Self Reflection

Roshan Shrestha

2023-02-27

Introduction

My growth in this course has always been progressive. I enjoyed the Modeling and Regression course as it helped me refreshed my knowledge and concept with the statistical concepts. Through this course, I believe I have built a strong foundation with statistical analysis and statistical modeling. Furthermore, I got the opportunity to implement the models and concept in R. This is my fourth semester using R and my coding journey with R has been progressively improved as well and I feel I'm pretty much proficient in using R now.

I started with doing the import of data, pre-processing my dataset and running some exploratory data analysis for the dataset in R. I have also performed some exploratory analysis over here as many of my classmate also wanted to see which skills were frequently preferred, optimal days of job postings, the top skill set for the data analysis role, and the locations of the job posting. I have also included my visualization in my final project so that everyone can benefit from the analysis I have made on the job postings.

```r
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
jobs <- read.csv("gsearch_jobs1.csv", row.names = 1)
```

I further performed some pre-processing on my datset to get the desired datset to run my model.

```r
# replace string values in location column
jobs$location <- gsub("\\+.*", "", jobs$location)
```

```r
library(qdap)
```

```
## Loading required package: qdapDictionaries

## Loading required package: qdapRegex
```

```
##
## Attaching package: 'qdapRegex'

## The following object is masked from 'package:dplyr':
##
##     explain

## Loading required package: qdapTools

##
## Attaching package: 'qdapTools'

## The following object is masked from 'package:dplyr':
##
##     id

## Loading required package: RColorBrewer

##
## Attaching package: 'qdap'

## The following objects are masked from 'package:base':
##
##     Filter, proportions
```

```r
skill_frequecny <- freq_terms(jobs$description_tokens)
```

```r
skill_frequecny
```

```
##    WORD            FREQ
## 1  'sql'           6217
## 2  'excel'         4250
## 3  'tableau'       3404
## 4  'python'        3319
## 5  'powerbi'       3186
## 6  'r'             2209
## 7  'sas'           1233
## 8  'powerpoint'     946
## 9  'word'           917
## 10 'snowflake'      845
## 11 'aws'            734
## 12 'azure'          674
## 13 'spss'           520
## 14 'jira'           499
## 15 'looker'         484
## 16 'go'             477
## 17 'microstrategy'  457
## 18 'spark'          449
## 19 'c'              437
## 20 'spreadsheet'    413
```

```r
library(plotly)
```

```
## Loading required package: ggplot2

##
## Attaching package: 'ggplot2'

## The following object is masked from 'package:qdapRegex':
##
##     %+%

##
## Attaching package: 'plotly'

## The following object is masked from 'package:ggplot2':
##
##     last_plot

## The following object is masked from 'package:stats':
##
##     filter

## The following object is masked from 'package:graphics':
##
##     layout
```

```r
analysis_skills <- data.frame("Categorie" = rownames(skill_frequecny), skill_frequecny)
data <- analysis_skills[, c('WORD', 'FREQ')]

colors <- c('rgb(211,94,96)', 'rgb(128,133,133)', 'rgb(144,103,167)', 'rgb(171,104,87)', 'rgb(114,147,2

fig <- plot_ly(data, labels = ~WORD, values = ~FREQ, type = 'pie',
        textposition = 'inside',
        textinfo = 'label+percent',
        insidetextfont = list(color = '#FFFFFF'),
        hoverinfo = 'text',
        text = ~paste(FREQ),
        marker = list(colors = colors,
                      line = list(color = '#FFFFFF', width = 1)),
                      #The 'pull' attribute can also be used to create space between the sectors
        showlegend = FALSE)
fig <- fig %>% layout(title = 'Pie Chart of on demand Data Analysis skills',
         xaxis = list(showgrid = FALSE, zeroline = FALSE, showticklabels = FALSE),
         yaxis = list(showgrid = FALSE, zeroline = FALSE, showticklabels = FALSE))

fig
```

```
## PhantomJS not found. You can install it with webshot::install_phantomjs(). If it is installed, pleas
```

```r
library(plotly)
set.seed(1)

Skills <- data.frame("Categorie" = rownames(skill_frequecny), skill_frequecny)
data <- Skills[, c('WORD', 'FREQ')]

# Generate 20 random colors
colors <- paste0("rgb(", round(runif(20, 0, 255)), ",",
                 round(runif(20, 0, 255)), ",",
                 round(runif(20, 0, 255)), ")")

fig <- plot_ly(data, x = ~WORD, y = ~FREQ, type = 'bar',
               marker = list(color = colors))

fig <- fig %>% layout(title = 'Top 20 Skills set in Data Analysis Jobs',
                      xaxis = list(title = 'Skills '),
                      yaxis = list(title = 'Frequency'))

fig
```

```r
job_python <- read.csv("job_python.csv", row.names = 1)
cleanjobs_salary <- filter(job_python, job_python$salary_standardized != 'NA')
```

```r
library(ggplot2)
library(plotly)
library(lubridate)
```

```
##
## Attaching package: 'lubridate'

## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

```r
# Convert date column to date object
job_python$date_time <- ymd_hms(job_python$date_time)

# Create a new column with the day of the week
job_python$day <- weekdays(job_python$date_time)


# Create a new column for day of the week using base R weekdays() function
job_python$day <- weekdays(job_python$date_time)

# Aggregate job counts by day of the week
job_counts <- aggregate(title ~ day, data = job_python, FUN = length)

# Create the plot using ggplot2
job_counts_plot <- ggplot(data = job_counts, aes(x = day, y = title, group = 1)) +
  geom_line(color = "steelblue", size = 1.2) +
  labs(title = "Job Posts by Day of the Week", x = "Day of the Week", y = "Number of Job Posts") +
  scale_x_discrete(limits = c("Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturda
  theme_minimal()
```

```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use `linewidth` instead.
```

```r
# Create an interactive plot using plotly
ggplotly(job_counts_plot)
```

```r
library(tidyr)
Jobs_lon_lat <- read.csv("Jobs_lon_lat.csv", row.names = 1)
# Create a new data frame with the separated columns
Jobs_lon_lat_new <- Jobs_lon_lat %>%
  separate(location, into = c("city", "state"), sep = ", ")
```

```
## Warning: Expected 2 pieces. Missing pieces filled with `NA` in 128 rows [3, 59,
## 63, 167, 188, 190, 195, 228, 314, 380, 433, 482, 483, 484, 485, 507, 508, 608,
## 617, 618, ...].
```

```r
library(dplyr)

# group by location and count frequency
Jobs_lon_lat_freq <- Jobs_lon_lat %>%
  group_by(location) %>%
  summarise(freq = n())

# join the frequency count to Jobs_lon_lat
Jobs_lon_lat_with_freq <- left_join(Jobs_lon_lat, Jobs_lon_lat_freq, by = "location")
```

```r
library(ggplot2)
library(maps)
library(plotly)

# Filter Jobs_lon_lat dataframe to only include locations within the USA
Jobs_lon_lat_USA <- Jobs_lon_lat_with_freq[Jobs_lon_lat_with_freq$longitude > -125 &

# Create a US map using ggplot2 and maps
USA_map <- map_data("state")
p <- ggplot() +
  geom_polygon(data = USA_map, aes(x = long, y = lat, group = group), fill = "white", color = "black")
  geom_point(data = Jobs_lon_lat_USA, aes(x = longitude, y = latitude, label = location, text = paste("
  coord_map() +
  labs(title = "Job Postings by Location in USA")
```

