STAT 328 Final Project

Mehmet Arslan

12/9/2021

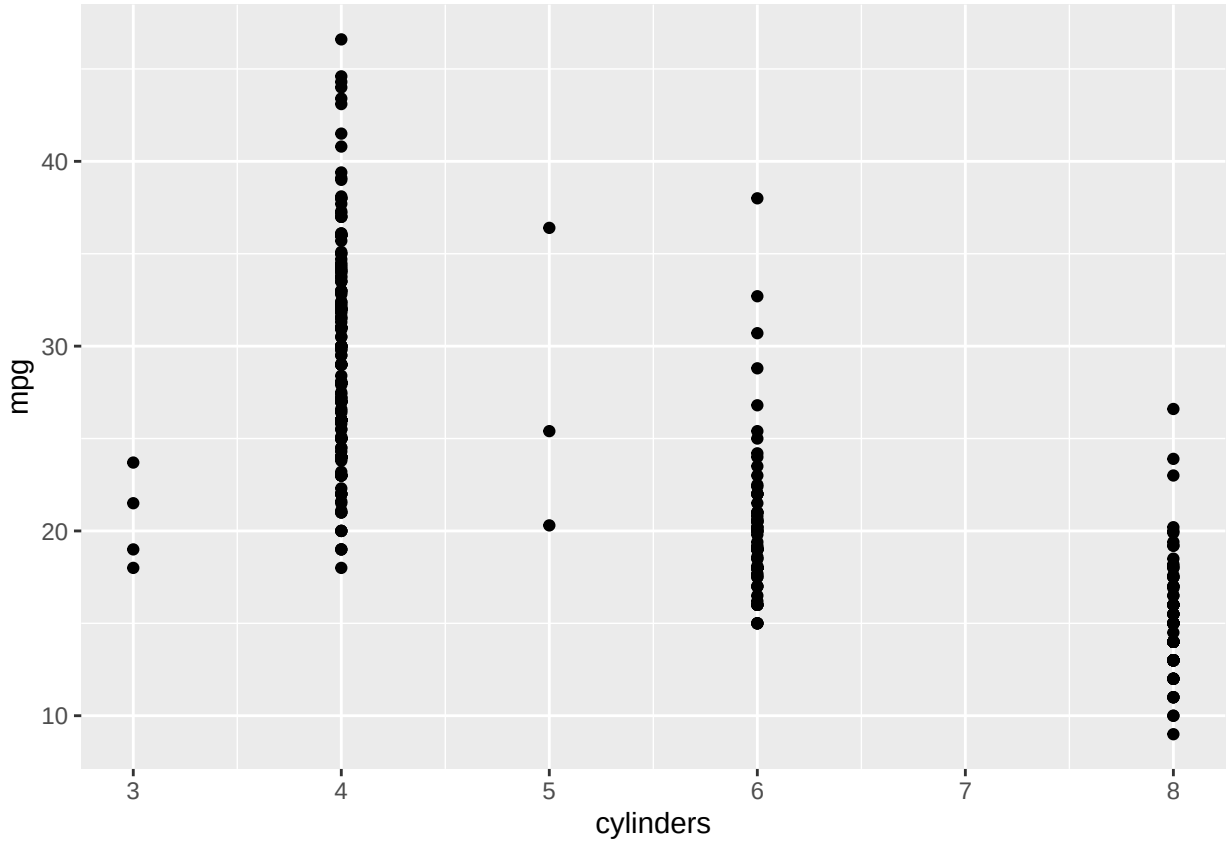
In this project, I will seek to employ the methods that I've been learning this semester. As someone who is interested in cars, and as someone who has been looking up how to program effectively online, I've stumbled across the dataset Auto, and the dataset mtcars quite often. Many of the programming guides and tutorials that I stumbled across mentioned these datasets and other similar ones when generating simple linear models.

