Code

Similarity and Ensemble: Regression

Aarya Patil

03/21/2023

Load the Data

Hide

```
df <- read.csv ("job_profitability.csv", header = TRUE) # specifies where to load dat
a from
df <- df[,c(3, 4, 5, 10)] # specifies which columns we want
str(df) # print data frame structure
```

```
'data.frame':   14479 obs. of  4 variables:
 $ Jobs_Gross_Margin: num  -4.01 254.13 151.83 -32.15 222.7 ...
 $ Labor_Pay        : num  0 91 0 0 0 ...
 $ Labor_Burden     : num  22.2 14.9 133.2 81.2 66.3 ...
 $ Jobs_Total       : num  79.5 360 289 49 289 ...
```

Sample Training and Testing Data

Hide

```
set.seed(1234)
i <- sample(1:nrow(df), nrow(df) * 0.8, replace = FALSE) # split data into 80/20 trai
n/test
train <- df[i, ] # training data
test <- df[-i, ] # testing data
```

Explore Training Data Statistically

1. Create a summary of the data

Hide

```
summary(df) # creates a summary of the data
```

```
 Jobs_Gross_Margin     Labor_Pay         Labor_Burden       Jobs_Total
 Min.   :-11522.96   Min.   :   0.00   Min.   :   0.10   Min.   :  -50.0
 1st Qu.:   -60.41   1st Qu.:  50.12   1st Qu.:  23.37   1st Qu.:    0.0
 Median :   100.61   Median :  78.46   Median :  48.17   Median :  251.0
 Mean   :   297.54   Mean   : 108.24   Mean   :  82.52   Mean   :  599.5
 3rd Qu.:   431.64   3rd Qu.: 128.19   3rd Qu.:  96.66   3rd Qu.:  714.0
 Max.   : 19446.88   Max.   :3066.42   Max.   :5940.65   Max.   :34104.4
```

2. Display the number of rows in the data set

Hide

```
nrow(df) # generates the number of rows
```

```
[1] 14479
```

3. Display the number of missing values in each variable

Hide

```
df[df == 0.00] <- NA # replaces all values of 0.00 with NA
colSums(is.na(df)) # returns the number of missing values in each variable
```

```
Jobs_Gross_Margin         Labor_Pay      Labor_Burden        Jobs_Total
                0               348                 0              3957
```

4. Output the outliers in the data set

Hide

```
df[is.na(df)] <- 0 # replaces all values of NA with 0

# Code for finding the outliers of Jobs_Total
quartiles <- quantile(df$Jobs_Total, probs = c(.25, .75), na.rm = FALSE)
IQR <- IQR(df$Jobs_Total)

Lower <- quartiles[1] - 1.5 * IQR
Upper <- quartiles[2] + 1.5 * IQR

df_outliers <- subset(df, df$Jobs_Total <= Lower | df$Jobs_Total >= Upper) # saves ou
tliers of Jobs_Total into df_outliers

# Code for finding the outliers of Jobs_Gross_Margin
quartiles <- quantile(df$Jobs_Gross_Margin, probs = c(.25, .75), na.rm = FALSE)
IQR <- IQR(df$Jobs_Gross_Margin)

Lower <- quartiles[1] - 1.5 * IQR
Upper <- quartiles[2] + 1.5 * IQR

df_outliers <- subset(df, df$Jobs_Gross_Margin <= Lower | df$Jobs_Gross_Margin >= Upp
er) # saves outliers of Jobs_Gross_Margin into df_outliers

# Code for finding the outliers of Labor_Pay
quartiles <- quantile(df$Labor_Pay, probs = c(.25, .75), na.rm = FALSE)
IQR <- IQR(df$Labor_Pay)

Lower <- quartiles[1] - 1.5 * IQR
Upper <- quartiles[2] + 1.5 * IQR

df_outliers <- subset(df, df$Labor_Pay <= Lower | df$Labor_Pay >= Upper) # saves outl
iers of Labor_Pay into df_outliers

# Code for finding the outliers of Labor_Burden
quartiles <- quantile(df$Labor_Burden, probs = c(.25, .75), na.rm = FALSE)
IQR <- IQR(df$Labor_Burden)

Lower <- quartiles[1] - 1.5 * IQR
Upper <- quartiles[2] + 1.5 * IQR

df_outliers <- subset(df, df$Labor_Burden <= Lower | df$Labor_Burden >= Upper) # save
s outliers of Labor_Burden into df_outliers

# Output the outliers
str(df_outliers) # outputs the outliers
```

```
'data.frame':   1160 obs. of  4 variables:
 $ Jobs_Gross_Margin: num  731 6657 2091 5171 1486 ...
 $ Labor_Pay        : num  420 695 193 782 243 ...
 $ Labor_Burden     : num  208 387 208 408 261 ...
 $ Jobs_Total       : num  1494 10297 3156 7056 2617 ...
```

Hide

```
summary(df_outliers) # outputs a summary
```

```
 Jobs_Gross_Margin    Labor_Pay       Labor_Burden      Jobs_Total
 Min.   :-11523.0   Min.   :   0.0   Min.   : 206.8   Min.   :  -50.0
 1st Qu.:   166.2   1st Qu.: 206.6   1st Qu.: 241.5   1st Qu.:  850.5
 Median :   780.7   Median : 259.7   Median : 293.9   Median : 1701.7
 Mean   :   934.3   Mean   : 328.8   Mean   : 373.2   Mean   : 2176.9
 3rd Qu.:  1578.7   3rd Qu.: 361.9   3rd Qu.: 390.9   3rd Qu.: 2928.8
 Max.   : 19446.9   Max.   :3066.4   Max.   :5940.6   Max.   :34104.4
```

5. Find correlations between columns in the data (Labor_Pay and Labor_Burden, Jobs_Gross_Margin and Jobs_Total)

Hide

```
cor.test(df$Labor_Pay, df$Labor_Burden, use = "complete") # finds correlations betwee
n Labor_Pay and Labor_Burden
```

```

    Pearson's product-moment correlation

data:  df$Labor_Pay and df$Labor_Burden
t = 138.69, df = 14477, p-value < 2.2e-16
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.7482670 0.7622593
sample estimates:
      cor
0.7553492
```

Hide

```
cor.test(df$Jobs_Gross_Margin, df$Jobs_Total, use = "complete") # finds correlations
between Jobs_Gross_Margin and Jobs_Total
```

```
    Pearson's product-moment correlation

data:  df$Jobs_Gross_Margin and df$Jobs_Total
t = 227.62, df = 14477, p-value < 2.2e-16
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.8804737 0.8875900
sample estimates:
      cor
0.8840831
```