```
## Warning in geom_point(data = Jobs_lon_lat_USA, aes(x = longitude, y =
## latitude, : Ignoring unknown aesthetics: label and text
```

```r
ggplotly(p)
```

```r
# Aggregate job counts by location and sort in descending order
job_counts <- Jobs_lon_lat_USA %>%
  group_by(location) %>%
  summarize(count = n()) %>%
  arrange(desc(count)) %>%
```

```
  top_n(30, count)

# Create the plot using ggplot2
job_counts_plot <- ggplot(data = job_counts, aes(x = location, y = count)) +
  geom_bar(stat = "identity", fill = "steelblue") +
  labs(title = "Job Posts by Location", x = "Location", y = "Number of Job Posts") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust = 1))

# Create an interactive plot using plotly
ggplotly(job_counts_plot)
```

Removing the [] from description_tokens

```
clean_salary_desp <- filter(cleanjobs_salary, cleanjobs_salary$description_tokens != '[]')
```

```
library(qdap)
skill_frequecny <- freq_terms(cleanjobs_salary$description_tokens)
```

How I met the Course Objectives

The course objectives were fairly simple and straightforward where we had to mainly work with the statistical modeling and the statistical analysis of the data and how each statistical measure helped us in identifying the best model. Conducting and selecting the best model, I guess was one of the toughest part where we have to understand and look at each variable but I believe that integral part has built a strong foundation on me to determine and apply appropriate models in specific data context.

Objective 1: Determine and apply the appropriate generalized linear model for a specific data context

After completing the exploratory data analysis, I worked on creating a selected dataset for my model. The below code shows how I created it for my model.

```
library(dplyr)
library(caret)
```

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

```
pred_data <- clean_salary_desp[, c("salary_standardized", "airflow" ,"alteryx" ,"apl" , "asp.net","asse
 ,"bigquery"           ,"bitbucket"          ,"c"                 ,   "c.."                    , "c.c.."
 ,"cobol"              ,"cognos"             ,"crystal"          ,    "css"                    , "dart"
 ,"datarobot"          ,"dax"                ,"docker"            ,   "dplyr"                  , "excel"
 ,"fortran"            ,"gcp"                ,"gdpr"               ,  "ggplot2"                , "git"
  ,"github"            ,  "gitlab"            ,"go"                 ,  "golang"                 ,"graphql
 ,"groovy"             ,"hadoop"             ,"html"               , "java"                   , "javascri
 ,"jira"               ,"jquery"             ,"js"                 , "julia"                  , "jupyter"
 ,"keras"              ,"linux"              ,"linux.unix"         , "looker"                 , "matlab"
  ,"matplotlib"        , "microstrategy"     ,  "mongo"              , "mongodb"                ,  "mssql"
```

```
  ,"mxnet"                ,"mysql"              , "nltk"              , "no.sql"            ,  "node"
  ,"node.js"              ,"nosql"              , "nuix"              , "numpy"              , "outlook
  ,"pandas"               ,"perl"                ,"php"               , "pl.sql"              ,"plotly"
 ,"postgres"             ,"postgresql"          ,"power_bi"          , "powerpoint"         ,"powerpoi
 ,"powershell"           ,"pyspark"            , "python"            , "pytorch"           , "qlik"
, "r"                    ,"redis"              , "redshift"          , "rshiny"            , "ruby"
 ,"rust"                 ,"sap"                , "sas"               , "scala"              ,"scikit.l
 ,"seaborn"              ,"selenium"            ,"sharepoint"        , "shell"             , "snowflak
 ,"solidity"             ,"spark"            ,  "splunk"             , "spreadsheet"       , "spss"
 ,"sql"                  ,"ssis"               , "ssrs"              , "swift"             , "t.sql"
 ,"tableau"              ,"tensorflow"         , "terminal"           ,"tidyr"             , "twilio"
 ,"typescript"           ,"unix"               , "unix.linux"         ,"vb.net"            , "vba"
 ,"visio"                ,"visual_basic"       , "vue"               , "vue.js"            , "word"  ))
```

As I further did some cleaning of my data because it had little information which wouldn't be useful in predicting a model. So, the below code calculates the column sums and it selects the column of pred_data except those with a sum of 0. Hence, this code removes any columns from pred_data that have sum less than 4.

```r
# calculate the column sums and find the indices of the columns with a sum of 0
zero_cols <- which(colSums(pred_data) < 4)

# select all columns except those with a sum of 0
pred_data <- pred_data[, -zero_cols]
```

Logistic Regression

Logistic Regression is used to fit the sigmoid function to the data which estimates the probability of particular observation belonging to positive class which is salary_binary. Similarly, the logistic regression is trained on the training data and is used to predict the binary outcome variable for the test data.

With logistic regression, you can predict the probability of an individual having a certain skill or not, based on the other skills they possess. This can be useful for tasks such as predicting the likelihood of a candidate being a good fit for a job based on their skillset, or identifying which skills are most important for a particular role.

I have implemented the below code for the logistic regression and accuracy is calculated to see the performance of the model.

```r
pred_data1 <- pred_data

# Calculate the mean salary
mean_salary <- mean(pred_data1$salary_standardized)

# Set salary to 0 or 1 based on the mean
pred_data1$salary_binary <- ifelse(pred_data1$salary_standardized < mean_salary, 0,
                                   ifelse(pred_data1$salary_standardized >= mean_salary, 1, NA))

# Split data into train and test sets
train_index <- sample(nrow(pred_data1), 0.7 * nrow(pred_data1))
train_data <- pred_data1[train_index, ]
test_data <- pred_data1[-train_index, ]

# Build logistic regression model
log_model <- glm(salary_binary ~ ., data = train_data, family = "binomial")
```

```
## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```r
# Make predictions on the test set
predictions <- predict(log_model, newdata = test_data, type = "response")

# Calculate accuracy
threshold <- 0.5
predicted_classes <- ifelse(predictions > threshold, 1, 0)
accuracy <- mean(predicted_classes == test_data$salary_binary)

# Print accuracy
print(paste0("Accuracy: ", round(accuracy, 2)))
```

```
## [1] "Accuracy: 0.99"
```

Linear Regression

I have also performed linear regression in my final project where the linear regression is used to model the relationship between the salary_standardized variable and the other variables. It uses the `lm` function to fit the linear regression model by finding the best-fit line that minimizes the sum of the squared difference between the observed values of the dependent variable and the predicted value of the model.

This code divides the pred_data dataset into a training set and a test set. It then uses the glm() function to create a logistic regression model and the predict() function to make predictions on the test set. The predict() function gives the projected odds of each person having a salary above or below the threshold when the type = "response" argument is used.

