HW3 Data Challenge

First, load the data and view with "str()":

```r
test = read.csv("./loan_testx.csv", stringsAsFactors = T)
train = read.csv("./loan_train.csv", stringsAsFactors = T)
str(train)
```

```
## 'data.frame':	3000 obs. of  31 variables:
##  $ default            : int  0 1 1 1 1 1 1 1 0 0 ...
##  $ n_collect          : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ credit_ratio       : num  82.7 86.1 51.6 52.3 58.4 41.1 99 22.4 47 94.3 ...
##  $ interest           : num  14.5 17.6 16.2 20.9 11.4 ...
##  $ initial_list_status: Factor w/ 2 levels "a","b": 1 1 1 1 2 2 1 2 1 1 ...
##  $ recover            : num  0 0 2016 453 0 ...
##  $ coll_fee           : num  0 0 362.89 4.98 0 ...
##  $ out_prncp          : num  11815 8148 0 0 14514 ...
##  $ total_cc           : num  16347 1274 10772 9523 4733 ...
##  $ term               : Factor w/ 2 levels " 3 yrs"," 5 yrs": 1 1 2 2 1 2 1 2 1 1 ...
##  $ fees_rec           : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ total_acc          : int  18 19 23 51 26 34 31 35 18 23 ...
##  $ employment         : Factor w/ 11 levels "< 1","1","10+",..: 3 NA 1 6 3 9 NA 3 2 3 ...
##  $ amount             : int  23725 8925 27575 24000 18000 20000 2400 22000 15000 23900 ...
##  $ monthly_payment    : num  817 321 674 454 593 ...
##  $ funded             : int  23725 8925 27575 16800 18000 20000 2400 22000 15000 23900 ...
##  $ status             : Factor w/ 3 levels "checked","partial",..: 2 1 2 1 2 1 3 3 3 1 ...
##  $ v1                 : num  13.6 24.5 27.5 22.9 18.2 ...
##  $ int_rec            : num  4437 497 4506 5269 1247 ...
##  $ reason             : Factor w/ 13 levels "boat","business",..: 4 4 4 4 4 4 11 4 3 3 ...
##  $ last_payment       : num  816.5 320.7 673.5 26.4 593.1 ...
##  $ pymnt_rec          : num  16347 1231 10772 9523 4733 ...
##  $ quality            : Factor w/ 7 levels "q1","q2","q3",..: 3 4 3 7 2 3 3 2 2 3 ...
##  $ out_prncp_inv      : num  11815 7874 0 0 14514 ...
##  $ violations         : int  1 0 0 0 0 0 0 0 0 0 ...
##  $ del                : int  0 4 0 0 0 0 0 0 1 0 ...
##  $ inc                : num  112000 40000 62000 136000 150000 ...
##  $ prin_rec           : num  11910 777 4250 3801 3486 ...
##  $ credit_bal         : int  18121 4046 13827 9149 22613 20582 4057 17844 9963 26782 ...
##  $ ncc                : int  9 7 10 20 16 21 3 24 11 14 ...
##  $ req                : int  1 0 0 5 1 3 1 0 1 0 ...
```

```r
str(test)
```

```
## 'data.frame':	10000 obs. of  30 variables:
##  $ n_collect          : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ credit_ratio       : num  68.7 76.6 28.2 87.6 58 88.9 42.9 52.1 87.6 76.8 ...
##  $ interest           : num  12.69 8.19 18.25 15.31 18.25 ...
```

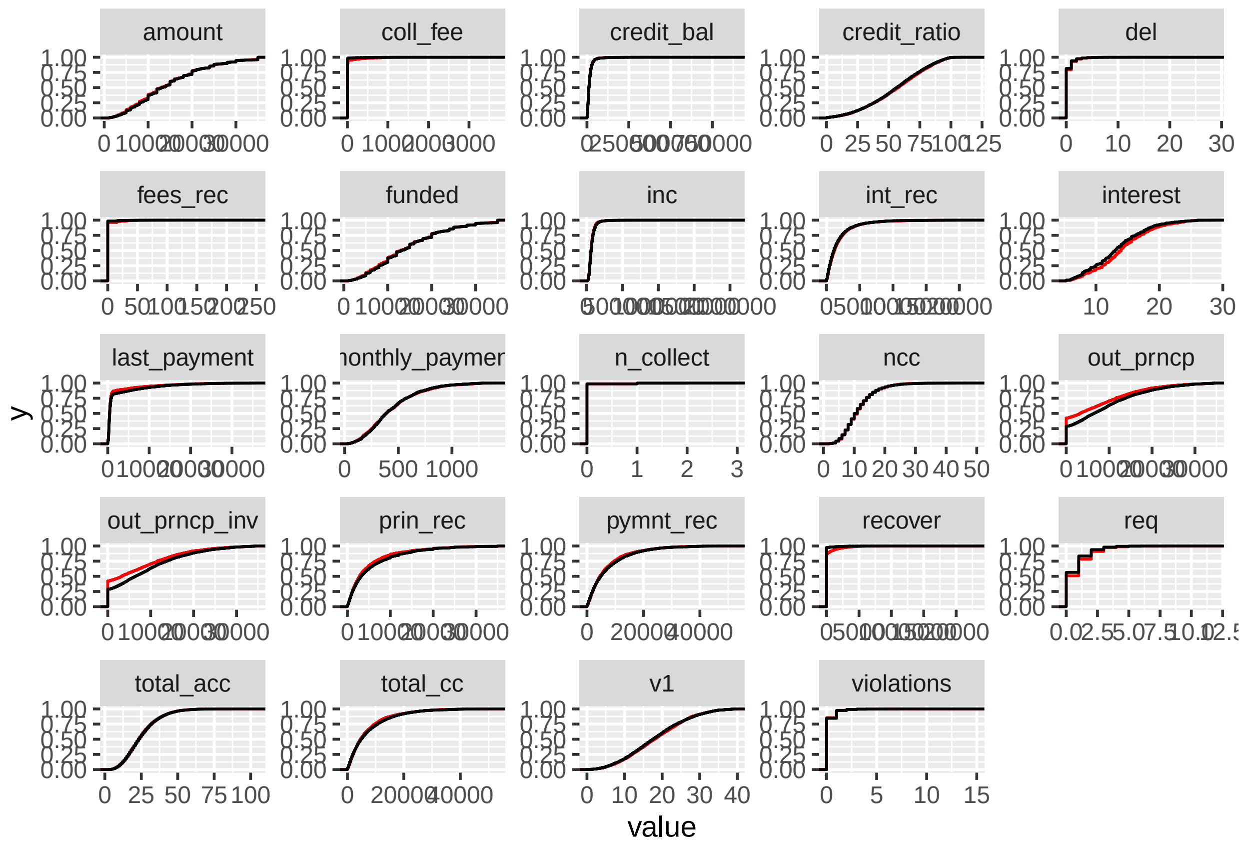
```
##  $ initial_list_status: Factor w/ 2 levels "a","b": 1 2 1 1 1 1 1 2 1 1 ...
##  $ recover            : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ coll_fee           : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ out_prncp          : num  30522 10194 10244 3781 11722 ...
##  $ total_cc           : num  2124 5537 4174 5849 2372 ...
##  $ term               : Factor w/ 2 levels " 3 yrs"," 5 yrs": 1 1 1 1 1 1 1 1 2 1 ...
##  $ fees_rec           : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ total_acc          : int  25 10 11 14 23 18 10 21 25 31 ...
##  $ employment         : Factor w/ 11 levels "< 1","1","10+",..: 1 1 7 3 4 6 NA 6 3 4 ...
##  $ amount             : int  32000 14700 12825 8000 13150 15450 4675 8775 25000 5600 ...
##  $ monthly_payment    : num  1073 462 465 279 477 ...
##  $ funded             : int  32000 14700 12825 8000 13150 15450 4675 8775 25000 5600 ...
##  $ status             : Factor w/ 3 levels "checked","partial",..: 1 3 2 2 1 3 1 1 2 1 ...
##  $ v1                 : num  15.3 10.5 13.6 17.7 25.9 ...
##  $ int_rec            : num  646 1031 1594 1630 944 ...
##  $ reason             : Factor w/ 14 levels "boat","business",..: 4 4 4 4 4 4 11 4 4 14 ...
##  $ last_payment       : num  1073 462 931 279 477 ...
##  $ pymnt_rec          : num  2124 5537 4174 5849 2345 ...
##  $ quality            : Factor w/ 7 levels "q1","q2","q3",..: 3 1 5 3 5 2 3 6 5 2 ...
##  $ out_prncp_inv      : num  30522 10194 10244 3781 11588 ...
##  $ violations         : int  0 0 0 3 2 0 0 0 0 0 ...
##  $ del                : int  1 0 0 0 3 0 0 0 0 0 ...
##  $ inc                : num  115000 55000 38000 40000 80000 ...
##  $ prin_rec           : num  1478 4506 2581 4219 1428 ...
##  $ credit_bal         : int  17101 10722 3013 10857 16289 17964 3435 5625 10427 4759 ...
##  $ ncc                : int  9 6 6 9 10 8 7 13 9 18 ...
##  $ req                : int  1 0 0 1 1 0 0 0 0 2 ...
```

Similarity Test

The first task is to get an understanding of similarity between the test and training datasets. One way to accomplish this is to view the differences in the Empirical Cumulative Distributions. Here is a visualization using black as the test and red as the training data:

```
train_n = train %>%
  select(-default) %>%
  keep(is.numeric)
test_n = test %>%
  keep(is.numeric)

train_n %>%
  gather() %>%
  ggplot() +
  facet_wrap(~ key, scales = 'free') +
  stat_ecdf(aes(value), geom = "step", col = 'red') +
  stat_ecdf(data = test_n %>% gather(), aes(value), geom = "step", col = 'black')
```



A way to quantify the difference in these samples is by using the two sample Kolmogorov- Smirnov test. This test focuses on the D statistic:

$$D_{n,m} = \sup_x |F_{1,n}(x) - F_{2,m}(x)|$$

Where $F(x)$ is are the respective empirical distribution functions. The null hypothesis is that the samples came from the same distribution and the alternative is that the distributions are different. Here is the output of the KS test applied over each column of test and train:

```r
result <- colnames(train_n) %>%
  set_names() %>%
  map(~ ks.test(train_n[, .x], test_n[, .x])) %>%
  map_dfr(., broom::tidy, .id = "parameter") %>%
  arrange(p.value)
result
```

```
## # A tibble: 24 x 5
##    parameter     statistic    p.value method                          alternative
##    <chr>             <dbl>      <dbl> <chr>                           <chr>
##  1 interest         0.092  0          Two-sample Kolmogorov-Smirnov~ two-sided
##  2 recover          0.112  0          Two-sample Kolmogorov-Smirnov~ two-sided
##  3 coll_fee         0.108  0          Two-sample Kolmogorov-Smirnov~ two-sided
##  4 out_prncp        0.138  0          Two-sample Kolmogorov-Smirnov~ two-sided
##  5 out_prncp_inv    0.138  0          Two-sample Kolmogorov-Smirnov~ two-sided
##  6 req              0.0558 0.00000117 Two-sample Kolmogorov-Smirnov~ two-sided
##  7 prin_rec         0.0519 0.00000798 Two-sample Kolmogorov-Smirnov~ two-sided
##  8 last_payment     0.0501 0.0000183  Two-sample Kolmogorov-Smirnov~ two-sided
##  9 pymnt_rec        0.0336 0.0109     Two-sample Kolmogorov-Smirnov~ two-sided
## 10 total_cc         0.0326 0.0150     Two-sample Kolmogorov-Smirnov~ two-sided
## # ... with 14 more rows
```

