Opening Glassdoor

Jon Wayland

January 21, 2019

Overview

Per Wikipedia, Glassdoor is a "website where employees and former employees anonymously review companies and their management". One other feature that Glassdoor has is the estimated salary for open-listed positions. The website uses anonymous salary reports among other inputs to determine this amount. For job-seekers, this feature can be very useful when determining where to apply.

The goal of this project is to scrape data from Glassdoor for job searches on key words 'data scientist' and predict Glassdoor's salary estimate for roles where this feature is not provided. The metric that will be used to assess performance will be root mean squared error (RMSE). The data that is gathered include the following fields for 1,085 job postings (code to scrape can be found in the second section of this report):

- `URL`: The url to the job posting
- `Description`: A character string containing the job description (very lengthy)
- `Job_Title`: A character string containing the title of the position
- `Company_Rating`: Overall rating of the company (1-5)
- `Company_Name`: The name of the company
- `Company_Location`: The city and state of the position
- `Estimated_Salary`: Glassdoor's salary estimate for the position
- `Location_Code`: The code assigned to the location of the search

For the reader's convenience, I posted the scraped datasets on my GitHub (https://github.com/JonWayland/Opening-Glassdoor) as both a .csv and .Rda extension. In the third section of this report, I will identify additonal features that will be used for modeling the estimated salary. Most of these additional features are derived through analyzing the `Description` field.

Note: The data was gathered on January 20, 2019.

Gathering the Data

The data for this project was not in a clean and structured format, as is the case with most real-world data science projects. Using the R library `rvest`, I scraped the elements of interest from the first page of results for key words 'data scientist' with a location filter.

To start, I load all necessary libraries:

```
library(rvest) # For web-scraping
library(stringr) # For text manipulation and analysis
library(tidyverse) # For dplyr operations and ggplot2 visualizations
library(data.table) # For tabular functionality
library(caret) # For data split and machine learning
library(kableExtra) # For in-report tables
```

Next, I will define helper functions for both web-scraping and string manipulation:

```
# Using substr from the right-hand side
substrRight <- function(x, n){
  substr(x, nchar(x)-n+1, nchar(x))
}

# Specifying the url given a specific location code
urlByLocation <- function(location_code){
  paste0("https://www.glassdoor.com/Job/jobs.htm?suggestCount=0&suggestChosen=false",
         "&clickSource=searchBtn&typedKeyword=data+scientist&sc.keyword=data+scientist&locT=C&locId=",
        location_code,
        "&jobType=")
}
```

The scraping will utilize a search for 47 U.S. cities. I found these location codes by searching each city of interest and then pulling back the specific code. In order to loop through each location, the codes must be compiled into a list:

```
# Compiling list of locations
Location_Codes <- c(
"1152990", # Pittsburgh
"1152672", # Philadelphia
"1132348", # New York
"1154532", # Boston
"1153527", # Baltimore
"1138213", # Washington DC
"1138960", # Raleigh
"1147401", # San Francisco
"1139761", # Austin
"1154170", # Miami
"1148170", # Denver
"1131850", # Buffalo
"1144541", # Nashville
"1149603", # Las Vegas
"1146821", # Los Angeles
"1140494", # San Antonio
"1150505", # Seattle
"1151614", # Portland
"1128808", # Chicago
"1133904", # Phoenix
"1139977", # Dallas
```

```
"1147311", # San Diego
"1147436", # San Jose
"1154093", # Jacksonville
"1145845", # Columbus
"1139993", # Fort Worth
"1145013", # Indianapolis
"1138644", # Charlotte
"1134644", # Detroit
"1144463", # Memphis
"1136950", # Oklahoma City
"1137724", # Louisville
"1133579", # Milwaukee
"1133996", # Tucson
"1145778", # Cleveland
"1137959", # Albuquerque
"1147229", # Sacramento
"1131040", # Kansas City
"1155583", # Atlanta
"1147380", # Oakland
"1131270", # St. Louis
"1130337", # Arlington
"1154247", # Orlando
"1154429", # Tampa
"1140171", # Houston
"1148136", # Colorado Springs
"1130324"  # Virginia Beach
)
```

Next, I am defining an empty dataframe that includes the fields of interest (specified in the first section) that I will populate during the scraping step:

```
# Defining DataFrame
DSdata <- data.frame(
  URL = as.character(),
  Description = as.character(),
  Job_Title = as.character(),
  Company_Rating = as.character(),
  Company_Name = as.character(),
  Company_Location = as.character(),
  Estimated_Salary = as.character(),
  Location_Code = as.character()
)
```

