Default

2023-08-26

Introduction

Commercial banks play a pivotal role in the global financial ecosystem by serving as intermediaries between borrowers and depositors. One of the primary functions of a Bank is to accurately evaluate risk when extending loans to individuals. Modern banks employ statistical learning models to reduce the likelihood of loaning to potential defaulters. Traditionally, credit underwriting involved a manual financial statement analysis and was fraught with limitations. However statistical learning has revolutionized the way people are evaluated for loans. As of September 2023, the profit margin of commercial banks in the U.S. was approximately 26.9%. To ensure these positive profit margins aswell as preventing solvency, being able to predict loan default with accuracy is in the best interest of any loan issuer.

Data : https://www.kaggle.com/datasets/deependraverma13/lending-club-loan-data-analysis-deep-learning

The data set of interest includes 13 predictor variables ranging from FICO score, interest rate, purpose of loan, income to debt ratio, delinquency rate, revolving balance, total monthly installment, and the total days since the credit line has been opened. This data was collected from 2007 to 2015 by a financial service company called Lending Club and is publicly available on kaggle.com. There are 9578 total observations and 13 predictor variables. There is no missing data.

Load in the appropriate libraries

```r
library(readr)
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.2     v purrr     1.0.1
## v forcats   1.0.0     v stringr   1.5.0
## v ggplot2   3.4.2     v tibble    3.2.1
## v lubridate 1.9.2     v tidyr     1.3.0
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```r
library(tidymodels)
```

```
## -- Attaching packages -------------------------------------- tidymodels 1.1.0 --
## v broom        1.0.4     v rsample      1.1.1
## v dials        1.2.0     v tune         1.1.1
## v infer        1.0.4     v workflows    1.1.3
## v modeldata    1.1.0     v workflowsets 1.0.1
## v parsnip      1.1.0     v yardstick    1.2.0
## v recipes      1.0.6
## -- Conflicts ----------------------------------------- tidymodels_conflicts() --
## x scales::discard() masks purrr::discard()
## x dplyr::filter()   masks stats::filter()
## x recipes::fixed()  masks stringr::fixed()
## x dplyr::lag()      masks stats::lag()
## x yardstick::spec() masks readr::spec()
## x recipes::step()   masks stats::step()
## * Use suppressPackageStartupMessages() to eliminate package startup messages
```

```r
library(dplyr)
library(corrplot)
```

```
## corrplot 0.92 loaded
```

```r
library(ggplot2)
library(kableExtra)
```

```
##
## Attaching package: 'kableExtra'
##
## The following object is masked from 'package:dplyr':
##
##     group_rows
```

```r
library(kknn)
library(yardstick)
library(recipes)
library(tune)
library(workflowsets)
library(themis)
library(ggridges)
library(xgboost)
```

```
##
## Attaching package: 'xgboost'
##
## The following object is masked from 'package:dplyr':
##
##     slice
```

```r
library(probably)
```

```
##
## Attaching package: 'probably'
##
## The following objects are masked from 'package:base':
##
##     as.factor, as.ordered
```

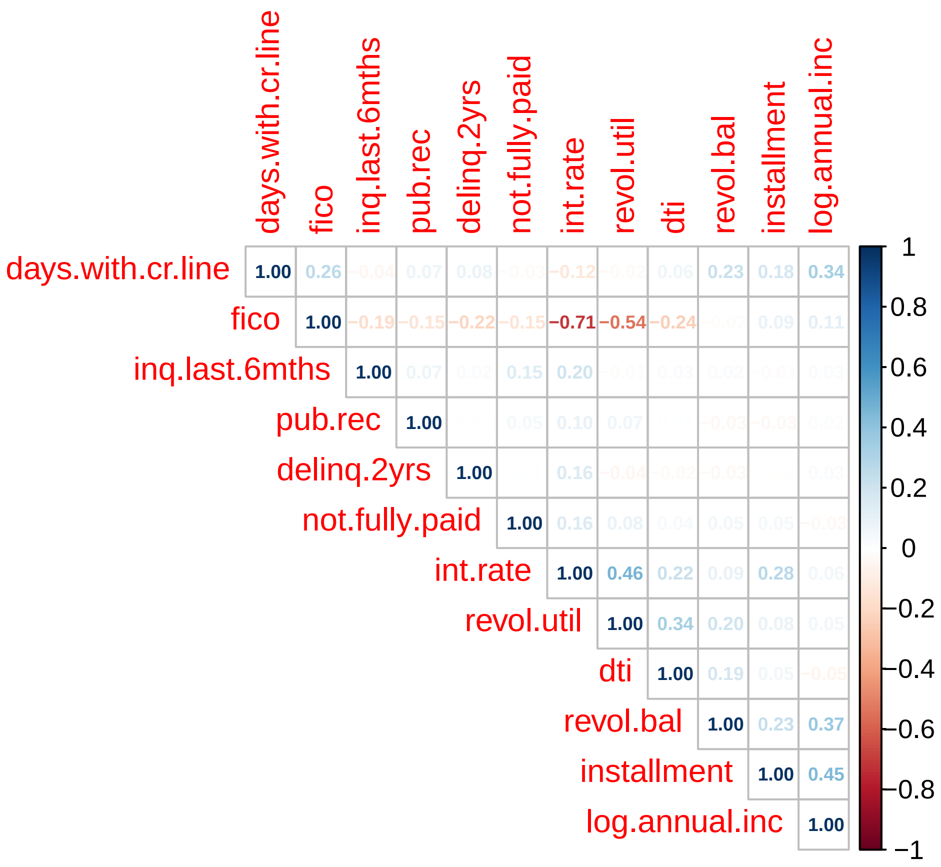
Load in the Data

```r
loan_data <- read_csv("loan_data.csv", show_col_types = FALSE)
#omit credit.policy column its not necessary
loan_data <- subset(loan_data, select = - credit.policy)
```

Exploratory Data Analysis

Correlation Matrix

```r
loan_data_for_corr <- mutate(loan_data, not.fully.paid = as.integer(not.fully.paid))


short <- loan_data_for_corr %>% select(int.rate,installment,log.annual.inc,
                              dti,fico,days.with.cr.line,revol.bal,revol.util,inq.last.6mths,
                              delinq.2yrs,pub.rec, not.fully.paid)
short.cor = cor(short)
corrplot(short.cor, method = 'number', order = 'AOE', type = 'upper', number.cex =0.6)
```



The correlation plot exposes a negative linearity between FICO score and Interest rates. This is not surprising as a perk of having a higher FICO score is lenders offer lower interest rates since you have a record of paying things off on time.

A interesting discovery is that FICO score and revolving unpaid balance at the end of the month are negatively correlated. This means that the higher the FICO score of a borrower the less balance they have left over at the end of the month. One would expect those with higher FICO scores to have higher balances as they have access to larger credit lines but that is not the case.