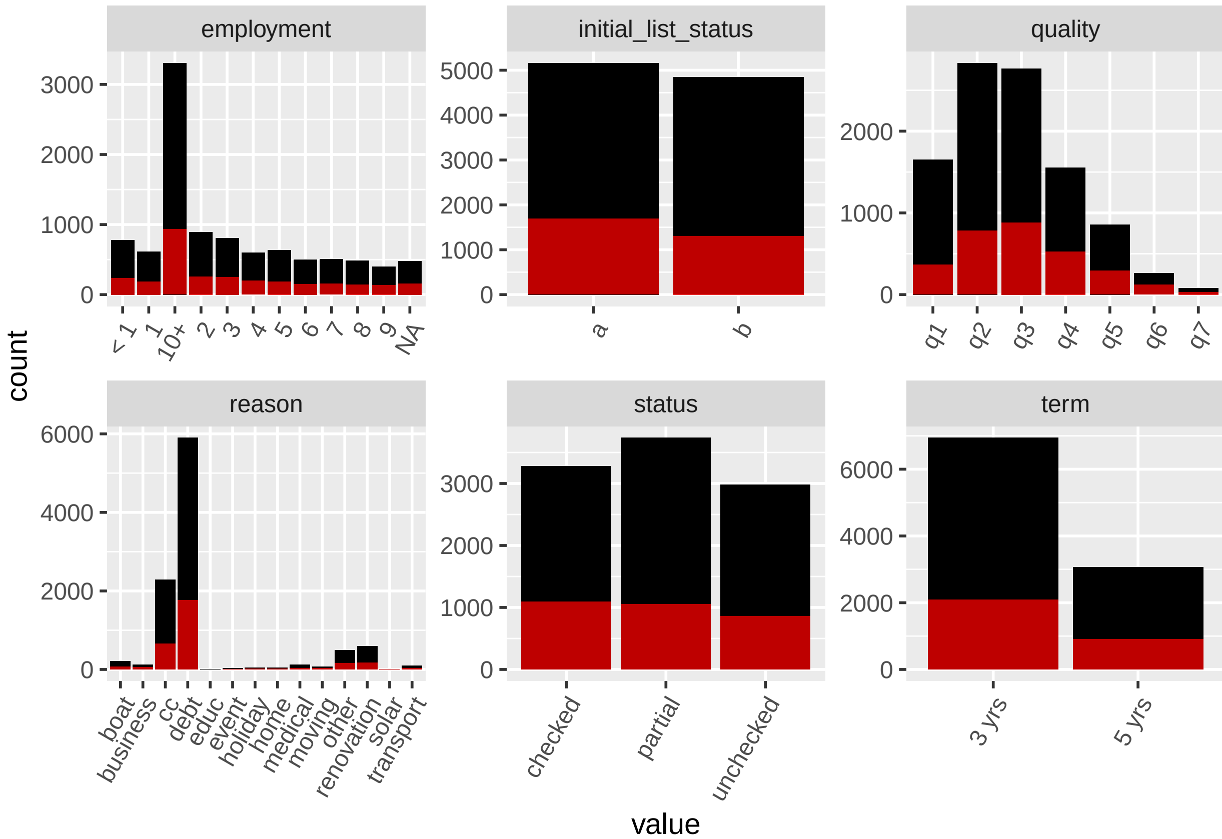
It appears that for some of the predictors, the KS test was significant. This is a caution sign against overfitting these predictors too closely as they may not generalize to the test data very well.

Similarly, we can look at a comparison of histograms for the factor variables:

```r
train_f = train %>%
  dplyr::select(-default) %>%
  keep(is.factor)

test_f = test %>%
  keep(is.factor)

train_f %>%
  gather() %>%
  ggplot(aes(value)) +
  facet_wrap(~ key, scales = 'free') +
  geom_histogram(data = test_f %>% gather(), fill = 'black', stat = 'count') +
  geom_histogram(stat = 'count', fill = 'red', alpha = .75) +
  theme(axis.text.x = element_text(angle = 60, hjust = 1))
```


employment
initial_list_status
quality
reason
status
term
count
value
< 1
1
10+
2
3
4
5
6
7
8
9
NA
a
b
q1
q2
q3
q4
q5
q6
q7
boat
business
cc
debt
educ
event
holiday
home
medical
moving
other
renovation
solar
transport
checked
partial
unchecked
3 yrs
5 yrs

VIF and Correlation

Next, we want to look at the VIF scores from a naive logisitic regression model to indicate the predictors that have the highest multicollinearity. The statistic of interest is now:

$$VIF_i = \frac{1}{1 - R_i^2}$$

Generally, $VIF(\hat{\beta}_i) > 10$ is considered multicollinear.

```
lm.fit = lm(default ~ ., data = train)
vifplot <- function(vif){
  data.frame(name = rownames(vif), vif) %>%
  arrange(-GVIF) %>%
  mutate(name = factor(name, levels = name)) %>%
  ggplot(aes(y=GVIF, x = name)) +
  geom_bar(stat="identity") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
}
```

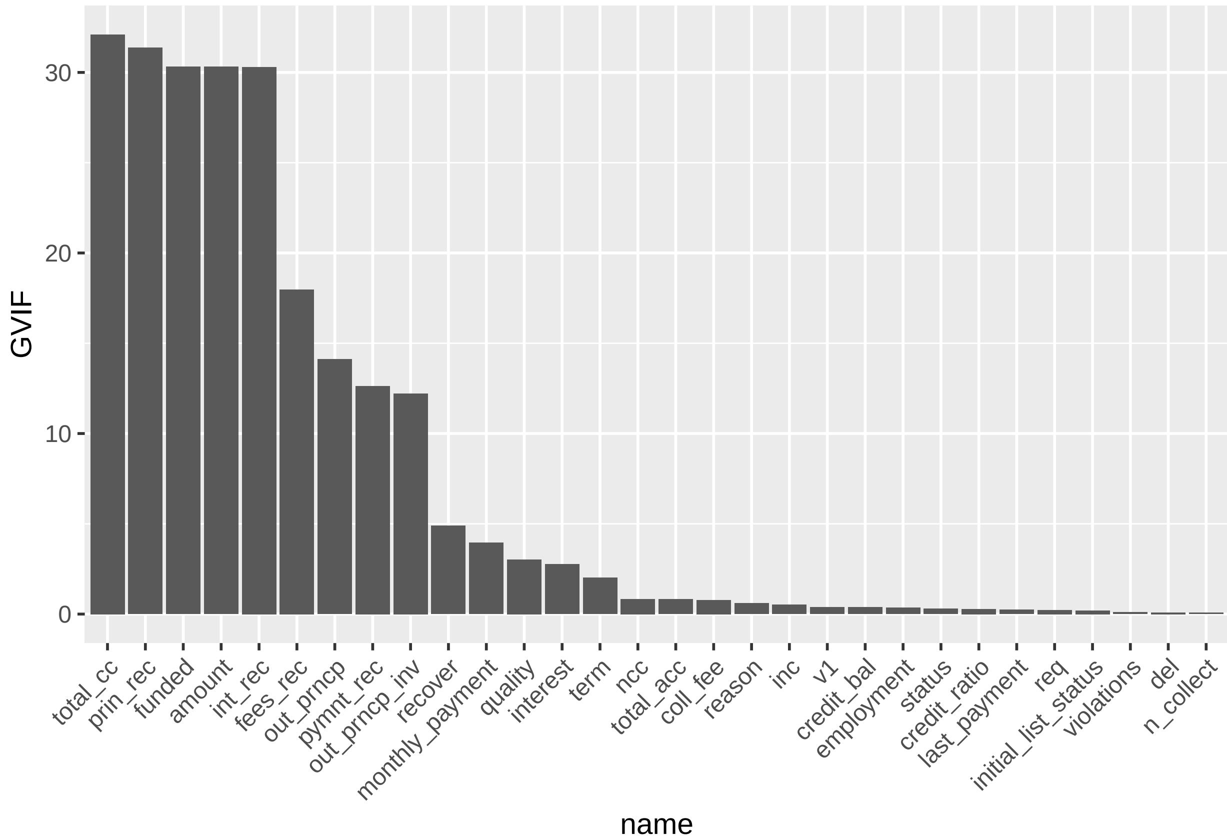
This plot shows a breakdown of VIF by variable. Note that this is plotted on a log scale to compensate for the large magnitude of VIFs in the naive model.

```
fit = vif(glm(default~., data = train, family = 'binomial'))
```

```
## Warning: glm.fit: algorithm did not converge
```

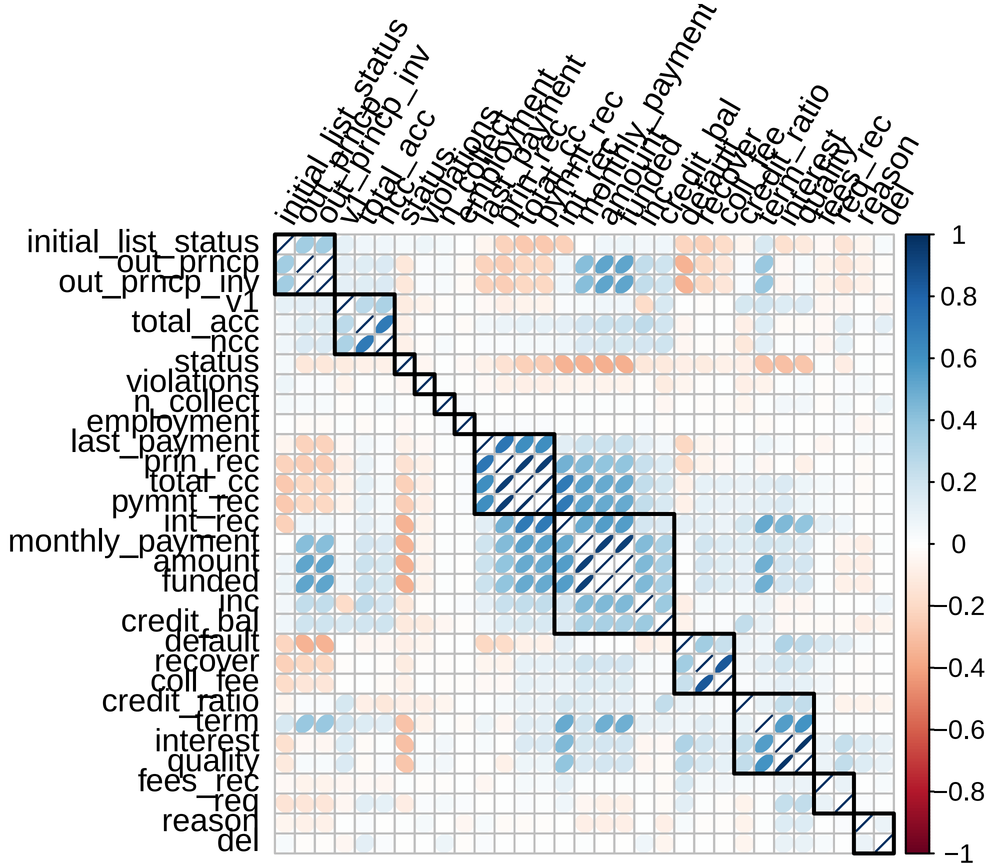
```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
fit[,'GVIF'] = log(fit[,'GVIF'])
vifplot(fit)
```



After checking for multicollinearlity, we can isolate pairwise collinearities by computing pairwise correlations and plotting it in a correlation plot. We further used hierarchial clustering to help identify clusters of predictors in a unsupervised manner.

```
hetcor = hetcor(train)
cors = hetcor$correlations
corrplot(cors,
         pch.cex = .9,
         method = 'ellipse',
         addrect = 12,
         order = "hclust",
         tl.col = "black",
         tl.srt = 60)
```



Based off of this information, our goal is to remove enough predictors to reduce the multicollinearity without adversely affecting the model's ability to predict the response. After iterating this process, this is the final list of predictors to remove:

train_s = train %>% select(-out_prncp_inv, -recover,-prin_rec,-pymnt_rec,-amount,-ncc,-total_acc,-monthly_payment)

```
train_s = train %>% select(-total_cc,
                           -amount,
                           -out_prncp,
                           -pymnt_rec,
                           -funded,
                           -monthly_payment,
                           -initial_list_status
                           )
```

We can further use an F test to compare the naive model versus the model with the multicollinear variables removed:

