Introduction
The Rationale of Choosing this Topic as My Final Project
Acknowledgements
Inspiration
Loading Data and Packages
Data Split
Exploratory Data Analysis
Building Model
Analysis of The Testing Set
Conclusion

# Final Project

Code

**Wayne Luo**

Code

## Introduction

The purpose of this project is to find the best model to predict the chance of admit for graduate admission.

## The Rationale of Choosing this Topic as My Final Project

As a graduating senior, I have just completed my application season for graduate school. I received a few acceptance letters, but more rejections. Most of the results seemed reasonable to me, but that didn't preclude a few of the safety schools I applied to from rejecting me. In my opinion, graduate school admissions is a very metaphysical thing, so it is my keen interest to explore the logic behind the admissions office's decisions. Which attributes they are more emphrasizing and which attributes can increase the chance of getting admitted.

## Acknowledgements

This dataset is inspired by the UCLA Graduate Dataset. The test scores and GPA are in the older format. The dataset is owned by Mohan S Acharya.

## Inspiration

This dataset was built with the purpose of helping students in shortlisting universities with their profiles. The predicted output gives them a fair idea about their chances for a particular university.

# Loading Data and Packages

The dataset contains several parameters which are considered important during the application for Masters Programs. The parameters included are :

- `GRE Scores` : Graduate Record Examination Scores ( out of 340 )
- `TOEFL Scores` : Test of English as a Foreign Language Scores ( out of 120 )
- `University Rating` : The Rating of Undergraduate School( out of 5 )
- `SOP` : Statement of Purpose (out of 5)
- `LOP` : Letter of Recommendation Strength ( out of 5 )
- `CGPA` : Undergraduate Cumulative GPA Based on Indian Colleges ( out of 10 )
- `Research` : Whether or not Have Research Experience ( either 0 or 1 )
- `Chance of Admit` : Chance of getting admission ( ranging from 0 to 1 )

*Note: a full copy of the codebook is availabe in my zipped files.*

# Import Data and Data Understanding

Code

```
## # A tibble: 6 × 9
##   `Serial No.` `GRE Score` `TOEFL Score` `University Rating`   SOP   LOR  CGPA
##          <dbl>       <dbl>         <dbl>                <dbl> <dbl> <dbl> <dbl>
## 1            1         337           118                    4   4.5   4.5  9.65
## 2            2         324           107                    4   4     4.5  8.87
## 3            3         316           104                    3   3     3.5  8   
## 4            4         322           110                    3   3.5   2.5  8.67
## 5            5         314           103                    2   2     3    8.21
## 6            6         330           115                    5   4.5   3    9.34
## # … with 2 more variables: Research <dbl>, `Chance of Admit` <dbl>
```

Code

```
## spec_tbl_df [400 × 9] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ Serial No.       : num [1:400] 1 2 3 4 5 6 7 8 9 10 ...
##  $ GRE Score        : num [1:400] 337 324 316 322 314 330 321 308 302 323 ...
##  $ TOEFL Score      : num [1:400] 118 107 104 110 103 115 109 101 102 108 ...
##  $ University Rating: num [1:400] 4 4 3 3 2 5 3 2 1 3 ...
##  $ SOP              : num [1:400] 4.5 4 3 3.5 2 4.5 3 3 2 3.5 ...
##  $ LOR              : num [1:400] 4.5 4.5 3.5 2.5 3 3 4 4 1.5 3 ...
##  $ CGPA             : num [1:400] 9.65 8.87 8 8.67 8.21 9.34 8.2 7.9 8 8.6 ...
##  $ Research         : num [1:400] 1 1 1 1 0 1 1 0 0 0 ...
##  $ Chance of Admit  : num [1:400] 0.92 0.76 0.72 0.8 0.65 0.9 0.75 0.68 0.5 0.45
...
##  - attr(*, "spec")=
##   .. cols(
##   ..   `Serial No.` = col_double(),
##   ..   `GRE Score` = col_double(),
##   ..   `TOEFL Score` = col_double(),
##   ..   `University Rating` = col_double(),
##   ..   SOP = col_double(),
##   ..   LOR = col_double(),
##   ..   CGPA = col_double(),
##   ..   Research = col_double(),
##   ..   `Chance of Admit` = col_double()
##   .. )
##  - attr(*, "problems")=<externalptr>
```

The variables such as strength of SOP and LOR, university_rating, and research experience need to be factored, because basically they are categorical variables in numeric form. In this case they can be used in statistical modeling where they will be implemented correctly.

Code

The dataset is pretty tidy already, so I just did some simple data cleaning so that the space from original dataset turned into "_" and the variable's name turned into lower case. This is more convenient for me to call on each variables and manipulate data. And Serial Number don't do much help with our goal, so I removed it.

Code

```
## # A tibble: 400 × 8
##    gre_score toefl_score university_rating sop   lor    cgpa research
##        <dbl>       <dbl> <fct>             <fct> <fct> <dbl> <fct>
##  1       337         118 4                 4.5   4.5    9.65 1
##  2       324         107 4                 4     4.5    8.87 1
##  3       316         104 3                 3     3.5    8    1
##  4       322         110 3                 3.5   2.5    8.67 1
##  5       314         103 2                 2     3      8.21 0
##  6       330         115 5                 4.5   3      9.34 1
##  7       321         109 3                 3     4      8.2  1
##  8       308         101 2                 3     4      7.9  0
##  9       302         102 1                 2     1.5    8    0
## 10       323         108 3                 3.5   3      8.6  0
## # … with 390 more rows, and 1 more variable: chance_of_admit <dbl>
```

# Data Split

I plan to split my data in a proportion of 75% training, 25% testing split, stratifying on the outcome variable `chance_of_admit`.

The data split was conducted prior to the EDA as I did not want to know anything about my testing data set before I tested my model on those observations

Code

let's verify that the training and testing data sets have the appropriate number of observations.

Code

```
## [1] 298   8
```

Code

```
## [1] 102   8
```

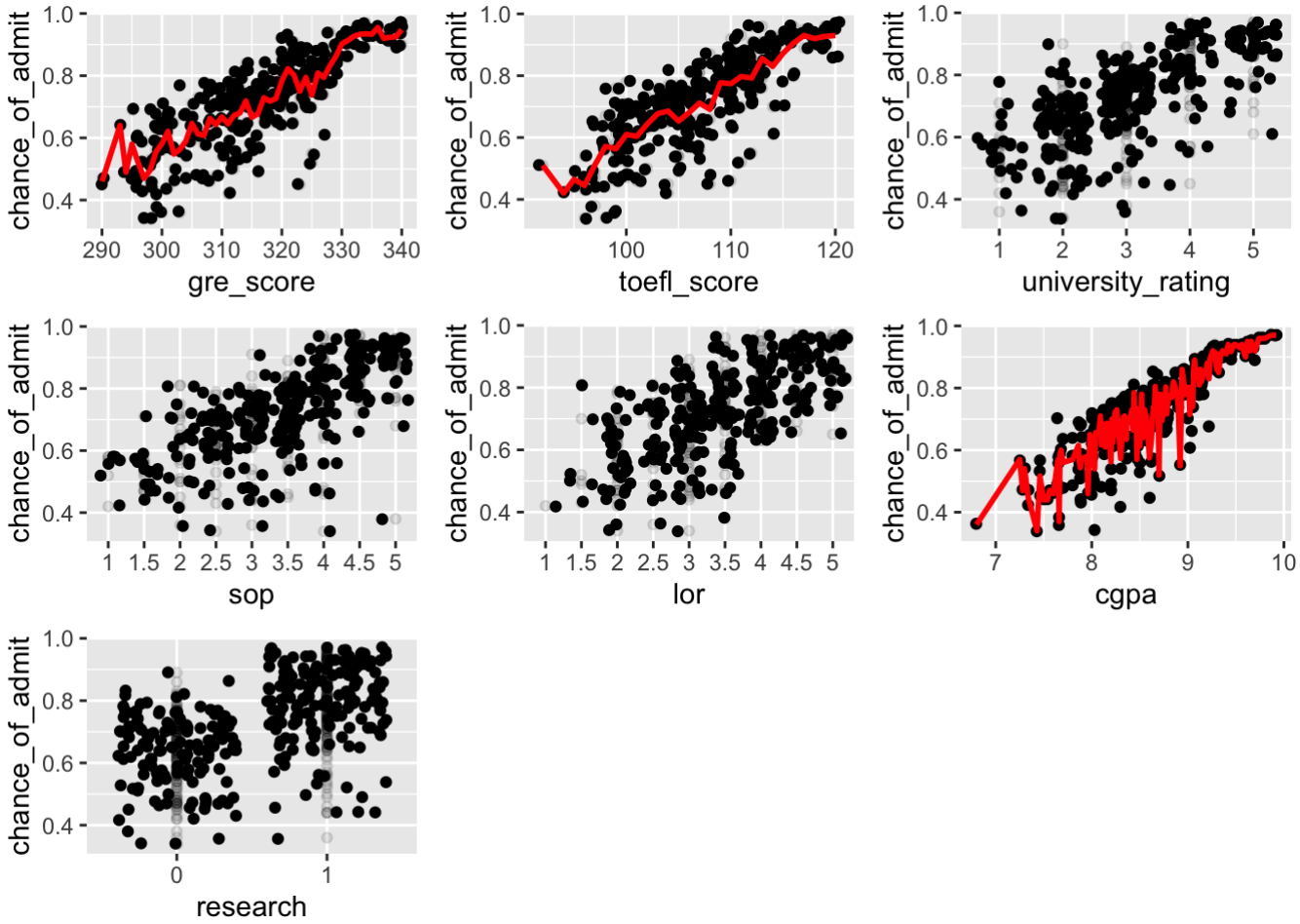
The number is correct: 298+102 = 400

# Exploratory Data Analysis

This entire EDA process will be based on the training set.

My strategy here is to generate plots of each predictors with respect to the `chance of admit`. The purpose here is to find each of their correlation with the `chance of admit`.

