PSTAT 131 Final Project: Model comparison for predicting the salary of a data science job

Nicholas Axl Andrian

2023-11-23

Dataset used: https://www.kaggle.com/datasets/arnabchaki/data-science-salaries-2023

In this project, we will be fitting several different machine learning algorithms to find out which method of prediction is the most accurate in getting the predicted salary(in usd).

About the data set's variables (excerpt from the kaggle site)

- work_year: The year the salary was paid.
- experience_level: The experience level in the job during the year
- employment_type: The type of employment for the role
- job_title: The role worked in during the year.
- salary: The total gross salary amount paid.
- salary_currency: The currency of the salary paid as an ISO 4217 currency code.
- salary_in_usd: The salary in USD
- employee_residence: Employee's primary country of residence in during the work + year as an ISO 3166 country code.
- remote_ratio: The overall amount of work done remotely
- company_location: The country of the employer's main office or contracting branch
- company_size: The median number of people that worked for the company during the year

```r
library(dplyr)
library(randomForest)
library(gbm)
library(ISLR)
library(tree)
library(tidyverse)
library(ggplot2)
library(gridExtra)
library(class)
library(FNN)
library(tibble)
library(recipes)
library(maptree)
```

Part 1: Exploratory Data Analysis

Loading the dataset

```r
salaries <- read.csv("ds_salaries.csv")
head(salaries)
```

Checking the structure of the dataset

```r
str(salaries)
```

Already we can see an issue that needs to be worked on. Several variables seem to supposedly be read in as factors. We will finish conducting checks on the dataset before converting said columns.

Checking the summary of the dataset

```r
summary(salaries)
```

Checking for null values

```r
colSums(is.na(salaries))
```

```
##          work_year   experience_level    employment_type          job_title
##                  0                  0                  0                  0
##             salary    salary_currency      salary_in_usd employee_residence
##                  0                  0                  0                  0
##       remote_ratio   company_location       company_size
##                  0                  0                  0
```

Fortunately, we have no null values so imputing is not required

Checking potential factor columns for their unique values

```r
factor_cols <- salaries[, c(1, 2, 3, 4, 6, 8, 10, 11)]

# finding unique values, referenced code from https://www.kaggle.com/code/abdulfaheem11/data-science-sa

# output ommitted to prevent too much space being taken up
uniques <- sapply(factor_cols, function(col) unique(col))
```

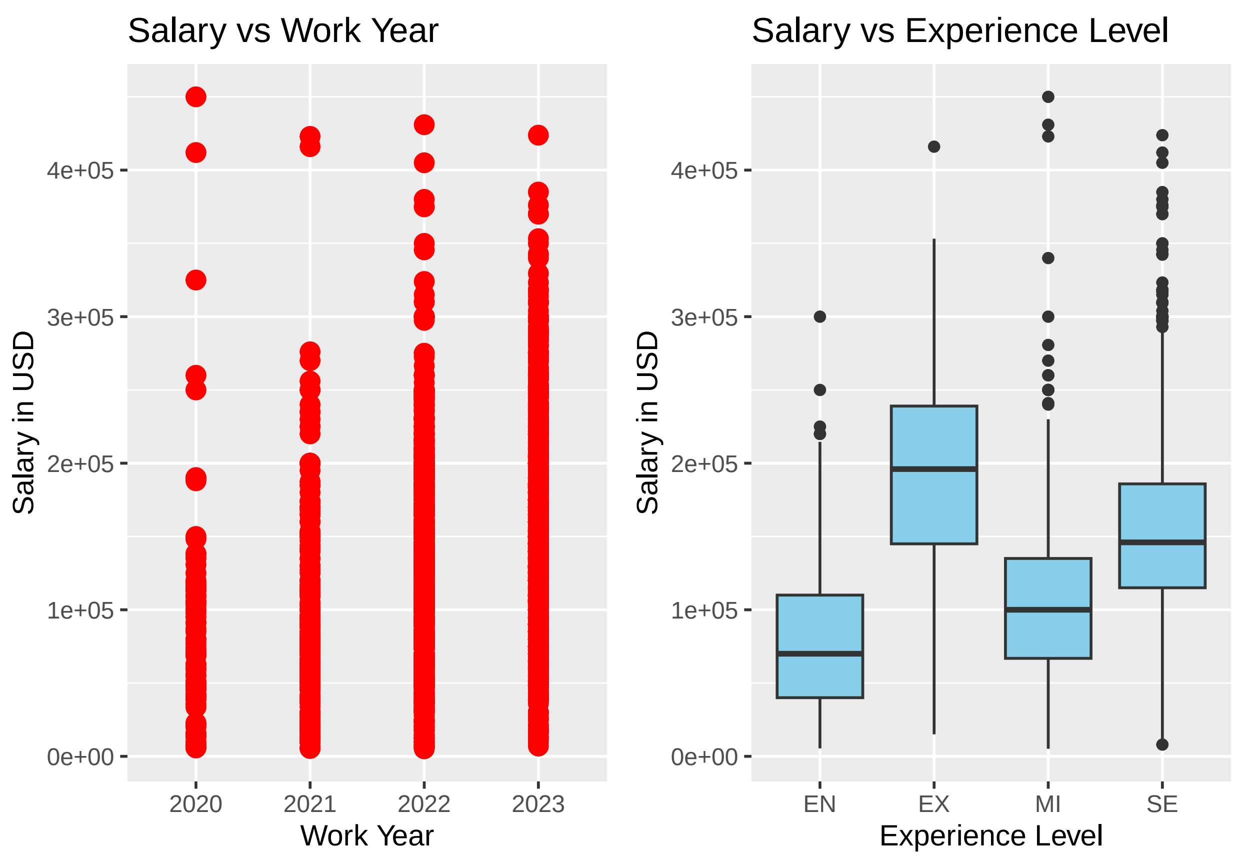
Changing said variables to become factors

```r
salaries[, c(1, 2, 3, 4, 6, 8, 10, 11)] <- lapply(factor_cols, factor)
str(salaries)
```

Visualization to search for patterns with regards to the salary__in__usd

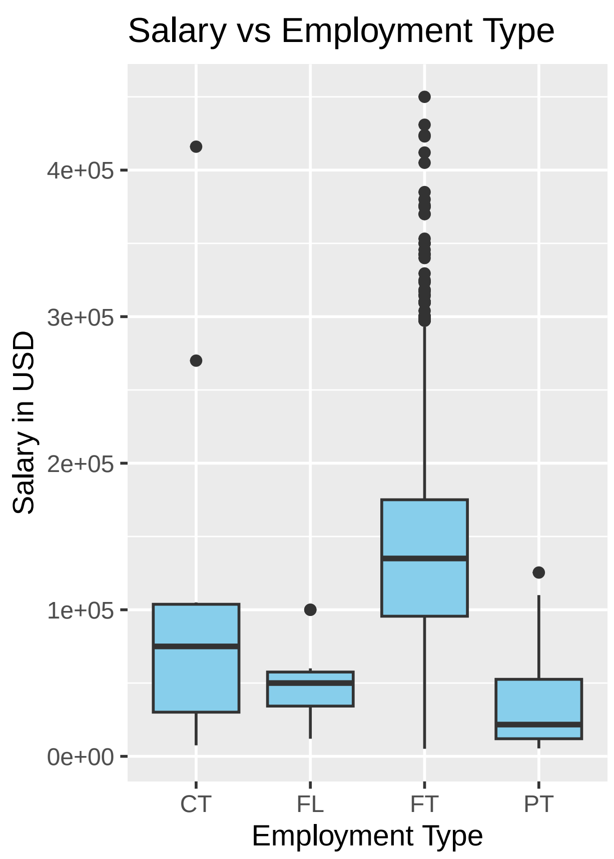
Prioritizing focus on work__year, experience__level, employment__type, job__title, employee__residence, remote__ratio, company__location, company__size