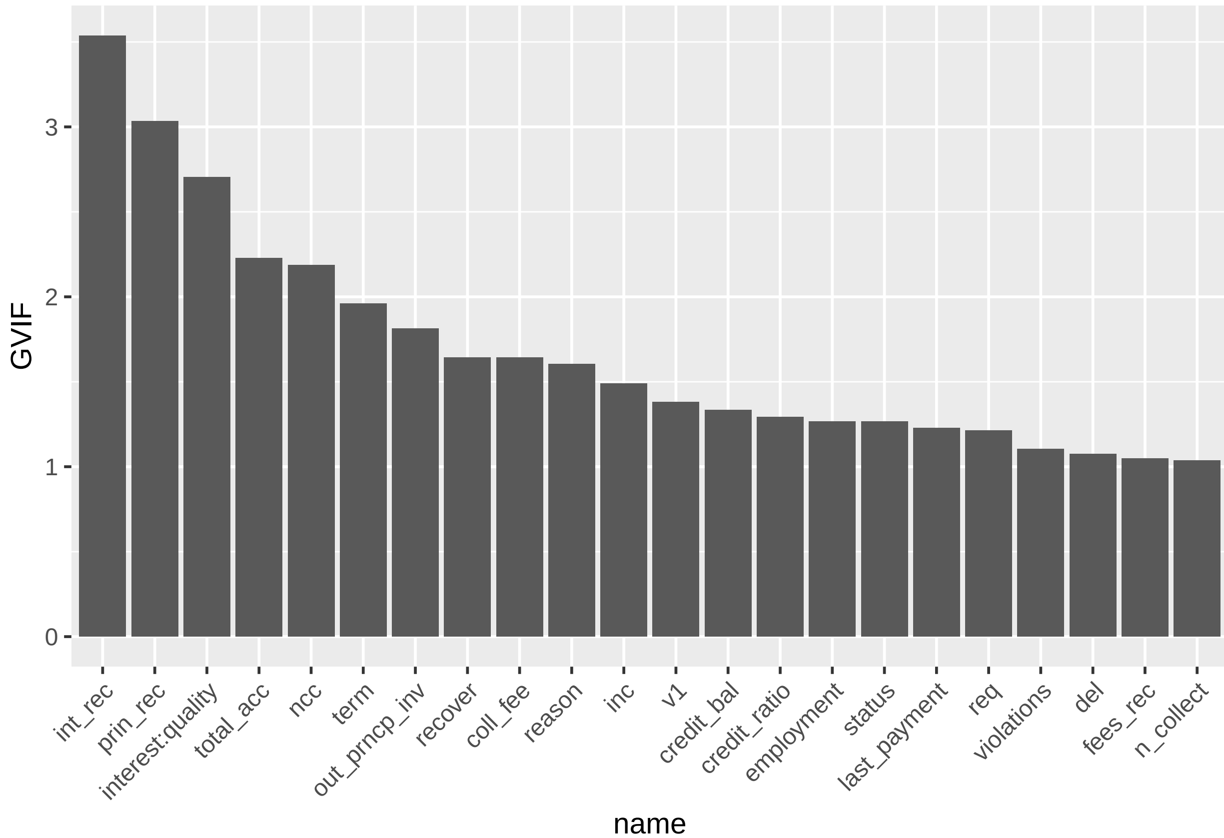
```
fit2 = glm(default~. + quality * interest - quality - interest, data = train_s, family = 'binomial')
anova(fit2, fit)
```

```
## Analysis of Deviance Table
##
## Model: binomial, link: logit
##
```

```
## Response: default
##
## Terms added sequentially (first to last)
##
##
##                  Df Deviance Resid. Df Resid. Dev
## NULL                             2839     3615.9
## n_collect         1     0.00      2838     3615.9
## credit_ratio      1    19.64      2837     3596.2
## recover           1  1013.82      2836     2582.4
## coll_fee          1     0.00      2835     2582.4
## term              1     4.36      2834     2578.0
## fees_rec          1    56.98      2833     2521.1
## total_acc         1     0.13      2832     2520.9
## employment       10    15.99      2822     2505.0
## status            2     8.29      2820     2496.7
## v1                1     5.51      2819     2491.2
## int_rec           1     7.09      2818     2484.1
## reason           12    27.23      2806     2456.8
## last_payment      1   141.70      2805     2315.1
## out_prncp_inv     1   272.04      2804     2043.1
## violations        1     2.02      2803     2041.1
## del               1     1.03      2802     2040.1
## inc               1     1.63      2801     2038.4
## prin_rec          1   110.97      2800     1927.5
## credit_bal        1     0.05      2799     1927.4
## ncc               1     0.38      2798     1927.0
## req               1     0.00      2797     1927.0
## interest:quality  7    69.30      2790     1857.7
```

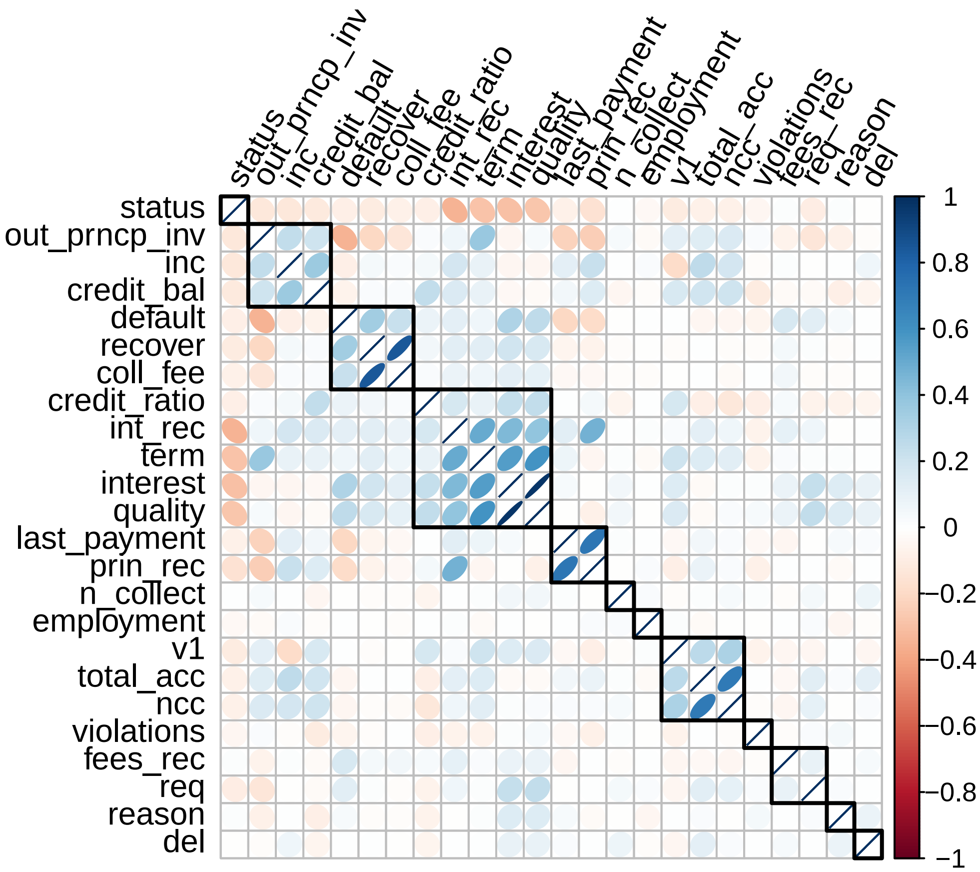
Check the VIF of the reduced model with a similar plot to the first (But now not log transformed):

```r
vifplot(vif(fit2))
```



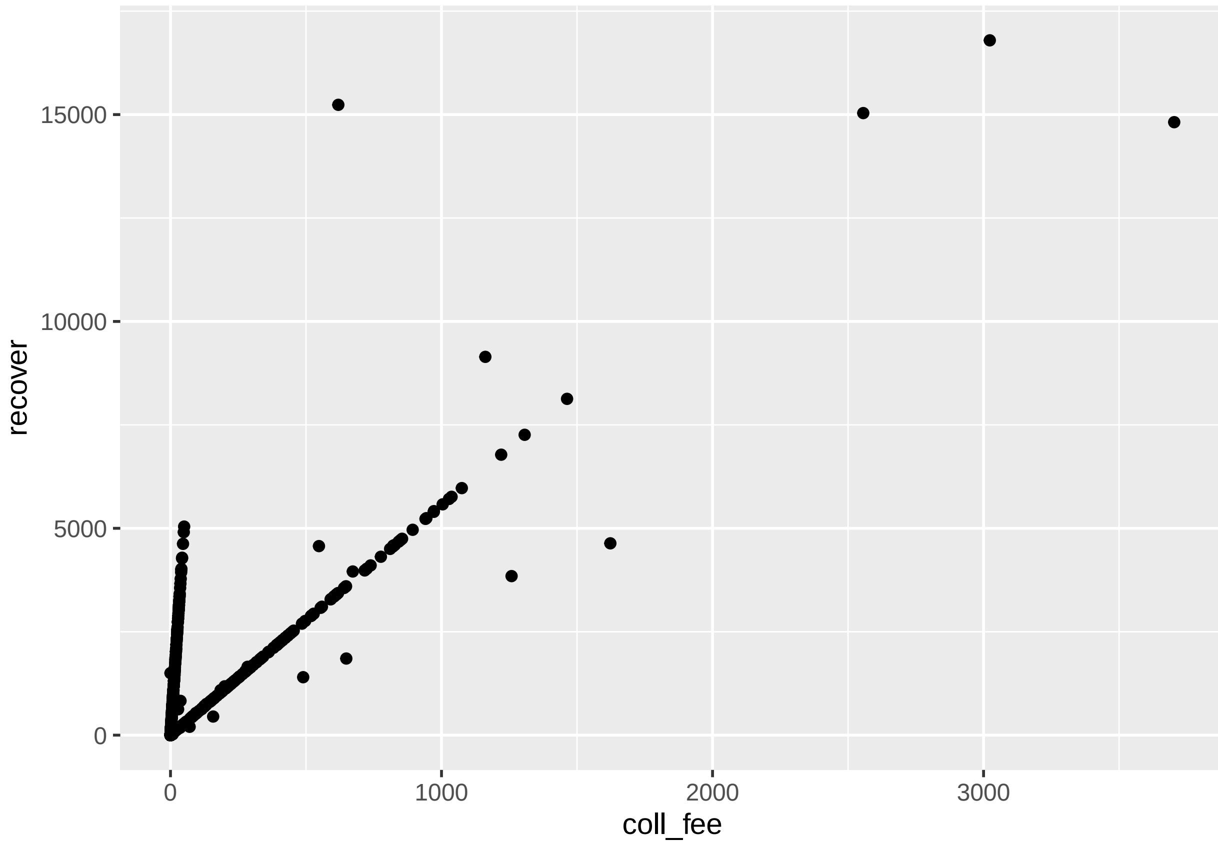
View the updated pairwise correlations. Note that size of each cluster has been reduced indicating that some of the multicollinearity has been removed from the model.

```
hetcor = hetcor(train_s)
cors = hetcor$correlations
corrplot(cors,
         pch.cex = .9,
         method = 'ellipse',
         addrect = 12,
         order = "hclust",
         tl.col = "black",
         tl.srt = 60)
```

