Predicting Salary on Glassdoor

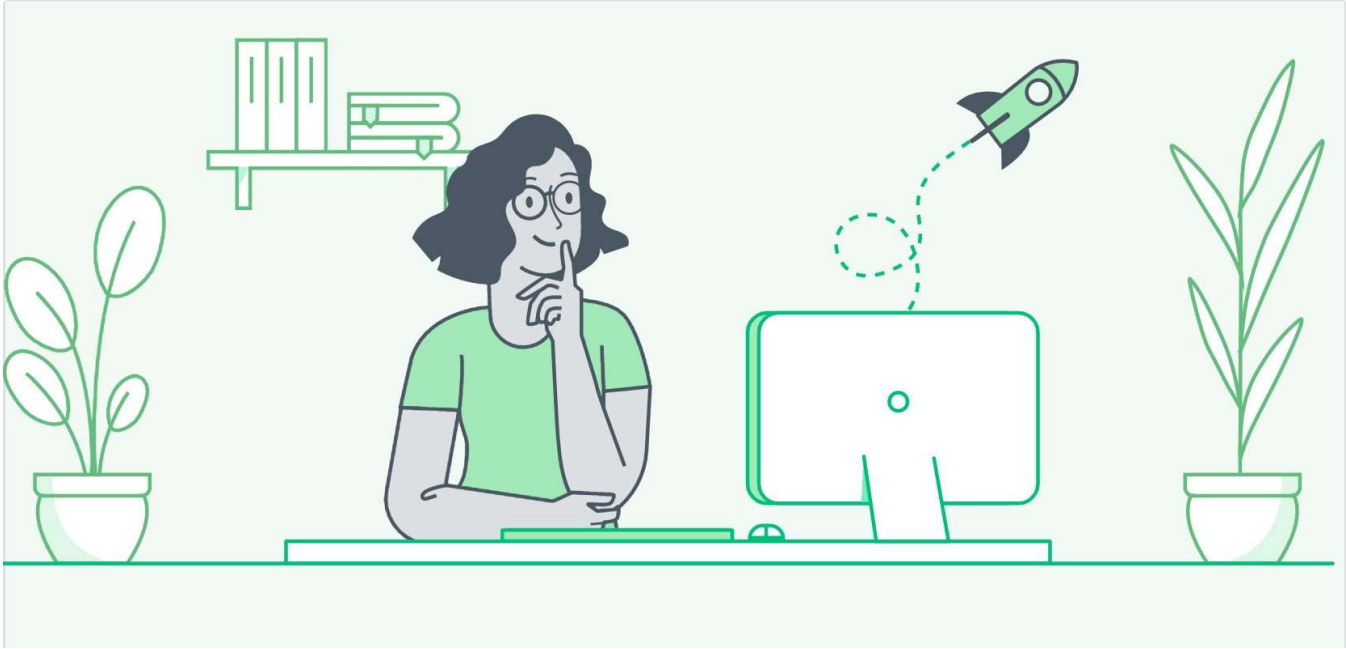
Code

Olivia Wu

March 27, 2023

Introduction

The purpose of this project is to develop a model that will predict salaries on Glassdoor.



What is Glassdoor?

Glassdoor is a digital platform that gathers company reviews and information from past and current employees anonymously. It acts as a job board where prospective employees can view company ratings, job postings, average salary range, benefits, and more. It also provides useful information like affiliated companies, HQ/company locations, interview ratings, and more. However recent

controversies have caused the company to reveal the identities of users who leave negative reviews of former employers. While the reliability of the site remains questionable, it is nevertheless a useful source for reference when job hunting.

Hide

```
knitr::include_graphics("C:/Users/olivi/Documents/ucsb winter 2023/pstat 131/final projec
t/assets/images/2560px-Glassdoor_logo.svg.png")
```

Hide

```
# ![Glassdoor logo](C:/Users/olivi/Documents/ucsb winter 2023/pstat 131/final project/asse
ts/images/2560px-Glassdoor_logo.svg.png)
```

Glassdoor logo

What are we trying to do?

A question we are more focused on asking is: what combination of factors results in the highest salary for separate positions on Glassdoor? The same job position could have varying salaries based on multiple factors like larger companies or different locations. I am interested in exploring what combination of these factors result in the highest salary. Our focus is on data science related positions as that is the most relevant to this class.

Why?

When we perform job searches the biggest aim is always about money. We aim for the highest salary possible within our realm of qualifications and capabilities. Instead of having to search, compile, and compare each result one by one on an excel sheet, we can take the aggregate data and get a overview of the bigger picture. While sites like LinkedIn may be more useful for job searches, often times pay is not included in the job posting. Glassdoor gives us a better sense of what the company is like in terms of work environment, pay, and benefits.

Data

Loading Packages and Data

Hide

```
library(tidymodels)
library(tidyverse)
library(tidyr)
library(ISLR)
library(ISLR2)
library(kknn)
library(glmnet)
library(modeldata)
library(ggthemes)
library(janitor)
library(naniar)
library(corrplot)
library(themis)
library(ggplot2)
tidymodels_prefer()

library(readr)
library(discrim)
library(forcats)
library(vip)
library(dplyr)
library(randomForest)
library(xgboost)
library(gsubfn) # for string extraction
library(stringr)
library(fastDummies)

# eda <- read.csv("C:/Users/olivi/Documents/ucsb winter 2023/pstat 131/final project/data/eda_data.csv")
# glassdoor_uncleaned <- read.csv("C:/Users/olivi/Documents/ucsb winter 2023/pstat 131/final project/data/glassdoor_jobs.csv")
```

Hide

```
glassdoor <- read_csv("C:/Users/olivi/Documents/ucsb winter 2023/pstat 131/final project/data/processed/salary_data_cleaned.csv") %>% clean_names()

head(glassdoor)
```

job_title <chr>	salary_estimate <chr>
Data Scientist	$53K-$91K (Glassdoor est.)
Healthcare Data Scientist	$63K-$112K (Glassdoor est.)

job_title <chr>	salary_estimate <chr>
Data Scientist	$80K-$90K (Glassdoor est.)
Data Scientist	$56K-$97K (Glassdoor est.)
Data Scientist	$86K-$143K (Glassdoor est.)
Data Scientist	$71K-$119K (Glassdoor est.)

6 rows | 1-2 of 28 columns

This dataset was taken from the Kaggle dataset "Salary Prediction (https://www.kaggle.com/datasets/thedevastator/jobs-dataset-from-glassdoor?resource=download&select=eda_data.csv)" and contains job postings from glassdoor.com (https://www.glassdoor.com/member/home/index.htm) in 2017. It was scraped by Ramiro Gomez and is licensed to CC0 1.0 - Public Domain Dedication.

Data Cleaning

i.e. variable selection

Hide

```
# check how much data we have; output is formatted row col
dim(glassdoor) # 742 28
```

```
## [1] 742  28
```

Hide

```
# checking number of unique classifications
glassdoor %>% distinct(job_title) %>% count() # 264
```

n <int>
264

1 row

Hide

```
# reorder avg_salary to 1st column to make it easier to access and clean the column names
glassdoor <- glassdoor %>% select(c("avg_salary", "min_salary", "max_salary", "job_title",
"rating", "size", "type_of_ownership", "industry", "revenue", "job_state", "age", "python_
yn", "r_yn", "spark", "aws", "excel")) %>% clean_names()
```

After further work it appears that one of the observation had "Los Angeles" in `job_state` so we will need to take care of that as well.

Hide

```
filterIndex = which(glassdoor$job_state == "Los Angeles")

glassdoor[filterIndex, ]$job_state = "CA"

# check that the observation is now correct
glassdoor[filterIndex, ]
```

avg_salary <dbl>	min_salary <dbl>	max_salary <dbl>	job_title <chr>	rating <dbl>	size <chr>
107.5	82	133	Data Scientist	3.6	1001 to 5000 employees

1 row | 1-6 of 16 columns

We can see that we now have 17 variables to work with. We removed `salary_estimate` as it gives a range in a string. To make analysis easier on ourselves we'll be using `avg_salary` to account for the range given.

`hourly` and `employer_provided` have values of 0 so they do not provide useful information, so obviously those are removed as well. Whether a job applier lives in the same state is irrelevant to estimating salary, so we also remove `same_state`. `location` is given in `job_state` so it is redundant information.

`job_description` is helpful for getting a sense of the job responsibilities, and would be interesting to see some repeating key words that appear throughout different job descriptions. From a glance we can see many descriptions include desired qualifications or skills like bachelor's degree in science/math, Tableau, Matlab, Python, SQL, C++, and so on. While it would be very informative to scrape the descriptions for words and separate them into another table, this could be another classification project in and of itself. We will simply use the data we currently have.

Making Revenue a numeric variable

Revenue is given a range in text. To account for this we will split revenue into 2 columns: `revenue_low` and `revenue_high`. Fortunately for us this will be a easy process to automate, as the data was all entered in the same format.

Hide

```
totalRow = 742

# replace " billion" with "000"
glassdoor$revenue <- sub(" billion", "000 million (USD)", glassdoor$revenue)

glassdoor$revenue <- sub("+ billion", "000 million (USD)", glassdoor$revenue)

# replace "Unknown / Non-Applicable" to "NA"
glassdoor$revenue <- sub("Unknown / Non-Applicable", "", glassdoor$revenue )

# split string to extract 2 numbers from the string
split2 = str_split(glassdoor$revenue, " to ")

# add columns with the extracted values
for (i in 1:totalRow)
{
  glassdoor$revenue_low[i] = extract_numeric(split2[[i]][1])
  glassdoor$revenue_high[i] = extract_numeric(split2[[i]][2])
}

# remove revenue from the dataset from list
glassdoor <- glassdoor %>% select(-revenue)
```

Tidying Observations

I think it would be useful to see what the most common job positions being posted are, so we do just that here.

