Studying the Impact of Job Descriptions on Data Science Salaries

Team Humilty

CHLOE NEO TZE CHING ENG JING KEAT GOH WEI LUN GLENN
KANEKO YOSHIKI TAN WEI KEONG

2022-11-16

Abstract

Data mining is the process of sorting through large data sets to identify patterns and relationships, where we aim to gain meaningful insights through the use of algorithms to predict future trends in textual data. In this project, we aim to uncover key insights in data science salaries by comparing models that effectively associate keywords in job descriptions to high paying data science roles. The data is cleaned by preparing job description text for analysis, and filtering the keywords in job descriptions with a minimum occurrence of 40. Least Absolute Shrinkage and Selection Operator (LASSO) is then applied to select the most important variables to be used in three of the models. The models used are Multiple Linear Regression (MLR), Random Forest, XGBoost, Multivariate Adaptive Regression Splines (MARS), and Partial Least Squares (PLS), where their performances will be assessed in relation to one another. Across all models, MLR has the lowest Root Mean Square Error (RMSE) of 16.2. In other words, MLR is the best model that accurately predicts salary, and we identified keywords used in this model.

Contents

1 Introduction to the problem

1.1 Literature review

A rapid shift towards digitalisation of businesses has radically changed the employment landscape in Singapore, which means Singaporeans need to keep up with the changes if they wish to stay competitive at work (My Skills Future, 2021). Employers in Singapore are starting to place an emphasis on skills rather than education (Tan, 2021). New hires today are assessed not just by their qualifications and work history, but also by their soft skills as there are a variety of soft skills in demand (The Straits Times, 2021).

According to a new report by Instant Offices, 73% of Singaporean workers are dissatisfied with their jobs (Arora, 2022). When asked if they planned to change jobs over the following six months, 31% of respondents responded "yes" (Chong, 2022). Millions of workers worldwide are no longer willing to return home with just a fair paycheck. They prefer to know how well they are progressing towards a meaningful career, which is a wellness, freedom, security, and experience at work, so as to achieve job satisfaction. As a result, job seekers should take all these factors into consideration when applying for a job.

These factors can be further broken down into keywords in job descriptions. Keywords are crucial to job adverts because they allow job seekers to narrow their search related to a role, skill, or industry for suitable employment (Alexander, 2019). Suitable individuals are more likely to find the job post when hirers and recruiters add key terms and phrases that are relevant to a particular role. This increases the percentage of successfully matching job seekers with their ideal jobs.

1.2 Objective

As most employers will be impressed when they notice that the resume is customised, highlighting relevant skills and using certain keywords suited to that particular company or position, job seekers can use these findings to narrow down their job search based on their own preferences such as salary range or skills set. As such, this project aims to predict data science salary based on the keywords in job descriptions.

2 Dataset

Dataset: https://www.kaggle.com/datasets/nikhilbhathi/data-scientist-salary-us-glassdoor

2.1 Description of dataset

Import relevant libraries:

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --

## v ggplot2 3.3.6     v purrr   0.3.4
## v tibble  3.1.6     v dplyr   1.0.7
## v tidyr   1.1.4     v stringr 1.4.0
## v readr   2.1.1     v forcats 0.5.1

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(tm)
```

```
## Loading required package: NLP

##
## Attaching package: 'NLP'

## The following object is masked from 'package:ggplot2':
##
##     annotate
```

```
library(wordcloud)
```

```
## Loading required package: RColorBrewer
```

```
library(stringr)
library(GGally)
```

```
## Registered S3 method overwritten by 'GGally':
##   method from
##   +.gg   ggplot2
```

```
library(ggplot2)
library(reshape2)
```

```
##
## Attaching package: 'reshape2'

## The following object is masked from 'package:tidyr':
##
##     smiths
```

```
library(RColorBrewer)
library(scales)
```

```
##
## Attaching package: 'scales'

## The following object is masked from 'package:purrr':
##
##     discard
## The following object is masked from 'package:readr':
##
##     col_factor
```

```
library(text2vec)
library(caret)
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:purrr':
##
##     lift
```

```
library(fastDummies) # for creating dummy variables
library(randomForest)
```

```
## randomForest 4.7-1.1
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:dplyr':
##
##     combine
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
library(Metrics)
```

```
##
## Attaching package: 'Metrics'
```

```
## The following objects are masked from 'package:caret':
##
##     precision, recall
```

```
library(knitr)
library(earth)
```

```
## Loading required package: Formula
```

```
## Loading required package: plotmo
```

```
## Loading required package: plotrix
```

```
##
## Attaching package: 'plotrix'
```

```
## The following object is masked from 'package:scales':
##
##     rescale
```

```
## Loading required package: TeachingDemos
```

```
library(vip)
```

```
##
## Attaching package: 'vip'

## The following object is masked from 'package:utils':
##
##     vi
```

```
library(dplyr)
library(pls)
```

```
##
## Attaching package: 'pls'

## The following object is masked from 'package:caret':
##
##     R2

## The following object is masked from 'package:stats':
##
##     loadings
```

Below is a sample of our dataset:

```
# import data from local csv file
D <- read.csv("data_cleaned_2021.csv")
knitr::kable(head(D[, 20:25]),
             caption = "A sample of our dataset.")
```

Table 1: A sample of our dataset.

Avg.Salary.K.	company_txt	Job.Location	Age	Python	spark
72.0	Tecolote Research	NM	48	1	0
87.5	University of Maryland Medical System	MD	37	1	0
85.0	KnowBe4	FL	11	1	1
76.5	PNNL	WA	56	1	0
114.5	Affinity Solutions	NY	23	1	0
95.0	CyrusOne	TX	21	1	0

This dataset has 41 variables, consisting of numerical, categorical, and text. Some variables are derivations from others, and as such we will not be using all 41 variables.

2.2 Exploratory data analysis

2.2.1 Data cleaning

Note: the dataset downloaded has already been cleaned by the owner, but we will do some additional cleaning and data preparation so that it is suited for our needs.

1. Removing "\n" from job descriptions, cleaning job descriptions text, and creating a new variable to store lengths of job description texts:

```
# Keep only alphabets and spaces, changing texts to lowercase, and create a new variable to store lengt
D_clean <- D %>%
  # create a new variable containing only lowercase text from Job.Description
  mutate(cleaned_text = gsub("[^a-zA-Z0-9]", " ", Job.Description)) %>%
  mutate(cleaned_text = gsub("\\n", " ", cleaned_text)) %>%
  mutate(cleaned_text = tolower(cleaned_text))
```

2.2.2 Feature selection

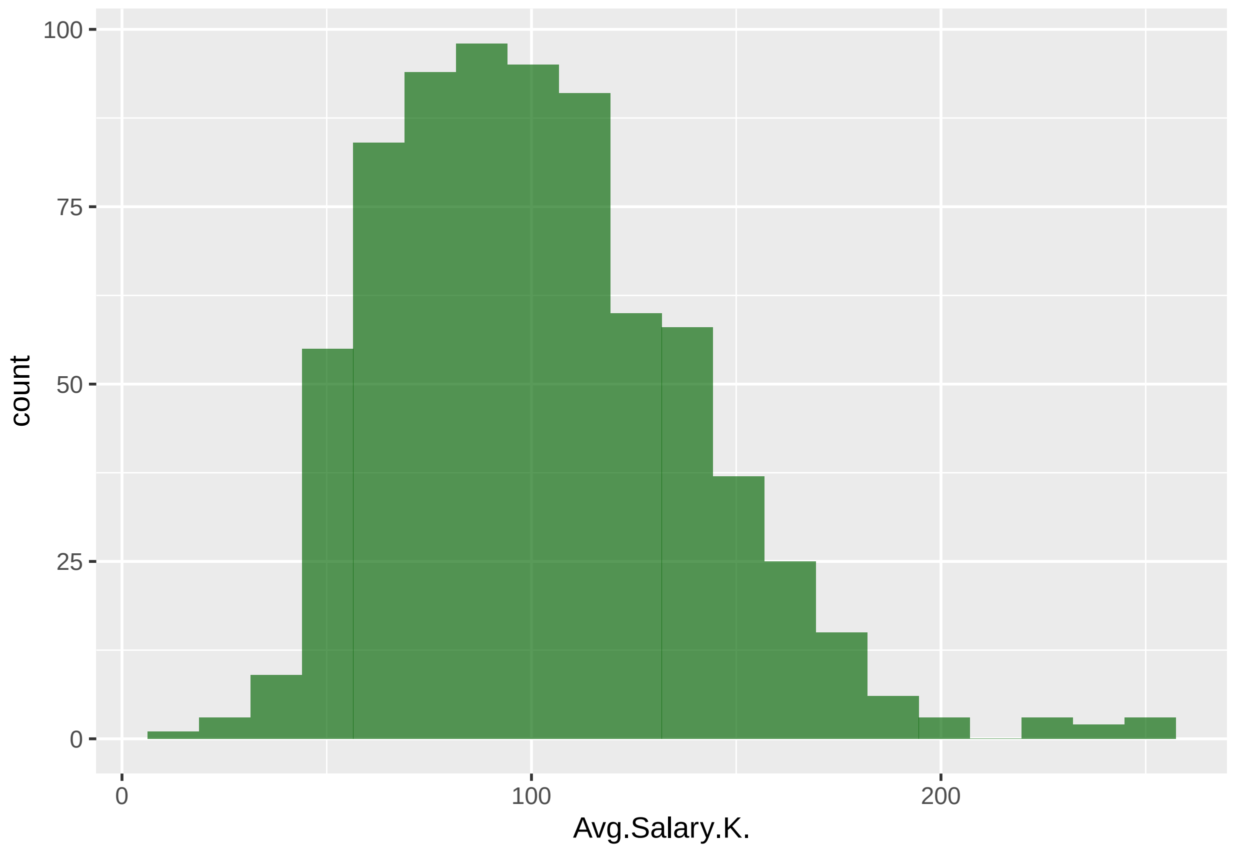
Our project's focus is on job descriptions and their relationship with data science salaries. As such, we will only keep these two variables