As mentioned, the scraping needs to loop through each location code. The following `for` loop will do just that, while grabbing all information of interest and structuring it into a dataframe. In the event that the loop gets terminated due to an error, there are printed steps for debugging. For purposes of this demonstration, the output will be suppressed as each step is printed for each of the 47 locations:

```
for(j in 1:length(Location_Codes)){

  print(paste("----Step 1 for location ", Location_Codes[j], "(",j,")"))

  # Clearing the url_lsit
  url_list<-data.frame(URL=as.character())
```

```
# Pulling the information for specified URL
d<-read_html(urlByLocation(Location_Codes[j])) %>%
  html_nodes(xpath='//*[@id="JobSearch"]') %>%
  html_text()

print(paste("----Step 2 for location ", Location_Codes[j], "(",j,")"))

# Pulling back all specific posting urls
dt<-data.frame(str_extract_all(d, 'https://.*'))
colnames(dt)<-c("URL")
url_list<-rbind(url_list,dt)

print(paste("----Step 3 for location ", Location_Codes[j], "(",j,")"))

# Defining the URL field
url_list$URL<-as.character(url_list$URL)
url_list$URL<-substr(url_list$URL,1,nchar(url_list$URL)-1)

print(paste("----Step 4 for location ", Location_Codes[j], "(",j,")"))

# Pulling the job description
url_list$Description<-NA

print(paste("----Step 5 for location ", Location_Codes[j], "(",j,")"))

for(i in 1:nrow(url_list)){
  tryCatch({
    link <- as.character(url_list$URL[i])
    web<-read_html(link)
    url_list$Description[i] <- web %>% html_node(".desc") %>% html_text()
  }, error=function(e){})
}

print(paste("----Step 6 for location ", Location_Codes[j], "(",j,")"))

# Pulling the job title
url_list$Job_Title<-NA

for(i in 1:nrow(url_list)){
  tryCatch({
    link <- as.character(url_list$URL[i])
    web<-read_html(link)
    url_list$Job_Title[i] <- web %>%
      html_nodes(xpath='//*[@id="HeroHeaderModule"]/div[3]/div[1]/div[2]/div[1]/h2') %>%
      html_text()
  }, error=function(e){})
}

print(paste("----Step 7 for location ", Location_Codes[j], "(",j,")"))

# Pulling the company rating
url_list$Company_Rating<-NA
```

```r
for(i in 1:nrow(url_list)){
  tryCatch({
    link <- as.character(url_list$URL[i])
    web<-read_html(link)
    url_list$Company_Rating[i] <- web %>%
      html_nodes(xpath='//*[@id="HeroHeaderModule"]/div[3]/div[1]/div[2]/span[1]') %>%
      html_text()
  }, error=function(e){})
}
## Note: If the rating isn't populated then the company name will appear here

print(paste("----Step 8 for location ", Location_Codes[j], "(",j,")"))

# Pulling the company name
url_list$Company_Name<-NA

for(i in 1:nrow(url_list)){
  tryCatch({
    link <- as.character(url_list$URL[i])
    web<-read_html(link)
    url_list$Company_Name[i] <- web %>%
      html_nodes(xpath='//*[@id="HeroHeaderModule"]/div[3]/div[1]/div[2]/span[2]') %>%
      html_text()
  }, error=function(e){})
}
## Note: If the rating isn't populated then the company location will appear here

print(paste("----Step 9 for location ", Location_Codes[j], "(",j,")"))

# Pulling the company location
url_list$Company_Location<-NA

for(i in 1:nrow(url_list)){
  tryCatch({
    link <- as.character(url_list$URL[i])
    web<-read_html(link)
    url_list$Company_Location[i] <- web %>%
      html_nodes(xpath='//*[@id="HeroHeaderModule"]/div[3]/div[1]/div[2]/span[3]') %>%
      html_text()
  }, error=function(e){})
}
## Note: If the rating isn't populated then the company location will appear BLANK here

print(paste("----Step 10 for location ", Location_Codes[j], "(",j,")"))

# Pulling the company location
url_list$Estimated_Salary<-NA

for(i in 1:nrow(url_list)){
  tryCatch({
    link <- as.character(url_list$URL[i])
    web<-read_html(link)
    url_list$Estimated_Salary[i] <- web %>%
```

```
        html_nodes(xpath='//*[@id="salWrap"]/h2') %>%
        html_text()
    }, error=function(e){})
  }
  ## Note: If the rating isn't populated then the company location will appear BLANK here

  print(paste("----Step 11 for location ", Location_Codes[j], "(",j,")"))

  url_list <- url_list %>%
    mutate(
      Company_Location = case_when(nchar(Company_Rating) > 4 ~ Company_Name,
                                   TRUE ~ Company_Location),
      Company_Name = case_when(nchar(Company_Rating) > 4 ~ Company_Rating,
                               TRUE ~ Company_Name)
    ) %>%
    mutate(Company_Rating = case_when(nchar(Company_Rating) > 4 ~ "NA",
                                      TRUE ~ Company_Rating))

  print(paste("----Step 12 for location ", Location_Codes[j], "(",j,")"))

  url_list$Location_Code<-Location_Codes[j]

  DSdata <- rbind(DSdata,url_list)
}
```

This produces the following dataset labeled `DSdata`:

```
str(DSdata)
```

```
## 'data.frame':    1085 obs. of  8 variables:
##  $ URL             : chr  "https://www.glassdoor.com/job-listing/data-scientist-indeed-prime-JV_IC115
##  $ Description     : chr  "Indeed Prime is a free service that connects qualified job-seekers (that's
##  $ Job_Title       : chr  NA "Machine Learning Research Scientist (Entry-Level" "Data Scientist I" "D
##  $ Company_Rating  : chr  "NA" " 3.3" " 2.7" " 4.9" ...
##  $ Company_Name    : chr  NA " Software Engineering Institute" " United States Steel" " Duolingo" ..
##  $ Company_Location: chr  NA " - Pittsburgh, PA" " - Pittsburgh, PA" " - Pittsburgh, PA" ...
##  $ Estimated_Salary: chr  NA "$92,000/ year" "$126,000/ year" NA ...
##  $ Location_Code   : chr  "1152990" "1152990" "1152990" "1152990" ...
```