Code

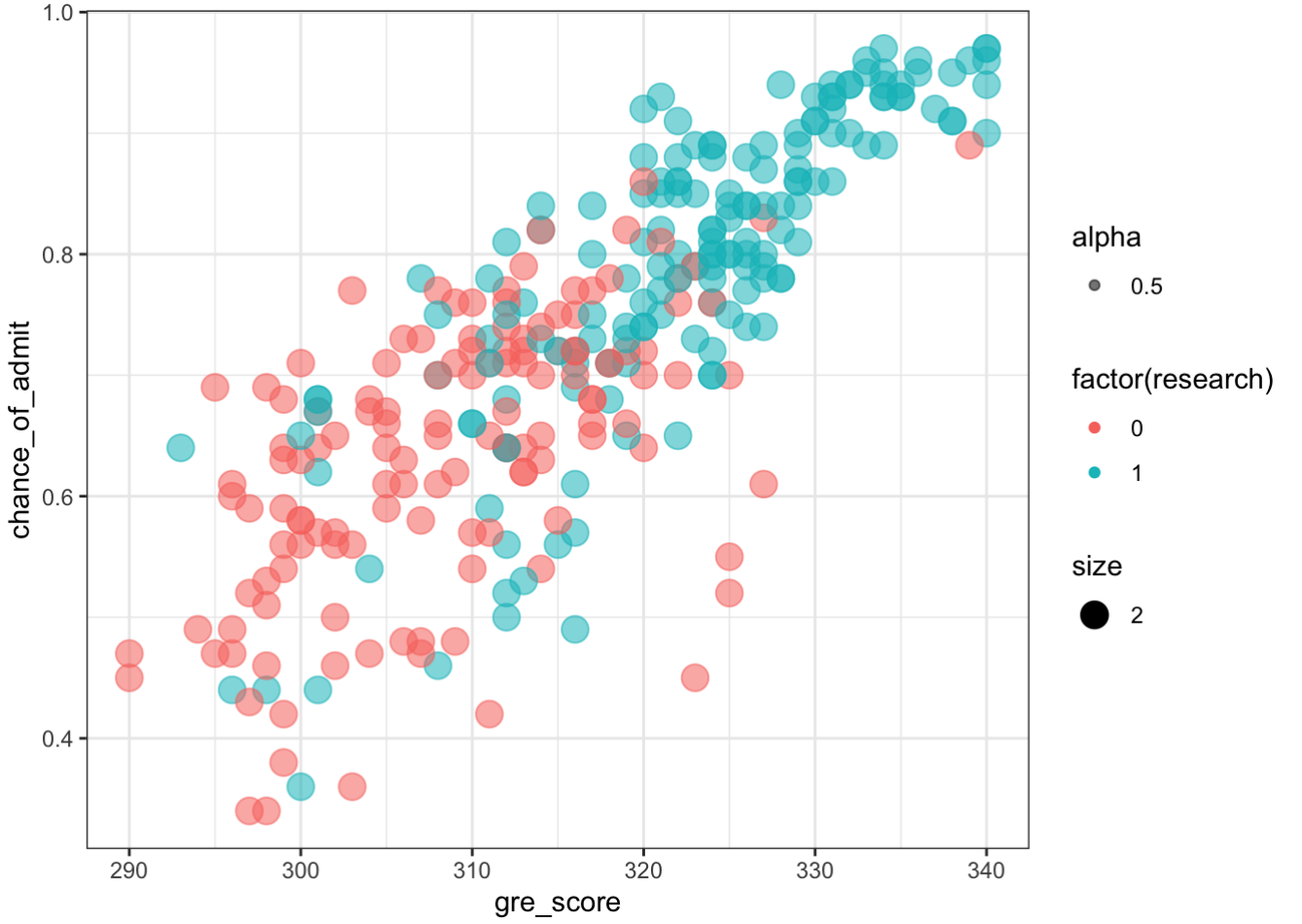



From the trend of these seven charts, they all have positively associated with chance of admissions. And my intuitive feeling is that GRE, TOEFL and CGPA are the most associated with admission chances respectively, showing highly strong correlation, because their data points are more dense and concentrated.

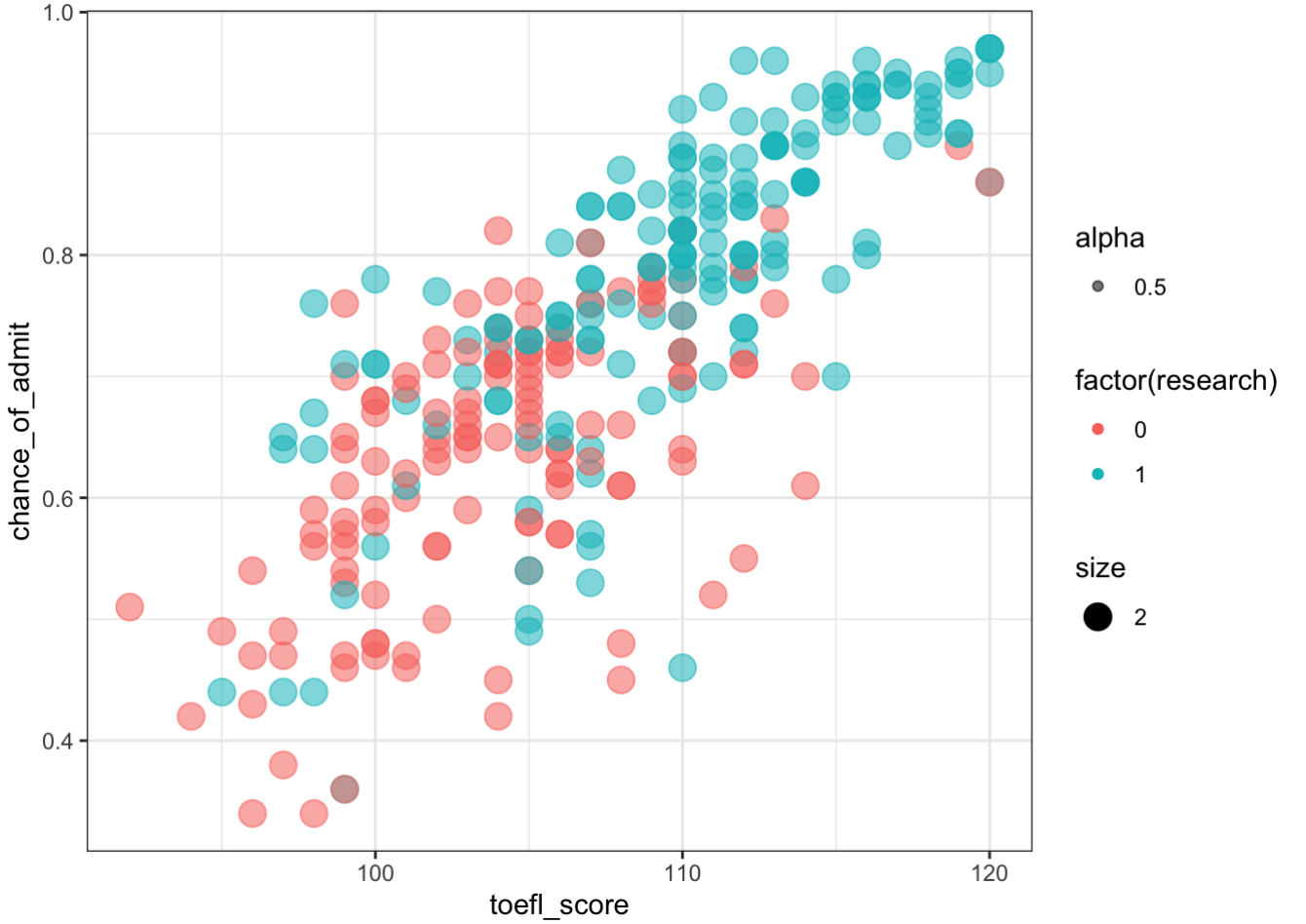
For the record, based on my application experience that three predictors, GRE, TOEFL and GPA will be especially useful. It also makes sense since universities need to use standardized tests and four years of academic performance as hard indicators to quickly screen out outstanding applicants.

Let's create more plots to further illustrate these seven variables. This part was originally illustrated by boxplot, but boxplot is not actually a good tool to visualize categorical variable. Hence I change them to scatter plots for three numerical variable (GRE, TOEFI, and GPA), differentiating by research experience.

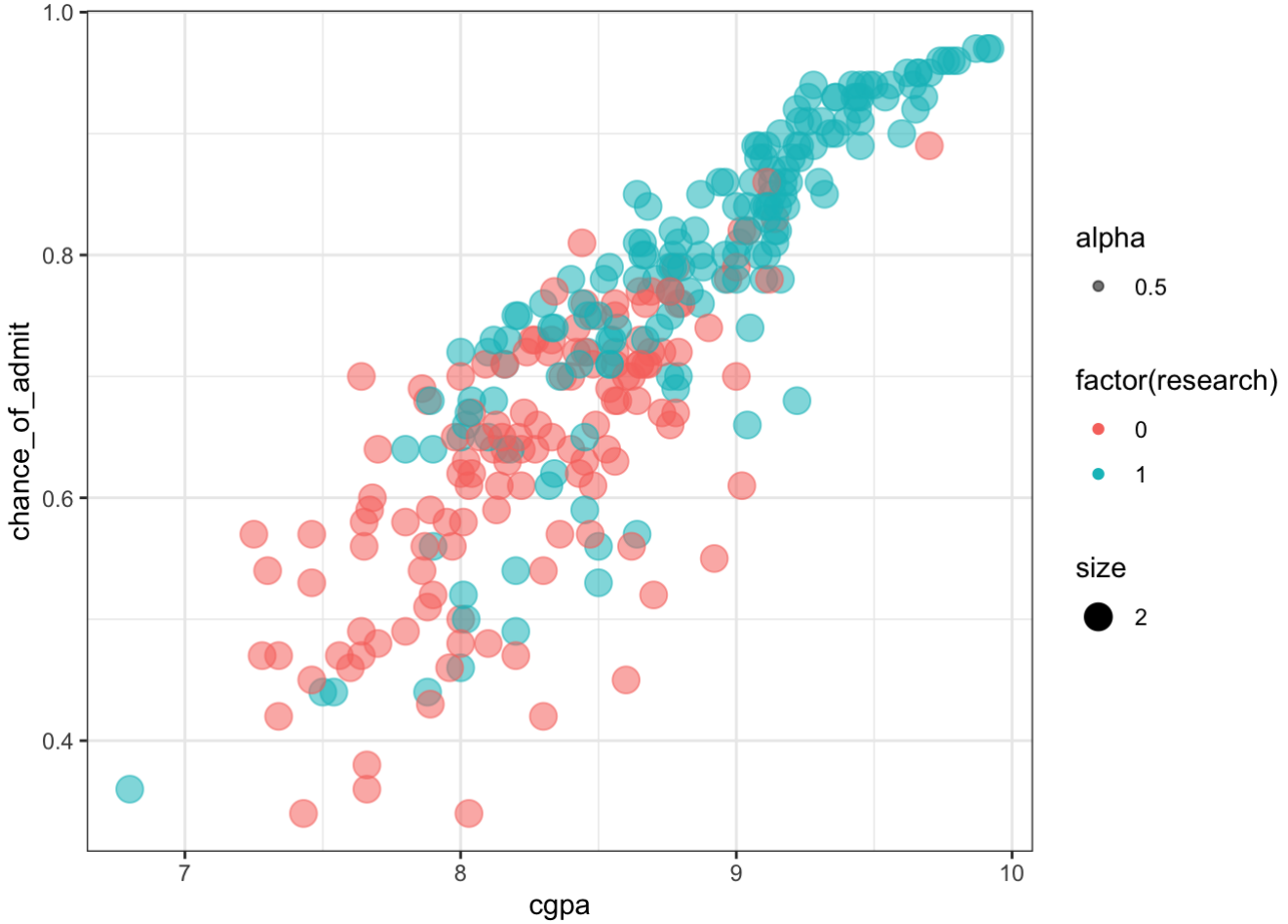
Code


1.0
0.8
0.6
0.4
chance_of_admit
290
300
310
320
330
340
gre_score
alpha
0.5
factor(research)
0
1
size
2