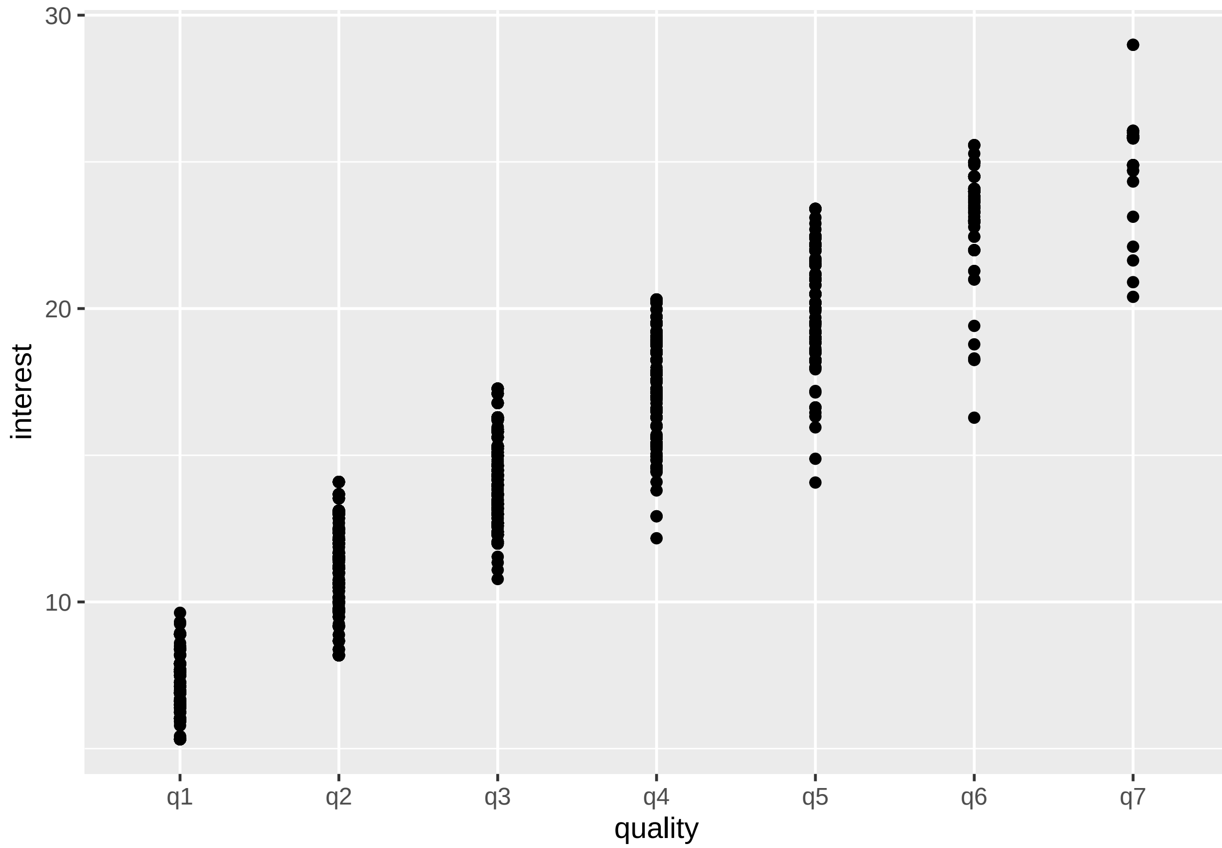


For a closer look at pairwise comparisions, we show selected variables plotted against each other.

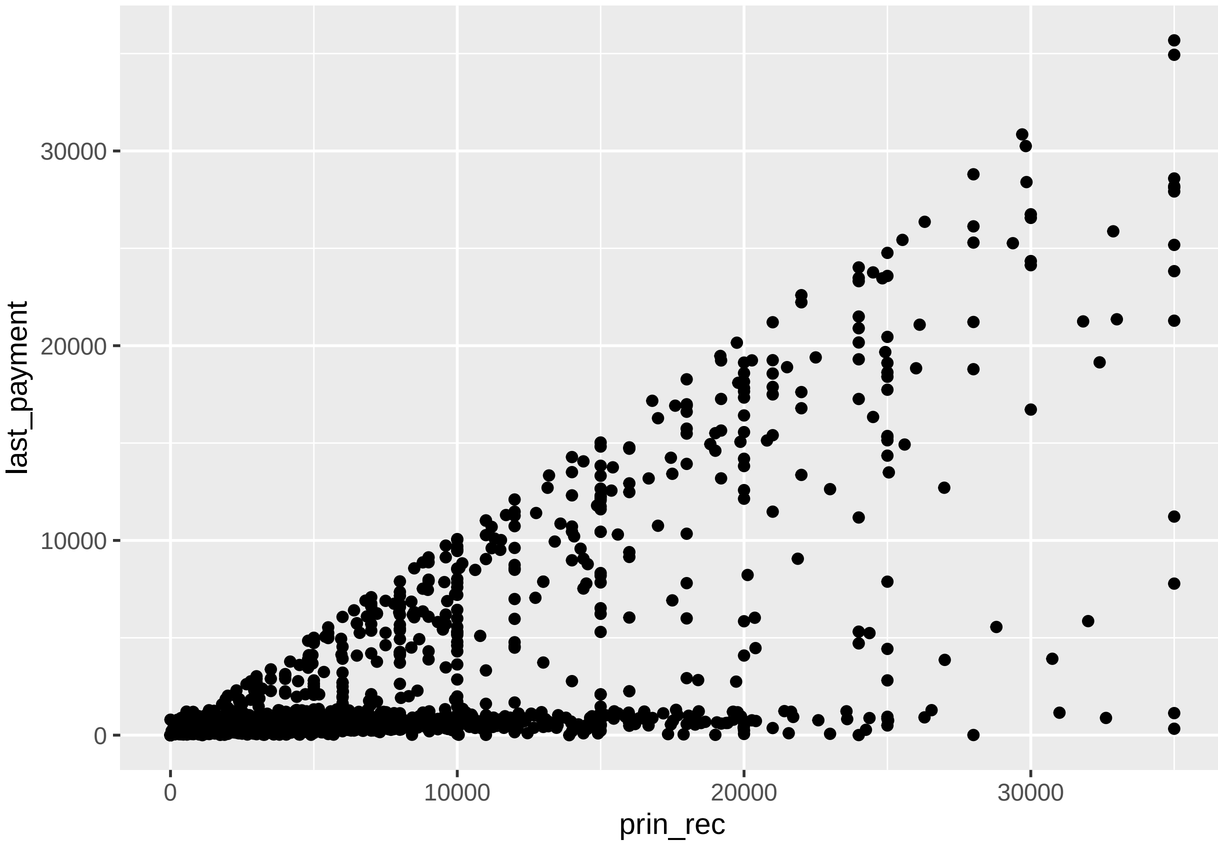
```
ggplot(train_s, aes(y = recover, x = coll_fee)) + geom_point()
```



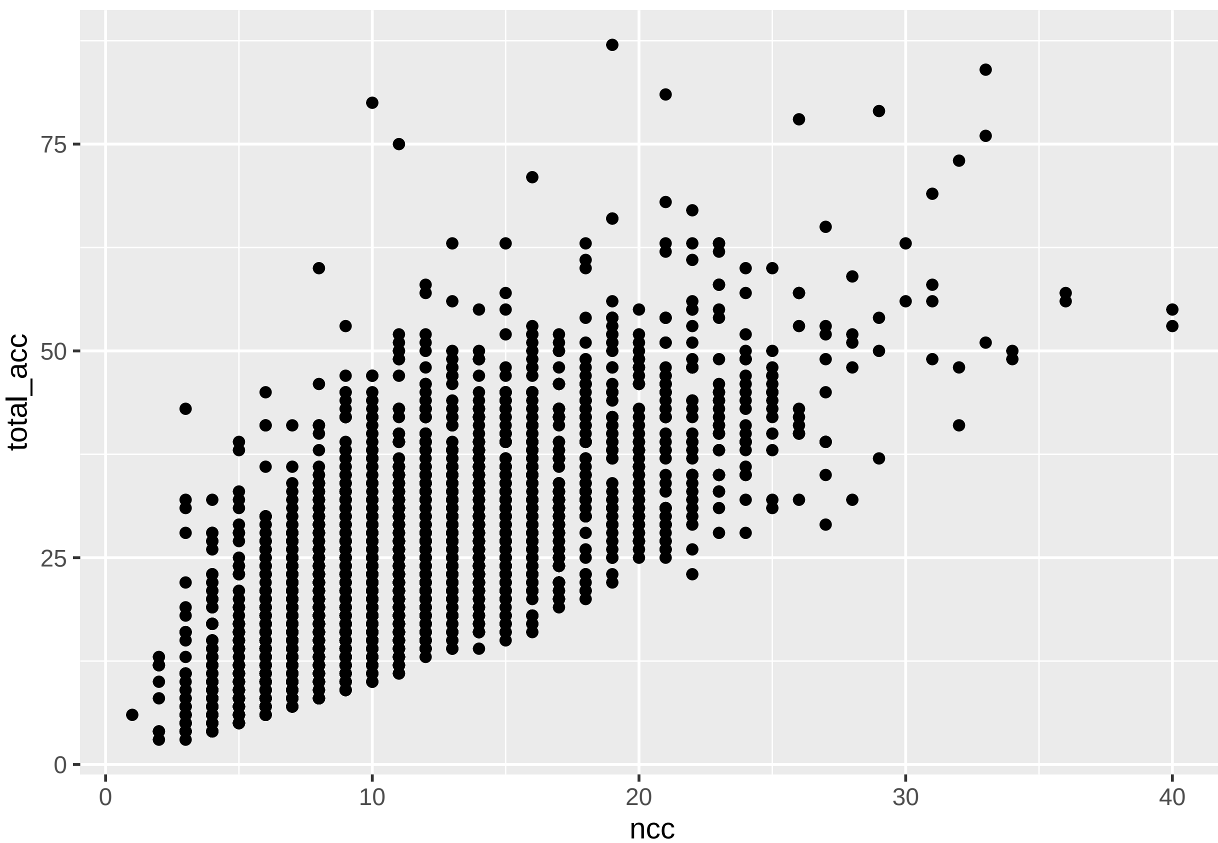

```
ggplot(train_s, aes(y = interest, x = quality)) + geom_point()
```




```
ggplot(train_s, aes(y = last_payment, x = prin_rec)) + geom_point()
```




```
ggplot(train_s, aes(y = total_acc, x = ncc)) + geom_point()
```


total_acc
ncc
0
25
50
75
0
10
20
30
40

Imputing

After reducing multicollinearity, we now check for any missing data.

```r
apply(train_s, 2, function(x) sum(is.na(x)))
```

```
##       default     n_collect  credit_ratio      interest       recover
##             0             0             0             0             0
##      coll_fee          term      fees_rec     total_acc    employment
##             0             0             0             0           160
##        status            v1       int_rec        reason  last_payment
##             0             0             0             0             0
##       quality out_prncp_inv    violations           del           inc
##             0             0             0             0             0
##      prin_rec    credit_bal           ncc           req
##             0             0             0             0
```

Employment seems to be an issue, specifically it is missing 160 entries out of 3000 training points. We can impute the missing values using a KNN implementation. Now an updated histogram shows 0 NA's for employment data.

```r
apply(test, 2, function(x) sum(is.na(x)))
```