```
#remove unnecessary columns
D_clean <- D_clean %>%
  select(c(Avg.Salary.K., cleaned_text))
```

2.2.3 Data visualizations

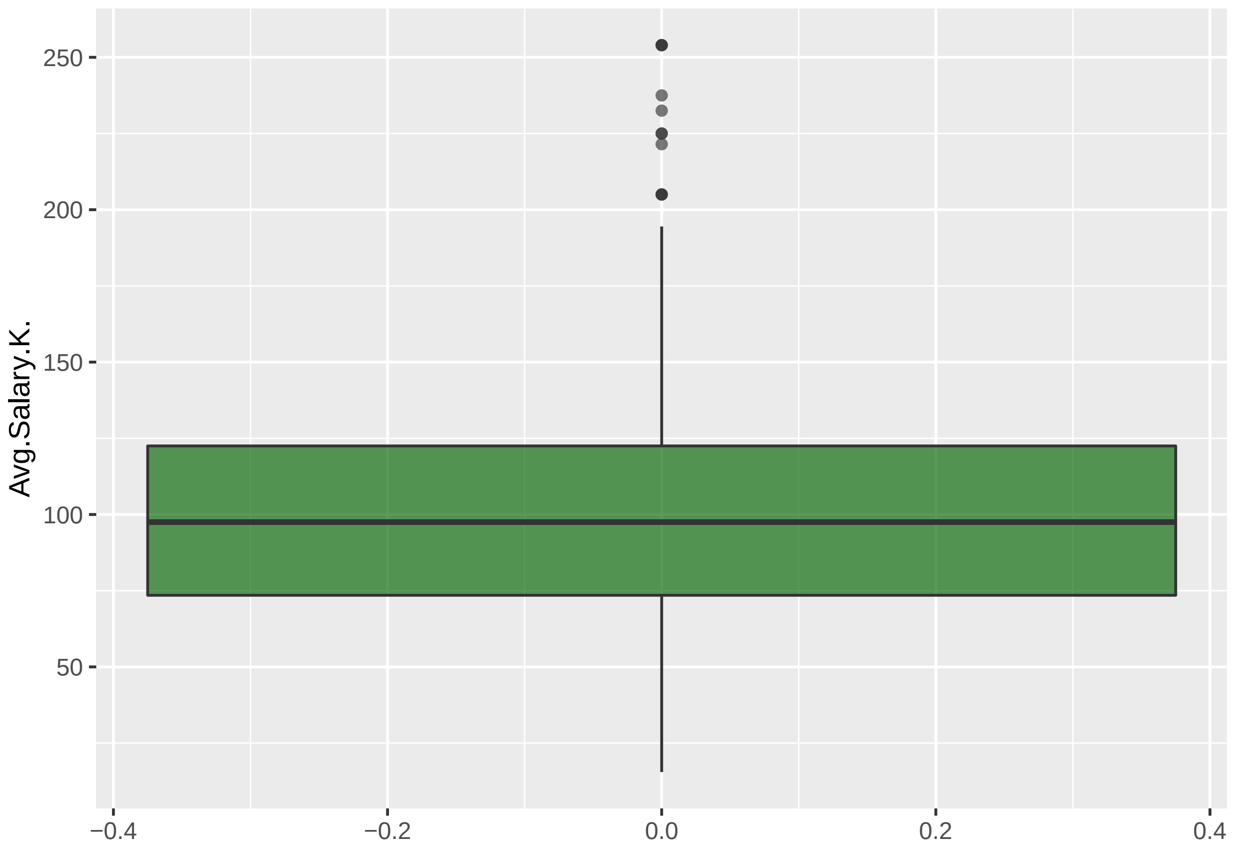
1. Histograms of average salaries in thousands:

```
ggplot(D_clean, aes(x=Avg.Salary.K.)) +
  geom_histogram(fill = "darkgreen", alpha = 0.65, bins = 20)
```



2. Boxplots of average salaries:

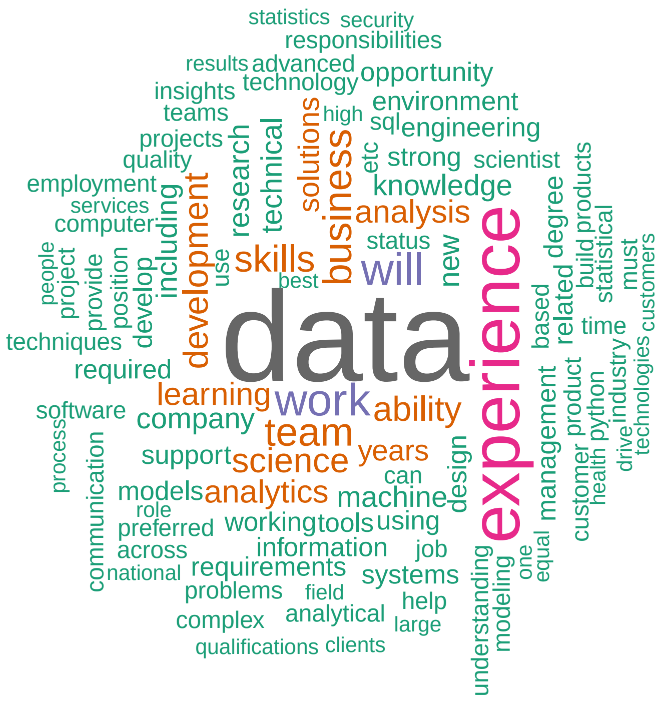
```
ggplot(D_clean, aes(y=Avg.Salary.K.)) +
  geom_boxplot(fill = "darkgreen", alpha = 0.65)
```



3. Word cloud for job descriptions:

We visualise the job descriptions with a word cloud after removal of words whose total frequency is below 40, and stopwords such as “the”, “he” etc. as these are common words that do not store any informational value in our analysis.

```r
visualize_text <- function(x) {
  # x is a character vector
  # the function will extract
  frequent_words <- termFreq(x)
  frequent_words <- frequent_words[!(names(frequent_words) %in% stopwords())]
  wordcloud(words = names(frequent_words),
            freq = frequent_words, min.freq = 40,
            max.words = 100, random.order=FALSE, rot.per=0.35,
            colors=brewer.pal(9, "Dark2"))

}

visualize_text(D_clean$cleaned_text)
```



We set the minimum occurrence of words to 40, and show the 100 most common words above.

2.2.4 Feature engineering

Next we create a document-term matrix for our job descriptions, setting minimum word frequency to 40.

```
corpus <- VCorpus(VectorSource(D_clean$cleaned_text))
dtm <- DocumentTermMatrix(corpus)

words_freq <- termFreq(D_clean$cleaned_text)
frequent_words <- words_freq[words_freq >= 40]

cat("Setting minimum word frequency to 40, we retain", length(frequent_words), "words out of the origina
```

```
## Setting minimum word frequency to 40, we retain 1306 words out of the original 10314 words.
```

```
frequent_words <- frequent_words[!(names(frequent_words) %in% stopwords())]
dtm <- dtm[ , names(frequent_words)]

cat("Our document-term matrix consists of", nrow(dtm), "job descriptions against", ncol(dtm), 'words pre
```

```
## Our document-term matrix consists of 742 job descriptions against 1238 words present in our vocab aft
```

Next, we create a new dataframe for our DTM.

```
D_dtm <- dtm %>%
  as.matrix %>%
  as_tibble %>%
  mutate(Y_salary = D_clean$Avg.Salary.K.) %>%
  select(c(Y_salary, everything()))

knitr::kable(head(D_dtm[, 1:5]),
             caption = "A sample of our DTM.")
```

Table 2: A sample of our DTM.

Y_salary	000	100	2020	401
72.0	0	1	0	0
87.5	0	0	0	0
85.0	0	0	0	0
76.5	0	0	2	0
114.5	0	0	0	0
95.0	0	0	0	0

3 Modelling

3.1 Feature selection using LASSO

We will use LASSO regularization to select features to prepare our data for three of our models: Multiple LInear Regression, Random Forest, and XGBoost. The last two models use their own methods of feature selection, so we will not use features selected by LASSO for those two models, but the entire dataset instead.

The LASSO procedure is as follows:

- We will try the following values of lambda for our LASSO regularization:

```
lambda <- 10^seq(-1, 0 , length = 10)
lambda
```

```
##  [1] 0.1000000 0.1291550 0.1668101 0.2154435 0.2782559 0.3593814 0.4641589
##  [8] 0.5994843 0.7742637 1.0000000
```

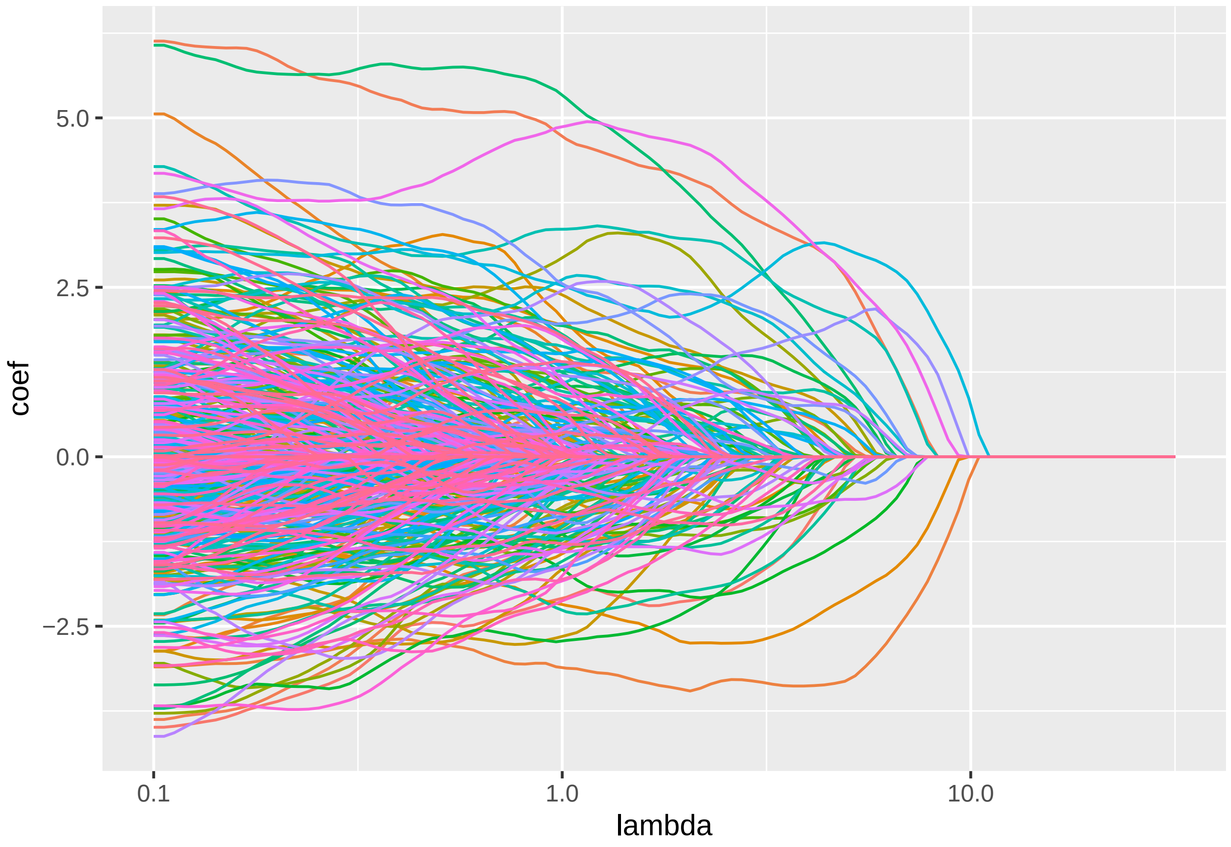
- Below we train our LASSO:

```
set.seed(100)

lasso <- train(
  Y_salary ~., data = D_dtm, method = "glmnet",
  trControl = trainControl("cv", number = 10),
  tuneGrid = expand.grid(alpha = 1, lambda = lambda),
  preProcess = c("center","scale")
)

lasso
```

```
## glmnet
##
##  742 samples
## 1238 predictors
##
## Pre-processing: centered (1238), scaled (1238)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 668, 668, 666, 668, 669, 668, ...
## Resampling results across tuning parameters:
##
##   lambda     RMSE      Rsquared   MAE
##   0.1000000  24.66909  0.6063107  13.82706
##   0.1291550  24.53151  0.6058293  14.03125
##   0.1668101  24.14206  0.6087953  14.26960
##   0.2154435  23.72767  0.6121538  14.48610
##   0.2782559  23.45697  0.6115914  14.90976
##   0.3593814  23.43562  0.6065680  15.45128
##   0.4641589  23.45016  0.6017484  16.05547
##   0.5994843  23.82772  0.5869512  16.88018
##   0.7742637  24.42560  0.5667097  17.74360
##   1.0000000  24.97889  0.5501730  18.56066
##
## Tuning parameter 'alpha' was held constant at a value of 1
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were alpha = 1 and lambda = 0.3593814.
```

- Let us plot coefficients of LASSO.

```
lambda_for_plotting <- 10^seq(from = -1, to = 1.5, length = 100)
lasso_coefs <- coef(lasso$finalModel, lambda_for_plotting) %>%
  as.matrix %>% t %>% as_tibble %>%
  mutate(lambda = lambda_for_plotting)

lasso_coefs %>%
  pivot_longer(-c(1, ncol(lasso_coefs)), names_to = "variable", values_to = "coef") %>%
  ggplot(aes(x = lambda, y = coef, group = variable, colour = variable)) +
  geom_line() + scale_x_log10() + theme(legend.position = "none")
```



- Now we store our variables selected by LASSO.

```
store <- filter(lasso_coefs, lambda == lasso$bestTune$lambda) %>%
  select_if(function(col) !all(col == 0))

selected_var <- names(store[2:(ncol(store)-1)])
```

- Finally we prepare our final dataset to be used for our models.

```
D_final <- D_dtm[c(gsub("`","", selected_var))] %>%
  mutate(Y_salary = D_dtm$Y_salary) %>%
  select(c(Y_salary, everything()))

knitr::kable(head(D_final[, 1:6]),
             caption = "A sample of our final dataset.")
```

Table 3: A sample of our final dataset.

Y_salary	2020	401k	academic	accordance	accountable
72.0	0	0	0	0	0
87.5	0	0	0	0	0
85.0	0	0	0	0	0
76.5	2	0	0	0	0
114.5	0	0	0	0	0
95.0	0	0	0	1	0

- Now we can prepare our training and test data for MLR, Random Forest, and XGBoost:

```
set.seed(100)
ind <- runif(nrow(D_final)) < 0.8

train_data <- D_final[ind , ]
test_data <- D_final[!ind , ]

cat("Dimensions of the training set are", dim(train_data), "\n")
```

```
## Dimensions of the training set are 585 355
```

```
cat("Dimensions of the test set are", dim(test_data), "\n")
```

```
## Dimensions of the test set are 157 355
```

3.2 Models

3.2.1 Multiple Linear Regression

The first model we use is Multiple Linear Regression, using the 354 features selected by LASSO.

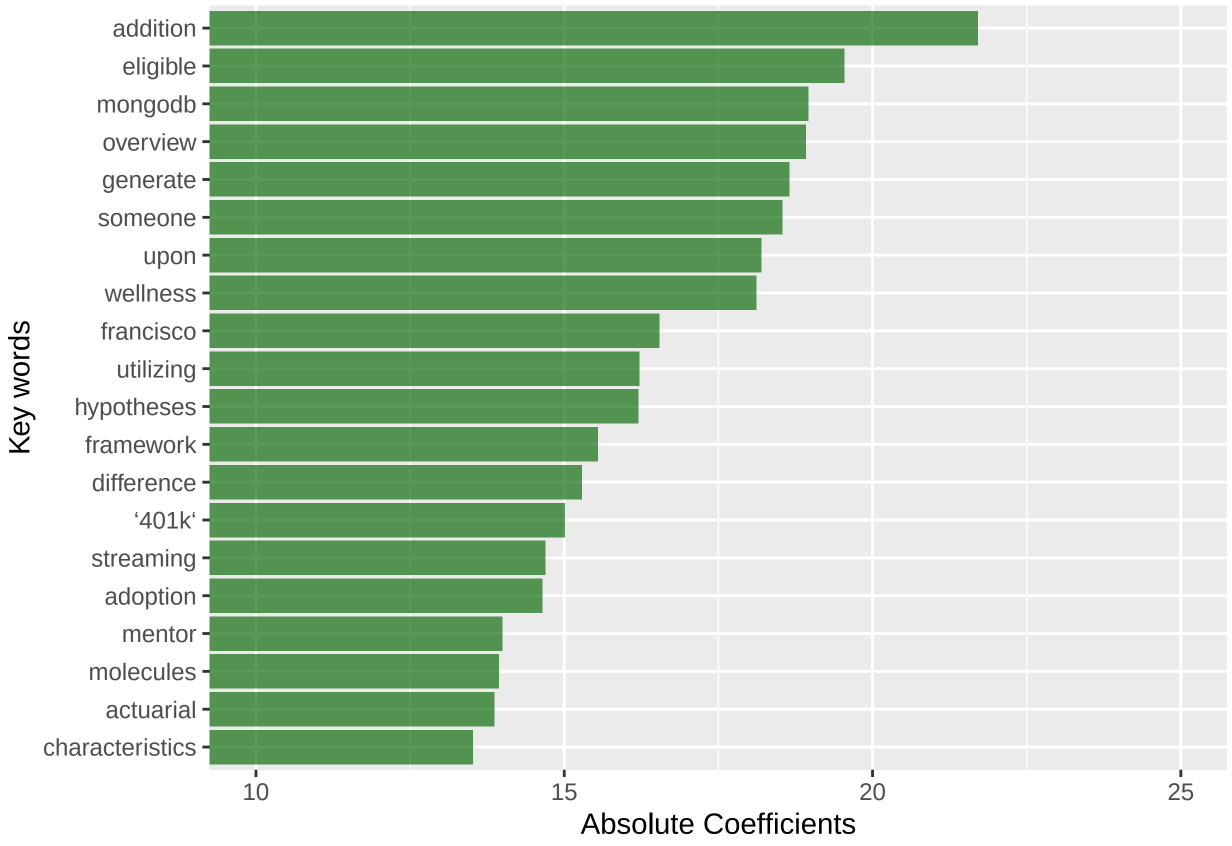
```
mlr_mod <- lm(Y_salary ~ .,
              data = train_data)

mlr_mod %>%
    predict(test_data) %>%
    rmse(test_data$Y_salary)
```

```
## [1] 16.20561
```

Our MLR model's RMSE value is 16.20561, the lowest we will achieve in this project.

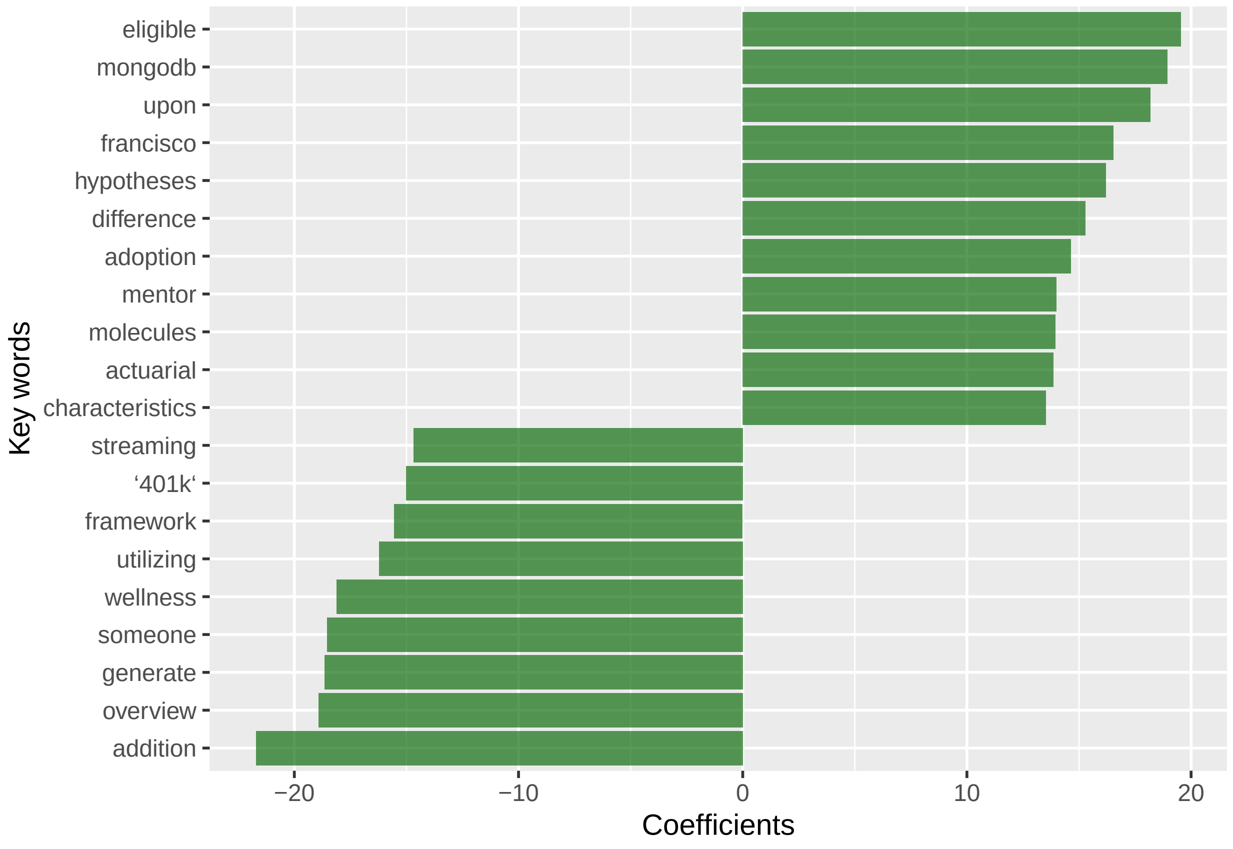
Below is a bar plot of the top 20 variables with the highest absolute coefficients.