Code

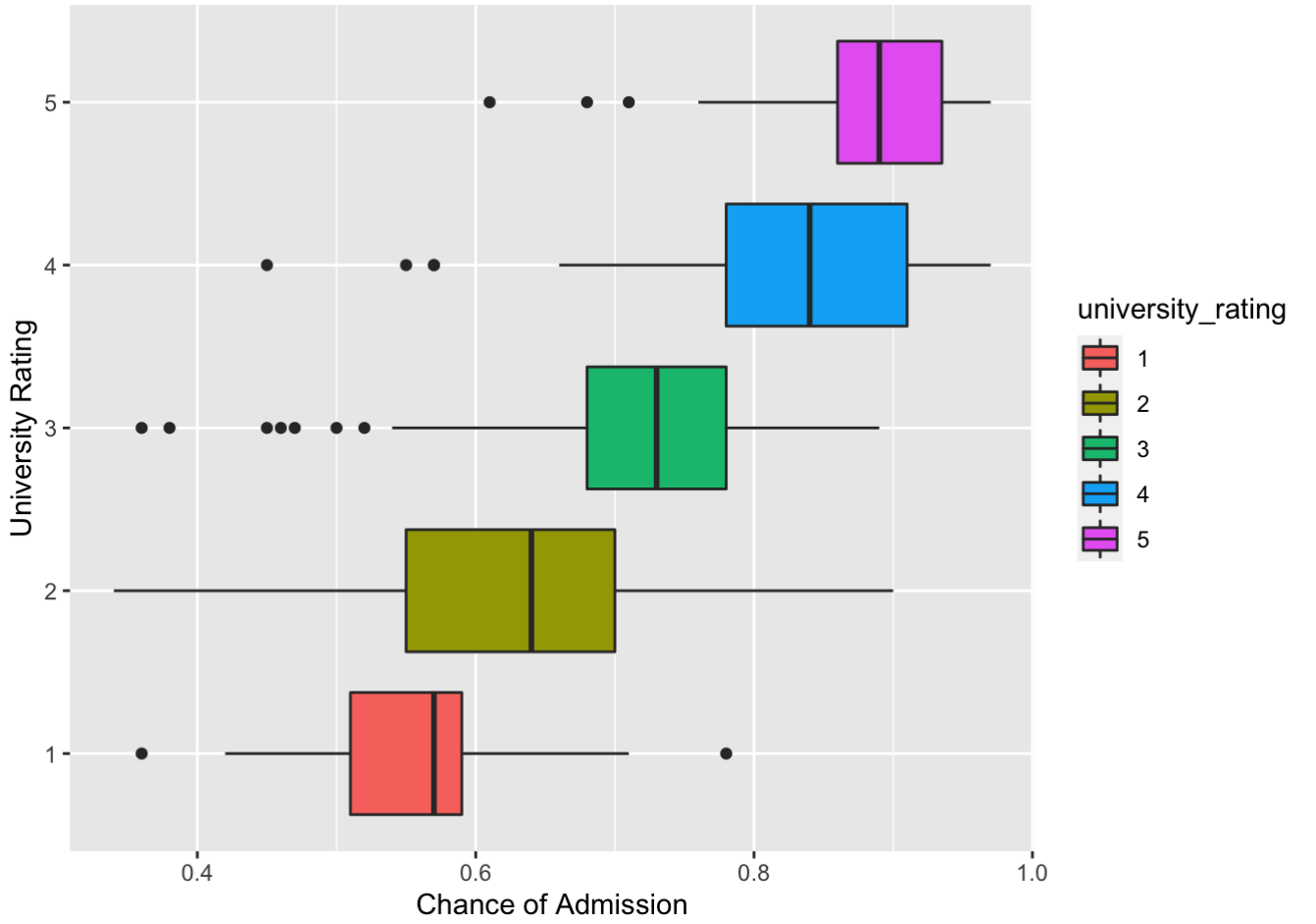

1.0
0.8
0.6
0.4
chance_of_admit
100
110
120
toefl_score
alpha
0.5
factor(research)
0
1
size
2

Code

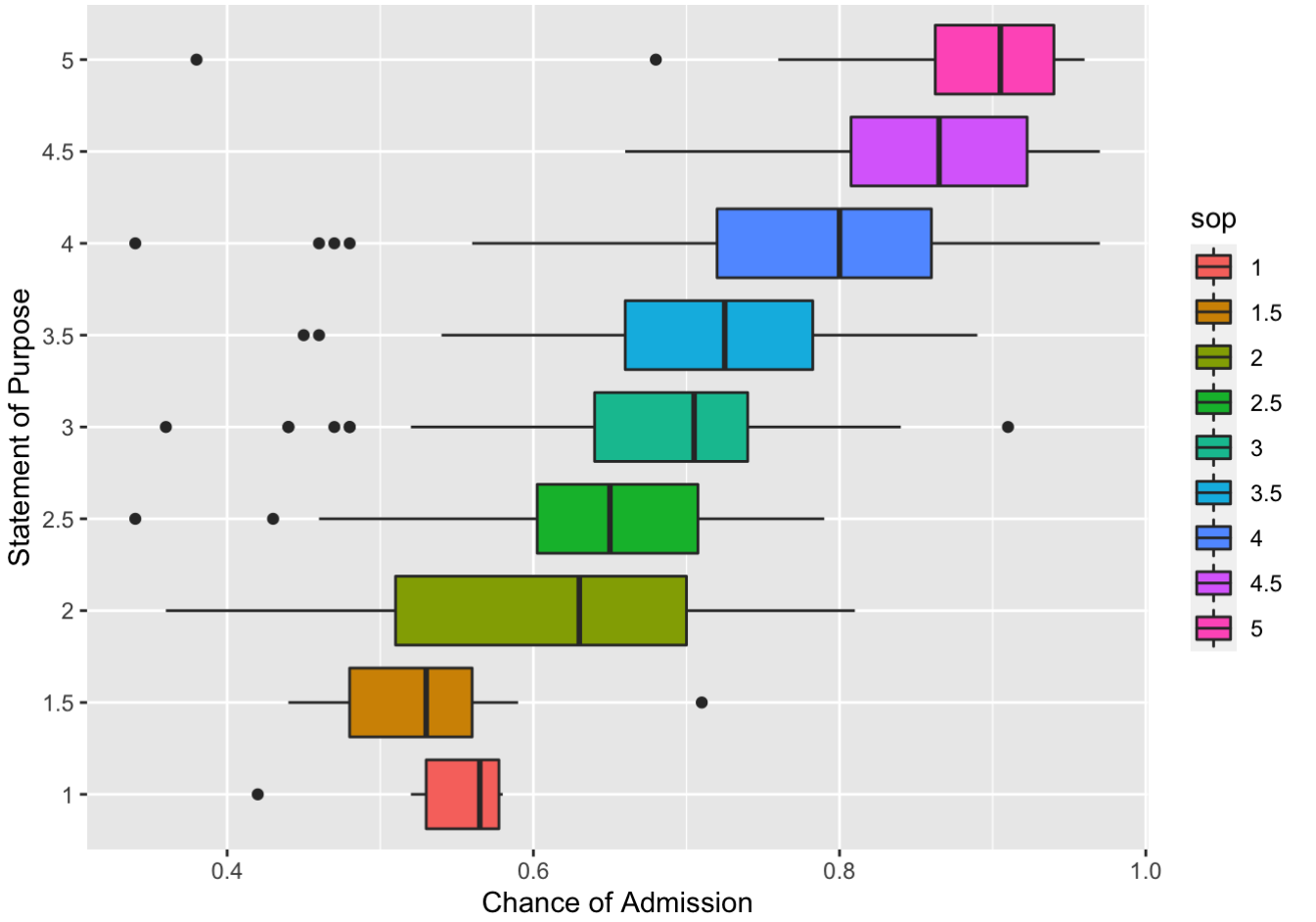



We can find out that high GRE(>320), TOEFL(>110), and CGPA(>9) combined with the research experience represented by blue dot will greatly increase the chances of getting accepted.

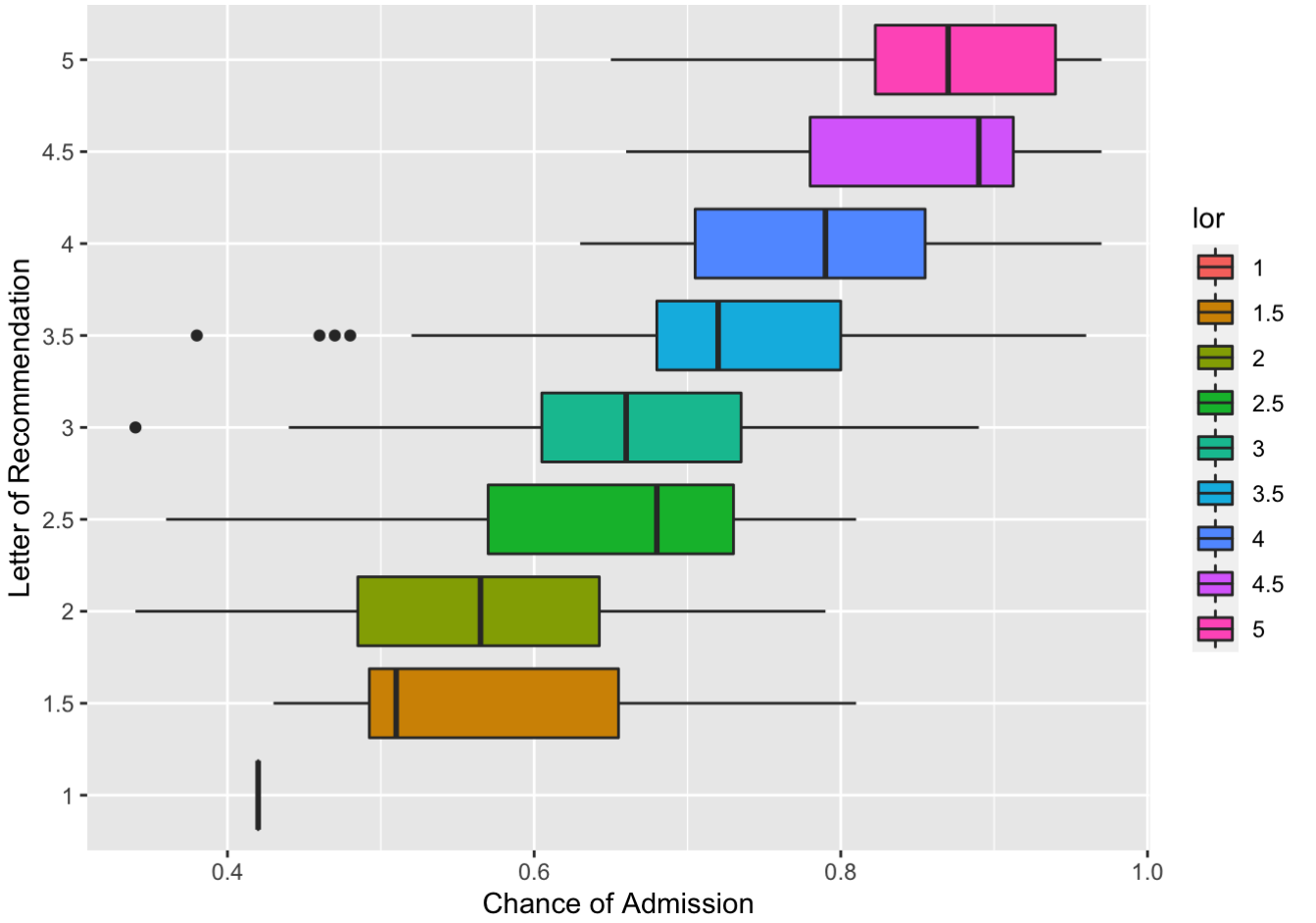
Code


University Rating
5
4
3
2
1
0.4
0.6
0.8
1.0
Chance of Admission
university_rating
1
2
3
4
5