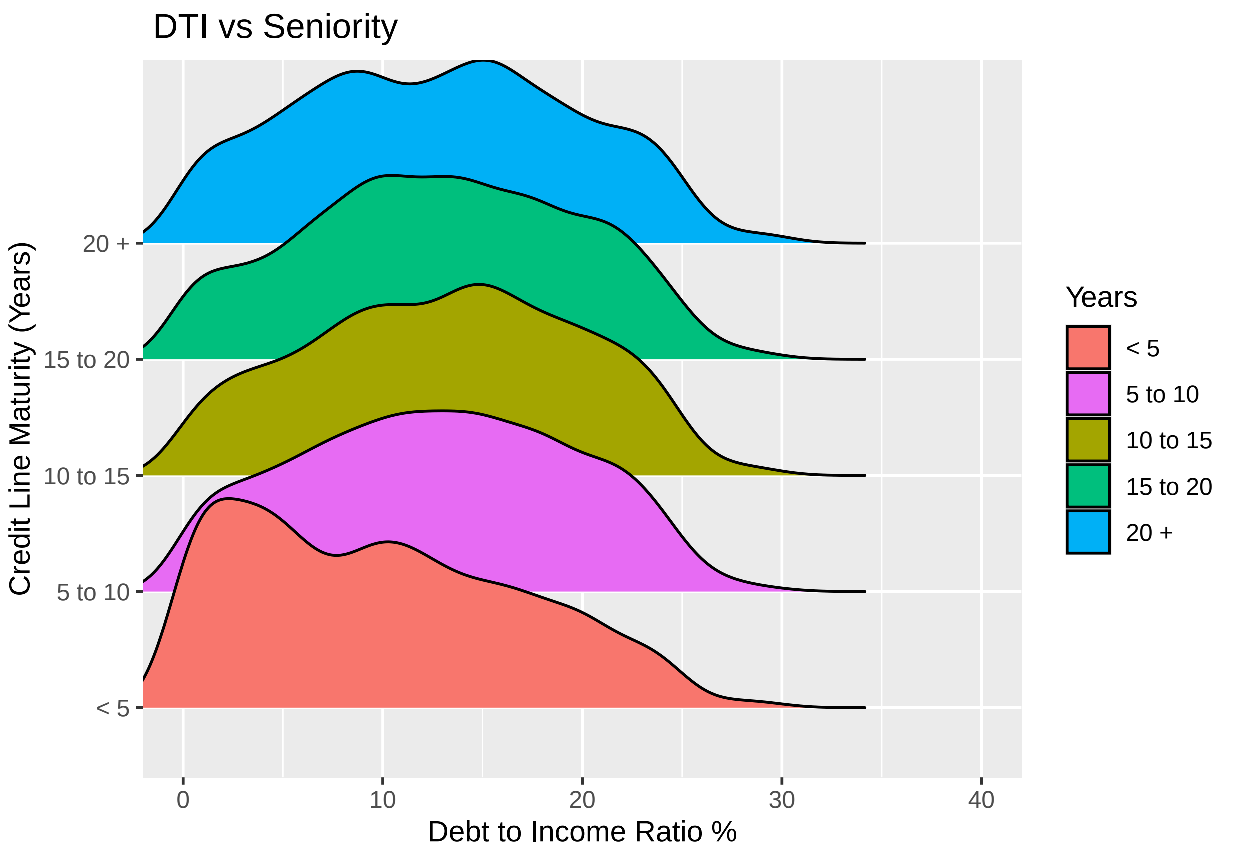
While multicollinearity is cause for concern in descriptive models, in a setting where probability of default is the main concern we are less interested in employing those kinds of models and instead choose non parametric predictive models.

Density ridge line plots

```
loan_data_density <- loan_data

loan_data_density <- loan_data_density %>%
  mutate(classification = case_when(days.with.cr.line < 1825 ~ '< 5',
                                days.with.cr.line > 1825 & days.with.cr.line < 3650 ~ '5 to 10',
                                       days.with.cr.line > 3650   & days.with.cr.line < 5475 ~ '10 to
                                               days.with.cr.line> 5475  & days.with.cr.line < 7300 ~
                                                      days.with.cr.line > 7300 ~ '20 +'))
```

```r
custom_order <- c('< 5', '5 to 10', '10 to 15', '15 to 20', '20 +')

loan_data_density$class_ordered <- factor(loan_data_density$classification, levels = custom_order)




ggplot(loan_data_density, aes(x = dti, y = class_ordered)) +
  geom_density_ridges(aes(fill = classification)) +
  scale_fill_manual(values = c('#FFB1AF','#FFDFD3','#FFEEA5','#FFCBA5','#C8F69B'))+
  scale_fill_discrete(breaks = custom_order) +
  coord_cartesian(xlim = c(0, 40)) +
  xlab('Debt to Income Ratio %') +
  ylab('Credit Line Maturity (Years)') +
  ggtitle(' DTI vs Seniority') +
  labs(fill = 'Years')
```

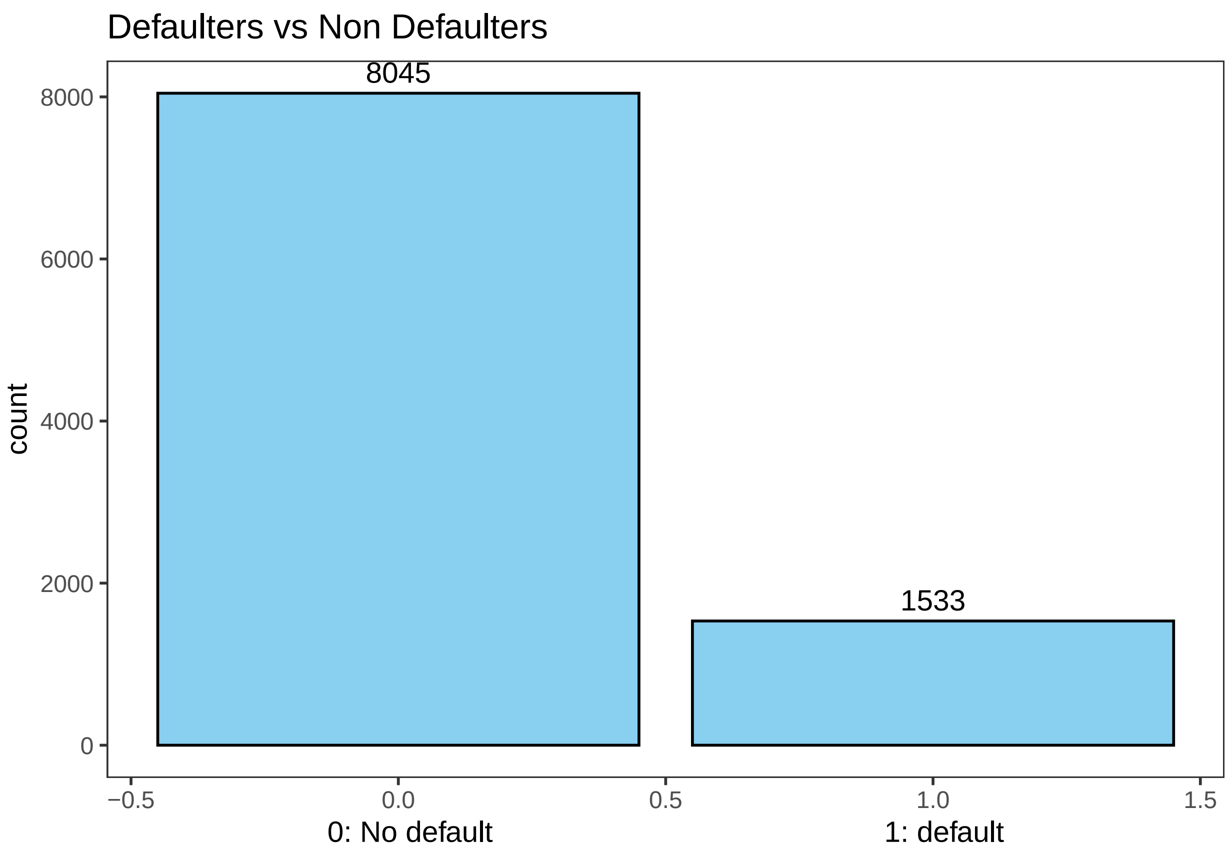
```
## Scale for fill is already present.
## Adding another scale for fill, which will replace the existing scale.
## Picking joint bandwidth of 1.4
```



With a density ridge line plot we can observe whether the years someone has had a credit line influence the amount of debt they take on. Interestingly it seems as someone has had a credit line for a longer period of time they are shown to have more debt relative to their income. Amongst the 9500 borrowers in this data set it looks like most have a DTI(debt to income ratio) of less than 20%.

Bar Chart

```r
ggplot(loan_data,aes(x=not.fully.paid), color = '#89CFF0') +
  geom_bar(fill = '#89CFF0', color = 'black') +
  xlab('0: No default                                         1: default') +
  ggtitle('Defaulters vs Non Defaulters') +
  geom_text(stat = "count", aes(label = after_stat(count)), vjust = -0.5)+
  theme_test()
```



Naturally due to the consequences of not paying back loans most people will pay back their loans. This is great news for the bank but unfortunate news for anyone trying to build predictive models. Defaulters make up about 16% of those who take out debts in the sample. This is an imbalanced data set that could result in inaccurate prediction rates. This will be taken to account and up sampling will be implemented into the recipe to make sure the models are trained on a higher proportion of defaulters.