Exploratory Data Analysis and Feature Engineering

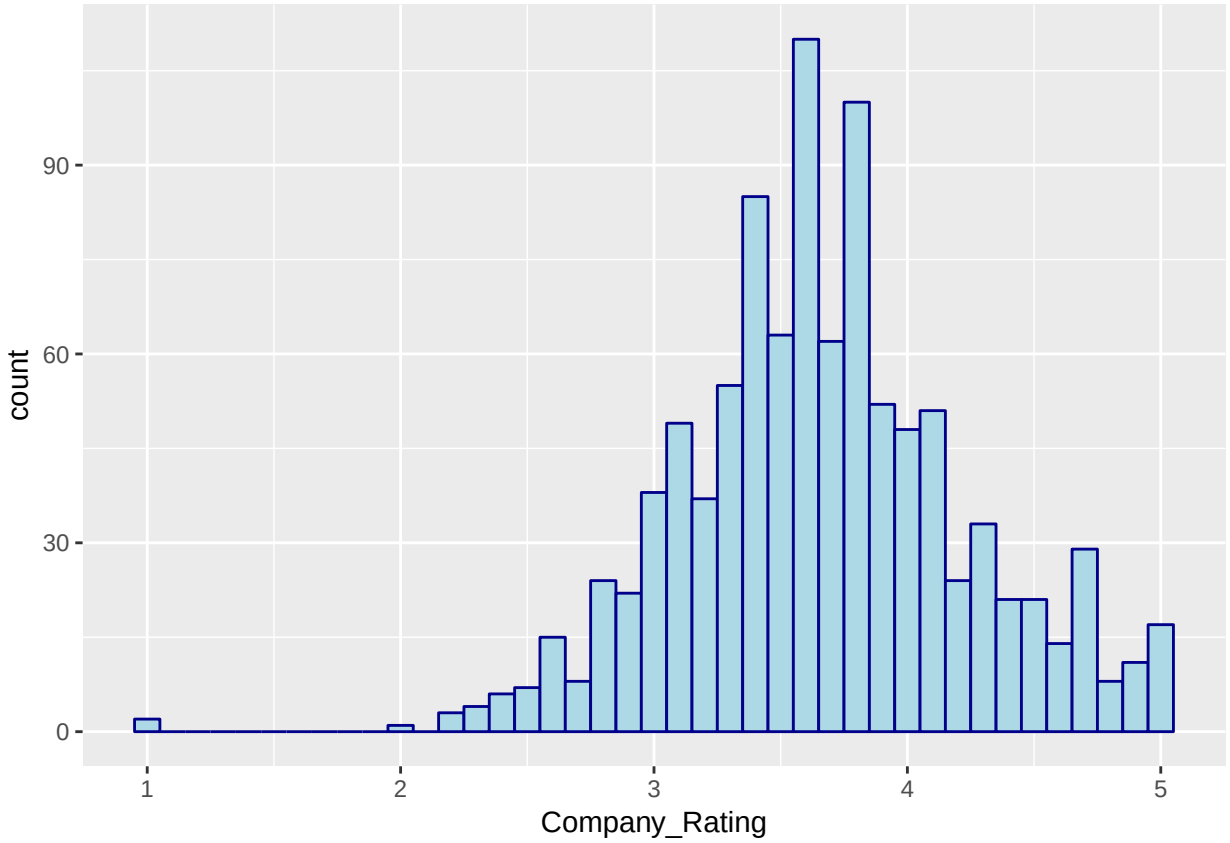
As can be seen, the data still needs some cleaning. The first thing I'm going to focus on is the company's rating. It is stored as a character field, and has obvious whitespace on the left-hand side. Additionally, it has `NA` fields as well as fields that I have previously forced to be `"NA"`. So, as an effort to clean this up, I convert `NA` values to `"NA"`, and remove the leading whitespace:

```
DSdata <- DSdata %>%
  mutate(Company_Rating = case_when(is.na(Company_Rating) ~ "NA",
                                    TRUE ~ Company_Rating)) %>%
  mutate(Company_Rating = trimws(Company_Rating, which = "left"))

# Seeing how many NAs
DSdata %>%
  filter(is.na(Company_Rating)) %>%
  nrow()
```

```
## [1] 0
```

As having a missing value for `Company_Rating` indicates that Glassdoor probably does not have a lot of information about the company and there are only 65 instances where it is missing, I am opting to drop these observations with the assumption that the estimated salary is not reliable:

```
DSdata <- DSdata %>%
  filter(Company_Rating != "NA") %>%
  mutate(Company_Rating = as.numeric(Company_Rating))

# Visualize the distribution
DSdata %>%
  ggplot(aes(Company_Rating))+
  geom_histogram(color="darkblue",fill="lightblue",binwidth=0.1)
```



The next field that has shown to be problematic is the `Estimated_Salary` field.

```
DSdata %>% select(Estimated_Salary) %>% unique %>% head()
```

```
##   Estimated_Salary
## 1    $92,000/ year
## 2   $126,000/ year
## 3              <NA>
## 4    $48,000/ year
## 5   $108,000/ year
## 8   $113,000/ year
```

As it contains unwanted characters, spaces, and inconsistent measurements, there are a few things that need to be done. This can all be done in one series of steps through `dplyr` syntax:

1. Remove all special characters (i.e. anything that is non-alphanumeric)
2. Decipher which instances have an hourly rate vs. annual salary
3. Remove the word year
4. Remove the word hour
5. Eliminate all whitespace
6. Convert the value to numeric
7. Convert the hourly rates to an annual salary and round to the nearest thousand

```
DSdata<-DSdata %>%
  mutate(Estimated_Salary = str_replace_all(Estimated_Salary, "[^[:alnum:]]", " ")) %>%
  mutate(Conversion = case_when(Estimated_Salary %like% 'hour' ~ 1, TRUE ~ 0)) %>%
  mutate(Estimated_Salary = gsub("year","",Estimated_Salary)) %>%
  mutate(Estimated_Salary = gsub("hour","",Estimated_Salary)) %>%
```

```
  mutate(Estimated_Salary = gsub(" ","",Estimated_Salary)) %>%
  mutate(Estimated_Salary = as.numeric(Estimated_Salary)) %>%
  mutate(Estimated_Salary = case_when(
    Conversion == 1 ~ round((Estimated_Salary*40*52)/1000)*1000,
    TRUE ~ Estimated_Salary)) %>%
  select(-c(Conversion))
```

Because much of what I want to use as predictors are hidden in the description field, I will need to make some efforts in analyzing it. A great method to do this is to use visualizations on word appearances, of which the most commonly used is a word cloud. For this step, I will need to leverage additional packages:

```
if(!require(tm))
  install.packages("tm",
                   repos = "http://cran.us.r-project.org") # For text-mining
if(!require(SnowballC))
  install.packages("SnowballC",
                   repos = "http://cran.us.r-project.org") # For word-collapsing algorithm
if(!require(wordcloud))
  install.packages("wordcloud",
                   repos = "http://cran.us.r-project.org") # For word cloud
if(!require(RColorBrewer))
  install.packages("RColorBrewer",
                   repos = "http://cran.us.r-project.org") # For color schemes
```

Next, I need to follow these steps:

1. Extract the text from each description in `DSdata` and store it as a Corpus object
2. Convert all text to lowercase, remove all special characters, and remove "stop words"
3. Convert the frequency of each word into a dataframe
4. Use the `wordcloud` function to plot these frequencies

```
# Defining the text
text <- VCorpus(VectorSource(DSdata$Description))

toSpace <- content_transformer(function (x , pattern ) gsub(pattern, " ", x))

# Certain Special Characters
text <- tm_map(text, toSpace, "/")
text <- tm_map(text, toSpace, "@")
text <- tm_map(text, toSpace, "\\|")

# Non-English Characters and Shared Links
text <- tm_map(text, toSpace, "[^[:graph:]]")

# Convert the text to lower case
text <- tm_map(text, content_transformer(tolower))

# Remove numbers
text <- tm_map(text, removeNumbers)

# Remove common English stopwords
stops <- stopwords("english")

text <- tm_map(text, removeWords, stops)
```

```r
# Remove your own stopwords as a character vector
text <- tm_map(text, removeWords, c("use","etc","also","like","one","will"))

# Remove punctuations
text <- tm_map(text, removePunctuation)

# Eliminate extra white spaces
text <- tm_map(text, stripWhitespace)

# Converting to matrix, sorting, and converting to dataframe
dtm <- TermDocumentMatrix(text)
m <- as.matrix(dtm)
v <- sort(rowSums(m),decreasing=TRUE)
d <- data.frame(word = names(v),freq=v)

# Plotting the word cloud
set.seed(7)
wordcloud(words = d$word, freq = d$freq, min.freq = 1,
          max.words=100, random.order=FALSE, rot.per=0.35,
          colors=brewer.pal(8, "Dark2"))
```

Based on these results and personal experience, I continue feature engineering by creating flags for instances where there is presence of buzz words pertaining to education and skills. In order to do this, I first convert the `Description` and `Job_Title` fields to upper-case and remove all special characters.

```
# Skills and Education Features
DSdata<-DSdata %>%
  mutate(Desc_Upper = toupper(str_replace_all(Description, "[^[:alnum:]]", " ")),
         Title_Upper = toupper(str_replace_all(Job_Title, "[^[:alnum:]]", " "))) %>%
  mutate(MS = case_when(Desc_Upper %like% 'MASTERS' |
                        Desc_Upper %like% ' MS ' |
                        substrRight(Desc_Upper,2) == ' MS' ~ 1, TRUE ~ 0)) %>%
  mutate(PHD = case_when(Desc_Upper %like% 'DOCTORATE' |
                         Desc_Upper %like% 'PHD' ~ 1, TRUE ~ 0)) %>%
  mutate(Statistics_Mathematics = case_when(Desc_Upper %like% 'STATISTICS' |
                                            Desc_Upper %like% 'MATHEMATICS' ~ 1,
                                            TRUE ~ 0)) %>%
  mutate(ComputerScience = case_when(Desc_Upper %like% 'COMPUTER SCIENCE' |
                                       Desc_Upper %like% ' CS ' ~ 1,
                                     TRUE ~ 0)) %>%
  mutate(Python = case_when(Desc_Upper %like% 'PYTHON' ~ 1,
                            TRUE ~ 0)) %>%
  mutate(R = case_when(Desc_Upper %like% ' R ' |
                         substrRight(Desc_Upper,2) == ' R' ~ 1, TRUE ~ 0)) %>%
  mutate(Scala = case_when(Desc_Upper %like% 'SCALA' ~ 1, TRUE ~ 0)) %>%
  mutate(SAS = case_when(Desc_Upper %like% 'SAS' ~ 1, TRUE ~ 0)) %>%
  mutate(TensorFlow = case_when(Desc_Upper %like% 'TENSORFLOW' ~ 1, TRUE ~ 0)) %>%
  mutate(Matlab = case_when(Desc_Upper %like% 'MATLAB' ~ 1, TRUE ~ 0)) %>%
  mutate(MachineLearning = case_when(Desc_Upper %like% 'MACHINE LEARNING' ~ 1,
                                     TRUE ~ 0)) %>%
  mutate(DeepLearning = case_when(Desc_Upper %like% 'DEEP LEARNING' ~ 1,
                                  TRUE ~ 0)) %>%
  mutate(NeuralNetwork = case_when(Desc_Upper %like% 'NEURAL NETWORK' ~ 1,
                                   TRUE ~ 0)) %>%
  mutate(StatisticalProgramming = case_when(Desc_Upper %like% 'STATISTICAL PROGRAMMING' ~ 1,
                                            TRUE ~ 0)) %>%
  mutate(EntryLevel = case_when(Desc_Upper %like% 'ENTRY LEVEL' |
                                Title_Upper %like% 'ENTRY LEVEL' ~ 1, TRUE ~ 0)) %>%
  mutate(Sr_Title = case_when(Title_Upper %like% 'SR' |
                              Title_Upper %like% 'SENIOR' ~ 1, TRUE ~ 0)) %>%
  mutate(Jr_Title = case_when(Title_Upper %like% 'JR' |
                              Title_Upper %like% 'JUNIOR' ~ 1, TRUE ~ 0)) %>%
  mutate(Intern = case_when(Title_Upper %like% 'INTERN' ~ 1, TRUE ~ 0)) %>%
  mutate(Specialized_R_Python = case_when(Title_Upper %like% ' R ' |
                                            substrRight(Title_Upper,2) == ' R' |
                                            Title_Upper %like% 'PYTHON' ~ 1,
                                            TRUE ~ 0)) %>%
  mutate(Consultant = case_when(Title_Upper %like% 'CONSULTANT' ~ 1, TRUE ~ 0)) %>%
  mutate(Lead = case_when(Title_Upper %like% 'LEAD' ~ 1, TRUE ~ 0)) %>%
  mutate(Analyst = case_when(Title_Upper %like% 'ANALYST' ~ 1, TRUE ~ 0)) %>%
  mutate(Scientist = case_when(Title_Upper %like% 'SCIENTIST' ~ 1, TRUE ~ 0))
```