The objective of this project is to demonstrate my understanding of vital machine learning topics such as: data preparation, exploratory data analysis, model building, and identifying significant variables within a dataset to name a few. As such, I will attempt to demonstrate my understanding of these non-trivial methods in the following paper.

Throughout this semester my professor, Dr. Kim, has instilled in me the importance of learning the the types of models to utilize, when to utilize them, and how to utilize them. As such, I have selected two datasets that are from the textbook "Intro to Statistical Learning in R Vol. 2". I have selected the dataset "Auto" as the dataset that I will be using for Linear Regression, and the dataset "Default" which I will use for Classification. In the following chunks I will attempt to demonstrate my understanding of not only predictive modeling, but also the entire data analysis process including but not limited to variable selection, correlation identification, etc.

```
library(tidyverse)
library(ggplot2)
library(caret)
library(caretEnsemble)
library(GGally)
library(randomForest)
library(kableExtra)
library(ISLR2)
library(car)
library(patchwork)
```

```
#attach(Auto)
Auto = Auto
Auto_tidy = Auto[, c(1:6)]
Auto %>% ggplot(aes(x = cylinders, y = mpg)) +
  geom_point()
```



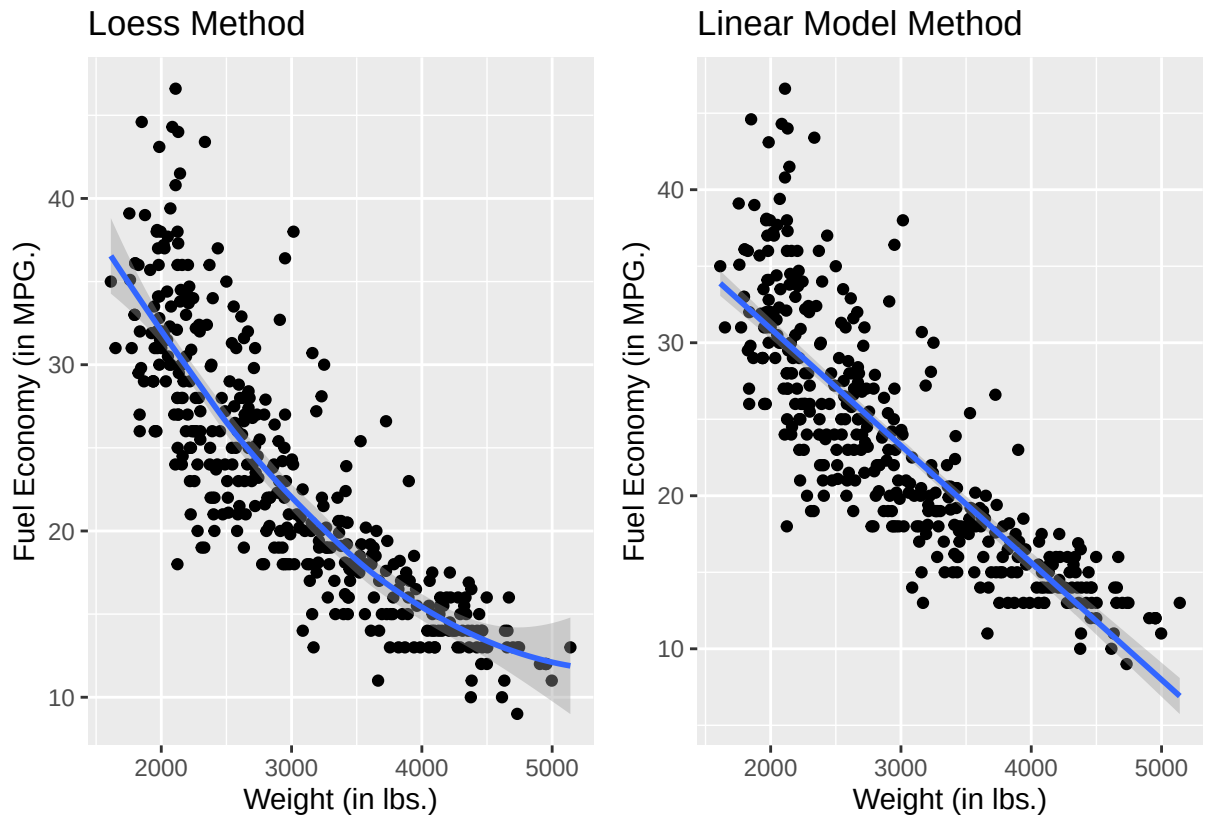
Notice how, in the graph generated, we have what appears to be a strange clustering of the data when selecting `cylinders` as our x variable. On the surface, this appears to be a classic case of a classification problem. For this reason, I will now attempt to generate the same plot using the variable `weight` as my x variable.