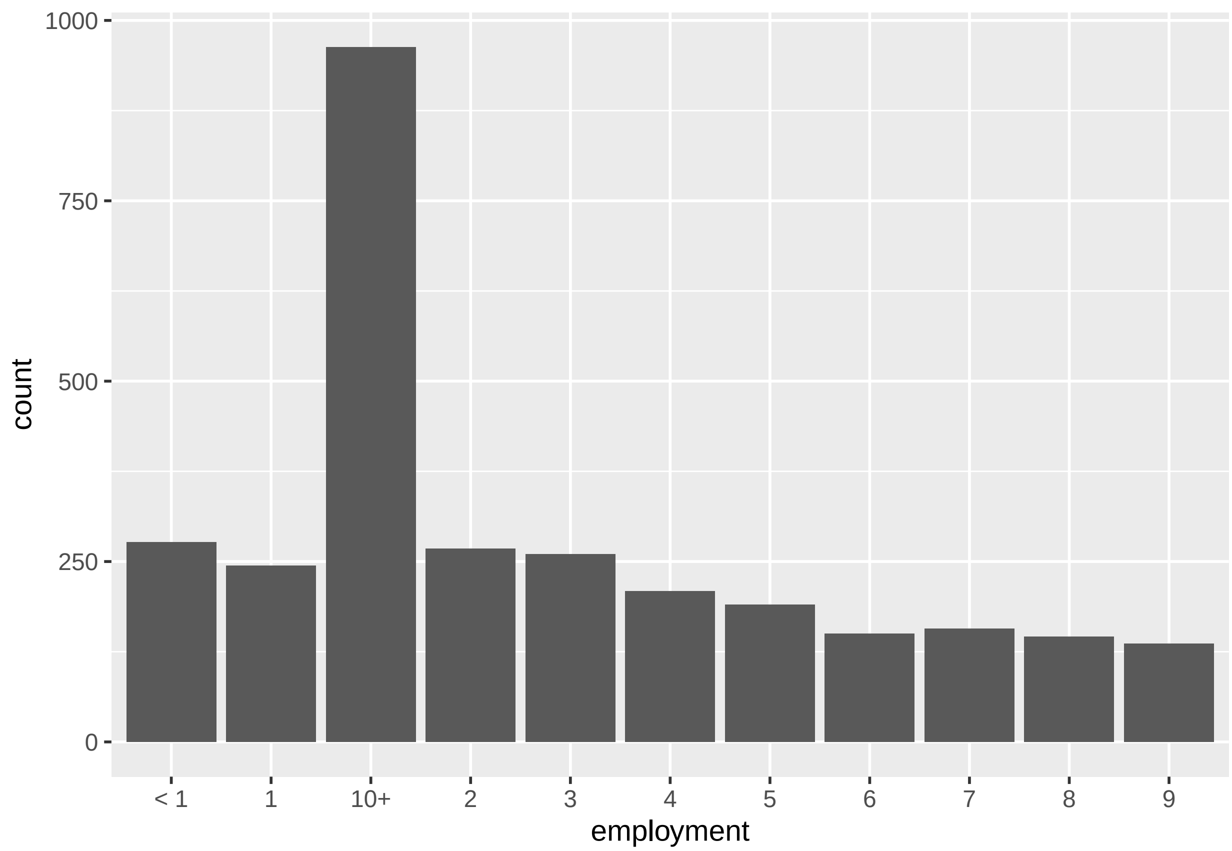
```
##           n_collect        credit_ratio            interest initial_list_status
##                   0                   0                   0                   0
##             recover            coll_fee           out_prncp            total_cc
##                   0                   0                   0                   0
##                term            fees_rec           total_acc          employment
##                   0                   0                   0                 480
##              amount     monthly_payment              funded              status
##                   0                   0                   0                   0
##                  v1             int_rec              reason        last_payment
##                   0                   0                   0                   0
##           pymnt_rec             quality       out_prncp_inv          violations
##                   0                   0                   0                   0
##                 del                 inc            prin_rec          credit_bal
##                   0                   0                   0                   0
##                 ncc                 req
##                   0                   0
```

```r
train_si = knnImputation(train_s)
apply(train_si, 2, function(x) sum(is.na(x)))
```

```
##       default     n_collect  credit_ratio      interest       recover
##             0             0             0             0             0
##      coll_fee          term      fees_rec     total_acc    employment
##             0             0             0             0             0
##        status            v1       int_rec        reason  last_payment
##             0             0             0             0             0
##       quality out_prncp_inv    violations           del           inc
##             0             0             0             0             0
##      prin_rec    credit_bal           ncc           req
##             0             0             0             0
```

```r
train_si %>% group_by(employment) %>% summarize(count = n()) %>% select(employment, count) %>%
  ggplot(aes(y = count, x = employment)) + geom_bar(stat = 'identity')
```

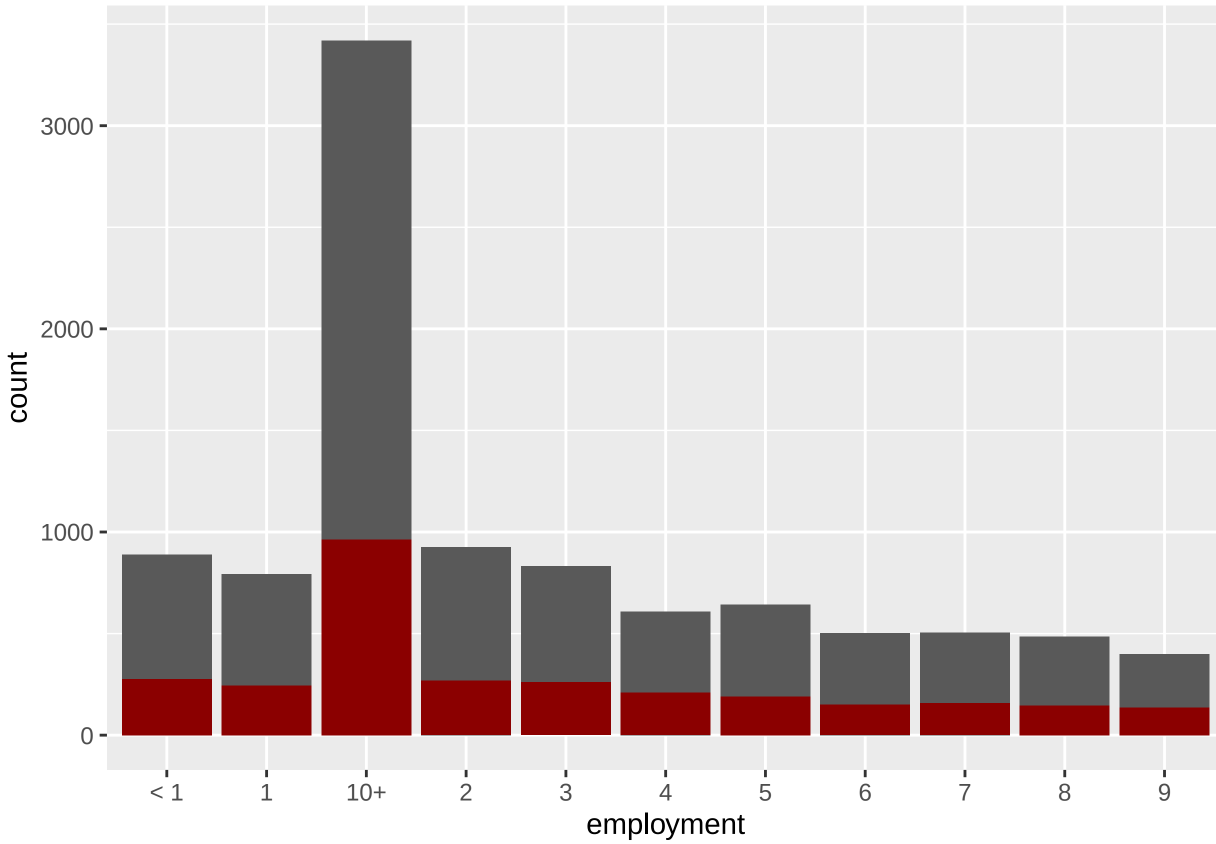
```
## `summarise()` ungrouping output (override with `.groups` argument)
```



We have to impute the employment column in test using the same method we used for the training data. We check to see that the distributions of employment in test is similar to employment in train.

```r
test_i = test
test_i$reason[test_i$reason == 'educ'] <- NA
test_i = knnImputation(test_i)
test_i %>% group_by(employment) %>% summarize(count = n()) %>% select(employment, count) %>%
  ggplot(aes(y = count, x = employment)) +
  geom_bar(stat = 'identity') +
  geom_bar(data = train_si %>% group_by(employment) %>% summarize(count = n()) %>% select(employment, co
           stat = 'identity', fill = 'darkred')
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
```



count
3000
2000
1000
0
< 1
1
10+
2
3
4
5
6
7
8
9
employment

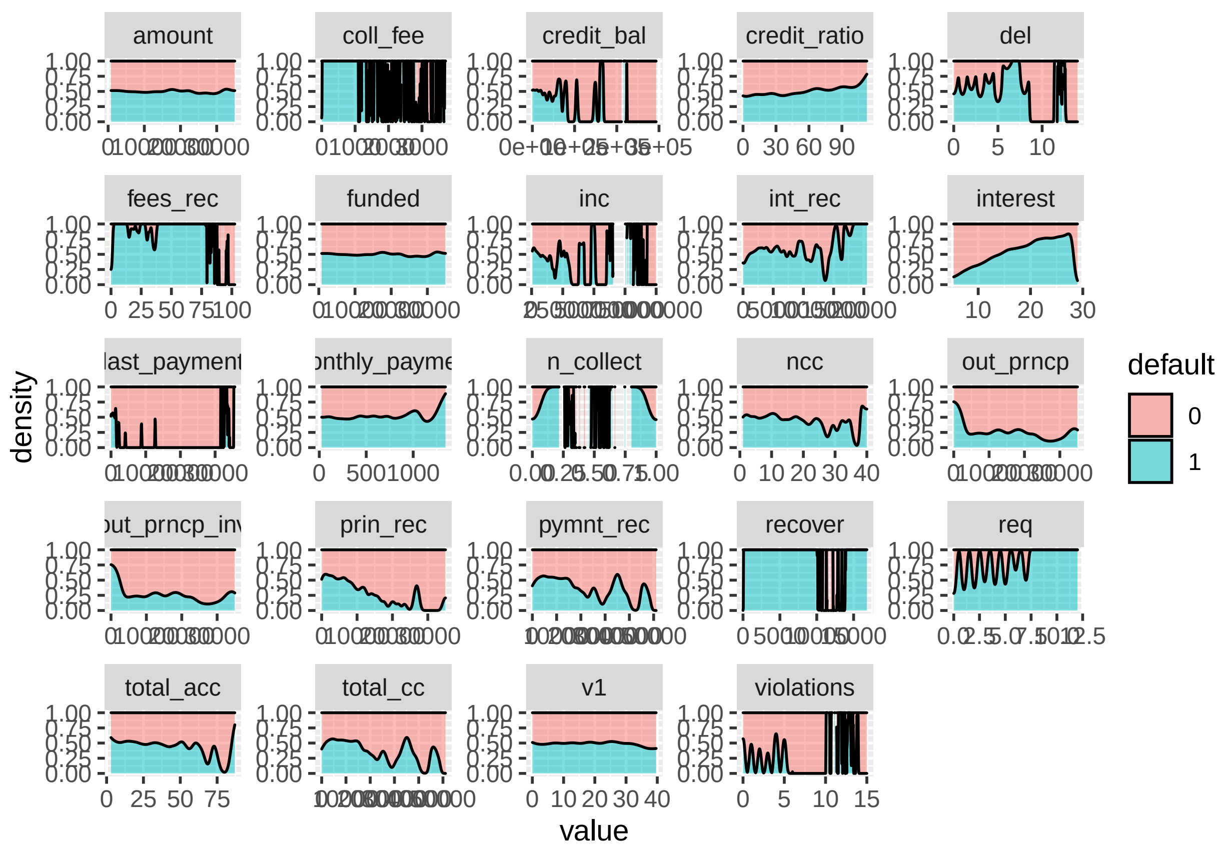
Conditional Plots

Since our ultimate goal is to estimate the Bayes Classifier, that is:

$$\hat{G}(x) = \max_{g \in \mathbb{G}} Pr(g|X = x)$$

It would be useful to view a conditional plot of default against all the predictor variables:

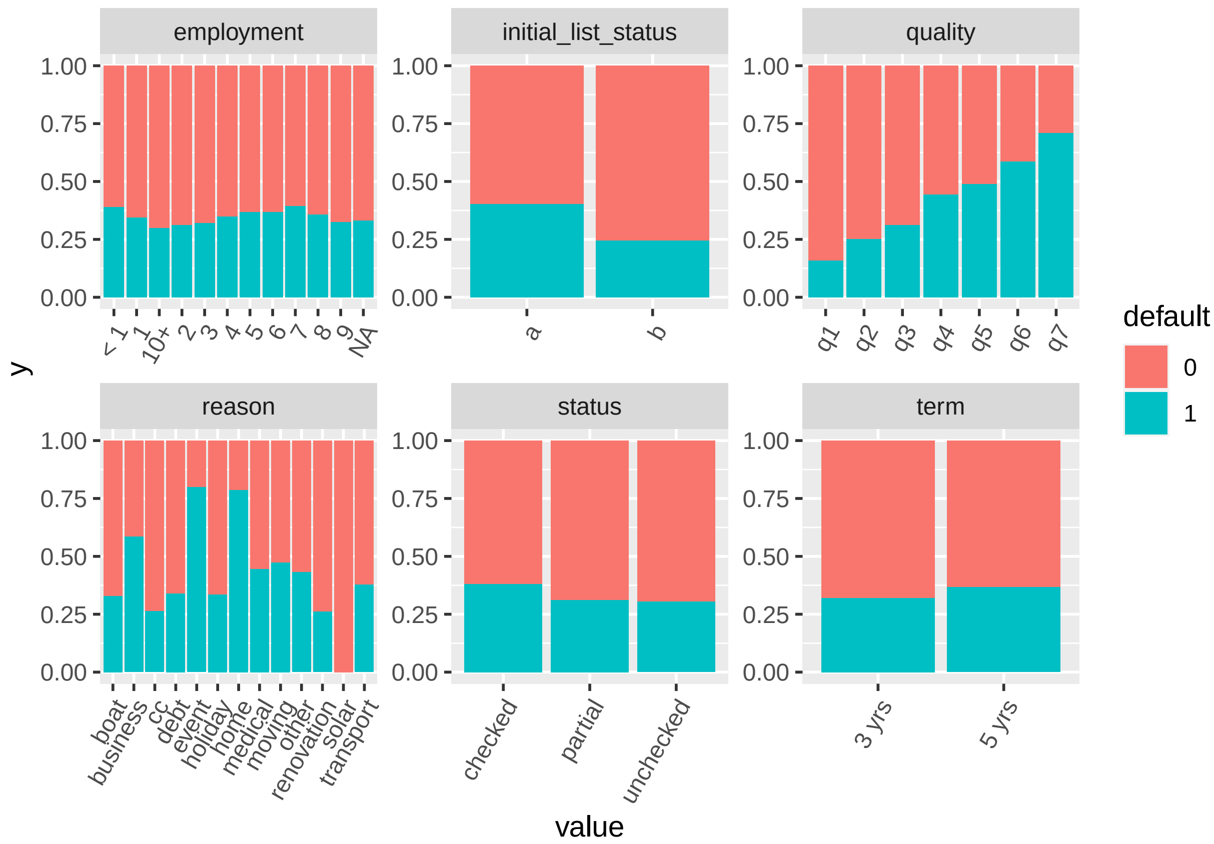
```r
train %>%
  keep(is.numeric) %>%
  mutate(default = as.factor(default)) %>%
  gather(-default, key = 'key', value = 'value') %>%
  ggplot(aes(value, fill = default)) +
  facet_wrap(~ key, scales = 'free') +
  geom_density(position = 'fill',alpha = .5)
```



