# R Notebook

Code

Hello! Below is my work/answers to the Week 4 HW assignment for ISYE 6501.

In my code / work / ideas section I placed various sources for a few ideas I implemented and concepts I discussed within my answers. Those are linked within my work.

I also received assistance from ChatGPT, an AI language model developed by OpenAI, to understand and implement different functions (hyper param tunning for tree models, PCA, oslrr, ) as well as certain loops and data manipulations for my analysis. This is my acknowledgement that I have used AI tools to assist me on this homework.

## Question 9.1

Using the same crime data set uscrime.txt as in Question 8.2, apply Principal Component Analysis and then create a regression model using the first few principal components. Specify your new model in terms of the original variables (not the principal components), and compare its quality to that of your solution to Question 8.2. You can use the R function prcomp for PCA. (Note that to first scale the data, you can include scale. = TRUE to scale as part of the PCA function. Don't forget that, to make a prediction for the new city, you'll need to unscale the coefficients (i.e., do the scaling calculation in reverse)!)

### Q9.1 ANSWER

My final model used a transformation with PCA 1-7 on our data to create the model below:

The linear regression model is represented by the following equation:

$$\begin{aligned} Y = \beta_0 &+ \beta_1 \cdot M + \beta_2 \cdot So + \beta_3 \cdot Ed + \beta_4 \cdot Po1 + \beta_5 \cdot Po2 \\ &+ \beta_6 \cdot LF + \beta_7 \cdot M.F + \beta_8 \cdot Pop + \beta_9 \cdot NW + \beta_{10} \cdot U1 \\ &+ \beta_{11} \cdot U2 + \beta_{12} \cdot Wealth + \beta_{13} \cdot Ineq + \beta_{14} \cdot Prob \\ &+ \beta_{15} \cdot Time + \epsilon \end{aligned}$$

where:

- $\beta_0$ (Intercept): -5498.458
- $\beta_1$ (M): 55.237
- $\beta_2$ (So): 139.757
- $\beta_3$ (Ed): -6.804
- $\beta_4$ (Po1): 44.586
- $\beta_5$ (Po2): 46.424
- $\beta_6$ (LF): 673.381
- $\beta_7$ (M.F): 44.403
- $\beta_8$ (Pop): 0.960
- $\beta_9$ (NW): 5.685
- $\beta_{10}$ (U1): -1027.735
- $\beta_{11}$ (U2): 24.416
- $\beta_{12}$ (Wealth): 0.029
- $\beta_{13}$ (Ineq): 12.451
- $\beta_{14}$ (Prob): -5170.569
- $\beta_{15}$ (Time): -2.215

The model predicts the response variable $Y$ based on the values of the predictor variables $M, So, Ed, Po1, Po2, LF, M.F, Pop, NW, U1, U2, Wealth, Ineq, Prob$, and $Time$.

This model posted an adjusted R squared of 63.2%, which is lower than the model I applied in the last assignment (~74%).

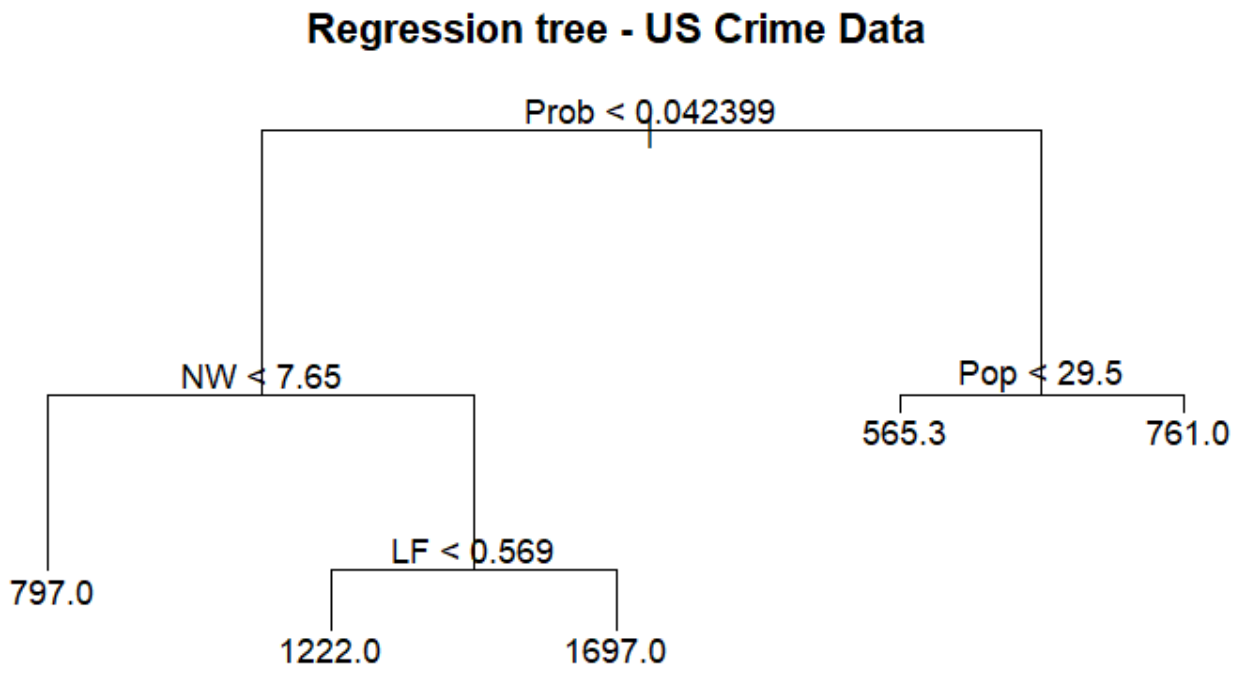
The final predicted value for the new data is 1230.418

# Question 10.1

*Using the same crime data set uscrime.txt as in Questions 8.2 and 9.1, find the best model you can using* (a) a regression tree model, and (b) a random forest model.
*In R, you can use the tree package or the rpart package, and the randomForest package. For each model, describe one or two qualitative takeaways you get from analyzing the results (i.e., don't just stop when you have a good model, but interpret it too).*

## Q10.1 a ANSWER

The regression tree model that is best based on my train/test split is 5 nodes based on deviance. Although this model is no where near perfect and overfits.

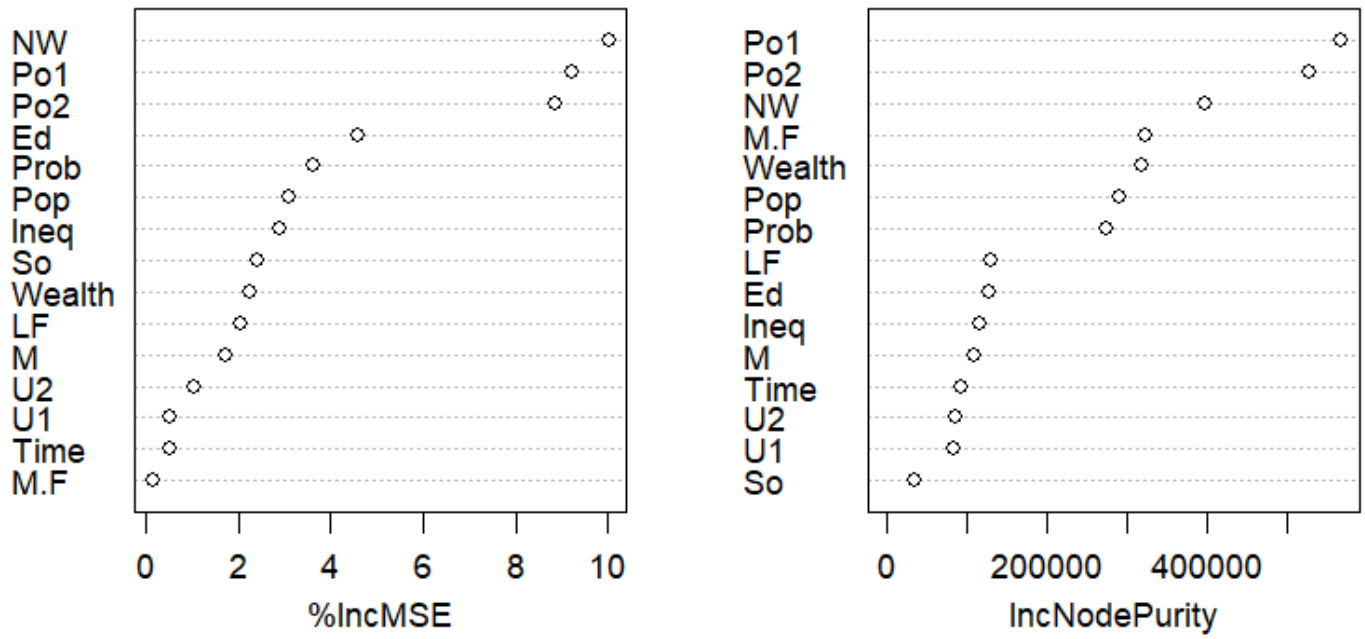



The 5 node model has some noteable splits. The tree initially splits at probability of imprisonment, followed by population and non whites in the population. The high population and high probability of imprisonment makes sense together. However with the split at non-white and the labor force split follows an inverse relationship. The highest rate coming from a low probability of imprisonment, high nonwhite population, and high labor force participation.