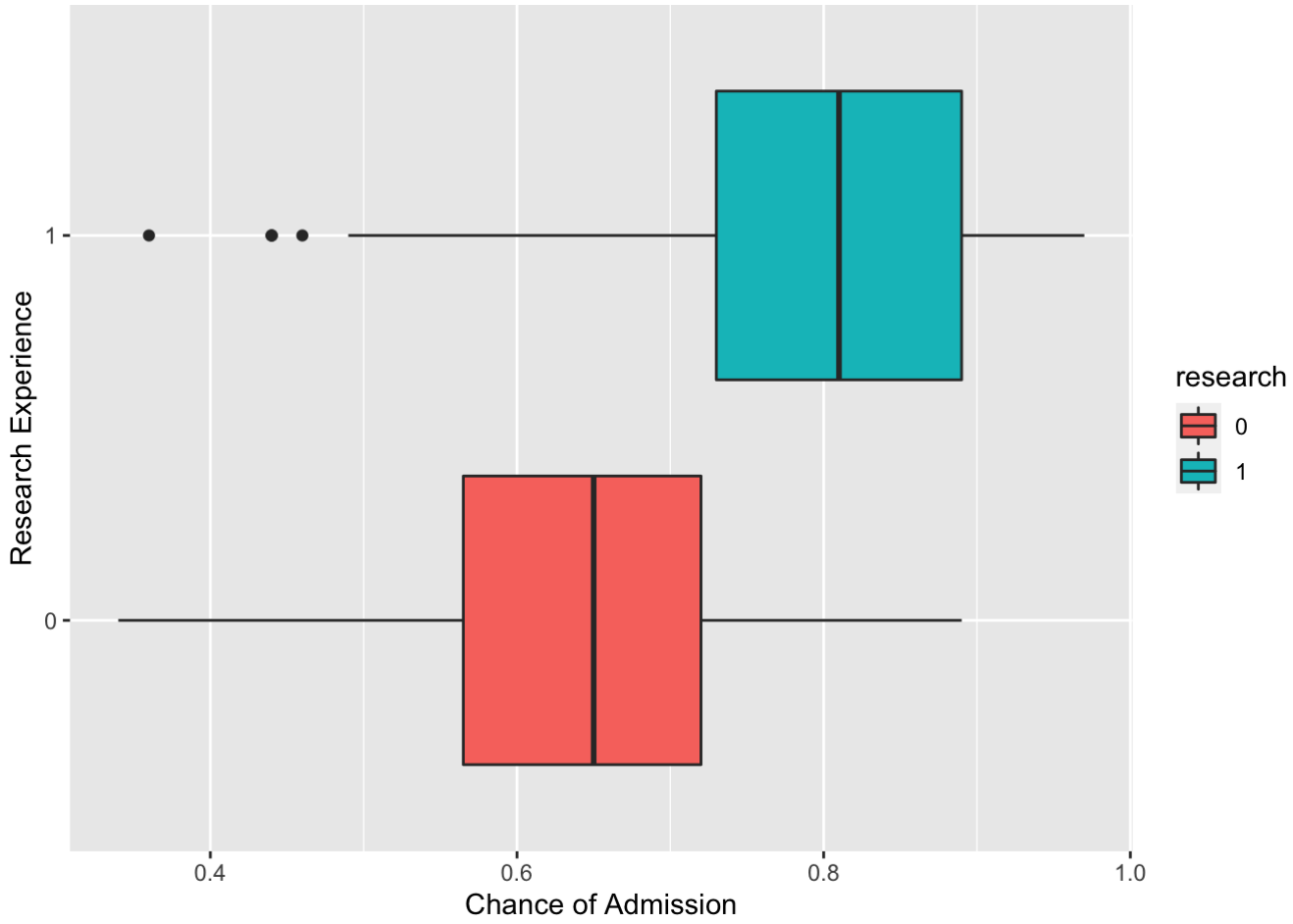
Code


Statement of Purpose
5
4.5
4
3.5
3
2.5
2
1.5
1
0.4
0.6
0.8
1.0
Chance of Admission
sop
1
1.5
2
2.5
3
3.5
4
4.5
5

Code


Letter of Recommendation
5
4.5
4
3.5
3
2.5
2
1.5
1
0.4
0.6
0.8
1.0
Chance of Admission
lor
1
1.5
2
2.5
3
3.5
4
4.5
5

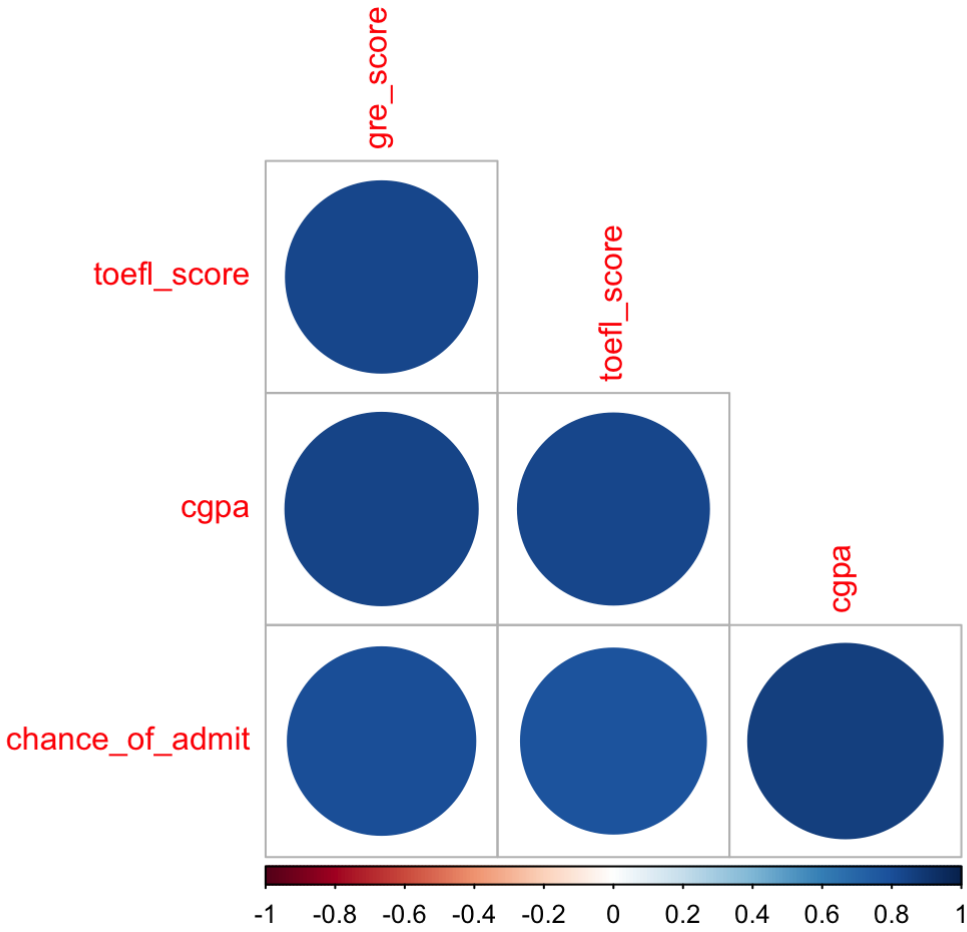
Code




Once again, we could easily detect the patterns embedded in each plots that as the number increases (means more competitive applicant), the chance of admission tends to increase.

Then, let's create a visualization of correlation matrix of the training set and see what we got.

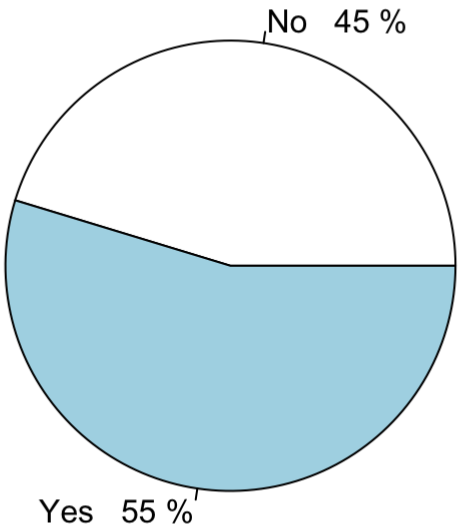
Code



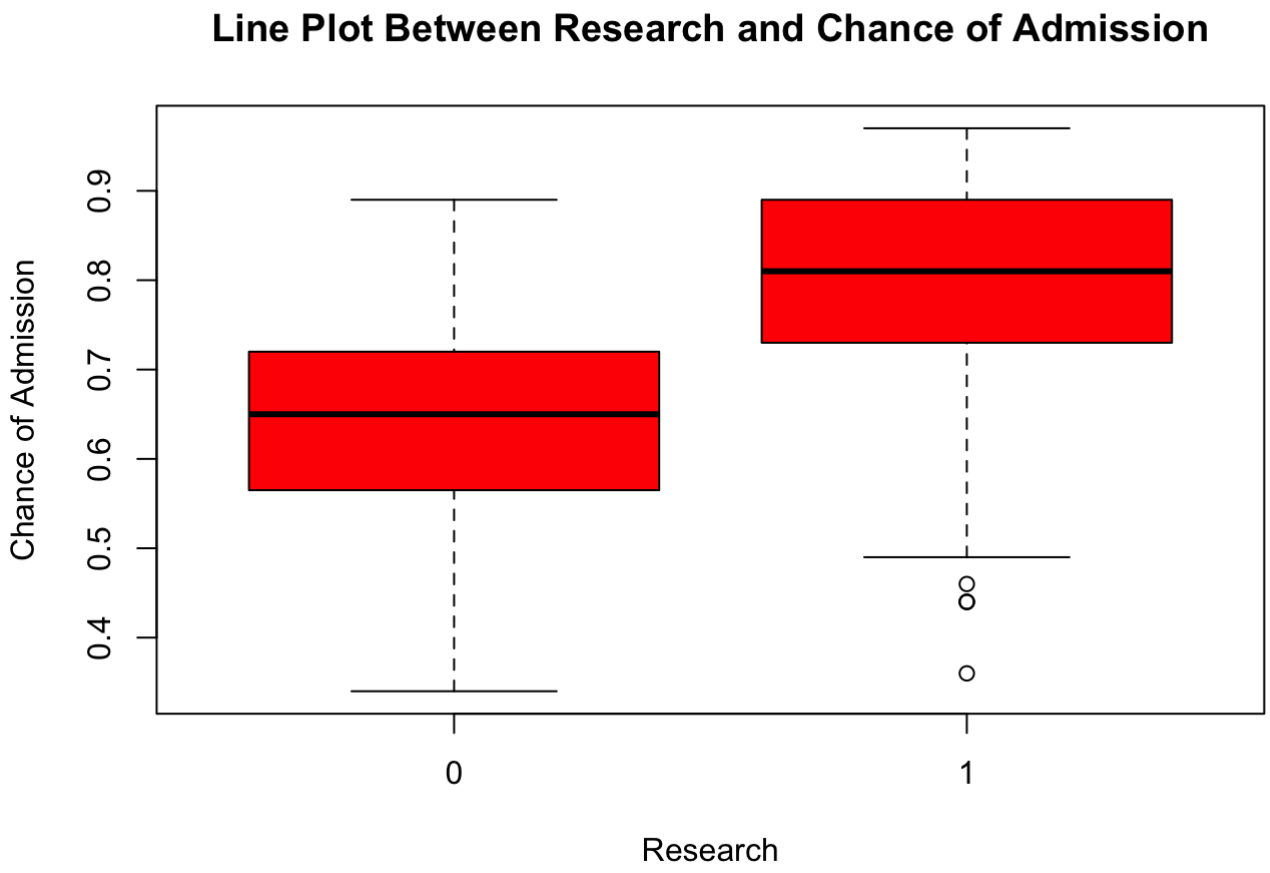

This makes it even clearer that, in addition to three numeric variables we analyzed before, the chance of admission are also highly positively related to the remaining 3 variables (university rating, sop and lor) except research experience (this one is bit weak). Therefore, I am going to include these 3 into the model.

Hence, let's focus on exploring if having research experience can bring some edges for applications admission. If it can, I will include it, and vice versa.