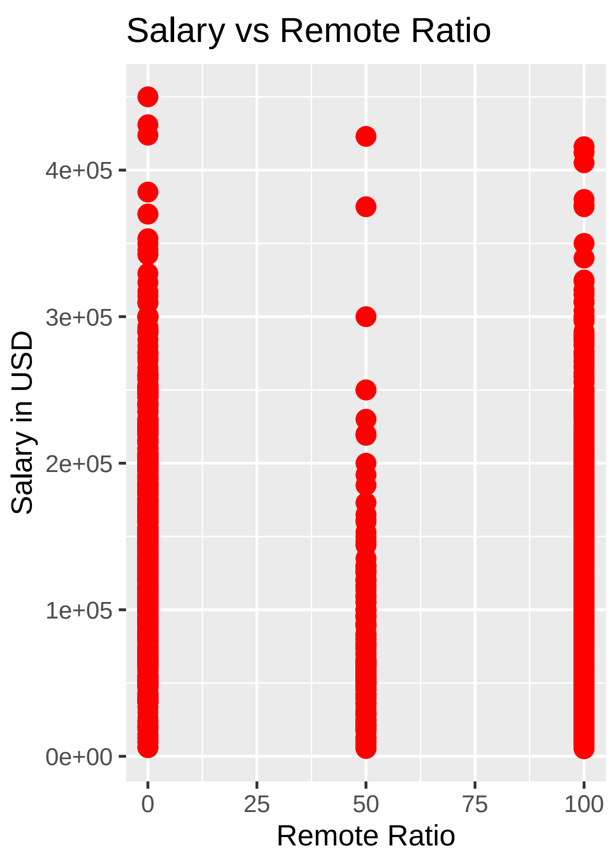
```r
yearplot <- ggplot(salaries, aes(x = work_year, y = salary_in_usd)) +
  geom_point(color = "red", size = 3) +
  labs(x = "Work Year", y = "Salary in USD", title = "Salary vs Work Year")
expplot <- ggplot(salaries, aes(x = experience_level, y = salary_in_usd)) +
  geom_boxplot(fill = "skyblue") +
  labs(x = "Experience Level", y = "Salary in USD", title = "Salary vs Experience Level")
grid.arrange(yearplot, expplot, ncol = 2)
```



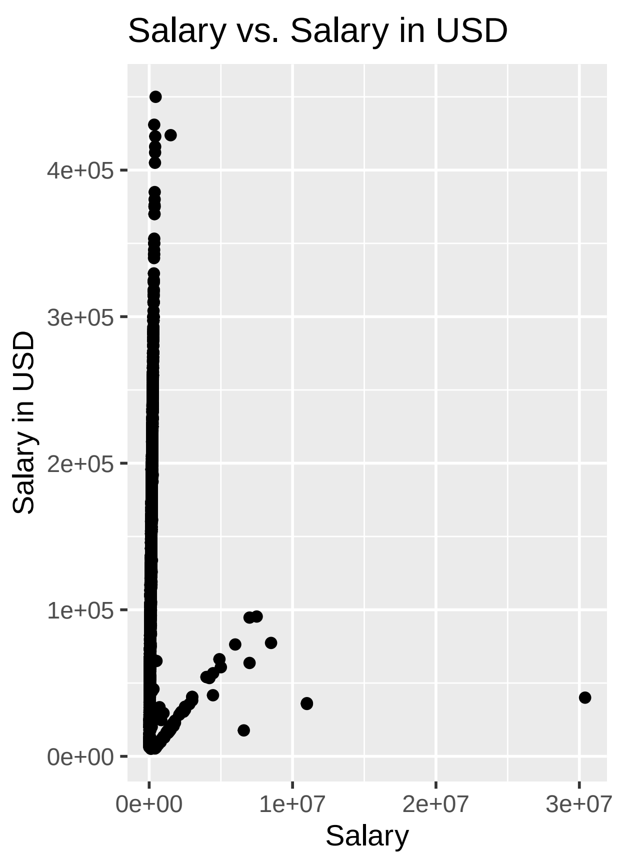
- We can see that the average salary in usd increases as the years go by, as the line congests further upwards towards the end.
- Experience level does not really show much of a trend as it goes towards seniority, We can tell though that EX has the highest average and MI has the highest peak.

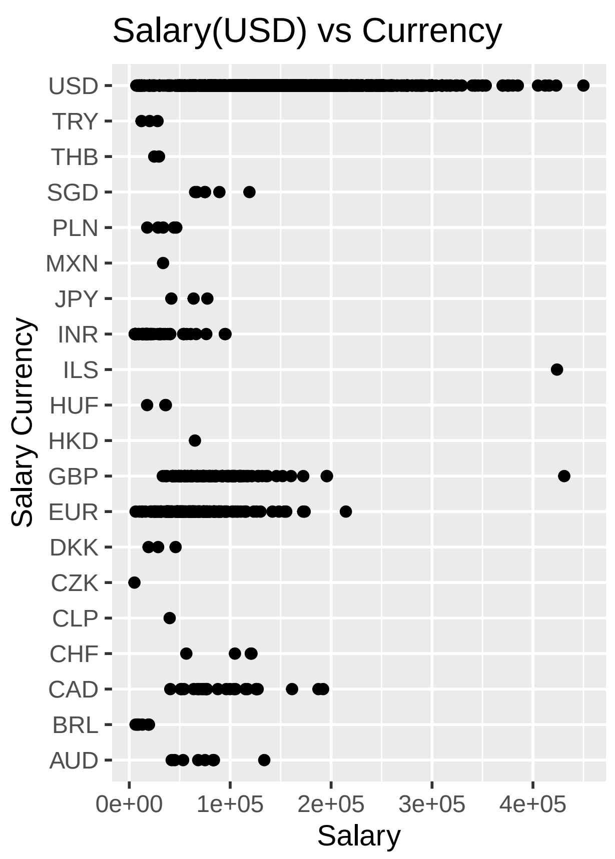
```r
employplot <- ggplot(salaries, aes(x = employment_type, y = salary_in_usd)) +
  geom_boxplot(fill = "skyblue") +
  labs(x = "Employment Type", y = "Salary in USD", title = "Salary vs Employment Type")
remoteplot <- ggplot(salaries, aes(x = remote_ratio, y = salary_in_usd)) +
  geom_point(color = "red", size = 3, shape = 19) +
  labs(x = "Remote Ratio", y = "Salary in USD", title = "Salary vs Remote Ratio")
grid.arrange(employplot, remoteplot, ncol = 2)
```





- In employment type, FT has the highest average as well as higher peaks
- Remote ratio consists of 0, 50 and 100. The highest points as well as average are in the order 0>100>50