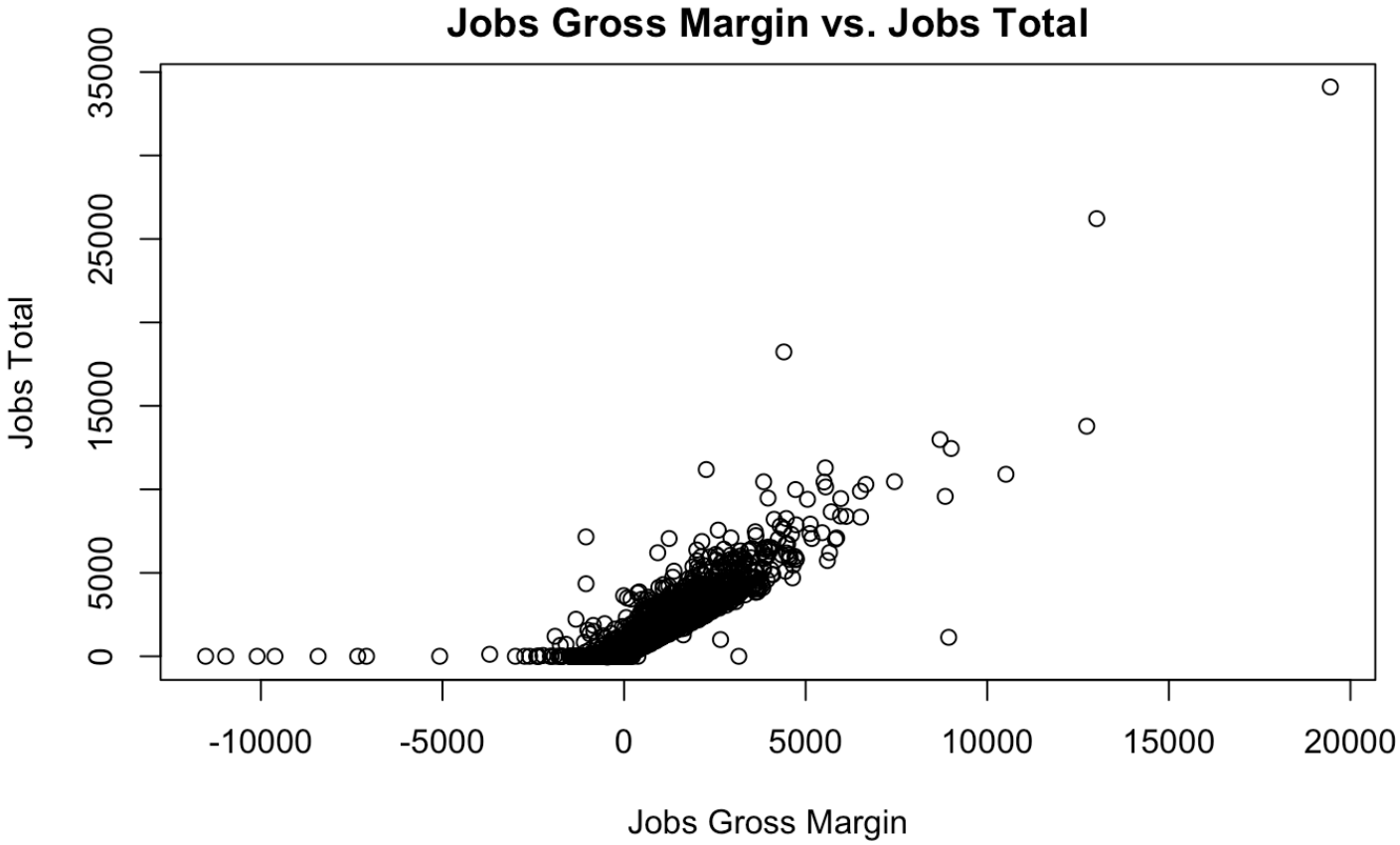
Explore Training Data Graphically

Hide

```
# Graph 1
plot(df$Labor_Pay, df$Labor_Burden, main = "Labor Pay vs. Labor Burden", xlab = "Labo
r Pay", ylab = "Labor Burden")
```

Hide

```
# Graph 2
plot(df$Jobs_Gross_Margin, df$Jobs_Total, main = "Jobs Gross Margin vs. Jobs Total",
xlab = "Jobs Gross Margin", ylab = "Jobs Total")
```