Similarly for factor predictors, we can view stacked histograms showing the percentage of default vs not default conditioned on different class values:

```r
train %>%
  mutate(default = as.factor(default)) %>%
  keep(is.factor) %>%
  gather(-default, key = 'key', value = 'value') %>%
  ggplot(aes(fill = default, y= 1, x=value)) +
  geom_bar(position = "fill",stat="identity") +
```

```r
facet_wrap(~ key, scales = 'free') +
theme(axis.text.x = element_text( angle = 60,  hjust = 1 ) )
```



Modeling

Now we want to begin modeling the response. We fit a series of models that vary in methodology and flexibility. We begin with a base model using linear logistic regression, explore flexible nonparametric methods using a vareity of kernel methods, and then extend the flexibility of the linear logisitic model by modeling with a generalized additive linear logisitic model.

First, we build a baseline model. This model will be logistic regression (from the family of generalized linear models) of default against all the predictors. The model is:

$$Pr(G = k|X = x) = \frac{e^{(\beta_{k0} + \beta_k^T x)}}{1 + \sum_{K-1}(\beta_{l0} + \beta_l^T x)}, k = 1, ..., K - 1$$

Where in our case $K = 2$ and this simplifies to a single linear model that is fit using Maximum Likelihood Estimation. Additionally, we split the training data into a train and dev set to enable the use of cross validation.

```
set.seed(123)
index = sample(nrow(train_si), .7*nrow(train_si))
dev = train_si[-index,]
train_sit = train_si[index,]
```

```
base = glm(default~., data = train_sit, family = 'binomial')
preds = predict(base, newdata = dev, type = 'response')
class = rep(0, length(preds))
class[preds>.5] = 1
mean(dev$default == class)
```

```
## [1] 0.8366667
```

```
summary(base)
```

```
##
## Call:
## glm(formula = default ~ ., family = "binomial", data = train_sit)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.0531  -0.5729  -0.2624   0.0000   5.2754
##
## Coefficients:
##                    Estimate Std. Error z value Pr(>|z|)
## (Intercept)      -3.430e+00  7.657e-01  -4.479 7.49e-06 ***
## n_collect         9.736e-01  5.577e-01   1.746 0.080820 .
## credit_ratio     -1.026e-03  3.176e-03  -0.323 0.746527
## interest          3.047e-01  5.711e-02   5.336 9.49e-08 ***
## recover           1.645e+00  4.449e+01   0.037 0.970497
## coll_fee         -1.908e+00  1.914e+02  -0.010 0.992047
## term 5 yrs        2.444e-01  2.085e-01   1.173 0.240933
## fees_rec          7.985e-02  1.529e-02   5.222 1.77e-07 ***
## total_acc         6.399e-03  8.327e-03   0.768 0.442221
## employment1      -8.092e-01  3.199e-01  -2.530 0.011422 *
## employment10+    -5.224e-01  2.501e-01  -2.089 0.036707 *
```

```
## employment2      -6.622e-01  3.147e-01  -2.105 0.035333 *
## employment3      -8.922e-01  3.391e-01  -2.631 0.008515 **
## employment4      -3.660e-01  3.385e-01  -1.081 0.279681
## employment5      -2.287e-02  3.298e-01  -0.069 0.944705
## employment6      -3.475e-01  3.892e-01  -0.893 0.371906
## employment7      -4.178e-01  3.677e-01  -1.136 0.255815
## employment8      -5.914e-01  3.624e-01  -1.632 0.102710
## employment9      -7.206e-01  3.846e-01  -1.874 0.060969 .
## statuspartial     1.353e-01  1.645e-01   0.823 0.410699
## statusunchecked  -4.150e-01  1.845e-01  -2.249 0.024505 *
## v1               -2.831e-03  9.554e-03  -0.296 0.766967
## int_rec           2.635e-04  5.785e-05   4.554 5.26e-06 ***
## reasonbusiness    1.238e+00  7.384e-01   1.676 0.093728 .
## reasoncc          6.576e-01  5.322e-01   1.236 0.216574
## reasondebt        7.295e-01  5.172e-01   1.411 0.158369
## reasonevent       2.889e+01  3.561e+05   0.000 0.999935
## reasonholiday     4.916e-02  9.285e-01   0.053 0.957772
## reasonhome        2.746e+00  1.390e+00   1.975 0.048257 *
## reasonmedical     5.405e-01  8.653e-01   0.625 0.532188
## reasonmoving      6.960e-01  7.330e-01   0.950 0.342332
## reasonother       5.497e-02  5.692e-01   0.097 0.923063
## reasonrenovation  3.057e-01  6.037e-01   0.506 0.612594
## reasonsolar      -2.428e+01  2.487e+05   0.000 0.999922
## reasontransport  -2.271e-01  9.529e-01  -0.238 0.811643
## last_payment     -2.652e-04  6.368e-05  -4.165 3.11e-05 ***
## qualityq2        -4.913e-01  3.566e-01  -1.378 0.168198
## qualityq3        -1.449e+00  4.794e-01  -3.023 0.002500 **
## qualityq4        -1.547e+00  6.151e-01  -2.515 0.011912 *
## qualityq5        -2.533e+00  7.804e-01  -3.246 0.001170 **
## qualityq6        -3.278e+00  9.971e-01  -3.288 0.001010 **
## qualityq7        -3.337e+00  1.301e+00  -2.566 0.010292 *
## out_prncp_inv    -1.666e-04  1.412e-05 -11.802  < 2e-16 ***
## violations       -1.996e-01  1.356e-01  -1.472 0.141142
## del               1.341e-01  7.483e-02   1.793 0.073048 .
## inc               5.119e-06  1.471e-06   3.479 0.000503 ***
## prin_rec         -1.672e-04  2.932e-05  -5.703 1.18e-08 ***
## credit_bal        5.013e-06  4.354e-06   1.151 0.249593
## ncc              -5.252e-03  1.974e-02  -0.266 0.790160
## req              -2.090e-02  6.498e-02  -0.322 0.747703
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2681.6  on 2099  degrees of freedom
## Residual deviance: 1372.4  on 2050  degrees of freedom
## AIC: 1472.4
##
## Number of Fisher Scoring iterations: 25
```

Based upon our EDA, we would like to consider adding the interaction term (quality * interest) and (last_payment * prin_rec). We can see that this improves the test performance marginally.

```r
interaction = glm(default~.+ quality * interest + last_payment * prin_rec,
           data = train_sit, family = 'binomial')
preds = predict(interaction, newdata = dev, type = 'response')
class = rep(0, length(preds))
class[preds>.5] = 1
mean(dev$default == class)
```

```
## [1] 0.8366667
```

For our next model we would like to consider a KNN classificer which makes estimations using:

$$\hat{Y}_k = \frac{1}{k} \sum_{x_i \in N_k(x)} y_i$$

For this model, we need to make "dummy" variables for the factors.

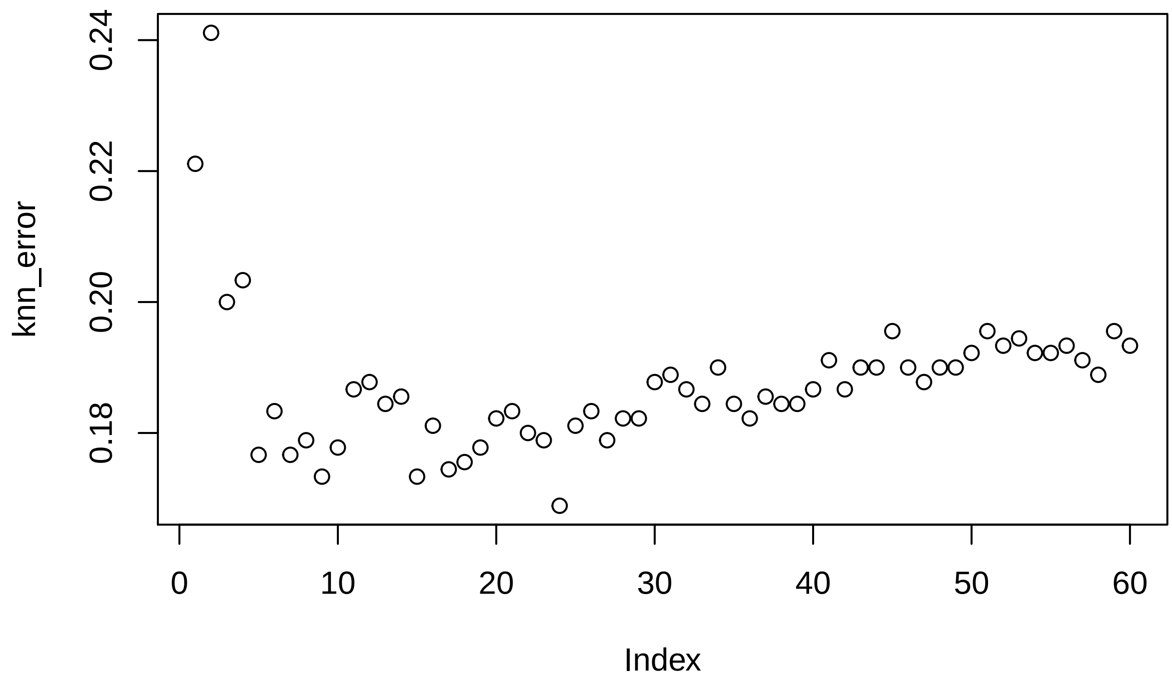
```r
train_knn = train_sit
test_knn = dev

train_knn$term = dummy.code(train_knn$term)
train_knn$employment = dummy.code(train_knn$employment)
train_knn$status = dummy.code(train_knn$status)
train_knn$reason = dummy.code(train_knn$reason)
train_knn$quality = dummy.code(train_knn$quality)

test_knn$term = dummy.code(test_knn$term)
test_knn$employment = dummy.code(test_knn$employment)
test_knn$status = dummy.code(test_knn$status)
test_knn$reason = dummy.code(test_knn$reason)
test_knn$quality = dummy.code(test_knn$quality)
```

Next we fit a series of models indexed by k and select the best performing parameter by cross validation/ 1 Standard Error Rule:

```r
set.seed(1)
library(class)
knn_error = c()
for(k in 1:60){
  knn.pred = knn(train_knn[,-1], test_knn[,-1], as.factor(train_sit[,1]), k = k)
  knn_error[k] = mean(knn.pred!=dev[,1])
}
plot(knn_error)
```



Finally, the optimal knn model.

```
knn.pred = knn(train_knn[,-1], test_knn[,-1], as.factor(train_sit[,1]), k = 9)
mean(knn.pred==dev[,1])
```

```
## [1] 0.8266667
```