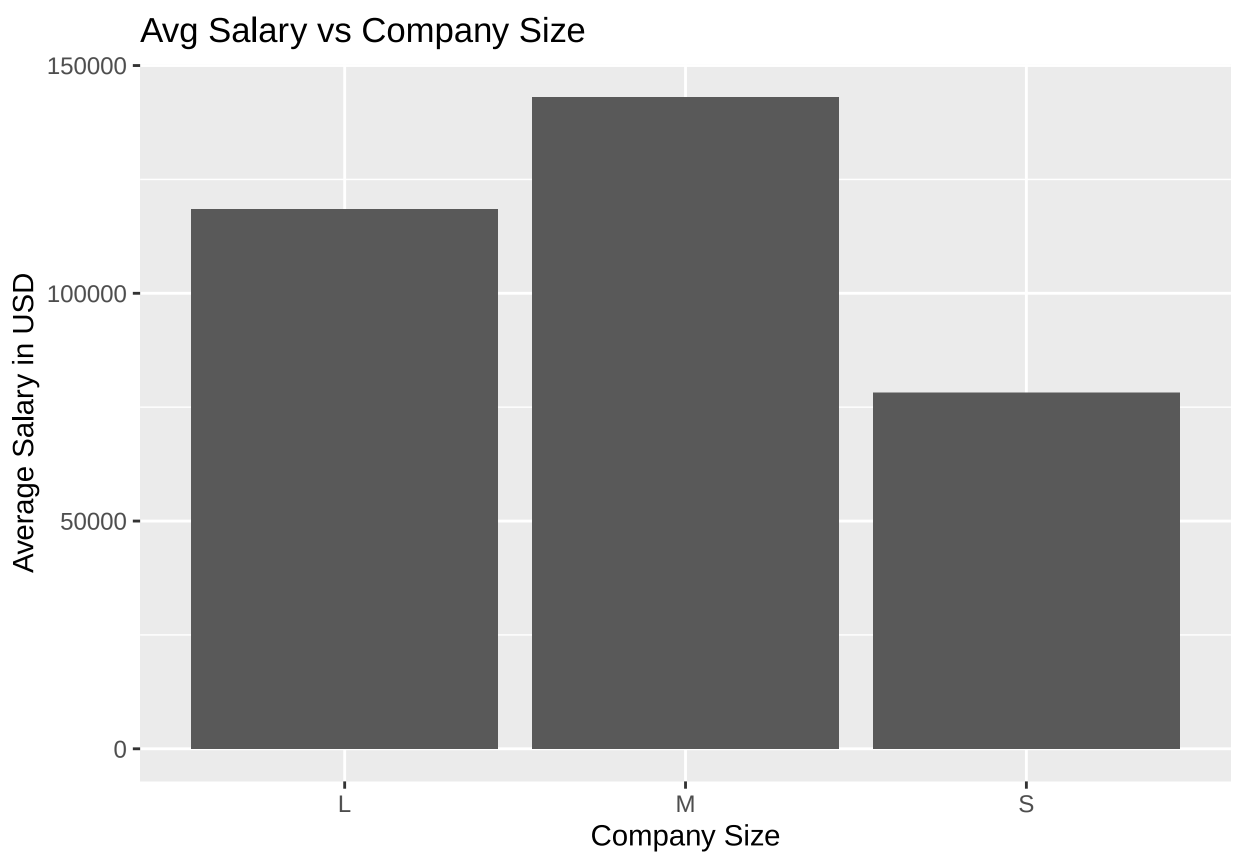


- In the first plot, we see that there is a point that is way further to the right than any of the other points. We will have to check that out right after this
- On the 2nd plot, aside from seeing that we might have the most observations in USD, we can also see that it has the highest salary just based on numbers

Checking out the "outlier" observation

```
max(salaries$salary)
outlierindex <- which(salaries$salary == max(salaries$salary))
salaries$salary_currency[outlierindex]
```

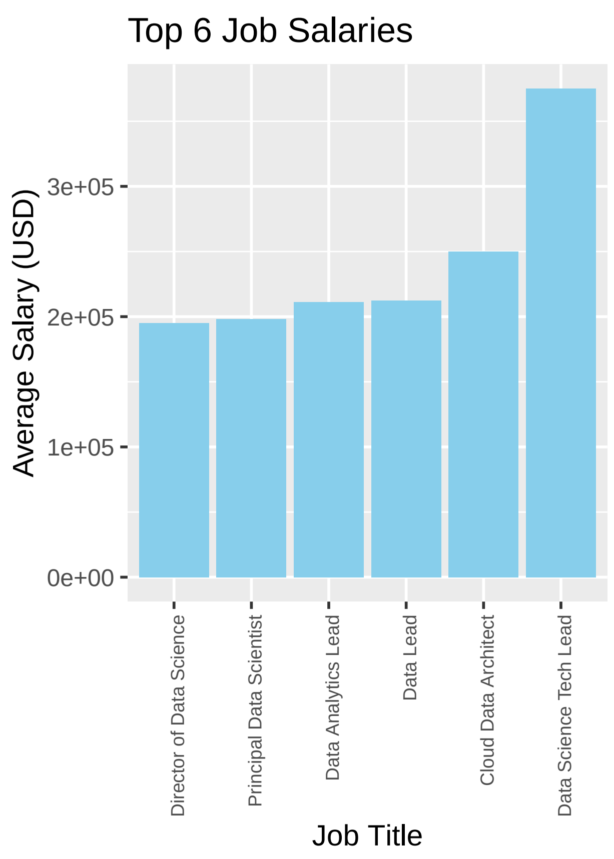
It seems to be an observation from the CLP currency. and connecting it back to the graph, we can see that it also only has that one observation. I will deem it insignificant for now, to avoid a potential leverage point affecting the models I will be removing said observation.

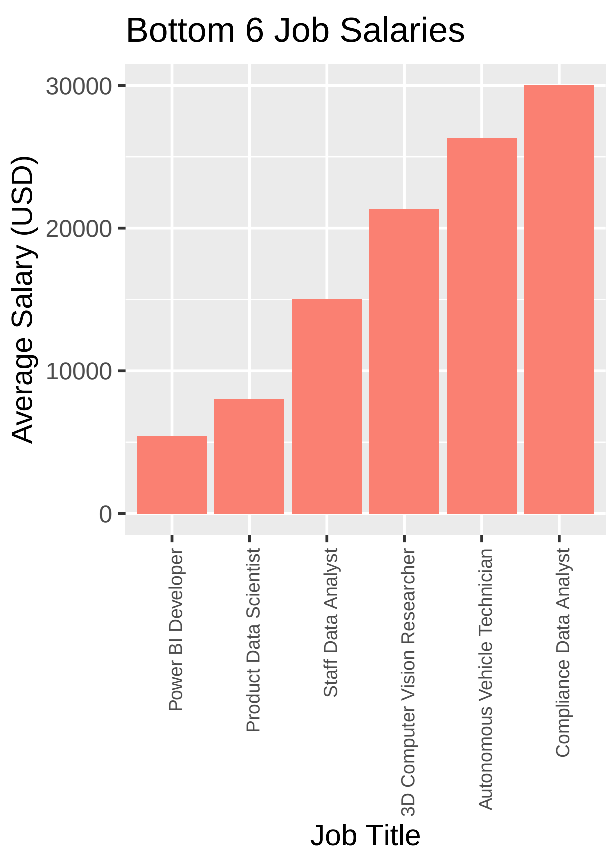


- From this plot we can also see that medium sized companies pay the largest on average, followed by large then small

Tabling the 6 highest and lowest paying jobs

```
top_6_job_salaries<-salaries%>%
  group_by(job_title)%>%
  summarise(Avg_Sal=mean(salary_in_usd))%>%
  arrange(desc(Avg_Sal))%>%
  head()
top_6_job_salaries
bottom_6_job_salaries<-salaries%>%
  group_by(job_title)%>%
  summarise(Avg_Sal=mean(salary_in_usd))%>%
  arrange(Avg_Sal)%>%
  head()
bottom_6_job_salaries
```