In preparation for building the predictive model, I choose to join in the city of the search by using `Location_Code` and removing the now-unneeded fields `URL`, `Description`, `Desc_Upper`, `Job_Title`, and `Description`. Because I don't necessarily want to remove `DSdata` from memory, I will rename this new dataframe as `data`:

```
LocationDF <- data.frame(Location_Code = Location_Codes, City = c(
  "Pittsburgh","Philadelphia","New York","Boston","Baltimore","Washington DC","Raleigh",
  "San Francisco","Austin","Miami","Denver","Buffalo","Nashville","Las Vegas","Los Angeles",
  "San Antonio","Seattle","Portland","Chicago","Phoenix","Dallas","San Diego","San Jose",
  "Jacksonville","Columbus","Fort Worth","Indianapolis","Charlotte","Detroit","Memphis",
  "Oklahoma City","Louisville","Milwaukee","Tucson","Cleveland","Albuquerque","Sacramento",
  "Kansas City","Atlanta","Oakland","St. Louis","Arlington","Orlando","Tampa","Houston",
  "Colorado Springs","Virginia Beach"))

data <- DSdata %>%
  select(-c(URL, Description, Desc_Upper, Job_Title)) %>%
  inner_join(LocationDF, by = "Location_Code")
```

```
## Warning: Column `Location_Code` joining character vector and factor,
## coercing into character vector
```

At this point, I have everything that I want for predictions and can set aside the job postings that do not have an estimated salary so that I can conclude with providing those estimates:

```
forPrediction <- data %>% filter(is.na(Estimated_Salary))
data <- data %>% filter(!is.na(Estimated_Salary))
```

Machine Learning Model

Again, the goal of this project is to predict the estimated salary on Glassdoor data science job postings. In order to assess performance, I chose to minimize RMSE for two reasons:

1. It's a good metric for assessing the variance between predictions and actual estimates
2. The results are easily interpreted

Before getting started, I will split the data into a `train` and `test` set so that I can assess my performance on the `test` set. I define these as 90% and 10% of `data`, respectively.

```
test_index <- createDataPartition(data$Estimated_Salary, times = 1, p = 0.1, list = FALSE)
train <- data[-test_index,]
test <- data[test_index,]

nrow(train)
```

```
## [1] 691
```

```
nrow(test)
```

```
## [1] 79
```

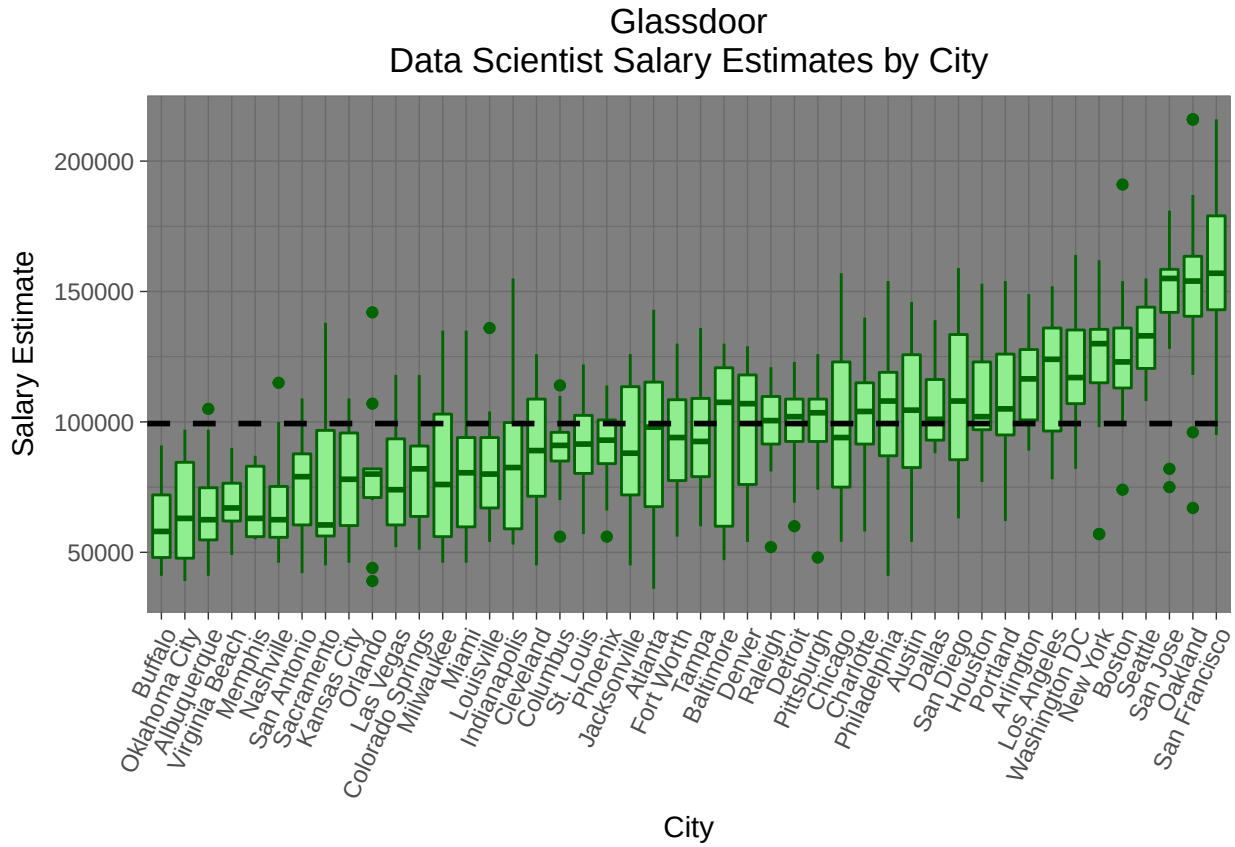
To start, I would like to get an idea of how the `Estimated_Salary` field is distributed:

```
train %>%
  ggplot(aes(Estimated_Salary))+
  geom_histogram(fill="pink", color="red")+
  xlab("Estimated Salary")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

Another way to view this distribution is to use a boxplot, and now that I have the `City` field, it is a good idea to see which cities have the highest salary estimates:

```
# Mean Salary Estimate
meanSal<-mean(train$Estimated_Salary)

train %>%
  mutate(City = reorder(City, Estimated_Salary)) %>%
  ggplot(aes(x = City, y = Estimated_Salary)) +
  geom_boxplot(color="darkgreen", fill = "lightgreen")+
  theme_dark()+
  theme(axis.text.x = element_text(angle = 65, hjust = 1))+
  geom_hline(yintercept = meanSal, lty = 2, col = "black", size = 1)+
  ggtitle("Glassdoor\nData Scientist Salary Estimates by City")+
  theme(plot.title = element_text(hjust = 0.5))+
  ylab("Salary Estimate")
```



According to this boxplot, it looks like the variance between cities might be an important indicator in predicting the estimated salary.

Prior to implementing any models, I want to define my formula so that I do not need to re-type it later. In doing this, I am adding in all of the predictors that I have previously defined:

```
# Setting up the formula for prediction
salaryFormula <- formula(Estimated_Salary ~ Company_Rating + MS + PHD + Python + R + Scala +
  SAS + TensorFlow + Matlab + MachineLearning + NeuralNetwork + StatisticalProgramming +
  DeepLearning + EntryLevel + Sr_Title + Jr_Title + Intern + Specialized_R_Python + Consultant +
  Lead + Analyst + Scientist + Statistics_Mathematics + ComputerScience + City)
```

As a baseline, I want to improve from using the mean as the salary estimate. All results are stored in the `results` dataframe for ease of comparison:

```
results <- data.frame(Model = "Baseline (Mean)",
                      RMSE = RMSE(test$Estimated_Salary, mean(train$Estimated_Salary)))
results %>% knitr::kable()
```

Model	RMSE
Baseline (Mean)	30799.09

First Improvement Attempt

For the first improvement to my baseline model, I want to use multiple linear regression:

```
# Fitting the linear model
fit <- lm(salaryFormula, data = train)
```

```r
# Seeing summary statistics on the fitted model
summary(fit)
```

```
##
## Call:
## lm(formula = salaryFormula, data = train)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -61651 -12056   -370  11309  68078
##
## Coefficients:
##                          Estimate Std. Error t value Pr(>|t|)
## (Intercept)              46416.27    8421.25   5.512 5.21e-08 ***
## Company_Rating            4529.99    1716.72   2.639 0.008530 **
## MS                        -136.87    1954.42  -0.070 0.944191
## PHD                         15.93    2220.97   0.007 0.994281
## Python                    5705.98    2294.71   2.487 0.013159 *
## R                        -1648.66    2001.92  -0.824 0.410516
## Scala                     -792.35    2319.30  -0.342 0.732742
## SAS                       1548.47    2030.45   0.763 0.445977
## TensorFlow               -3205.00    3577.15  -0.896 0.370619
## Matlab                   -3790.58    3127.36  -1.212 0.225946
## MachineLearning           4975.27    2096.67   2.373 0.017951 *
## NeuralNetwork            -1920.55    3118.67  -0.616 0.538236
## StatisticalProgramming   -6790.18    4391.04  -1.546 0.122524
## DeepLearning             -3458.61    3146.52  -1.099 0.272114
## EntryLevel              -10500.70    4522.85  -2.322 0.020572 *
## Sr_Title                 18050.41    3502.27   5.154 3.43e-07 ***
## Jr_Title                -14364.07    5708.36  -2.516 0.012110 *
## Intern                  -19759.06    4091.35  -4.829 1.73e-06 ***
## Specialized_R_Python      8080.43   21117.06   0.383 0.702111
## Consultant               -2777.60    6994.35  -0.397 0.691415
## Lead                      8145.16    8680.67   0.938 0.348450
## Analyst                 -24531.50    3270.15  -7.502 2.19e-13 ***
## Scientist                 5786.27    2940.49   1.968 0.049537 *
## Statistics_Mathematics    4701.07    2042.51   2.302 0.021687 *
## ComputerScience           2468.83    1967.72   1.255 0.210072
## CityArlington            41071.33    7037.27   5.836 8.60e-09 ***
## CityAtlanta              22739.17    7038.14   3.231 0.001299 **
## CityAustin               29147.53    6885.53   4.233 2.65e-05 ***
## CityBaltimore            26652.39    7565.15   3.523 0.000458 ***
## CityBoston               50902.08    7251.76   7.019 5.86e-12 ***
## CityBuffalo               1635.74    7681.68   0.213 0.831443
## CityCharlotte            33188.45    6952.68   4.773 2.26e-06 ***
## CityChicago              28465.88    6805.01   4.183 3.29e-05 ***
## CityCleveland            20387.20    7498.92   2.719 0.006737 **
## CityColorado Springs     14046.63    7490.42   1.875 0.061223 .
## CityColumbus             11993.78    7730.84   1.551 0.121311
## CityDallas               29687.74    7601.64   3.905 0.000104 ***
## CityDenver               21782.60    7851.35   2.774 0.005697 **
## CityDetroit              28566.97    7797.73   3.663 0.000270 ***
## CityFort Worth           25351.08    7523.91   3.369 0.000800 ***
## CityHouston              38587.55    7822.74   4.933 1.04e-06 ***
```