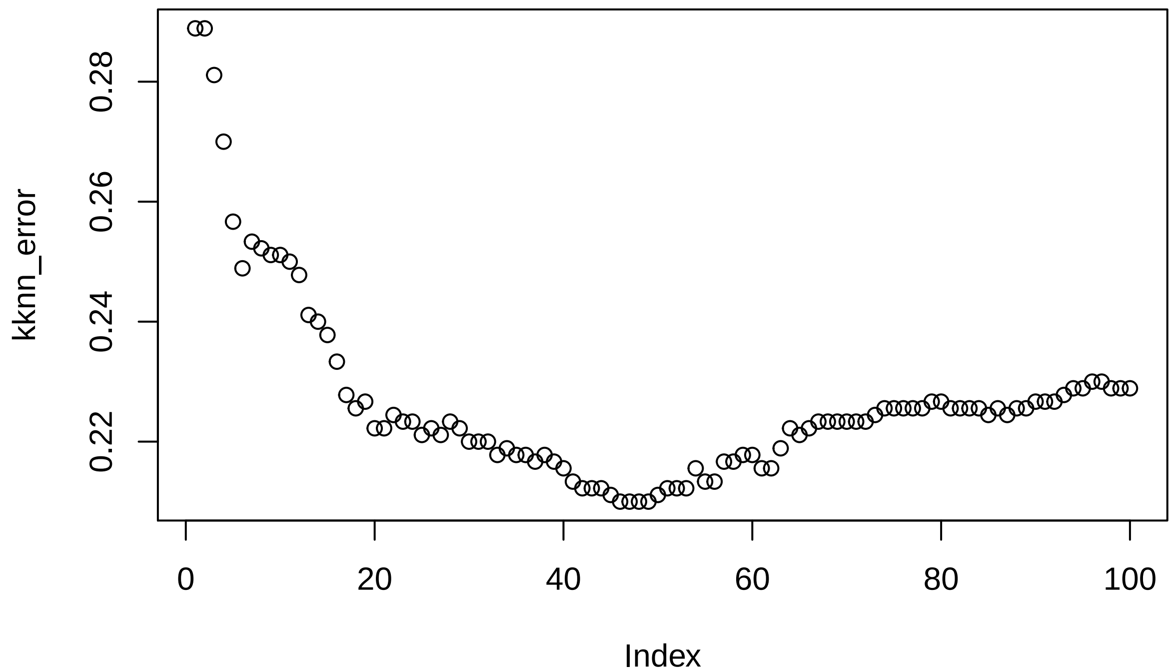
Now, we use a variant of KNN by generalizing the distance kernel from euclidean distance to the epanechinikov kernel. This allows a weighting of neighbors according to their distances following the formula:

$$K(u) = \frac{3}{4}(1 - u^2), |u| \leq 1$$

```
set.seed(1)
library(kknn)
```

```
## Warning: package 'kknn' was built under R version 4.0.3
```

```
kknn_error = c()
for(k in 1:100){
  kknn = kknn(as.factor(default) ~ ., train_knn, test_knn, kernel = 'epanechnikov', k = k)
  kknn.pred = kknn$fitted.values
  kknn_error[k] = mean(kknn.pred!=dev[,1])
}
plot(kknn_error)
```




```r
which.min(kknn_error)
```

```
## [1] 46
```

As seen below, the epanechinikov kernel performs worse than the standard euclidean on the cross validated dataset.

```r
kknn = kknn(as.factor(default) ~ ., train_sit, dev, kernel = 'epanechnikov', k = 46)
kknn.pred = kknn$fitted.values
mean(kknn.pred==dev[,1])
```

```
## [1] 0.7833333
```

We now move onto the naive Bayes model which makes the assumption that given a class $G = j$, the features X_k are independent:

$$f_j(x) = \prod_{k=1}^{p} f_{jk}(X_k)$$

```r
nb <- naiveBayes(default ~ ., data = train_sit)
preds = apply(predict(nb, dev, type = "raw"), 1, which.max)-1
mean(preds==dev[,1])
```

```
## [1] 0.7355556
```

The performance is not great so we decide to use SVM, a more flexible method. SVM solves the problem:

$$\max L_D = \max \sum_N \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{i'=1}^{N} \alpha_i \alpha_{i'} y_i y_{i'} x_i^T x_i'$$

Where we generalize the inner product of x_i and x_i' with transformed feature vectors $K(x, x') =< h(x), h(x') >$ in the form of a polynomial, radial, and sigmoid function.

```
set.seed(1)
svm.linear <- svm(default ~ ., data = train_sit, kernel = "linear", cross = 10)
svm.linear$tot.MSE
```

```
##             [,1]
## [1,] 0.2112108
```

```
svm.poly <- svm(default ~ ., data = train_sit, kernel = "polynomial", cross = 10)
svm.poly$tot.MSE
```

```
##            [,1]
## [1,] 0.259441
```

```
svm.rad <- svm(default ~ ., data = train_sit, kernel = "radial", cross = 10)
svm.rad$tot.MSE
```

```
##             [,1]
## [1,] 0.1239869
```

```
svm.sig <- svm(default ~ ., data = train_sit, kernel = "sigmoid", cross = 10)
svm.sig$tot.MSE
```

```
##            [,1]
## [1,] 1.186739
```

```
svm.rad.preds <- as.numeric(predict(svm.rad, newdata = dev))
svm.rad.class = rep(0, length(svm.rad.preds))
svm.rad.class[svm.rad.preds>.5] = 1
mean(dev$default == svm.rad.class)
```

```
## [1] 0.8211111
```

The performance is as good as our base model using linear logisitic regression. Due to our base model's interpretibility and ability to conduct inference, we omit these models. Next we would like to extend the flexibility of our base model by considering a Nonparametric Logistic Regression in the form of a Generalized Additive Model. This has the form:

$$g(\mu) = \alpha + f_1(X_1) + f_2(X_2) + ... + f_p(X_p)$$

Where $g(\mu) = \frac{\mu(X)}{1-\mu(X)}$ is the logit link function. More specifically, since some of our predictors are factors variables, this will be a semiparametric model of the form:

$$g(\mu) = X^T\beta + \alpha_k + f(Z)$$

```r
# Select all predictors that are numeric with more than 2 distinct values
ind = apply(train_sit %>% keep(is.numeric), 2, function(x) length(unique(x)))
ind = ind > 2
s_predictors = names(apply(train_sit %>% keep(is.numeric), 2, function(x) length(unique(x))))[ind]
# reason is omitted because of incapability with test set
s_predictors = s_predictors[s_predictors != "reason"]
mid = paste(s_predictors, collapse = ") + s(")
form = as.formula(paste('default ~ . + s(', mid, ')'))
form
```

```
## default ~ . + s(credit_ratio) + s(interest) + s(recover) + s(coll_fee) +
##     s(fees_rec) + s(total_acc) + s(v1) + s(int_rec) + s(last_payment) +
##     s(out_prncp_inv) + s(violations) + s(del) + s(inc) + s(prin_rec) +
##     s(credit_bal) + s(ncc) + s(req)
```

```r
gam = gam(form, family = binomial, data = train_sit)
```

```r
summary(gam)
```

```
##
## Call: gam(formula = form, family = binomial, data = train_sit)
## Deviance Residuals:
##        Min         1Q     Median         3Q        Max
## -1.885e+00 -4.540e-01 -1.772e-01  2.107e-08  3.178e+00
##
## (Dispersion Parameter for binomial family taken to be 1)
##
##     Null Deviance: 2681.6 on 2099 degrees of freedom
## Residual Deviance: 968.019 on 1991.999 degrees of freedom
## AIC: 1184.021
##
## Number of Local Scoring Iterations: NA
##
## Anova for Parametric Effects
##                      Df  Sum Sq Mean Sq  F value    Pr(>F)
## n_collect             1    1.95   1.945   2.5039  0.113726
## credit_ratio          1    6.01   6.009   7.7331  0.005473 **
## interest              1   15.09  15.094  19.4258 1.101e-05 ***
## recover               1    0.00   0.000   0.0000  0.999587
## coll_fee              1    0.00   0.000   0.0000  0.998334
## term                  1   44.01  44.008  56.6388 7.883e-14 ***
## fees_rec              1    2.40   2.405   3.0947  0.078702 .
## total_acc             1    3.30   3.298   4.2442  0.039514 *
## employment           10    8.89   0.889   1.1440  0.325007
## status                2    0.68   0.341   0.4386  0.644990
## v1                    1    0.99   0.986   1.2688  0.260133
## int_rec               1    4.74   4.738   6.0977  0.013619 *
## reason               12   11.00   0.917   1.1797  0.291748
## last_payment          1   24.75  24.749  31.8530 1.901e-08 ***
## quality               6   42.64   7.107   9.1465 6.752e-10 ***
## out_prncp_inv         1   82.44  82.441 106.1030 < 2.2e-16 ***
## violations            1    0.49   0.488   0.6281  0.428157
```

```
## del                1    2.43   2.432   3.1307  0.076985 .
## inc                1    0.03   0.030   0.0382  0.845155
## prin_rec           1  225.76 225.763 290.5618 < 2.2e-16 ***
## credit_bal         1    0.56   0.558   0.7187  0.396675
## ncc                1    1.12   1.118   1.4385  0.230530
## req                1    0.13   0.129   0.1663  0.683465
## s(recover)         1    0.00   0.000   0.0000  0.996086
## s(fees_rec)        1    0.15   0.149   0.1923  0.661030
## s(int_rec)         1    0.39   0.388   0.4998  0.479674
## s(last_payment)    1    0.21   0.211   0.2715  0.602353
## s(out_prncp_inv)   1    0.23   0.233   0.3004  0.583714
## s(inc)             1    0.50   0.502   0.6460  0.421655
## s(prin_rec)        1    0.10   0.097   0.1254  0.723238
## Residuals       1992 1547.76   0.777
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Anova for Nonparametric Effects
##                  Npar Df Npar Chisq    P(Chi)
## (Intercept)
## n_collect
## credit_ratio
## interest
## recover
## coll_fee
## term
## fees_rec
## total_acc
## employment
## status
## v1
## int_rec
## reason
## last_payment
## quality
## out_prncp_inv
## violations
## del
## inc
## prin_rec
## credit_bal
## ncc
## req
## s(credit_ratio)        3      6.838   0.07726 .
## s(interest)            3     10.033   0.01829 *
## s(recover)             3      0.000   1.00000
## s(coll_fee)            3      0.000   1.00000
## s(fees_rec)            3      9.266   0.02595 *
## s(total_acc)           3      1.969   0.57891
## s(v1)                  3      2.677   0.44423
## s(int_rec)             3     94.104 < 2.2e-16 ***
## s(last_payment)        3     40.679 7.649e-09 ***
## s(out_prncp_inv)       3    131.177 < 2.2e-16 ***
## s(violations)          3      0.914   0.82201
```

```
## s(del)                 3      5.073   0.16651
## s(inc)                 3      2.164   0.53906
## s(prin_rec)            3     59.211 8.665e-13 ***
## s(credit_bal)          3      5.583   0.13378
## s(ncc)                 3      4.127   0.24806
## s(req)                 3      0.866   0.83369
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
preds = predict(gam, newdata = dev)
class = rep(0, length(preds))
class[preds>.5] = 1
mean(dev$default == class)
```

```
## [1] 0.8833333
```

Final Model

The performance with this more flexible model appears to have improved the test MSE by a moderate margin. Now we would like to consider a larger set of predictors and use a greedy algorithm (forward stepwise) to choose the best subset based upon the *AIC* criterion. Additionally, we are going to allow the model to consider higher dgf for each predictor allowing for a much more flexible fit:

```
set.seed(123)
# Reimpute since we are using all the variables now
train_i = knnImputation(train)
index = sample(nrow(train_i), .7*nrow(train_i))
dev = train_i[-index,]
train_it = train_i[index,]
```