This is an important skill to master, learning how and when different models ought to be employed is potentially the largest part of our job as Data Scientists. Seeing this pattern only provides validity to just how important EDA, exploratory data analysis, is. Through the EDA process, one can interact with the data in a manner that is profoundly important.

Having established that, the problem that I am interested in answering is: "Does the weight of a car have an impact on its fuel efficiency?" I shall now try to demonstrate this in the following chunk:

```
A_loess = Auto %>%
  ggplot(aes(x = weight, y = mpg)) +
  geom_point() +
  geom_smooth(method = "loess") +
  labs(title = "Loess Method",
       x = "Weight (in lbs.)",
       y = "Fuel Economy (in MPG.)")
```

```
A_linear = Auto %>%
  ggplot(aes(x = weight, y = mpg)) +
  geom_point() +
  geom_smooth(method = "lm") +
  labs(title = "Linear Model Method",
       x = "Weight (in lbs.)",
       y = "Fuel Economy (in MPG.)")

A_loess + A_linear
```



When generating the graphs, I added the geom_smooth() function with two different arguments: `method = "lm"` and `method = "loess"`. The graph on the left is the loess method and the graph on the right is the linear model method. Through my years of learning about R-Code and how these programs work, I learned the importance of error, estimation and what those things actually mean. Notice how in the Loess method, the gray bar surrounding the line of best fit, seems to be increasing quite dramatically when approaching the ends of the data. This is because the `loess` method is fitting a quadratic curve or a polynomial curve to the data. This has a tendency to favor tightly clustered data and then performs poorly everywhere else. This is an example of an awful method. Also, note that the `lm` method has significantly less error in estimation toward the ends and still tends to fit *most* of the data within it. These two graphs indicate that the best possible solution for predictive modeling is the `lm` method or simply Linear Regression.

Recall that the linear model is simply:

$$Y_i = \hat{\beta}_0 + \hat{\beta}_1 x_i + \varepsilon_i$$

Where Y_i is the response variable, $\hat{\beta}$ is the coefficient, and x_i is the predictor variable. It makes sense that this is what is known as a simple linear model, wherein there is only 1 predictor variable. These models can perform well, but the model's performance is heavily dependent on the amount of predictors that we have in a dataset. (Chatterjee, 2019)