```r
# Split data into training and testing sets
set.seed(123)
trainIndex <- createDataPartition(pred_data$salary_standardized, p = .8, list = FALSE, times = 1)
trainData <- pred_data[trainIndex, ]
testData <- pred_data[-trainIndex, ]

# Create linear model
linear_model <- lm(salary_standardized ~ ., data = trainData)

# Predict on test data
predictions <- predict(linear_model, newdata = testData)
```

```r
# Load required libraries
library(dplyr)
library(tidyr)
library(broom)
library(stats)

# Fit linear model
linear_model <- lm(salary_standardized ~ ., data = pred_data)

# View summary of model
linm_sum <- summary(linear_model)
linm_sum
```

```
##
## Call:
## lm(formula = salary_standardized ~ ., data = pred_data)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -95393 -26312    129  19273 232468
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)    94602.43    1133.92  83.429  < 2e-16 ***
## airflow         8276.72   10412.39   0.795 0.426747
## alteryx         2548.64    4907.43   0.519 0.603565
## atlassian      -5577.63   15119.60  -0.369 0.712231
## aws             4879.38    4215.43   1.158 0.247170
## azure           1514.62    4891.22   0.310 0.756843
## bigquery       -6952.32    7540.29  -0.922 0.356600
## c              14197.02    5998.17   2.367 0.018009 *
## c..             9373.23   12930.06   0.725 0.468566
## cognos          5839.11    8972.19   0.651 0.515231
## crystal       -31439.79   14694.56  -2.140 0.032481 *
## dax            -7559.68    7792.57  -0.970 0.332078
## docker          8000.57   14582.65   0.549 0.583301
## excel         -10846.22    1909.77  -5.679 1.50e-08 ***
## gcp           -15121.24   13005.55  -1.163 0.245065
## ggplot2         1081.04   19221.93   0.056 0.955155
## git            13252.28   10796.73   1.227 0.219767
## github         -7688.03   12327.39  -0.624 0.532908
## go               224.69    4843.75   0.046 0.963005
## hadoop         10751.41    7062.90   1.522 0.128068
## html           -3030.75   12586.88  -0.241 0.809739
## java           14028.41    8270.15   1.696 0.089951 .
## javascript      7310.18    7569.83   0.966 0.334283
## jira            8140.17    4892.26   1.664 0.096253 .
## julia         -27414.50   20265.53  -1.353 0.176245
## jupyter        -6611.10   11293.71  -0.585 0.558342
## keras          62794.12   28605.39   2.195 0.028236 *
## linux         -17369.00   12919.04  -1.344 0.178917
## looker          6833.91    4435.87   1.541 0.123532
## matlab         -3342.84    8998.23  -0.371 0.710295
## matplotlib     -7172.36   15640.72  -0.459 0.646581
## microstrategy -10548.66    5220.88  -2.020 0.043434 *
## mongodb        11573.96   14856.17   0.779 0.436009
## mssql          39165.54   20782.99   1.885 0.059606 .
## mysql         -14497.90    7815.79  -1.855 0.063714 .
## nosql          18171.63    8253.18   2.202 0.027767 *
## numpy          19763.22   13788.41   1.433 0.151883
## outlook        -8649.96    6582.92  -1.314 0.188959
## pandas        -11798.51   12208.80  -0.966 0.333935
## pl.sql         20593.11   10417.36   1.977 0.048166 *
## postgres      -21767.39   13687.05  -1.590 0.111871
## postgresql     -9094.87   12489.03  -0.728 0.466538
## power_bi        1919.05    2373.41   0.809 0.418839
## powerpoint      7880.50    4098.26   1.923 0.054600 .
```

```
## powershell     11031.07   15928.14   0.693 0.488651
## pyspark         -179.11   13702.94  -0.013 0.989572
## python          3263.67    2547.23   1.281 0.200214
## pytorch       -29167.46   20475.74  -1.424 0.154422
## qlik            6681.14    5506.71   1.213 0.225133
## r               2911.96    2869.82   1.015 0.310349
## redshift       17314.93    7339.42   2.359 0.018388 *
## ruby           -1981.12   19470.65  -0.102 0.918964
## sap             1649.65    5115.52   0.322 0.747114
## sas             3832.62    3410.46   1.124 0.261205
## scala           2859.38   11477.67   0.249 0.803283
## scikit.learn   -3306.89   18411.01  -0.180 0.857469
## sharepoint      2070.10    5940.84   0.348 0.727528
## shell          -5548.51   21973.52  -0.253 0.800667
## snowflake      15484.67    3549.69   4.362 1.34e-05 ***
## spark          20230.41    8685.32   2.329 0.019919 *
## splunk         21069.92   19197.84   1.098 0.272515
## spreadsheet    -4923.52    3734.69  -1.318 0.187509
## spss          -16221.04    4770.58  -3.400 0.000683 ***
## sql             4209.58    1972.10   2.135 0.032887 *
## ssis             216.97    8171.39   0.027 0.978818
## ssrs            8037.97    8790.89   0.914 0.360614
## swift          11165.39    7460.49   1.497 0.134615
## t.sql           8850.78   11520.24   0.768 0.442388
## tableau          -91.57    2296.30  -0.040 0.968195
## tensorflow    -40057.70   22060.51  -1.816 0.069512 .
## terminal       -5674.81   15558.69  -0.365 0.715337
## unix            9578.68   17349.90   0.552 0.580934
## vba            -5826.71    6683.43  -0.872 0.383389
## visio           6442.98    8026.20   0.803 0.422195
## visual_basic   -4758.97   16401.78  -0.290 0.771724
## word           -7825.10    4249.77  -1.841 0.065689 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 37540 on 2669 degrees of freedom
## Multiple R-squared:  0.08721,    Adjusted R-squared:  0.06156
## F-statistic:   3.4 on 75 and 2669 DF,  p-value: < 2.2e-16
```

Training my model with my Final Project Dataset

I have performed the below code to see how my linear model is performing and here to test my function and the model, I have only selected sql skills to see the predicted salary and we can see that the predicted salary value is $98812.01

```r
# Let's assume that the user inputs the skills as a vector of strings
input_skills <- c("sql")

# Create an empty data frame with the same columns as pred_data
input_data <- data.frame(matrix(ncol = ncol(pred_data), nrow = 1))
colnames(input_data) <- colnames(pred_data)



# Set the values of the input data frame based on the user's input skills
```

```r
for (skill in input_skills) {
  input_data[[skill]] <- 1
}

input_data[is.na(input_data)] <- 0


# Convert the input data frame to the same format as train_data
input_data_scaled <- data.frame(input_data[, -1])


# colnames(input_data_scaled) <- colnames(train_data_scaled)

# Use the trained model to predict the salary
predicted_salary <- predict(linear_model, newdata = input_data_scaled)

# Print the predicted salary
cat("Predicted salary: $", round(predicted_salary, 2), "\n")
```

```
## Predicted salary: $ 98812.01
```

```r
# Calculate RMSE
library(Metrics)
```

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

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

```r
rmse <- rmse(testData$salary, predictions)
rmse
```

```
## [1] 36902.63
```

To calculate the accuracy of a linear regression model in R, I use the coefficient of determination (R-squared) metric. R-squared measures how well the model fits the data and ranges from 0 to 1, with higher values indicating a better fit.

```r
# R-squared value
rsq <- summary(linear_model)$r.squared
print(rsq)
```

```
## [1] 0.08721407
```

```r
library(Metrics)

# Predict on test data
predictions <- predict(linear_model, newdata = testData)
```

```r
# Calculate MAE
mae <- MAE(testData$salary, predictions)
print(paste0("MAE: ", round(mae, 2)))
```

```
## [1] "MAE: 25785.62"
```

Linear Discriminant Analysis

I have also tried implementing the linear discriminant analysis in my final project dataset through which I obtain a summary of my model's result. Discriminant Analysis is a statistical technique which is used for predicting the categorical outcomes based on a set of the predictor variable.