```
var_coeff <- data.frame(coefficients(mlr_mod)) %>%
  arrange(desc(abs(coefficients(mlr_mod)))) %>%
  dplyr::slice(2:21)

ggplot(var_coeff, aes(x=abs(coefficients.mlr_mod.),
                      y=reorder(rownames(var_coeff), +abs(coefficients.mlr_mod.)))) +
  geom_bar(stat = "identity", fill="darkgreen", alpha = 0.65) +
  xlab("Absolute Coefficients") + ylab("Key words") +
  coord_cartesian(xlim = c(10, 25))
```



Below is a bar plot of the top 20 variables with the highest absolute coefficients, but this time we reflect their relationship with salary as well.

```
ggplot(var_coeff, aes(x=coefficients.mlr_mod.,
                      y=reorder(rownames(var_coeff), +coefficients.mlr_mod.))) +
  geom_bar(stat = "identity", fill="darkgreen", alpha = 0.65) +
  xlab("Coefficients") + ylab("Key words")
```



3.2.2 Random Forest

The next step is to visualize our data using decision trees. Nevertheless, decision trees alone have a significant variance, which can make the trees trained on various datasets appear quite distinct from one another. Random forest was chosen over a bagged ensemble as there may be one or more really powerful predictors in our dataset, preventing other predictors from having a chance to be included. As a result, the predictions made by the trees will be strongly correlated and increasing the number of trees will not make the variance smaller Therefore, in order to lower the variance, we shall propose an ensemble technique.

We performed grid search with oob (out-of-bag error) while tuning the random forest to select the optimal values of the hyper-parameters try (number of variables) and min.node.size (minimal node size):

```
#getting optimal hyper parameters
mod_rf <- train(Y_salary ~., data = D_final, method = "ranger",
                num.trees = 50, importance = 'impurity',
                trControl = trainControl("oob"))
mod_rf
```

```
## Random Forest
##
## 742 samples
## 354 predictors
##
## No pre-processing
## Resampling results across tuning parameters:
##
##   mtry  splitrule   RMSE      Rsquared   MAE
##     2   variance    24.57864  0.6858209  18.50812
##     2   extratrees  25.99420  0.6529647  20.11052
##   178   variance    19.06971  0.7515014  12.42904
##   178   extratrees  19.52720  0.7474824  12.88705
##   354   variance    19.26346  0.7460176  12.69761
##   354   extratrees  18.78437  0.7650315  12.31477
##
## Tuning parameter 'min.node.size' was held constant at a value of 5
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were mtry = 354, splitrule = extratrees
##  and min.node.size = 5.
```

The optimal values of the hyper-parameters are:

```
knitr::kable(mod_rf$bestTune)
```

	mtry	splitrule	min.node.size
6	354	extratrees	5

Now, we retrain the model with the selected hyperparameters to produce the following output:

```
rfGrid <- expand.grid(mtry = 178,
                      min.node.size = 5,
                      splitrule = "variance")

mod_rf_tune <- train(Y_salary ~., data = D_final, method = "ranger",
                     num.trees = 50, importance = 'impurity',
                     tuneGrid = rfGrid, trControl = trainControl("oob"))
mod_rf_tune
```

```
## Random Forest
##
## 742 samples
## 354 predictors
##
## No pre-processing
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   18.97822  0.7547173  12.41199
##
## Tuning parameter 'mtry' was held constant at a value of 178
## Tuning
##  parameter 'splitrule' was held constant at a value of variance
##
## Tuning parameter 'min.node.size' was held constant at a value of 5
```

Next, we perform random forest on the train data set

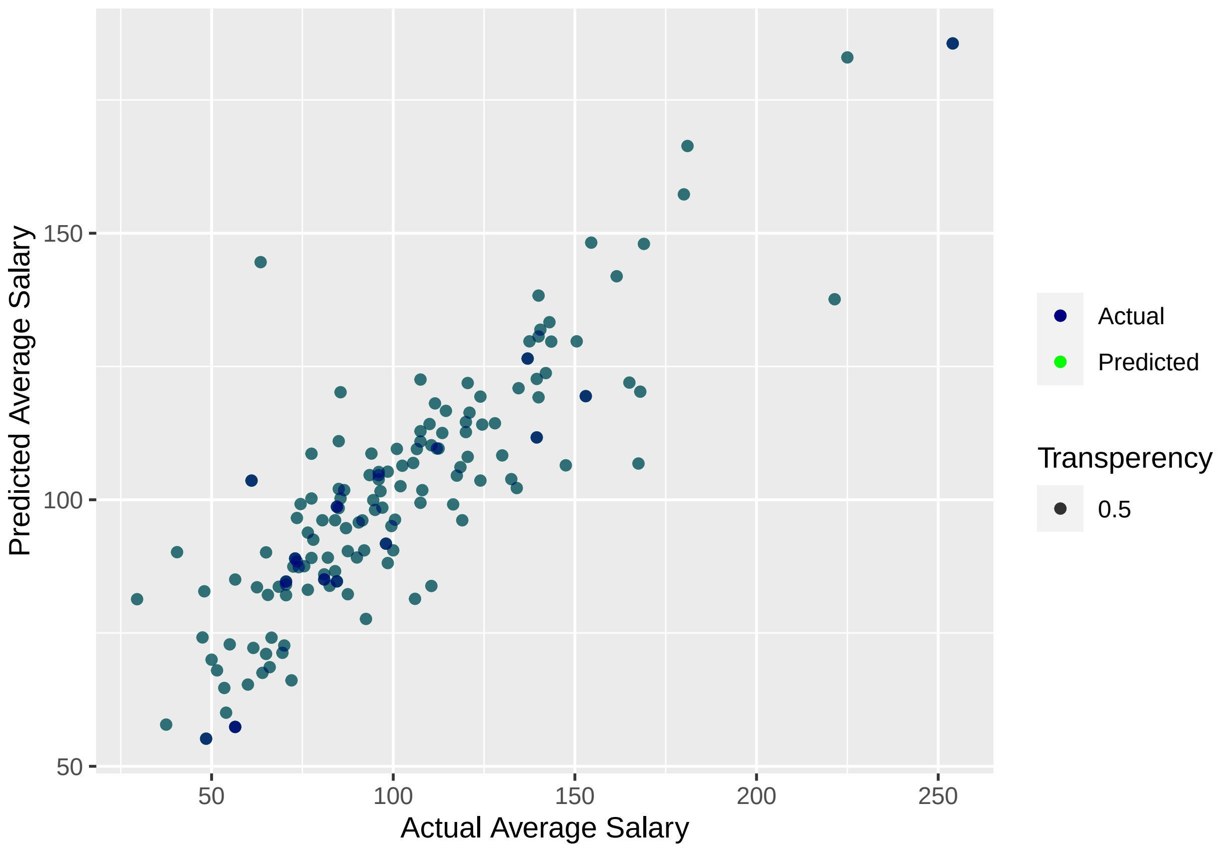
```r
#random forest regression
train_data_rf <- train_data %>%
  select(-c(Y_salary))
test_data_rf <- test_data %>%
  select(-c(Y_salary))

regr <- randomForest(x = train_data_rf,
                     y = train_data$Y_salary,
                     maxnodes = 100 ,
                     ntree = 1000)

predictions <- predict(regr, test_data_rf)

result <- test_data
result['predictions'] <- predictions

knitr::kable(head(result[, 1:6]),
             caption = "A sample of our prediction results.")
```

Table 5: A sample of our prediction results.

Y_salary	2020	401k	academic	accordance	accountable
73.5	0	0	0	0	0
85.0	0	0	0	0	0
47.5	1	0	0	0	0
96.0	0	0	0	0	0
121.0	0	0	0	0	0
106.0	0	0	0	0	0

The graph below shows the data between Actual Average Salary vs. Predicted Average Salary

```r
#plot graph to compare actual vs. predicted
ggplot(  ) +
  geom_point( aes(x = test_data$Y_salary, y = predictions, color = 'red', alpha = 0.5) ) +
  geom_point( aes(x = test_data$Y_salary, y = predictions, color = 'blue',  alpha = 0.5)) +
  labs(x = "Actual Average Salary", y = "Predicted Average Salary", color = "", alpha = 'Transperency')
  scale_color_manual(labels = c( "Actual", "Predicted"), values = c("navy", "green"))
```