## Q10.1 b ANSWER

The best model random forest model asks for a low mtry and nodesize. This final model also had similar important features as the my first untuned model, specifically NW, Po1, Po2. The test MSE is the lowest of all tree models tested. Take aways from this model is that less inital split at nodes amongst trees (lower mtry) and less data needed to split at each node (nodesize) allows us to better model the data than previous regression tree models.

```
"Best params: Mtry =  4  nodesize =  2  test mse =  117663.261851467"
```




# Question 10.2

Q10.2 ANSWER

*Describe a situation or problem from your job, everyday life, current events, etc., for which a logistic regression model would be appropriate. List some (up to 5) predictors that you might use.*

- Identifying the probability that a truck would be late or not:
  - Weather estimates for the route/lane the truck will be driving on.
    - Although not always super common, likely the number one reason for delays/late appointments.
  - Traffic of the route/lane during its transit.
    - Similar to weather, but likely even more interaction with time of day. Traffic could very depending on when/where the truck is traveling.
  - Carrier efficiency: Average distance a particular driver can cover in a lane/route within the eligible operating hours.
  - En route delays (different depending on appointment type (Pick up or drop off))
    - For pickup, if the carrier is not notified right away when the pickup date is, the delay in lead time can result in late pickups.
    - For drop off and pickup, dwell times (time in which the driver sits at the warehouse BEFORE unloading) can impact their on-time status. Higher dwell time could likely lead to more late pick ups/drop offs.
    - There could be other things to consider in this feature, but the two mentioned are likely the most relevant
  - Shipment/Cargo Type: What is on the truck may alter how a truck must operate when getting to its destination. Lots of different reasons on why this could alter on time delivery. To name a few:
    - Weight of cargo
    - Refrigerated loads
    - Hazmat loads.

# Question 10.3

*1. Using the GermanCredit data set germancredit.txt from http://archive.ics.uci.edu/ml/machine-learning-databases/statlog/german (http://archive.ics.uci.edu/ml/machine-learning-databases/statlog/german) / (description at http://archive.ics.uci.edu/ml/datasets/Statlog+%28German+Credit+Data%29 (http://archive.ics.uci.edu/ml/datasets/Statlog+%28German+Credit+Data%29) ), use logistic regression to find a good predictive model for whether credit applicants are good credit risks or not. Show your model (factors used and their coefficients), the software output, and the quality of fit. You can use the glm function in R. To get a logistic regression (logit) model on data where the response is either zero or one, use family=binomial(link="logit") in your glm function call.*

*2. Because the model gives a result between 0 and 1, it requires setting a threshold probability to separate between "good" and "bad" answers. In this data set, they estimate that incorrectly identifying a bad customer as good, is 5 times worse than incorrectly classifying a good customer as bad. Determine a good threshold probability based on your model.*

Q10.3 Part 1 ANSWER

The model that best fits the data based on accuracy and specificty with a .5 threshold is below:

Metrics:

Accuracy = 74.67%, Specificity = 80.26%

The logistic regression model is represented by the following equation:

$$\begin{aligned}\text{logit}(P(Y=1)) = \beta_0 &+ \beta_1 \cdot V1A12 + \beta_2 \cdot V1A13 + \beta_3 \cdot V1A14 + \beta_4 \cdot V2 \\ &+ \beta_5 \cdot V3A31 + \beta_6 \cdot V3A32 + \beta_7 \cdot V3A33 + \beta_8 \cdot V3A34 \\ &+ \beta_9 \cdot V4A41 + \beta_{10} \cdot V4A410 + \beta_{11} \cdot V4A42 + \beta_{12} \cdot V4A43 \\ &+ \beta_{13} \cdot V4A44 + \beta_{14} \cdot V4A45 + \beta_{15} \cdot V4A46 + \beta_{16} \cdot V4A48 \\ &+ \beta_{17} \cdot V4A49 + \beta_{18} \cdot V5 + \beta_{19} \cdot V6A62 + \beta_{20} \cdot V6A63 \\ &+ \beta_{21} \cdot V6A64 + \beta_{22} \cdot V6A65 + \beta_{23} \cdot V8 + \beta_{24} \cdot V9A92 \\ &+ \beta_{25} \cdot V9A93 + \beta_{26} \cdot V9A94 + \beta_{27} \cdot V10A102 + \beta_{28} \cdot V10A103 \\ &+ \beta_{29} \cdot V14A142 + \beta_{30} \cdot V14A143 + \beta_{31} \cdot V15A152 + \beta_{32} \cdot V15A153 \\ &+ \beta_{33} \cdot V20A202\end{aligned}$$

where:

- $\beta_0$ (Intercept): 1.783
- $\beta_1$ (V1A12): -0.512
- $\beta_2$ (V1A13): -0.997
- $\beta_3$ (V1A14): -2.027
- $\beta_4$ (V2): 0.036
- $\beta_5$ (V3A31): -0.353
- $\beta_6$ (V3A32): -1.067
- $\beta_7$ (V3A33): -1.195
- $\beta_8$ (V3A34): -1.925
- $\beta_9$ (V4A41): -2.124
- $\beta_{10}$ (V4A410): -2.623
- $\beta_{11}$ (V4A42): -1.140
- $\beta_{12}$ (V4A43): -1.256
- $\beta_{13}$ (V4A44): -0.330
- $\beta_{14}$ (V4A45): -0.324
- $\beta_{15}$ (V4A46): 0.450
- $\beta_{16}$ (V4A48): -2.333
- $\beta_{17}$ (V4A49): -1.331
- $\beta_{18}$ (V5): 0.000176
- $\beta_{19}$ (V6A62): -0.381

- $\beta_{20}$ (V6A63): -0.490
- $\beta_{21}$ (V6A64): -1.082
- $\beta_{22}$ (V6A65): -1.365
- $\beta_{23}$ (V8): 0.395
- $\beta_{24}$ (V9A92): -0.366
- $\beta_{25}$ (V9A93): -1.110
- $\beta_{26}$ (V9A94): -0.621
- $\beta_{27}$ (V10A102): 0.514
- $\beta_{28}$ (V10A103): -1.482
- $\beta_{29}$ (V14A142): -0.358
- $\beta_{30}$ (V14A143): -0.836
- $\beta_{31}$ (V15A152): -0.607
- $\beta_{32}$ (V15A153): -0.384
- $\beta_{33}$ (V20A202): -1.134

Q10.3 Part 2 ANSWER

With a lower threshold, between 0.2 and 0.3, we get a better trade off of accuracy and specificity.

| thresh_val | spec | acc |
|---|---|---|
| <dbl> | <dbl> | <dbl> |
| 0.2 | 0.8243243 | 0.6066667 |
| 0.3 | 0.8105263 | 0.6800000 |

# CODE / WORK / IDEAS

First loading in some general libraries before starting the questions

Hide

```
#the best around, lets give it up for the tidyverse
library('tidyverse')


#psych library for descriptive stats
library('psych')

# for ts analysis
library('stats')

#olsrr for detailed OLS regression
library('olsrr')

#for data manipulation
library('reshape2')

#caret for test/train splits
library('caret')

#for pretty outputs:
library('knitr')
```

Q9.1 WORK

Starting by reading in the data:

Running my inital PCA analysis on the data

In order to determine the number of meaningful components, we will do a bit of EDA on the components.

Article I used to better understand this process: https://rpubs.com/DragonflyStats/pca_components_number (https://rpubs.com/DragonflyStats/pca_components_number)

Following the article above, there are a few ways to do this. One way is using some visual analysis via a screeplot, and utilizing the Eigenvalue-One Criterion. This method just has us take any PC with the variance/eigenvalue above 1. The eigenvalue-one criterion suggests that only principal components with eigenvalues greater than 1.00 should be retained because they account for more variance than a single observed variable, making them meaningful. Components with eigenvalues less than 1.00 are considered trivial as they account for less variance than individual variables and should be discarded to effectively reduce the dataset's dimensionality.

This chart shows me that PCs 1-5 explain notable variance. This type of analysis would suggest that we use 1-5 PCs in order to model our data. This approach is simple, but is also a bit flawed. It is a tad bit biased to due a rule of thumb approach.