Perform Linear Regression

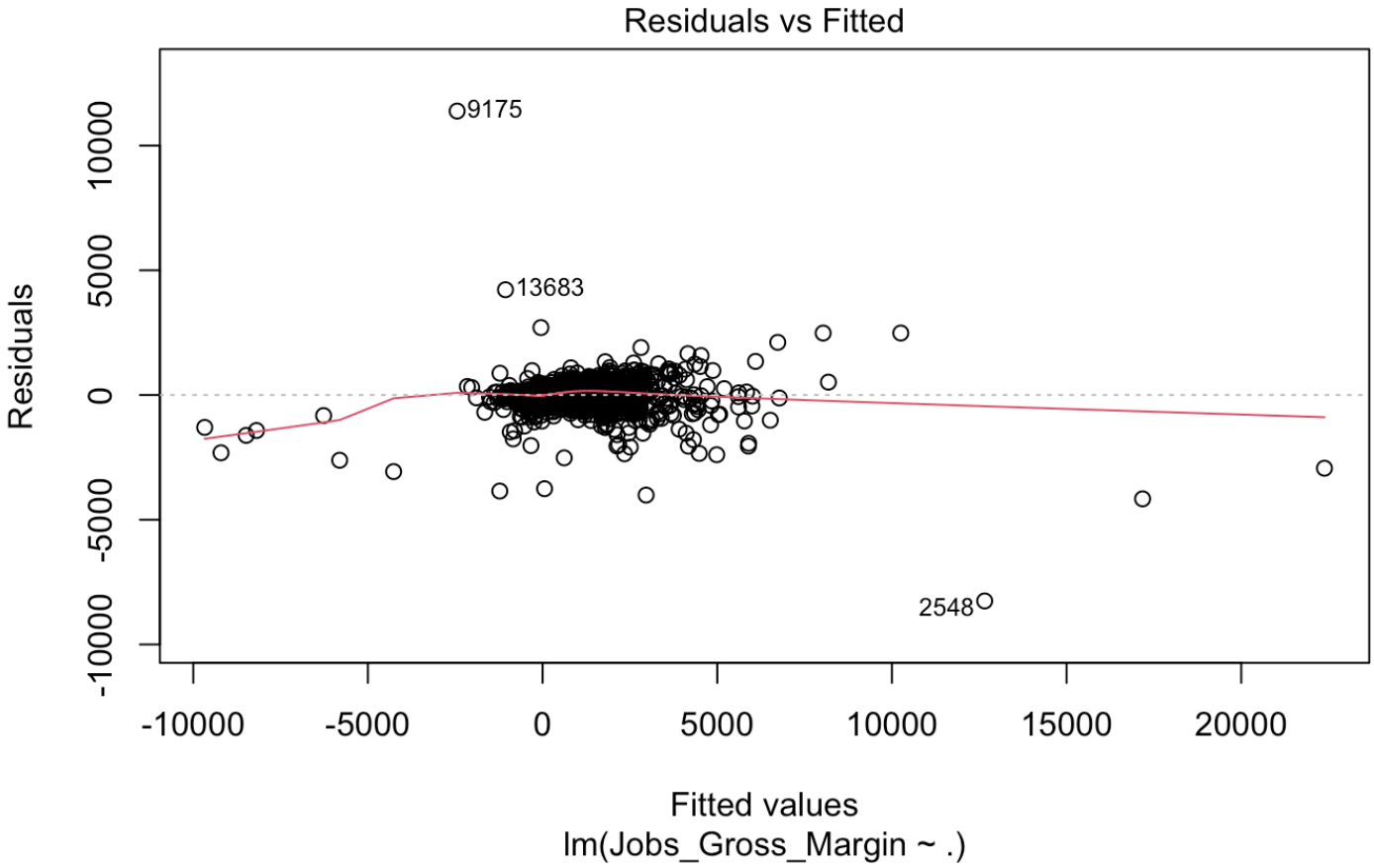
Hide

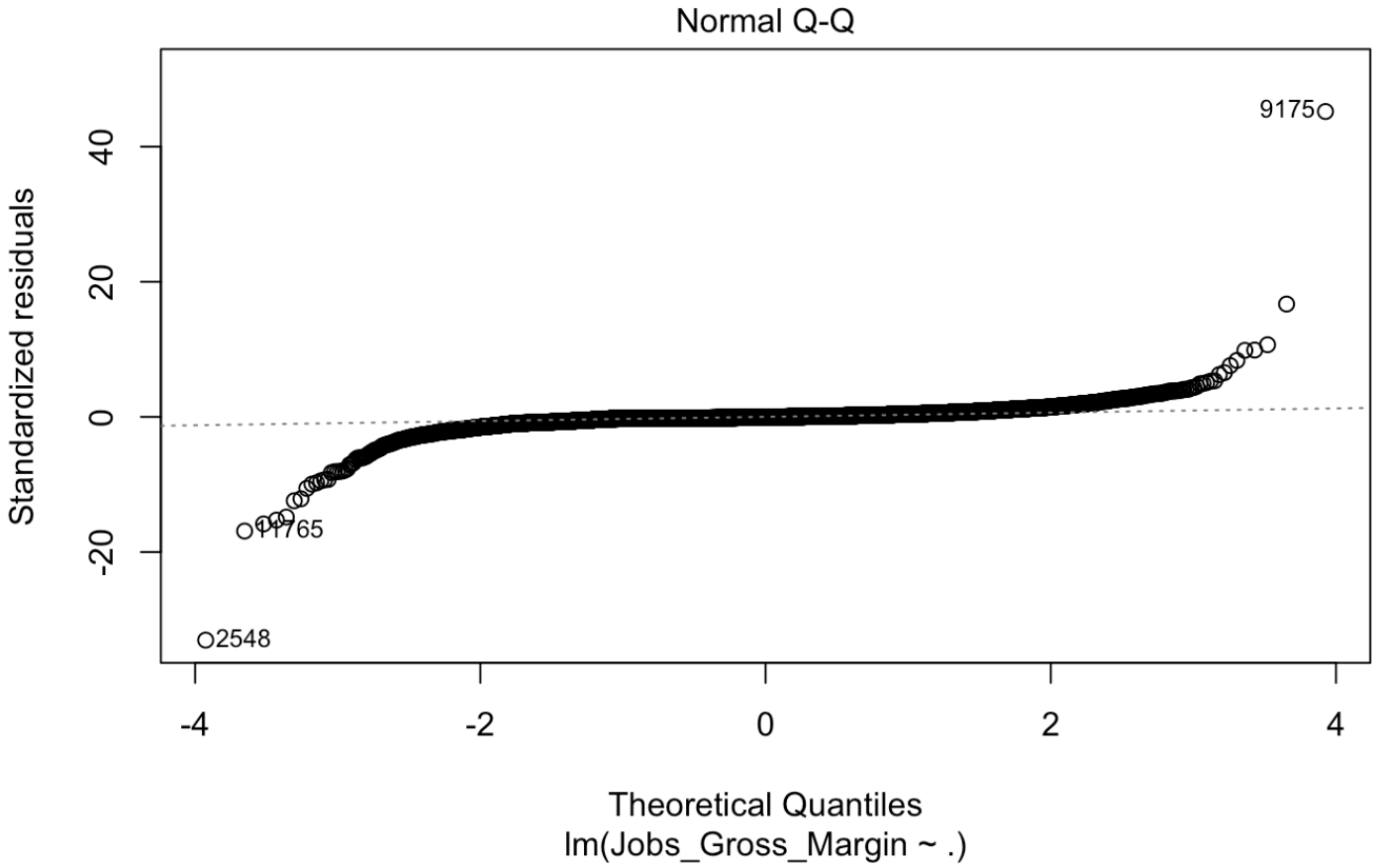
```
lm1 <- lm(Jobs_Gross_Margin ~ ., data = train) # builds linear regression model with
multiple predictors, Jobs_Total and Labor_Burden, on Jobs_Gross_Margin
summary(lm1) # shows the linear regression model summary
```