```
#using metrics to calculate RMSE
print(paste0('MAE: ' , mae(predictions , test_data$Y_salary) ))
```

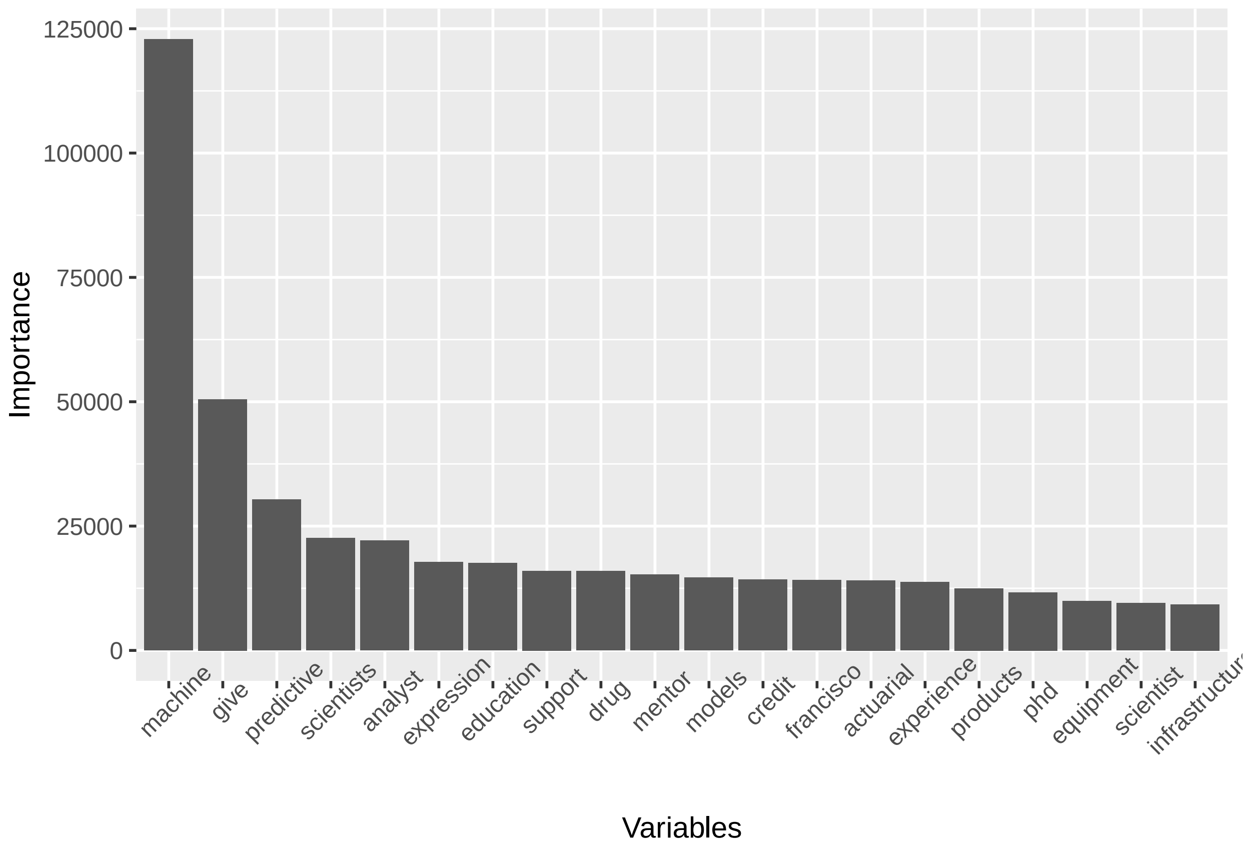
```
## [1] "MAE: 15.4145895594765"
```

```
print(paste0('RMSE: ' ,caret::postResample(predictions , test_data$Y_salary)['RMSE'])) #21.44
```

```
## [1] "RMSE: 21.6769392180064"
```

The bar graph below shows the top 20 most important variables in this prediction. Based on the bar graph, the top 5 most important variables are “machine”, “give”, “expression”, “predictive”, and “groups”.

```
#top 20 most important predictors
var_importance = mod_rf_tune$finalModel$variable.importance %>%
  sort(decreasing = TRUE) %>% head(20)

data.frame(variable = names(var_importance), importance = var_importance) %>%
  ggplot(aes(x = reorder(variable, -importance), y = importance)) + geom_col() +
  xlab("Variables") + ylab("Importance") + theme(axis.text.x = element_text(angle = 45))
```


Importance
125000
100000
75000
50000
25000
0
machine
give
predictive
scientists
analyst
expression
education
support
drug
mentor
models
credit
francisco
actuarial
experience
products
phd
equipment
scientist
infrastructure
Variables

3.2.3 XGBoost

Like random forests, gradient boosting machines does classification based on decision trees. However, while random forest builds an ensemble of deep (i.e complex) trees that are independent of one another, gradient boosting machines build shallow trees sequentially, where each tree learns and improves from the previous tree. This means that one would start with a weak model and sequentially boost its performance by allowing each new tree to focus on training data where the previous tree had the largest errors in prediction (or residuals). This is done by fitting each tree in the sequence according to the residuals of the previous tree.

Moreover, it computes the second-order gradients, i.e. second partial derivatives of the loss function, which provides more information about the direction of gradients and how to get to the minimum of our loss function while gradient boost uses the loss function of simple decision tree model as a proxy to minimize the error of the overall model. In addition, it uses advanced regularization (L1 and L2), which improves model generalization.

Each weight in all the trees would be multiplied by the learning rate in an XGBoost model, such that

$$w_j = \text{Learning Rate} \times \frac{\sum_{i \in I_j} \frac{\partial Loss}{\partial(\hat{y}=0)}}{\sum_{i \in I_j} \frac{\partial^2 Loss}{\partial(\hat{y}=0)^2} + \lambda}$$

where I_j is a set containing all the instances ((x, y) data points) at a leaf, and wj is the weight at leaf j with regularization from the λ constant.

We create our XGBoost model from the caret library, using hyperparameters as shown below:

```
set.seed(100)

trControl <- trainControl(
    method = 'cv',
    number = 5,
    verboseIter = TRUE,
    allowParallel = TRUE)

xgbGrid <- expand.grid(
  nrounds = 650,
  eta = 0.21,
  max_depth = 3,
  gamma = 0.04,
  colsample_bytree = 1,
  min_child_weight = 1,
  subsample = 1
)

xgb_caret <- train(Y_salary ~.,
                   data = train_data,
                   method = "xgbTree",
                   trControl = trControl,
                   tuneGrid = xgbGrid)
```

```
## + Fold1: nrounds=650, eta=0.21, max_depth=3, gamma=0.04, colsample_bytree=1, min_child_weight=1, subs
## - Fold1: nrounds=650, eta=0.21, max_depth=3, gamma=0.04, colsample_bytree=1, min_child_weight=1, subs
## + Fold2: nrounds=650, eta=0.21, max_depth=3, gamma=0.04, colsample_bytree=1, min_child_weight=1, subs
## - Fold2: nrounds=650, eta=0.21, max_depth=3, gamma=0.04, colsample_bytree=1, min_child_weight=1, subs
## + Fold3: nrounds=650, eta=0.21, max_depth=3, gamma=0.04, colsample_bytree=1, min_child_weight=1, subs
## - Fold3: nrounds=650, eta=0.21, max_depth=3, gamma=0.04, colsample_bytree=1, min_child_weight=1, subs
## + Fold4: nrounds=650, eta=0.21, max_depth=3, gamma=0.04, colsample_bytree=1, min_child_weight=1, subs
## - Fold4: nrounds=650, eta=0.21, max_depth=3, gamma=0.04, colsample_bytree=1, min_child_weight=1, subs
## + Fold5: nrounds=650, eta=0.21, max_depth=3, gamma=0.04, colsample_bytree=1, min_child_weight=1, subs
## - Fold5: nrounds=650, eta=0.21, max_depth=3, gamma=0.04, colsample_bytree=1, min_child_weight=1, subs
## Aggregating results
## Fitting final model on full training set
```

```
xgb_caret
```

```
## eXtreme Gradient Boosting
##
## 585 samples
## 354 predictors
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 467, 469, 469, 468, 467
## Resampling results:
##
##   RMSE      Rsquared  MAE
##   20.88346  0.678471  11.60141
##
## Tuning parameter 'nrounds' was held constant at a value of 650
## Tuning
##  held constant at a value of 1
## Tuning parameter 'subsample' was held
##  constant at a value of 1
```

As shown above, we have used these hyperparameters:

1. *gamma*: Pseudo-regularisation hyperparameter that controls the complexity of each tree.
2. *nrounds*: Number of decision trees in the final model
3. *eta*: Learning rate; determines the contribution of each tree on the final outcome and also how quickly the algorithm goes down the gradient descent.
4. *max_depth*: Depth of each tree
5. *min_child_weight*: Minimum number of observations in terminal nodes; controls complexity of the trees
6. *colsample_bytree*: subsample of columns used for each tree (repeated for every tree)
7. *subsample*: subsampling ratio of training data for growing trees to prevent over-fitting

The hyperparameters were tuned using 5-fold cross validation and grid search to find the best model, and we arrived at the optimal values for the hyperparameters.

```
pred_y = predict(xgb_caret, test_data)

#measure prediction accuracy
caret::MAE(test_data$Y_salary, pred_y) #mae
```

```
## [1] 8.035509
```

```
caret::RMSE(test_data$Y_salary, pred_y) #rmse
```

```
## [1] 16.5914
```

Overall, XGBoost gave an RMSE value of 16.5914.

Below we extracted the 20 most important features (words) from the XGBoost model.

```
library(xgboost)
```

```
##
## Attaching package: 'xgboost'

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

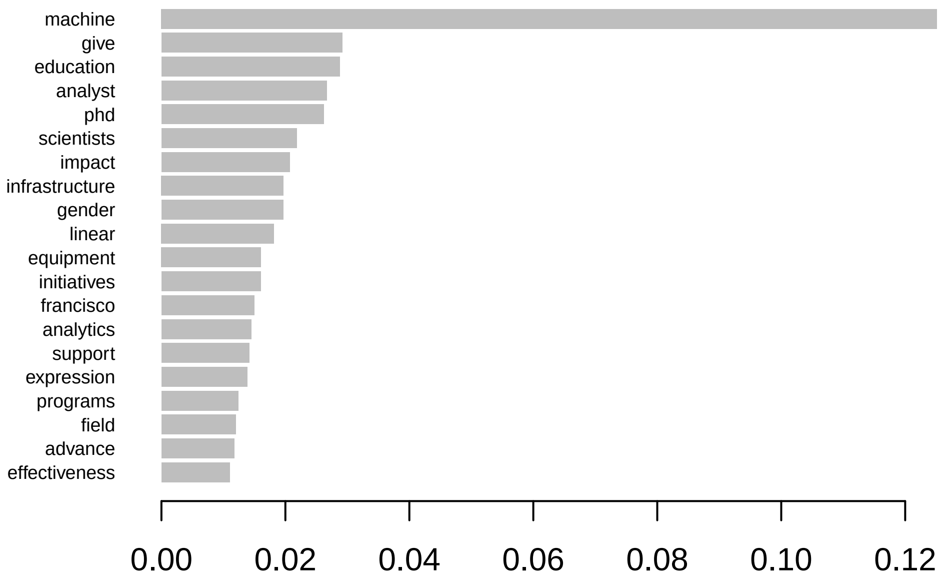
```
xgb_imp <- xgb.importance(feature_names = xgb_caret$finalModel$feature_names,
                          model = xgb_caret$finalModel)

knitr::kable(head(xgb_imp),
             caption = "Some of our 20 most important features.")
```

Table 6: Some of our 20 most important features.

