Pstat 131 Project

Erasmo Rivas

2022-11-24

```
library(janitor)
library(tidyverse)
library(tidymodels)
library(ISLR)
library(glmnet)
library(ISLR2)
library(corrplot)
library(dplyr)
library(ggplot2)
library(ggthemes)

library(readr)
library(corrr)

library(discrim)
library(klaR)
library(tune)
tidymodels_prefer()

knitr::opts_chunk$set(   # basic chunk settings
    echo = TRUE,
    fig.height = 5,
    fig.width = 7,
    tidy = TRUE,
    tidy.opts = list(width.cutoff = 60)
)
```

Yelp Rating Prediction Model

Erasmo Rivas

UCSB Fall 2022

The aim of this project is to build a machine learning model that can predict the rating a restaurant will receive on yelp. We will use data from the yelp dataset, and implement different techniques to yield the most accurate model for this classification. A classification approach was chosen because, the yelp ratings were only integer multiples of 0.5, the lowest being 0 and highest being 5.

What is Yelp?

Yelp is a mobile app which publishes crowd-sourced reviews about businesses. Yelp operates as a social network site that focuses on reviewing businesses and sharing information about them. For businesses like restaurants, this sort of dispersal of information is the lifesourse of the business, without positive recommendations or reviews by satisfied customers the restaurant may never develop the clientele required to remain afloat. Furthermore, without a source of negative feedback a restaurant may never correct its faults. Yelp, also offers to its users a reliable measure when making a decision on which restaurant to.

Importance of Model

As mentioned before, Yelp is a social network site whose purpose is to review businesses and share information about them. These reviews and ratings, can have much influence on whether a restaurant will succeed or fail.

When opening a restaurant there are many decisions to be made. It would be a useful tool, to know beforehand which expenses should be included. Thus, this model, hopes to serve as a prediction tool for such situations. It will allow for informed decisions on which combination of expenses – predictors, will result in the greatest overall customer satisfaction, which should manifest in the yelp reviews.

Data

The project is centered around Restaurants, thus we only kept observations from the yelp dataset - that are those of restaurants. Furthermore, for the sake of eliminating observations with NA values, we only keep data points with more than seven attributes. We then, collected all the unique attributes within the entirety of the remaining observations, and these are our predictors. The remaining attributes were predominantly binary, thus, if there was NA values, they were replaced with False values. In total, there are 26 variables, this including our response variable star. The predictor variables and their significances are listed below.

review_count : an integer value for the amount of reviews the restaurant has recieved on yelp

is_open : a boolean value (a two level factor) that evaluates to True if the restaurant is open

RestaurantsReservations : a boolean value (or a two level factor) that evaluates to True if the Restaurant has a reservation service

OutdoorSeating : a boolean value (a two level facotr) that evaluates to True if the Restaurant offers outdoor seating

HappyHour : a boolean value (a two level factor) that evaluates to True is the Restaurant offers a happyhour

DriveThru : a boolean value (a two level factor) that evaluates to True if the Restaurant has a drivethru

Caters : a boolean value (a two level factor) that evaluates to True if the Restaurant offers catering services

RestaurantsDelivery : a boolean value (a two level factor) that evaluates to True if the Restaurant offers delivery services

Alcohol : a boolean value (a two level factor) that evaluates to True if the Restaurant serves alcohol

Music : a boolean value (a two level factor) that evaluates to True if the Restaurant plays music

WiFi : a boolean value (a two level factor) that evaluates to True if the Restaurant offers free wifi

RestaurantsGoodForGroups : a boolean value (a two level factor) that evaluates to True if the Restaurant provides a good environment for groups

GoodForKids : a boolean value (a two level factor) that evaluates to True if the Restaurant provides a good environment for children

WheelchairAccessible :a boolean value (a two level facotr) that evaluates to True if the Restaurant is wheel chair accessible

RestaurantsTableService : a booolean value (a two level factor) that evaluate to True if the Restaurant offers table service

HasTV : a boolean value (a two level factor) that evaluates to True if the Restaurant has a television for entertainment purposes

RestaurantsPriceRange2 : a 4 level factor that indicates the price range 4 being the highest, and 0 being the lowest

daysOpen : an integer which represents the number of days the restaurant is open

hoursWeek : a float that represents the number of hours in a week that the Restaurant is open

numCategories : an integer that represents teh number of categories the Restaurant falls into (i.e. fast-food, Italian, Mexican, etc.)

GoodForDessert : a boolean value (a two level factor) that evaluate to True if the Restaurant is good for dessert

GoodForLatenight : a boolean value (a two level factor) that evaluate to True if the Restaurant is good for late night food

GoodForLunch : a boolean value (a two level factor) that evaluate to True if the Restaurant is good for lunch

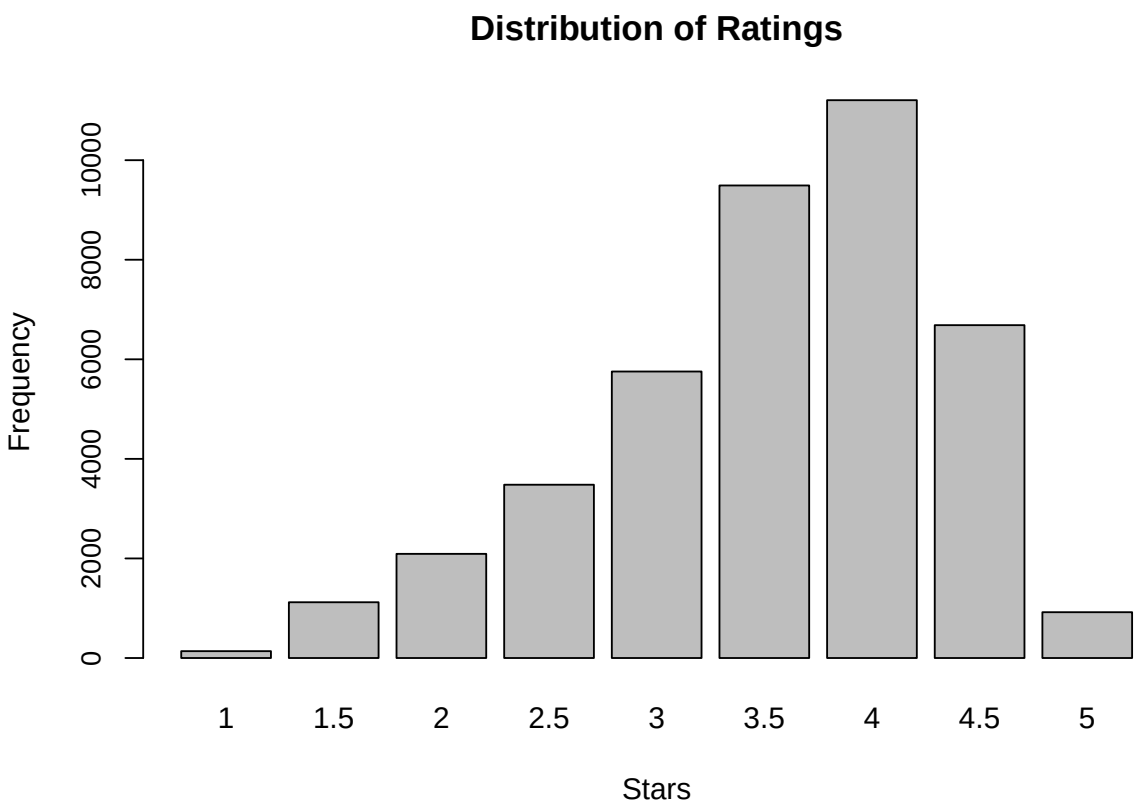
GoodForBrunch : a boolean value (a two level factor) that evaluate to True if the Restaurant is good for brunch

GoodForBreakfast : a boolean value (a two level factor) that evaluate to True if the Restaurant is good for breakfast

Visual Exploratory Data Analysis

Initially, we would like to see the distribution of our target variable.

```
yelp_df <- read.csv("ProjectDatasetComplete.csv", stringsAsFactors = F)

yelp_df <- select(yelp_df,-1,-2,-3)




stars <- yelp_df %>% group_by(stars) %>%
  summarise(count = n()) %>%
  mutate(precentage = (count/sum(count))*100)



barplot(stars$count, names.arg = stars$stars, cex.axis = 0.9, xlab = "Stars", ylab ="Frequency", main =
```