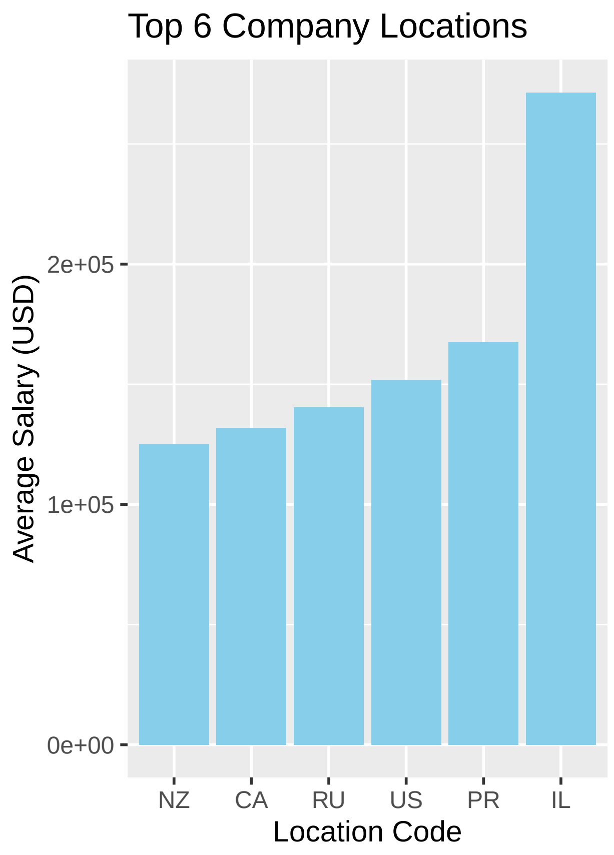


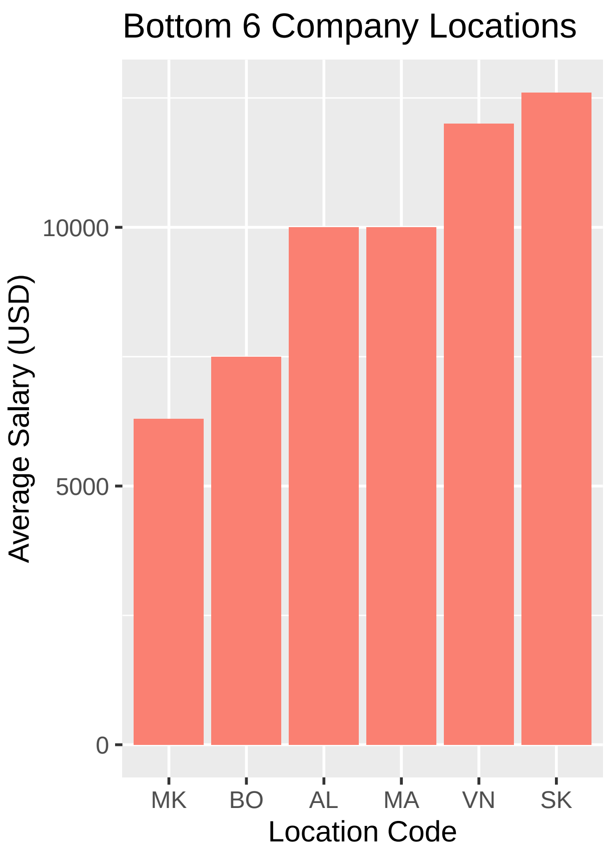


- We can tell that the data science tech lead job has the highest average pay by far
- we can also tell that the power bi developer has the lowest pay out of all the jobs

Tabling the 6 highest and lowest paying company locations

```r
top_6_loc<-salaries%>%
  group_by(company_location)%>%
  summarise(Avg_Sal=mean(salary_in_usd))%>%
  arrange(desc(Avg_Sal))%>%
  head()
top_6_loc
bottom_6_loc<-salaries%>%
  group_by(company_location)%>%
  summarise(Avg_Sal=mean(salary_in_usd))%>%
  arrange(Avg_Sal)%>%
  head()
bottom_6_loc
```





- It seems that IL has the highest paying jobs when it comes to company location
- MK has the lowest paying job out of all the locations by far

Part 2: Problem formulation and discussion of statistical learning algorithms used

The main goal of this project, as mentioned before is to compare several statistical learning methods in predicting the salary given the parameters. We will be focusing on the following methods:

- K-Nearest Neighbors
- Linear Regression
- Regression Trees
  - Decision Trees
  - Random-Forest
  - Bagging
  - Boosting

I want to start off by deciding that the salary_currency and salary are not significant to our prediction of the salary_in_usd. Similarly for employee_residence, as the company_location should be sufficient

```r
salaries <- subset(salaries, select = -c(salary, salary_currency, employee_residence))
head(salaries)
```

Due to a huge amount of levels in factors, our training dataset may not have the columns present in the testing dataset and vice versa. To prevent this, I will create a new data set where I use one-hot encoding on those categorical variables

```r
library(caret)

cat_cols <- c("job_title", "company_location")
formula <- as.formula(paste("~ ."))
encoded_data <- dummyVars(formula, data = salaries[, cat_cols])
encoded_salaries <- predict(encoded_data, newdata = salaries)
encoded_salaries <- cbind(salaries, encoded_salaries)
encoded_salaries <- subset(encoded_salaries, select = -c(job_title, company_location))
colnames(encoded_salaries) <- gsub(" ", "_", colnames(encoded_salaries))
head(encoded_salaries)
```

Preparing the encoded training and testing data set with Train/Test/Split

```r
set.seed(123)
encoded_train = sample(1:nrow(encoded_salaries), 3003)
encoded_salaries_train = encoded_salaries[encoded_train, ]
encoded_salaries_test = encoded_salaries[-encoded_train, ]

encoded_YTrain = encoded_salaries_train$salary_in_usd
encoded_XTrain = encoded_salaries_train %>% select(-salary_in_usd)

encoded_YTest = encoded_salaries_test$salary_in_usd
encoded_XTest = encoded_salaries_test %>% select(-salary_in_usd)
```

Part 3: Model Fitting

K-NN Regression

We need to encode all of the categorical variables for this. To do so, create a new dataset just for the KNN's use

```r
cat_cols_2 <- c("job_title", "company_location", "experience_level", "employment_type", "company_size",
formula <- as.formula(paste("~ ."))
encoded_data2 <- dummyVars(formula, data = salaries[, cat_cols])
encoded_salaries2 <- predict(encoded_data2, newdata = salaries)
encoded_salaries2 <- cbind(salaries, encoded_salaries2)
encoded_salaries2 <- subset(encoded_salaries2, select = -c(job_title, company_location, experience_level

set.seed(123)
nrow(encoded_salaries2)
encoded_train2 = sample(1:nrow(encoded_salaries2), 3003)
knn_salaries_train = encoded_salaries2[encoded_train2, ]
knn_salaries_test = encoded_salaries2[-encoded_train2, ]

knn_YTrain = knn_salaries_train$salary_in_usd
knn_XTrain = knn_salaries_train %>% select(-salary_in_usd)

knn_YTest = knn_salaries_test$salary_in_usd
```

```
knn_XTest = knn_salaries_test %>% select(-salary_in_usd)

head(knn_YTrain)
```

Training the reggresor using the encoded training set, for now we will use k=10, decide afterwards if it is worth spending time on cross-validation with this method.

```
options(max.print = 1000)
set.seed(123)
```

Calculating the Training MSE

```
pred_YTrain = knn.reg(train=knn_XTrain, test=knn_XTrain, y=knn_YTrain, k=10)
knn_mse_train <- mean((pred_YTrain$pred - knn_YTrain)^2) #2903523191
knn_mse_train
```

Calculating test MSE

```
pred.YTest = knn.reg(train=knn_XTrain, test=knn_XTest, y=knn_YTrain, k=10)
knn_mse_test <- mean((pred.YTest$pred - knn_YTest)^2) #3560557411
knn_mse_test
```

Our MSE is very high due to the curse of dimensionality. When we use OHE, we make the predictor count very high due to the amount of added variables that K-NN will take into consideration, thus affecting the overall prediction negatively.

I believe that the KNN regression method is not worth for this sort of data set. Firstly, the original data set is full of categorical variables, we were only able to fit it into the model after encoding basically everything. Due to this, our training and test MSE seems to be very high. (train mse stays, test mse lowers as k goes from 1 to 10 which is an interesting connection) Visualization is inaccurate too due to KNN regression visualizations only being able to be graphed with 1 response, and in this case the only variable I can graph on would be the work_year, which wouldn't really show anything. Due to the above reasons, I will not be spending time in tuning the lambda for said method.

Linear Regression

Before going into the code below, I need to state that due to the large amount of factor levels, we had to use the encoded dataset as when I tried running the test data from a model uding the training data, it was shown that some factor levels were found in the testing data that arent in the training data, thus leading to error messages and inability to function.