Feature	Gain	Cover	Frequency
machine	0.1251407	0.0105748	0.0093227
give	0.0291924	0.0036677	0.0021936
education	0.0287696	0.0024835	0.0024678
analyst	0.0266963	0.0013853	0.0019194
phd	0.0261783	0.0075795	0.0054840
scientists	0.0218101	0.0076164	0.0049356

Plotting our top 20 most important variables:

```
xgb.plot.importance(xgb_imp[1:20])
```



3.2.4 Multivariate Adaptive Regression Splines (MARS)

We used multivariate adaptive regression splines (MARS) (Friedman 1991) model here, it is an approach that automatically generates a piecewise linear model that serves as an understandable stepping stone into non-linearity after learning the notion of multiple linear regression.

By evaluating cutpoints (knots) similar to step functions, MARS offers a practical method to capture the nonlinear relationships in the data. This method evaluates every data point for every predictor as a knot and builds a linear regression model using the candidate feature(s).

Consider non-linear, non-monotonic data where $Y = f(X)$. The MARS method will initially search for a single point within a range of X values where two distinct linear relationships between Y and X provide the lowest loss. The outcome is referred to as a hinge function $h(x-a)$, where a is the cutpoint value.

For example, if $a = 1$, our hinge function is $h(x-1)$ such that the linear models for y are:

$$y = \begin{cases} \beta_0 + \beta_1(1-x), & x < 1 \\ \beta_0 + \beta_1(x-1), & x > 1 \end{cases}$$

After the first knot is identified, the search for a second one begins, and it is discovered at $x = 2$. Now the linear models for y are:

$$y = \begin{cases} \beta_0 + \beta_1(1-x), & x < 1 \\ \beta_0 + \beta_1(x-1), & 1 < x < 2 \\ \beta_0 + \beta_1(2-x), & x > 2 \end{cases}$$

This process is repeated until several knots are identified, leading to the creation of a highly non-linear prediction equation. Even if using a lot of knots could help us fit a particularly excellent relationship to our training data, it might not perform well to unseen data. Once all of the knots have been found, we may systematically eliminate knots that do not significantly improve predictive accuracy. This is pruning process, and we may use cross-validation to determine the optimal number of knots.

We will use the following packages. First of all, we divided the dataset into training dataset and test dataset:

```r
set.seed(100)

ind_mars <- which(runif(nrow(D_dtm)) < 0.7)

train_mars <- D_dtm %>% dplyr::slice(ind_mars)
test_mars <- D_dtm %>% dplyr::slice(-ind_mars)

cat("Dimensions of the training dataset are", dim(train_mars), "\n")
```

```
## Dimensions of the training dataset are 512 1239
```

```r
cat("Dimensions of the test dataset are", dim(test_mars), "\n")
```

```
## Dimensions of the test dataset are 230 1239
```

MARS model have two hyperparameters: the maximum degree of interactions and the number of terms retained in the final model. To achieve the optimal combination of these tuning parameters, we must conduct a grid search that minimize the error of prediction.

Here, we built up a grid with 30 different combinations of interaction complexity (degree) and the number of terms to include in the final model (nprune).

```r
marsgrid <- expand.grid(
  degree = 1:3,
  nprune = seq(1, 100, length.out = 20) %>% floor()
)
```

We performed required grid search by using 10-fold cross-validation:

- Our chosen parameters.

```r
set.seed(100)

mars_cv <- train(Y_salary ~., data = train_mars,
  method = "earth",
  metric = "RMSE",
  trControl = trainControl(method = "cv", number = 10),
  tuneGrid = marsgrid
)
```

```
## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo, :
## There were missing values in resampled performance measures.
```

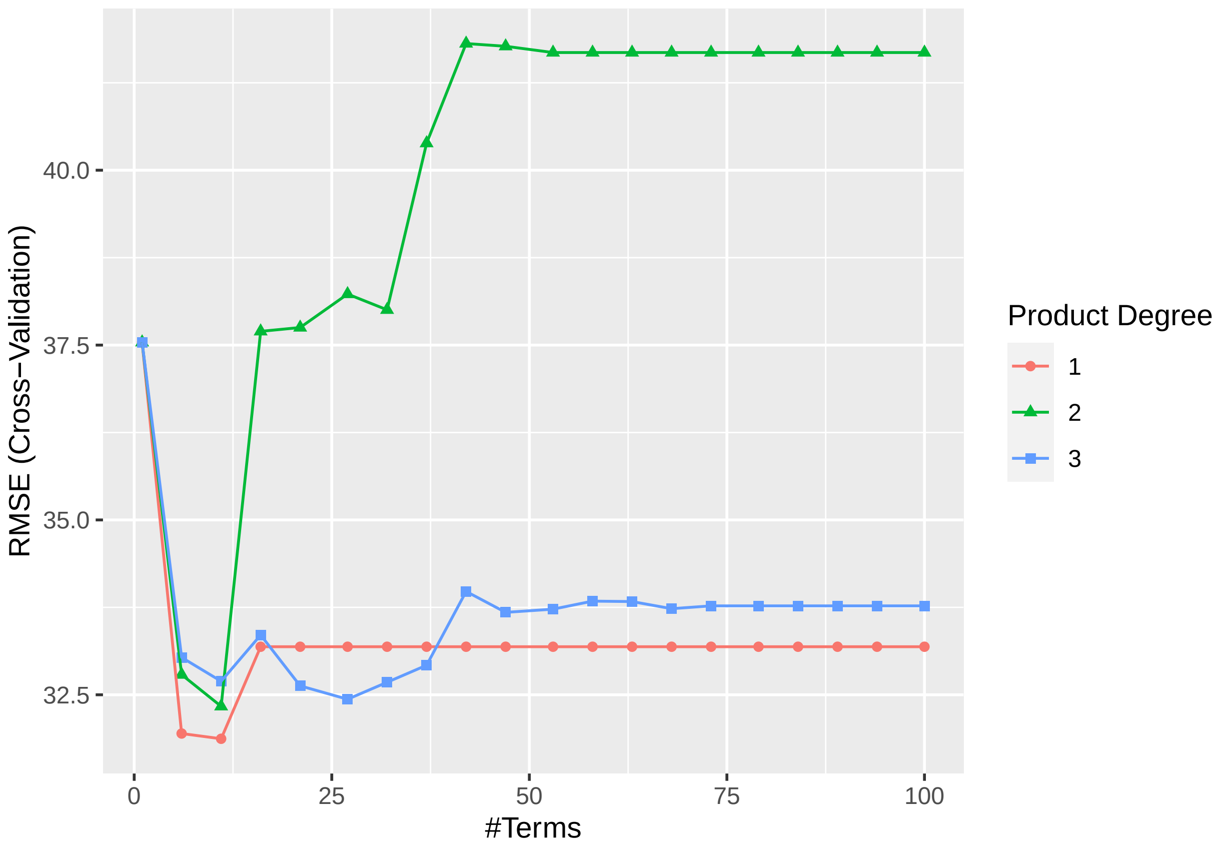
```r
knitr::kable(mars_cv$bestTune) #nprune = 11, degree = 1
```

	nprune	degree
3	11	1

```r
knitr::kable(mars_cv$results %>%
  filter(nprune == mars_cv$bestTune$nprune, degree == mars_cv$bestTune$degree))
```

degree	nprune	RMSE	Rsquared	MAE	RMSESD	RsquaredSD	MAESD
1	11	31.87122	0.3140381	25.10864	2.810016	0.0945591	2.344299

```r
ggplot(mars_cv)
```

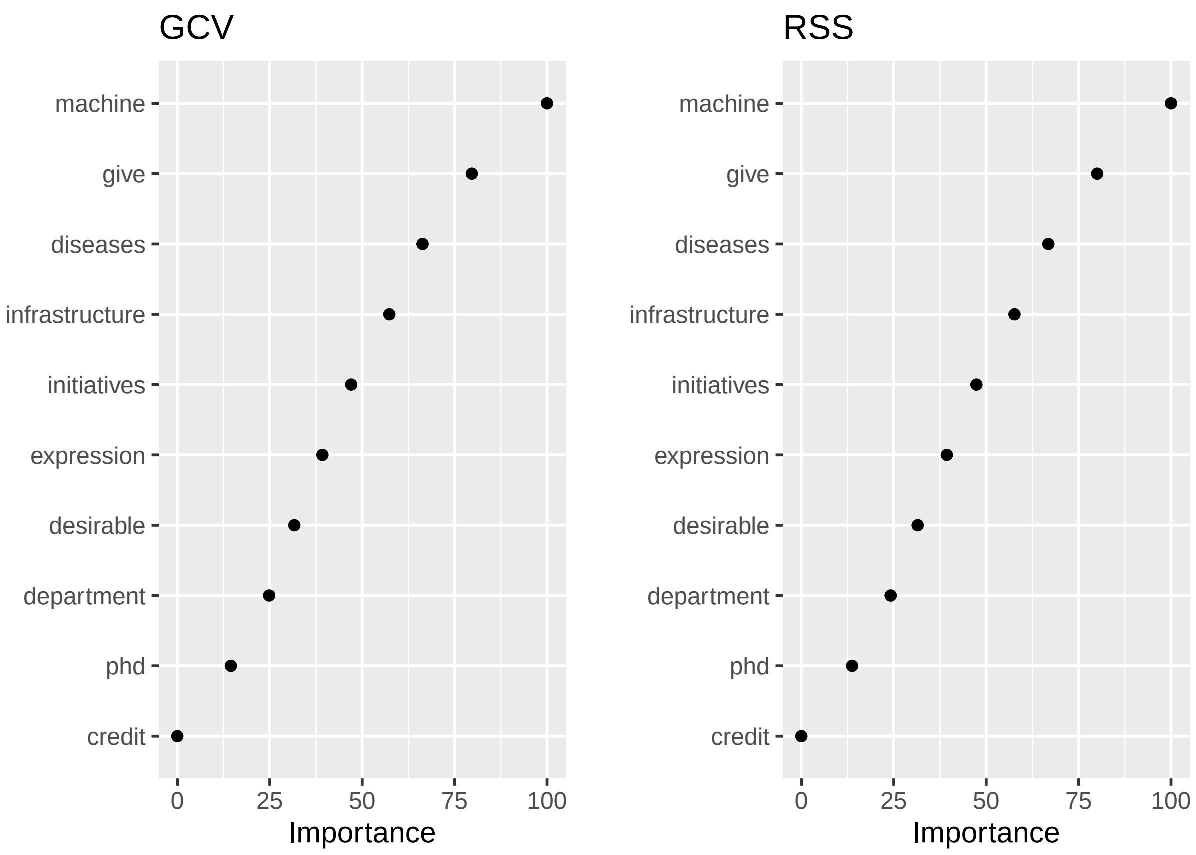


The backwards elimination feature selection process used in MARS models seeks for reductions in the generalized cross-validation (GCV) estimate of error when each additional predictor is introduced to the model. The variable importance is based on this overall reduction. MARS effectively accomplishes automated feature selection since it will automatically include and remove variables throughout the pruning phase.