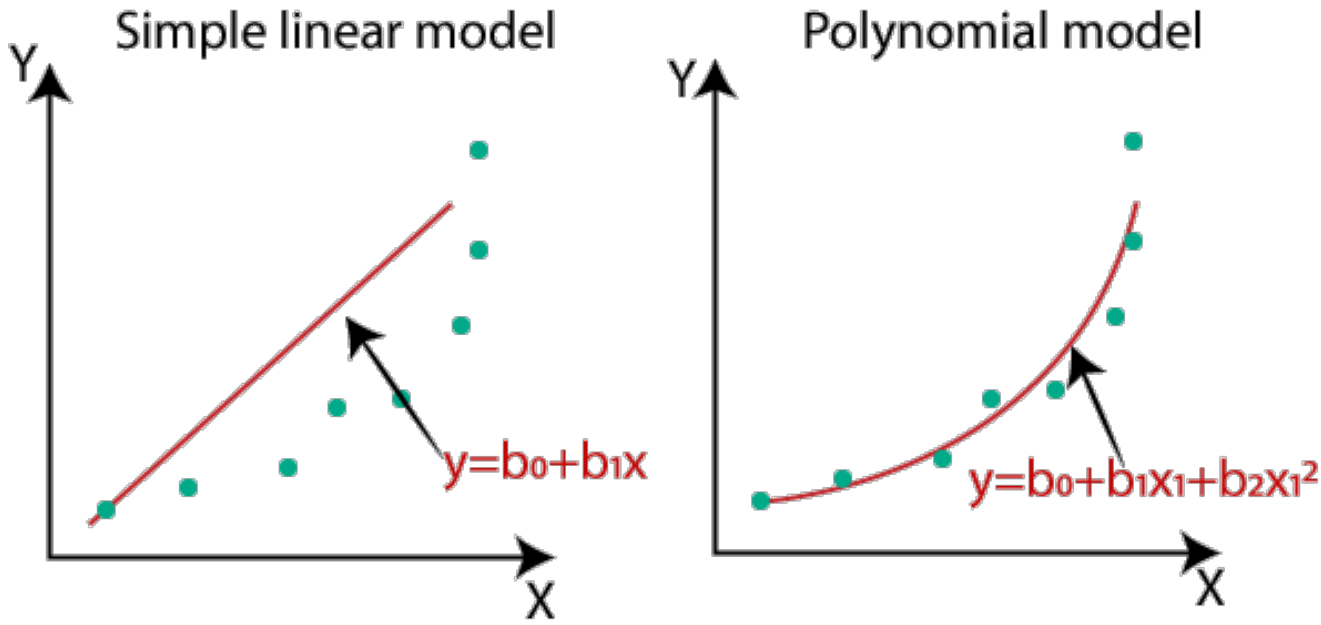
Admittedly, I still have a lot to learn about how mathematical functions interact with data, but I do know that polynomial curves tend to perform better for clustered data, and linear models tend to perform better when the data is more spread out. (James et al., 2021/2013)

Polynomial regression is given by:

$$Y_i = \hat{\beta}_0 + \hat{\beta}_1 x_i + \hat{\beta}_2 x_i^2 + \hat{\beta}_3 x_i^3 + \varepsilon_i$$

Where Y_i is the response variable, $\hat{\beta}$ is the coefficient, and x_i is the predictor variable. Now, notice that this model has 3 predictor variables, and thus has become a polynomial regression curve. While this method can minimize errors around clusters, as per the nature of a polynomial curve, it can still perform poorly overall. Meaning that there could be accuracy issues the further we go from the origin of a graph of certain data.

Observe the differences between the two models in a more conceptual way (Abhigyan, 2020):



With polynomial regression, one can clearly see how the polynomial function performs better overall and tends to perform really well with clustered data!

The method I will employ next is simply to build a simple linear model that is using all the variables except `mpg` as predictors.

```
set.seed(100)
lm = lm(mpg ~ ., data = Auto_tidy)
summary(lm)
```

```
##
## Call:
## lm(formula = mpg ~ ., data = Auto_tidy)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -11.5816  -2.8618  -0.3404   2.2438  16.3416
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)   4.626e+01  2.669e+00  17.331   <2e-16 ***
## cylinders    -3.979e-01  4.105e-01  -0.969   0.3330
## displacement -8.313e-05  9.072e-03  -0.009   0.9927
## horsepower   -4.526e-02  1.666e-02  -2.716   0.0069 **
## weight       -5.187e-03  8.167e-04  -6.351    6e-10 ***
## acceleration -2.910e-02  1.258e-01  -0.231   0.8171
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.247 on 386 degrees of freedom
## Multiple R-squared:  0.7077, Adjusted R-squared:  0.7039
## F-statistic: 186.9 on 5 and 386 DF,  p-value: < 2.2e-16
```

```
vif(lm)
```

```
##    cylinders displacement   horsepower       weight acceleration
##    10.630870    19.535061     8.916017    10.430271     2.609487
```

Importantly, I saw that the residuals of my model is normally distributed. Moreover, this was the case with other models tried as well, where I used all variables, and then just the variables that I was interested in. When performing VIF, I noticed that the predictor variables weight and cylinders are almost identical in terms of VIF while the variable displacement has a really high VIF.