Using log transformation on the response variable to follow a normal

```
log.train <- encoded_salaries_train
log.train$salary_in_usd <- log1p(encoded_salaries_train$salary_in_usd)
```

Fitting the model

```
lmod <- lm(salary_in_usd ~ ., log.train)
summary(lmod)
```

Checking Predictions

```
options(max.print = 6)
predicted_values <- predict(lmod)
comparison_log_df <- data.frame(Actual = log.train$salary_in_usd, Predicted = predicted_values)
comparison_log_df
```

```
##        Actual Predicted
## 2463 12.38840  11.89292
## 2511 12.04356  11.89292
## 2227 12.10072  11.72904
##  [ reached 'max' / getOption("max.print") -- omitted 3000 rows ]
```

Creating plot of predicted values vs the actual values (displayed with the non log plot below)

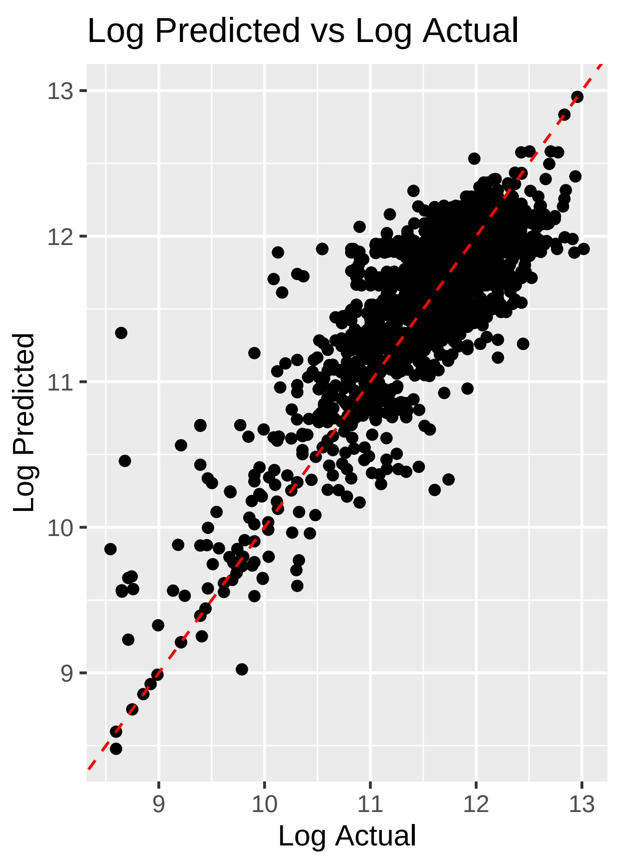
```
gg_comparison_log <- ggplot(comparison_log_df, aes(x = log.train$salary_in_usd, y = predicted_values)) +
  geom_point() +
  geom_abline(slope = 1, intercept = 0, color = "red", linetype = "dashed") +
  labs(title = "Log Predicted vs Log Actual", x = "Log Actual", y = "Log Predicted")
```

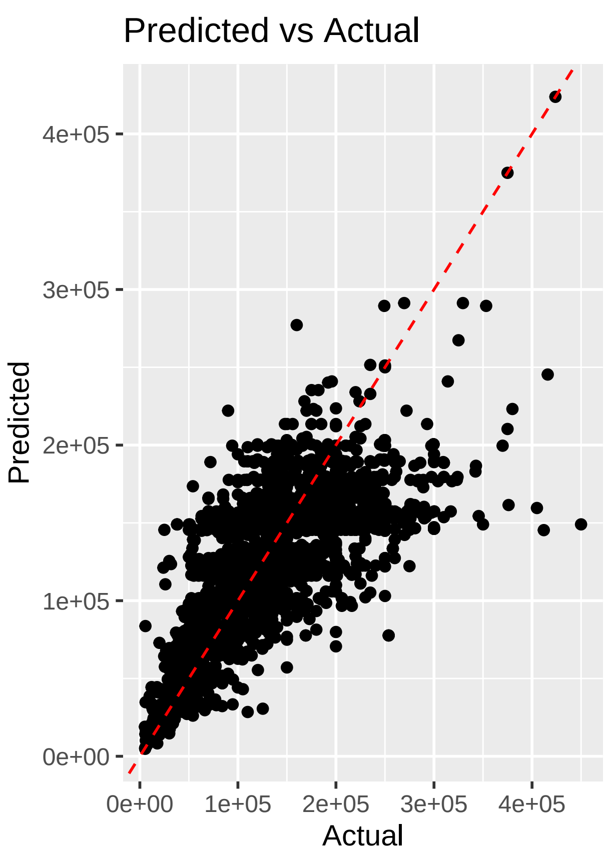
checking non log'd predictions

```
options(max.print = 6)
comparison_df <- data.frame(Actual = encoded_salaries_train$salary_in_usd, Predicted = exp(predicted_val
comparison_df
```

Displaying the 2 plots

```
grid.arrange(gg_comparison_log,gg_comparison, ncol = 2)
```





Getting R^2, Mean absolute error and Mean Squared Error for the log training response

```r
rsquared_log <- summary(lmod)$r.squared
residuals_log <- residuals(lmod)
mae_log <- mean(abs(residuals_log))
mse_log <- mean(residuals_log^2)
rsquared_log #0.6678302
mae_log #0.2639758
mse_log #0.1206688
```

MSE of the non log training response

```r
predicted_values_nonlog <- exp(predict(lmod))
residuals_nonlog <- encoded_salaries_train$salary_in_usd - predicted_values_nonlog
mse_nonlog <- mean(residuals_nonlog^2)
mse_nonlog #2110682376
```

Predicting the test data & its MSE using the model we built

```r
lmod_test_predicted <- predict(lmod, newdata = encoded_XTest)
```

```
## Warning in predict.lm(lmod, newdata = encoded_XTest): prediction from
## rank-deficient fit; attr(*, "non-estim") has doubtful cases
```

```r
mse_test <- mean((lmod_test_predicted - encoded_YTest)^2)
mse_test #23967797489
```

Overall, although we are able to fit the response variable quite well into a linear model thanks to the logarithmic transformation, the overall prediction is not very good judging from the very high MSE we get. This is probably due to the dataset mostly only having categorical predictor variables (factors with 70+ levels), making it really bad for fitting a linear model. Multicolinearity could also affect this since we had to use encoding, thus creating a lot more variables that could be collinear to each other.

Tree based methods

Preparing the non-encoded training and testing data set with Train/Test/Split

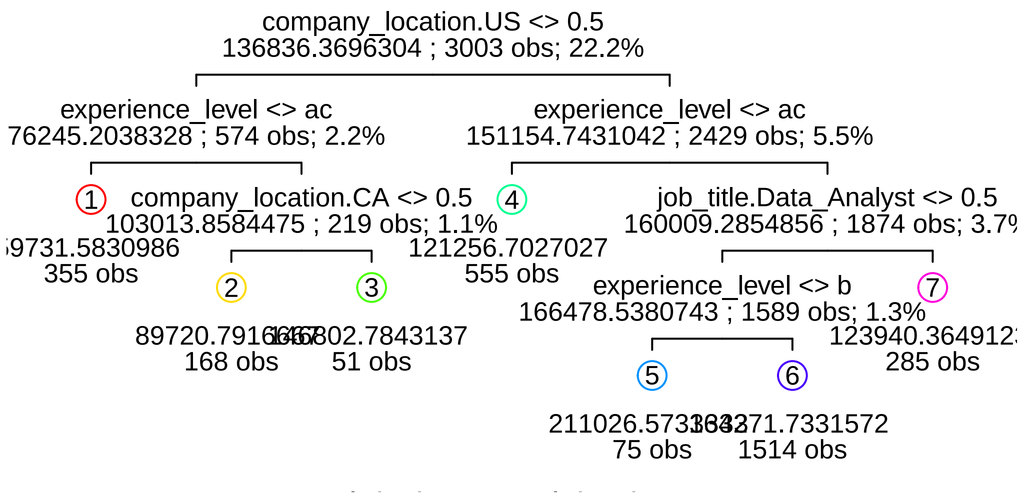
```r
set.seed(123)
salaries_count = sample(1:nrow(salaries), 3003)
salaries_train = salaries[salaries_count, ]
salaries_test = salaries[-salaries_count, ]

YTrain = salaries_train$salary_in_usd
XTrain = salaries_train %>% select(-salary_in_usd)

YTest = salaries_test$salary_in_usd
XTest = salaries_test %>% select(-salary_in_usd)
```