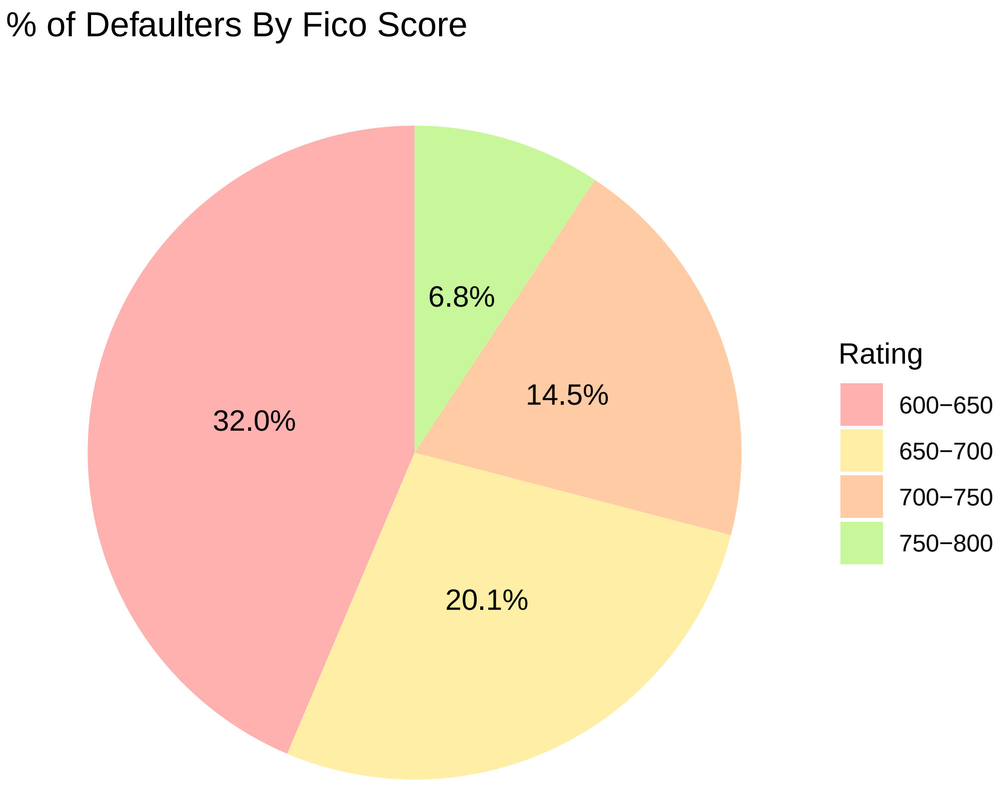
Pie Chart

```r
#spliting the credit scores into 4 ranges

#600-650
total1 <- sum(loan_data$not.fully.paid[loan_data$fico > 600 & loan_data$fico < 650])
count1 <- sum(loan_data$fico > 600 & loan_data$fico < 650)
low = total1/count1

#650 - 700
total2 <- sum(loan_data$not.fully.paid[loan_data$fico > 650 & loan_data$fico < 700])
count2 <- sum(loan_data$fico > 650 & loan_data$fico < 700)
fair = total2/count2

#700 - 750
```

```r
total3 <- sum(loan_data$not.fully.paid[loan_data$fico > 700 & loan_data$fico < 750])
count3 <- sum(loan_data$fico > 700 & loan_data$fico < 750)
better = total3/count3

#750-800
total4 <- sum(loan_data$not.fully.paid[loan_data$fico > 750 & loan_data$fico < 800])
count4 <- sum(loan_data$fico > 750 ) #& loan_data$fico < 800)
best = total4/count4
```

```r
Rating <- c('600-650','650-700','700-750','750-800')
Default_rate <- c(low,fair,better,best)
RDR <- data.frame(Rating,Default_rate)
```

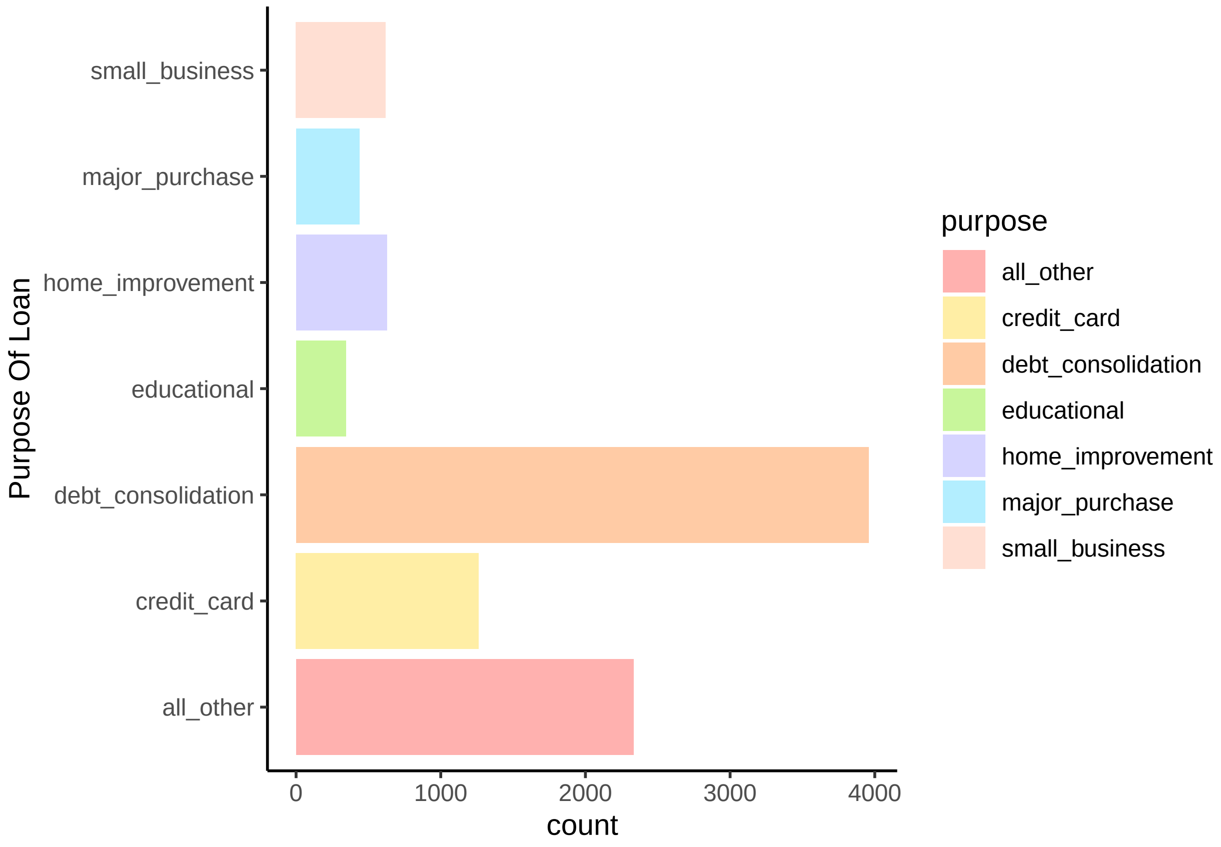
```r
pie_plot <- ggplot(RDR, aes(x = "", y = Default_rate, fill = Rating)) +
  geom_bar(stat = "identity", width = 1) +
  coord_polar(theta = "y") +
  theme_void() +
  labs(title = "% of Defaulters By Fico Score")+
  scale_fill_manual(values=c('#FFB1AF','#FFEEA5','#FFCBA5','#C8F69B'))+
  geom_text(aes(label = scales::percent(Default_rate)), position = position_stack(vjust = 0.5))+
  theme(legend.position = "right")

print(pie_plot)
```



A FICO score in itself is a 3 digit number based on an analysis of someones credit files. Upon analyzing the distribution of defaulters we can see Fico is a powerful indicator of whether someone is a defaulter. 1/3 of the defaulters come from the 600-650 range while only 6.8% comes from the 750-800 range. FICO score must be a good predictor for whether or not someone pays back their loan.

Loan Purpose Profile

```r
ggplot(data = loan_data,aes(x = purpose, fill = purpose)) +
  geom_bar() +
  coord_flip() +
  scale_fill_manual(values=c('#FFB1AF','#FFEEA5','#FFCBA5','#C8F69B','#D6D4FF','#B3EEFF','#FFDFD3')) +
  xlab('Purpose Of Loan') +
  theme_classic()
```



A majority of the loans are debt consolidation which is taking out one new loan to replace all your old loans. Not necessarily a different loan but just an aggregate. After that you have other loans and then other credit card debt.

Data Splitting and Cross-Validation

Splitting the data

```r
loan_data$not.fully.paid <- as.factor(loan_data$not.fully.paid)

loan_split <- initial_split(loan_data, prop = 0.75, strata = not.fully.paid)

loan_train <- training(loan_split)

loan_test <- testing(loan_split)
```

Data is split into a testing and training split to ensure that we have a test set that our model has not seen and a training set to which the model is trained on. This allows us to emulate real data points to see how the model would perform in a real world setting.