Code




Code

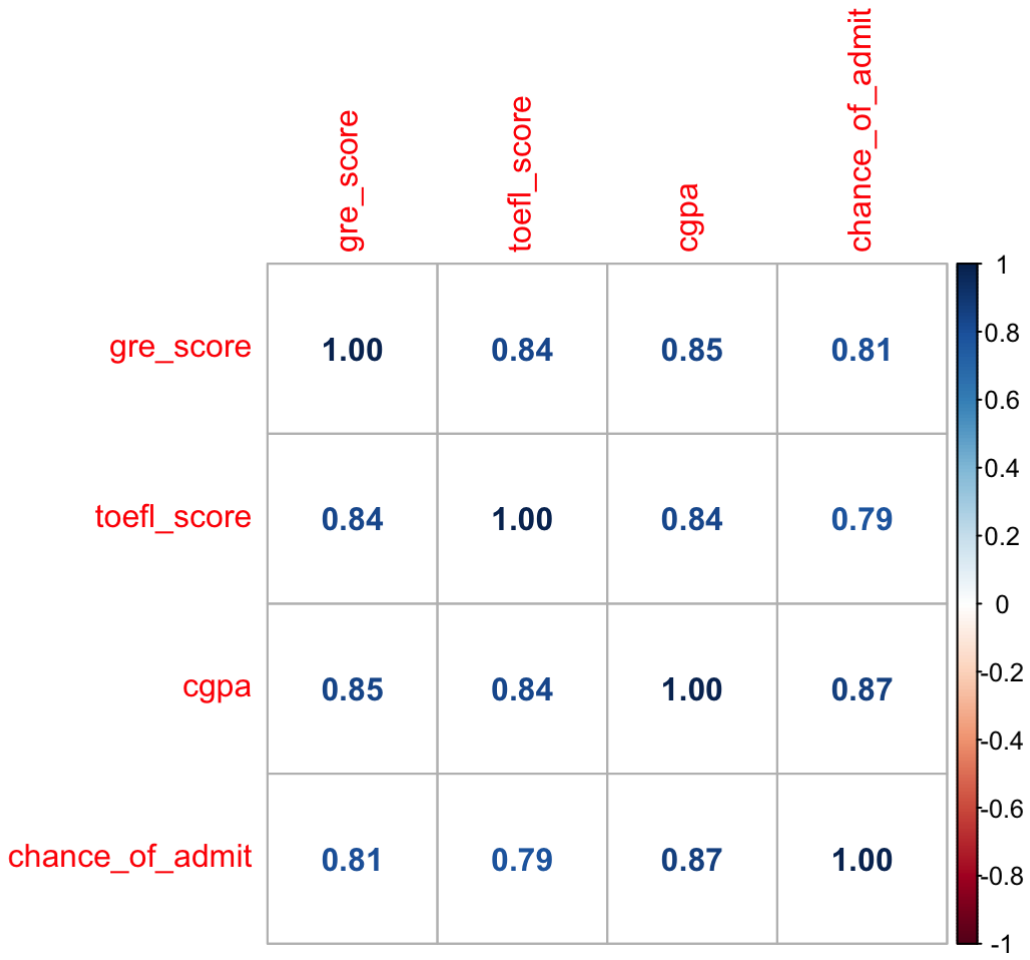



There are 55% of the student had research experience during their Undergraduate and if we take a look at the second plots, we know that applicants who conducted research before could bring them some edges to get admission, and thus I will include research into my model as a predictor.

In all, this dataset is well-organized， carefully selected， and cleaned. It seems every each variables exist reasonably in the table and all has a positive relationship with the response variable.

# Create Interactions?

Code




I replaced the `method = number` to better quantify the connection between each variable to determine if I needed to cover the interactions in my recipe. And from above correlation matrix we could detect that all numeric variables have strong positive correlations with each other.

# Building Model

Since my outcome, response variable `chance of admit` , is continuous, I believe my question would be best answered in regression approach.The models I would like to build are Polynomial Regression, Lasso Regression, Ridge Regression, Random Forest, and Boosted Trees. In the end, I will compare each model's performance and pick the best-performing one to fit the testing set.

First thing first, create my recipe:

1. predict chance of admit by all seven predictors. three variables are numerical and the rest of them are factored.

2. use `step_dummy` to dummy code any categorical predictors.

3. use `step_novel` to dealwith the factor variables.

4. use `step_normalize` to center and scale all predictors.

5. use `step_zv` to remove variables that contain only a single value.

6. Since all the numeric predictors are strongly correlated, it might make more sense to do a `step_pca()` than to include interaction terms.

Code

```
## Recipe
##
## Inputs:
##
##       role #variables
##    outcome          1
##  predictor          7
##
## Operations:
##
## Novel factor level assignment for all_nominal_predictors()
## Dummy variables from all_nominal_predictors()
## PCA extraction with all_numeric_predictors()
## Zero variance filter on all_predictors()
## Centering and scaling for all_predictors()
```

# k-fold Cross-Validation

This can be done using the vfold_cv() function. Common choices for k/v are 5 or 10. Here, I use 10 folds. And I stratify the folds by chance_of_admit.

Stratification is the process of rearranging the data as to ensure each fold is a good representative of the whole. For example in a binary classification problem where each class comprises 50% of the data, it is best to arrange the data such that in every fold, each class comprises around half the instances.

Code

```
## #  10-fold cross-validation using stratification
## # A tibble: 10 × 2
##    splits           id
##    <list>           <chr>
##  1 <split [266/32]> Fold01
##  2 <split [266/32]> Fold02
##  3 <split [267/31]> Fold03
##  4 <split [267/31]> Fold04
##  5 <split [268/30]> Fold05
##  6 <split [269/29]> Fold06
##  7 <split [269/29]> Fold07
##  8 <split [270/28]> Fold08
##  9 <split [270/28]> Fold09
## 10 <split [270/28]> Fold10
```

# Polynomial Regression

Next, fit a polynomial regression model. We can use the linear model specification `lm_spec` to add a preprocessing unit with `recipe()` and `step_poly()`.

suppose we want to find the best value of degree that yields the “closest” fit. This is known as hyperparameter tuning, and it is a case where we can use k-Fold Cross-Validation.

First, I need to use `lm()` to check which variable has significant correlation with our response variable with degree = 2 and then, I will use this variable in the `step_poly()`.

Code

```
##
## Call:
## lm(formula = chance_of_admit ~ gre_score^2 + toefl_score^2 +
##     university_rating^2 + sop^2 + lor^2 + cgpa^2 + research^2,
##     data = AD_train)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.246500 -0.026039  0.009889  0.037402  0.182191
##
## Coefficients:
##                      Estimate Std. Error t value Pr(>|t|)
## (Intercept)        -1.2913314  0.1723086  -7.494 9.32e-13 ***
## gre_score           0.0019659  0.0007712   2.549  0.01134 *
## toefl_score         0.0025182  0.0013804   1.824  0.06920 .
## university_rating2 -0.0258663  0.0194317  -1.331  0.18425
## university_rating3 -0.0119863  0.0207618  -0.577  0.56419
## university_rating4 -0.0217565  0.0248296  -0.876  0.38167
## university_rating5 -0.0020143  0.0268859  -0.075  0.94033
## sop1.5              0.0002817  0.0363593   0.008  0.99382
## sop2                0.0106342  0.0350902   0.303  0.76208
## sop2.5              0.0174574  0.0355503   0.491  0.62378
## sop3                0.0115301  0.0359989   0.320  0.74899
## sop3.5              0.0007603  0.0366590   0.021  0.98347
## sop4               -0.0051001  0.0375937  -0.136  0.89219
## sop4.5              0.0081933  0.0394460   0.208  0.83561
## sop5                0.0088405  0.0405480   0.218  0.82757
## lor1.5              0.0141544  0.0776204   0.182  0.85544
## lor2                0.0544130  0.0747355   0.728  0.46719
## lor2.5              0.0714057  0.0748508   0.954  0.34094
## lor3                0.0530584  0.0753190   0.704  0.48175
## lor3.5              0.0768206  0.0756697   1.015  0.31090
## lor4                0.0953375  0.0759864   1.255  0.21067
## lor4.5              0.1000959  0.0767974   1.303  0.19354
## lor5                0.1127360  0.0772010   1.460  0.14536
## cgpa                0.1208181  0.0158326   7.631 3.91e-13 ***
## research1           0.0274643  0.0097865   2.806  0.00537 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.06718 on 273 degrees of freedom
## Multiple R-squared:  0.8002, Adjusted R-squared:  0.7826
## F-statistic: 45.56 on 24 and 273 DF,  p-value: < 2.2e-16
```

Seems like our winner is `cgpa`. Put it into my recipe and right now is the time to get my specified the value of degree.

Code

Next, we can specify the model engine that we want to fit, and then set up the workflow.

Code

```
## ═══ Workflow ═══════════════════════════════════════════════════════════════
═══════════════════════════════════════
## Preprocessor: Recipe
## Model: linear_reg()
##
## ─── Preprocessor ───────────────────────────────────────────────────────────
──────────────────────────────────────
## 6 Recipe Steps
##
## • step_poly()
## • step_novel()
## • step_dummy()
## • step_pca()
## • step_zv()
## • step_normalize()
##
## ─── Model ──────────────────────────────────────────────────────────────────
──────────────────────────────────────
## Linear Regression Model Specification (regression)
##
## Computational engine: lm
```

The next thing we need is a tibble of possible values we want to explore.

Code

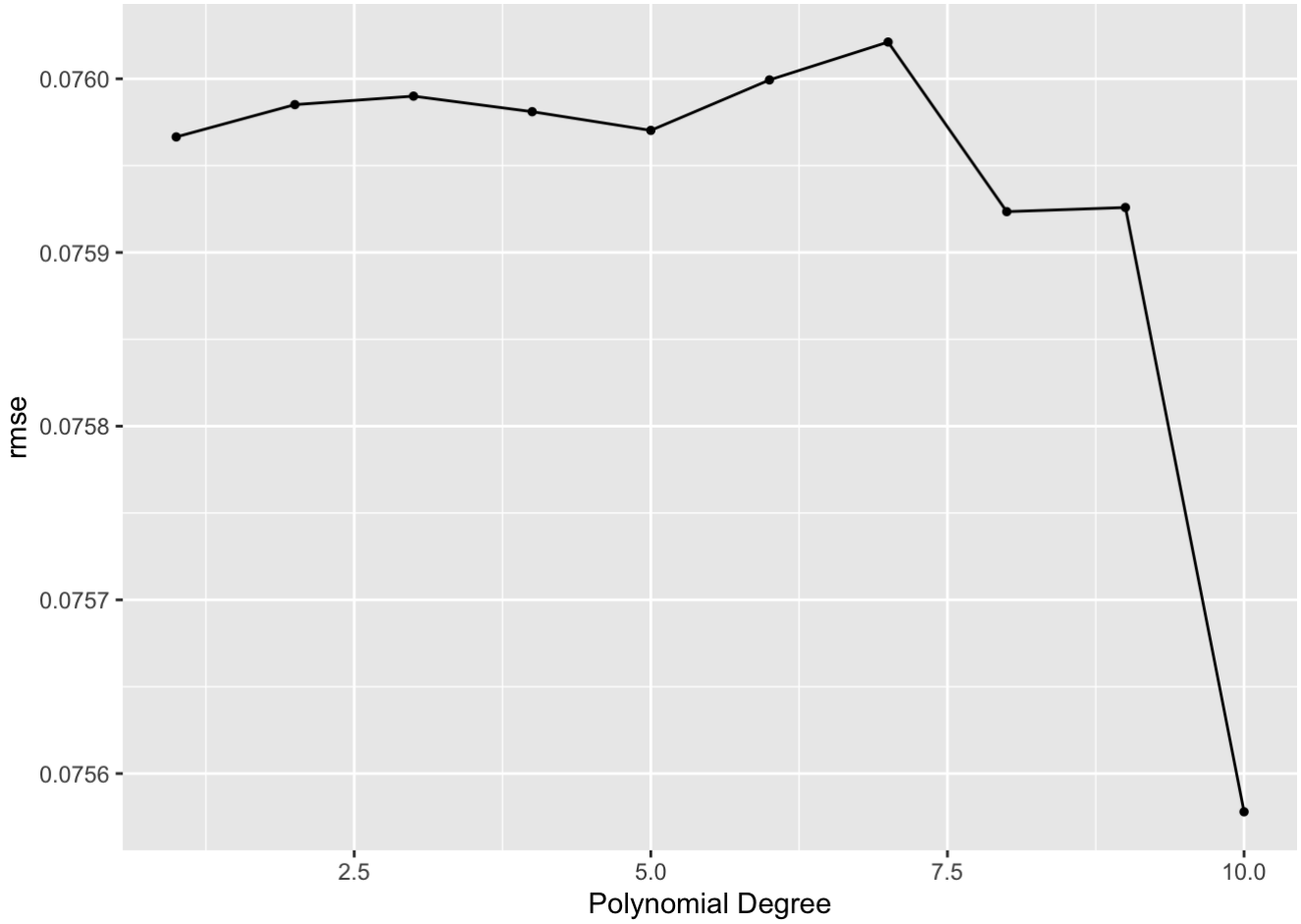
Now that all the necessary objects have been created, we can pass them to tune_grid(), which will fit the models within each fold for each value specified in degree_grid.