Hide

```
agg_glassdoor <- glassdoor %>% group_by(job_title) %>% summarise(count = n(), avg_salary =
mean(avg_salary)) %>%  arrange(desc(count))

agg_glassdoor %>% arrange(count)
```

job_title <chr>
Ag Data Scientist
Analytics Consultant
Assistant Director/Director, Office of Data Science
Associate Data Scientist/Computer Scientist
Associate Research Scientist I (Protein Expression and Production)
Associate Scientist
Associate Scientist / Sr. Associate Scientist, Antibody Discovery

job_title <chr>
Associate Scientist/Scientist, Process Analytical Technology - Small Molecule Analytical Chemistry
Big Data Engineer - Chicago - Future Opportunity
Business Data Analyst, SQL

1-10 of 264 rows | 1-1 of 3 columns Previous **1** 2 3 4 5 6 ... 27 Next

Hide

```
head(agg_glassdoor, 10)
```

job_title <chr>	count <int>	avg_salary <dbl>
Data Scientist	131	106.18
Data Engineer	53	91.66
Senior Data Scientist	34	134.87
Data Analyst	15	66.30
Senior Data Engineer	14	121.93
Senior Data Analyst	12	83.42
Lead Data Scientist	8	161.25
Marketing Data Analyst	6	48.67
Sr. Data Engineer	6	115.00
Machine Learning Engineer	5	104.90

1-10 of 10 rows

We see that `job_title` has many unique names. For example, “Data Scientist” is being called “Data Scientist, Rice University”. “Rice University” only specifies the location, which is already taken care of. We need to modify the observations a little to account for cases like this. However, a good number of these positions also specify what department the applier will be working in, such as “Senior Scientist (Neuroscience)”. I chose to combine these with their respective generic title. For this case, it would become “Senior Scientist.” While department could affect pay, because of how many positions do not have their department specified, it would be difficult to analyze.

While it is extremely tedious to make a separate if statement for each case, I’m not sure of a faster method to achieve this so it’s exactly what I did.

Hide

```
# get indexes that have counts fewer than 3
# since the list is already sorted in ascending order by count, we only need to get the la
st index number
# filterIndex <- max(which(agg_glassdoor$count < 3)) # 209
#
# agg_glassdoor[1:filterIndex,] # view observations that fall into counts <= 2

# rename job_title to generic titles
# note: we can't use sub as it doesn't replace the entire string, which means painstakingl
y making separate
# if statements per case, although I am well aware this is poor practice
for (i in 1:totalRow)
{
  if (grepl("Assistant Director", glassdoor$job_title[i])) { glassdoor$job_title[i] = "Ass
istant Director" }
  else  if (grepl("Director", glassdoor$job_title[i])) { glassdoor$job_title[i] = "Directo
r" }

  # regex is for string search in same syntax as assembly
  if (grepl("Senior Associate Scientist", glassdoor$job_title[i])) { glassdoor$job_title
[i] = "Senior Associate Scientist" }
  else if (grepl("Data Science Analyst", glassdoor$job_title[i])) { glassdoor$job_title[i]
= "Data Analyst" }
  else if (grepl("Software Engineer", glassdoor$job_title[i])) { glassdoor$job_title[i] =
"Software Engineer" }
  else if (grepl("Senior Data Scientist", glassdoor$job_title[i])) { glassdoor$job_title
[i] = "Senior Data Scientist" }
  else if (grepl(regex("Research Scientist", ignore_case = TRUE), glassdoor$job_title[i]))
{ glassdoor$job_title[i] = "Research Scientist" }
  else if (grepl("Senior Data Engineer", glassdoor$job_title[i])) { glassdoor$job_title[i]
= "Senior Data Engineer" }
  else if (grepl("Data Engineer", glassdoor$job_title[i])) { glassdoor$job_title[i] = "Dat
a Engineer" }
  else if (grepl("Machine Learning Engineer", glassdoor$job_title[i])) { glassdoor$job_tit
le[i] = "Machine Learning Engineer" }
  else if (grepl("Manager", glassdoor$job_title[i])) { glassdoor$job_title[i] = "Manager"
}
  else if (grepl("Senior Data Analyst", glassdoor$job_title[i])) { glassdoor$job_title[i]
= "Senior Data Analyst" }
  else if (grepl("Data Analyst", glassdoor$job_title[i])) { glassdoor$job_title[i] = "Data
Analyst" }
  else if (grepl("Data Scientist", glassdoor$job_title[i])) { glassdoor$job_title[i] = "Da
ta Scientist" }
  else if (grepl("Scientist", glassdoor$job_title[i])) { glassdoor$job_title[i] = "Scienti
st" }
  else if (grepl("MED TECH/LAB SCIENTIST- SOUTH COASTAL LAB", glassdoor$job_title[i])) { g
lassdoor$job_title[i] = "Research Scientist" }
  else if (grepl("Senior Data Science Systems Engineer", glassdoor$job_title[i])) { glassd
oor$job_title[i] = "Senior Data Engineer" }
  else if (grepl("Data Modeler - Data Solutions Engineer", glassdoor$job_title[i])) { glas
sdoor$job_title[i] = "Data Engineer" }
  else if (grepl("RESEARCH COMPUTER SCIENTIST - RESEARCH ENGINEER - SR. COMPUTER SCIENTIST
```

```
- SOFTWARE DEVELOPMENT", glassdoor$job_title[i])) { glassdoor$job_title[i] = "Software Eng
ineer" }
  else if (grepl("Data Science Engineer", glassdoor$job_title[i])) { glassdoor$job_title
[i] = "Data Engineer" }
  else {  glassdoor$job_title[i] = "Other" }
}

agg_glassdoor <- glassdoor %>% group_by(job_title) %>% summarise(count = n(), avg_salary =
mean(avg_salary)) %>%  arrange(desc(count))

agg_glassdoor
```

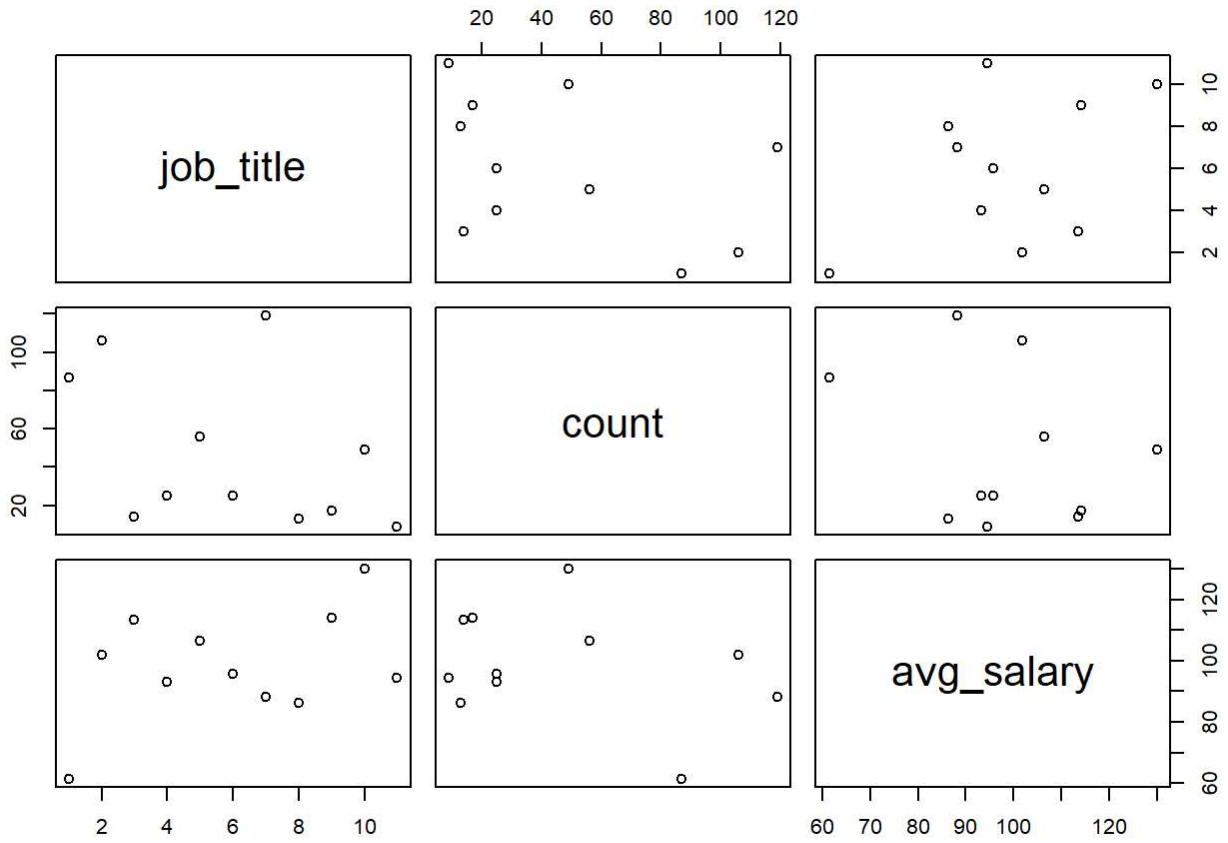
job_title <chr>	count <int>	avg_salary <dbl>
Data Scientist	222	114.72
Scientist	119	88.18
Data Engineer	106	101.88
Data Analyst	87	61.41
Other	56	106.44
Senior Data Scientist	49	130.06
Manager	25	93.22
Research Scientist	25	95.82
Senior Data Engineer	17	114.09
Machine Learning Engineer	14	113.46

1-10 of 12 rows Previous **1** 2 Next

We can see that the frequencies for each position has increased from last we checked. We've now reduced the job_titles so they can be classified together much more easily. Lastly we convert `job_title` to a factor as it is a categorical variable. ### Number of Job Titles We filter out the observations where the job title appears less than 7 times and plot the rest.