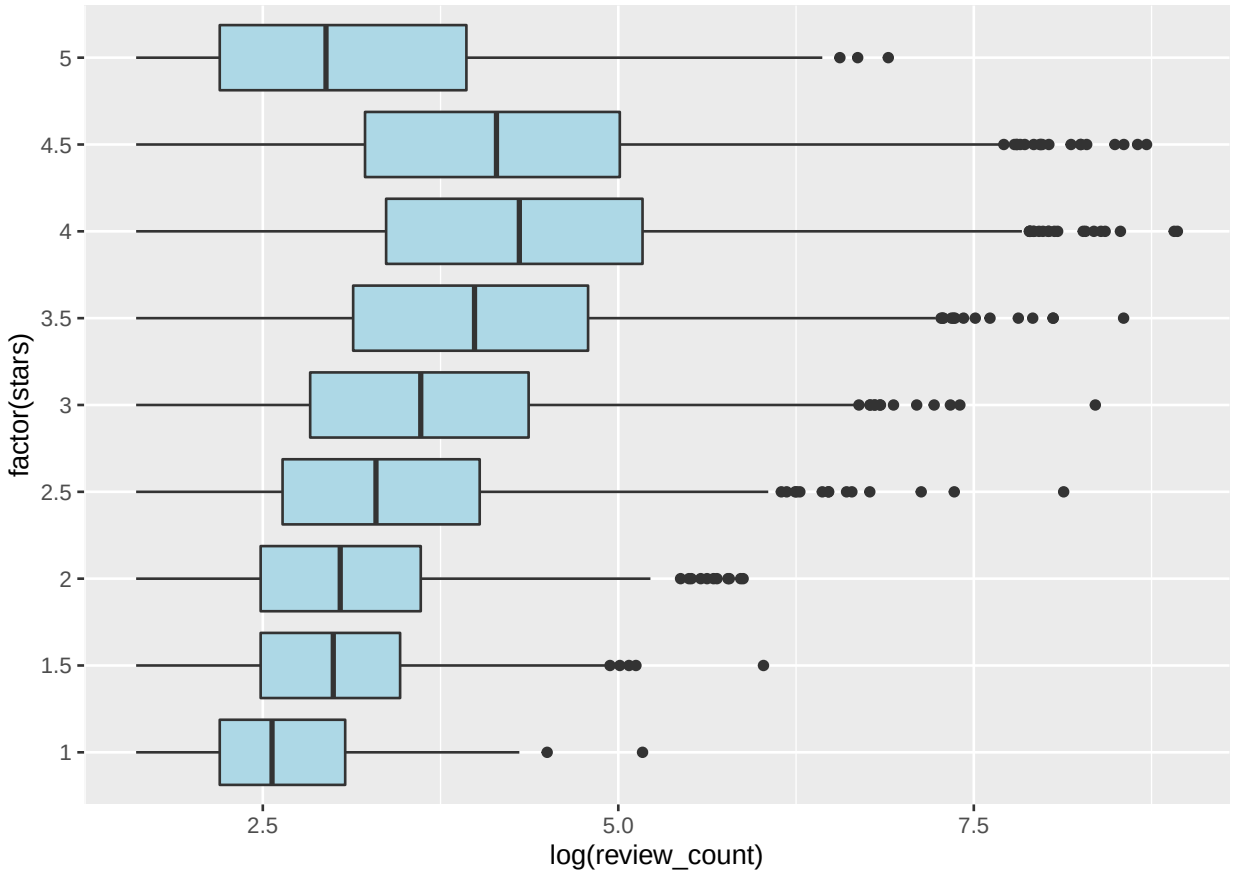
From the above barplot we see that the most common ratings are 4 stars, 3.5 stars, and 4.5 stars, in that order. The majority of the data, has a rating above or equal to 3 stars, that is 83.27% of the data has a rating above 3 stars. It is worth noting that, ratings on the extremes that is either a 1 or a 5 are very uncommon. In fact, only 0.3% of observations have a rating of a 1 and 2% of observations have a rating of a 5.This is key when considering, how we split our data, that is we would like to stratify on stars, so we get a similar distribution in both our training set as in the greater population.

Exploratory Data Analysis

Number of Reviews

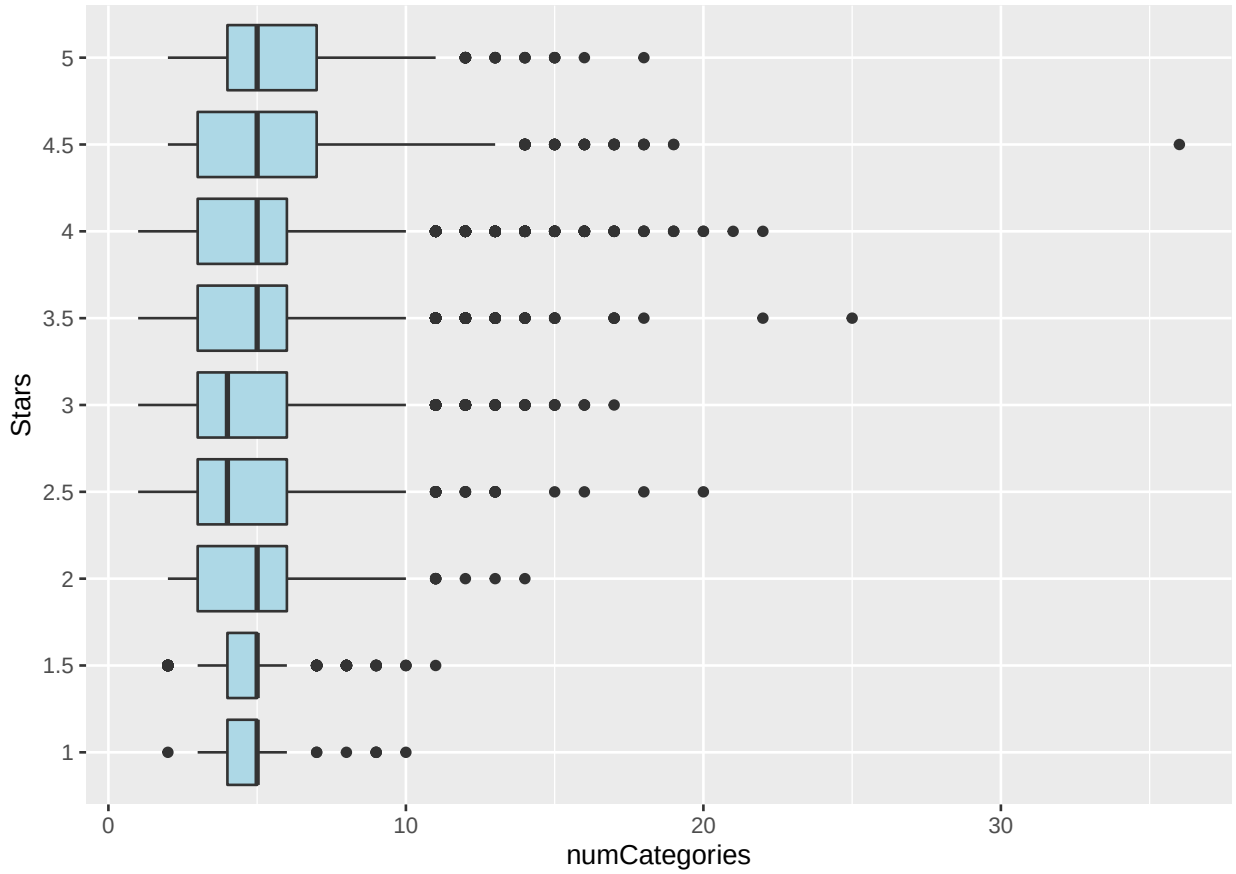
The review_count variable is an integer that represents the number of reviews a restaurant has on yelp. It will not be included in the models, as the aim of the project is to be able to predict the rating of a restaurant – from its opening, based on factors. However, I will include a boxplot of review_countvsstars, to illustrate that there seems to be a relationship between the two. This is to say that, it might be of importance to either incentivize writing a review, as it seems there is initially some positive linear relationship between the two.

```
yelp_df %>% ggplot(aes(x = log(review_count), y = factor(stars))) +
                    geom_boxplot(fill = "lightblue")
```



Categories

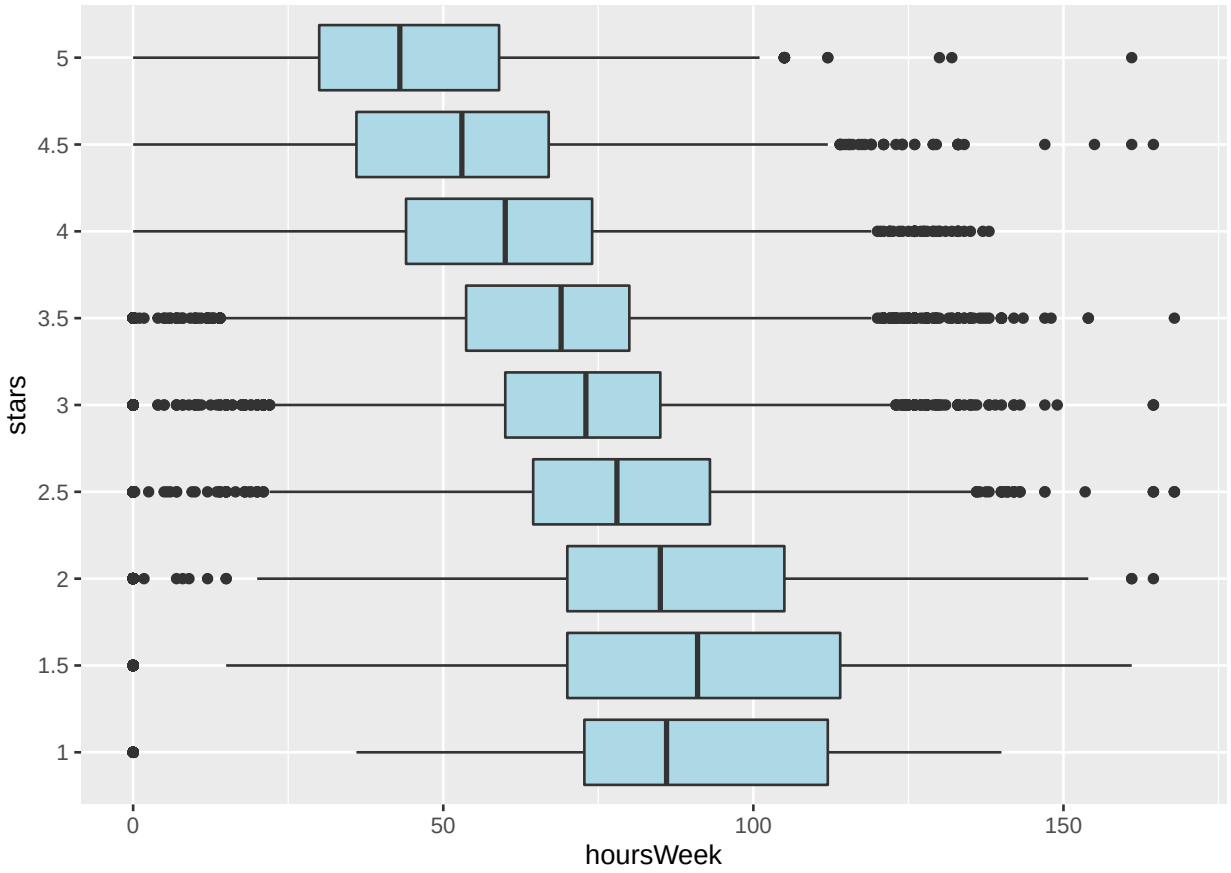
The numCategories variable is an integer which represent the number of distinct categories that a restaurant may fall under. These Categories, span a wide and very inclusive range, some examples of the categories are: Fast-Food, Mexican, Italian, Steak-House, Asain-Cusine, SeaFood, etc. In a sense, this numCategories variable quantifies the versatility of a given restaurant. Thus, given a restaurant high versatility, it is not unreasonable to think, such a restaurant would have wide appeal, and this might manifest in the ratings. Thus we take a look at the following boxplot:



Generally, we see an increase in the rating as the number of categories increases.

Hours Week

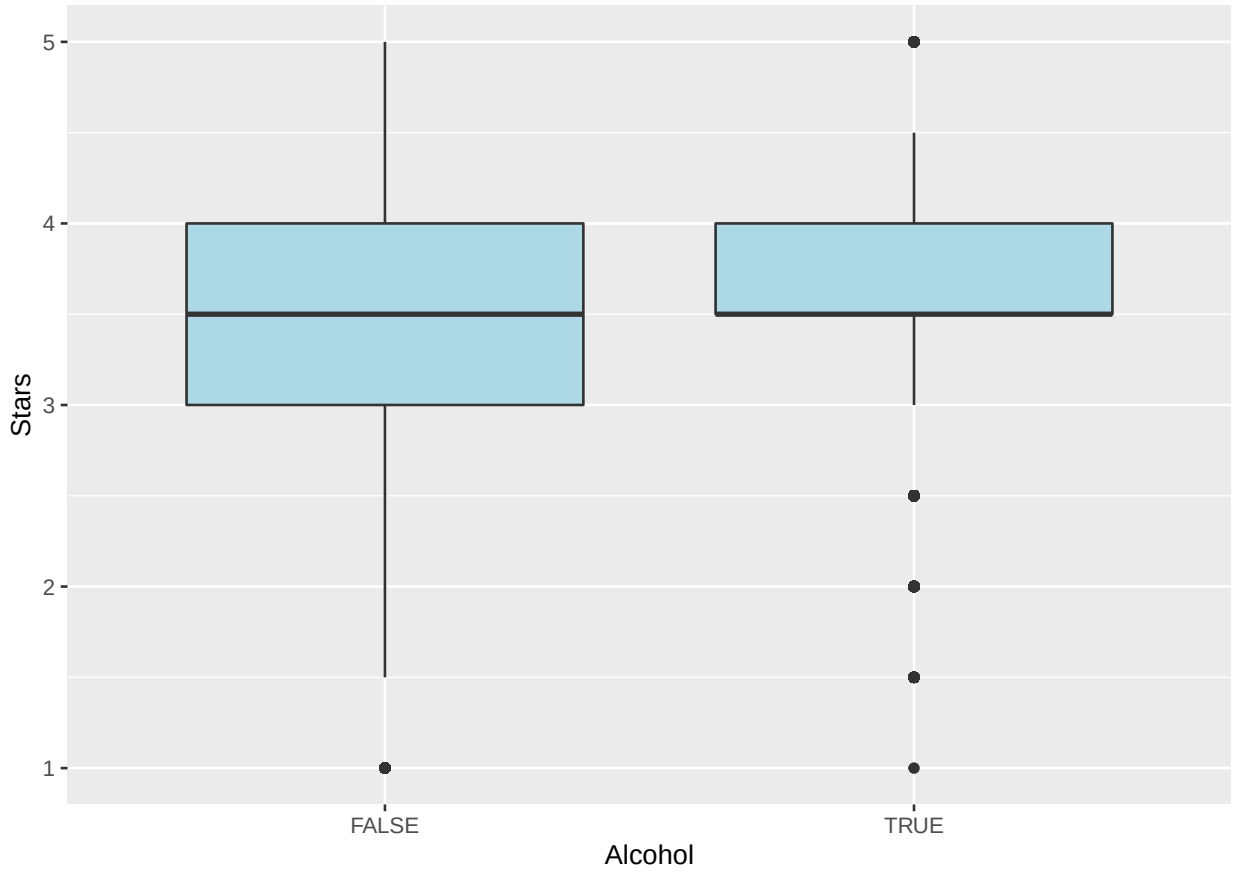
The hoursWeek variable is a numerical variable that is equal to the number of hours a restaurant is open for service in a normal work week. It seems reasonable to believe that the availability of a restaurant would play a role in the rating it would receive, thus we investigate the boxplot of

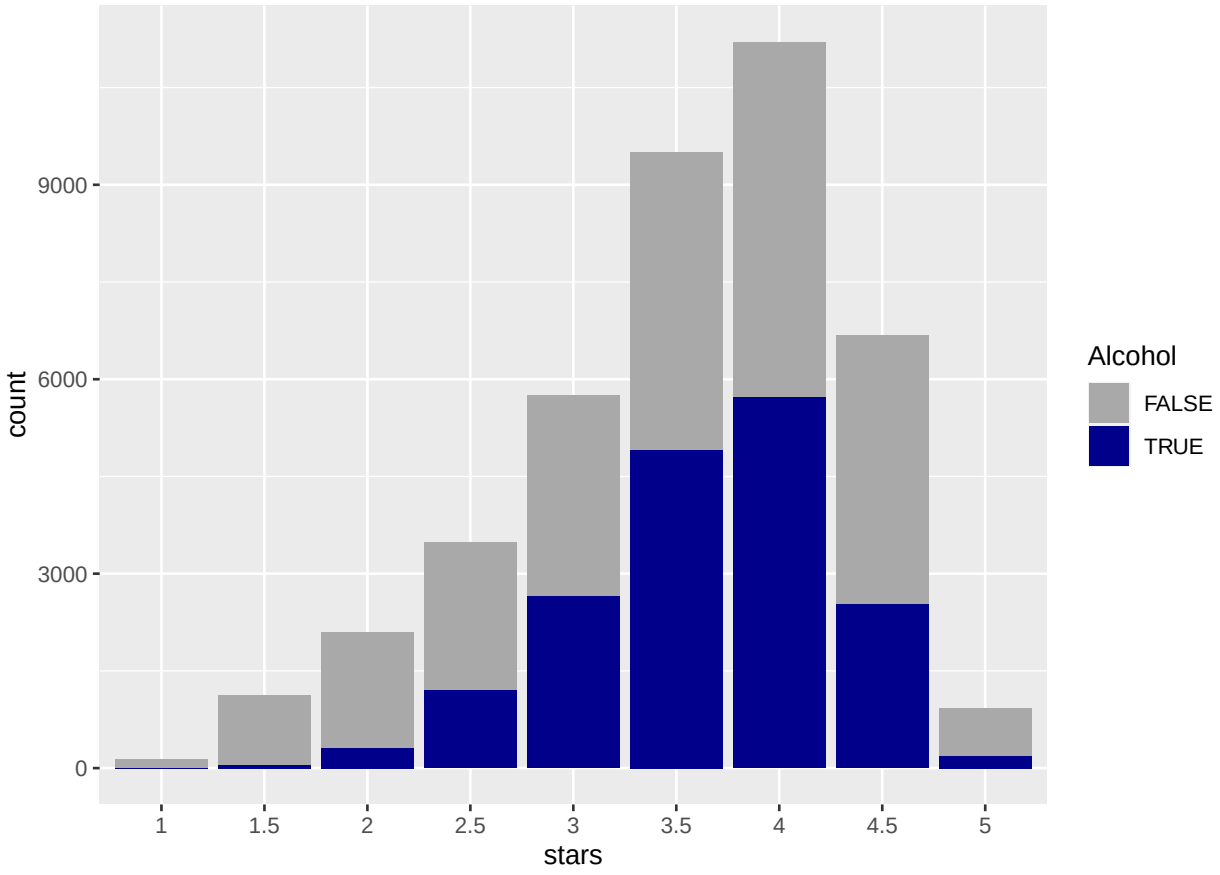


Here it is interesting to see that as we increase the number of hours (or availability) of the restaurant, the rating seems to decrease.