Code

```
## # Tuning results
## # 10-fold cross-validation using stratification
## # A tibble: 10 × 4
##    splits           id     .metrics          .notes
##    <list>           <chr>  <list>            <list>
##  1 <split [266/32]> Fold01 <tibble [10 × 5]> <tibble [0 × 3]>
##  2 <split [266/32]> Fold02 <tibble [10 × 5]> <tibble [0 × 3]>
##  3 <split [267/31]> Fold03 <tibble [10 × 5]> <tibble [0 × 3]>
##  4 <split [267/31]> Fold04 <tibble [10 × 5]> <tibble [0 × 3]>
##  5 <split [268/30]> Fold05 <tibble [10 × 5]> <tibble [0 × 3]>
##  6 <split [269/29]> Fold06 <tibble [10 × 5]> <tibble [0 × 3]>
##  7 <split [269/29]> Fold07 <tibble [10 × 5]> <tibble [0 × 3]>
##  8 <split [270/28]> Fold08 <tibble [10 × 5]> <tibble [0 × 3]>
##  9 <split [270/28]> Fold09 <tibble [10 × 5]> <tibble [0 × 3]>
## 10 <split [270/28]> Fold10 <tibble [10 × 5]> <tibble [0 × 3]>
```

Create an visual overview of the performance of different hyperparameter pairs

Code




Let's see the best-performing tuned degree.

Code

```
## # A tibble: 1 × 7
##   degree .metric .estimator   mean     n std_err .config
##    <dbl> <chr>   <chr>       <dbl> <int>   <dbl> <chr>
## 1     10 rmse    standard   0.0756    10 0.00326 Preprocessor10_Model1
```

Okay, the best tuned degree is 1. Use `select_best()` to choose the model that has the optimal `rmse` to fit the linear model to the training set and view the model results

Code

```
## # A tibble: 1 × 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard      0.0752
```

# Ridge Regression

Using `linear_reg()` and setting `mixture = 0` to specify a ridge model. When using the glmnet engine, I also need to tune the parameter `penalty` to find the best value to fit the model.

Code

Create my recipe that predicting the outcome variable, `chance_of_admit` , with all other predictor variables. And then create a workflow, adding the model and my recipe in it.

Code

Right now I need is the values of penalty I am trying. This can be created using grid_regular(), which creates a grid of evenly spaced parameter values. I follow the lab example to set the range and level.

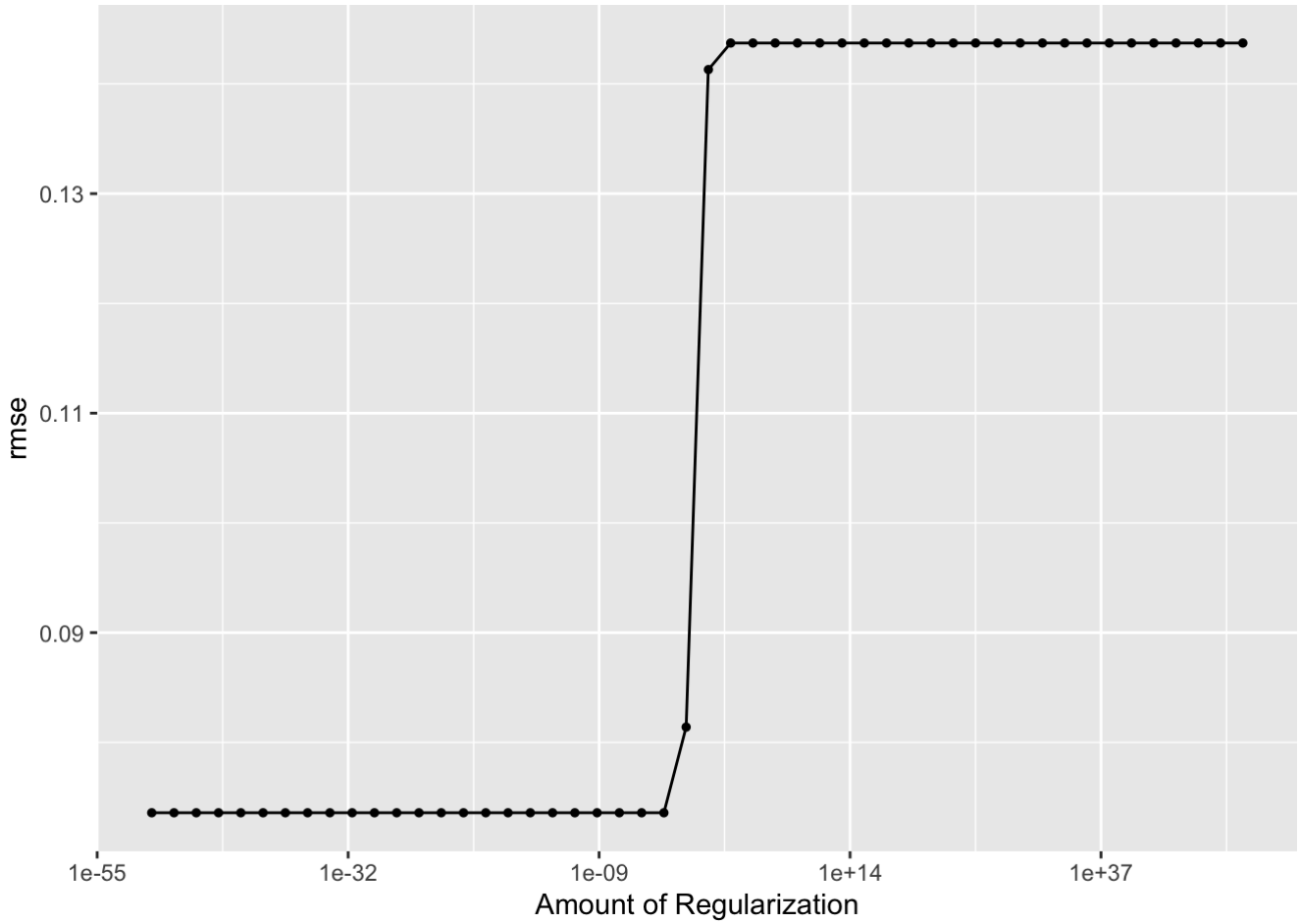
Although using 50 levels for one parameter might seem overkill. But glmnet fits all the models in one go, so adding more levels to penalty doesn't affect the computational speed much for linear or logistic regression.

Code

```
## # Tuning results
## # 10-fold cross-validation using stratification
## # A tibble: 10 × 4
##    splits           id     .metrics          .notes
##    <list>           <chr>  <list>            <list>
##  1 <split [266/32]> Fold01 <tibble [50 × 5]> <tibble [0 × 3]>
##  2 <split [266/32]> Fold02 <tibble [50 × 5]> <tibble [0 × 3]>
##  3 <split [267/31]> Fold03 <tibble [50 × 5]> <tibble [0 × 3]>
##  4 <split [267/31]> Fold04 <tibble [50 × 5]> <tibble [0 × 3]>
##  5 <split [268/30]> Fold05 <tibble [50 × 5]> <tibble [0 × 3]>
##  6 <split [269/29]> Fold06 <tibble [50 × 5]> <tibble [0 × 3]>
##  7 <split [269/29]> Fold07 <tibble [50 × 5]> <tibble [0 × 3]>
##  8 <split [270/28]> Fold08 <tibble [50 × 5]> <tibble [0 × 3]>
##  9 <split [270/28]> Fold09 <tibble [50 × 5]> <tibble [0 × 3]>
## 10 <split [270/28]> Fold10 <tibble [50 × 5]> <tibble [0 × 3]>
```

Visualize the tuning result

Code




The “best” values of this can be selected using select_best() and set the `rmse` as performance metrics for evaluation.

Code

```
## # A tibble: 1 × 2
##   penalty .config
##     <dbl> <chr>
## 1   1e-50 Preprocessor1_Model01
```

Using finalize_workflow() to update the recipe by replacing tune() with the value of best_penalty.

Code

Now, fit this best model again, using the whole training data set. Finally, let’s check its performance.

Code

```
## # A tibble: 1 × 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard      0.0731
```

# Lasso Regression

This recipe is the same as the ridge_recipe. I just changed its variable name.

Use the glmnet package to perform lasso linear regression. For a linear lasso regression, I need to use linear_reg() and set mixture = 1 to specify a lasso model. And then, combine them into the workflow.

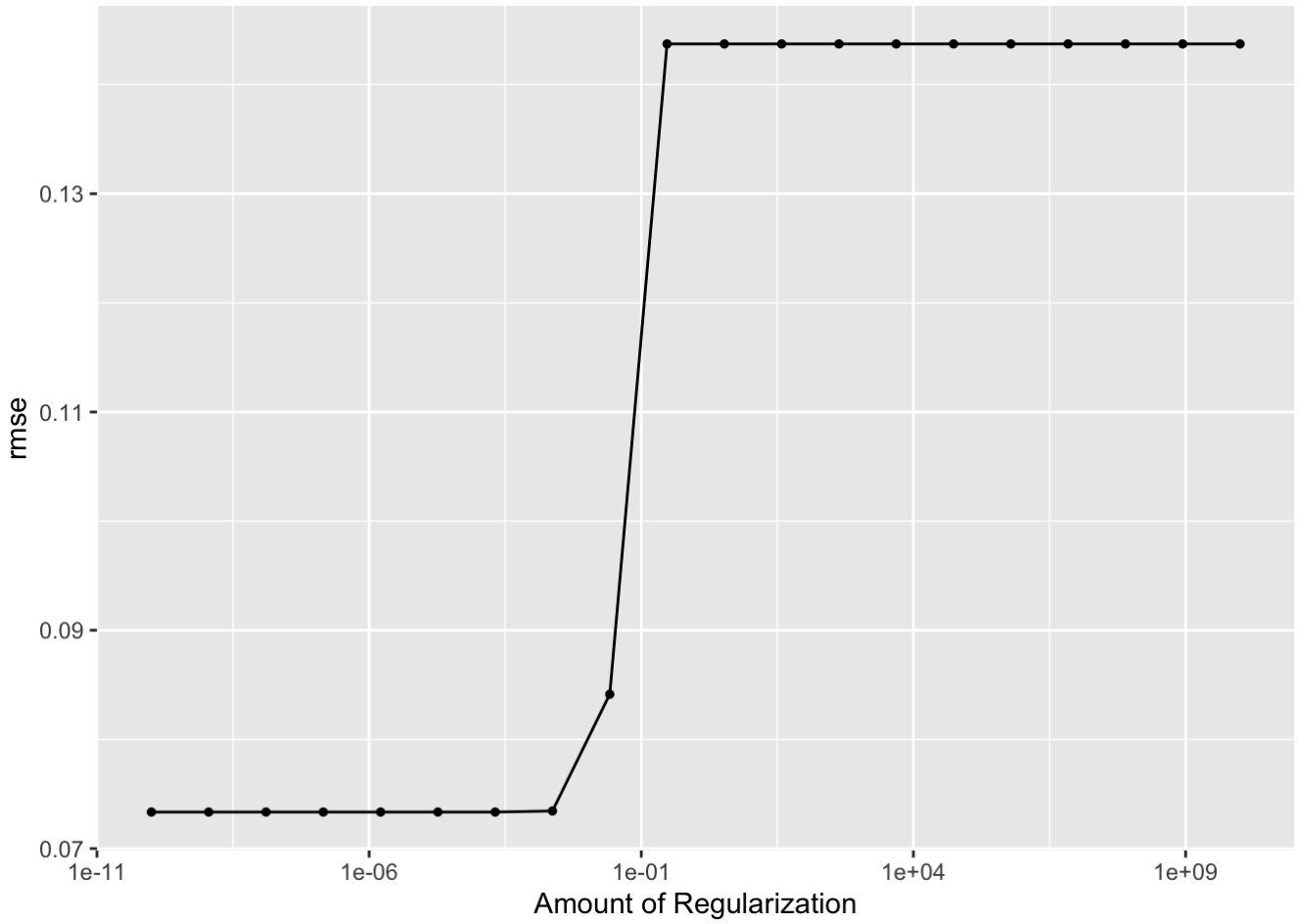
Code

The procedure will be very similar to the ridge regression section. The preprocessing needed is the same. For the penalty range, I will from (-100, 100) to (-10, 10), the best model is the same one.