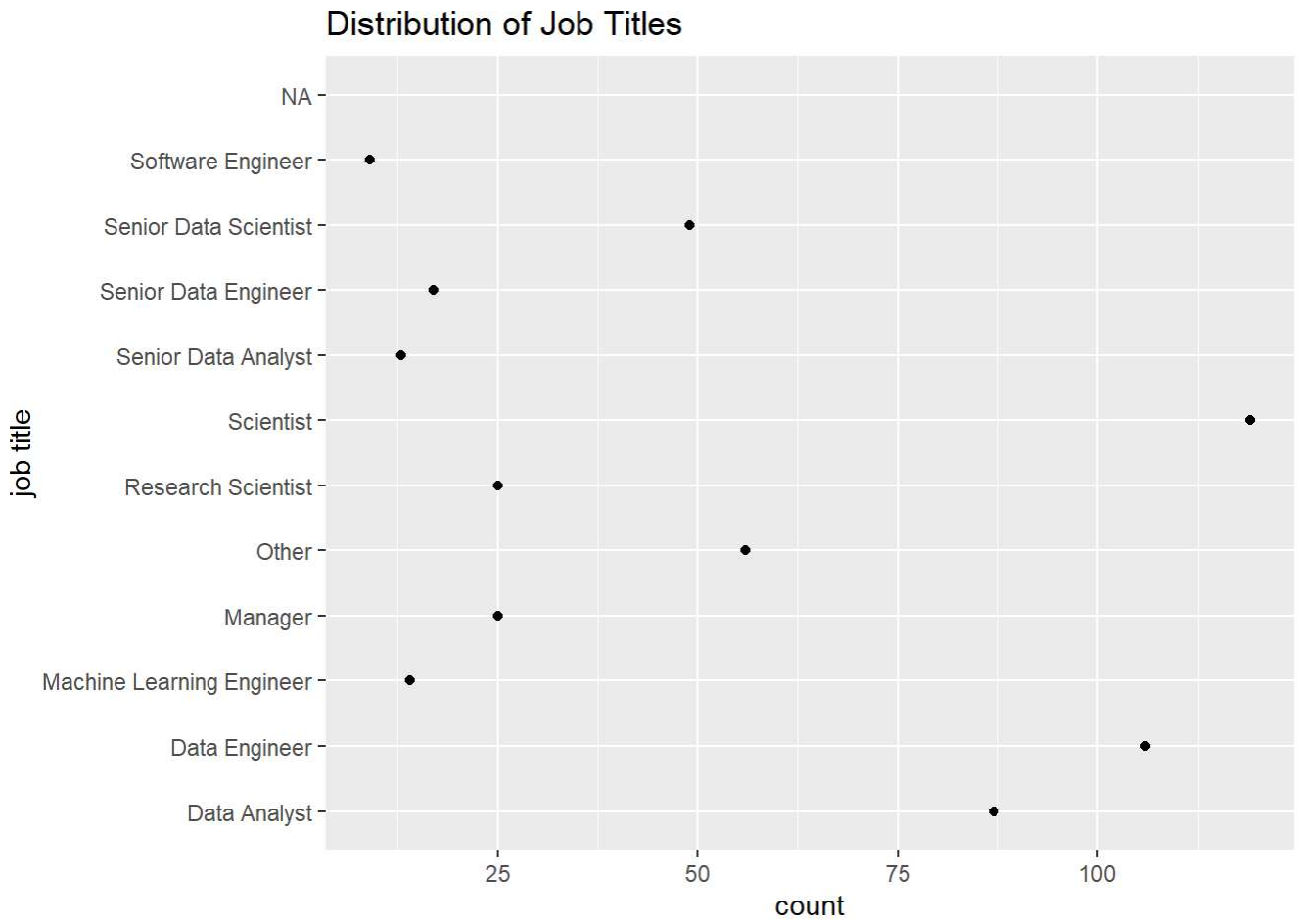
Hide

```
# separate uncommon job titles into a "Other" column
desc_agg_glassdoor <- agg_glassdoor %>% arrange(count)
filterIndex <- max(which(desc_agg_glassdoor$count < 7))
# agg_glassdoor_filtered <- agg_glassdoor %>% mutate(job_title = fct_lump_n(job_title, 7))

# dim(desc_agg_glassdoor) # 37 3
desc_agg_glassdoor_filtered = agg_glassdoor[2:37,]
desc_agg_glassdoor_filtered <- desc_agg_glassdoor_filtered %>% mutate(job_title = as.facto
r(job_title))

plot(desc_agg_glassdoor_filtered)
```



Hide

```
# barplot(desc_agg_glassdoor_filtered$count, main = "Distribution of Job Titles")

desc_agg_glassdoor_filtered %>%
  ggplot(aes(x = count, y = job_title)) +
  geom_point() +
  labs(title = "Distribution of Job Titles", y = "job title" )
```



As we can see from the plots, “Data Scientist” appears a lot more which is not a surprise, as we are focusing on data science related positions. “Data Engineer” and “Scientist” interestingly appear about the same amount of times. Data Analyst comes after, which is also not surprise. As expected, there are also not many senior level positions being posted as it is a job requiring more industry experience.

Checking for Missing Values

After filtering the dataset we need to check for missing data as they could influence the analysis significantly. We see that revenue_low and revenue_high both have missing variables. As these are non negligible amounts of missing data we will omit these observations from our analysis.

Hide

```
# check missing data
cbind(lapply(lapply(glassdoor, is.na), sum))
```

```
##                   [,1]
## avg_salary        0
## min_salary        0
## max_salary        0
## job_title         0
## rating            0
## size              0
## type_of_ownership 0
## industry          0
## job_state         0
## age               0
## python_yn         0
## r_yn              0
## spark             0
## aws               0
## excel             0
## revenue_low       203
## revenue_high      332
```