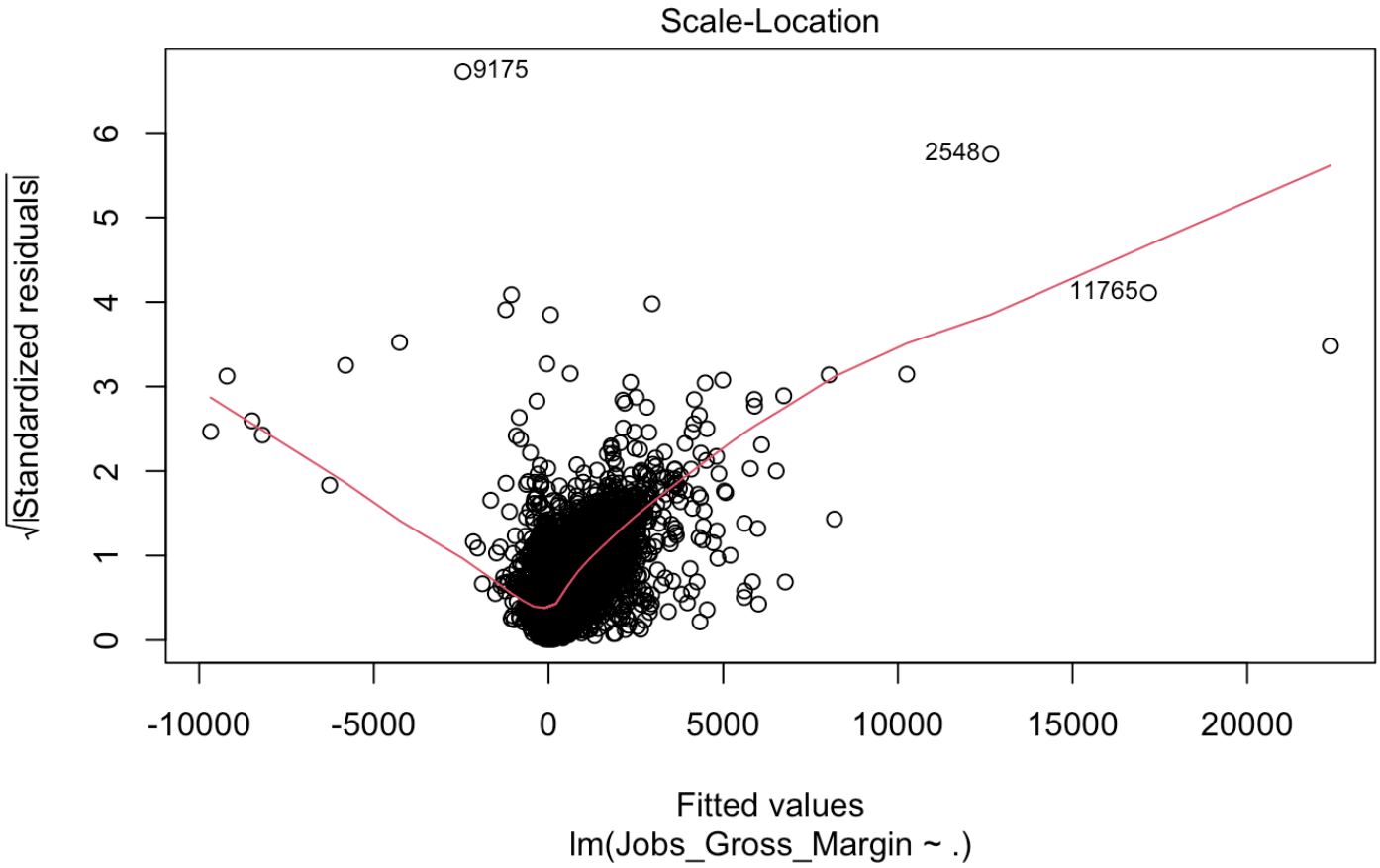
```
Call:
lm(formula = Jobs_Gross_Margin ~ ., data = train)

Residuals:
    Min      1Q  Median      3Q     Max 
-8255.9   -50.5    -9.3    56.1 11383.4 

Coefficients:
             Estimate Std. Error t value Pr(>|t|)    
(Intercept)  57.291767   3.195990   17.93   <2e-16 ***
Labor_Pay    -0.838964   0.033567  -24.99   <2e-16 ***
Labor_Burden -1.575908   0.024559  -64.17   <2e-16 ***
Jobs_Total    0.768291   0.002754  278.96   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 253.5 on 11579 degrees of freedom
Multiple R-squared:  0.8888,    Adjusted R-squared:  0.8888 
F-statistic: 3.085e+04 on 3 and 11579 DF,  p-value: < 2.2e-16
```

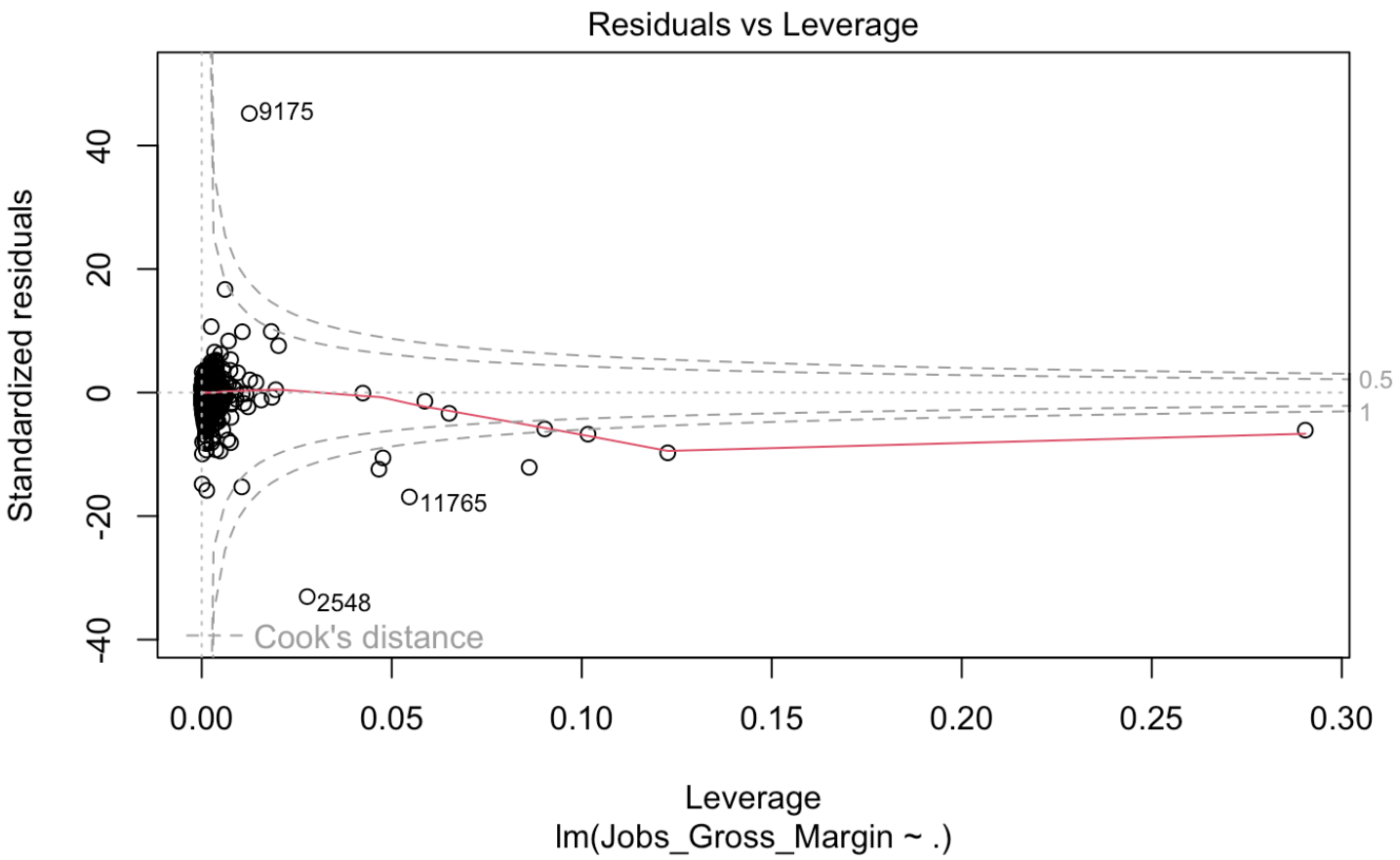
Hide

```
plot(lm1) # displays the linear regression model plots
```