After pruning, a predictor's significance value is 0 if it was never used in any of the MARS basis functions in the final model. There are only 11 features have importance values greater than 0, whereas the other features all have importance values of zero since they were excluded from the final model.

We also kept track of how the residual sums of squares (RSS) change when terms are added. However, we noticed that there is no much difference between these two measures.

```
plot_GCV <- vip(mars_cv, num_features = 20, geom = "point", value = "gcv") + ggtitle("GCV")
plot_RSS <- vip(mars_cv, num_features = 20, geom = "point", value = "rss") + ggtitle("RSS")

gridExtra::grid.arrange(plot_GCV, plot_RSS, ncol = 2)
```



We used the optimal hyperparameters to train the model and then calculated the RMSE:

```r
set.seed(100)

marsgrid_tuned <- expand.grid(
  degree = 1,
  nprune = 11
)

mars_cv_tuned <- train(Y_salary ~., data = train_data,
  method = "earth",
  metric = "RMSE",
  trControl = trainControl(method = "cv", number = 10),
  tuneGrid = marsgrid_tuned
)

pred_mars = predict(mars_cv_tuned, test_data)

print(caret::MAE(test_data$Y_salary, pred_mars)) #23.6061
```

```
## [1] 23.22016
```

```r
print(caret::RMSE(test_data$Y_salary, pred_mars)) #34.19323
```

```
## [1] 30.0025
```

3.2.5 Partial Least Squares (PLS)

Partial Least Squares (PLS) is a common technique to analyse relative importance when the data includes more predictors than observations. It is an useful dimension reduction method which is similar with principal component analysis (PCA).

We do a regression against the response variable inside the narrower space created by mapping the predictor variables to a smaller set of variables. The response variable is not taken into account during the dimension reduction process in PCA. PLS, on the other hand, seeks to select newly mapped factors that best describe the response variable.

Below are the required packages. We divided the dataset into training dataset and test dataset first:

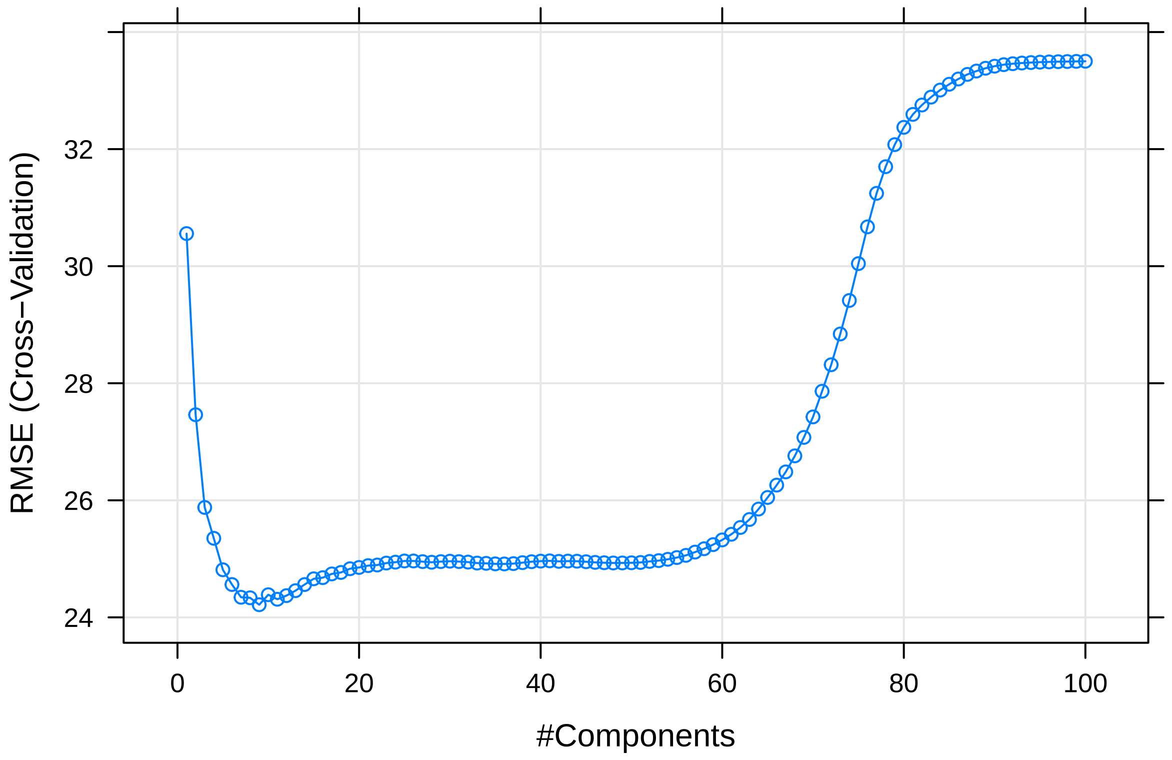
```
set.seed(100)
# 80/20 split
inTraining <- createDataPartition(D_dtm$Y_salary, p = .7, list = FALSE)
train_pls <- D_dtm[inTraining,]
test_pls  <- D_dtm[-inTraining,]
```

The hyperparameter for PLS model is the number of components used in the model (ncomp) .We conduct a grid search that minimize the prediction error to achieve the optimal hyperparameter. The grid search was conducted by 10-fold cross-validation:

```
set.seed(100)

plsGrid <- expand.grid(
  ncomp   = seq(1, 100, by = 1)
)

pls_cv <- train(
  Y_salary ~ .,
  data = train_pls,
  method = 'pls',
  metric = "RMSE",
  preProcess = c("center", "scale"),
  trControl = trainControl(method = "cv", number = 10),
  tuneGrid = plsGrid
)

pls_cv
```

```
## Partial Least Squares
##
##  522 samples
## 1238 predictors
##
## Pre-processing: centered (1238), scaled (1238)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 470, 470, 470, 470, 470, 470, ...
## Resampling results across tuning parameters:
##
##   ncomp  RMSE      Rsquared   MAE
##     1    30.55685  0.3561268  23.85095
##     2    27.46180  0.4790637  20.92784
##     3    25.87809  0.5397913  18.80301
##     4    25.35099  0.5627683  18.25695
##     5    24.81328  0.5881916  17.57429
##     6    24.56162  0.5937114  16.77198
##     7    24.34335  0.6048332  16.25550
##     8    24.33422  0.6083873  15.75751
##     9    24.21590  0.6134054  15.33192
##    10    24.38686  0.6123953  15.14165
##    11    24.30930  0.6153090  14.84166
##    12    24.37149  0.6153701  14.75219
##    13    24.45631  0.6139073  14.63586
##    14    24.56083  0.6128587  14.53326
##    15    24.65877  0.6111942  14.46135
##    16    24.67875  0.6119272  14.37184
##    17    24.74250  0.6110657  14.32525
##    18    24.76746  0.6107837  14.31129
##    19    24.83094  0.6098625  14.28116
##    20    24.85262  0.6102663  14.24766
##    21    24.88418  0.6101602  14.22263
##    22    24.89411  0.6104530  14.20213
##    23    24.92743  0.6101396  14.18672
##    24    24.94306  0.6101353  14.18171
##    25    24.96360  0.6098228  14.16983
##    26    24.96351  0.6098946  14.15406
##    27    24.95206  0.6102344  14.13375
##    28    24.94173  0.6105859  14.11613
##    29    24.95276  0.6103578  14.12949
##    30    24.95922  0.6104932  14.13336
##    31    24.95314  0.6108676  14.12952
##    32    24.94442  0.6111849  14.12934
##    33    24.92755  0.6116710  14.13662
##    34    24.92303  0.6119636  14.14047
##    35    24.91448  0.6122413  14.14332
##    36    24.91293  0.6123034  14.15126
##    37    24.91831  0.6124318  14.16217
##    38    24.93497  0.6120947  14.16408
##    39    24.95060  0.6118200  14.14368
##    40    24.96103  0.6116358  14.14237
##    41    24.96535  0.6115435  14.14120
##    42    24.95686  0.6117652  14.12223
##    43    24.96132  0.6116145  14.11387
##    44    24.95934  0.6116392  14.10458
##    45    24.95039  0.6117579  14.09207
##    46    24.93932  0.6119917  14.08525
##    47    24.93333  0.6120841  14.08155
##    48    24.92913  0.6121838  14.07734
##    49    24.92962  0.6121315  14.07476
##    50    24.93263  0.6120156  14.08027
##    51    24.93780  0.6119590  14.08329
##    52    24.95569  0.6115354  14.08895
##    53    24.96948  0.6113024  14.09910
##    54    24.99160  0.6107836  14.10741
##    55    25.02186  0.6102651  14.12571
##    56    25.05910  0.6096706  14.16597
##    57    25.11512  0.6086555  14.21128
##    58    25.17222  0.6076005  14.24717
##    59    25.24292  0.6063190  14.30495
##    60    25.32304  0.6049449  14.35376
##    61    25.41967  0.6031783  14.42555
##    62    25.53586  0.6011149  14.51264
##    63    25.67264  0.5989126  14.62360
##    64    25.85027  0.5958021  14.75393
##    65    26.04849  0.5924079  14.87680
##    66    26.25961  0.5891112  15.00149
##    67    26.48520  0.5854223  15.11939
##    68    26.75942  0.5810219  15.29575
##    69    27.07495  0.5757469  15.49269
##    70    27.42511  0.5699539  15.71880
##    71    27.86316  0.5626824  15.97971
##    72    28.31573  0.5555383  16.24821
##    73    28.84147  0.5471291  16.54778
##    74    29.41420  0.5379223  16.86831
##    75    30.04348  0.5279954  17.20660
##    76    30.67214  0.5183705  17.56716
##    77    31.24464  0.5097036  17.91835
##    78    31.69984  0.5031265  18.19972
##    79    32.07666  0.4978152  18.43163
##    80    32.37185  0.4936442  18.60105
##    81    32.59291  0.4906123  18.72859
##    82    32.75350  0.4886184  18.81300
##    83    32.88793  0.4870460  18.88022
##    84    33.00868  0.4855643  18.93815
##    85    33.10833  0.4844639  18.99277
##    86    33.19829  0.4833427  19.03446
##    87    33.27666  0.4824162  19.07229
##    88    33.33412  0.4817344  19.09775
##    89    33.38200  0.4811172  19.11904
##    90    33.41696  0.4806746  19.13089
##    91    33.44293  0.4803971  19.13767
##    92    33.45951  0.4802222  19.14236
##    93    33.47079  0.4800991  19.14126
##    94    33.47893  0.4799860  19.14255
##    95    33.48559  0.4798903  19.14386
##    96    33.49032  0.4798213  19.14601
##    97    33.49402  0.4797704  19.14704
##    98    33.49643  0.4797355  19.14803
##    99    33.49889  0.4797001  19.14895
##   100    33.50150  0.4796691  19.14942
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was ncomp = 9.
```

```
plot(pls_cv)
```