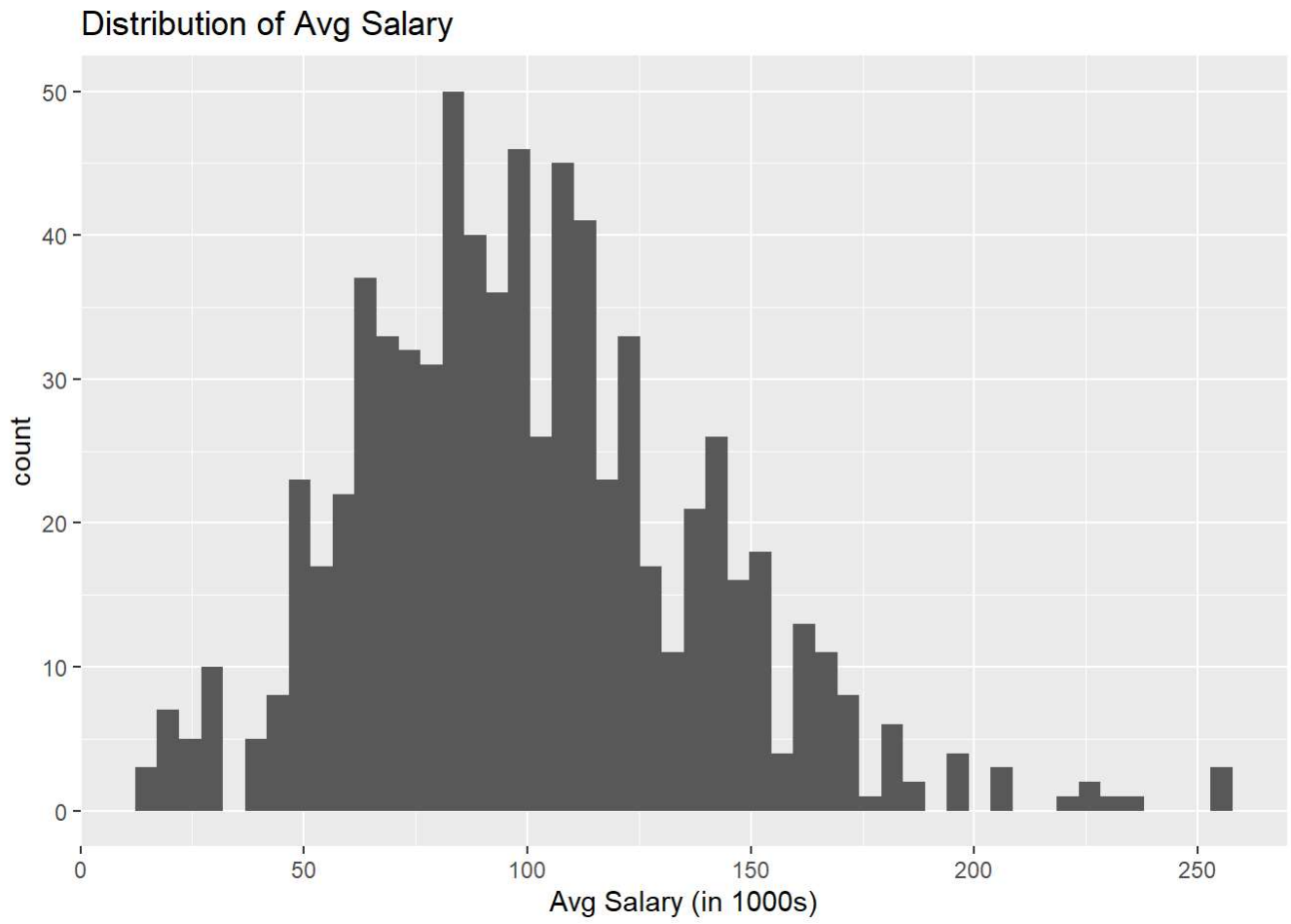
Explanatory Data Analysis

Average Salary

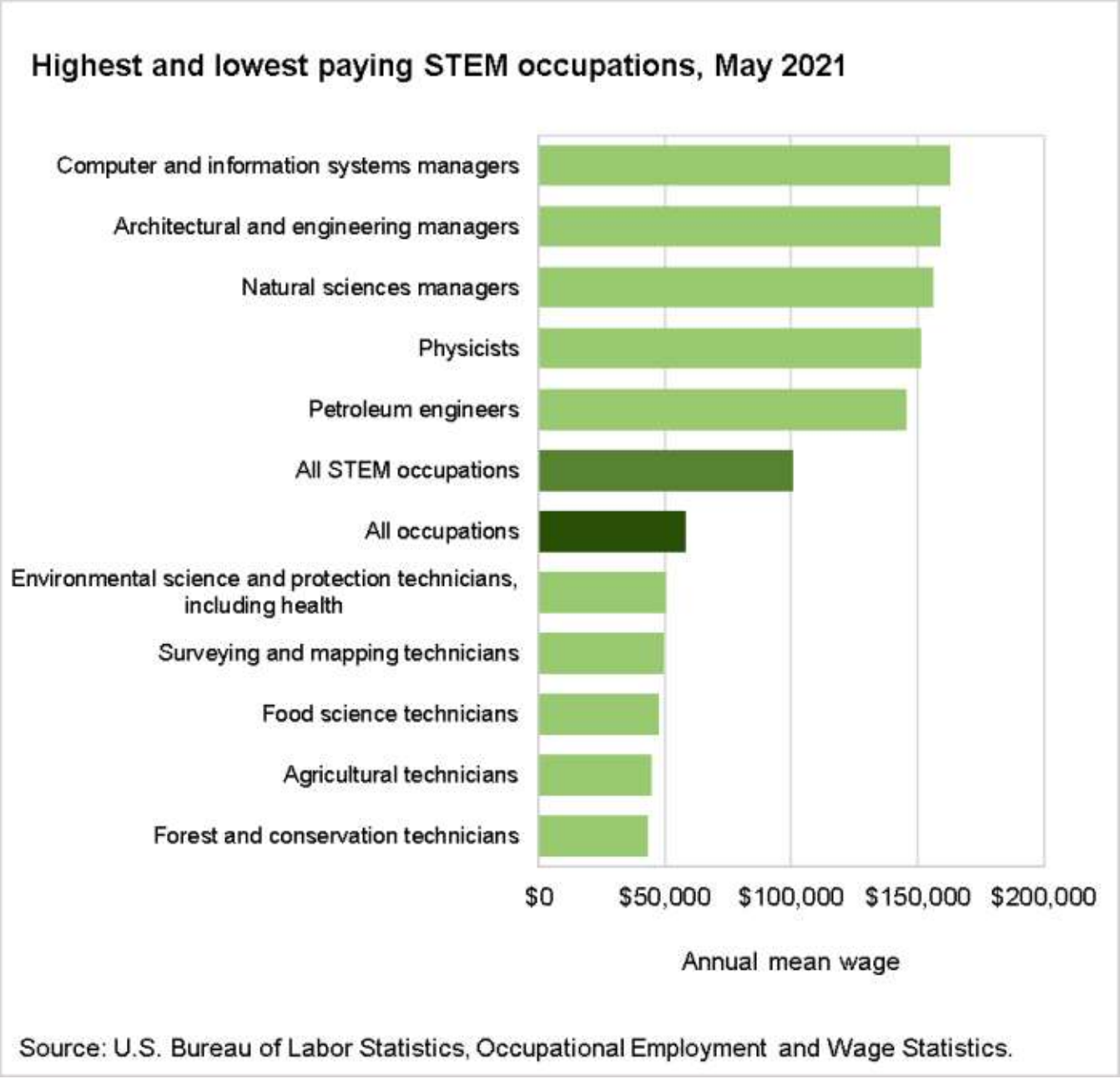
We'll first plot average salary to see what kind of trends it has with a histogram.

Hide

```
ggplot(glassdoor, aes(x = avg_salary)) +
  geom_histogram(bins = 50) +
  labs( title = "Distribution of Avg Salary", x = "Avg Salary (in 1000s)")
```



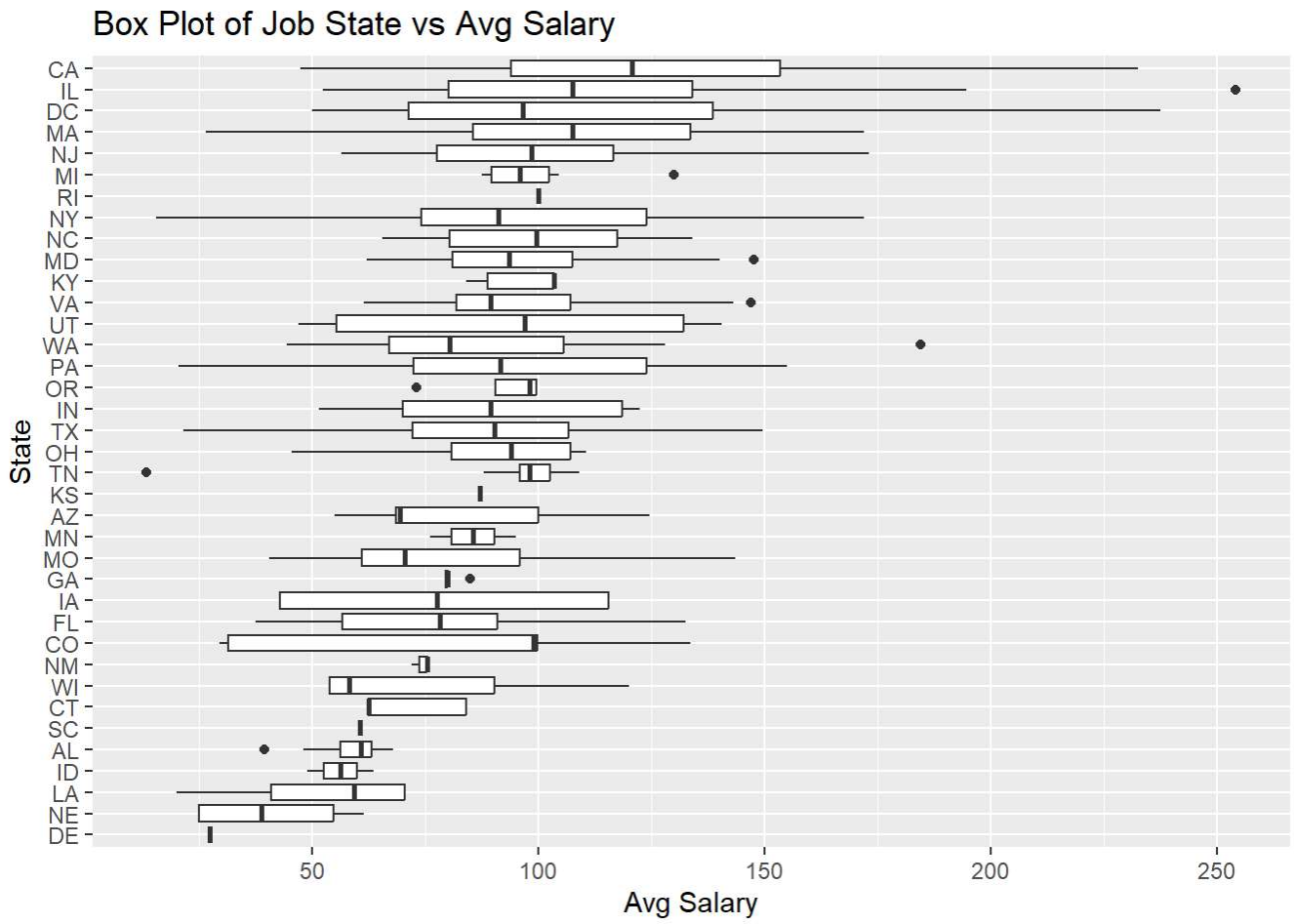
We can see average salary has a left skew of what visually seems to be a mostly normal trend. The highest concentration seems to be around a salary of $80k. It's interesting to note that the data seems to be concentrated at every other bin. After $80k, the next biggest concentration is around $100k, then $120k. We can see that data scientists earn what most would consider a respectable amount of income, given that the national average income is around $31k. However, according to the Bureau of Labor Statistics (https://www.bls.gov/oes/2021/may/stem4.htm), average salary for STEM occupations is around $100k, which means data scientists on average earn lower than their industry peers.



Highest and lowest paying STEM occupations in May 2021, according to BLS

Hide

```
glassdoor %>% ggplot(aes(x = avg_salary, y = reorder(job_state, avg_salary))) +
  geom_boxplot() +
  labs( title = "Box Plot of Job State vs Avg Salary", y = "State", x = "Avg Salary")
```

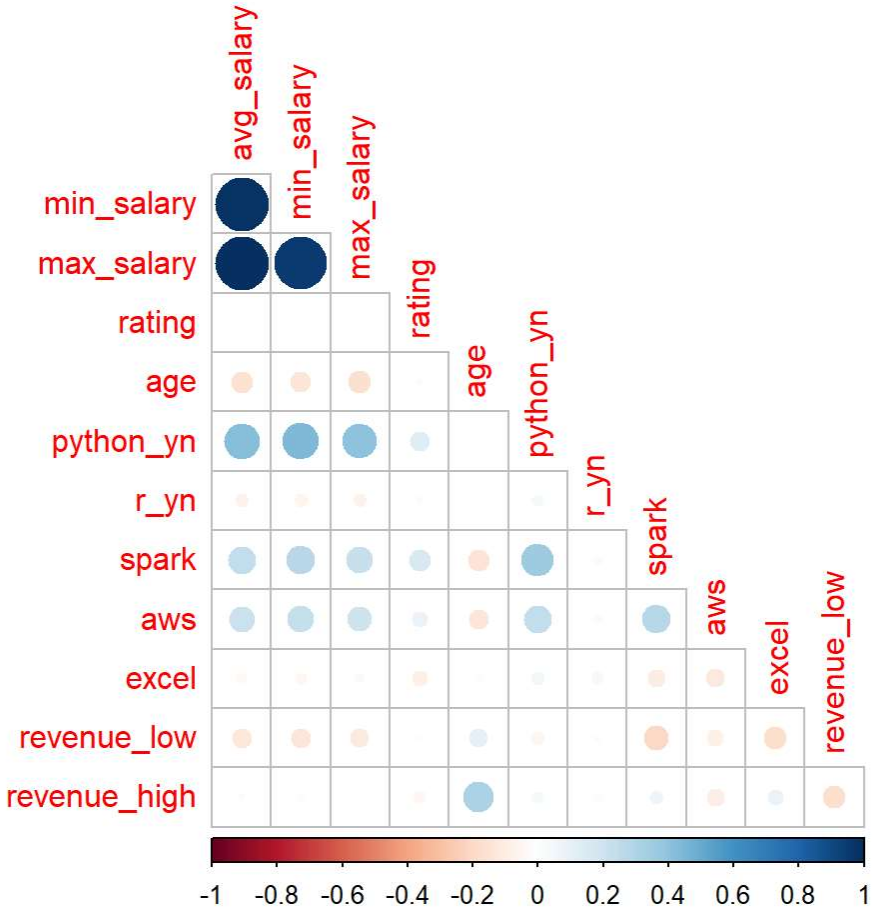


It is no surprise that California has the highest IQR, as it is home to the Silicon Valley, where most tech company headquarters are located at. It is unexpected that DC has the highest maximum however, and Illinois has the highest outlier. Since IQR seems to differ for each state, while we can't say for sure that location affects salary, from the box plots there appears to be a decreasing trend as we go down the list of states.