```r
#discriminant Analysis
library(MASS)
```

```
##
## Attaching package: 'MASS'

## The following object is masked from 'package:plotly':
##
##     select

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

```r
# Subset the data to exclude the outcome variable
predictors <- pred_data[, -which(names(pred_data) == "job_category")]

# Fit the discriminant analysis model
discriminant <- lda(salary_standardized ~ ., data = pred_data)

summary(discriminant)
```

```
##         Length Class  Mode
## prior     417  -none- numeric
## counts    417  -none- numeric
## means   31275  -none- numeric
## scaling  5625  -none- numeric
## lev       417  -none- character
## svd        75  -none- numeric
## N           1  -none- numeric
## call        3  -none- call
## terms       3  terms  call
## xlevels     0  -none- list
```

Objective 2: Describe probability as a foundation of statistical modeling, inlcuding inference and maximum likelihood estinamtion

Confidence Interval

I have implemented the confidence interval for the linear model where it calculates the confidence interval for the coefficient of the linear regression model. Through the below confidence interval, it helps in identifying the uncertainty and significance of the relationship between the predictor variable and the dependent variable in the linear regression model.

```
# Calculate confidence intervals for coefficients
ci <- confint(linear_model)
ci
```

```
##                     2.5 %      97.5 %
## (Intercept)    92378.9698  96825.8874
## airflow       -12140.4502  28693.8883
## alteryx        -7074.1203  12171.3915
## atlassian     -35224.9514  24069.6972
## aws            -3386.4629  13145.2158
## azure          -8076.3429  11105.5831
## bigquery      -21737.7144   7833.0706
## c               2435.4807  25958.5536
## c..           -15980.7176  34727.1848
## cognos        -11754.0278  23432.2514
## crystal       -60253.6738  -2625.9076
## dax           -22839.7724   7720.4066
## docker        -20593.8768  36595.0090
## excel         -14591.0001  -7101.4338
## gcp           -40623.2075  10380.7249
## ggplot2       -36610.3358  38772.4231
## git            -7918.5188  34423.0839
## github        -31860.2270  16484.1638
## go             -9273.2001   9722.5806
## hadoop         -3097.9047  24600.7323
## html          -27711.7742  21650.2701
## java           -2188.1447  30244.9714
## javascript     -7533.1554  22153.5100
## jira           -1452.8431  17733.1764
## julia         -67152.2359  12323.2369
## jupyter       -28756.3987  15534.2001
## keras           6703.1468 118885.0906
## linux         -42701.3444   7963.3400
## looker         -1864.1666  15531.9951
## matlab        -20987.0404  14301.3649
## matplotlib    -37841.5208  23496.7991
## microstrategy -20786.0381   -311.2766
## mongodb       -17556.8037  40704.7308
## mssql          -1586.8448  79917.9288
## mysql         -29823.5128    827.7165
## nosql           1988.3540  34354.9061
## numpy          -7273.8296  46800.2791
## outlook       -21558.1040   4258.1781
## pandas        -35738.1814  12141.1657
## pl.sql           166.2014  41020.0101
## postgres      -48605.6997   5070.9106
## postgresql    -33584.0308  15394.2929
## power_bi       -2734.8515   6572.9481
## powerpoint      -155.5889  15916.5806
```

```
## powershell    -20201.6769  42263.8199
## pyspark       -27048.5686  26690.3504
## python         -1731.0854   8258.4215
## pytorch       -69317.3889  10982.4712
## qlik           -4116.7193  17478.9970
## r              -2715.3438   8539.2648
## redshift        2923.4131  31706.4443
## ruby          -40160.2085  36197.9694
## sap            -8381.1306  11680.4395
## sas            -2854.7800  10520.0267
## scala         -19646.6500  25365.4122
## scikit.learn  -39408.1816  32794.3997
## sharepoint     -9579.0110  13719.2112
## shell         -48635.3667  37538.3449
## snowflake       8524.2423  22445.1022
## spark           3199.7692  37261.0575
## splunk        -16574.2298  58714.0690
## spreadsheet   -12246.6939   2399.6560
## spss          -25575.4412  -6866.6367
## sql              342.5796   8076.5869
## ssis          -15805.9272  16239.8736
## ssrs           -9199.6756  25275.6217
## swift          -3463.5320  25794.3032
## t.sql         -13738.7187  31440.2773
## tableau        -4594.2659   4411.1330
## tensorflow    -83315.1097   3199.7196
## terminal      -36183.1132  24833.4863
## unix          -24441.9213  43599.2905
## vba           -18931.9331   7278.5231
## visio          -9295.2311  22181.1869
## visual_basic  -36920.4502  27402.5008
## word          -16158.2786    508.0839
```

Maximum Likelihood Estimates

Finding the parameter values that maximize the likelihood of receiving the observed data under a specific statistical model is done statistically using the maximum likelihood estimation approach. The MLE estimates for linear regression represent the coefficient values that give the observed data the highest likelihood. I tried implementing the function for the maximum likelihood coefficient to find out the coefficient from the linear regression model.

```r
# Calculate maximum likelihood estimates for coefficients
lm_mle <- coef(linear_model)
lm_mle
```

```
##   (Intercept)       airflow       alteryx     atlassian           aws
##   94602.42863    8276.71909    2548.63563   -5577.62708    4879.37644
##         azure      bigquery             c            c..        cognos
##    1514.62007   -6952.32189   14197.01718    9373.23361    5839.11180
##       crystal           dax        docker         excel           gcp
##  -31439.79070   -7559.68288    8000.56608  -10846.21699  -15121.24130
##       ggplot2           git        github            go        hadoop
##    1081.04367   13252.28259   -7688.03162     224.69025   10751.41382
##          html          java    javascript          jira         julia
```

```
##   -3030.75207   14028.41332    7310.17731    8140.16662  -27414.49947
##       jupyter         keras         linux        looker        matlab
##   -6611.09928   62794.11871  -17369.00220    6833.91424   -3342.83775
##    matplotlib microstrategy       mongodb         mssql         mysql
##   -7172.36082  -10548.65737   11573.96356   39165.54197  -14497.89811
##         nosql         numpy       outlook        pandas        pl.sql
##   18171.63003   19763.22475   -8649.96292  -11798.50781   20593.10573
##      postgres    postgresql      power_bi    powerpoint    powershell
##  -21767.39456   -9094.86895    1919.04833    7880.49585   11031.07151
##       pyspark        python       pytorch          qlik             r
##    -179.10910    3263.66808  -29167.45887    6681.13887    2911.96054
##      redshift          ruby           sap           sas         scala
##   17314.92870   -1981.11958    1649.65444    3832.62331    2859.38106
##  scikit.learn    sharepoint         shell     snowflake         spark
##   -3306.89093    2070.10012   -5548.51090   15484.67226   20230.41336
##        splunk   spreadsheet          spss           sql          ssis
##   21069.91957   -4923.51897  -16221.03892    4209.58326     216.97320
##          ssrs         swift         t.sql       tableau    tensorflow
##    8037.97303   11165.38559    8850.77933     -91.56644  -40057.69502
##      terminal          unix           vba         visio  visual_basic
##   -5674.81345    9578.68461   -5826.70501    6442.97787   -4758.97473
##          word
##   -7825.09734
```

Objective 3: Conduct model selection for a set of candidate models.