```
## CityIndianapolis        27886.60    8262.20   3.375 0.000784 ***
## CityJacksonville        21746.85    6983.82   3.114 0.001931 **
## CityKansas City          2757.75    8271.90   0.333 0.738954
## CityLas Vegas           14939.84    7450.20   2.005 0.045365 *
## CityLos Angeles         40340.65    7454.31   5.412 8.93e-08 ***
## CityLouisville          16479.22    7692.88   2.142 0.032571 *
## CityMemphis               325.87   10499.57   0.031 0.975251
## CityMiami               15644.17    8867.19   1.764 0.078177 .
## CityMilwaukee           10536.58    7614.22   1.384 0.166916
## CityNashville           18261.67    7576.47   2.410 0.016228 *
## CityNew York            43402.08    7071.69   6.137 1.50e-09 ***
## CityOakland             77717.84    7057.77  11.012  < 2e-16 ***
## CityOklahoma City        8686.41    7283.10   1.193 0.233450
## CityOrlando             14429.36    8094.89   1.783 0.075152 .
## CityPhiladelphia        33664.52    7226.12   4.659 3.89e-06 ***
## CityPhoenix             22826.64    7191.88   3.174 0.001578 **
## CityPittsburgh          26845.13    7581.41   3.541 0.000429 ***
## CityPortland            34734.76    7222.70   4.809 1.90e-06 ***
## CityRaleigh             25007.79    7872.49   3.177 0.001564 **
## CitySacramento          20014.63    7848.92   2.550 0.011012 *
## CitySan Antonio          6250.94    7496.73   0.834 0.404702
## CitySan Diego           38141.41    6965.17   5.476 6.32e-08 ***
## CitySan Francisco       82823.30    7115.45  11.640  < 2e-16 ***
## CitySan Jose            69060.43    6908.39   9.997  < 2e-16 ***
## CitySeattle             54322.19   12900.89   4.211 2.92e-05 ***
## CitySt. Louis           26928.36    7081.17   3.803 0.000157 ***
## CityTampa               14541.01    8437.45   1.723 0.085316 .
## CityVirginia Beach      15335.01    7418.65   2.067 0.039140 *
## CityWashington DC       42129.03    7145.00   5.896 6.11e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 20120 on 621 degrees of freedom
## Multiple R-squared:  0.6729, Adjusted R-squared:  0.6366
## F-statistic: 18.51 on 69 and 621 DF,  p-value: < 2.2e-16
```

```
results <- rbind(results,
                 data.frame(Model = "Linear Regression",
                            RMSE = RMSE(test$Estimated_Salary, predict(fit, newdata = test))))
results %>% knitr::kable()
```

Model	RMSE
Baseline (Mean)	30799.09
Linear Regression	21060.90

As can be seen, this provides a huge improvement. The RMSE decreased by roughly $10,000. Further, the model explains roughly 64% of the variance in the estimated salary, with all of the significant predictors shown above.

Second Improvement Attempt

For this improvement attempt, I want to explore a few machine learning models:

1. Decision Tree
2. Random Forest
3. Gradient Boosting

To do this, I want to fit each model with their default hyperparameters, and then choose the model with the best performance to tune:

```
# Keeping a consistent seed
seed <- 7

# 10-Fold Cross-Validation (Repeated 3 Times)
control<- trainControl(method="repeatedcv",
                       number=10,
                       repeats=3,
                       classProbs = TRUE)

########
# CART #
########
set.seed(seed)
fit.cart <- train(salaryFormula,
                  data=train,
                  method="rpart",
                  metric="RMSE",
                  trControl=control)


#################
# Random Forest #
#################
set.seed(seed)
fit.rf <- train(salaryFormula,
                data=train,
                method="rf",
                metric="RMSE",
                trControl=control)


#####################################################################
# Stochastic Gradient Boosting (Generalized Boosted Modeling) #
#####################################################################
set.seed(seed)
fit.gbm <- train(salaryFormula,
                data=train,
                method="gbm",
                metric="RMSE",
                trControl=control,
                verbose = FALSE)
```

Next, I take the predictions for each model and assess my performance on the `test` set:

```
results <- rbind(results, data.frame(
  Model = c("Decision Tree", "Random Forest", "Gradient Boosting"),
  RMSE = c(RMSE(test$Estimated_Salary,predict(fit.cart, newdata = test)),
           RMSE(test$Estimated_Salary,predict(fit.rf, newdata = test)),
           RMSE(test$Estimated_Salary,predict(fit.gbm, newdata = test)))
  ))
results %>% knitr::kable()
```

Model	RMSE
Baseline (Mean)	30799.09
Linear Regression	21060.90
Decision Tree	26415.52
Random Forest	22797.16
Gradient Boosting	21936.03

The linear model continues to have the best performance thus far. As `gbm` gave me the best results out of the three machine learning models tested in the second improvement attempt, I will try to optimize its hyperparameters.

Third Improvement Attempt

In this attempt, I will be taking the best-performing model of the three I tested above (i.e. `gbm`) and optimizing the hyperparameter choices. To do this, I will first define a parameter grid which I will call `gbmGrid`:

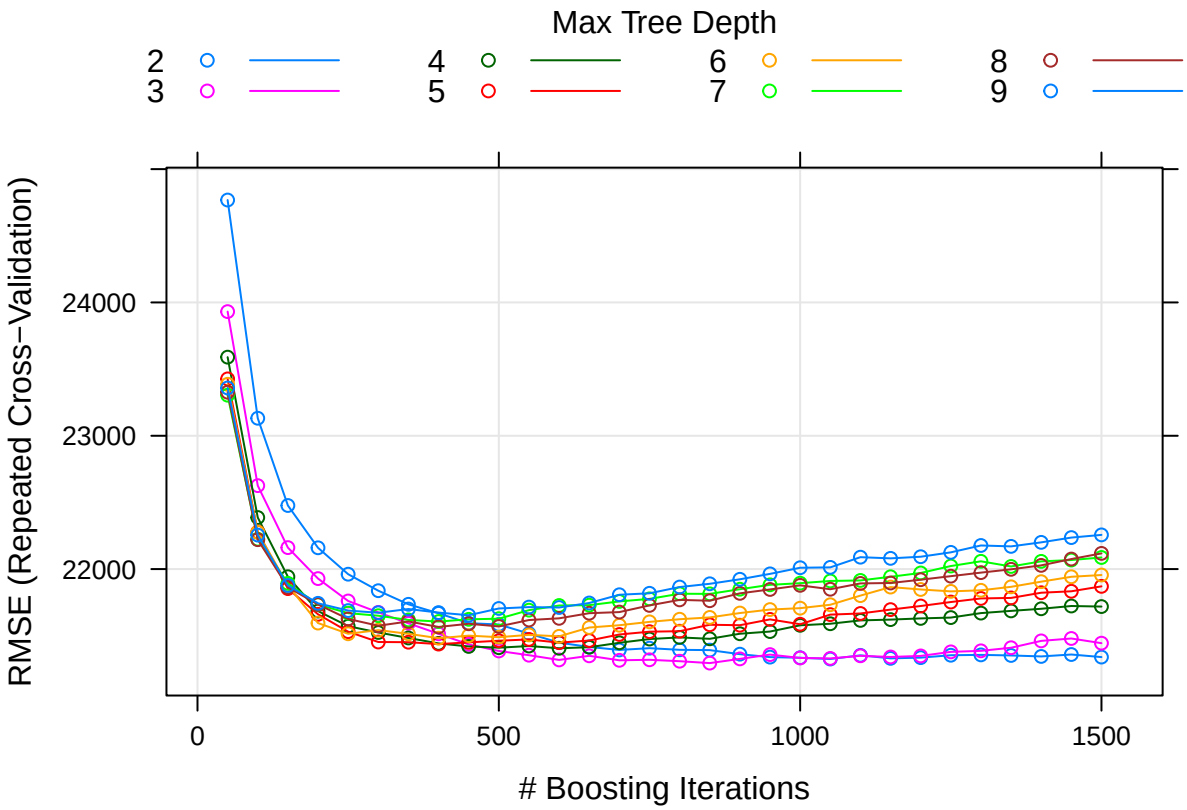
```
gbmGrid <-  expand.grid(interaction.depth = seq(2,9,1),
                        n.trees = (1:30)*50,
                        shrinkage = 0.05,
                        n.minobsinnode = 10)
nrow(gbmGrid)
```