Correlation Plot

Hide

```
# filter out only numeric values
glassdoor_numeric <- glassdoor %>%
  select(is.numeric)

# correlation matrix
glassdoor_corr <- cor(glassdoor_numeric)

# glassdoor_corrplot <- corrplot(glassdoor_corr, method = "circle", addCoef.col = 1, numbe
r.cex = 0.7, diag=FALSE, type = "lower")

glassdoor %>%
  select(where(is.numeric)) %>%
  cor(use = "complete.obs") %>%
  corrplot(type = "lower", diag = FALSE)
```

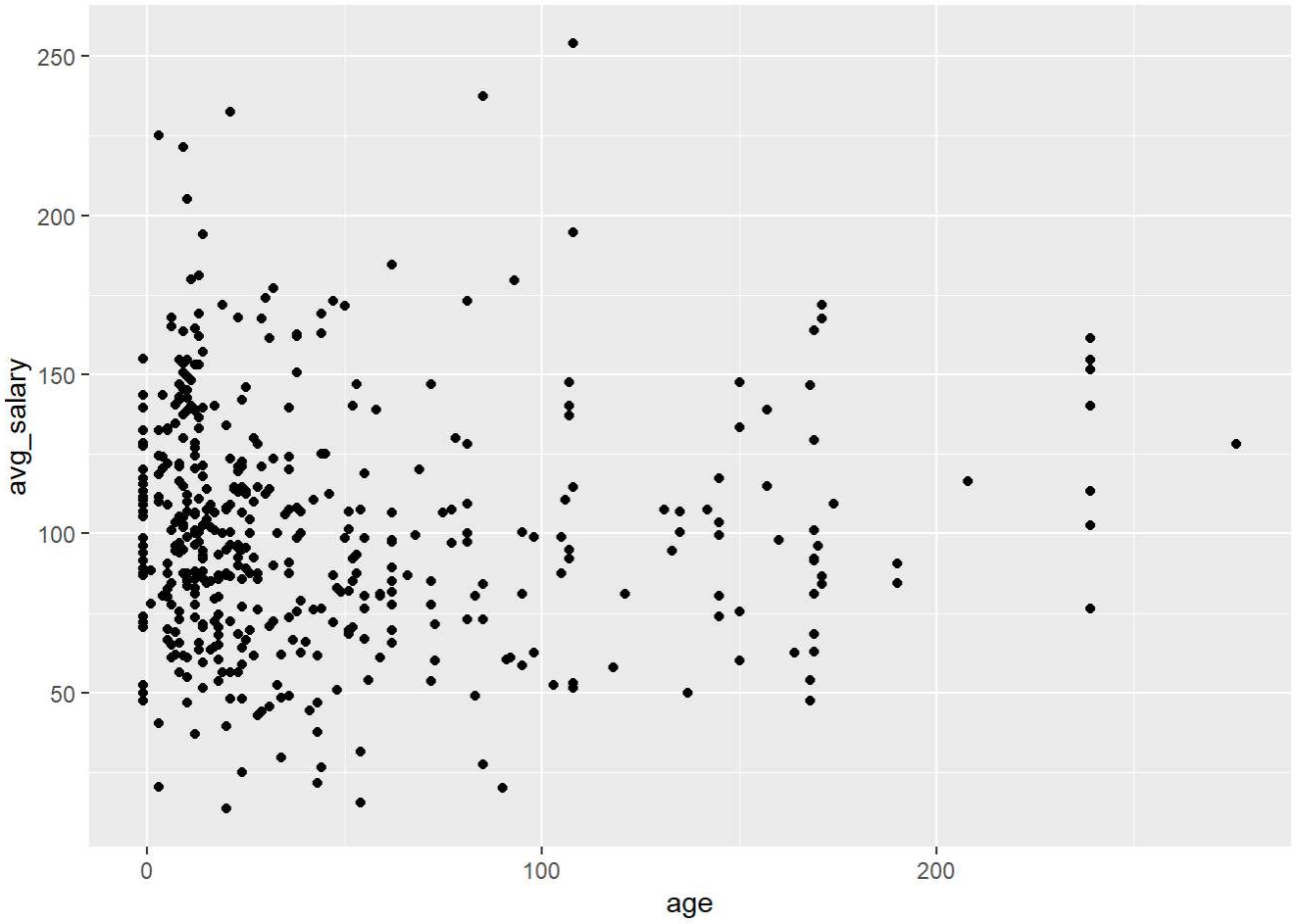


We have almost no numeric data that's correlated with `avg_salary`. Since we don't have many numeric variables a correlation plot does not actually provide much useful information. `min_salary` and `max_salary` is what `avg_salary` is generated from so we ignore that.

We can see that the average salary has almost no correlation with age and Glassdoor rating. It is interesting however to note that `revenue_high` has a mild positive correlation with company `age`. This is most likely due to companies that have been around longer tends to earn higher revenue given that they've stayed afloat for longer. `python_yn` is positively correlated with average salary, since Python is a common skill listed as a qualification for data scientists.

Hide

```
# scatterplot in relation to each predictor
glassdoor %>%
  ggplot(aes(x = age, y = avg_salary)) +
  geom_point()
```



Most of the data is clumped under 50 for company age. Silicon Valley tech companies didn't really start appearing in what we can consider recent timeline, so this also makes sense. Famous companies like Amazon, Facebook (Meta), Apple, Nvidia, Microsoft, AMD, and so on were not founded until the 1970 - 2000s.

Model Building

Converting to Factors

We see that the data has 742 rows and 28 columns, which means we have 28 different variables, with our response being `avg_salary` , so 27 different predictors. We also see that we have 264 unique job titles in the data set. Some of these columns are empty or not useful for us, so we'll remove them.

Hide

```
# convert categorical data to factors
glassdoor <- glassdoor %>%
  mutate(job_title = factor(job_title), type_of_ownership = factor(type_of_ownership), ind
ustry = factor(industry), job_state = factor(job_state), python_yn = factor(python_yn), r_
yn = factor(r_yn), spark = factor(spark), aws = factor(aws), excel = factor(excel))

dim(glassdoor) # 742  17
```

```
## [1] 742  17
```

Splitting the Data

Hide

```
set.seed(3435)
glassdoor <- glassdoor %>% select(-revenue_low, -revenue_high, -min_salary, -max_salary)

glassdoor_split <- glassdoor %>%
  initial_split(prop = 0.7, strata = "avg_salary")

glassdoor_train <- training(glassdoor_split)
glassdoor_test <- testing(glassdoor_split)
```

Setting up Recipe

Hide

```
glassdoor_recipe <- recipe(avg_salary ~ rating + size + python_yn + r_yn + spark + aws + e
xcel + job_title + job_state, data=glassdoor_train) %>%
  step_dummy(python_yn, r_yn, spark, aws, excel, size, job_title, job_state) %>%
  step_center((all_predictors())) %>%
  step_scale(all_predictors())
```

K Fold Cross Validation

We split the model into 5 folds and use cross validation to tune the model and select the best one using `rsme` as the criteria. `rsme` is the square root of MSE, often used to measure differences between observed and predicted values.

Hide

```
# set.seed(11)
glassdoor_folds <- vfold_cv(glassdoor_train, v = 5, strata = "avg_salary")
```

Fitting the Models

steps: setup model by specifying model you want to fit, tuning parameters, engine, and mode (regression) set up workflow for the model, add model to recipe create tuning grid to specify range of params you want to tune and # levels save tuned models to RDS file so we don't have to rerun it load back saved fiels collect metrics of tuned models, arrange in ascending order of mean to see lowest RMSE for the tuned model, and slice to choose loweset RMSE. save the RMSE value for further comparison

Hide

```
# linear regression
lm_model <- linear_reg() %>%
  set_engine("lm")

# workflow
lm_workflow <- workflow() %>%
  add_model(lm_model) %>%
  add_recipe(glassdoor_recipe)

# linear regression has no tuning parameters

# ridge regression: penalty, mixture = 0
ridge_spec <- linear_reg(mixture = 0, penalty = tune()) %>%
  set_mode("regression") %>%
  set_engine("glmnet")

ridge_workflow <- workflow() %>%
  add_recipe(glassdoor_recipe) %>%
  add_model(ridge_spec)

# tuning
penalty_grid <- grid_regular(penalty(range = c(-5,5)), levels = 50)

# lasso
lasso_spec <- linear_reg(penalty = tune(), mixture = 1) %>%
  set_mode("regression") %>%
  set_engine("glmnet")

lasso_workflow <- workflow() %>%
  add_recipe(glassdoor_recipe) %>%
  add_model(lasso_spec)

# lasso uses same grid as ridge

# k nearest neighbors
knn_model <- nearest_neighbor(neighbors = tune()) %>%
  set_mode("regression") %>%
  set_engine("kknn")

knn_workflow <- workflow() %>%
  add_model(knn_model) %>%
  add_recipe(glassdoor_recipe)

knn_grid <- grid_regular(neighbors(range = c(1,15)), levels = 5)

# random forest
rf_model <- rand_forest(mtry = tune(), trees = tune(), min_n = tune()) %>%
  set_engine("ranger", importance = "impurity") %>%
  set_mode("regression")

rf_workflow <- workflow() %>%
  add_recipe(glassdoor_recipe) %>%
```

```
  add_model(rf_model)

rf_parameter_grid <- grid_regular(mtry(range = c(1, 12)), trees(range = c(200,1000)), min_
n(range = c(5,20)), levels = 8)

# boosted trees
boosted_model <- boost_tree(trees = tune(), learn_rate = tune(), min_n = tune()) %>%
  set_engine("xgboost") %>%
  set_mode("regression")

boosted_workflow <- workflow() %>%
  add_recipe(glassdoor_recipe) %>%
  add_model(boosted_model)

boosted_grid <- grid_regular(trees(range = c(5, 200)), learn_rate(range = c(0.01,0.1), tra
ns = identity_trans()), min_n(range = c(40, 60)), levels = 5)
```