Residuals vs Fitted
9175
13683
2548
Residuals
10000
5000
0
-5000
-10000
-10000
-5000
0
5000
10000
15000
20000
Fitted values
lm(Jobs_Gross_Margin ~ .)


Normal Q-Q
Standardized residuals
40
20
0
-20
9175
11765
2548
-4
-2
0
2
4
Theoretical Quantiles
lm(Jobs_Gross_Margin ~ .)


Scale-Location
9175
2548
11765
√|Standardized residuals|
0
1
2
3
4
5
6
-10000
-5000
0
5000
10000
15000
20000
Fitted values
lm(Jobs_Gross_Margin ~ .)



Hide

```
# Evaluate
pred <- predict(lm1, newdata = test)
cor_lm <- cor(pred, test$Jobs_Gross_Margin)
mse_lm <- mean((pred - test$Jobs_Gross_Margin)^2)

# Print out values
print(paste('cor:', cor_lm))
```

```
[1] "cor: 0.959206463271686"
```

Hide

```
print(paste('mse:', mse_lm))
```

```
[1] "mse: 36101.9618087527"
```

Hide

```
print(paste('rmse:', sqrt(mse_lm)))
```

```
[1] "rmse: 190.005162584475"
```

Perform kNN Regression

Do initial kNN regression

Hide

```
# Download library
library(caret)

# Fit the model
fit <- knnreg(train[, 2:4], train[, 1], k = 3)

# Evaluate
pred2 <- predict(fit, test[, 2:4])
cor_knn1 <- cor(pred2, test$Jobs_Gross_Margin)
mse_knn1 <- mean((pred2 - test$Jobs_Gross_Margin)^2)

# Print out values
print(paste('cor:', cor_knn1))
```

```
[1] "cor: 0.949799349568579"
```

Hide

```
print(paste('mse:', mse_knn1))
```

```
[1] "mse: 43812.1511090969"
```

Hide

```
print(paste('rmse:', sqrt(mse_knn1)))
```

```
[1] "rmse: 209.313523473991"
```

Scale the data

Hide

```
# Scale data to find the best k value
train_scaled <- train[, 2:4]
means <- sapply(train_scaled, mean)
stdvs <- sapply(train_scaled, sd)
train_scaled <- scale(train_scaled, center = means, scale = stdvs)
test_scaled <- scale(test[, 2:4], center = means, scale = stdvs)

# Test change in values after scaling data
# Fit the model
fit <- knnreg(train_scaled, train$Jobs_Gross_Margin, k = 3)
# Evaluate
pred3 <- predict(fit, test_scaled)
cor_knn2 <- cor(pred3, test$Jobs_Gross_Margin)
mse_knn2 <- mean((pred3 - test$Jobs_Gross_Margin)^2)
print(paste('cor:', cor_knn2))
```

```
[1] "cor: 0.94888742604617"
```

Hide

```
print(paste('mse:', mse_knn2))
```

```
[1] "mse: 44526.6421906298"
```

Hide

```
print(paste('rmse:', sqrt(mse_knn2)))
```

```
[1] "rmse: 211.0133696964"
```

Plot the correlation to find the best k value

Hide

```
cor_k <- rep(0, 20)
mse_k <- rep(0, 20)
i <- 1

# Find cor and mse values for all k's from 1-39
for (k in seq(1, 39, 1)){
  fit_k <- knnreg(train_scaled,train$Jobs_Gross_Margin, k = k)
  pred_k <- predict(fit_k, test_scaled)
  cor_k[i] <- cor(pred_k, test$Jobs_Gross_Margin)
  mse_k[i] <- mean((pred_k - test$Jobs_Gross_Margin)^2)
  print(paste("k = ", k, cor_k[i], mse_k[i]))
  i <- i + 1
}
```