Another approach is the scree test. This is simlar to our approach with finding the elbow in kmeans clustering. However I will use the max absolute second derivative to find the optimal PC based on the eigenvalues

Hide

```
# Grabbing some variables to create a table of important info for our screetest
eigenvalues = pca_result$sdev^2
proportion_variance = eigenvalues / sum(eigenvalues)
cumulative_variance = cumsum(proportion_variance)

# Create a data frame
scree_data <- data.frame(
  Principal_Component =1:length(eigenvalues),
  Eigenvalue = eigenvalues,
  Proportion_of_Variance = proportion_variance,
  Cumulative_Proportion = cumulative_variance
)

scree_data = scree_data %>% arrange(Principal_Component)

#Finding derivatives
first_derivative = diff(scree_data$Eigenvalue, differences = 1)
second_derivative = diff(first_derivative, differences = 1)

#Find the index of the maximum absolute second derivative
max_abs_second_derivative_index = which.max(abs(second_derivative)) + 2 #to account for differencing

scree_data[max_abs_second_derivative_index,]
```

| | Principal_Component <int> | Eigenvalue <dbl> | Proportion_of_Variance <dbl> | Cumulative_Proportion <dbl> |
|---|---|---|---|---|
| 3 | 3 | 2.004944 | 0.133663 | 0.7217163 |

1 row

Hide

```
NA
NA
NA
```

This approach says that the breakpoint in the scree plot data is at PC = 3.

The last approach would be looking for a majority of variance explained using the least amount of components. So say we aim for 90% variance explained, we would look at the table below to figure out what principle components make up that total.

Hide

```
scree_data
```

| Principal_Component <int> | Eigenvalue <dbl> | Proportion_of_Variance <dbl> | Cumulative_Proportion <dbl> |
|---|---|---|---|
| 1 | 6.018952657 | 0.401263510 | 0.4012635 |
| 2 | 2.801847026 | 0.186789802 | 0.5880533 |
| 3 | 2.004944334 | 0.133662956 | 0.7217163 |
| 4 | 1.162207801 | 0.077480520 | 0.7991968 |
| 5 | 0.958298972 | 0.063886598 | 0.8630834 |
| 6 | 0.553193900 | 0.036879593 | 0.8999630 |
| 7 | 0.321818687 | 0.021454579 | 0.9214176 |
| 8 | 0.307401270 | 0.020493418 | 0.9419110 |
| 9 | 0.235155292 | 0.015677019 | 0.9575880 |
| 10 | 0.199880931 | 0.013325395 | 0.9709134 |

1-10 of 15 rows Previous **1** 2 Next

This table shows us 7 would be the right number of PCs to be used.

Going to use cross validation to test 3, 5, 7 PCs to find the best model based on R2.

Hide

```
# Load necessary libraries
library(DAAG)


# Assuming uscrime_data and pca_result are already defined
# Define the parameter grid
param_grid <- expand.grid(C=c(3, 5, 7))

Model_output = list()

for (i in 1:nrow(param_grid)) {

  # Grabbing PC value
  param = param_grid[i, ]


  # Extract the desired number of principal components
  pc_values = pca_result$x[, 1:param]

  pc_pred = ncol(pc_values)


  # Combine with the 'Crime' column
  crime_pc = cbind(Crime = uscrime_data[, 16], pc_values)

  # Convert to data frame
  crime_pc_df = as.data.frame(crime_pc)

  # Create the formula for the linear model
  formula = as.formula(paste("Crime ~", paste(colnames(pc_values), collapse = " + ")))

  folds = 3
  # Use cv.lm to perform cross-validation
  cv_results = cv.lm(crime_pc_df, formula, m = folds, plotit = FALSE, printit = FALSE)

  summary(cv_results)

  fold_R2 = numeric(folds)
  #looping through each fold and calculating adjusted R^2
  for (k in 1:folds){
    fold_df = cv_results%>%filter(fold==k)
    obs = fold_df$Crime
    pred = fold_df$cvpred

     R2 = cor(obs,pred)^2


    fold_R2[k] = R2

    mean_R2 = mean(fold_R2)
  }
```

```
  Model_output[[i]] = list( formula=formula, R2=mean_R2)

}

print(Model_output)
```

```
[[1]]
[[1]]$formula
Crime ~ PC1 + PC2 + PC3

[[1]]$R2
[1] 0.2129415


[[2]]
[[2]]$formula
Crime ~ PC1 + PC2 + PC3 + PC4 + PC5

[[2]]$R2
[1] 0.5709027


[[3]]
[[3]]$formula
Crime ~ PC1 + PC2 + PC3 + PC4 + PC5 + PC6 + PC7

[[3]]$R2
[1] 0.6027343
```

Using cross validation, we can see that model with PC 1-7 has the highest R2. We will role with that for the final model.

Hide

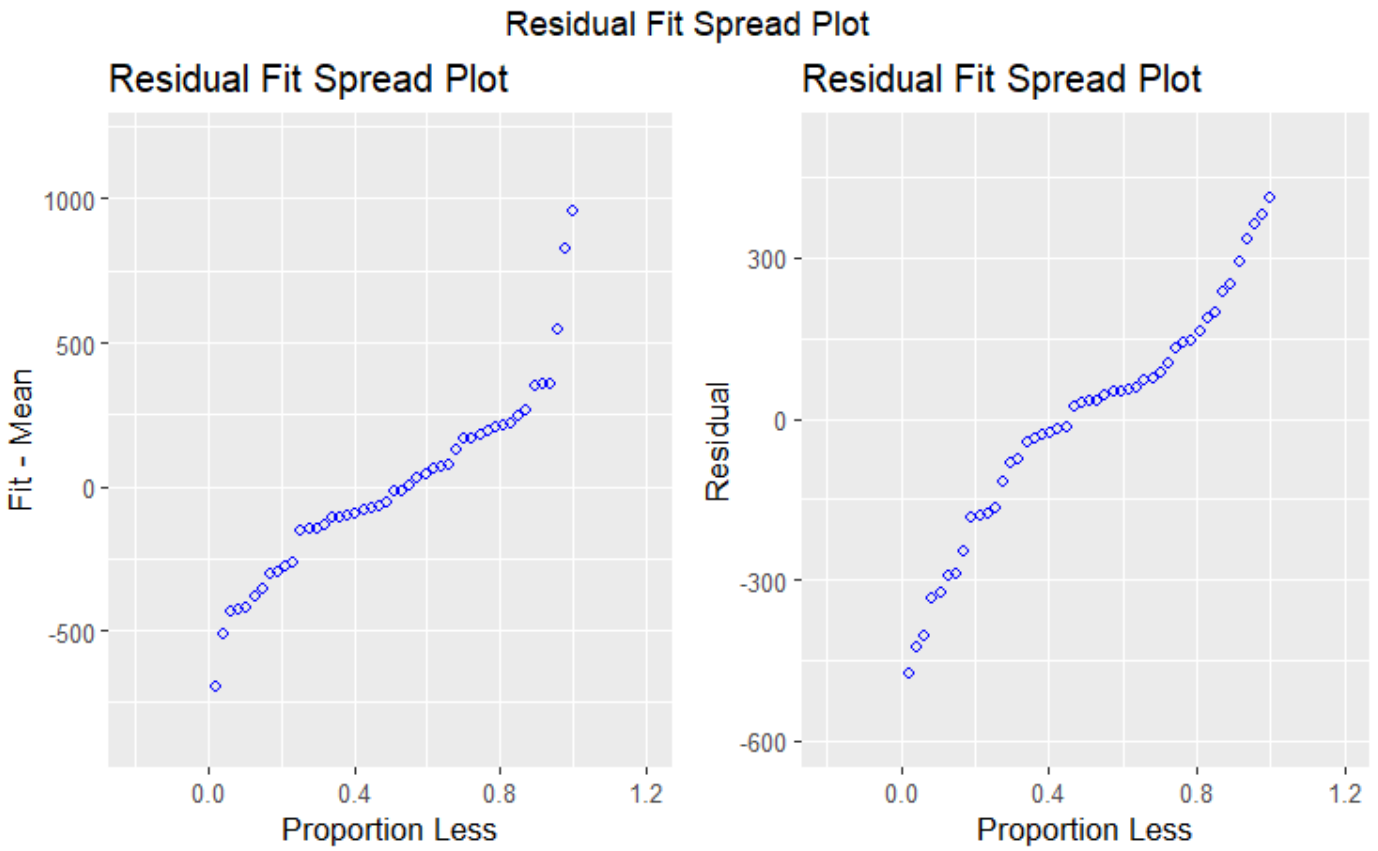
```
# Combine with the 'Crime' column

pc_values = pca_result$x[, 1:7]
crime_pc = cbind(Crime = uscrime_data[, 16], pc_values)

# Convert to data frame
crime_pc_df = as.data.frame(crime_pc)



# Create the formula for the linear model
formula = as.formula(paste("Crime ~", paste(colnames(pc_values), collapse = " + ")))

final_PC_model = lm(formula, data=crime_pc_df)


ols_regress(final_PC_model)
```

```
                         Model Summary
-----------------------------------------------------------------
R                         0.830       RMSE                213.661
R-Squared                 0.688       MSE               55015.333
Adj. R-Squared            0.632       Coef. Var            25.915
Pred R-Squared            0.530       AIC                 655.633
MAE                     168.184       SBC                 672.284
-----------------------------------------------------------------
 RMSE: Root Mean Square Error
 MSE: Mean Square Error
 MAE: Mean Absolute Error
 AIC: Akaike Information Criteria
 SBC: Schwarz Bayesian Criteria

                               ANOVA
--------------------------------------------------------------------------
                   Sum of
                  Squares        DF    Mean Square      F         Sig.
--------------------------------------------------------------------------
Regression    4735329.685         7     676475.669    12.296    0.0000
Residual      2145597.975        39      55015.333
Total         6880927.660        46
--------------------------------------------------------------------------

                                  Parameter Estimates
-----------------------------------------------------------------------------------------------------
      model        Beta    Std. Error    Std. Beta      t        Sig        lower       upper
-----------------------------------------------------------------------------------------------------
(Intercept)     905.085        34.213                26.454    0.000     835.882     974.288
        PC1      65.216        14.096        0.414    4.626    0.000      36.704      93.728
        PC2     -70.083        20.660       -0.303   -3.392    0.002    -111.873     -28.293
        PC3      25.194        24.424        0.092    1.032    0.309     -24.208      74.596
        PC4      69.446        32.079        0.194    2.165    0.037       4.560     134.332
        PC5    -229.043        35.327       -0.580   -6.483    0.000    -300.499    -157.586
        PC6     -60.213        46.497       -0.116   -1.295    0.203    -154.262      33.836
        PC7     117.256        60.962        0.172    1.923    0.062      -6.051     240.563
-----------------------------------------------------------------------------------------------------
```