Creating k folds for cross validation

```r
loan_folds <- vfold_cv(loan_train, v = 5)
loan_folds
```

```
## #  5-fold cross-validation
## # A tibble: 5 x 2
##   splits              id
##   <list>              <chr>
## 1 <split [5745/1437]> Fold1
## 2 <split [5745/1437]> Fold2
## 3 <split [5746/1436]> Fold3
## 4 <split [5746/1436]> Fold4
## 5 <split [5746/1436]> Fold5
```

K-folds cross validation is a way to see how your model would perform on a test set without really testing it on a test set. This is done by splitting the training set once again into a validation set. K is the number of ways you split the training set into subgroups. Since we have a training set of 7662 observations it will be split in 5 different ways of approximately 1532 different data points in each fold. Then the model is trained on the 4/5th of the data points and tested on the remaining 1/5th which is the validation set. You repeat this K times and it gives you more accurate figures than testing metrics based off of your test set.

Recipe

Before data can be used to train a model it must go through pre-processing. The tidyverse package utilizes a function called recipe for this purpose.

Up sampling is a critical method to building a model that predicting loan default with greater accuracy . Since the bank is more concerned with who will not be able to pay rather than pay, It would be advantageous to increase the amount of data points that default to half as many who dont default.

Cleaned data set so there is no missing values. No need to impute any values.

Nominal predictors such as purpose of loan was transformed into dummy variables.

Normalized all numeric variables excluding binary indicators, this allows for stability of the model and makes sure its consistent among predictors.

```r
loan_recipe <- recipe(not.fully.paid ~ int.rate + installment + log.annual.inc + dti + fico +
                 days.with.cr.line + revol.bal + revol.util + inq.last.6mths +
                 delinq.2yrs + pub.rec + purpose,  data = loan_train) %>%
  step_dummy(all_nominal_predictors()) %>%
  step_upsample(not.fully.paid, over_ratio = 0.6, skip = TRUE) %>%
  step_normalize( int.rate, installment, log.annual.inc, dti, fico,
                 days.with.cr.line, revol.bal, revol.util, inq.last.6mths,
                 delinq.2yrs, pub.rec)

prep(loan_recipe) %>%
  bake(new_data = loan_train) %>%
  head() %>%
  kable() %>%
  kable_styling(full_width = F) %>%
  scroll_box(width = '100%', height = '200px')
```

K-Nearest Neighbors

```r
knn_model <- nearest_neighbor(neighbors = tune()) %>%
  set_engine('kknn') %>%
```

int.rate	installment	log.annual.inc	dti	fico	days.with.cr.line	revol.bal	revol.util	inq.las
-0.6795155	-0.4569386	0.2538856	0.2360025	-0.0138197	-0.7155282	0.3804033	0.9839492	-0.
0.3825359	0.1960969	-0.8692307	-0.1471793	-0.6754633	0.0591495	-0.3499795	-0.7595790	-0.
-0.9134639	-0.7672529	0.6790476	-0.6556875	0.1185091	-0.7393810	0.3814705	0.8645295	-0.
0.6387651	-1.0471388	0.5987357	0.3339588	-1.0724494	-0.1966928	-0.3201694	-0.2853120	-0.
-1.7304265	-0.9425231	1.5579516	0.6235059	0.5154952	0.6193176	0.7972105	0.1070671	-0.
-0.4455671	-1.1212317	0.7696359	0.6624003	-0.6754633	-0.2272826	1.2605397	0.1104791	-0.

```
  set_mode('classification')

knn_wflow <- workflow() %>%
  add_model(knn_model) %>%
  add_recipe(loan_recipe)
```

K-nearest Neighbors works based off a hyper parameter called neighbors. Neighbors is the size of the surrounding observations around the Data point for which a prediction is to be generated. In a classification setting, the predicted outcome will be what has the highest observed probability of occurring within this K size group of neighbors.

Since K is a hyper parameter, it can be tuned to return the best metric desired on the folds. Since the goal of the project is to supply banks with a model that has a strong ability to distinguish between defaulters and non defaulters, the highest possible roc_auc is the goal.

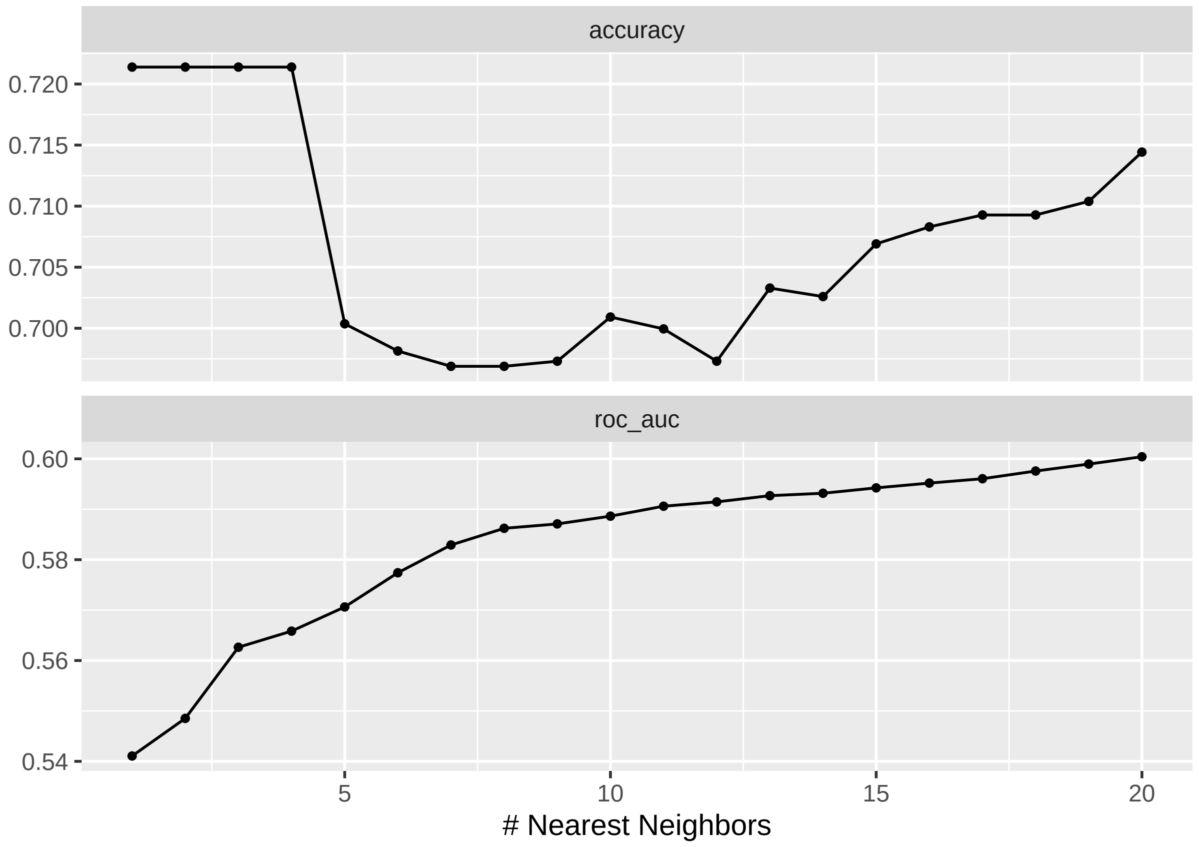
```
neighbors_grid <- grid_regular(neighbors(range = c(1, 20)), levels = 20)
neighbors_grid
```

```
## # A tibble: 20 x 1
##    neighbors
##        <int>
##  1         1
##  2         2
##  3         3
##  4         4
##  5         5
##  6         6
##  7         7
##  8         8
##  9         9
## 10        10
## 11        11
## 12        12
## 13        13
## 14        14
## 15        15
## 16        16
## 17        17
## 18        18
## 19        19
## 20        20
```

Here we will fit models ranging from K = 1 to K = 20.

```
tune_knn <- tune_grid(
  object = knn_wflow,
  resamples = loan_folds,
  grid = neighbors_grid,
  control = control_grid(verbose = TRUE)
)
```

```
#write_rds(tune_knn, file = "tune_knn.rds")

tune_knn <- read_rds("tune_knn.rds")



autoplot(tune_knn)
```



The roc_auc is a metric of the model, the closer it is to 1 the better that the model is able to distinguish between a defaulter and not a defaulter. Therefore we will tune to highest roc_auc level.

```
collect_metrics(tune_knn)
```

```
## # A tibble: 40 x 7
##    neighbors .metric  .estimator  mean     n std_err .config
##        <int> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
##  1         1 accuracy binary     0.721     5 0.00562 Preprocessor1_Model01
##  2         1 roc_auc  binary     0.541     5 0.00491 Preprocessor1_Model01
##  3         2 accuracy binary     0.721     5 0.00562 Preprocessor1_Model02
##  4         2 roc_auc  binary     0.549     5 0.00561 Preprocessor1_Model02
##  5         3 accuracy binary     0.721     5 0.00562 Preprocessor1_Model03
##  6         3 roc_auc  binary     0.563     5 0.00679 Preprocessor1_Model03
##  7         4 accuracy binary     0.721     5 0.00562 Preprocessor1_Model04
##  8         4 roc_auc  binary     0.566     5 0.00851 Preprocessor1_Model04
##  9         5 accuracy binary     0.700     5 0.00614 Preprocessor1_Model05
## 10         5 roc_auc  binary     0.571     5 0.00693 Preprocessor1_Model05
## # i 30 more rows
```