In the summary of the overall model, there were only 3 cases where the model was significant: the null-model, the model with only cylinders, and the model with weight. It turns out that weight is equally as important as cylinders in terms of the mpg that one can expect out of these cars! Suppose then, that I wanted to predict the fuel economy of a car weighing 2,000 lbs. and having 6 cylinders:

```
(mpg_2500 = 46.26 + (-0.0051*(2000) + -0.3979*(6)))
```

```
## [1] 33.6726
```

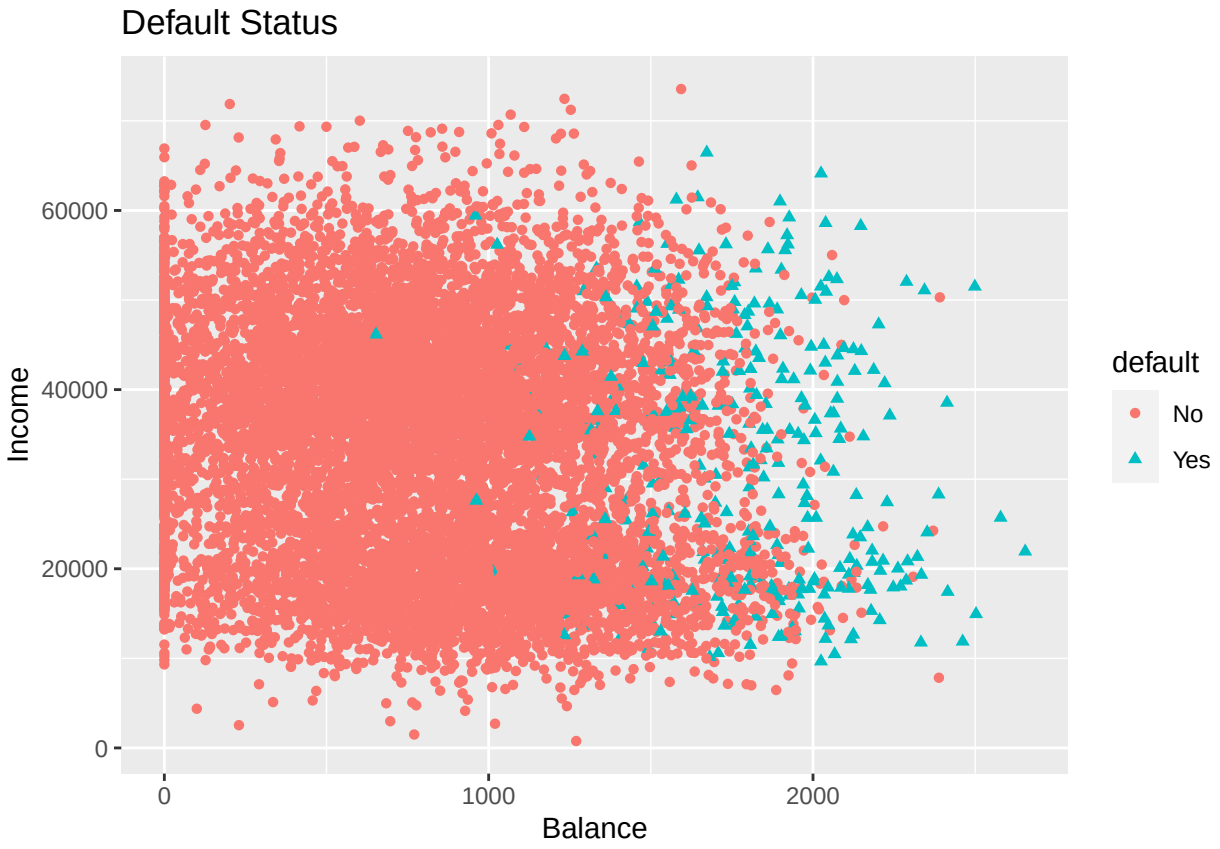
Notice how, even with the linear model method, the output generated is nowhere near close enough to actual outcome! I can speculate that this is caused by the serious variance in the data as well as the clustering and the overall shape of how the data points themselves are so far apart from the regression line. I can further speculate that because our *best* predictor variable `cylinders` is categorical, that there is indeed a method to be employed to rectify the poorly performing model. Although using categorical predictor variables can be a nuisance, we can rectify these issues with programming in "dummy variables" and then using those as our