Running the regression diagnostics to see if the model has some heteroskedatsicity problems:

Hide

```
ols_plot_resid_fit_spread(final_PC_model)
```




Hide

```
ols_test_breusch_pagan(final_PC_model)
```

```
Breusch Pagan Test for Heteroskedasticity
-----------------------------------------
Ho: the variance is constant
Ha: the variance is not constant

             Data
---------------------------------
Response : Crime
Variables: fitted values of Crime

       Test Summary
-----------------------------
DF            =    1
Chi2          =    3.146312
Prob > Chi2   =    0.07609877
```

Although the model has a better R2, there are signs of no normally distributed errors using PC1-7.

Just for yucks, we will try the PC1-5 model, since there were also some variables in the first model that we insignificant.

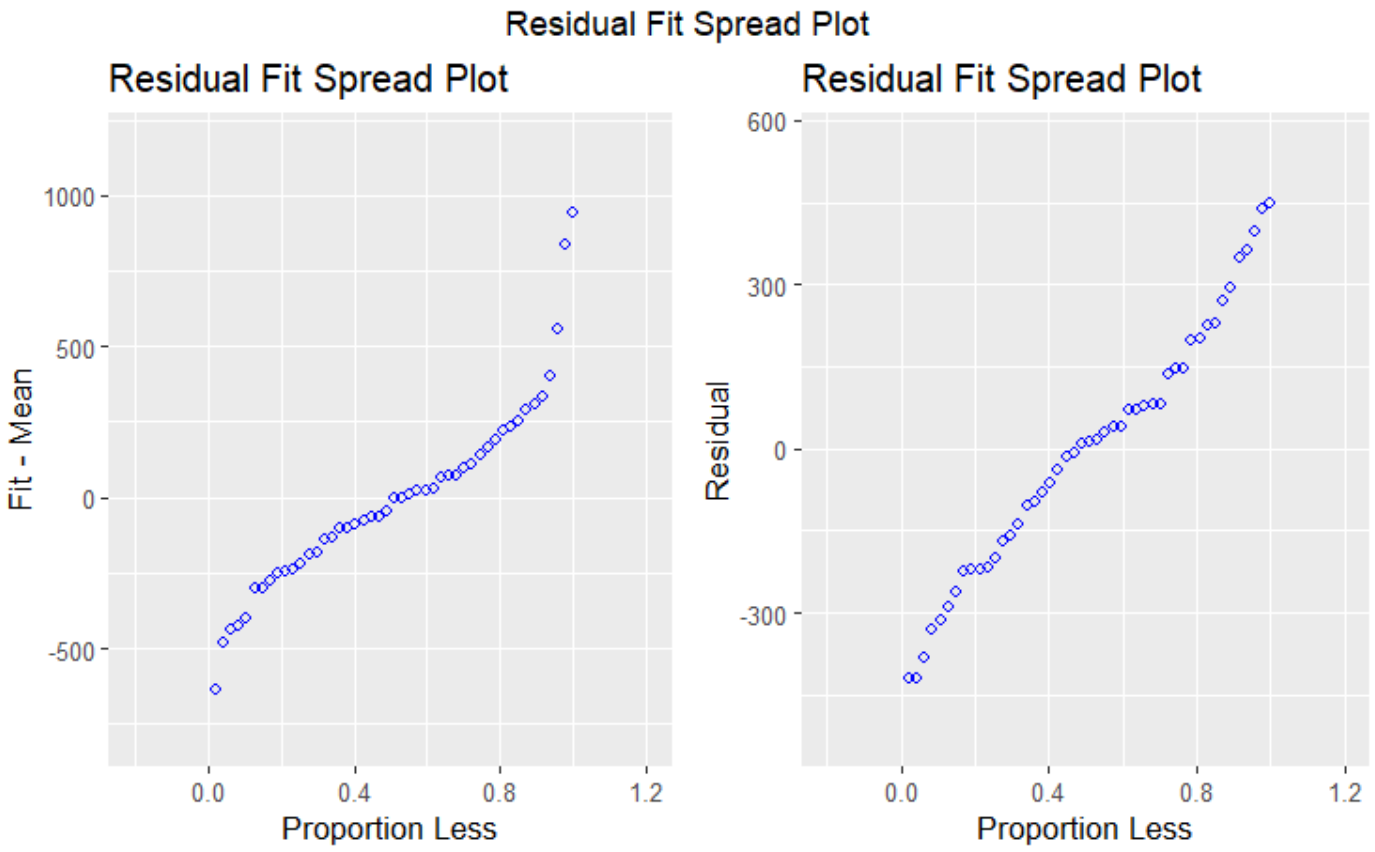
Hide

```
#doing PC1-5
pc_values <- pca_result$x[, 1:5]
crime_pc <- cbind(Crime = uscrime_data[, 16], pc_values)

# Convert to data frame
crime_pc_df <- as.data.frame(crime_pc)



# Create the formula for the linear model
formula <- as.formula(paste("Crime ~", paste(colnames(pc_values), collapse = " + ")))

PC15_model = lm(formula, data=crime_pc_df)


ols_regress(PC15_model)
```

```
                         Model Summary
-----------------------------------------------------------------
R                        0.803       RMSE                227.913
R-Squared                0.645       MSE               59546.196
Adj. R-Squared           0.602       Coef. Var            26.961
Pred R-Squared           0.516       AIC                 657.703
MAE                    186.006       SBC                 670.654
-----------------------------------------------------------------
 RMSE: Root Mean Square Error
 MSE: Mean Square Error
 MAE: Mean Absolute Error
 AIC: Akaike Information Criteria
 SBC: Schwarz Bayesian Criteria

                                ANOVA
-------------------------------------------------------------------------
                  Sum of
                 Squares        DF    Mean Square      F         Sig.
-------------------------------------------------------------------------
Regression    4439533.610         5     887906.722    14.911    0.0000
Residual      2441394.050        41      59546.196
Total         6880927.660        46
-------------------------------------------------------------------------

                                   Parameter Estimates
-----------------------------------------------------------------------------------------------
      model        Beta    Std. Error    Std. Beta      t        Sig        lower       upper
-----------------------------------------------------------------------------------------------
(Intercept)     905.085        35.594                25.428    0.000     833.201     976.969
        PC1      65.216        14.665        0.414    4.447    0.000      35.599      94.833
        PC2     -70.083        21.494       -0.303   -3.261    0.002    -113.492     -26.674
        PC3      25.194        25.410        0.092    0.992    0.327     -26.122      76.510
        PC4      69.446        33.374        0.194    2.081    0.044       2.046     136.846
        PC5    -229.043        36.753       -0.580   -6.232    0.000    -303.268    -154.818
-----------------------------------------------------------------------------------------------
```

Hide

```
ols_plot_resid_fit_spread(PC15_model)
```




Hide

```
ols_test_breusch_pagan(PC15_model)
```

```
Breusch Pagan Test for Heteroskedasticity
-----------------------------------------
Ho: the variance is constant
Ha: the variance is not constant

             Data
---------------------------------
Response : Crime
Variables: fitted values of Crime

       Test Summary
----------------------------
DF            =    1
Chi2          =    2.486967
Prob > Chi2   =    0.1147928
```