Here is the results of the hyper parameter tuning.

```
knn_metrics <- collect_metrics(tune_knn)

best_knn <- select_by_one_std_err(tune_knn,
                          metric = "roc_auc",
                          neighbors)
best_knn
```

```
## # A tibble: 1 x 9
##   neighbors .metric .estimator  mean     n std_err .config           .best .bound
##       <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>             <dbl>  <dbl>
## 1         9 roc_auc binary     0.587     5 0.00893 Preprocessor1_M~ 0.600  0.587
```

Out of all the KNN models chosen, the model with the best combination of tuning values is chosen. The best K is 9 with an roc_auc of 0.587

Elastic Net

Workflow

```
elastic_loan_model <- logistic_reg(mixture = tune(),  penalty = tune()) %>%
  set_mode("classification") %>%
  set_engine("glmnet")

elastic_loan_wflow <- workflow() %>%
  add_model(elastic_loan_model) %>%
  add_recipe(loan_recipe)
```

```
en_grid <- grid_regular(penalty(range = c(0, 1),
                        trans = identity_trans()),
                        mixture(range = c(0, 1)),
                        levels = 10)
```

In an elastic net model, the first parameter tuned is penalty. Penalty shrinks the coefficients of the Betas when calculating the sum of squares residuals.

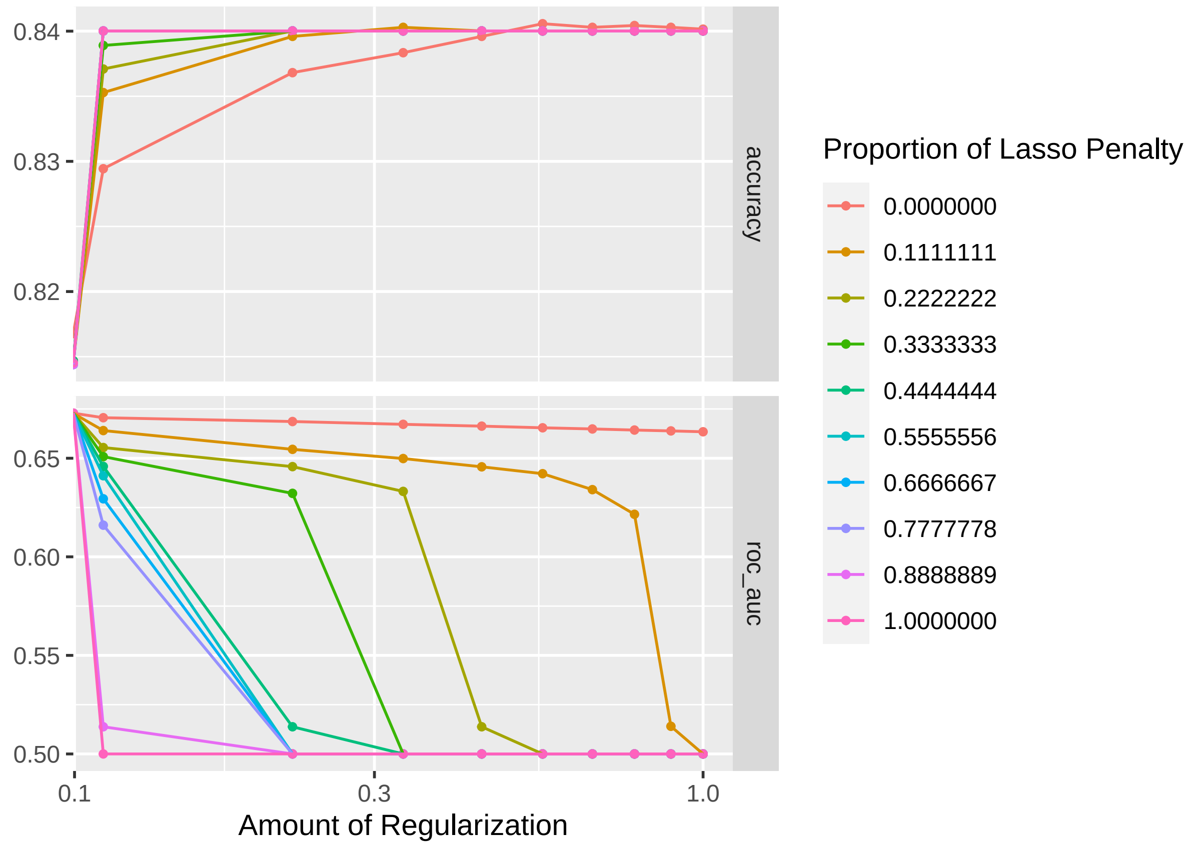
The second parameter tuned is mixture, which allows for the fitting of both ridge and lasso regression models. 0 being a ridge model and 1 being a lasso model. Lasso shrinks based of the absolute values of the Beta coefficients while Ridge regression shrinks based off of the beta coefficients squared.

```
tune_EN_loan <- tune_grid(
  elastic_loan_wflow,
  resamples = loan_folds,
  grid = en_grid,
  control = control_grid(verbose = TRUE)
)

tune_EN_loan
```

```
#write_rds(tune_EN_loan, file = "tune_EN.rds")

tune_EN_loan <- read_rds("tune_EN.rds")
```

```r
autoplot(tune_EN_loan)
```

```
## Warning: Transformation introduced infinite values in continuous x-axis
## Transformation introduced infinite values in continuous x-axis
```



```r
collect_metrics(tune_EN_loan)
```

```
## # A tibble: 200 x 8
##    penalty mixture .metric  .estimator  mean     n  std_err .config
##      <dbl>   <dbl> <chr>    <chr>      <dbl> <int>    <dbl> <chr>
##  1   0           0 accuracy binary     0.817     5 0.00281  Preprocessor1_Model~
##  2   0           0 roc_auc  binary     0.673     5 0.0111   Preprocessor1_Model~
##  3   0.111       0 accuracy binary     0.829     5 0.00165  Preprocessor1_Model~
##  4   0.111       0 roc_auc  binary     0.671     5 0.0122   Preprocessor1_Model~
##  5   0.222       0 accuracy binary     0.837     5 0.000924 Preprocessor1_Model~
##  6   0.222       0 roc_auc  binary     0.669     5 0.0124   Preprocessor1_Model~
##  7   0.333       0 accuracy binary     0.838     5 0.000908 Preprocessor1_Model~
##  8   0.333       0 roc_auc  binary     0.667     5 0.0124   Preprocessor1_Model~
##  9   0.444       0 accuracy binary     0.840     5 0.00133  Preprocessor1_Model~
## 10   0.444       0 roc_auc  binary     0.666     5 0.0124   Preprocessor1_Model~
## # i 190 more rows
```

By displaying the model's metrics we can see how it has performed across each level of penalty and mixture.

```r
en_metrics_loan <- collect_metrics(tune_EN_loan)
```

```
best_en <- select_by_one_std_err(tune_EN_loan,
                          metric = "roc_auc",
                          penalty,
                          mixture
                         )
best_en
```

```
## # A tibble: 1 x 10
##   penalty mixture .metric .estimator  mean     n std_err .config    .best .bound
##     <dbl>   <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>      <dbl>  <dbl>
## 1       0       0 roc_auc binary     0.673     5  0.0111 Preproces~ 0.673  0.662
```

The best model was a ridge model with no penalty, it has an roc_auc of .667 The closer to 1 this figure is means it can distinguish between defaulters and non defaulters with 100% accuracy. Since 0.5 is random chance the model performs with some predictive capability.