Decision Regression Trees

```r
tree_salaries <- tree(salary_in_usd ~ ., data = encoded_salaries_train)

tree_salaries.cv = cv.tree(tree_salaries, K=5)
best_cv = min(tree_salaries.cv$size[tree_salaries.cv$dev == min(tree_salaries.cv$dev)])
```

Seems like the best tree fitted using this R function is the one with the size of 7 (the default fitted one)

```r
draw.tree(tree_salaries, nodeinfo=TRUE, cex = 0.8)
```


company_location.US <> 0.5
136836.3696304 ; 3003 obs; 22.2%
experience_level <> ac
76245.2038328 ; 574 obs; 2.2%
experience_level <> ac
151154.7431042 ; 2429 obs; 5.5%
1
company_location.CA <> 0.5
103013.8584475 ; 219 obs; 1.1%
4
job_title.Data_Analyst <> 0.5
160009.2854856 ; 1874 obs; 3.7%
59731.5830986
355 obs
121256.7027027
555 obs
2
3
experience_level <> b
166478.5380743 ; 1589 obs; 1.3%
7
89720.7916667
168 obs
146802.7843137
51 obs
123940.364912
285 obs
5
6
211026.5733333
75 obs
163871.7331572
1514 obs

Total deviance explained = 36 %. It doesn't seem like we need to prune the tree, due to how less complex it is given the dataset with a huge amount of variables as we have right now. R seems to have already decided the important variables on its base iteration.

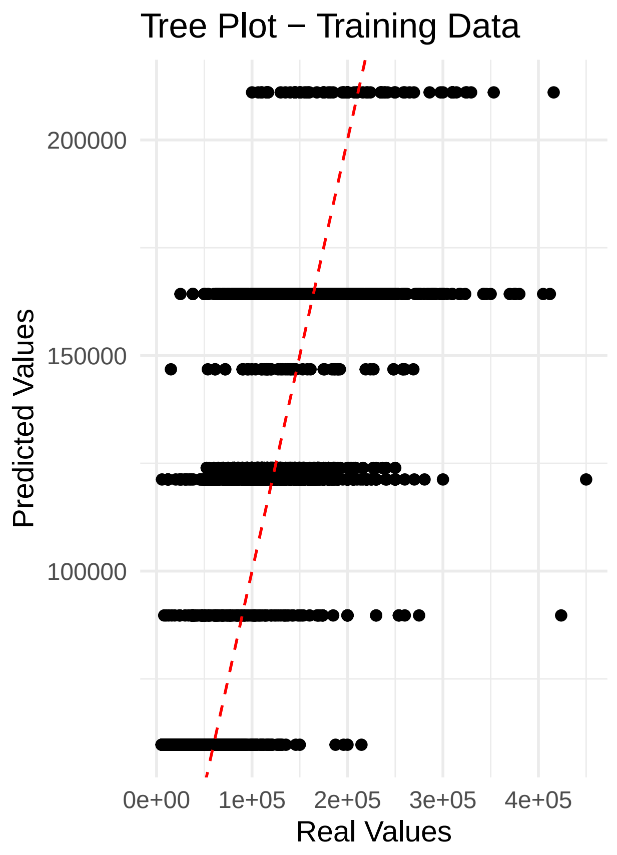
Tabling real values in salaries_train vs the predicted tree values

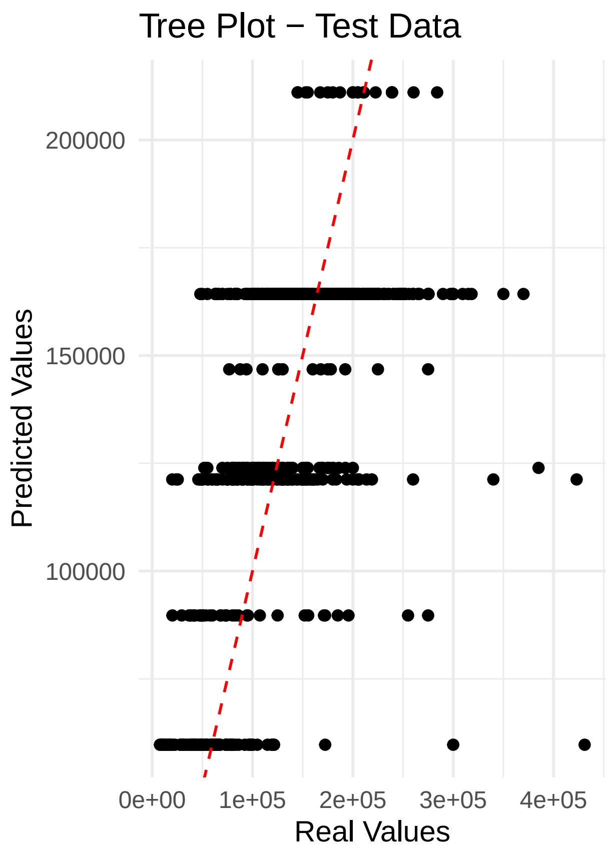
```
predicted_values1 <- predict(tree_salaries, newdata = encoded_salaries_train)

comparison_table1 <- data.frame(Real = encoded_salaries_train$salary_in_usd, Predicted = predicted_valu
```

Similarly for the training set

```
predicted_values2 <- predict(tree_salaries, newdata = encoded_salaries_test)

comparison_table2 <- data.frame(Real = encoded_salaries_test$salary_in_usd, Predicted = predicted_values
```

Visualization plot





Train and Test MSE calculation

```
residuals1 <- predicted_values1 - encoded_salaries_train$salary_in_usd
train_mse_tree <- mean(residuals1^2)
residuals2 <- predicted_values2 - encoded_salaries_test$salary_in_usd
test_mse_tree <- mean(residuals2^2)

train_mse_tree #2509462123
test_mse_tree #2818380612
```

We notice that with a regular simple decision tree, that other variables like remote_ratio, company_size and employment type were not fit into the decision. It also seems that through the visualization, factors that have a ton of levels did not split as much as expected. This causes a very small amount of possible responses which although helps with overfitting, makes the response less accurate. Still, the MSEs were way lower than those from the regression methods by 2 digits. To try and make a more complex structure, we will try other methods like random-forest and XGBoosting

Random Forest Regression Trees

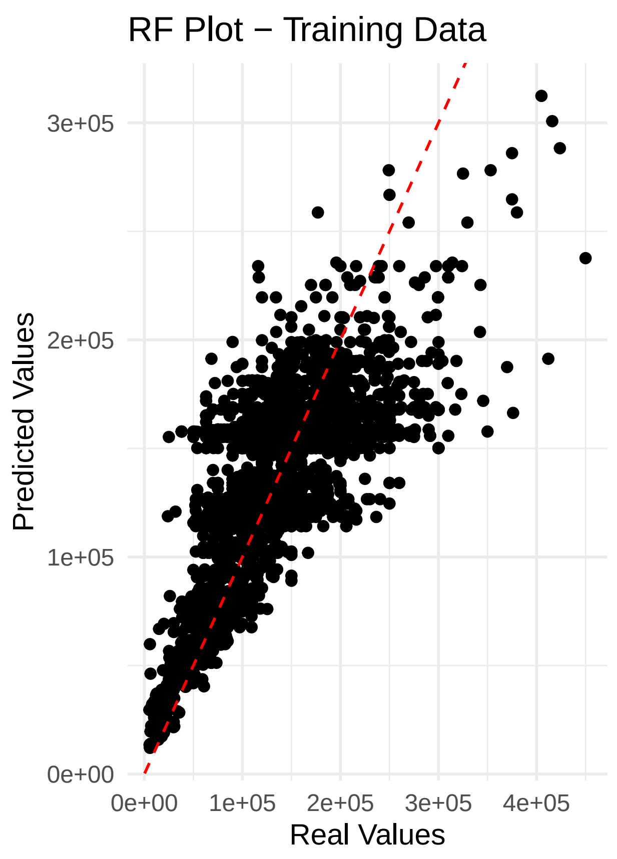
Fitting the regression tree (not specifying mtry sets it at p/3 for a regression tree)