Alcohol

The Alcohol variable is a boolean value that evaluates to True is the restaurant serves alcohol. Alcohol, in many situations is a beverage that may improve the experience. Thus it is reasonable to believe that, this variable would play some affect on the rating. Therefore, we investigate the following plots.

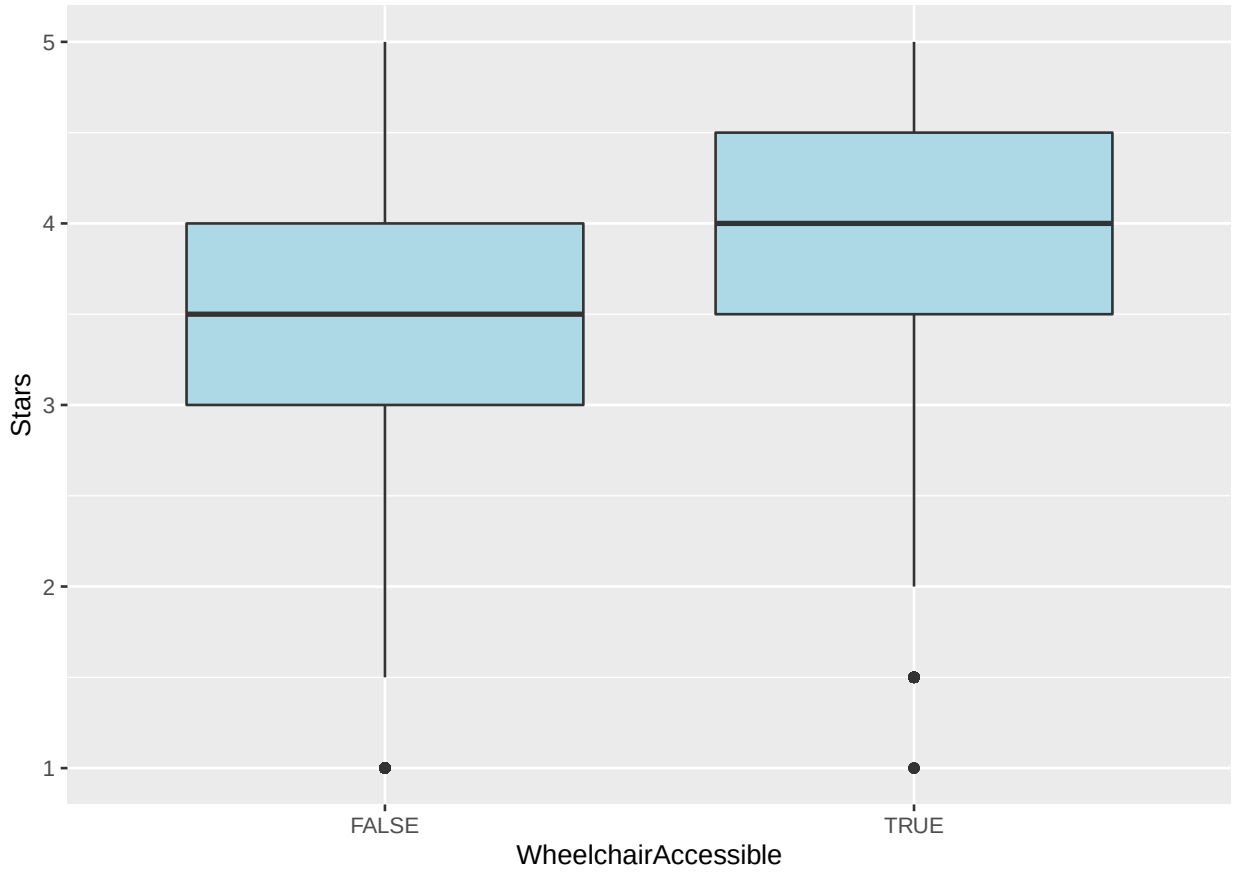

5
4
3
2
1
Stars
FALSE
TRUE
Alcohol



Looking at the box plot of alcohol vs stars we see that the median rating for restaurants, that offer alcohol is very close to the median of restaurants that do not offer alcohol. Which is to say that offering alcohol, makes little difference.

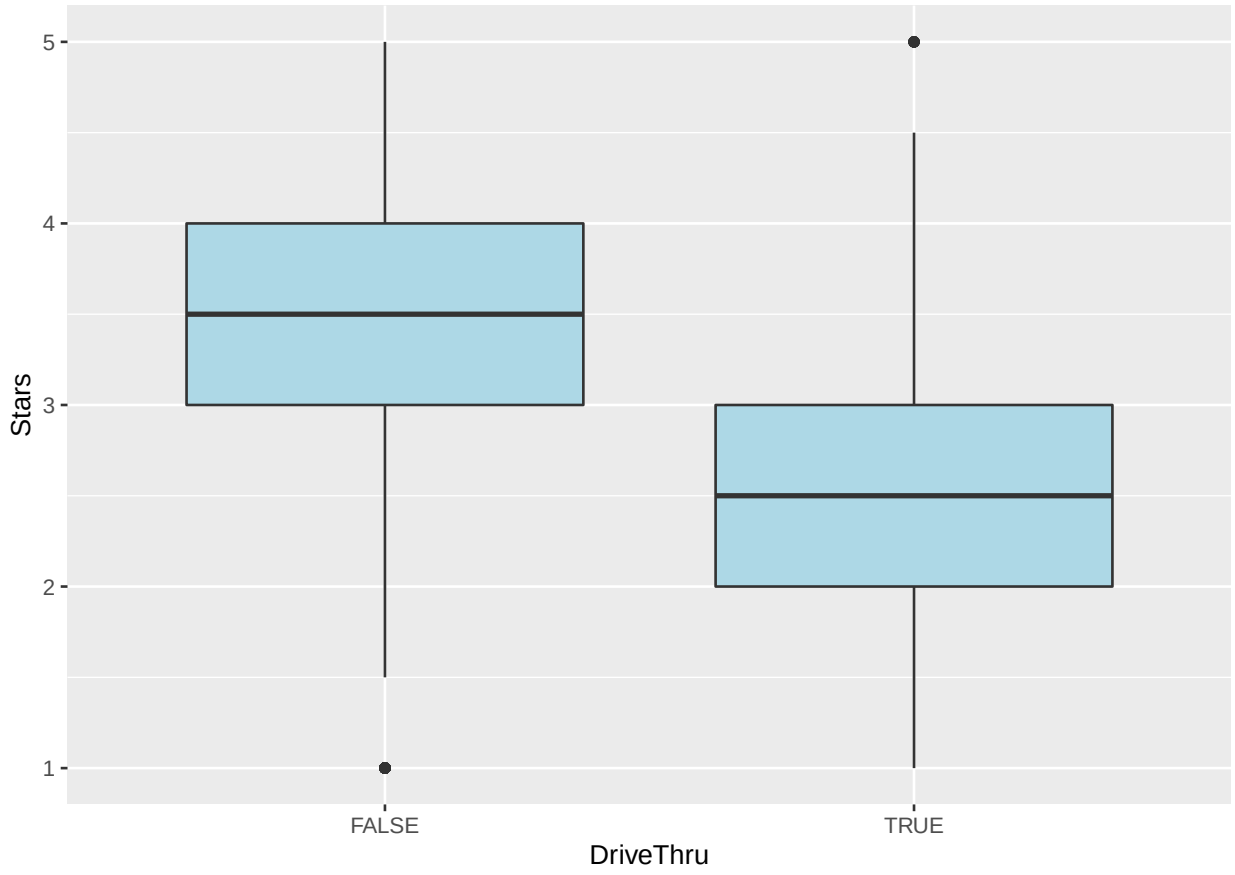
WheelChair Accessibility

The WheelChariAccessible variable is a boolean value that evaluates True if the restaurant makes accommodations for people in wheelchairs. When investigating the WheelChairAccessible vs Stars box plot, we see that the 25th percentile rating for restaurants with these accommodations is reight about the median for those without the accommodations.