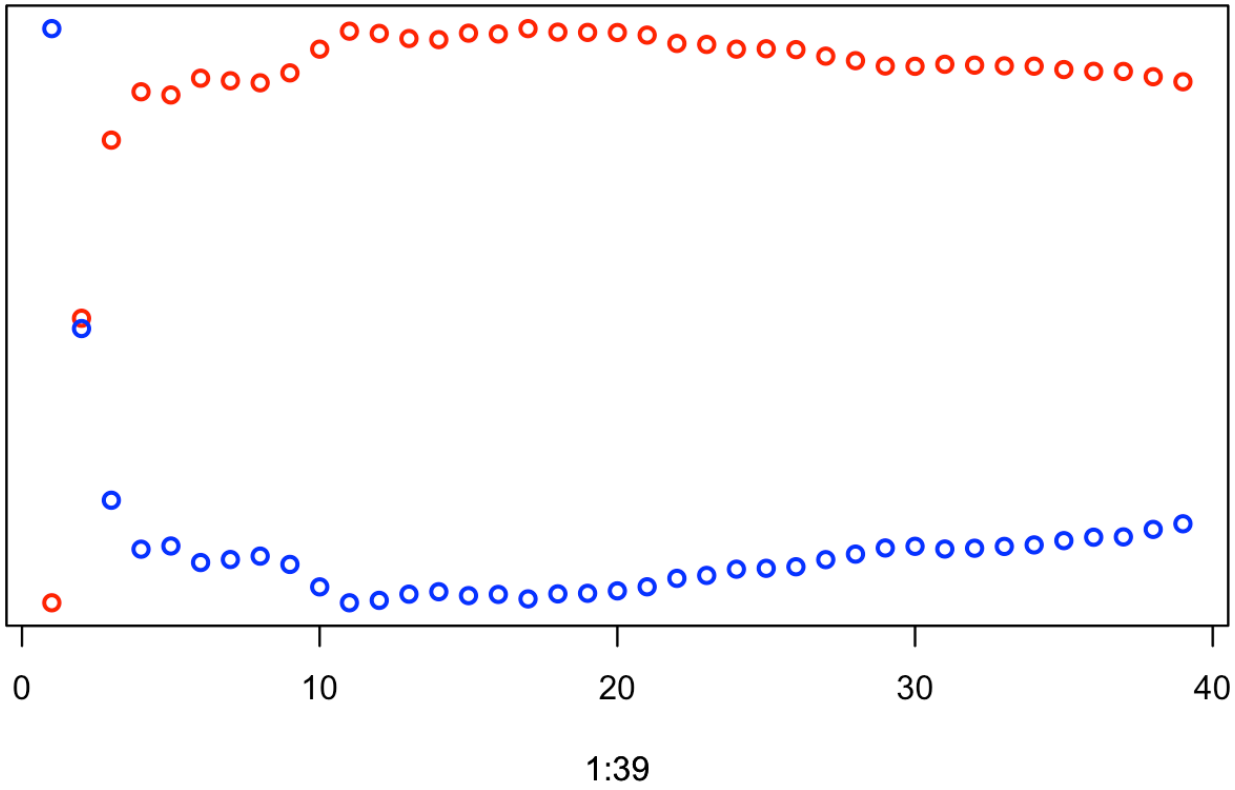
```
[1] "k =  1 0.914432401717133 75031.6581164278"
[1] "k =  2 0.935621657689153 55630.5140408005"
[1] "k =  3 0.94888742604617 44526.6421906298"
[1] "k =  4 0.952486064898813 41358.4751245161"
[1] "k =  5 0.952248849788457 41564.6587804882"
[1] "k =  6 0.953496947177131 40496.661097757"
[1] "k =  7 0.953309715670659 40692.1285460223"
[1] "k =  8 0.953156773410732 40909.7348446066"
[1] "k =  9 0.95389419337434 40374.0799654498"
[1] "k =  10 0.955663007864068 38919.4962736549"
[1] "k =  11 0.957000918797557 37892.6488154394"
[1] "k =  12 0.956836649848019 38049.3013389117"
[1] "k =  13 0.956454882738784 38450.262654351"
[1] "k =  14 0.956371302335426 38608.1579318218"
[1] "k =  15 0.956853777466828 38355.6825716498"
[1] "k =  16 0.956791040684961 38434.3227626724"
[1] "k =  17 0.957191866567773 38144.5006673951"
[1] "k =  18 0.956930753263043 38469.7697806985"
[1] "k =  19 0.956908410512684 38505.4071898562"
[1] "k =  20 0.956902179650106 38662.5505176019"
[1] "k =  21 0.956707262019351 38924.5985663727"
[1] "k =  22 0.95608821678382 39471.8969084156"
[1] "k =  23 0.956014178954004 39661.614608335"
[1] "k =  24 0.955664123663981 40065.5511307952"
[1] "k =  25 0.955691559449311 40121.3830442156"
[1] "k =  26 0.955626974032989 40220.8243791265"
[1] "k =  27 0.955140260553024 40683.7983995687"
[1] "k =  28 0.954804828032792 41039.5769561952"
[1] "k =  29 0.954408580536707 41436.3214254921"
[1] "k =  30 0.954385262623746 41544.8907338456"
[1] "k =  31 0.954539078596556 41368.8995530087"
[1] "k =  32 0.95447132179101 41421.0153402227"
[1] "k =  33 0.954419522005573 41538.0927123375"
[1] "k =  34 0.954394726850799 41635.3198565272"
[1] "k =  35 0.954142175137485 41914.4564722525"
[1] "k =  36 0.954012701873492 42132.5760480365"
[1] "k =  37 0.954002481148275 42153.8601048853"
[1] "k =  38 0.953608420843433 42638.5384826208"
[1] "k =  39 0.95323399207536 43002.7780514644"
```

Hide

```
# Plot the graph
plot(1:39, cor_k, lwd = 2, col = 'red', ylab = "", yaxt = 'n')
par(new = TRUE)
```

Hide

```
plot(1:39, mse_k, lwd = 2, col = 'blue', labels = FALSE, ylab = "", yaxt = 'n')
```



Hide

```
# Print out the best k value
cat("\nBest k value: ")
```

```
Best k value:
```

Hide

```
cat(max(which.min(mse_k), which.max(cor_k)))
```

```
17
```

Perform kNN regression using the found k value

Hide

```
# Fit the model
fit_17 <- knnreg(train_scaled, train$Jobs_Gross_Margin, k = 17)

# Evaluate
predictions_17 <- predict(fit_17, test_scaled)
cor_knn17 <- cor(predictions_17, test$Jobs_Gross_Margin)
mse_knn17 <- mean((predictions_17 - test$Jobs_Gross_Margin)^2)

# Print out values
print(paste('cor:', cor_knn17))
```

```
[1] "cor: 0.957191866567773"
```

Hide

```
print(paste('mse:', mse_knn17))
```

```
[1] "mse: 38144.5006673951"
```

Hide

```
print(paste('rmse:', sqrt(mse_knn17)))
```

```
[1] "rmse: 195.306171606007"
```

Using the scaled data with the best k value gives us a much better correlation value.

Perform Decision Tree Regression

Do initial decision tree regression

Hide

```
# Download libraries
library(tree)
library(MASS)

# Build tree
tree1 <- tree(Jobs_Gross_Margin~., data=train)
summary(tree1)
```

```
Regression tree:
tree(formula = Jobs_Gross_Margin ~ ., data = train)
Number of terminal nodes:  9
Residual mean deviance:  98960 = 1.145e+09 / 11570
Distribution of residuals:
     Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
-7260.000   -87.730    -8.877     0.000   111.000  8209.000
```

Hide

```
# Evaluate tree
pred <- predict(tree1, newdata=test)
cor_tree <- cor(pred, test$Jobs_Gross_Margin)
mse_tree <- mean((pred-test$Jobs_Gross_Margin)^2)

# Print out values
print(paste('cor:', cor_tree))
```

```
[1] "cor: 0.914753236805949"
```

Hide

```
print(paste('mse:', mse_tree))
```

```
[1] "mse: 72983.9813219516"
```