Pruned Decision Trees

```
tree_mod <- rand_forest(mtry = tune(),  trees = tune(), min_n = tune()) %>%
  set_engine("ranger") %>% #could be rpart aswell
  set_mode("classification")

tree_wf <- workflow() %>%
  add_model(tree_mod) %>%
  add_recipe(loan_recipe)
```

```
param_grid <- grid_regular(mtry(range = c(1,10)),
                           trees(range = c(50, 100)),
                           min_n(range = c(3,10)),
                           levels = 8)

tune_tree <- tune_grid(
  tree_wf,
  resamples = loan_folds,
  grid = param_grid,
  control = control_grid(verbose = TRUE)
)
```

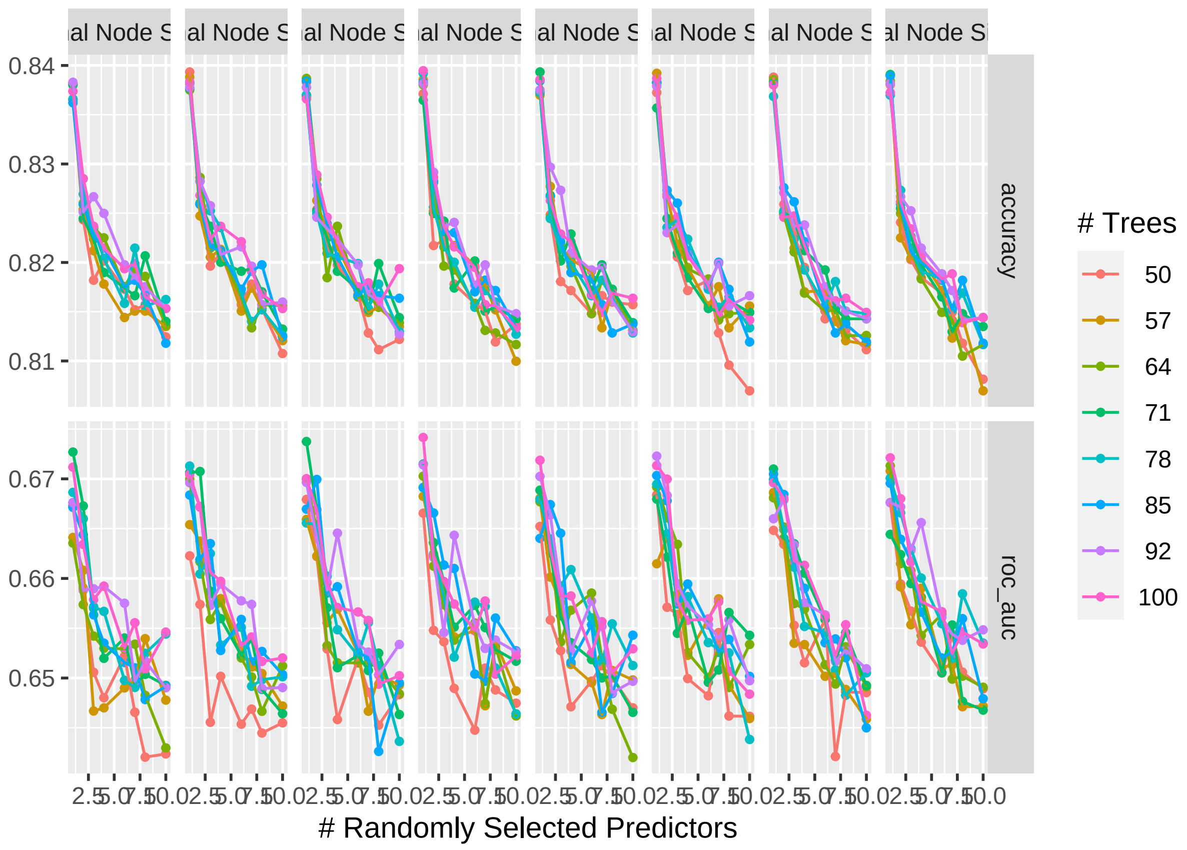
A decision tree is tuned by 3 parameters

mtry - which is the amount of predictors it considers at each split

trees - which is the amount of trees to make

min_n - the minimum number of observation in a split before a split is constructed

```
#write_rds(tune_tree, file = "tune_tree.rds")

tune_tree <- read_rds("tune_tree.rds")



autoplot(tune_tree)
```



```r
best_tree <- select_by_one_std_err(tune_tree, metric = "roc_auc",
                                    mtry, min_n, trees)

best_tree
```

```
## # A tibble: 1 x 11
##    mtry trees min_n .metric .estimator  mean     n std_err .config  .best .bound
##   <int> <int> <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>    <dbl>  <dbl>
## 1     1    71     3 roc_auc binary     0.673     5 0.00469 Preproc~ 0.674  0.667
```

The best prune tree model on the folds was a one that only considered 1 predictor at each split, it has 71 different trees and stopped created splits when they had less than 3 observations. It had an roc_auc of 0.672 proving to have some predictive power.

Gradient Boosted Trees

```r
boosted_loan <- boost_tree(mtry = tune(),
                           trees = tune(),
                           learn_rate = tune()) %>%
  set_engine("xgboost") %>%
  set_mode("classification")

boosted_loan_wf <- workflow() %>%
  add_model(boosted_loan) %>%
  add_recipe(loan_recipe)
```

```r
bt_grid <- grid_regular(mtry(range = c(1, 10)),
                        trees(range = c(200, 600)),
                        learn_rate(range = c(-10, -1)),
                        levels = 5)
bt_grid
```

```
## # A tibble: 125 x 3
##     mtry trees   learn_rate
##    <int> <int>        <dbl>
##  1     1   200 0.0000000001
##  2     3   200 0.0000000001
##  3     5   200 0.0000000001
##  4     7   200 0.0000000001
##  5    10   200 0.0000000001
##  6     1   300 0.0000000001
##  7     3   300 0.0000000001
##  8     5   300 0.0000000001
##  9     7   300 0.0000000001
## 10    10   300 0.0000000001
## # i 115 more rows
```

Boosted trees have 3 hyper parameters

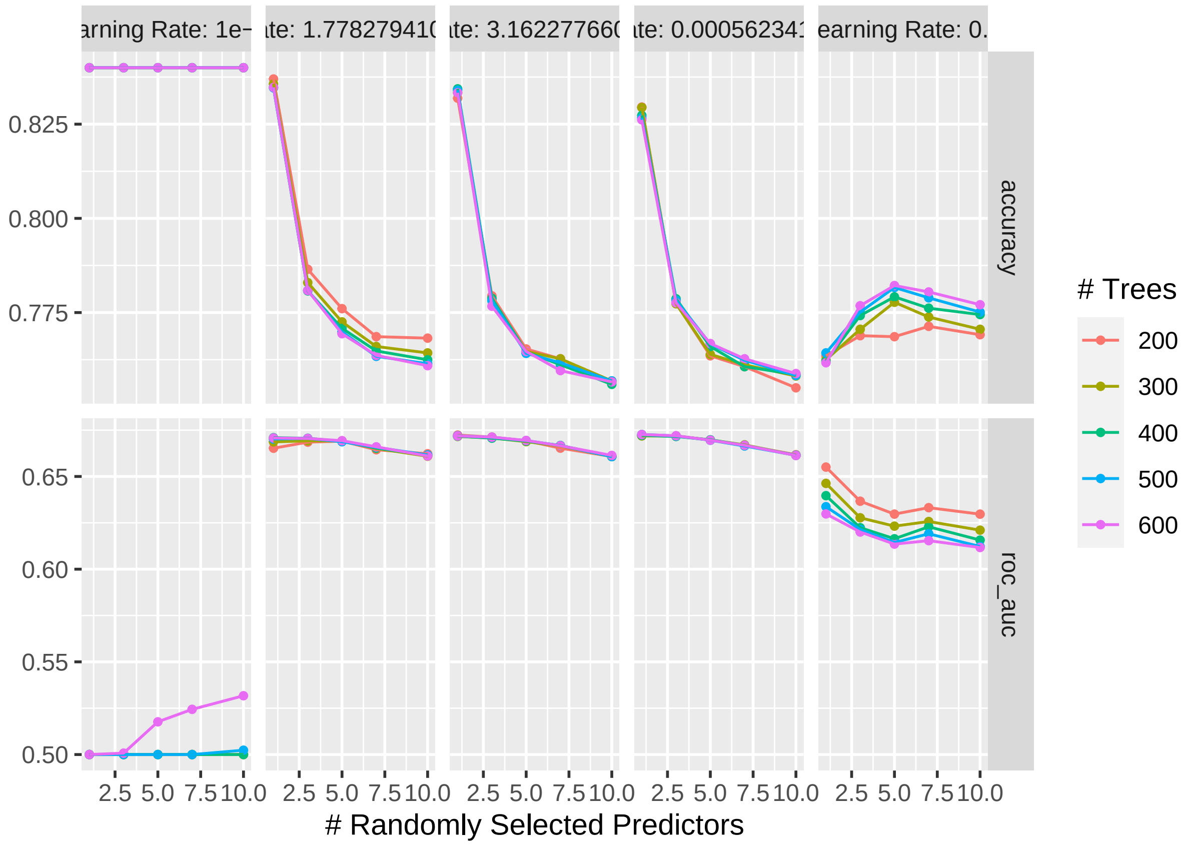
mtry, the number of predictors to consider at each split of the tree

trees, the amount of decision trees to be created in the Forrest

learning_rate, the shrinkage parameter lambda determines how much of an effect each individual tree has on the overall model.

```
tune_bt_loan <- tune_grid(
  boosted_loan_wf,
  resamples = loan_folds,
  grid = bt_grid,
  control = control_grid(verbose = TRUE)
)
```

```
#write_rds(tune_bt_loan, file = "boostT_loan.rds")

tune_bt_loan <- read_rds("boostT_loan.rds")

autoplot(tune_bt_loan)
```



```r
best_boosted <- select_by_one_std_err(tune_bt_loan,
                          metric = "roc_auc",
                          mtry,
                          trees,
                          learn_rate)

best_boosted
```

```
## # A tibble: 1 x 11
##    mtry trees learn_rate .metric .estimator  mean     n std_err .config    .best
##   <int> <int>      <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>      <dbl>
## 1     1   200 0.00000316 roc_auc binary     0.672     5 0.00658 Preproces~ 0.673
## # i 1 more variable: .bound <dbl>
```