To look for the model selection,we could refer to various **goodness-of-fit** measures such as R-square, adjusted R-square, Akaike Information Criterion (AIC) or Bayesian Information Criterion(BIC). I implemented the statistical modeling through the below code which provides various statistics foe each coefficient. The tidy format helps in easily manipulating, visualizing, and analyzing the model output. Furthermore, with the values we get from the *AIC* and *BIC*, it helps in assessing and comparing different models based on their goodness of fit and complexity.

```r
library(broom)

tidy(linear_model)
```

```
## # A tibble: 76 x 5
##    term        estimate std.error statistic p.value
##    <chr>          <dbl>     <dbl>     <dbl>   <dbl>
##  1 (Intercept)   94602.     1134.    83.4    0
##  2 airflow        8277.    10412.     0.795  0.427
##  3 alteryx        2549.     4907.     0.519  0.604
##  4 atlassian     -5578.    15120.    -0.369  0.712
##  5 aws            4879.     4215.     1.16   0.247
##  6 azure          1515.     4891.     0.310  0.757
##  7 bigquery      -6952.     7540.    -0.922  0.357
##  8 c             14197.     5998.     2.37   0.0180
##  9 c..            9373.    12930.     0.725  0.469
## 10 cognos         5839.     8972.     0.651  0.515
## # ... with 66 more rows
```

```r
glance(linear_model)
```

```
## # A tibble: 1 x 12
##   r.squared adj.r.~1  sigma stati~2  p.value    df  logLik    AIC    BIC devia~3
##       <dbl>    <dbl>  <dbl>   <dbl>    <dbl> <dbl>   <dbl>  <dbl>  <dbl>   <dbl>
## 1    0.0872   0.0616 37538.    3.40 3.39e-20    75 -32770. 65694. 66149. 3.76e12
## # ... with 2 more variables: df.residual <int>, nobs <int>, and abbreviated
## #   variable names 1: adj.r.squared, 2: statistic, 3: deviance
```

Cross-Validation

Cross-validation is used to for estimating the performance of the predictive model. Cross-validation involves diving the data into two parts. The validation set is used to estimate the model's performance on new or the unseen data. We mainly use cross-validation to avoid the overfitting as implementing the cross-validation helps in getting more accurate estimate of the model's performance.

Here, I have implemented the cross validation where I have set up a 10-fold cross validation which means that the data will be split into 10 equal parts and the model will be trained on 9 parts and tested on the remainig part.

```r
# training a linear regression model using 10-fold cross-validation
control <- trainControl(method = "cv", number = 10)
Fit <- train(salary_standardized ~ ., data = trainData, method = "lm", metric = "RMSE", trControl = con
# pred_data$salary_standardized

print(Fit)
```

```
## Linear Regression
##
## 2198 samples
##   75 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1979, 1979, 1979, 1977, 1979, 1978, ...
## Resampling results:
##
##   RMSE      Rsquared    MAE
##   38922.76  0.04047155  28356.68
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

From the above output, we can compute that the EMSE is the measure of the average difference between the predicted value and the actual value. Similarly, by looking at the R-squared value, mean absolute error, RMSE we can have a model selection based on that. The R-squared measures the proportion of the variance while the mean absolute error measure the absolute difference between the predicted and actual values.

Calculating RMSE

This code calculates the root mean square between the predicted salaries and the actual salaries in the rest dataset. The difference between the predicted and actual salaries is first squared, then the mean of these squared differences is taken, and finally the square root of this mean is calculated.

```r
# Calculate RMSE
mse <- mean((predictions - testData$salary_standardized)^2)
rmse <- sqrt(mse)
rmse
```

```
## [1] 34833.4
```

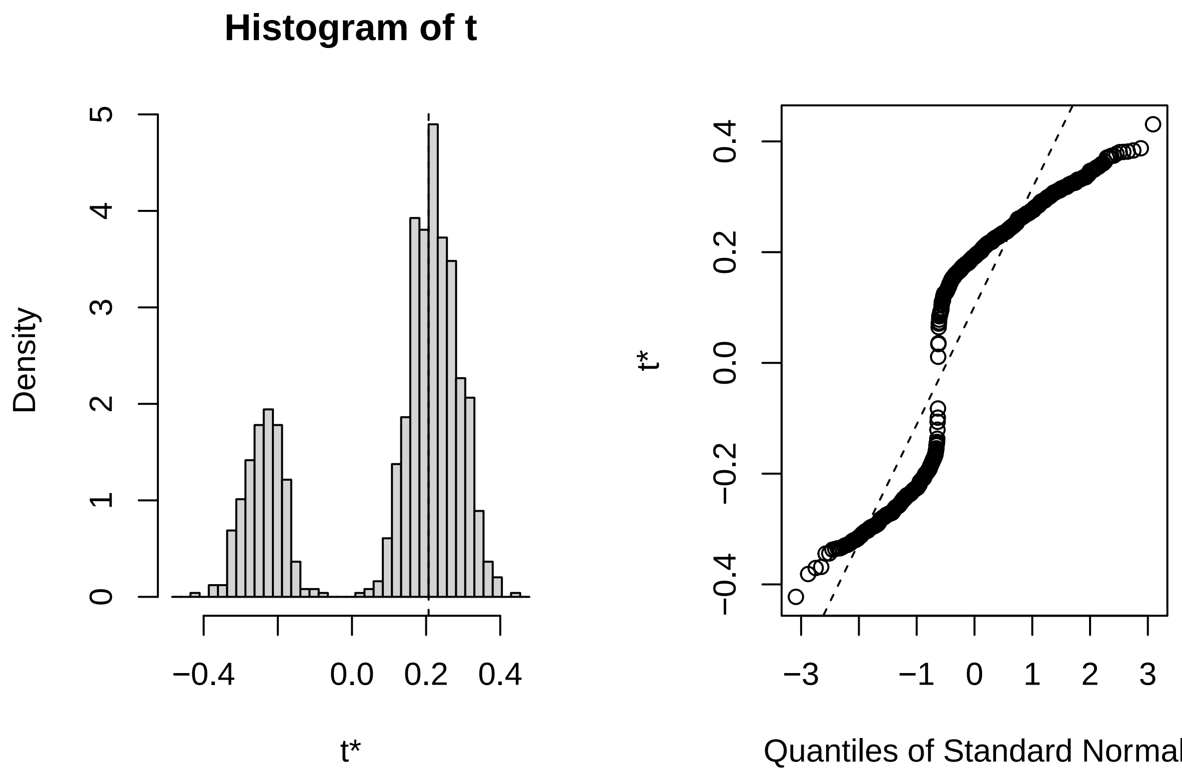
Bootstrapping

```r
library(caret)

# Identify near zero variance predictors in the data frame
nzv <- nearZeroVar(pred_data, saveMetrics = TRUE)

# Remove the identified near zero variance predictors
pred_data <- pred_data[, !nzv$nzv]

lda_model <- lda(salary_standardized ~ ., data = pred_data)
```

```r
#Bootstrapping
library(MASS)
library(boot)
```

```
##
## Attaching package: 'boot'

## The following object is masked from 'package:lattice':
##
##     melanoma
```

```r
# Subset the data to exclude the outcome variable
predictors <- pred_data[, -which(names(pred_data) == "job_category")]


lda_coef <- function(data, index) {
  fit <- lda(salary_standardized ~ ., data = data[index, ])
  coef(fit)
}