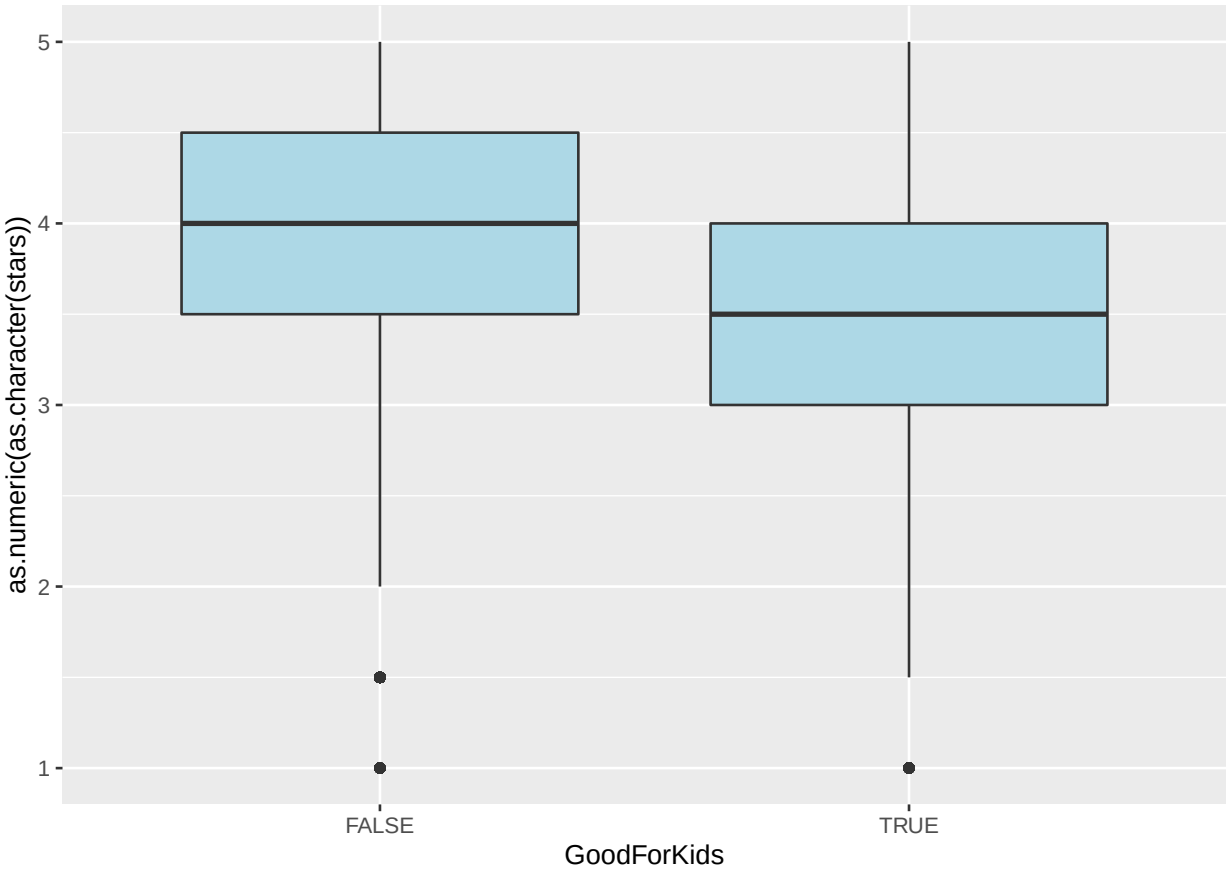
DriveThru

The DriveThru variable is a boolean value, or a two factor level that evaluate to True if the restaurant has a DriveThru. When inspecting the boxplot of DriveThru vs Stars (rating), it is apparent that the observations without a Drivethru have a higher median rating. In fact, the boxplot indicates that the 25th percentile of ratings of restaurants without DriveThrus is higher than the 75th percentile of restaurants with Drivethrus.



Good For Kids

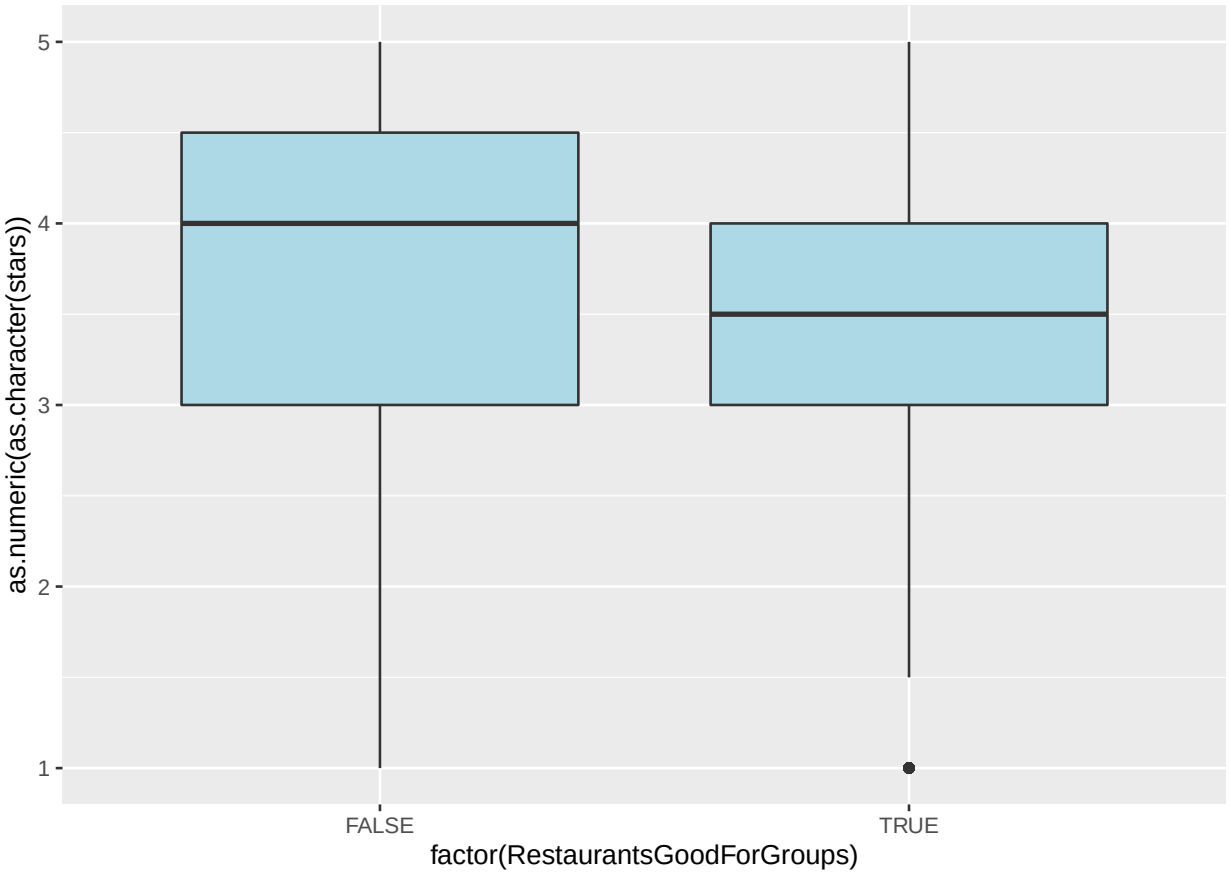
The GoodForKids variable is a boolean, or a two level factor that evaluates to True if the restaurant creates a good environment for children. Investigating the, GoodForKids vs Stars boxplot we see that, the restaurants that create an environment that is not as welcoming towards children have a higher median rating.



Good For Groups

The RestaurantsGoodForGroups variable is a boolean, or a two level factor that evaluates to True if the restaurant provides an environment that is hospitable and welcoming for groups of people. Investigating the Boxplot, we see that, the median for groups that choose to not focus on groups as much (True) is slightly lower than the other (False).

```
ggplot(yelp_df, aes(y = as.numeric(as.character(stars)), x = factor(RestaurantsGoodForGroups))) +
                    geom_boxplot(fill = "LightBlue")
```



Data Split

The data was split 75% training and 25% testing. Stratified sampling was used on our response variable stars in order to capture the greater distribution of the population in each split.

```
set.seed(1111)

yelp_df <- select(yelp_df,-2)

yelp_split <- initial_split(yelp_df, prop = 0.75, strata = stars)

yelp_train <- training(yelp_split)
yelp_test <- testing(yelp_split)

dim(yelp_train)
```

```
## [1] 30665    25
```

```
dim(yelp_test)
```

```
## [1] 10224    25
```

The training data set has 30,665 observations and the testing set has 10,224 observations.

```
yelp_folds <- vfold_cv(yelp_train, v= 10, strata = stars)
```

Here we fold our to avoid both overfitting and selection bias. We use 10 folds, as it is one of the recommended number of folds.

SetUp

Since we are dealing with classification, we will develop four different models and compare them to see which performs the best. We will, create a Linear Discriminant model, a K-nearest neighbor model, an elastic net model, and a random forest model. In the following I set up the models and the workflows.

Recipe

In the following we set up the recipe with the predictor variables that were mentioned before. We normalize the predictors to ease the computations.

```
yelp_recipe <- recipe(stars ~ ., data = yelp_train) %>% step_dummy(all_nominal_predictors()) %>%
  step_normalize(all_predictors())
```

Linear Discriminant Analysis

Here we set up a Linear Discriminant Model

```
lda_mod <- discrim_linear() %>%
  set_engine("MASS") %>%
  set_mode("classification")

lda_wkflow <- workflow() %>%
  add_recipe(yelp_recipe) %>%
  add_model(lda_mod)
```

K Nearest Neighbor

Here we set up a K nearest neighbor model and tune the neightbors hyperparameter.

```
knn_model <- nearest_neighbor(neighbors = tune(),
                              mode = "classification") %>%
  set_engine("kknn")

knn_wkflow <- workflow() %>%
  add_model(knn_model) %>%
  add_recipe(yelp_recipe)
```

```
knn_params <- parameters(knn_model)

knn_grid <-grid_regular(knn_params, levels = 2 )
```

```
knn_tune <- knn_wkflow %>%
  tune_grid(
    resamples = yelp_folds,
    grid = knn_grid
  )


save(knn_tune, knn_wkflow, file = "knn.rda")
```