The results were similar with a lower R2. Will continue with the PC1-7 model

Converting the model in order to predict the new data

Hide

```
options(scipen=999)

#Transformation back to original variables
pca_intercept = final_PC_model$coefficients[1]
pca_model_coefficents = final_PC_model$coefficients[2:(7+1)]
alpha = pca_result$rotation[,1:7] %*% pca_model_coefficents
mu = sapply(uscrime_data[,1:15],mean)
sigma = sapply(uscrime_data[,1:15],sd)
original_feature_weights = alpha/sigma
original_intercept = pca_intercept - sum(alpha*mu /sigma)
estimates = as.matrix(uscrime_data[,1:15]) %*% original_feature_weights + original_intercept

print(original_feature_weights)
```

```
                [,1]
M         55.23734970
So       139.75711512
Ed        -6.80383633
Po1       44.58638138
Po2       46.42432022
LF       673.38092394
M.F       44.40293135
Pop        0.95990758
NW         5.68493965
U1     -1027.73478505
U2        24.41589109
Wealth     0.02883565
Ineq      12.45113438
Prob   -5170.56913662
Time      -2.21509529
```

Hide

```
print(original_intercept)
```

```
(Intercept)
  -5498.458
```

Projecting the new data into the PCA, then using our PCA model we can predict the crime value

Hide

```
# Create a new data frame with your inputs
new_data = data.frame(M = 14.0,
                      So = 0,
                      Ed = 10.0,
                      Po1 = 12.0,
                      Po2 = 15.5,
                      LF = 0.640,
                      M.F = 94.0,
                      Pop = 150,
                      NW = 1.1,
                      U1 = 0.120,
                      U2 = 3.6,
                      Wealth = 3200,
                      Ineq = 20.1,
                      Prob = 0.04,
                      Time = 39.0)


new_data_pca = predict(pca_result, new_data)

new_data_pca_df = data.frame(new_data_pca)

#predicting the value using the PC1-7 model

crime_prediction = predict(final_PC_model, new_data_pca_df)

print(crime_prediction)
```

```
       1
1230.418
```

## WORK 10.1 a

Hide

```
#loading some packages and creating some functions
library("tree")


train_index = sample(1:nrow(uscrime_data), 0.7* nrow(uscrime_data))

train_uscrime_data = uscrime_data[train_index,]
test_uscrime_data = uscrime_data[-train_index,] #stopping at test dataset due to sample size

calculate_r_squared = function(actual, predicted) {
  ss_total = sum((actual - mean(actual))^2)
  ss_residual = sum((actual - predicted)^2)
  r_squared = 1 - (ss_residual / ss_total)
  return(r_squared)
}

calculate_mse = function(actual, predicted) {
  mean((actual - predicted)^2)
}
```

Hide

```
# Fit the regression tree model
tree_model = tree(Crime ~ ., data = train_uscrime_data)

summary(tree_model)
```

```

Regression tree:
tree(formula = Crime ~ ., data = train_uscrime_data)
Variables actually used in tree construction:
[1] "Po1"  "M"    "Prob"
Number of terminal nodes:  5
Residual mean deviance:  42400 = 1145000 / 27
Distribution of residuals:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-420.00  -90.18    1.40    0.00   80.50  644.00
```

Hide

```
train_tree_predict = predict(tree_model, newdata = train_uscrime_data)
test_tree_predict = predict(tree_model, newdata = test_uscrime_data)
train_r2 = calculate_r_squared(train_uscrime_data$Crime, train_tree_predict)
train_mse = calculate_mse(train_uscrime_data$Crime, train_tree_predict)
test_r2 = calculate_r_squared(train_uscrime_data$Crime, test_tree_predict)
```

```
Warning in actual - predicted :
  longer object length is not a multiple of shorter object length
```

Hide

```
test_mse = calculate_mse(train_uscrime_data$Crime, test_tree_predict)
```

```
Warning in actual - predicted :
  longer object length is not a multiple of shorter object length
```

Hide

```
print(paste("Train R2 = ", train_r2, " Train MSE = ", train_mse))
```

```
[1] "Train R2 =  0.689867532847674  Train MSE =  35773.5035714286"
```

Hide

```
print(paste("Test R2 = ", test_r2, " Test MSE = ", test_mse))
```

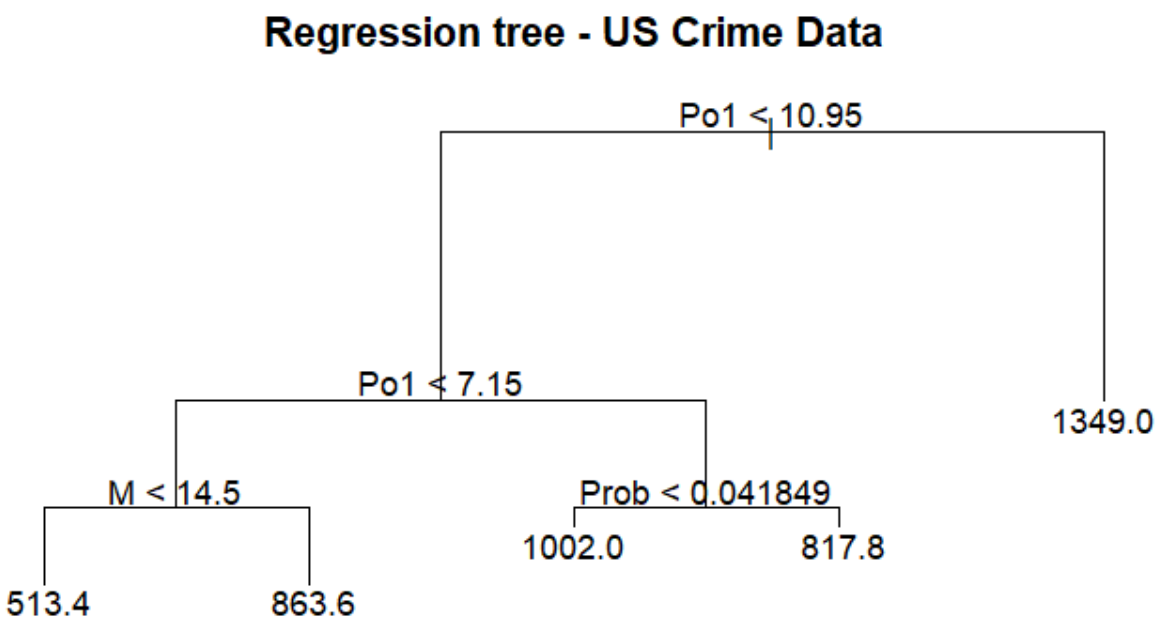
```
[1] "Test R2 =  -0.391178760247103  Test MSE =  160471.229617347"
```

Hide

```
plot(tree_model)
text(tree_model)
```

Hide

```
title("Regression tree - US Crime Data")
```




The model fit the data using 4 different variables. The performance on the train data indicated a R2 value of 84% with an MSE of 27662.1. The test data on the other hand did not do so great…with a negative R2 and almost 10x the MSE than the train dataset. Likely a problem of overfitting with the current data. Going to try and prune the data subjectively and then

programatically. I followed this article for inspiration: https://homerhanumat.github.io/tigerTree/pruneTree.html (https://homerhanumat.github.io/tigerTree/pruneTree.html)

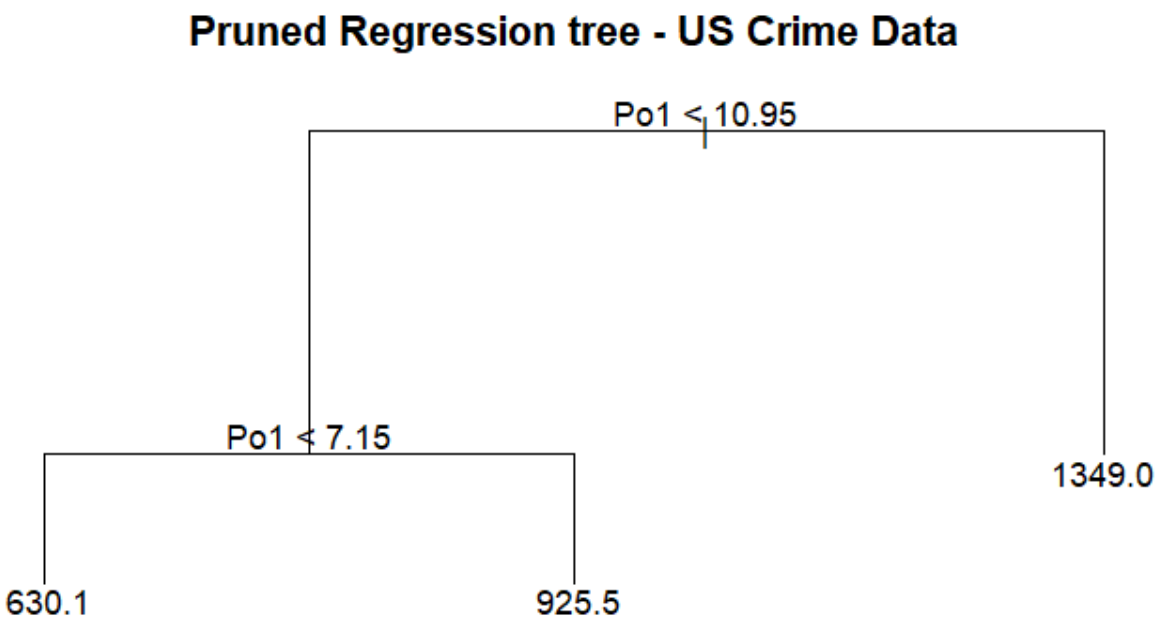
Subjective fitting, prunning down to 3 nodes.