main methods to perform the predictive modeling. (Zhang & Wang, 2016)

As you can see by the predictive model that I built, considering just those two variables generates quite bizarre predictions. This is due to the point that the data is following a trend, but it is following it too weakly to make accurate and meaningful predictions out of. This is the main limitation of linear models for predictive model building as there isn't really a way that the linear model can correct the trends issues. (Machine Learning: Linear Regression and its applications, 2021)

As for the classification method, I have selected the dataset `Default` which is available in the ISLR2 package also. We have gone over predictive modeling of this dataset in the Chapter 4 lab, I have decided that I will continue on in this manner, but using a Random Forest for Classification as my method. The reason I selected the Random Forest is because, when working through Homework 7, I found that the Random Forest model performed quite well. In terms of the performance of this model's performance, it performed well in important metrics such Specificity, sensitivity, and F-1 score to list a few.

```
Default = Default

ggplot(Default, aes(x = balance, y = income,
                    shape = default, colour = default)) +
  geom_point() +
  labs(color = "default", title = "Default Status",
       x = "Balance", y = "Income")
```



Code for the above chunk is coming from Chapter 4 lab that we performed in class.

It seems nearly impossible to tell where the students are and where the categorical response variable is and where the categorical predictor is at any given moment. This creates a unique challenge for us as Data Scientists, we must be able to employ methods and tactics that address issues like these and can make the difference between a meaningful interpretation and complete nonsense.

When going over these predictive models in Chapter 4, we learned that there are several important and very useful ways of modeling categorical data. In fact, the lab of Chapter 4 was tremendously eye-opening. I used the code generated by Dr. Kim which made our life considerably easier! I decided that I would continue on with the `Default` dataset but employ a new method I learned through the course of completing Homework 7, Random Forests.

I decided on Random Forests as their methods are rather simple and the default parameters in the 'randomForest' function have proven to be more than adequate for our purposes. I cannot emphasize this enough however, just because I managed to get the code to work is simply not enough. There is a difference between truly *understanding* a topic and *working* with a topic.