More specifically, we would like to set these predictors to the appropriate DGF. We would like to pick predictors that are similar between the test and training sets (using the previous KS test) and also look back at the conditional plots to see which predictors warrant higher DGF.

```
gam.fit = gam(default ~ . + interest*quality + last_payment*prin_rec, family = "binomial", data = train_

stepGAM = step.Gam(gam.fit, scope = list(
  # For factor variables and variables that have less than two distinct values, include options of not 
  # or removing a dummy variable into the model.
  "reason" = ~1 + reason,
  "n_collect" = ~1 + n_collect,
  "initial_list_status" = ~1 + initial_list_status,
  "recover" = ~1 + recover,
  "coll_fee" = ~1 + coll_fee,
  "term" = ~1 + term,
  "total_acc" = ~1 + total_acc,
  "amount" = ~1 + amount,
  "monthly_payment" = ~1 + monthly_payment,
  "status" = ~1 + status,
  "pymnt_rec" = ~1 + pymnt_rec,
  "quality" = ~1 + quality,
  "violations" = ~1 + violations,
  "del" = ~1 + del,
  "employment" = ~1 + employment,
  # These are the variables that were signnificantly different b/n test and train. Cap their DGFs to 4
  "interest" = ~1 + interest + s(interest),
  "out_prncp_inv" = ~1 + out_prncp_inv + s(out_prncp_inv),
  "req" = ~1 + req + s(req),
  "prin_rec" = ~1 + prin_rec + s(prin_rec),
  "total_cc" = ~1 + total_cc + s(total_cc),
  "out_prncp" = ~1 + out_prncp + s(out_prncp),
  "last_payment" = ~1 + last_payment + s(last_payment),
  "inc" = ~1 + inc + s(inc),
  # For all other variables, consider nonlinear terms of varying degrees of freedom
  "funded" = ~1 + funded + s(funded) + s(funded,10) + s(funded, 20),
  "v1" = ~1 + v1 + s(v1) + s(v1, 10) +  s(v1, 20),
  "int_rec" = ~1 + int_rec + s(int_rec) + s(int_rec, 10) + s(int_rec, 20) ,
  "credit_bal" = ~1 + credit_bal + s(credit_bal)  + s(credit_bal, 10) + s(credit_bal, 20),
  "ncc" = ~1 + ncc + s(ncc)  + s(ncc, 10)  + s(ncc, 20),
```

```r
  "fees_rec" = ~1 + fees_rec + s(fees_rec) + s(fees_rec, 10) + s(fees_rec, 20),
  "credit_ratio" = ~1 + credit_ratio + s(credit_ratio, 10) + s(credit_ratio, 20)),
  direction = "forward",
  trace = FALSE)

stepGAM$formula
```

```
## default ~ reason + n_collect + initial_list_status + recover +
##     coll_fee + term + total_acc + amount + monthly_payment +
##     status + pymnt_rec + quality + violations + del + employment +
##     s(interest) + out_prncp_inv + req + prin_rec + total_cc +
##     s(out_prncp) + last_payment + inc + funded + v1 + s(int_rec) +
##     credit_bal + ncc + s(fees_rec, 10) + credit_ratio + last_payment:prin_rec
```

```r
summary(stepGAM)
```

```
##
## Call: gam(formula = default ~ reason + n_collect + initial_list_status +
##     recover + coll_fee + term + total_acc + amount + monthly_payment +
##     status + pymnt_rec + quality + violations + del + employment +
##     s(interest) + out_prncp_inv + req + prin_rec + total_cc +
##     s(out_prncp) + last_payment + inc + funded + v1 + s(int_rec) +
##     credit_bal + ncc + s(fees_rec, 10) + credit_ratio + last_payment:prin_rec,
##     family = "binomial", data = train_it, trace = FALSE)
## Deviance Residuals:
##        Min         1Q     Median         3Q        Max
## -1.609e+00 -3.776e-01 -1.285e-01  2.107e-08  2.981e+00
##
## (Dispersion Parameter for binomial family taken to be 1)
##
##     Null Deviance: 2681.6 on 2099 degrees of freedom
## Residual Deviance: 735.7481 on 2023.999 degrees of freedom
## AIC: 887.7493
##
## Number of Local Scoring Iterations: NA
##
## Anova for Parametric Effects
##                          Df  Sum Sq Mean Sq F value    Pr(>F)
## reason                   12    8.04  0.6698  1.1673 0.3010944
## n_collect                 1    3.82  3.8226  6.6618 0.0099198 **
## initial_list_status       1    6.28  6.2756 10.9368 0.0009592 ***
## recover                   1    0.00  0.0044  0.0078 0.9298447
## coll_fee                  1    0.00  0.0002  0.0003 0.9854371
## term                      1    3.52  3.5186  6.1320 0.0133568 *
## total_acc                 1    0.27  0.2729  0.4756 0.4904859
## amount                    1    1.80  1.7978  3.1332 0.0768648 .
## monthly_payment           1   13.19 13.1853 22.9788 1.757e-06 ***
## status                    2    1.54  0.7701  1.3421 0.2615167
## pymnt_rec                 1    1.52  1.5199  2.6489 0.1037772
## quality                   6   39.45  6.5747 11.4582 1.225e-12 ***
## violations                1    0.22  0.2191  0.3818 0.5367277
## del                       1    3.04  3.0422  5.3019 0.0214033 *
```

```
## employment              10   13.87  1.3875  2.4181 0.0073782 **
## s(interest)              1    7.32  7.3166 12.7511 0.0003641 ***
## out_prncp_inv            1    0.00  0.0027  0.0047 0.9452533
## req                      1    0.26  0.2602  0.4534 0.5008018
## prin_rec                 1    0.57  0.5657  0.9859 0.3208705
## total_cc                 1    1.10  1.1005  1.9179 0.1662404
## s(out_prncp)             1    0.75  0.7465  1.3009 0.2541786
## last_payment             1    1.43  1.4301  2.4923 0.1145617
## inc                      1    0.11  0.1142  0.1991 0.6555001
## funded                   1    0.20  0.1962  0.3420 0.5587699
## v1                       1    2.69  2.6879  4.6844 0.0305541 *
## s(int_rec)               1   16.41 16.4120 28.6022 9.892e-08 ***
## credit_bal               1    0.92  0.9176  1.5992 0.2061622
## ncc                      1    0.04  0.0424  0.0738 0.7858791
## s(fees_rec, 10)          1    0.48  0.4754  0.8285 0.3628103
## credit_ratio             1    0.08  0.0846  0.1474 0.7011020
## prin_rec:last_payment    1    8.51  8.5055 14.8230 0.0001218 ***
## Residuals             2024 1161.38  0.5738
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Anova for Nonparametric Effects
##                       Npar Df Npar Chisq    P(Chi)
## (Intercept)
## reason
## n_collect
## initial_list_status
## recover
## coll_fee
## term
## total_acc
## amount
## monthly_payment
## status
## pymnt_rec
## quality
## violations
## del
## employment
## s(interest)                 3     8.1807 0.0424211 *
## out_prncp_inv
## req
## prin_rec
## total_cc
## s(out_prncp)                3     9.0061 0.0292145 *
## last_payment
## inc
## funded
## v1
## s(int_rec)                  3    17.9092 0.0004593 ***
## credit_bal
## ncc
## s(fees_rec, 10)             9    15.2158 0.0851888 .
## credit_ratio
```

```
## prin_rec:last_payment
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

It appears a variety of predictors are important in predicting the response, such as reason, initial__list__status, monthly__payment, pymnt__rec, employment, v1, the interaction of prin__rec:last__payment and a nonlinear term of int__rec. Now we want to compute the dev MSE:

```
pred = predict(stepGAM, newdata = dev, type = "response")
class = rep(0, length(pred))
class[pred>.5] = 1
mean(dev$default == class)
```

```
## [1] 0.9133333
```

The performance of this model appears to be the best. We are concerned with the model overfitting to the data so we look at the RSS:

```
class = rep(0, length(stepGAM$fitted.values))
class[stepGAM$fitted.values>.5] = 1
mean(train_it$default == class)
```

```
## [1] 0.9338095
```

Prediction on Test Set

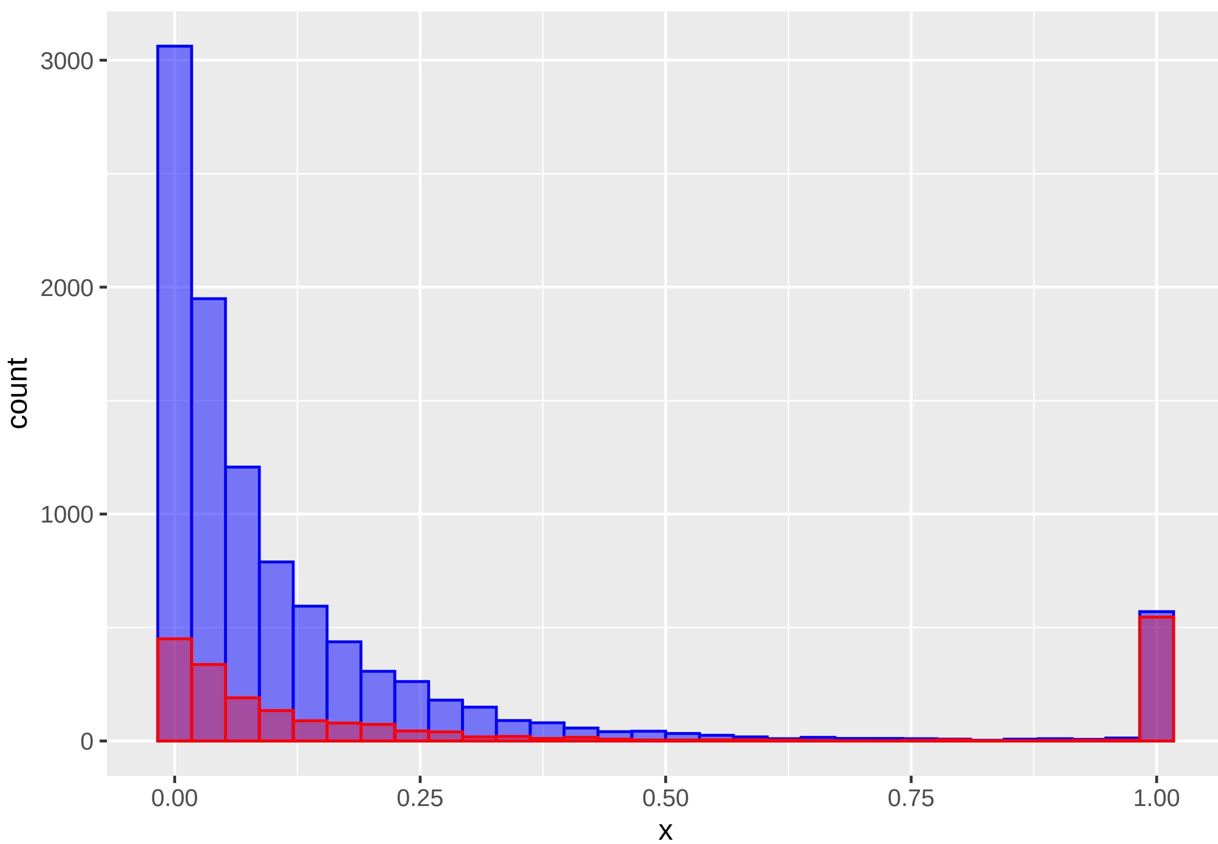
Finally we predict on the test set. We would also like to check to see if the default rate in the test set is roughly 7%:

```
test_preds = predict(stepGAM, newdata = test_i, type = "response")
class = rep(0, length(test_preds))
class[test_preds>.5] = 1
sum(class)/length(class)
```

```
## [1] 0.0753
```

Now view a breakdown of fitted values to the training data compared to predicted values on the test data:

```
test_preds = data.frame(x = test_preds); train_preds = data.frame(x = stepGAM$fitted.values)
ggplot() +
  geom_histogram(data = test_preds, aes(x), stat="bin", bins = 30, col = 'blue', fill = "blue", alpha =
  geom_histogram(data = train_preds, aes(x), stat="bin", bins = 30, col = 'red', fill = "red", alpha =
```



From this output we can see that the proportion of fitted values indicating default/not default is much higher than the ratio of default/not default in the test set. This behavior is expected because the train was a case control sample which forced alot more defaults into the training set.

```r
names(test_preds) = "Predictions"
write.csv(test_preds, "./loan_testy.csv", row.names = FALSE, col.names = FALSE)
```

```
## Warning in write.csv(test_preds, "./loan_testy.csv", row.names = FALSE, :
## attempt to set 'col.names' ignored
```