Elastic Net

Here we set up an elastic net model, here we tune the penalty and the mixture hyperparameters.

```
?multinom_reg
elastic_net_spec <- multinom_reg(penalty = tune(),
                                 mixture = tune()) %>%
  set_mode("classification") %>%
  set_engine("glmnet")

en_wkflow <- workflow() %>%
  add_recipe(yelp_recipe) %>%
  add_model(elastic_net_spec)

en_grid <- grid_regular(penalty(range = c(-5,5)),
                        mixture(range = c(0,1)),
                        levels = 10)
```

```
tune_res <- tune_grid(
  en_wkflow,
  resamples = yelp_folds,
  grid = en_grid
)
```

Random Forest

Here we set up a random forest model, we allow for the tuning of min_n, mtry, and trees. Mtry will be the number of predictors randomly sampled at each split when creating tree models. Trees is for the number of trees, and min_n is for the number data points that are required for the node to be split further.

```
rf_model <- rand_forest(
  min_n = tune(),
  mtry = tune(),
  trees = tune(),
  mode = "classification") %>%
  set_engine("ranger", importance = "impurity")

rf_wkflow <- workflow() %>%
  add_model(rf_model) %>%
  add_recipe(yelp_recipe)
```

```
rf_params <- parameters(rf_model) %>%
  update(mtry = mtry(range=c(2,10)))

rf_grid <- grid_regular(rf_params, levels = 3)
```

```
rf_tune <- rf_wkflow %>%
  tune_grid(
    resamples = yelp_folds,
    grid = rf_grid,
    metrics = metric_set(roc_auc, accuracy)
  )

#?metric_set


save(rf_tune, rf_wkflow, file = "rf.rda")
```

Now below we'll see the model performances on the folded data.

```
load("knn.rda")
load("rf.rda")
load("elas_net.rda")
```

```
lda_fit <- fit_resamples(lda_wkflow,
                         resamples = yelp_folds,
                         control = control_resamples())

collect_metrics(lda_fit)
```

LDA metrics

```
## # A tibble: 2 x 6
##   .metric  .estimator  mean     n std_err .config
##   <chr>    <chr>      <dbl> <int>   <dbl> <chr>
## 1 accuracy multiclass 0.312    10 0.00262 Preprocessor1_Model1
## 2 roc_auc  hand_till  0.763    10 0.00242 Preprocessor1_Model1
```

We see that the lda model has an roc_auc of 0.7630, which indicates that the model is still significantly better than randomly assigning one of the 9 possible ratings.

```
collect_metrics(knn_tune)
```

K Nearest Neightbor Metrics

```
## # A tibble: 4 x 7
##   neighbors .metric  .estimator  mean     n std_err .config
```

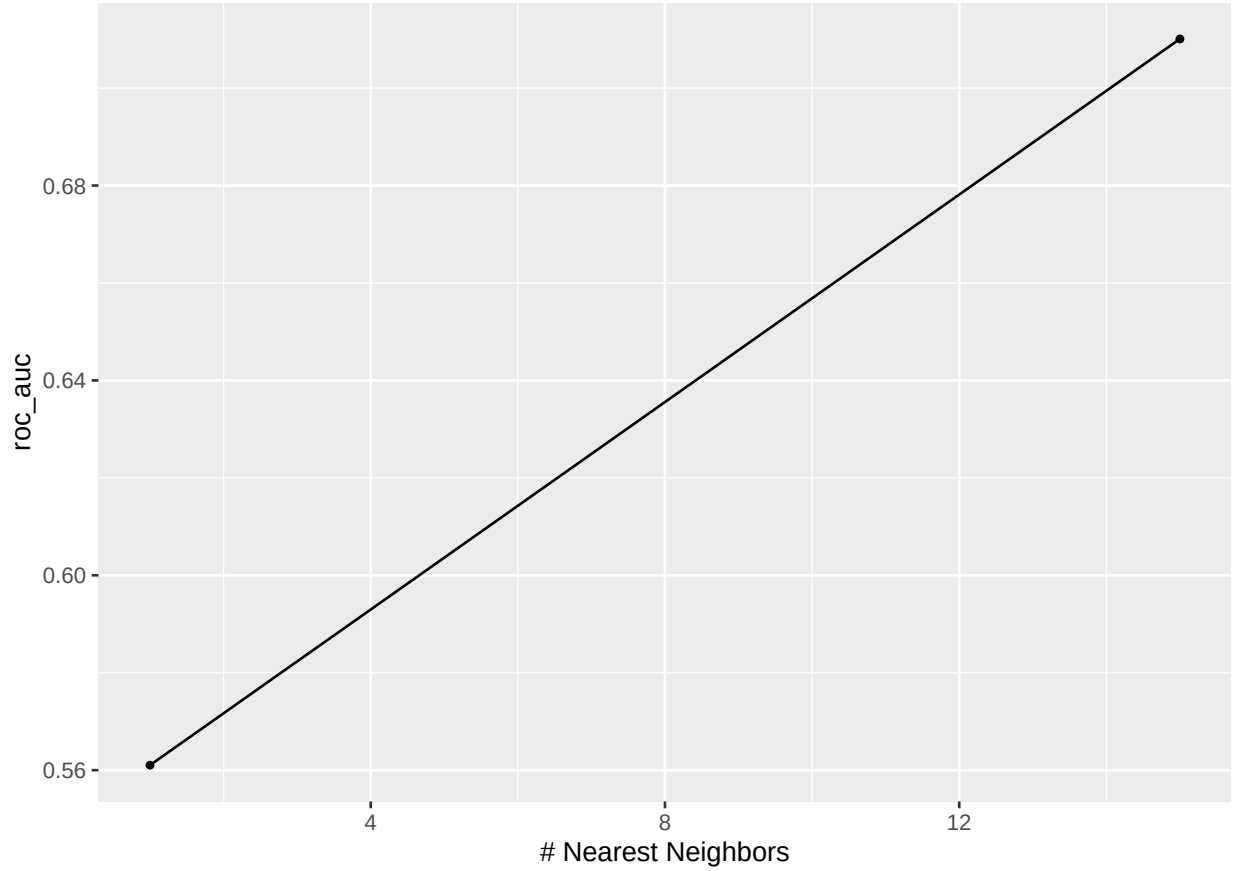
```
##       <int> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
## 1         1 accuracy multiclass 0.264    10 0.00307 Preprocessor1_Model1
## 2         1 roc_auc  hand_till  0.561    10 0.00230 Preprocessor1_Model1
## 3        15 accuracy multiclass 0.305    10 0.00373 Preprocessor1_Model2
## 4        15 roc_auc  hand_till  0.710    10 0.00281 Preprocessor1_Model2
```

```
show_best(knn_tune)
```

```
## Warning: No value of ‘metric‘ was given; metric ’roc_auc’ will be used.
```

```
## # A tibble: 2 x 7
##   neighbors .metric .estimator  mean     n std_err .config
##       <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1        15 roc_auc hand_till  0.710    10 0.00281 Preprocessor1_Model2
## 2         1 roc_auc hand_till  0.561    10 0.00230 Preprocessor1_Model1
```

```
autoplot(knn_tune, metric = "roc_auc")
```



We see that the roc_auc = 071008 for the best k nearest neighbor model, which is to say that it performs better than random assignment (of flipping a fair 9 sided coin).

```
collect_metrics(tune_res)
```