# Use bootstrapping to estimate the standard errors of the coefficients
set.seed(123) # for reproducibility
boot_lda <- boot(data = pred_data, statistic = lda_coef, R = 1000)

plot(boot_lda)
```

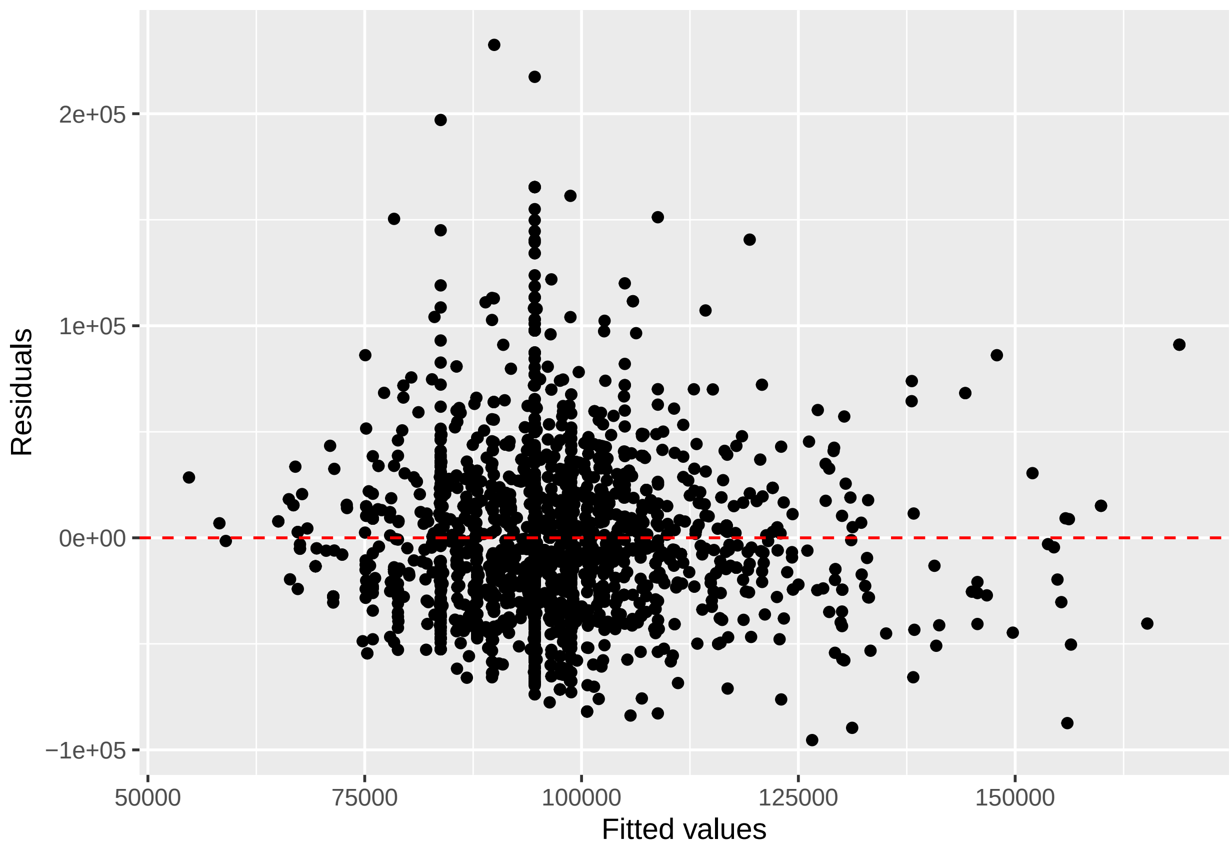


Objective 4: Communicate the results of statistical models to a general audience

Communicating the result plays an important role and when communicating the result, it should be very precise so that the audience can easily understand it. It is better to use plain language and while there might be some of the technical concepts and technical terminology, it is better to provide context to the audience. I'm a great believer of visual aids such as graph, chart, table which could easily grab an attention of the audience and they can easily understand the findings.

The below plot is one of the visual aid for my linear model. It is a residual plot which is used to check the assumption of the linear model. The horizontal axix represents the predicted values and the vertical axis represents the residuals or the difference between the observed values and the predicted values. As this plot shows a random scatter points we can compute that my linear model met the assumption.

```r
R_fit <- augment(linear_model)

# plot fitted values and residuals
ggplot(data = R_fit, aes(x = .fitted, y = .resid)) +
  geom_point() +
  geom_hline(yintercept = 0, linetype = "dashed", color = "red") +
  xlab("Fitted values") +
  ylab("Residuals")
```



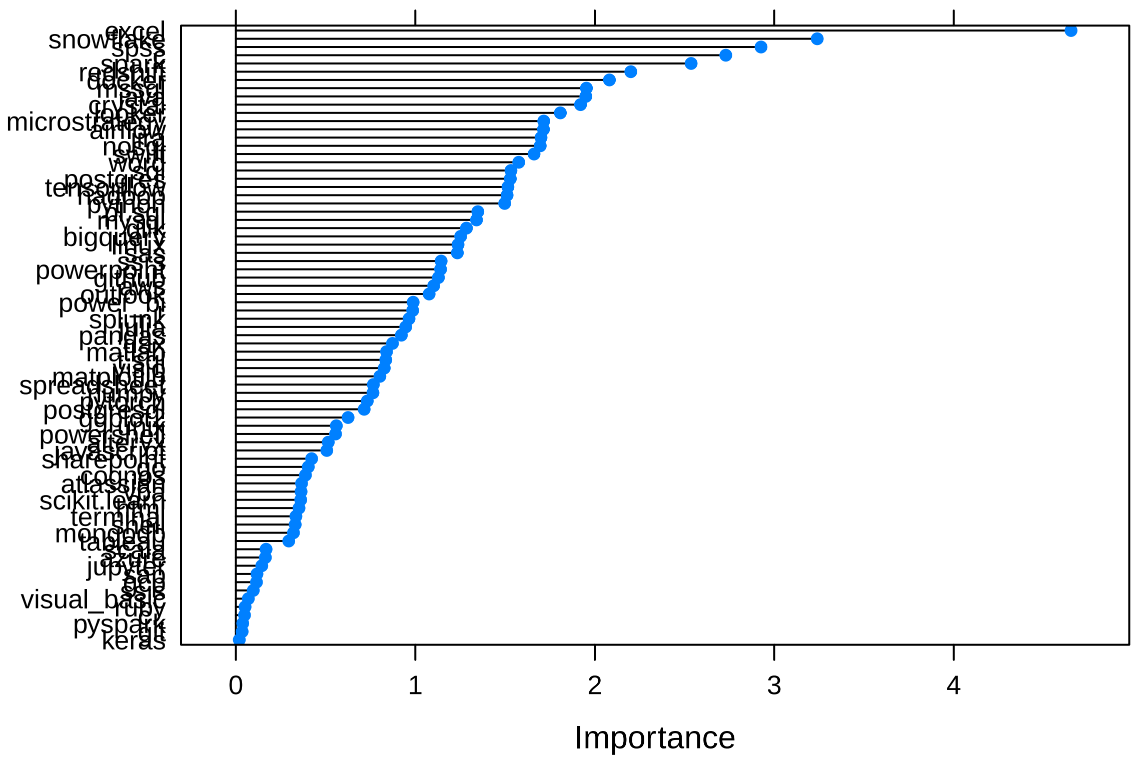
The below code is another method of communicating the result to the audience which is through the table data. This helps us in showing the importance of each variable as per the ranking.