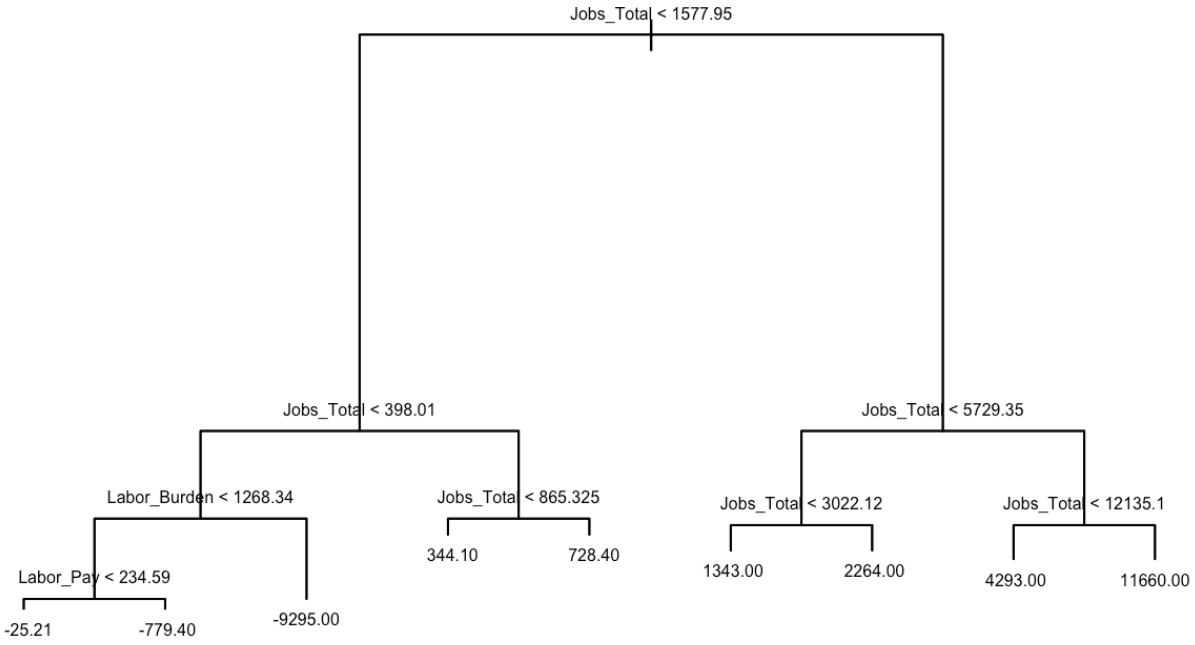
Hide

```
print(paste("rmse :", sqrt(mse_tree)))
```

```
[1] "rmse : 270.155476202041"
```

Hide

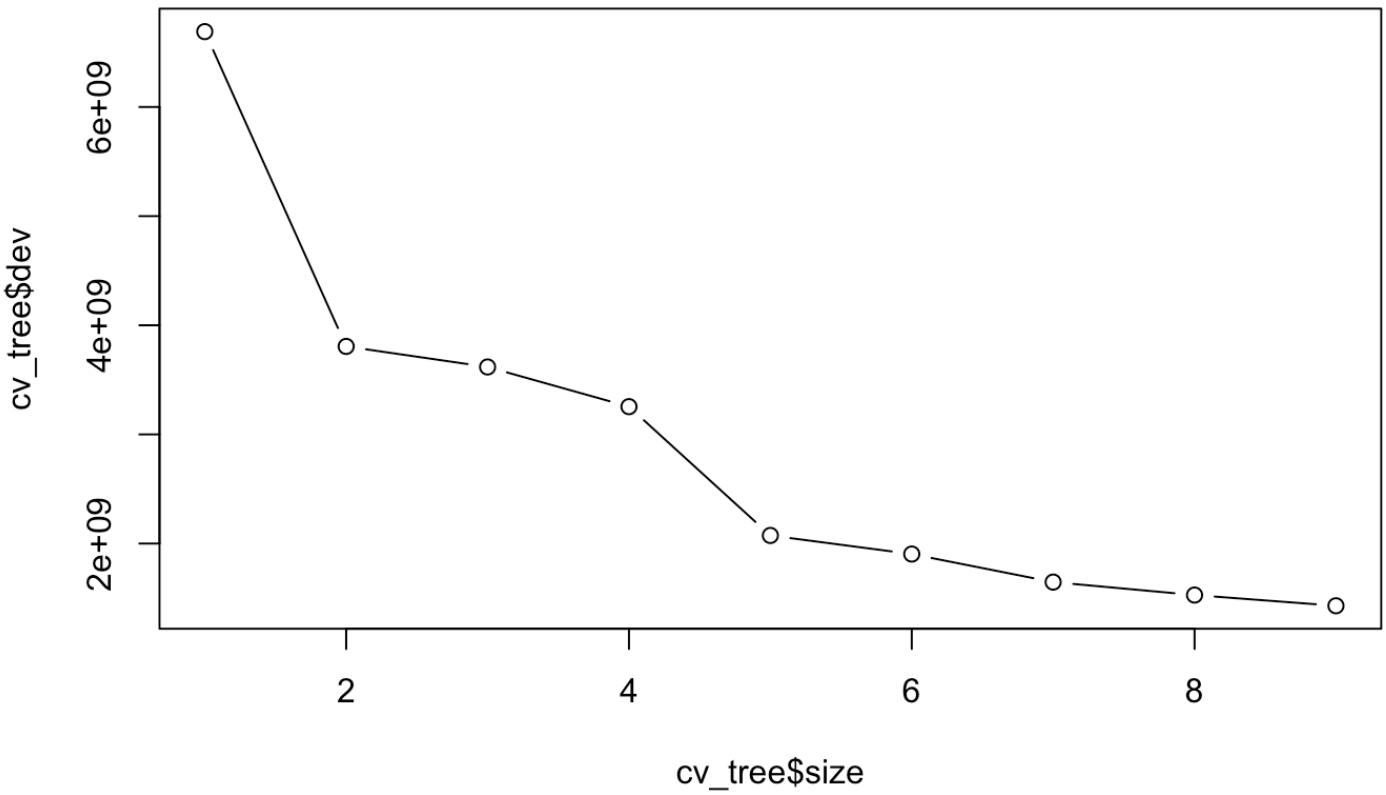
```
# Plot tree
plot(tree1)
text(tree1, cex = 0.5, pretty = 0)
```