Code

```
## # Tuning results
## # 10-fold cross-validation using stratification
## # A tibble: 10 × 4
##    splits           id     .metrics          .notes
##    <list>           <chr>  <list>            <list>
##  1 <split [266/32]> Fold01 <tibble [20 × 5]> <tibble [0 × 3]>
##  2 <split [266/32]> Fold02 <tibble [20 × 5]> <tibble [0 × 3]>
##  3 <split [267/31]> Fold03 <tibble [20 × 5]> <tibble [0 × 3]>
##  4 <split [267/31]> Fold04 <tibble [20 × 5]> <tibble [0 × 3]>
##  5 <split [268/30]> Fold05 <tibble [20 × 5]> <tibble [0 × 3]>
##  6 <split [269/29]> Fold06 <tibble [20 × 5]> <tibble [0 × 3]>
##  7 <split [269/29]> Fold07 <tibble [20 × 5]> <tibble [0 × 3]>
##  8 <split [270/28]> Fold08 <tibble [20 × 5]> <tibble [0 × 3]>
##  9 <split [270/28]> Fold09 <tibble [20 × 5]> <tibble [0 × 3]>
## 10 <split [270/28]> Fold10 <tibble [20 × 5]> <tibble [0 × 3]>
```

Code



Code

```
## # A tibble: 1 × 2
##        penalty .config
##          <dbl> <chr>
## 1 0.0000000001 Preprocessor1_Model01
```

Then, same as what I did for ridge regression, finalize the workflow and apply the best penalty value to fit the model with training set.

Code

let's view the performance.

Code

```
## # A tibble: 1 × 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard      0.0728
```

Also, both poly, ridge, and lasso regressions have given us nearly identical results. Hence, it seems that the linear model cannot get past a RMSE of 0.06 .

Let us look at some of the regressions that could yield better results.

# Random Forest

Set mode to "regression" (outcome is a numeric variable), and used the ranger engine. I stored this model and my recipe in a workflow.

Code

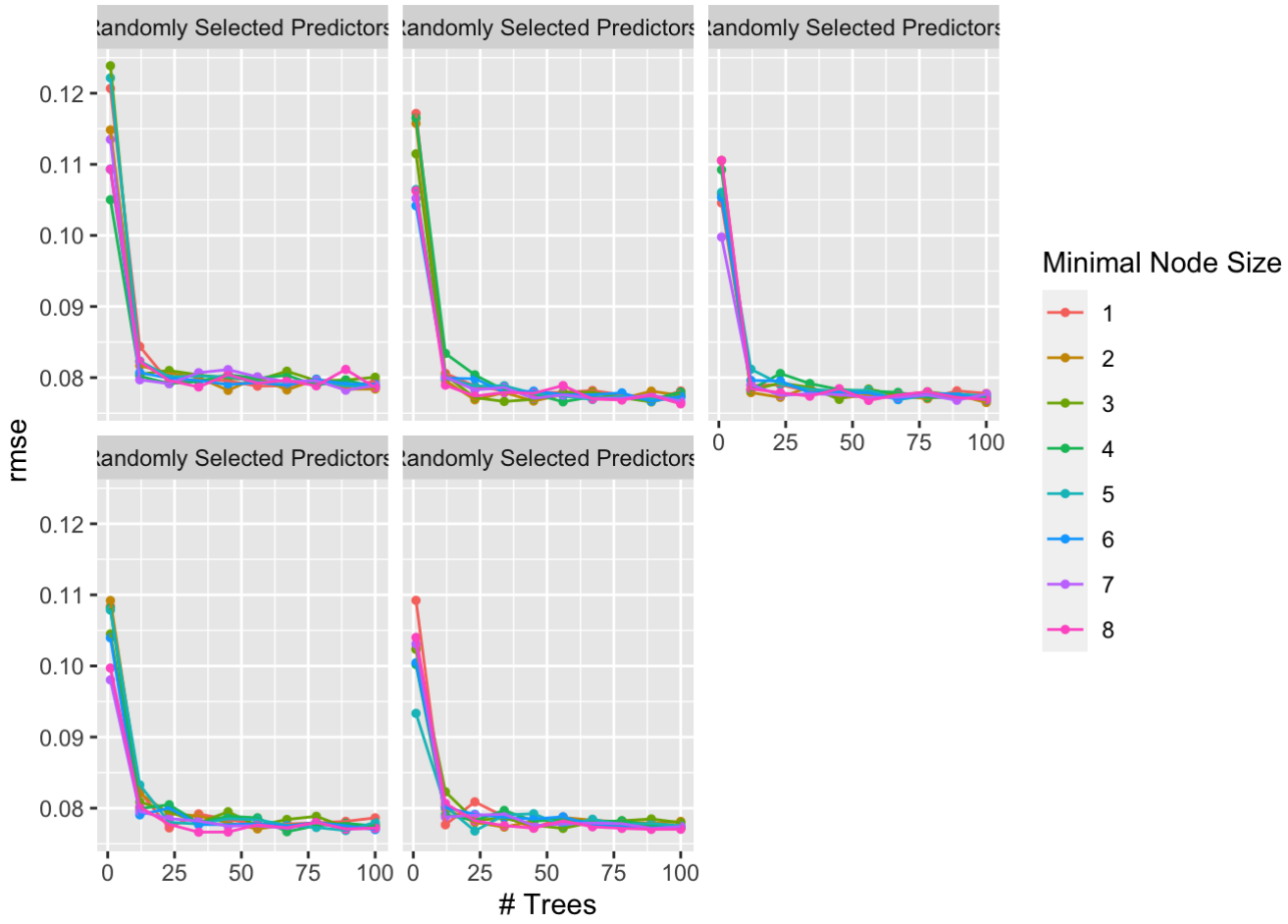
Here comes the step of tunning grid, the three parameters I need to tune is mtry, trees, and min_n. Note that the upper range of mtry cannot go larger than 7 because we only have 7 predictors here. Also, because my dataset is relatively small, so setting level = 10 will not be an issue on computing power.

Code

Taking a swift glimpse at the visualized result, we can see that `rmse` decrease since the beginning and trend down all the way until it reach around 12 randomly selected predictors, and since there, the overall trend of `rmse` is in a steady state without excessive ups and downs. This makes sense because as the

This makes sense because at first, as the number of randomly selected predictors increases, it means that there is a greater chance of getting the `chance_of_admit` right. However, as the amount of data increases, the results of the predicted admissions odds fluctuate slightly but do not change much. This is probably what we call a program's metaphysical admissions (which can vary from the expected results) and bar (which is relatively consistent in overall admissions standard).

Code




Using `collect_metrics` to view the best model with tunning parameters

Code

```
## # A tibble: 1 × 9
##    mtry trees min_n .metric .estimator   mean     n std_err .config
##   <int> <int> <int> <chr>   <chr>       <dbl> <int>   <dbl> <chr>
## 1     2   100     8 rmse    standard   0.0763    10 0.00363 Preprocessor1_Model…
```

Get the best model, finalize its workflow, and the fit the model like normal.

Code

Let's view the performance

Code

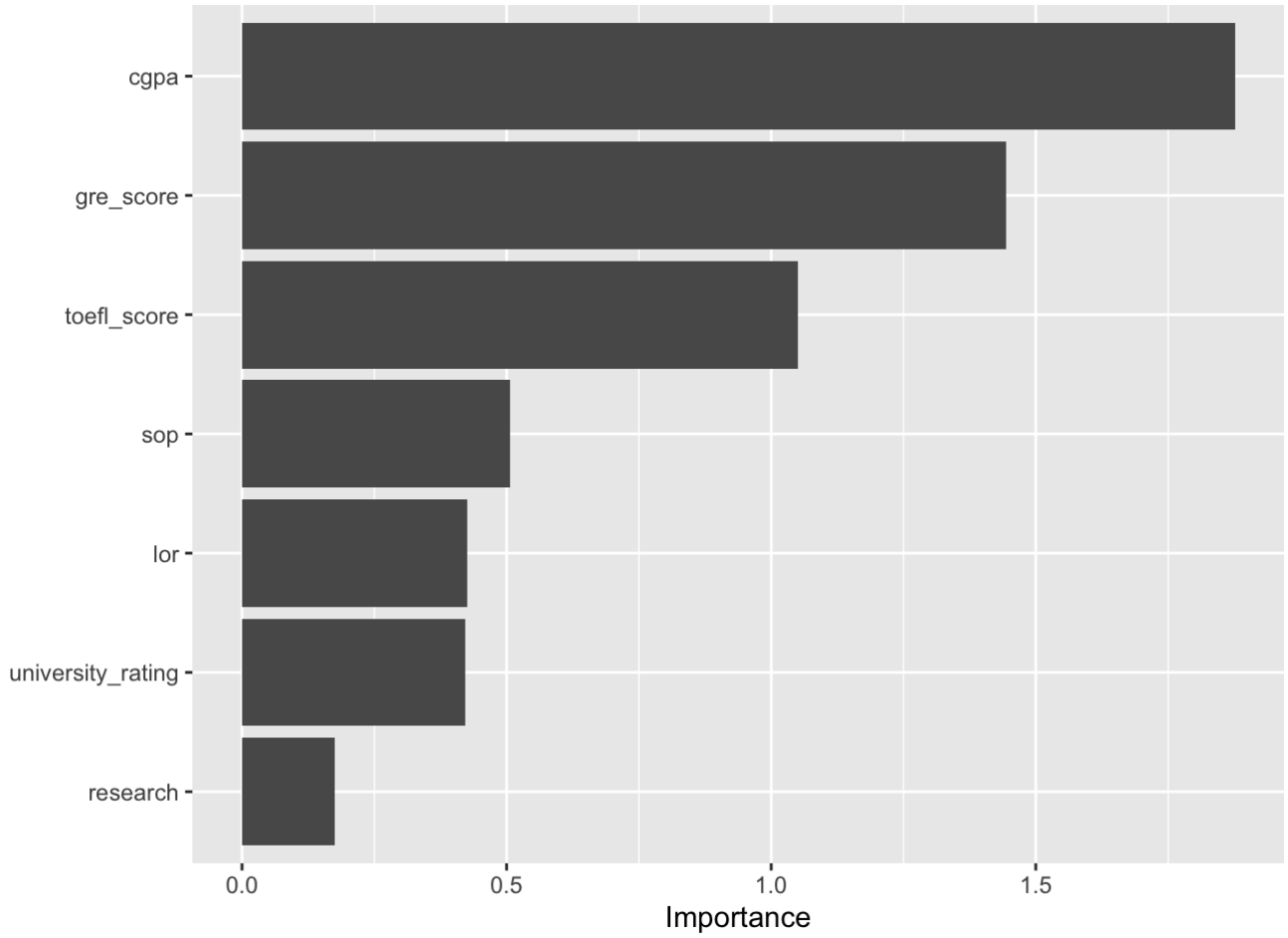
```
## # A tibble: 1 × 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard      0.0403
```

Using `vip()` to look at the variable importance plot:

From the plot below we can see that cgpa is the most important indicator in predicting chance of admissions, followed by GRE and TOEFL scores, and research is the least important. This tends to be consistent with the results of my previous EDA analysis.