The best performing boosted tree had an mtry of 1, 200 trees and a learning rate of 0.0000036 which is a relatively small value. The overall roc_auc was 0.672

Which Model had the best ROC_AUC?

```r
Model_fold_roc_auc <- c(best_boosted$mean,
                        best_en$mean,
                        best_knn$mean,
                        best_tree$mean)
models <- c('Boosted Trees', 'Elastic Net', 'KNN', 'Pruned Trees')

results <- tibble(ROC_AUC = Model_fold_roc_auc, models = models)
results %>%
  arrange(-ROC_AUC)
```

```
## # A tibble: 4 x 2
##   ROC_AUC models
##     <dbl> <chr>
## 1   0.673 Elastic Net
## 2   0.673 Pruned Trees
## 3   0.672 Boosted Trees
## 4   0.587 KNN
```

Since the roc_auc was the best among the folds at 0.68 we will fit this to the test data. It is the best at distinguishing between defaulters and non defaulters.

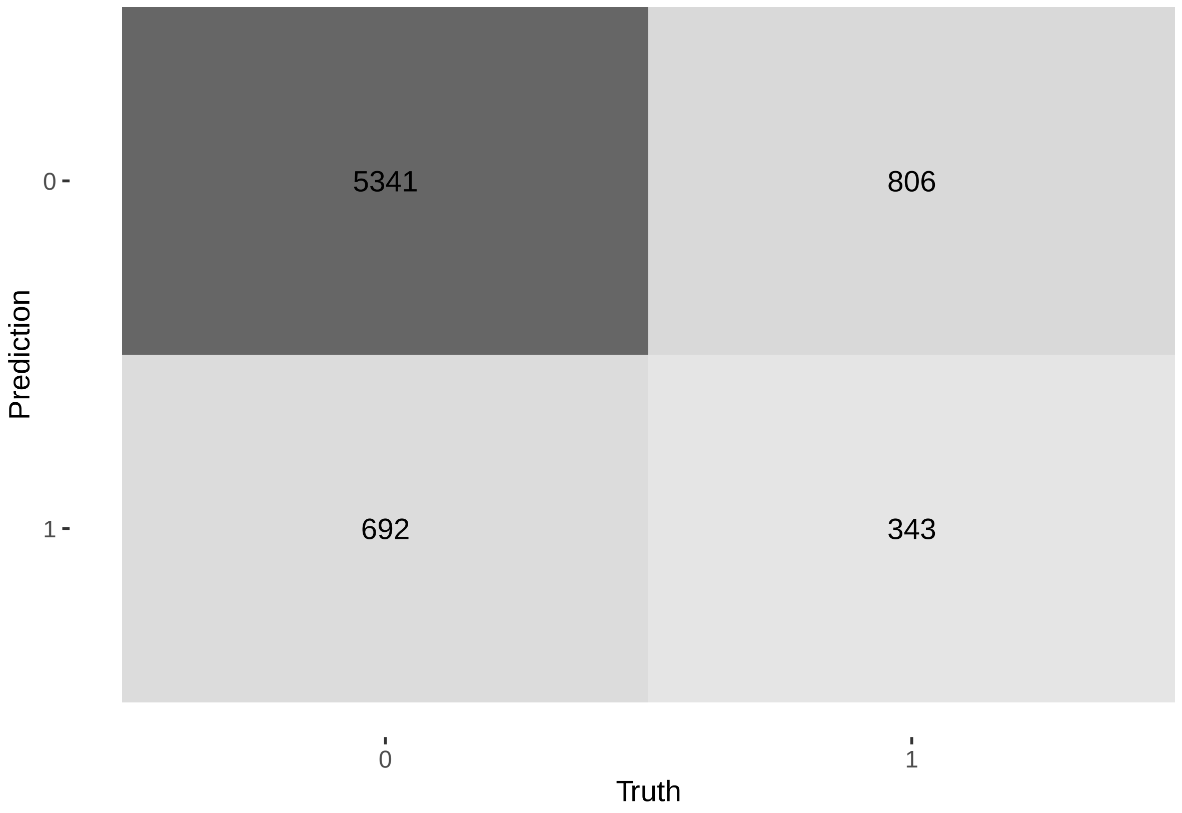
Finalize Elastic Net

```
EN_final <- finalize_workflow(elastic_loan_wflow, best_en)

EN_final <- fit(EN_final, data = loan_train)
```

The Elastic net model has now been trained on all the training data.

Model Performance on Training Set

```
final_en_model_train <- augment(EN_final, new_data = loan_train) %>%
  select(not.fully.paid, starts_with('.pred'))

conf_mat(final_en_model_train, truth = not.fully.paid,
         .pred_class) %>%
  autoplot(type = "heatmap")
```



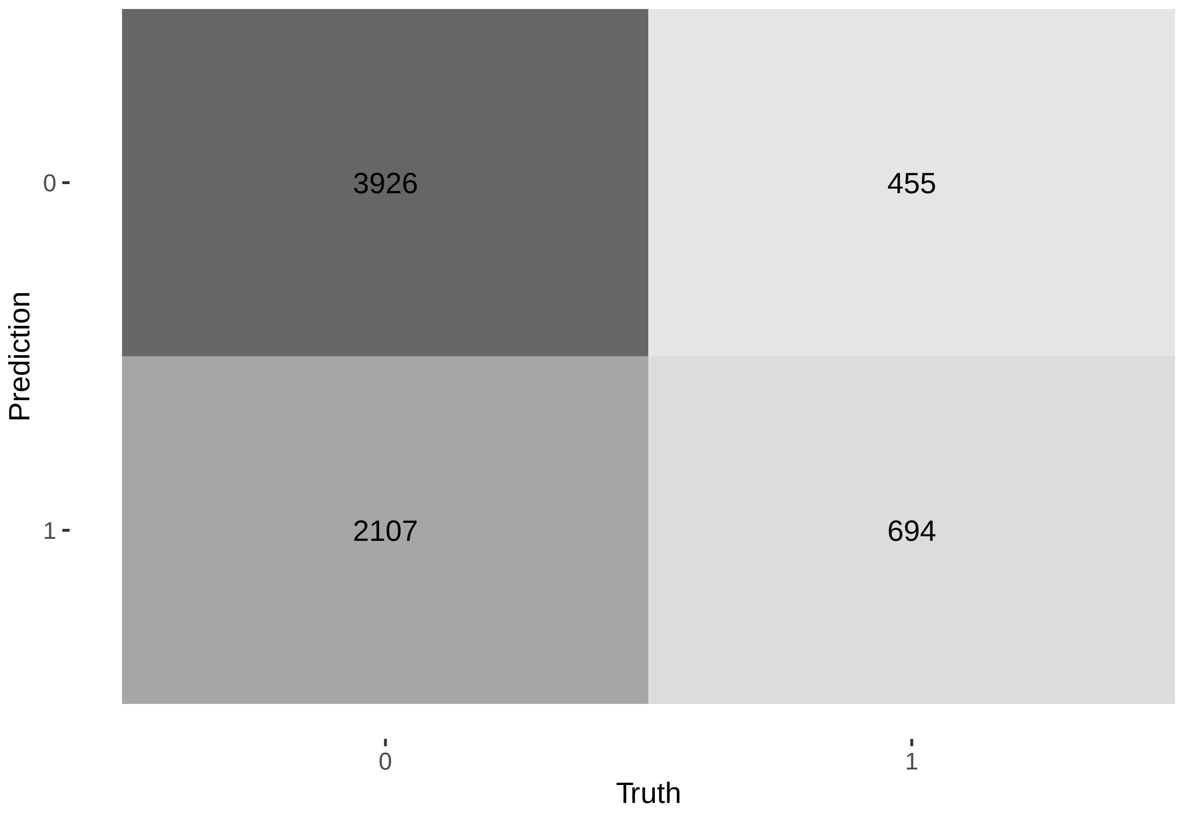
Based on the training set, the Elastic net model is under predicting the positive class of defaulters. Sensitivity is only at 0.25 This can be fixed with a threshold adjustment. After all Banks are most concerned with catching those who will not pay. It makes sense to predict a default in some cases if the probability is a little lower than 0.50.