Hide

```
pruned_tree = prune.tree(tree_model, best = 3)
# Plot the pruned tree
plot(pruned_tree)
text(pruned_tree, pretty = 0)
```

Hide

```
title("Pruned Regression tree - US Crime Data")
```




Hide

```
train_tree_predict = predict(pruned_tree, newdata = train_uscrime_data)
test_tree_predict = predict(pruned_tree, newdata = test_uscrime_data)
train_r2 = calculate_r_squared(train_uscrime_data$Crime, train_tree_predict)
train_mse = calculate_mse(train_uscrime_data$Crime, train_tree_predict)
test_r2 = calculate_r_squared(train_uscrime_data$Crime, test_tree_predict)
```

```
Warning in actual - predicted :
  longer object length is not a multiple of shorter object length
```

Hide

```
test_mse = calculate_mse(train_uscrime_data$Crime, test_tree_predict)
```

```
Warning in actual - predicted :
  longer object length is not a multiple of shorter object length
```

Hide

```
print(paste("Train R2 = ", train_r2, " Train MSE = ", train_mse))
```

```
[1] "Train R2 =  0.552181537127803  Train MSE =  51655.4604166667"
```

Hide

```
print(paste("Test R2 = ", test_r2, " Test MSE = ", test_mse))
```

```
[1] "Test R2 =  -0.376268790336787  Test MSE =  158751.377881944"
```

The prunned model is actually worse. If we verify the “best” tree model based on devations, we can figure the best number of nodes for the data

Hide

```
# Prune the tree to the optimal size
pruned_tree = prune.tree(tree_model,method = "deviance")
prune_deviance_data = data.frame(size = pruned_tree$size, dev = pruned_tree$dev)

print(prune_deviance_data)
```

| size<br><int> | dev<br><dbl> |
|---|---|
| 5 | 1144752 |
| 4 | 1244175 |
| 3 | 1652975 |
| 2 | 2234585 |
| 1 | 3691171 |

5 rows

Hide

```
NA
NA
```

## WORK 10.1 b

Running a similar process but using random forest

Hide

```
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:psych’:

    outlier

The following object is masked from ‘package:dplyr’:

    combine

The following object is masked from ‘package:ggplot2’:

    margin
```

Hide

```
rf_model = randomForest(Crime~.,data=train_uscrime_data, importance=TRUE)

print(rf_model)
```

```

Call:
 randomForest(formula = Crime ~ ., data = train_uscrime_data,      importance = TRUE)
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 5

          Mean of squared residuals: 79413.2
                    % Var explained: 31.15
```

Hide

```
predict_train = predict(rf_model, newdata=train_uscrime_data)

actual_train = train_uscrime_data$Crime

train_mse = mean((actual_train - predict_train)^2)

predict_test = predict(rf_model, newdata=test_uscrime_data)

actual_test = test_uscrime_data$Crime

test_mse = mean((actual_test - predict_test)^2)

print(paste("Train MSE = ", train_mse))
```

```
[1] "Train MSE =  15002.7593492236"
```

Hide

```
print(paste("Test MSE = ", test_mse))
```

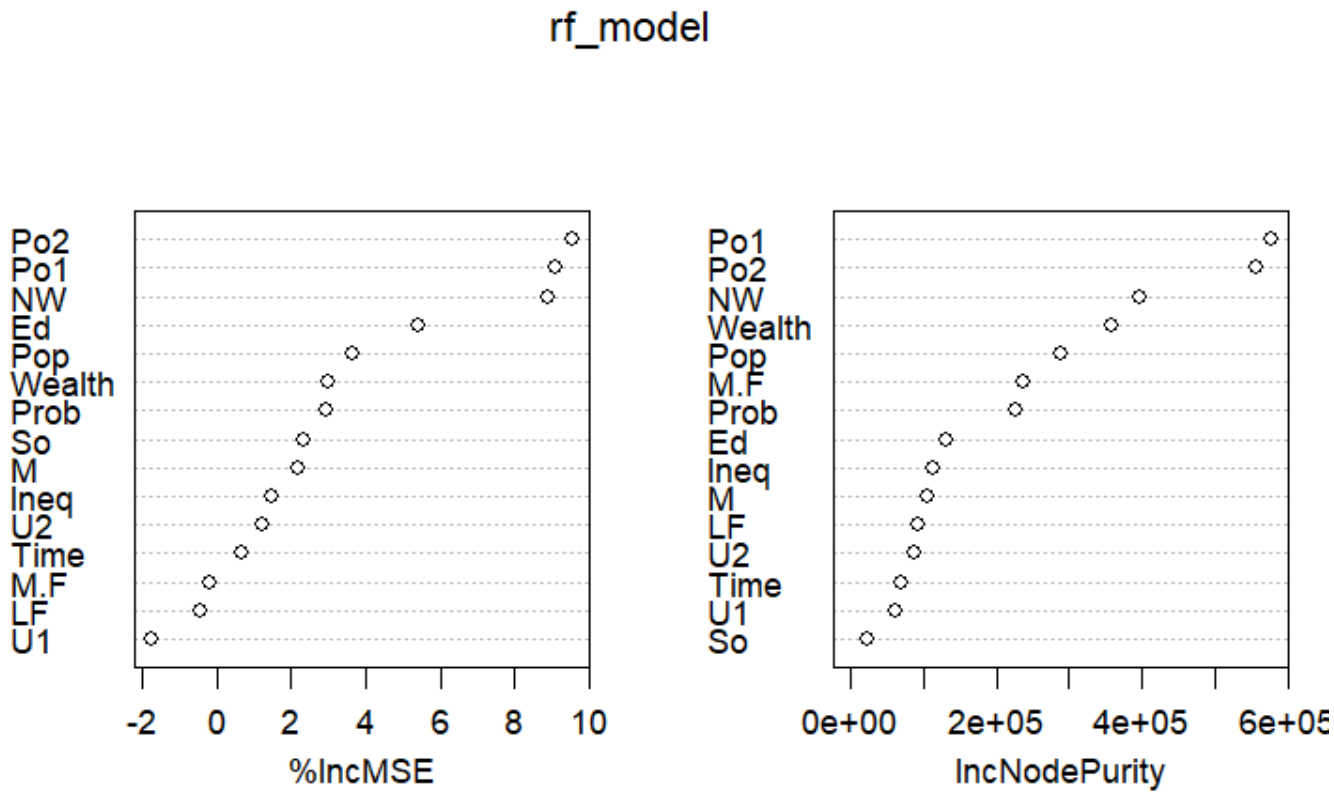
```
[1] "Test MSE =  126387.570917621"
```

Although the model has more error, it is much less overfit. The increase in MSE between the test and train sets are much smaller in comparsion to the regression tree models.

Going to take a look at feature importance first to see how each variables is impacting the model.

Hide

```
varImpPlot(rf_model)
```




Focusing on %IncMSE, which helps us narrow down which variables had more impact on predication error when producing our forest of regression trees, we see that variables Prob, NW, Po1, Po2, and U2 stick out the most amongst other variables.

Going to try and grid search my way to find the best hyper parameters to fit our random forest model in hopes to decrease overall MSE.

