clear correlations (beyond a couple of factors) or coverance which drove my models. As we applied the statistical analysis.

Feature_3 had the most profound effect on the models accuracy and would be the most important factor to look at for marketing. Features 11,13, 1 all seemed to have an impact on the models accuracy and would be worth investigating further.I also found that although the initial EDA showed that there were significant outliers in the continuous variables. Applying the IQR range to remove outliers was actually detrimental to the overall accuracy of the model, with the exception of feature 4. This suggests to me that the extreme data in the features (except for feature 4) was actually quite important and not noise, based off of my analysis.

Below is the final model I would use:

```
#Load in training data and inspect
train_data <- read.csv("trainSet.csv")
test_data <- read.csv("testSet.csv")


#Remove outliers for Feature 4 only
Q_4 <- quantile(train_data$feature_4, probs=c(0.25, 0.75), na.rm = FALSE)
iqr_4 <- IQR(train_data$feature_4)
up_4 <- Q_4[2]+1.5*iqr_4
low_4 <- Q_4[1] - 1.5*iqr_4

train_data <- subset(train_data, train_data$feature_4 > low_4 & train_data$feature_4
 < up_4)

#Create factors for labels in both test and training set
train_data$labels <- factor(train_data$labels)
table(train_data$labels)
```

```
##
##     0     1
## 21487  3037
```

```
test_data$labels <- factor(test_data$labels)
table(test_data$labels)
```

```
##
##    0    1
## 5997  785
```

```
train_data_norm$labels <- factor(train_data_norm$labels)
test_data_norm$labels <- factor(test_data_norm$labels)



test_sample <- c(4,14,12,2,10,5,1,9,15,17)
train_data_refined <- train_data[, test_sample]
test_data_refined <- test_data[, test_sample]
LDA_original <- train(labels ~., data=train_data_refined, method = "lda", preProcess=
c('scale','center'))
confusionMatrix(test_data_refined$labels, predict(LDA_original, test_data_refined[-10
]))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 5840  157
##          1  580  205
##
##                Accuracy : 0.8913
##                  95% CI : (0.8837, 0.8986)
##     No Information Rate : 0.9466
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.3068
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.9097
##             Specificity : 0.5663
##          Pos Pred Value : 0.9738
##          Neg Pred Value : 0.2611
##              Prevalence : 0.9466
##          Detection Rate : 0.8611
##    Detection Prevalence : 0.8843
##       Balanced Accuracy : 0.7380
##
##        'Positive' Class : 0
##
```

You can see above this model has an accuracy of 89.13%. It has the strongest specificty of over 50% at 0.5663 and still strong sensitivity, over 90%.

6 Make suggestions/provide commercial insights to marketing based off of these findings. Assuming a non data science audience.

My hope for marketing is that they could apply domain knowledge to the identified factors to see what they could do to decrease the churn rate. Feature_3 is the most influential, and I would advise looking at this in the most detail. Other important features as mentioned above are features 11,13 1.

Refences: I used the below websites to help construct some of my plotting, removing outliers and knn algorithims. EDA - https://www.r-bloggers.com/2020/01/how-to-remove-outliers-in-r/ (https://www.r-bloggers.com/2020/01/how-to-remove-outliers-in-r/) https://www.r-bloggers.com/2021/08/how-to-plot-categorical-data-in-r-quick-guide/ (https://www.r-bloggers.com/2021/08/how-to-plot-categorical-data-in-r-quick-guide/) https://www.edureka.co/blog/knn-algorithm-in-r/ (https://www.edureka.co/blog/knn-algorithm-in-r/)