We used the optimal hyperparameter to train the model and calculated the RMSE as well:

```
set.seed(100)

plsGrid_tuned <- expand.grid(
  ncomp = 9
)

pls_cv_tuned <- train(Y_salary ~., data = train_pls,
  method = "pls",
  metric = "RMSE",
  trControl = trainControl(method = "cv", number = 10),
  tuneGrid = plsGrid_tuned
)

pred_pls = predict(pls_cv_tuned, test_pls)

print(caret::MAE(test_pls$Y_salary, pred_pls)) #16.63214
```

```
## [1] 16.63214
```

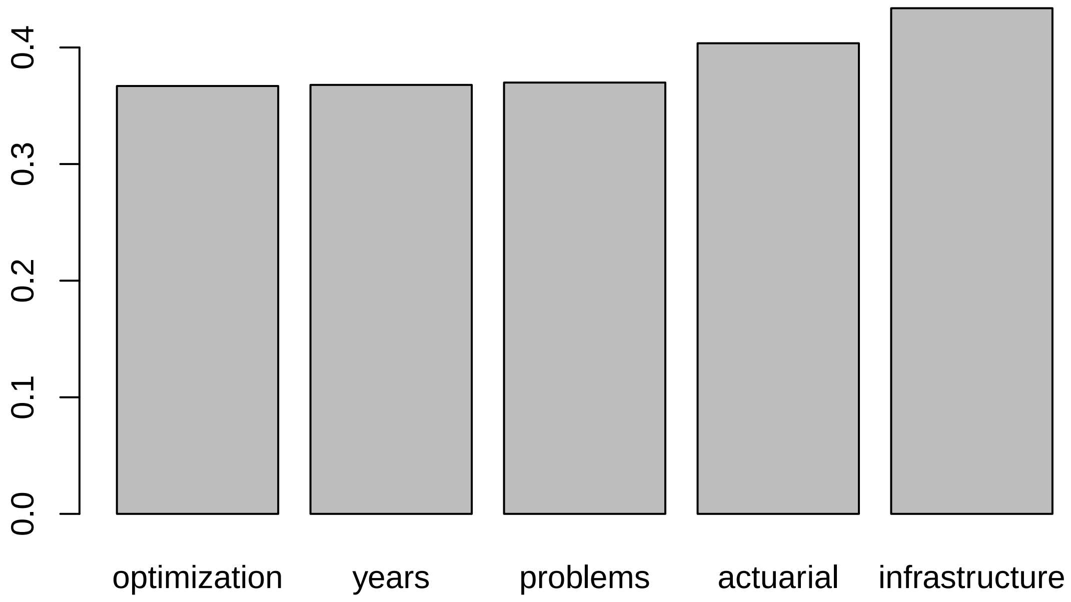
```
print(caret::RMSE(test_pls$Y_salary, pred_pls)) #23.87552
```

```
## [1] 23.87552
```

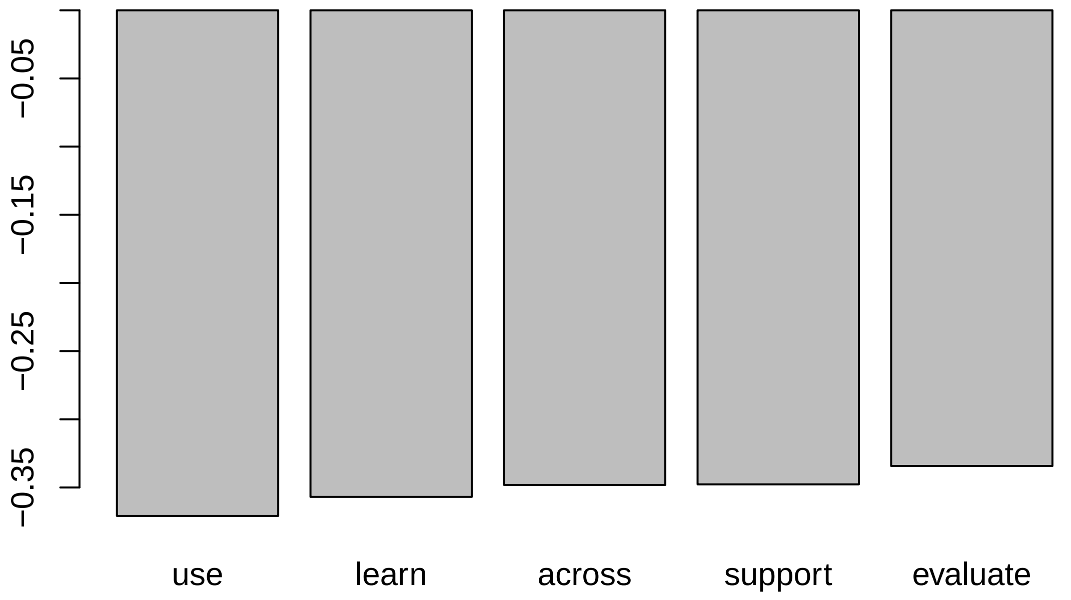
The barplots below show that ‘credit’, ‘lead’, ‘mission’, ‘actuarial’ and ‘scientists’ are positive predictors, while ‘support’, ‘use’, ‘project’, ‘solutions’ and ‘food’ are negative predictors:

```
coefficients = coef(pls_cv_tuned$finalModel)
sum.coef = sum(sapply(coefficients, abs))
coefficients = coefficients * 100 / sum.coef
coefficients = sort(coefficients[, 1 , 1])
```

```
barplot(tail(coefficients, 5))
```



```
barplot(head(coefficients, 5))
```



3.3 Summary of results

```r
final_results <- data.frame(Model = c('MLR', 'Random Forest', 'XGBoost', 'MARS', 'PLS'),
                            RMSE = c(16.20561, 21.6223370884116, 16.5914, 30.0025, 23.87552))
knitr::kable(final_results,
             caption = "Accuracy of models.")
```

Table 9: Accuracy of models.

Model	RMSE
MLR	16.20561
Random Forest	21.62234
XGBoost	16.59140
MARS	30.00250
PLS	23.87552

4 Conclusion

Recommendation engines are a commonly found solution applied to job search portals, such as the Singapore government's national jobs portal, MyCareersFuture. However, more can be done to bridge the gap between job seekers and their desired careers. In this project, we have produced models that predict the expected average salary earned given a job description. Our best performing model, Multiple Linear Regression, can be used to help workers set expectations of salary based on keywords they value as important in search of a job. This will help job seekers focus on searching for their desired job role rather than focus on maximizing salary earned, which will hopefully increase job satisfaction. The model also identifies key terms such as "eligible", "mongodb", "upon", "francisco", and "hypotheses". Some of these keywords may not seem to make sense, and understanding the importance of these words is unclear. Some of these words, on the other hand, give insight into areas that job seekers can focus on, be it upskilling (for example, learning MongoDB), or narrowing their job search to sectors such as actuarial science or molecular chemistry in order to maximize their potential salary.

Based on RMSE, our best model uses Multiple Linear Regression, and it has identified key terms that have a significant impact on data science salary. However, Multiple Linear Regression does not identify the same important features (words) as our other models. Further research is required to better understand the difference between models and why they identify vastly different features as important.

5 References

1. Alexander, L. (2019). The importance of keywords in job ads. SEEK. Retrieved November 16, 2022, from https://www.seek.com.au/employer/hiring-advice/the-importance-of-keywords-in-job-ads
2. Arora, P. (2022, August 29). What's keeping Singapore employees unhappy at work? - ETHRWorldSEA. HR News Southeast Asia. Retrieved November 7, 2022, from https://hrsea.economictimes.indiatimes.com/news/employee-experience/whats-keeping-singapore-employees-unhappy-at-work/93833788
3. Chong, C. (2022, May 17). Nearly 1 in 3 workers in S'pore plans to change employers in first half of 2022: Survey. The Straits Times. Retrieved November 4, 2022, from https://www.straitstimes.com/singapore/jobs/nearly-1-in-3-workers-in-spore-plan-to-change-employers-in-first-half-of-2022-survey
4. My Skills Future. (2021, June 21). 5 Crucial Skills You Need to Remain Employable in the Wake of Covid-19 | Myskillsfuture.gov.sg. MySkillsFuture. Retrieved October 14, 2022, from https://www.myskillsfuture.gov.sg/content/portal/en/career-resources/career-resources/education-career-personal-development/5-crucial-skills-you-need-to-remain-employable-during-covid.html
5. The Straits Times. (2021, December 22). It's a match: How skills-based hiring fits in the future of work. The Straits Times. Retrieved October 14, 2022, from https://www.straitstimes.com/singapore/jobs/its-a-match-how-skills-based-hiring-fits-in-the-future-of-work
6. Tan, E. (2021, April 14). S'pore employers prioritise skills over education, experience: LinkedIn survey. The Straits Times. Retrieved October 14, 2022, from https://www.straitstimes.com/singapore/jobs/singapore-employers-prioritise-skills-over-education-experience-linkedin-survey