Model Tuning

Hide

```
# linear regression does not need tuning

# ridge
# ridge_tune <- tune_grid(
#   ridge_workflow,
#   resamples = glassdoor_folds,
#   grid = penalty_grid)
#
# #lasso
# lasso_tune <- tune_grid(
#   lasso_workflow,
#   resamples = glassdoor_folds,
#   grid = penalty_grid)
#
# #k nearest neighbors
# knn_tune <- tune_grid(
#     knn_workflow,
#     resamples = glassdoor_folds,
#     grid = knn_grid)
#
# # random forest
# rf_tune_res <- tune_grid(
#   rf_workflow,
#   resamples = glassdoor_folds,
#   grid = rf_parameter_grid)
#
# # boosted trees
# boosted_tune_res <- tune_grid(
#   boosted_workflow,
#   resamples = glassdoor_folds,
#   grid = boosted_grid)
```

Hide

```
# save tuned models so we don't need to rerun them
# write_rds(ridge_tune, file = "C:/Users/olivi/Documents/ucsb winter 2023/pstat 131/final project/data/processed/ridge2.rds")
# write_rds(lasso_tune, file = "C:/Users/olivi/Documents/ucsb winter 2023/pstat 131/final project/data/processed/lasso2.rds")
# write_rds(knn_tune, file = "C:/Users/olivi/Documents/ucsb winter 2023/pstat 131/final project/data/processed/knn2.rds")
# write_rds(rf_tune_res, file = "C:/Users/olivi/Documents/ucsb winter 2023/pstat 131/final project/data/processed/rf2.rds")
# write_rds(boosted_tune_res, file = "C:/Users/olivi/Documents/ucsb winter 2023/pstat 131/final project/data/processed/boosted2.rds")
```

Hide

```
# load back saved files
ridge_tuned <- read_rds(file = "C:/Users/olivi/Documents/ucsb winter 2023/pstat 131/final project/data/processed/ridge2.rds")
lasso_tuned <- read_rds(file = "C:/Users/olivi/Documents/ucsb winter 2023/pstat 131/final project/data/processed/lasso2.rds")
knn_tuned <- read_rds(file = "C:/Users/olivi/Documents/ucsb winter 2023/pstat 131/final project/data/processed/knn2.rds")
rf_tuned <- read_rds(file = "C:/Users/olivi/Documents/ucsb winter 2023/pstat 131/final project/data/processed/rf2.rds")
boosted_tuned <- read_rds(file = "C:/Users/olivi/Documents/ucsb winter 2023/pstat 131/final project/data/processed/boosted2.rds")
```

Hide

```
# collect metrics and slice to save rmse
# linear regression
lm_fit <- fit_resamples(lm_workflow, resamples = glassdoor_folds)
lm_rmse <- collect_metrics(lm_fit) %>% slice(1)
lm_rmse
```

.metric	.estimator	mean	n	std_err	.config
<chr>	<chr>	<dbl>	<int>	<dbl>	<chr>
rmse	standard	32.15	5	1.335	Preprocessor1_Model1

1 row

Hide

```
# ridge
ridge_rmse <- collect_metrics(ridge_tuned) %>%
  arrange(mean) %>% slice(49)
ridge_rmse
```

penalty	.metric	.estimator	mean	n	std_err	.config
<dbl>	<chr>	<chr>	<dbl>	<int>	<dbl>	<chr>
4.095e-05	rmse	standard	32.15	5	1.472	Preprocessor1_Model04

1 row

Hide

```
# lasso
lasso_rmse <- collect_metrics(lasso_tuned) %>%
  arrange(mean) %>% slice(35)
lasso_rmse
```

penalty	.metric	.estimator	mean	n	std_err	.config
<dbl>	<chr>	<chr>	<dbl>	<int>	<dbl>	<chr>
0.1207	rmse	standard	31.96	5	1.413	Preprocessor1_Model21

1 row

Hide

```
#
# knn
knn_rmse <- collect_metrics(knn_tuned) %>%
  arrange(mean) %>% slice(6)
knn_rmse
```

neighbors	.metric	.estimator	mean	n	std_err	.config
<int>	<chr>	<chr>	<dbl>	<int>	<dbl>	<chr>
4	rmse	standard	31.61	4	1.372	Preprocessor1_Model2

1 row

Hide

```
# random forest
rf_rmse <- collect_metrics(rf_tuned) %>%
  arrange(mean) %>% slice(513)
rf_rmse
```

m...	trees	mi...	.metric	.estimator	me...	n	std_err	.config
<int>	<int>	<int>	<chr>	<chr>	<dbl>	<int>	<dbl>	<chr>
12	314	5	rmse	standard	28.24	4	1.416	Preprocessor1_Model016

1 row

Hide

```
# boosted trees
boosted_rmse <- collect_metrics(boosted_tuned) %>%
  arrange(mean) %>% slice(126)
boosted_rmse
```

tre...	mi...	learn_rate	.metric	.estimator	me...	n	std_err	.config
<int>	<int>	<dbl>	<chr>	<chr>	<dbl>	<int>	<dbl>	<chr>
200	40	0.1	rmse	standard	32.92	5	1.788	Preprocessor1_Model105

1 row

Model Results

We pick the model based off the lowest mean RMSE. From the performance of the models based off the cross-validation data, random forest performed the best.

Hide

```
model <- c('Linear Regression', 'Ridge Regression', 'Lasso Regression', 'K nearest Neighbo
rs', 'Random Forest', "Boosted Trees")
RMSE <- c(32.15, 32.15, 31.96, 31.61, 28.24, 32.92)
data.frame(model, RMSE) %>% arrange(RMSE)
```

model <chr>	**RMSE** <dbl>
Random Forest	28.24
K nearest Neighbors	31.61
Lasso Regression	31.96
Linear Regression	32.15
Ridge Regression	32.15
Boosted Trees	32.92

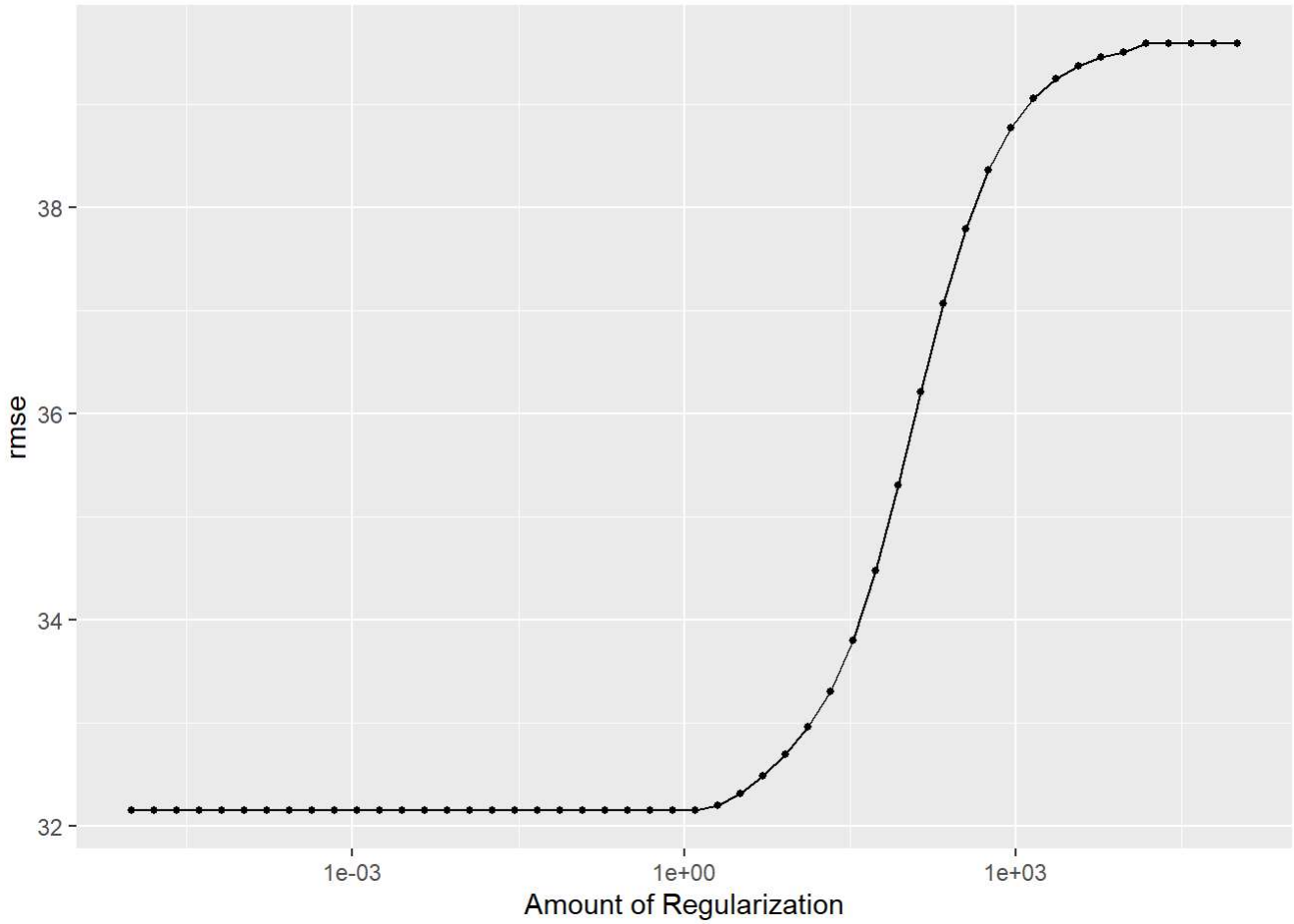
6 rows

Model Autoplots

Ridge Regression The graph shows that as rmse increases penalty also increases. Lowest rmse occurs when penalty is a little more than 1. It plateaus at the top, appearing to increase at a very small constant rate.