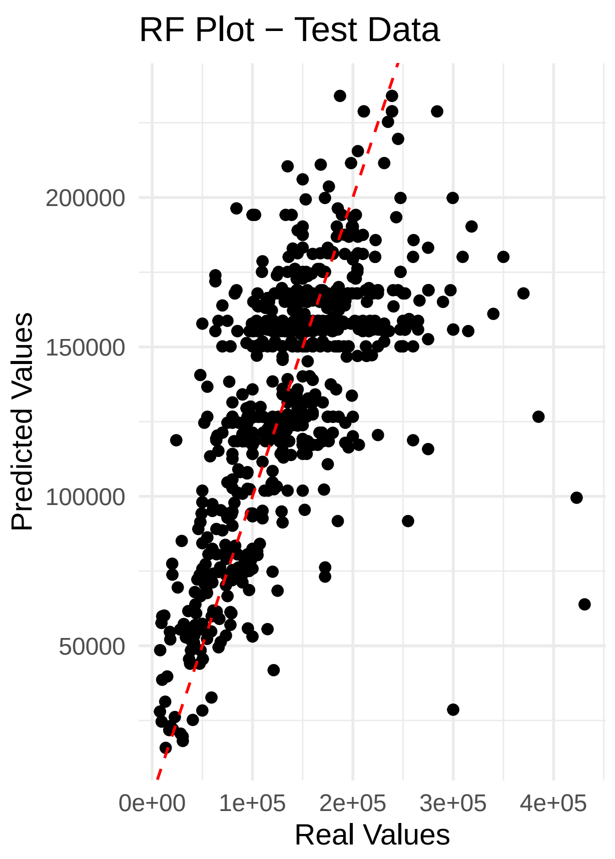
```
set.seed(123)
rf_salaries <- randomForest(salary_in_usd ~ ., data = encoded_salaries_train, importance=TRUE)
rf_salaries
```

Checking predicted values for both the train and test dataset to calculate the comparison tables which will later be used for plotting

```
predicted_values3 <- predict(rf_salaries, newdata = encoded_salaries_train)
predicted_values4 <- predict(rf_salaries, newdata = encoded_salaries_test)
comparison_table3 <- data.frame(Real = encoded_salaries_train$salary_in_usd, Predicted = predicted_valu
head(comparison_table3)
comparison_table4 <- data.frame(Real = encoded_salaries_test$salary_in_usd, Predicted = predicted_values
head(comparison_table4)
```

Visualizing the 2 fits





Judging from the plot, the training data plot suggests that we are beginning to fit the response better as opposed to the decision tree plot where it barely even seems to follow a pattern going towards the dotted line. Similarly for the Test data plot. where it seems to

Calculating the MSE

```
residuals3 <- predicted_values3 - encoded_salaries_train$salary_in_usd
residuals4 <- predicted_values4- encoded_salaries_test$salary_in_usd
train_mse_rf <- mean(residuals3^2)
test_mse_rf <- mean(residuals4^2)
train_mse_rf #1720059149
test_mse_rf #2524548390
```

The MSE is also observably lower than the previous MSE gotten from fitting a decision tree. We can still attempt to tune this even more though by trying the bagging method followed by boosting.

Bagging

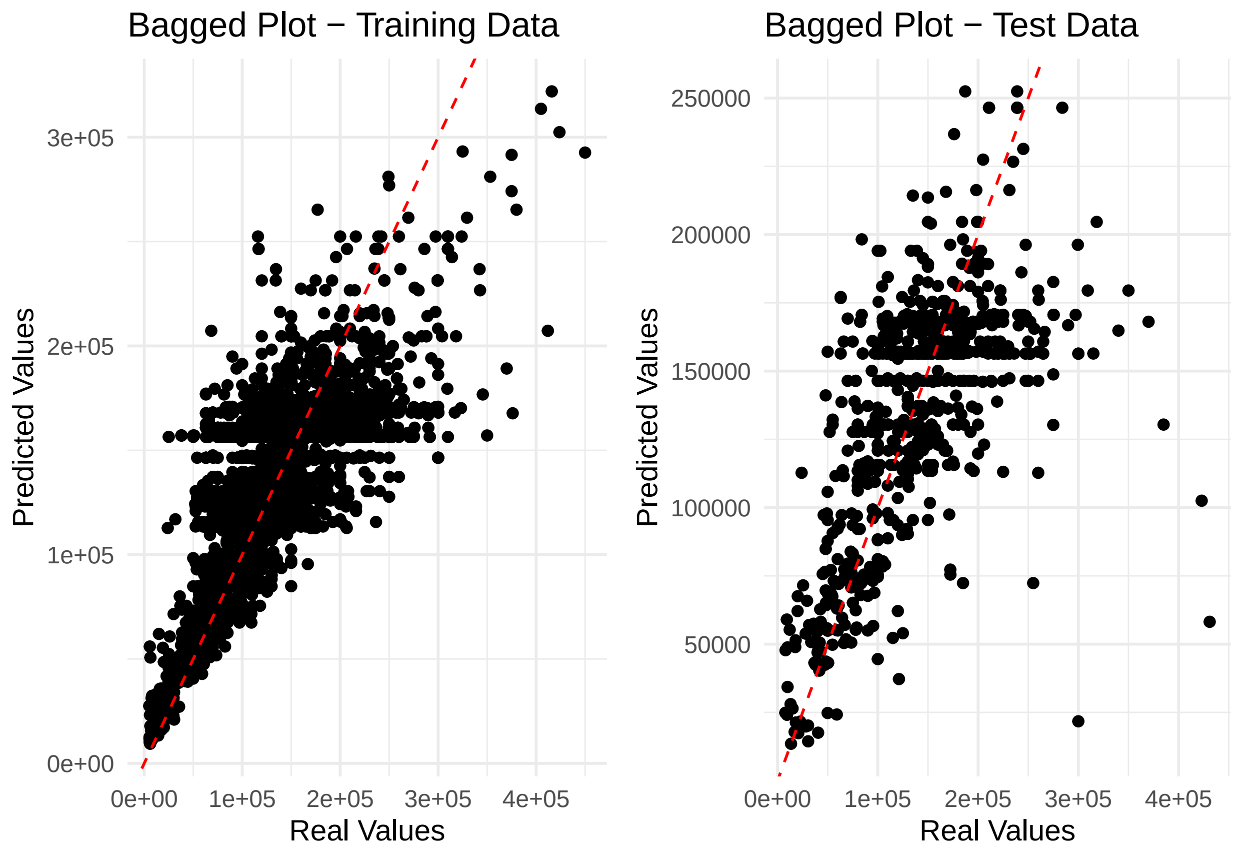
Fitting the bagged model

```
set.seed(123)
bag_salaries <- randomForest(salary_in_usd ~ ., data = encoded_salaries_train, mtry = ncol(encoded_XTra
bag_salaries
```

Predicted values for train and test datasets based on the model

```
predicted_values5 <- predict(bag_salaries, newdata = encoded_salaries_train)
predicted_values6 <- predict(bag_salaries, newdata = encoded_salaries_test)
comparison_table5 <- data.frame(Real = encoded_salaries_train$salary_in_usd, Predicted = predicted_valu
head(comparison_table5)
comparison_table6 <- data.frame(Real = encoded_salaries_test$salary_in_usd, Predicted = predicted_value
head(comparison_table6)
```

Visualizing those tables



calculating train and test MSEs

```
residuals5 <- predicted_values5 - encoded_salaries_train$salary_in_usd
residuals6 <- predicted_values6- encoded_salaries_test$salary_in_usd
train_mse_bag <- mean(residuals5^2)
test_mse_bag <- mean(residuals6^2)
train_mse_bag #1665896460
test_mse_bag #2581181189
```

The rf and bagging methods are incredible similar in terms of the visualization and MSE. We have a better training MSE but a worse testing MSE though, likely due to overfitting as we are using way more trees than with this method than the random forest method. Due to this, there is no reason to use the bagging method instead due to its way worse computational strain.