```
## [1] 240
```

Based on this, I will be fitting 240 different models, and comparing the performance of each to find the best combination of hyperparameters using cross-validation:

```
set.seed(seed)
gbmFit <- train(salaryFormula,
                data=train,
                method="gbm",
                metric="RMSE",
                trControl=control,
                verbose = FALSE,
                tuneGrid = gbmGrid)
```

```
# Plotting the model results by iteration
plot(gbmFit)
```




```
# Assigning the new prediction performance
results <- rbind(results, data.frame(
  Model = c("Optimized Gradient Boosting"),
  RMSE = RMSE(test$Estimated_Salary,predict(gbmFit, newdata = test)
)))
```

Results

```
results %>% knitr::kable()
```

Model	RMSE
Baseline (Mean)	30799.09
Linear Regression	21060.90
Decision Tree	26415.52
Random Forest	22797.16
Gradient Boosting	21936.03
Optimized Gradient Boosting	21938.05

Some times the simplest models are the best, and in this case, that continues to hold true. The multiple linear regression model was not only the easiest to implement, it had the best RMSE at just over $20,000. Given that the range in salary estimates go from below $50,000 to above $200,000, this isn't a bad performance.

As was part of this project, I will end with making predictions on the job postings that did not have an estimated salary:

```
# Re-fitting on the full dataset
fit<-lm(salaryFormula, data = data)

# Removing Tucson as there are no reported salaries for any postings
forPrediction <- forPrediction %>% filter(City != 'Tucson')

forPrediction$SalaryEstimate <- predict(fit, newdata = forPrediction)

# Viewing the predictions
forPrediction %>%
  select(Company_Name, Company_Rating, City, Title_Upper, SalaryEstimate) %>%
  head(5) %>% knitr::kable()
```

Company_Name	Company_Rating	City	Title_Upper	SalaryEstimate
Duolingo	4.9	Pittsburgh	DATA SCIENCE INTERN	88208.59
Software Engineering Institute	3.3	Pittsburgh	CHIEF SCIENTIST	103505.18
Cognistx	4.8	Pittsburgh	DATA SCIENTIST	113896.11
Gecko Robotics, Inc.	5.0	Pittsburgh	DATA ANALYST	80721.00
Avacend, Inc.	2.3	Philadelphia	DATA SCIENTIST	102993.33

```
# Top 5 Lowest Estimates
forPrediction %>%
  filter(!is.na(Company_Name)) %>% # In order to show the companies as well
  select(Company_Name, Company_Rating, City, Title_Upper, SalaryEstimate) %>%
  arrange(SalaryEstimate) %>%
  head(5) %>%
  knitr::kable() %>%
  kable_styling(latex_options="scale_down")
```

Company_Name	Company_Rating	City	Title_Upper	SalaryEstimate
Evergreen Health Services	2.7	Buffalo	DATA ANALYST	36978.57
Software Guidance & Assistance	3.7	Oklahoma City	DATA ANALYST	37589.48
EPMA	3.5	Memphis	DATA ANALYST	41736.61
UnitedHealth Group	3.3	San Antonio	DATA ANALYST	42837.62
Lummus Group	4.1	Louisville	JR MASTER DATA ANALYST	44473.36

```
# Top 5 Highest Estimates
forPrediction %>%
```

```
filter(!is.na(Company_Name)) %>% # In order to show the companies as well
select(Company_Name, Company_Rating, City, Title_Upper, SalaryEstimate) %>%
arrange(desc(SalaryEstimate)) %>%
head(5) %>%
knitr::kable() %>%
kable_styling(latex_options="scale_down")
```

Company_Name	Company_Rating	City	Title_Upper	SalaryEstimate
Everlaw	4.7	San Francisco	DATA SCIENTIST	170777.8
MasterClass	5.0	San Francisco	DATA SCIENTIST	168435.7
Plaid	4.8	San Francisco	DATA SCIENTIST	167186.6
Stanza	3.2	San Francisco	DATA SCIENTIST	163717.6
Everlaw	4.7	Oakland	DATA SCIENTIST	162210.4

To no surprise, the top 5 lowest estimates were analyst positions at companies with mostly lower ratings, while the top 5 highest estimates were scientist positions at companies with mostly higher ratings. Additionally as expected, data science intern positions get a lower estimate than data scientist positions.

Conclusion

In conclusion, Glassdoor salary estimates definitely seem to be a function of other provided information. The biggest shortcoming that I faced was that I do not have previously reported salaries per company, and thus was put at a disadvantage. That said, I still believe that the performance of the final model choice was not only interesting, but quite impressive given these limitations.

For further improvements, I would consider scraping more data from Glassdoor to have a larger sample. At the time of my writing this, there are 25,600 job postings for 'data scientist' in the United States. Surely adding to my dataset would only yield more accurate estimates.