Hide

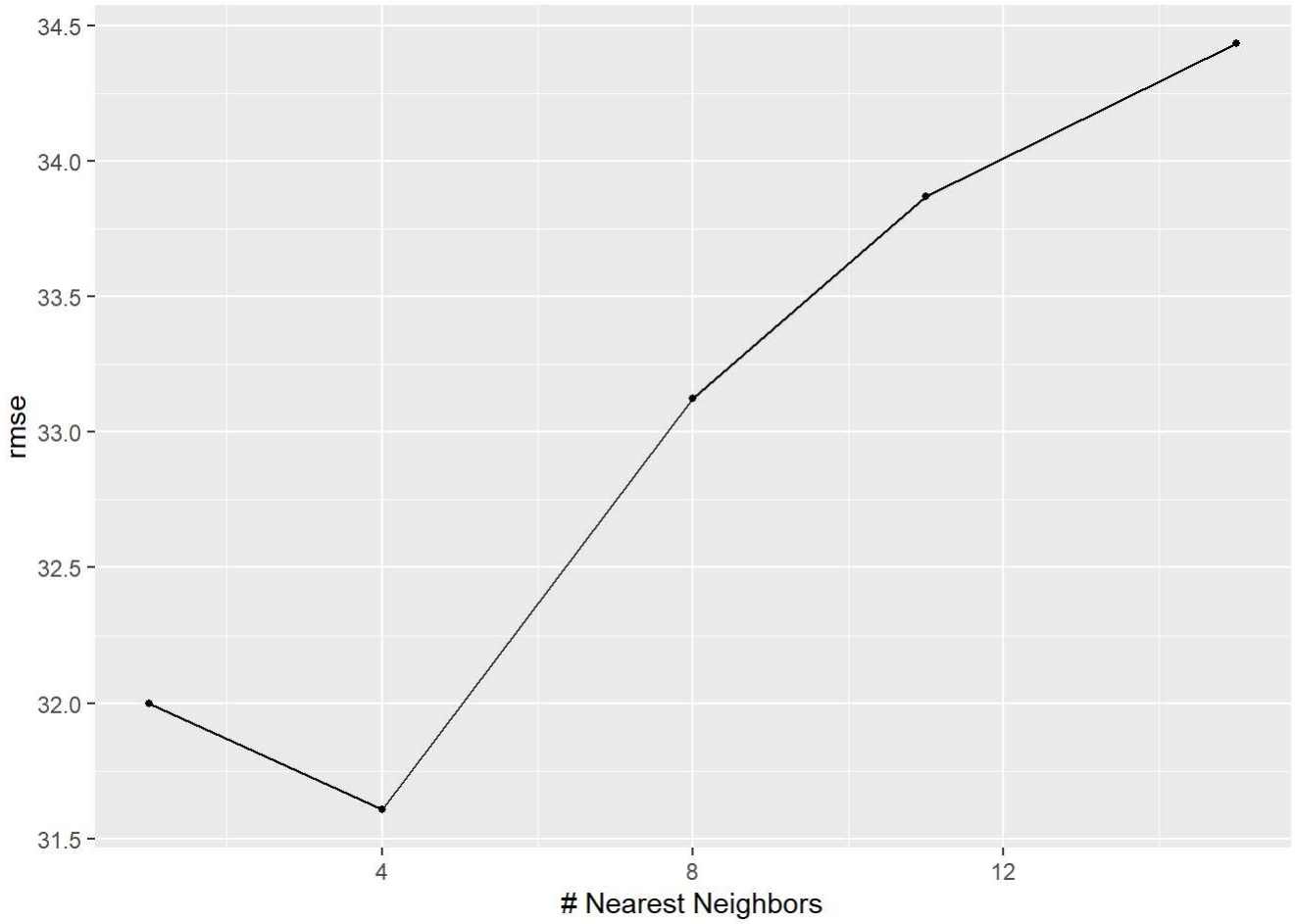
```
# ridge
autoplot(ridge_tuned, metric = "rmse")
```



K nearest Neighbors: Except for the start which is a constant rate of decrease, as number of neighbors increases, rmse also increases. The lowest rmse appears at the beginning, when neighbor = 4.

Hide

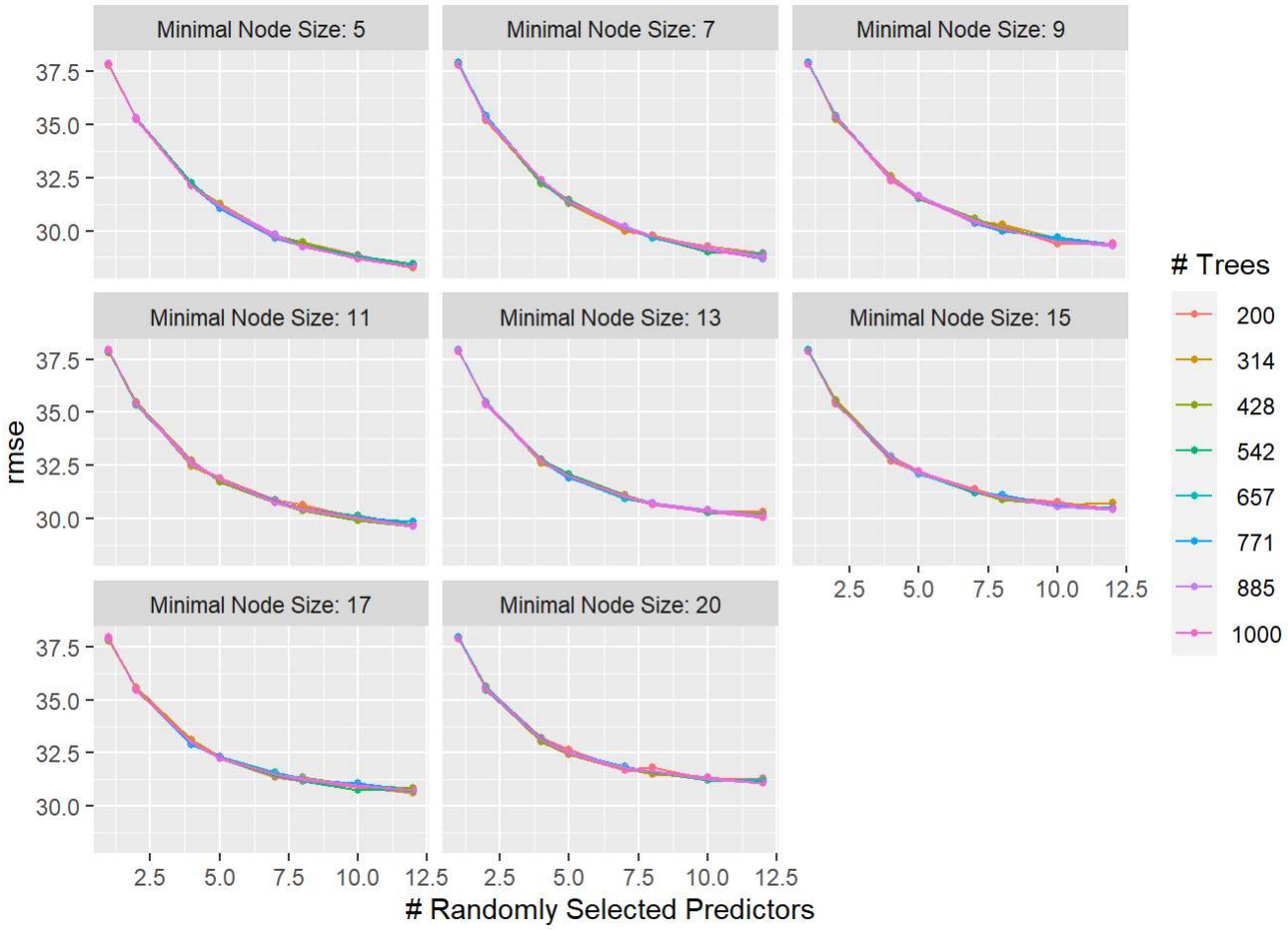
```
# knn
autoplot(knn_tuned, metric = "rmse")
```



Random forest: The functions appear to be decreasing log graphs. Lowest rmse appears when trees= 428, minimal node size is 5 or 7.