Do cross validation

Hide

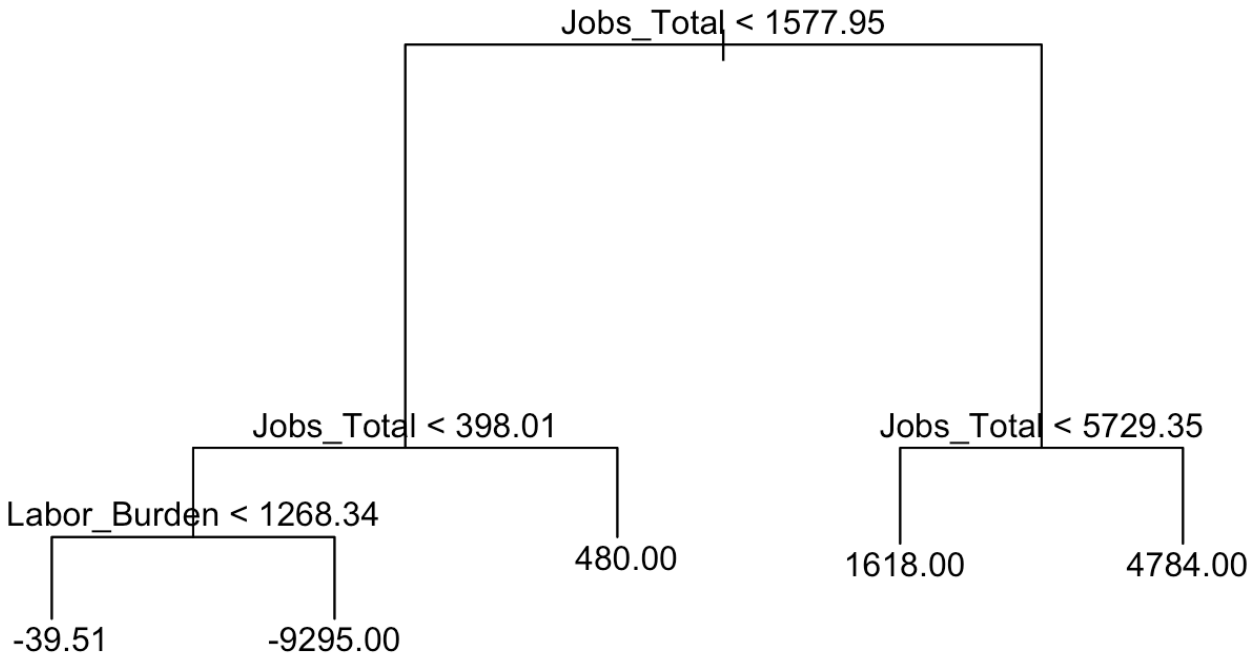
```
cv_tree <- cv.tree(tree1)
plot(cv_tree$size, cv_tree$dev, type = 'b')
```



Prune the tree

Hide

```
tree_pruned <- prune.tree(tree1, best = 5)
plot(tree_pruned)
text(tree_pruned, pretty = 0)
```



Run prediction on the pruned tree

Hide

```
# Evaluate
pred_pruned <- predict(tree_pruned, newdata=test)
cor_pruned <- cor(pred_pruned, test$Jobs_Gross_Margin)
mse_pruned <- mean((pred_pruned-test$Jobs_Gross_Margin)^2)

# Print out values
print(paste('cor:', cor_pruned))
```

```
[1] "cor: 0.859246727417677"
```

Hide

```
print(paste('mse:', mse_pruned))
```

```
[1] "mse: 116650.137250333"
```

Hide

```
print(paste('rmse:', sqrt(mse_pruned)))
```

```
[1] "rmse: 341.54082808697"
```

The cor is now 0.858, slightly lower than the unpruned tree. The rmse is 314.5, much higher than the unpruned tree. In this case pruning did not improve results on the test data but the tree is simpler and easier to interpret.

Random Forest

Hide

```
library(randomForest)
set.seed(1234)
rf <- randomForest(Jobs_Gross_Margin~., data = train, importance = TRUE)
rf
```

```

Call:
 randomForest(formula = Jobs_Gross_Margin ~ ., data = train, importance = TRUE)
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 1

          Mean of squared residuals: 85378.09
                    % Var explained: 85.22
```

Run prediction on the random forest

Hide

```
# Evaluate
pred_rf <- predict(rf, newdata = test)
cor_rf <- cor(pred_rf, test$Jobs_Gross_Margin)
mse_rf <- mean((pred_rf-test$Jobs_Gross_Margin)^2)

# Print out values
print(paste('cor:', cor_rf))
```

```
[1] "cor: 0.956297620429714"
```

Hide

```
print(paste('mse:', mse_rf))
```

```
[1] "mse: 38520.938978973"
```

Hide

```
print(paste('rmse:', sqrt(mse_rf)))
```

```
[1] "rmse: 196.267518909709"
```

The correlation is now much higher and the rmse is almost half.

Bagging

Hide

```
bag <- randomForest(Jobs_Gross_Margin~., data=train, mtry = 3)
bag
```

```

Call:
 randomForest(formula = Jobs_Gross_Margin ~ ., data = train, mtry = 3)
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 3

          Mean of squared residuals: 80956.55
                    % Var explained: 85.98
```

Run prediction on bagging

Hide

```
# Evaluate
pred_bag <- predict(bag, newdata=test)
cor_bag <- cor(pred_bag, test$Jobs_Gross_Margin)
mse_bag <- mean((pred_bag-test$Jobs_Gross_Margin)^2)

# Print out values
print(paste('cor:', cor_bag))
```

```
[1] "cor: 0.957401702050157"
```

Hide

```
print(paste('mse:', mse_bag))
```

```
[1] "mse: 37216.2941883724"
```

Hide

```
print(paste('rmse:', sqrt(mse_bag)))
```

```
[1] "rmse: 192.915251310964"
```

The results for bagging are a little better than the results for the random forest.

Result Comparison

The results for the three regression algorithms were as followed:

1. Linear regression
    - cor: 0.959206463271686
    - mse: 36101.9618087527
    - rmse: 190.005162584475
2. kNN regression
    - cor: 0.957191866567773
    - mse: 38144.5006673951
    - rmse: 195.306171606007
3. Decision tree regression
    - Pruned tree:
        - cor: 0.859246727417677
        - mse: 116650.137250333
        - rmse: 341.54082808697
    - Random forest:
        - corr: 0.956297620429714
        - mse: 38520.938978973
        - rmse: 196.267518909709
    - Bagging:
        - cor: 0.957350111251879
        - mse: 37287.5106863535
        - rmse: 193.09974284383v

Overall, the linear regression algorithm had the best results in regards to both correlation and rmse values. Within the kNN regression, the values slightly decreased after scaling the data but once I found the best k value and used it alongside the scaled data, the values improved significantly. Within the decision tree regression, the pruned tree had the worst values, then the random forest, and finally bagging, which had the best values. All of the algorithms performed fairly similarly and the final values differed by only a small margin.

Analysis

Although the linear regression algorithm resulted in better values than kNN regression algorithm, the difference in correlation between the two was fairly marginal. When it comes to data that is all over the place or really complicated, or when there's a clear non-linear pattern in the data, kNN usually outperforms linear regression. Our data was probably not too complex or was more linear than non-linear since the values were better in linear regression.

The decision tree regression initally did much worse than the other 2 algorithms, but after bagging, the results were even better than the kNN regression algorithm. One of the most common problems that decision trees face is overfitting. It usually leads to issues that result in lower values for correlation and rmse. When we used random forest and bagging, the values shot up. This is because they are designed to fix overfitting. We can see this directly in our results.