```
Imp <- varImp(Fit, scale = FALSE)
print(Imp)
```

```
## lm variable importance
##
##   only 20 most important variables shown (out of 75)
##
##                Overall
## excel            4.653
## snowflake        3.239
## spss             2.926
## c                2.730
## spark            2.537
## redshift         2.201
## docker           2.081
## mssql            1.953
## java             1.950
## crystal          1.920
## looker           1.808
## microstrategy    1.716
## airflow          1.714
## jira             1.701
## nosql            1.696
```

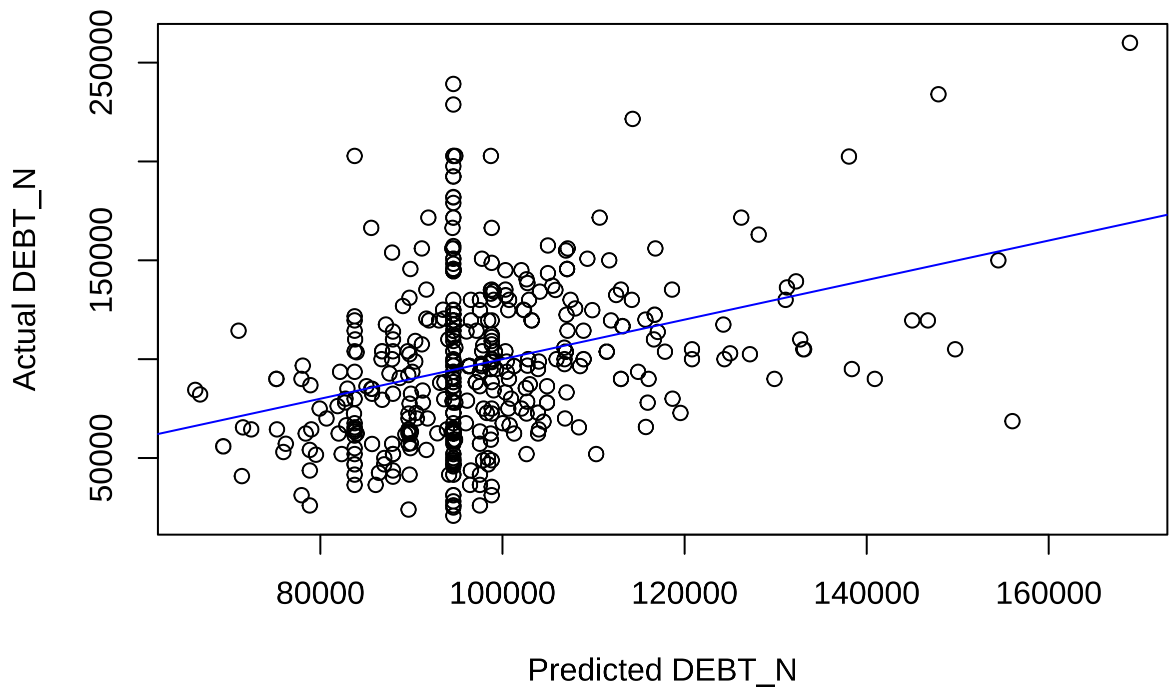
```
## swift              1.662
## word               1.576
## sql                1.533
## postgres           1.529
## tensorflow         1.516
```

```r
plot(Imp)
```



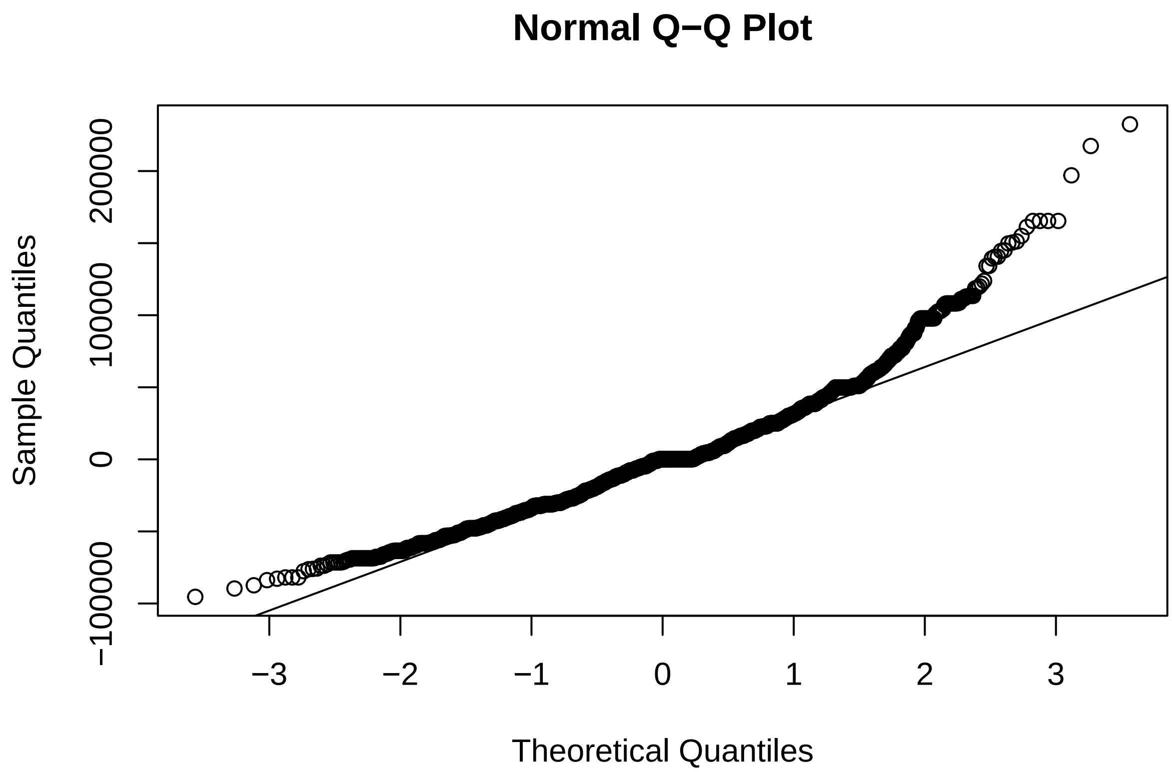
This code below creates a scatter plot of predicted vs the actual values. It helps the audience in getting an understanding of how the model is performing and the diagonal line to the plot represents the prediction.

```r
# Create scatter plot of predicted vs. actual values
plot(predictions, testData$salary_standardized, xlab = "Predicted DEBT_N", ylab = "Actual DEBT_N")

# Add a diagonal line to show perfect predictions
abline(0, 1, col = "Blue")
```



I have also used a QQ plot in my final project to have a overview of how my model is performing and to check the assumptions for my linear model. QQ plot helps in visually comparing the distribution of the dataset which is also useful for identifying deviations from normality, detecting outliers, adn assessign the overall shape of the data distribution.

```
# Get the residuals
residuals <- residuals(linear_model)

#  QQ plot
qqnorm(residuals)
qqline(residuals)
```



Objective 5: Use programming software (i.e. R) to fit and assess statistical models

Apart from implementing linear regression, logistic regression and discriminant analysis, I also tried implementing other various machine learning algorith in my dataset to see how each different machine learning models performed.

Random Forest

I implemented the below code to see how it predicts for the regression analysis. I have used the randomForest function to fit the random forest model and it then calculates the mean absolute error to evaluate the model's performance.

```
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:ggplot2':
##
##     margin
```

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

```r
set.seed(123)
# Split data into train and test sets
train_index <- sample(nrow(pred_data), 0.7 * nrow(pred_data))
train_data <- pred_data[train_index, ]
test_data <- pred_data[-train_index, ]

# Fit random forest model
rf_model <- randomForest(salary_standardized ~ ., data = train_data, ntree = 75, mtry = sqrt(ncol(train_

# Make predictions on the test data
predictions <- predict(rf_model, test_data)

# Calculate the mean absolute error
mae <- mean(abs(predictions - test_data$salary_standardized))
print(paste0("MAE: ", round(mae, 2)))
```

```
## [1] "MAE: 27567.19"
```

XGBOOST

I also tried implementing the XGBoost and tried to see the same result on how the model performed through fitting the model and finding the mean absolute error.