Threshold on training

```r
threshold <- 0.38  # Adjust the threshold as needed

# Create a new column with adjusted predictions based on the threshold
df <- final_en_model_train %>%
  mutate(adjusted_prediction = ifelse(.pred_1 >= threshold, "1", "0"))

df
```

```
## # A tibble: 7,182 x 5
##    not.fully.paid .pred_class .pred_0 .pred_1 adjusted_prediction
##    <fct>          <fct>         <dbl>   <dbl> <chr>
##  1 0              0             0.775   0.225 0
##  2 0              0             0.603   0.397 1
##  3 0              0             0.758   0.242 0
##  4 0              0             0.804   0.196 0
##  5 0              0             0.870   0.130 0
##  6 0              0             0.607   0.393 1
##  7 0              0             0.722   0.278 0
##  8 0              0             0.732   0.268 0
##  9 0              0             0.815   0.185 0
## 10 0              0             0.611   0.389 1
## # i 7,172 more rows
```

```r
df$adjusted_prediction <- factor(df$adjusted_prediction, levels = levels(df$not.fully.paid))


conf_mat(df, truth = not.fully.paid,
         adjusted_prediction) %>%
  autoplot(type = "heatmap")
```



After expirimenting with different threshold values we found that if we classify any given person with a probability of defaulting with 0.38 or higher we obtain a predictive accuracy of the positive class of 58%. The result is 8% better than a random guess. This is at the trade off given that now the overall error rate is at 36%. This may be acceptable as its better to classify a defaulter as a defaulter rather than incorrectly classifying a defaulter as a non defaulter.

Implementing Model on Test Set

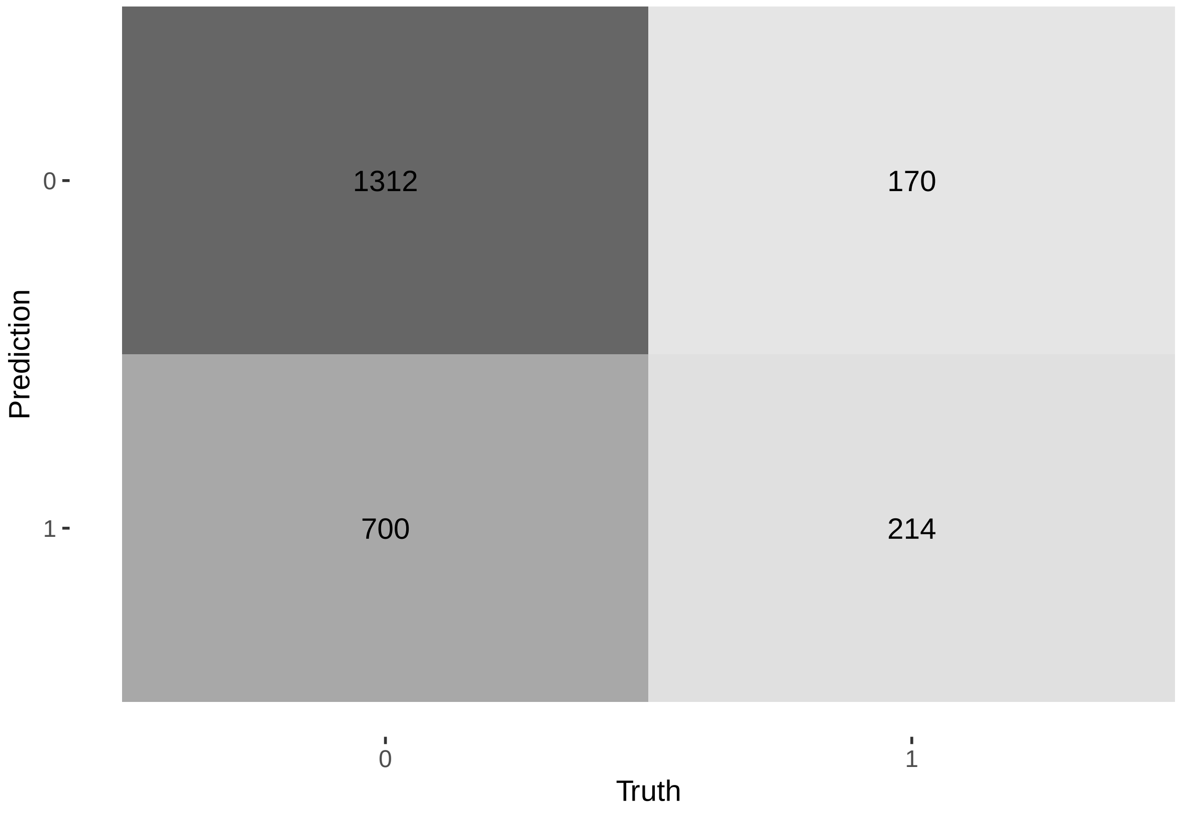
```
final_en_model_tester <- augment(EN_final, new_data = loan_test) %>%
  select(not.fully.paid, starts_with('.pred'))



threshold <- 0.62  # Adjust the threshold as needed

# Create a new column with adjusted predictions based on the threshold
dfe <- final_en_model_tester %>%
  mutate(adjusted_prediction = ifelse(.pred_0 >= threshold, "0", "1"))

dfe
```

```
## # A tibble: 2,396 x 5
##    not.fully.paid .pred_class .pred_0 .pred_1 adjusted_prediction
##    <fct>          <fct>         <dbl>   <dbl> <chr>
##  1 0              0             0.663   0.337 0
##  2 0              1             0.473   0.527 1
##  3 0              0             0.843   0.157 0
##  4 0              0             0.712   0.288 0
##   5 0               0             0.659   0.341 0
##   6 0               0             0.815   0.185 0
##   7 0               0             0.875   0.125 0
##   8 0               0             0.874   0.126 0
##   9 0               0             0.855   0.145 0
## 10 0               0             0.739   0.261 0
## # i 2,386 more rows
```

```r
dfe$adjusted_prediction <- factor(dfe$adjusted_prediction, levels = levels(dfe$not.fully.paid))

accuracy(dfe, truth = not.fully.paid , adjusted_prediction)
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.637
```

```r
conf_mat(dfe, truth = not.fully.paid,
         adjusted_prediction) %>%
  autoplot(type = "heatmap")
```



Overall Accuracy for the Elastic net model was 63.4%

Performance

```r
#Sensitivity

print(232 / (152 + 232))
```

```
## [1] 0.6041667
```

```r
#Specificity

print(1288 / (1288 + 724))
```

```
## [1] 0.640159
```

Conclusion

This project was aimed at predicting the outcome that a person will default on their loan, it employed a range of statistical learning models including Elastic Net, pruned Decision Tree and Boosted trees and K nearest neighbors, Elastic Net on a k-fold cross validation set. These models were tuned based off their hyper parameters to achieve the greatest roc_auc. Elastic Net delivered the highest ability to differentiate between negative and positive classes on the folds with an roc_auc of 0.6788. Before testing the model on the test set some preliminary checks were done on the training set. A confusion matrix testing the model on the training set revealed that there were some issues with sensitivity which is a crucial metric when identifying true positives is a priority. Prior to changing the threshold the sensitvity was 0.25. After increasing the threshold for a positive to 0.38, a sensitivity of 0.58 was achieved on the training set. With this new threshold, the model was ready to be tested on the test set. On the test set the following metrics were achieved

Overall Accuracy: 63.4%

Positive Accuracy (sensitivity): 60.4%

Negative Accuracy (specificity) : 64.0%

Given that randomly choosing whether someone will default or not will give you an accuracy rate of 50%. This elastic net model performs 13.4% better than guessing. Although not very definitive results, the model is able to give some predictive power in determining whether a issued loan will be payed back. In order to improve accuracy rate, more data would be helpful I believe the size of the data set may have had an effect on accuracy. Considering there are 12 predictor variables, it requires more increasingly more amount of data.