To use a common explanation based on the actual phenomenon, universities are now offering more and more job-oriented taught master's degrees at the graduate level, which are not involved in research. Therefore, when it comes to admissions, admissions offices do not consider research experience as an important indicator, but rather they focus more on the academic performance of the applicant.

Code



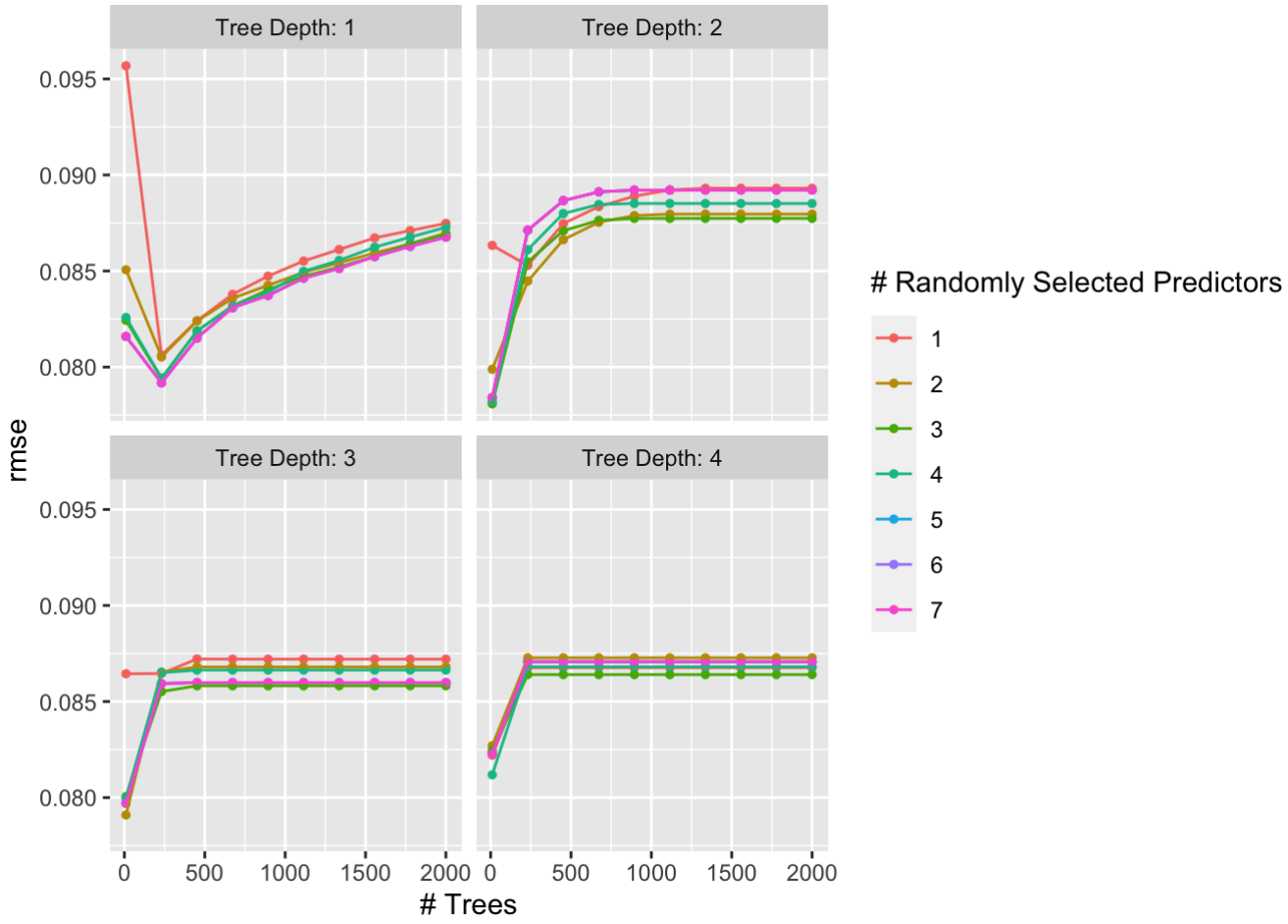

# Boosted Tree

Finally, set up a boosted tree model and workflow. Use the `xgboost` engine. Tune `trees`, `tree_depth`, and `mtry`. Create a regular grid with 10 levels

Code

Create a visualization. From the plot, we can detect that the best model should be at the very beginning with `mtry = 3` and `tree depth = 2`

Code




Code

```
## # A tibble: 1 × 9
##    mtry trees tree_depth .metric .estimator   mean     n std_err .config
##   <int> <int>      <int> <chr>   <chr>       <dbl> <int>   <dbl> <chr>
## 1     3    10          2 rmse    standard   0.0781    10 0.00285 Preprocessor1_…
```

Finalize the workflow, fit the model with training set and view the performance.

Code

```
## # A tibble: 1 × 4
##    mtry trees tree_depth .config
##   <int> <int>      <int> <chr>
## 1     3    10          2 Preprocessor1_Model091
```

Code

```
## # A tibble: 1 × 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard       0.0658
```

I combined the performance of five models in regard to `rmse` to determine which one performs best.

Code

```
## # A tibble: 5 × 2
##   .estimate Model_name
##       <dbl> <chr>
## 1    0.0752 Polynomial
## 2    0.0731 Ridge
## 3    0.0728 Lasso
## 4    0.0403 Random Forest
## 5    0.0658 Boosted tree
```

Seems like our winner with best-performance is Random Forest, so let's continue with this one to fit the testing data in.

# Analysis of The Testing Set

Let's fit the model to the testing dataset and create a few stored data sets for some analysis.

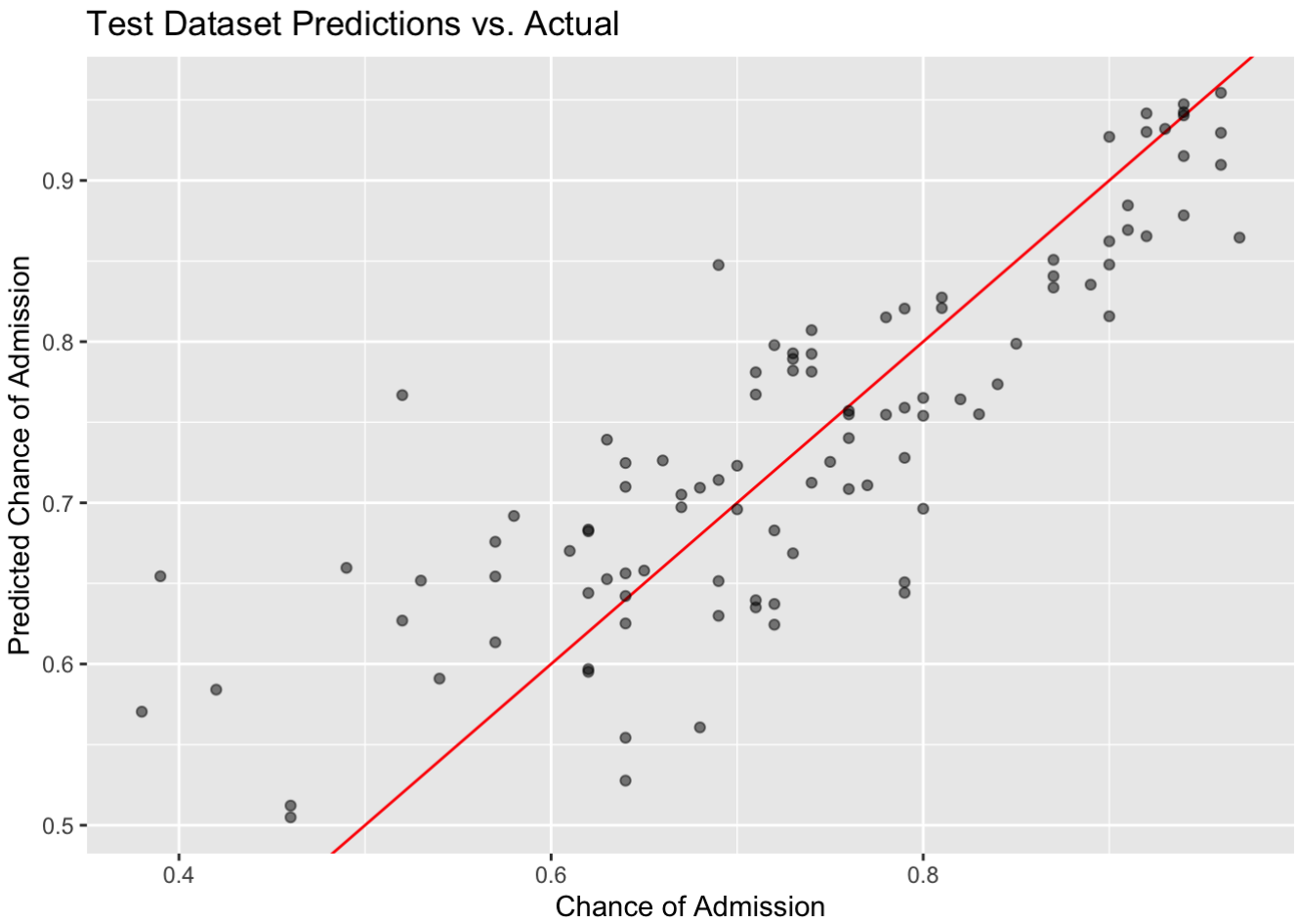
Code

```
## # A tibble: 1 × 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard      0.0763
```

Our model returned an `rmse` on our testing dataset is slightly and acceptably bigger than the rmse on our training dataset. This means my model has a some overfitting issue to the training data.

To check where some problem might lie, we can likewise plot the true values against the predicted values:

Code




Ohhh okay, seems like my machine had high variances and low bias.

# Conclusion

After testing various models, I ultimately decided to go with a random forest model when comparing the metrics.

Given the result of prediction on testing dataset, my model had a issue with overfitting, which is saying that model becomes too specialized on solving for the training data and starts to perform worse when validated on the test data. In other word, the model memorizes the answers in the training data set and does not generalize to the test data set. This model working well under the Train dataset, the seen data. We can see some predictions are on this red line but for the unseen data the predictions are scattered.

There are a few ways to overcome overfitting. The first one is to use K-fold cross validation, which I have already applied. I originally use 5 folds, but changed it to 10 after my first testing result. The second one is to remove some of the features and complexity of the model, like the research experience. Another one is to do the early stopping, which is to stop just when the gap between training error and validation error is as small as possible before it starts overfitting. And then, we can use the regularization, but it is a board concept of various techniques and algorithms so I am going to elaborate on this one. The last method I could think of is to use the assemble learning, which is to take advantages of power of multiple models and get some kind of average instead and hopefully that gives us a better prediction and hopefully it has less overfitting as we have randomized these models when we created them.

In brief, this final project provided me the chance to apply what I learned into actual case scenario and led to a decent outcome to predict a chance of getting admitted in these past competitive years. (ps: I got into Columbia)