Boosting

Removing the variables with no variation

```
boost_salaries_train <- encoded_salaries_train %>%
  select(-company_location.BA, -company_location.CL, -company_location.IR, -company_location.NZ)

boost_salaries_test <- encoded_salaries_test %>%
  select(-company_location.BA, -company_location.CL, -company_location.IR, -company_location.NZ)
```

Fitting the boosted model using gbm

```
set.seed(123)
boost_salaries <- gbm(salary_in_usd ~ . ,
                      data = boost_salaries_train,
                      distribution = "gaussian",
                      n.trees = 500,
                      interaction.depth = 2)

#removed those variables as fitting the gbm with them gives a warning due to them not having variation
```

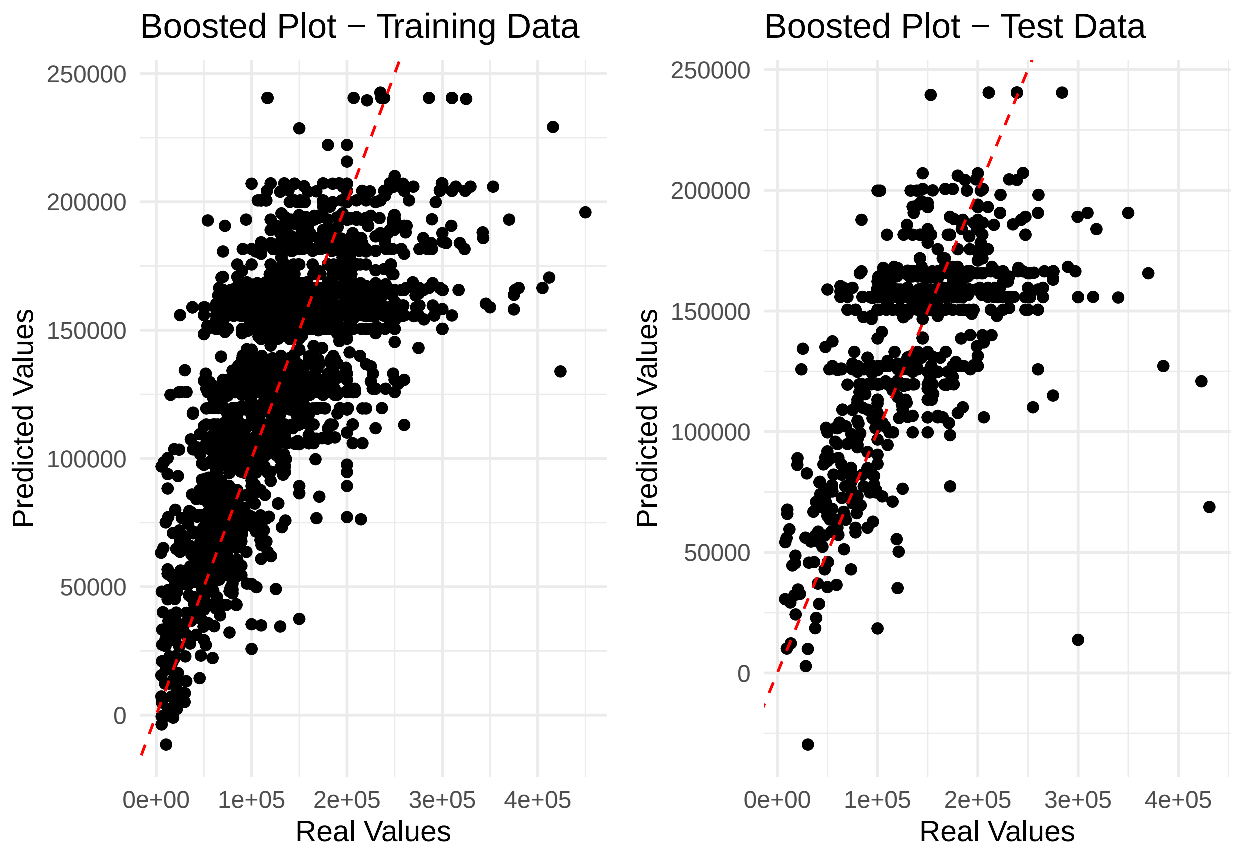
Summary to produce a relative influence plot as well as statistics

```
#summary(boost_salaries) commented out due to taking up too much space
```

Predicted values for train and test datasets based on the model

```
predicted_values7 <- predict(boost_salaries, newdata = encoded_salaries_train)
predicted_values8 <- predict(boost_salaries, newdata = encoded_salaries_test)
comparison_table7 <- data.frame(Real = encoded_salaries_train$salary_in_usd, Predicted = predicted_valu
head(comparison_table7)
comparison_table8 <- data.frame(Real = encoded_salaries_test$salary_in_usd, Predicted = predicted_values
head(comparison_table8)
```

Visualizing those tables



calculating train and test MSEs

```r
residuals7 <- predicted_values7 - encoded_salaries_train$salary_in_usd
residuals8 <- predicted_values8 - encoded_salaries_test$salary_in_usd
train_mse_boost <- mean(residuals7^2)
test_mse_boost <- mean(residuals8^2)
train_mse_boost #2166136013
test_mse_boost #2575802860
```

With 500 trees being fitted using this boosting method, we actually have a higher train MSE than the previous 2 tree methods but a slightly lower test mse than the bagging method. Once again making sense since we did not overfit the training MSE but are still less fitting than the default p/3 n.tree used in the random forest method.

Part 3: Conclusion and comparisons

Overall, we can see that tree methods are generally the better model for fitting a data set with a huge amount of levels for its categorical factors. Using the MSE as a measure of accuracy through counting its errors, we can easily see the immense amount of differences between the regression methods against the tree methods.

Tabling the MSEs

```r
options(max.print = 1000)
methods <- c("K-NN", "Linear Regression", "Decision Tree", "Random Forest", "Bagging", "Boosting")
```

```r
train_mse_combination <- c(knn_mse_train, mse_nonlog, train_mse_tree, train_mse_rf, train_mse_bag, trai
test_mse_combination <- c(knn_mse_test, mse_test, test_mse_tree, test_mse_rf, test_mse_bag, test_mse_bo

MSE_comparison_table <- data.frame(Method = methods, `Train MSE` = train_mse_combination, `Test MSE` = 

MSE_comparison_table
```

```
##              Method  Train.MSE    Test.MSE
## 1             K-NN 2903523191  3560557411
## 2 Linear Regression 2098735946 23967797489
## 3     Decision Tree 2509462123  2818380612
## 4     Random Forest 1720059149  2524548390
## 5           Bagging 1665896460  2581181189
## 6          Boosting 2166136013  2575802860
```

From these observations, we can conclude that the Random Forest tree method is the best out of all the other methods.

What can we do to further improve on predicting the salary_in_usd? Due to how the earlier EDA portion was done prior to realizing I required encoding for alot of these methods, there was no comfortable way early on to pick out outliers/influential observations which may have resulted in a higher MSE. Also due to the limitations, I was not able to experiment with XGBoosting, which is a way more sophisticated method of fitting a decision tree, likely having a lower MSE than all these other methods. Another problem was the fact that despite using one-hot encoding, R was not able to display a complete tree in which it overfits through including every other predictor. It instead disappointingly fitted a decision tree with way less branches than I expected, ruining the chances of performing a tree-pruning/cv method to fit my own custom measure for fit. It may be wise to attempt similar methods in another programming language to fix these limitations, namely Python.