Hide

```
mtry_grid = 2:15
node_grid  = 1:25

model_results = data.frame(mtry = integer(), nodesize = integer(), train_mse = numeric(), test_mse = nu
meric())

for (node_size in node_grid) {
  for (m in mtry_grid) {
    rf_model_grid = randomForest(Crime~.,data=train_uscrime_data, importance=TRUE, nodesize=node_size,
mtry=m)
    predict_train = predict(rf_model_grid, newdata=train_uscrime_data)
    predict_test = predict(rf_model_grid, newdata=test_uscrime_data)

    #accuracy measures
    actual_train = train_uscrime_data$Crime
    train_mse = mean((actual_train - predict_train)^2)
    actual_test = test_uscrime_data$Crime
    test_mse = mean((actual_test - predict_test)^2)

    model_results = rbind(model_results, data.frame(mtry = m, nodesize = node_size, train_mse = train_m
se, test_mse = test_mse))

  }
}

print(model_results)
```

| mtry <int> | nodesize <int> | train_mse <dbl> | test_mse <dbl> |
|---|---|---|---|
| 2 | 1 | 9850.987 | 123548.5 |
| 3 | 1 | 9801.955 | 123795.9 |
| 4 | 1 | 10681.384 | 125695.4 |
| 5 | 1 | 9894.414 | 126117.2 |
| 6 | 1 | 11188.260 | 128568.8 |
| 7 | 1 | 10501.938 | 125383.5 |
| 8 | 1 | 10228.199 | 126956.9 |
| 9 | 1 | 9761.336 | 130369.1 |
| 10 | 1 | 11706.503 | 129440.7 |
| 11 | 1 | 10408.003 | 130269.5 |



Hide

```
NA
```

Looking at the best model

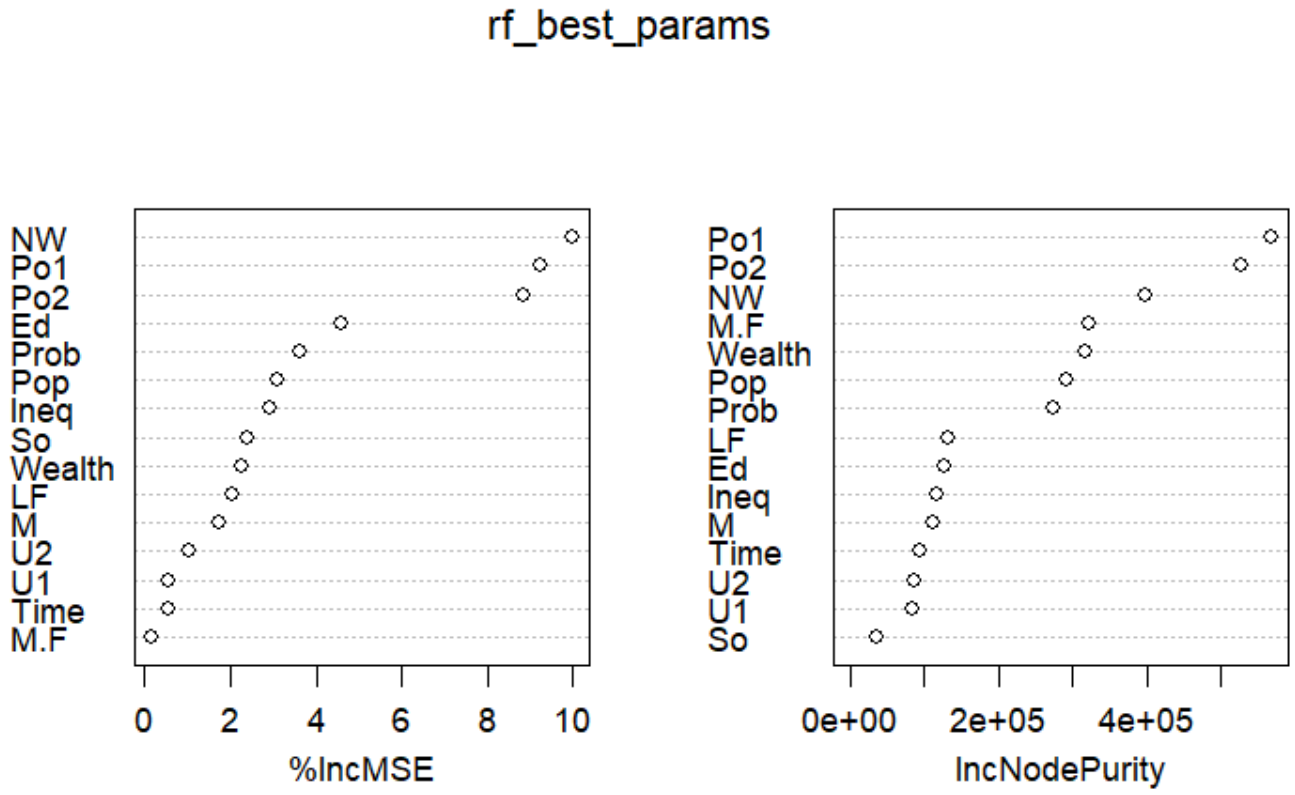
Hide

```
best_rf_model = model_results[which.min(model_results$test_mse),]

print(paste("Best params: Mtry = ",best_rf_model$mtry, " nodesize = ", best_rf_model$nodesize, " test m
se = ", best_rf_model$test_mse))
```

```
[1] "Best params: Mtry =  4  nodesize =  2  test mse =  117663.261851467"
```

Hide

```
rf_best_params = randomForest(Crime~.,data=train_uscrime_data, importance=TRUE, nodesize=best_rf_model
$nodesize, mtry=best_rf_model$mtry)

varImpPlot(rf_best_params)
```




WORK 10.3 part 1

Reading in the data

Hide

```
german_credit_url = "http://archive.ics.uci.edu/ml/machine-learning-databases/statlog/german/german.dat
a"


# Read the data from the URL
german_credit_data = read.table(german_credit_url, header = FALSE)

#changing V21 (target good/bad to 0 = good, 1 = bad)

german_credit_data = german_credit_data %>% mutate(V21 = ifelse(V21 == 1, 0, 1))

# Print the first few rows of the data to check
head(german_credit_data)
```

Getting a better idea of the data:

Hide

```
describe(german_credit_data)
```

astricts in my summary indicates categorical variables. These will be converted into dummy variables

Seeing if class is imbalanced

Hide

```
german_credit_data %>%
  group_by(V21) %>%
  summarise(count = n())
```

I would call that an imbalance….. will talk about upsampling later.

Creating test train split and double checking imbalance

Hide

```
train_index = sample(1:nrow(german_credit_data), 0.7* nrow(german_credit_data))

train_gcd = german_credit_data[train_index,]
test_gcd = german_credit_data[-train_index,]

train_gcd %>%
  group_by(V21) %>%
  summarise(count = n())
```

| V21 <dbl> | count <int> |
|---|---|
| 0 | 487 |
| 1 | 213 |

2 rows

Hide

```
test_gcd %>%
  group_by(V21) %>%
  summarise(count = n())
```

| V21 <dbl> | count <int> |
|---|---|
| 0 | 213 |
| 1 | 87 |

2 rows

Hide

```
NA
NA
NA
NA
```

Fitting the base logistic regression model

Hide

```
train_lr = glm(V21 ~ ., data = train_gcd, family=binomial(link="logit"))

summary(train_lr)
```

```
Call:
glm(formula = V21 ~ ., family = binomial(link = "logit"), data = train_gcd)

Coefficients:
               Estimate  Std. Error z value        Pr(>|z|)
(Intercept)  1.51717041  1.34663672   1.127        0.259896
V1A12       -0.49883066  0.28394981  -1.757        0.078959 .
V1A13       -0.97646308  0.46418839  -2.104        0.035414 *
V1A14       -1.99214483  0.30308838  -6.573 0.0000000000494 ***
V2           0.03919019  0.01191614   3.289        0.001006 **
V3A31        0.05517355  0.67928279   0.081        0.935264
V3A32       -0.73510528  0.55585954  -1.322        0.186013
V3A33       -1.02224769  0.60616563  -1.686        0.091716 .
V3A34       -1.81933587  0.56735452  -3.207        0.001343 **
V4A41       -2.19102567  0.46585654  -4.703 0.0000025609055 ***
V4A410      -2.78386957  1.15697625  -2.406        0.016121 *
V4A42       -1.19676962  0.33431841  -3.580        0.000344 ***
V4A43       -1.24758809  0.32159706  -3.879        0.000105 ***
V4A44       -0.25832758  0.98869684  -0.261        0.793876
V4A45       -0.49267418  0.67597305  -0.729        0.466101
V4A46        0.46125060  0.54254192   0.850        0.395233
V4A48       -2.22955772  1.33149425  -1.674        0.094037 .
V4A49       -1.32169968  0.42848262  -3.085        0.002038 **
V5           0.00019586  0.00005727   3.420        0.000627 ***
V6A62       -0.43089367  0.35813258  -1.203        0.228911
V6A63       -0.44945132  0.54509985  -0.825        0.409638
V6A64       -1.09327547  0.64995775  -1.682        0.092555 .
V6A65       -1.36274984  0.35852250  -3.801        0.000144 ***
V7A72       -0.20055896  0.56712569  -0.354        0.723608
V7A73       -0.24599480  0.55563405  -0.443        0.657962
V7A74       -1.01965426  0.60082573  -1.697        0.089680 .
V7A75       -0.06014122  0.55008508  -0.109        0.912940
V8           0.40673138  0.11324625   3.592        0.000329 ***
V9A92       -0.38846730  0.49111475  -0.791        0.428949
V9A93       -1.16855209  0.48070642  -2.431        0.015061 *
V9A94       -0.66637341  0.57871346  -1.151        0.249537
V10A102      0.51652345  0.51134033   1.010        0.312430
V10A103     -1.38561241  0.55467753  -2.498        0.012488 *
V11         -0.03583306  0.11223949  -0.319        0.749533
V12A122      0.12134459  0.32493789   0.373        0.708821
V12A123      0.09239282  0.30454925   0.303        0.761604
V12A124      0.38833968  0.56149932   0.692        0.489181
V13         -0.01392280  0.01182621  -1.177        0.239083
V14A142     -0.47056892  0.51211279  -0.919        0.358160
V14A143     -0.81176176  0.30559765  -2.656        0.007900 **
V15A152     -0.64023054  0.31532467  -2.030        0.042317 *
V15A153     -0.64509072  0.62602846  -1.030        0.302799
V16          0.29373985  0.23162891   1.268        0.204745
V17A172      0.01466917  0.81356031   0.018        0.985614
V17A173      0.04734817  0.77699477   0.061        0.951409
V17A174      0.04835435  0.78619372   0.062        0.950958
V18          0.28721895  0.31140663   0.922        0.356358
V19A192     -0.27074463  0.26240325  -1.032        0.302171
V20A202     -1.18920523  0.70738096  -1.681        0.092736 .
```

```
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 861.88  on 699  degrees of freedom
Residual deviance: 567.93  on 651  degrees of freedom
AIC: 665.93

Number of Fisher Scoring iterations: 5
```