Elastic Net

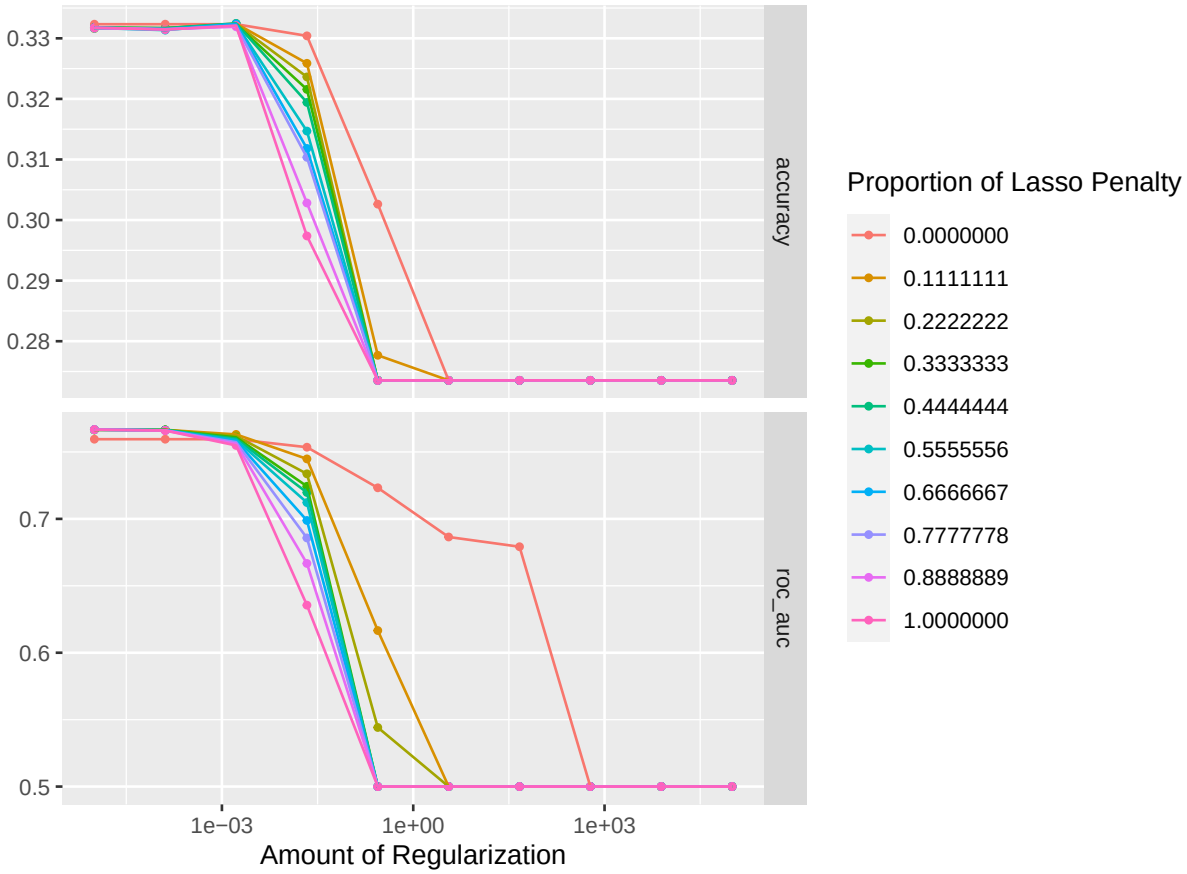
```
## # A tibble: 200 x 8
##     penalty mixture .metric  .estimator  mean     n std_err .config
##       <dbl>   <dbl> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
##  1 0.00001        0 accuracy multiclass 0.332    10 0.00158 Preprocessor1_Model~
##  2 0.00001        0 roc_auc  hand_till  0.760    10 0.00228 Preprocessor1_Model~
##  3 0.000129       0 accuracy multiclass 0.332    10 0.00158 Preprocessor1_Model~
##  4 0.000129       0 roc_auc  hand_till  0.760    10 0.00228 Preprocessor1_Model~
##  5 0.00167        0 accuracy multiclass 0.332    10 0.00158 Preprocessor1_Model~
##  6 0.00167        0 roc_auc  hand_till  0.760    10 0.00228 Preprocessor1_Model~
##  7 0.0215         0 accuracy multiclass 0.330    10 0.00221 Preprocessor1_Model~
##  8 0.0215         0 roc_auc  hand_till  0.754    10 0.00230 Preprocessor1_Model~
##  9 0.278          0 accuracy multiclass 0.303    10 0.00160 Preprocessor1_Model~
## 10 0.278          0 roc_auc  hand_till  0.723    10 0.00247 Preprocessor1_Model~
## # ... with 190 more rows
```

```
show_best(tune_res)
```

```
## Warning: No value of ‘metric‘ was given; metric ’roc_auc’ will be used.
```

```
## # A tibble: 5 x 8
##   penalty mixture .metric .estimator  mean     n std_err .config
##     <dbl>   <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1 0.00001   1     roc_auc hand_till  0.767    10 0.00220 Preprocessor1_Model091
## 2 0.00001   0.889 roc_auc hand_till  0.767    10 0.00221 Preprocessor1_Model081
## 3 0.00001   0.778 roc_auc hand_till  0.767    10 0.00219 Preprocessor1_Model071
## 4 0.00001   0.667 roc_auc hand_till  0.767    10 0.00220 Preprocessor1_Model061
## 5 0.00001   0.556 roc_auc hand_till  0.767    10 0.00220 Preprocessor1_Model051
```

```
autoplot(tune_res)
```



We see that the best performing model has an roc_auc of 0.76668, which is the best so far.

```
collect_metrics(rf_tune)
```

Random Forest

```
## # A tibble: 54 x 9
##     mtry trees min_n .metric  .estimator  mean     n std_err .config
##    <int> <int> <int> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
##  1     2     1     2 accuracy multiclass 0.245    10 0.0180  Preprocessor1_Mode~
##  2     2     1     2 roc_auc  hand_till  0.594    10 0.0219  Preprocessor1_Mode~
##  3     6     1     2 accuracy multiclass 0.267    10 0.0135  Preprocessor1_Mode~
##  4     6     1     2 roc_auc  hand_till  0.573    10 0.00911 Preprocessor1_Mode~
##  5    10     1     2 accuracy multiclass 0.249    10 0.0148  Preprocessor1_Mode~
##  6    10     1     2 roc_auc  hand_till  0.554    10 0.00766 Preprocessor1_Mode~
##  7     2  1000     2 accuracy multiclass 0.334    10 0.0140  Preprocessor1_Mode~
##  8     2  1000     2 roc_auc  hand_till  0.754    10 0.00980 Preprocessor1_Mode~
##  9     6  1000     2 accuracy multiclass 0.340    10 0.0169  Preprocessor1_Mode~
## 10     6  1000     2 roc_auc  hand_till  0.753    10 0.0102  Preprocessor1_Mode~
## # ... with 44 more rows
```

```
show_best(rf_tune)
```

```
## Warning: No value of 'metric' was given; metric 'roc_auc' will be used.
```

```
## # A tibble: 5 x 9
##    mtry trees min_n .metric .estimator  mean     n std_err .config
##   <int> <int> <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1     6  2000    40 roc_auc hand_till  0.765    10 0.00760 Preprocessor1_Model26
## 2     6  1000    40 roc_auc hand_till  0.763    10 0.00914 Preprocessor1_Model23
## 3     6  2000    21 roc_auc hand_till  0.762    10 0.00824 Preprocessor1_Model17
## 4     6  1000    21 roc_auc hand_till  0.761    10 0.00807 Preprocessor1_Model14
## 5    10  1000    40 roc_auc hand_till  0.759    10 0.0106  Preprocessor1_Model24
```

We see that the best model, has an roc_auc of 0.765 which is slightly less than that of the elastic net model. However, it is still very good, as it greatly would outperform random (uniform) assignment.

Final Model Building

The model with the best roc_auc was the elastic net model. It beat out the random forest model by 0.001. Thus in the following, we fit the model with the training data and then we, test it with the test dataset.

```
elasnet_wkflow_final <- finalize_workflow(en_wkflow, select_best(tune_res, metric = "roc_auc"))

elas_final_fit <- fit(elasnet_wkflow_final, yelp_train)
```

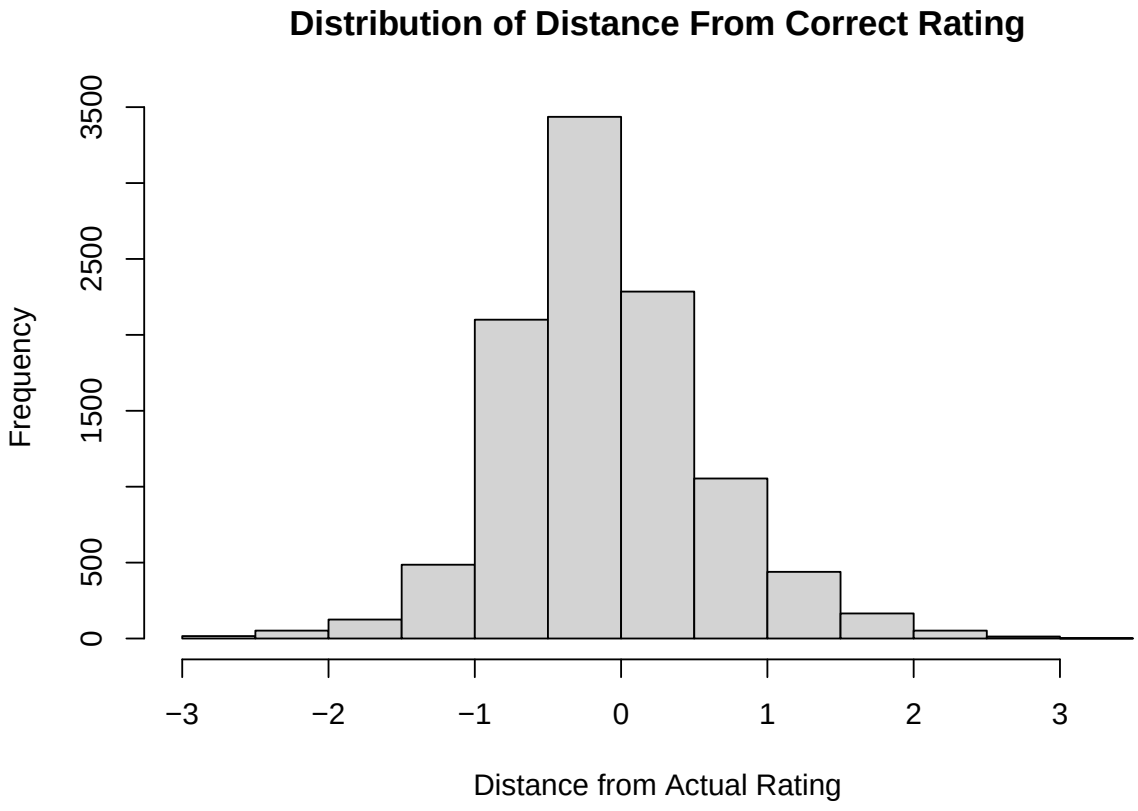
```
yelp_metric <- metric_set(accuracy)

test_predictions <- predict(elas_final_fit, new_data = yelp_test) %>%
  bind_cols(yelp_test %>% select(stars))

Test_pred <- test_predictions %>% mutate(error = as.numeric(as.character(.pred_class))-as.numeric(as.cha

Test_pred
```

```
## # A tibble: 10,224 x 3
##    .pred_class stars error
##    <fct>       <fct> <dbl>
##  1 4           4.5    -0.5
##  2 4           2.5     1.5
##  3 3.5         4      -0.5
##  4 3.5         3       0.5
##  5 4           3.5     0.5
##  6 4           3       1
##  7 4           3.5     0.5
##  8 4           3       1
##  9 4.5         4.5     0
## 10 3.5         3.5     0
## # ... with 10,214 more rows
```

```
Test_pred %>% group_by(error) %>%
  summarise(count = n(), percentage = n()/10224)
```

```
## # A tibble: 14 x 3
##    error count percentage
##    <dbl> <int>      <dbl>
##  1  -3       5  0.000489
##  2  -2.5    11  0.00108
##  3  -2      52  0.00509
##  4  -1.5   125  0.0122
##  5  -1     486  0.0475
##  6  -0.5  2100  0.205
##  7   0    3436  0.336
##  8   0.5  2285  0.223
##  9   1    1054  0.103
## 10   1.5   439  0.0429
## 11   2     165  0.0161
## 12   2.5    52  0.00509
## 13   3      13  0.00127
## 14   3.5     1  0.0000978
```

```
hist(Test_pred$error, main = "Distribution of Distance From Correct Rating", xlab = "Distance from Actua
```