```r
library(xgboost)
```

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

## The following object is masked from 'package:plotly':
##
##     slice

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

```r
# Split data into train and test sets
train_index <- sample(nrow(pred_data), 0.7 * nrow(pred_data))
train_data <- pred_data[train_index, ]
test_data <- pred_data[-train_index, ]

# Set up xgboost matrix
xg_train <- xgb.DMatrix(as.matrix(train_data[, -1]), label = train_data$salary_standardized)
xg_test <- xgb.DMatrix(as.matrix(test_data[, -1]), label = test_data$salary_standardized)

# Define hyperparameters
params <- list(
  objective = "reg:squarederror",
```

```r
  eta = 0.1,
  max_depth = 6,
  subsample = 0.8,
  colsample_bytree = 0.8,
  min_child_weight = 3,
  nthread = 4
)

# Train the model
xgb_model <- xgb.train(
  params = params,
  data = xg_train,
  nrounds = 1000,
  watchlist = list(train = xg_train, test = xg_test),
  early_stopping_rounds = 10,
  verbose = 1
)
```

```
## [1]  train-rmse:96020.964474 test-rmse:93977.405804
## Multiple eval metrics are present. Will use test_rmse for early stopping.
## Will train until test_rmse hasn't improved in 10 rounds.
##
## [2]  train-rmse:88097.680713 test-rmse:86009.915517
## [3]  train-rmse:81074.660717 test-rmse:78992.486075
## [4]  train-rmse:74819.956821 test-rmse:72726.828975
## [5]  train-rmse:69376.697053 test-rmse:67243.279901
## [6]  train-rmse:64690.134319 test-rmse:62509.749484
## [7]  train-rmse:60508.315487 test-rmse:58315.733676
## [8]  train-rmse:56950.965177 test-rmse:54794.452672
## [9]  train-rmse:53890.448203 test-rmse:51759.291708
## [10] train-rmse:51273.947813 test-rmse:49141.791289
## [11] train-rmse:48995.768933 test-rmse:46910.634849
## [12] train-rmse:47128.160359 test-rmse:45072.054185
## [13] train-rmse:45475.906196 test-rmse:43457.174830
## [14] train-rmse:44115.155108 test-rmse:42167.571793
## [15] train-rmse:42952.980757 test-rmse:41043.255626
## [16] train-rmse:42003.083490 test-rmse:40155.308343
## [17] train-rmse:41209.517964 test-rmse:39416.541371
## [18] train-rmse:40562.671356 test-rmse:38811.084027
## [19] train-rmse:40019.672351 test-rmse:38324.546546
## [20] train-rmse:39555.123575 test-rmse:37926.341045
## [21] train-rmse:39142.506116 test-rmse:37567.713271
## [22] train-rmse:38814.647607 test-rmse:37274.830352
## [23] train-rmse:38570.967583 test-rmse:37079.322012
## [24] train-rmse:38348.979761 test-rmse:36906.736672
## [25] train-rmse:38161.833708 test-rmse:36766.393295
## [26] train-rmse:38021.502608 test-rmse:36660.898206
## [27] train-rmse:37890.194362 test-rmse:36556.609104
## [28] train-rmse:37799.752530 test-rmse:36507.090796
## [29] train-rmse:37708.812529 test-rmse:36432.474030
## [30] train-rmse:37616.912460 test-rmse:36398.688892
## [31] train-rmse:37546.698973 test-rmse:36375.187311
## [32] train-rmse:37486.279222 test-rmse:36347.119599
```

```
## [33] train-rmse:37451.311796 test-rmse:36333.674719
## [34] train-rmse:37413.284153 test-rmse:36298.362195
## [35] train-rmse:37379.921109 test-rmse:36279.097538
## [36] train-rmse:37348.650128 test-rmse:36270.241732
## [37] train-rmse:37321.478766 test-rmse:36274.198836
## [38] train-rmse:37304.975568 test-rmse:36268.267291
## [39] train-rmse:37288.446425 test-rmse:36260.234863
## [40] train-rmse:37268.247789 test-rmse:36267.342676
## [41] train-rmse:37253.598762 test-rmse:36273.011713
## [42] train-rmse:37241.606795 test-rmse:36269.352969
## [43] train-rmse:37230.073626 test-rmse:36272.121434
## [44] train-rmse:37219.686886 test-rmse:36273.704319
## [45] train-rmse:37213.842137 test-rmse:36266.278565
## [46] train-rmse:37203.748594 test-rmse:36269.870040
## [47] train-rmse:37195.645609 test-rmse:36271.315259
## [48] train-rmse:37189.421663 test-rmse:36277.955429
## [49] train-rmse:37183.046888 test-rmse:36284.884097
## Stopping. Best iteration:
## [39] train-rmse:37288.446425 test-rmse:36260.234863
```

```r
# Make predictions on the test data
predictions <- predict(xgb_model, xg_test)

# Calculate the mean absolute error
mae <- mean(abs(predictions - test_data$salary_standardized))
print(paste0("MAE: ", round(mae, 2)))
```

```
## [1] "MAE: 26650.92"
```

I also tried implementing my cod of the XGBoost to predict the salary of the skills for the sql which showed a slightly different salary than that of the linear model.

```r
# Let's assume that the user inputs the skills as a vector of strings
input_skills <- c("sql")

# Create an empty data frame with the same columns as pred_data
input_data <- data.frame(matrix(ncol = ncol(pred_data), nrow = 1))
colnames(input_data) <- colnames(pred_data)


# Set the values of the input data frame based on the user's input skills
for (skill in input_skills) {
  input_data[[skill]] <- 1
}

input_data[is.na(input_data)] <- 0


# Convert the input data frame to the same format as train_data
input_data_scaled <- data.frame(input_data[, -1])


# Predict salary using xgboost model
```

```r
input_xg <- xgb.DMatrix(as.matrix(input_data_scaled))
predicted_salary <- predict(xgb_model, input_xg)

# Print predicted salarys
print(paste0("Predicted Salary: ", round(predicted_salary, 2)))
```

```
## [1] "Predicted Salary: 96306.84"
```

Support vector machine

```r
library(e1071)

# Split data into train and test sets
train_index <- sample(nrow(pred_data), 0.7 * nrow(pred_data))
train_data <- pred_data[train_index, ]
test_data <- pred_data[-train_index, ]

# Train SVM model
svm_model <- svm(salary_standardized ~ ., data = train_data)

# Make predictions on the test data
predictions <- predict(svm_model, test_data)

# Calculate the mean absolute error
mae <- mean(abs(predictions - test_data$salary_standardized))
print(paste0("MAE: ", round(mae, 2)))
```

```
## [1] "MAE: 26891.75"
```

Support Vector Regression (SVR)

```r
library(e1071)

# Split data into train and test sets
train_index <- sample(nrow(pred_data), 0.7 * nrow(pred_data))
train_data <- pred_data[train_index, ]
test_data <- pred_data[-train_index, ]

# Define the formula for the SVR
formula <- as.formula("salary_standardized ~ .")

# Train the SVR model
svm_model <- svm(formula, data=train_data, kernel="radial")

# Make predictions on the test data
predictions <- predict(svm_model, test_data)

# Calculate the mean absolute error
mae <- mean(abs(predictions - test_data$salary_standardized))
print(paste0("MAE: ", round(mae, 2)))
```

```
## [1] "MAE: 27043.18"
```

Among all the machine learning models I tried Support Vector machine had the lower mean absolute error.

Conclusion

I believe I have met all the objective of this course as this course has been a great learning experience for me to hone my skills and brush my theroteical concepts regarding the statistical modeling. I have attached my codes and the description on hiw I have met all the objective of the course above.