Hide

```
predictions = predict(train_lr, newdata = test_gcd, type = "response")
predicted_classes = ifelse(predictions > 0.5, 1, 0)
predicted_classes = as.factor(predicted_classes)

# Evaluate the model
conf_matrix = table(test_gcd$V21, predicted_classes)

acc = sum(ifelse(predicted_classes == test_gcd$V21, 1, 0)) / nrow(test_gcd)

fp = conf_matrix[2,1]
tn = conf_matrix[1,1]

spec = tn / (fp + tn)

print(paste("Accuracy = ", acc, " Specificity = ", spec))
```

```
[1] "Accuracy =  0.73  Specificity =  0.795555555555556"
```

Hide

```
print(conf_matrix)
```

```
   predicted_classes
      0   1
  0 179  35
  1  46  40
```

The final model results in decent accuracy and specificity. Going to do some stepwise logistic regression to find better variables to use via AIC. Followed this article to better understand this process of variable selection: http://www.sthda.com/english/articles/36-classification-methods-essentials/150-stepwise-logistic-regression-essentials-in-r/# (http://www.sthda.com/english/articles/36-classification-methods-essentials/150-stepwise-logistic-regression-essentials-in-r/#)>:~:text=Stepwise%20logistic%20regression%20consists%20of,best%20performing%20logistic%20regression%20model.

Hide

```
library("MASS")

best_step_lr = glm(V21 ~ ., data = train_gcd, family=binomial(link="logit")) %>%
  stepAIC(trace = FALSE)

summary(best_step_lr)
```

```
Call:
glm(formula = V21 ~ V1 + V2 + V3 + V4 + V5 + V6 + V8 + V9 + V10 +
    V14 + V15 + V20, family = binomial(link = "logit"), data = train_gcd)

Coefficients:
              Estimate  Std. Error z value         Pr(>|z|)
(Intercept)  1.78254468  0.83345133   2.139         0.032456 *
V1A12       -0.51192653  0.27377400  -1.870         0.061499 .
V1A13       -0.99724973  0.44774087  -2.227         0.025928 *
V1A14       -2.02695781  0.29142616  -6.955 0.00000000000352 ***
V2           0.03555048  0.01121431   3.170         0.001524 **
V3A31       -0.35300152  0.64137444  -0.550         0.582057
V3A32       -1.06661919  0.52065686  -2.049         0.040501 *
V3A33       -1.19514617  0.58905655  -2.029         0.042467 *
V3A34       -1.92511944  0.54853371  -3.510         0.000449 ***
V4A41       -2.12352954  0.44904614  -4.729 0.00000225652309 ***
V4A410      -2.62288095  1.13442775  -2.312         0.020774 *
V4A42       -1.13979305  0.32358458  -3.522         0.000428 ***
V4A43       -1.25566755  0.31317212  -4.010 0.00006084420675 ***
V4A44       -0.32965201  0.96071931  -0.343         0.731500
V4A45       -0.32382983  0.65297523  -0.496         0.619944
V4A46        0.44990936  0.52814398   0.852         0.394287
V4A48       -2.33292916  1.29215800  -1.805         0.071004 .
V4A49       -1.33058065  0.41744740  -3.187         0.001435 **
V5           0.00017622  0.00005279   3.338         0.000843 ***
V6A62       -0.38127919  0.34510328  -1.105         0.269235
V6A63       -0.49026111  0.53481737  -0.917         0.359306
V6A64       -1.08219422  0.62048064  -1.744         0.081138 .
V6A65       -1.36470399  0.34297261  -3.979 0.00006919218479 ***
V8           0.39541499  0.10897859   3.628         0.000285 ***
V9A92       -0.36635067  0.46682164  -0.785         0.432585
V9A93       -1.10987936  0.45803853  -2.423         0.015388 *
V9A94       -0.62053472  0.55348230  -1.121         0.262226
V10A102      0.51408660  0.50368050   1.021         0.307415
V10A103     -1.48214851  0.55157974  -2.687         0.007208 **
V14A142     -0.35826536  0.49880398  -0.718         0.472604
V14A143     -0.83606703  0.30060523  -2.781         0.005415 **
V15A152     -0.60724044  0.28968823  -2.096         0.036066 *
V15A153     -0.38421090  0.40961941  -0.938         0.348260
V20A202     -1.13395755  0.70228889  -1.615         0.106384
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 861.88  on 699  degrees of freedom
Residual deviance: 581.37  on 666  degrees of freedom
AIC: 649.37

Number of Fisher Scoring iterations: 5
```

Hide

```
predictions = predict(best_step_lr, newdata = test_gcd, type = "response")
predicted_classes = ifelse(predictions > 0.5, 1, 0)
predicted_classes = as.factor(predicted_classes)

# Evaluate the model
conf_matrix = table(test_gcd$V21, predicted_classes)

acc = sum(ifelse(predicted_classes == test_gcd$V21, 1, 0)) / nrow(test_gcd)

fp = conf_matrix[2,1]
tn = conf_matrix[1,1]

spec = tn / (fp + tn)

print(paste("Accuracy = ", acc, " Specificity = ", spec))
```

```
[1] "Accuracy =  0.746666666666667  Specificity =  0.802631578947368"
```

Hide

```
print(conf_matrix)
```

```
   predicted_classes
      0   1
  0 183  31
  1  45  41
```

By fitting the model through variable selection, I was able to improve accuracy and specificity. However, I need to increase specificty due to the second part of the question. I will do this by raising the threshold level by different increments.

Hide

```
thresh_grid = expand.grid(thresh =c(0.2,0.3,0.4,0.5,0.6, 0.7, 0.8, 0.9))

thresh_results = data.frame(thresh_val = integer(), spec = numeric(), acc = numeric())

for (i in 1:nrow(thresh_grid)) {

  thresh = thresh_grid[i, ]

  predicted_classes = ifelse(predictions > thresh, 1, 0)
  predicted_classes = as.factor(predicted_classes)

  # Evaluate the model
  conf_matrix = table(test_gcd$V21, predicted_classes)

  acc = sum(ifelse(predicted_classes == test_gcd$V21, 1, 0)) / nrow(test_gcd)

  fp = conf_matrix[2,1]
  tn = conf_matrix[1,1]

  spec = tn / (fp + tn)
  thresh_results = rbind(thresh_results, data.frame(thresh_val = thresh, spec = spec, acc = acc))
}

thresh_results
```

| thresh_val <dbl> | spec <dbl> | acc <dbl> |
|---|---|---|
| 0.2 | 0.8243243 | 0.6066667 |
| 0.3 | 0.8105263 | 0.6800000 |
| 0.4 | 0.7932692 | 0.6933333 |
| 0.5 | 0.8026316 | 0.7466667 |
| 0.6 | 0.7933884 | 0.7600000 |
| 0.7 | 0.7500000 | 0.7266667 |
| 0.8 | 0.7279152 | 0.7166667 |
| 0.9 | 0.7182131 | 0.7100000 |

8 rows

Hide

```
NA
```

Looks like a lower threshold improves specificity, which in return minimizes false positives.