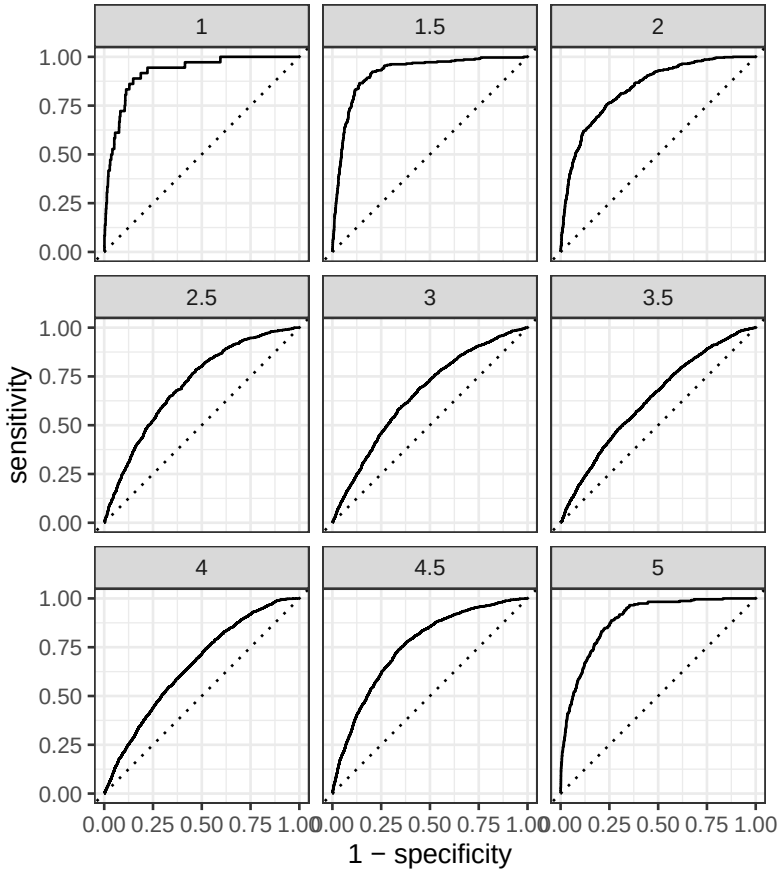
Since the predicted variable is a factor but is also an ordered factor we can consider the distribution of the error, or how far the predicted factor (stars) is from the actual stars. Inspecting the predicted data,

76.4962% of the predicted data is within in 1 factor level or within 0.5 stars of the actual rating (which is to say that if the real rating is 3.5 that it predicts either a 4 or a 3). The distribution of the error, or distance from the predicted rating to the actual rating can be see by the histogram. The distribution seem to be normal about 0.

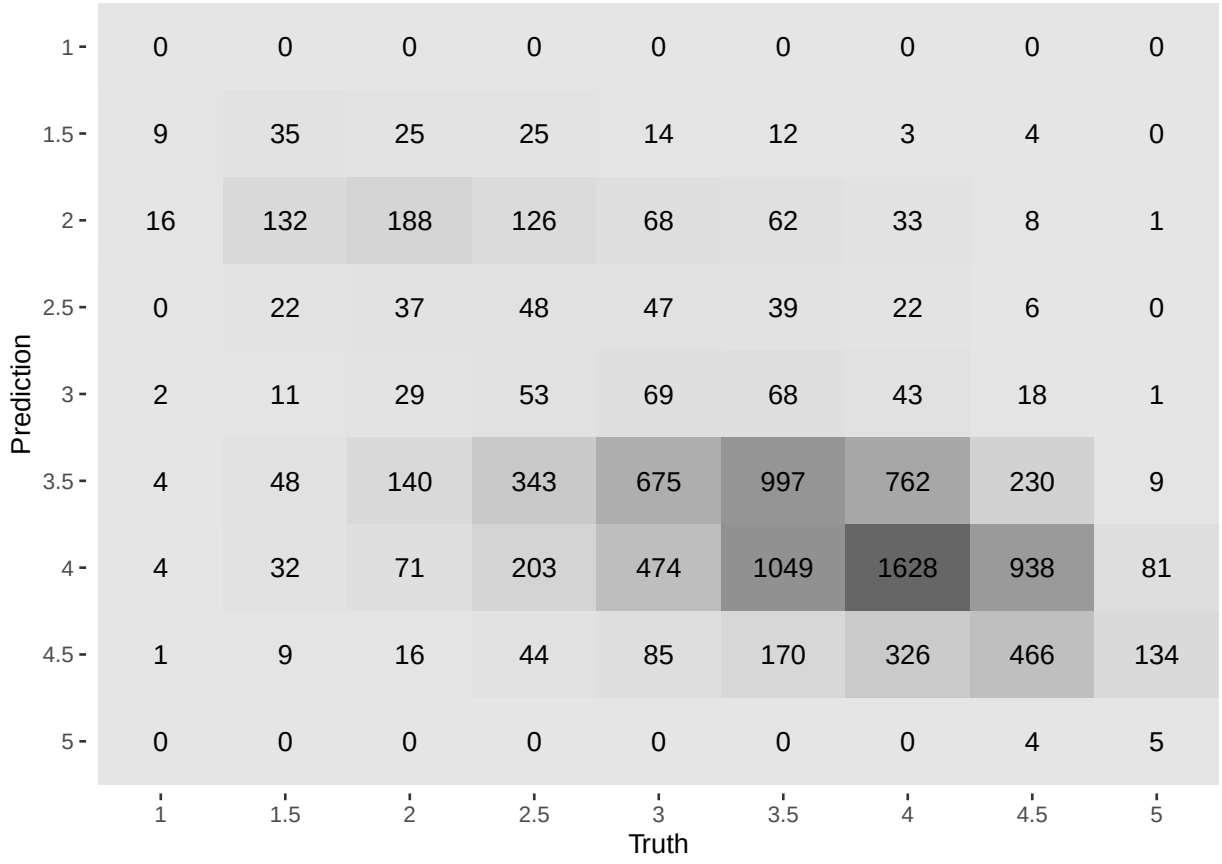
```
test_predictions %>%
  yelp_metric(truth = stars, estimate = .pred_class)
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy multiclass     0.336
```

```
predicted_data <- augment(elas_final_fit, new_data = yelp_test) %>%
  select(stars, starts_with(".pred"))

predicted_data %>% roc_curve(stars, .pred_1:.pred_5) %>% autoplot()
```




```
predicted_data %>% conf_mat(truth = stars, estimate = .pred_class) %>%
  autoplot("heatmap")
```




```
predicted_data %>% roc_auc(stars, .pred_1:.pred_5)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc hand_till      0.766
```

The accuracy of the model with regards to predicting the correct rating class or stars is 33.6072%, this better than randomly selecting a class, as there are 9 possible classes and flipping a 9 sided fair coin would result in about 11.111% success, thus, the model is fairly good at predicting when all considered. Furthermore, the roc_auc is about 0.767 which fairly decent, it is much higher than 0.5 - which again indicates that it is better than the flip of a 9 sided fair coin. The one issue, however, the model struggles to predict ratings of 1 and 5 this is perhaps due to the lack of the observations with such ratings.

Conclusion

Our analysis, of the yelp data produced fairly good results - far better than expected. We were able to determine the correct rating 33% of the time, and we were able to land within one rating or 0.5 stars away 76% of the time. Where we may be able to improve, is creating or finding a datset with more observations with ratings of 1 and 5. Each model had issue predicting ratings of 1 and 5, this is probably due to scarcity of observations with a rating of 1 and 5 in the data.

In the end, it was the elastic net model that performed best, with respect to the metric roc_auc. In a very close second came the random forest model, which was to my surprise, as I expected the random forest model to outperform the elastic net model.

Furthermore, other possible endeavors may include expanding upon this model and including non-restaurant businesses. We can also, include models that are more interpretation friendly so that we can know beforehand which predictors are of greater importance. That is to say, we do not treat this model as a black box and plug in predictor values to get the prediction. Which, is not to negate the importance and applicability of this models predictive powers.