Hide

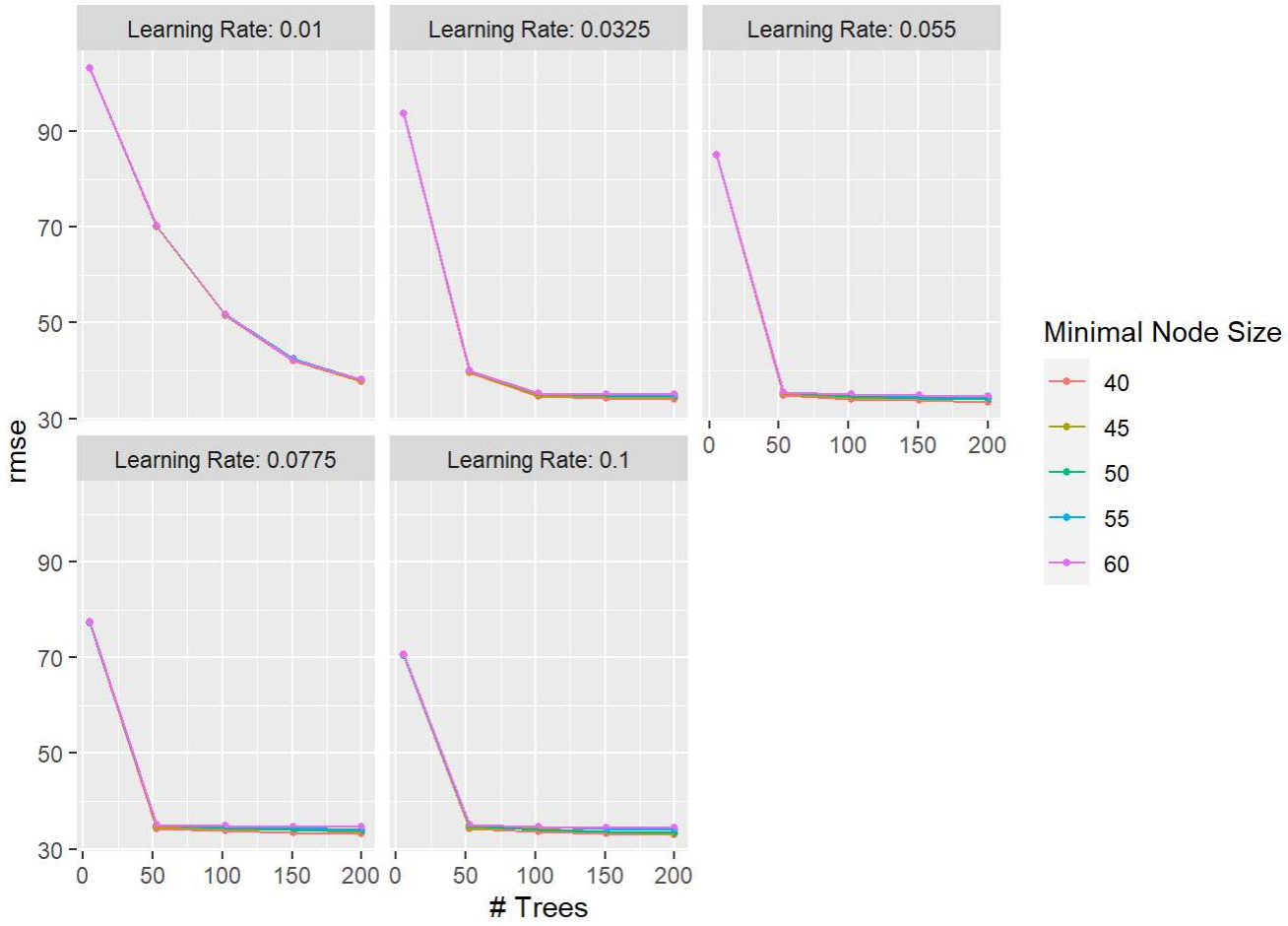
```
# random forest
autoplot(rf_tuned, metric = "rmse")
```



Boosted Trees: As trees increase from 0 to 200, rmse generally has a sharp decrease and remains constant. For node size = 40, we see that rmse takes on a more log function pattern. Nevertheless as trees increase rmse decreases. The lowest rmse appears at trees = 5, learn rate = 0.1.

Hide

```
# boosted trees
autoplot(boosted_tuned, metric = "rmse")
```



Best Model and Evaluating Prediction Performance Linear regression was our best performing model, with a rmse of almost 0. Now we need to fit the model to the training data and test it.

Hide

```
#select the best model
final_model <- select_best(rf_tuned)

#set up workflow for the last model
final_workflow <- finalize_workflow(rf_workflow, final_model)

#fit the model to the training set
final_fit <- fit(final_workflow, data = glassdoor_train)

#Evaluate the model performance on the testing set
multi_metric <- metric_set(rmse, rsq, mae)
final_predict <- predict(final_fit, glassdoor_test) %>%
  bind_cols(glassdoor_test %>% select(avg_salary))
multi_metric(final_predict, truth = avg_salary, estimate = .pred)
```

.metric	.estimator	.estimate
<chr>	<chr>	<dbl>
rmse	standard	24.524
rsq	standard	0.572

.metric	.estimator	.estimate
<chr>	<chr>	<dbl>
mae	standard	17.994

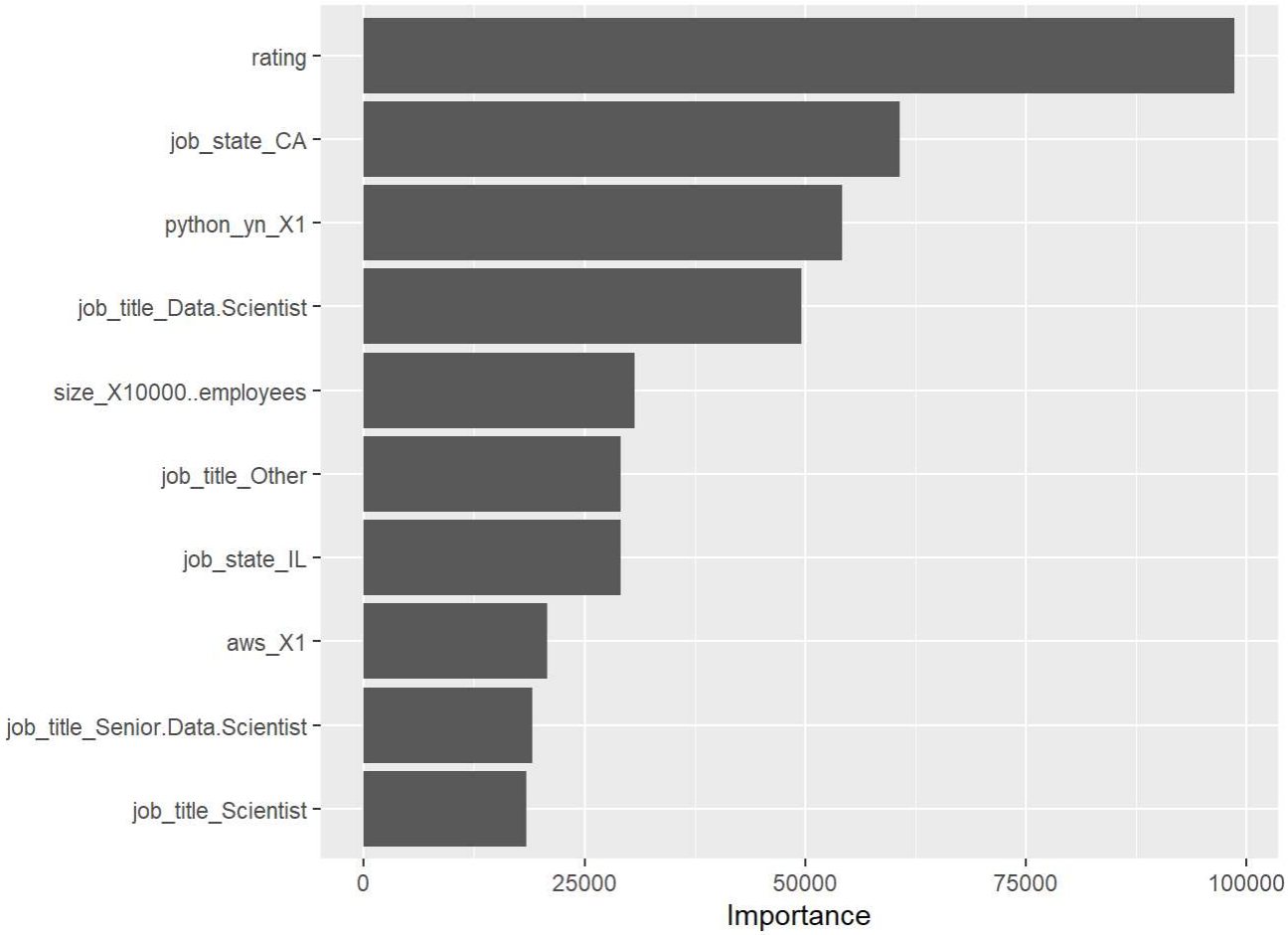
3 rows

By looking at the prediction results, we can see that random forest has a pretty good prediction performance for average salary. Rmse = 24.36, which is pretty large but rsq is 0.57, which means less than 57% of the variance can be explained by the model.

Variable Importance

Hide

```
final_fit %>% extract_fit_engine() %>% vip()
```



As expected, we see that rating is the most important variable, followed by working in California and knowing Python, perhaps the most common qualification listed for Data scientists.

Conclusion

In conclusion, the best performing model is random forest in comparison to linear regression, lasso, ridge, k-nearest neighbors, and boosted trees. This does not come across as a surprise since random forest is the best fitting model for a lot of data as it is a nonparametric, so it makes no assumptions on

outcomes or distributions. However, since the rmse was so high it appears the model doesn't actually perform that well.

Overall, it appears that on Glassdoor, for obvious reasons commpanies with higher ratings will have higher average salaries. Knowing Python and working in the state of California as a Data Scientist is the highest income. This does not come across as surprising since many tech companies are concentrated in California, and Python is a common listed qualification for data scientists. Larger companies with 10k+ employees tend to also have higher salaries. The second best state to work in is Illinois, followed by job positions looking for scientists. Apart from Python, AWS and excel are also common qualifications listed.

It is unexpected that the other skillsets are not considered as important, like Spark. I was under the impression that tech jobs usually list many qualifications, with many of them being the same, so I was surprised that they were not of higher importance.

Potential issues that we ran into while analyzing the data is that because most of our predictors were categorical, how effective were they truly at predicting regression? To tack on to that, each categorical variable had many responses. For example, the US has 50 states, so `job_state` already had 50 different responses that required dummy coding. We also could not exactly view the correlation plot for these categorical variables unless we converted them into dummy variables, but then we would end up with far too many predictors in the correlation plot. Additionally, because we had to combine job titles to make analyzing the data an easier task, doing so may have removed crucial information, but we would be none the wiser.

Overall, the conclusions we reached were pretty much along the lines of what I expected; data science jobs in california looking for Python as a key skill has the highest average salary. However, due to the high rmse value it appears our model does a poor job at predicting average salary on Glassdoor.