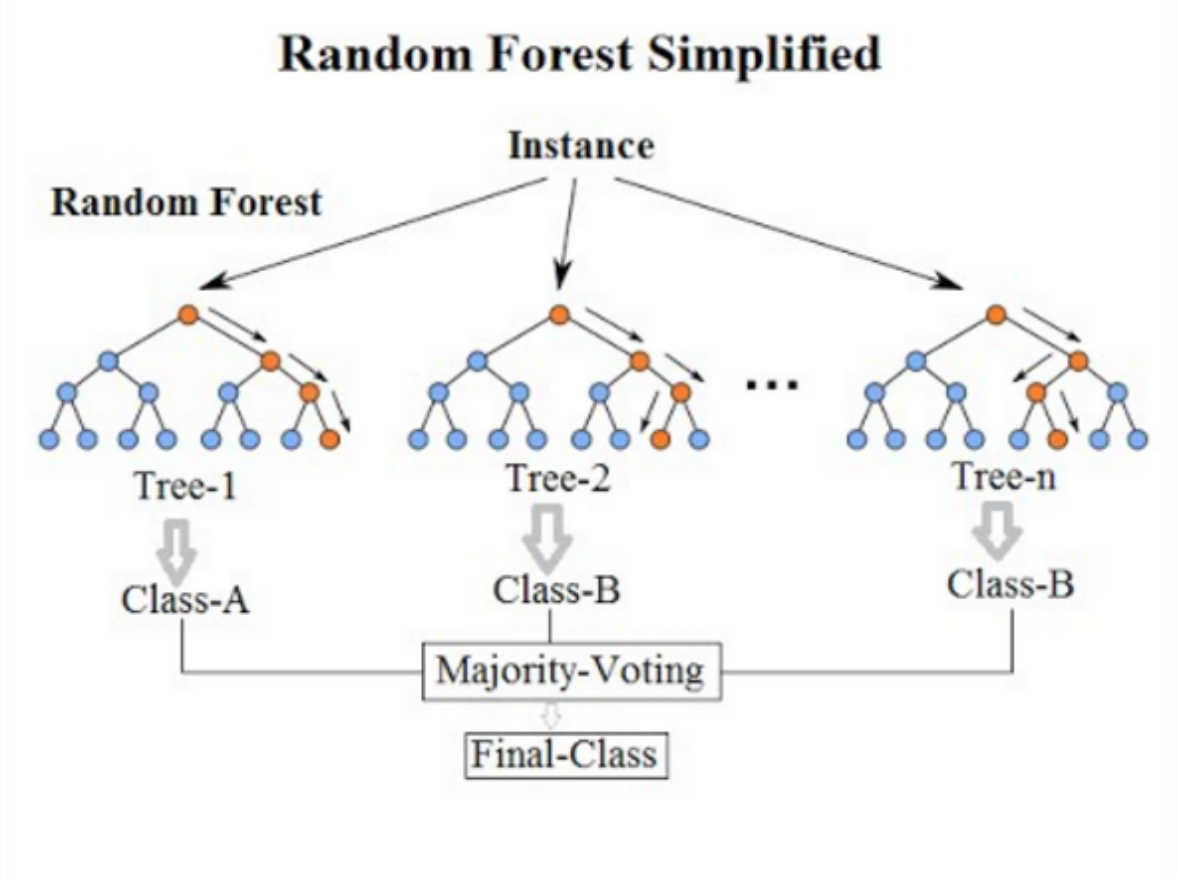
```
set.seed(34859)
{nr <- nrow(Default)
tr.id <- sample(nr, floor(0.7*nr), replace = F,
                prob = NULL)
tr.dat <- Default[tr.id,]
ts.dat <- Default[-tr.id,]}

random_forest = randomForest(factor(default) ~ .,
                             data = tr.dat, mtry = 2,
                             ntrees = 100, importance = T)

random_forest_preditcion = predict(random_forest,
                                   newdata = ts.dat,
                                   ntrees= 100)

table(random_forest_preditcion, ts.dat$default)
```

```
##
## random_forest_preditcion   No  Yes
##                       No  2871   71
##                       Yes   17   41
```

```
cm_proj = confusionMatrix(factor(random_forest_preditcion), factor(ts.dat$default),
                          mode = "everything")
```

The Random Forest method is an interesting one, because it lays out a simple logical flow for the machine to follow. The machine will make the predictions that we ask for based on a concept known as a decision tree. This is similar to both form and function to a logic gate. The Random Forest Method also helps to minimize errors. Moreover, we can finely tune the forest's parameters to generate *even* better fitting results. Parameters for Random Forests, and decision trees in general, help to apply "bounds" and direction to the branches of the decision tree. Hyperparameters are similar and they are the main philosophy as to how a Random Forest is able to generate such meaningful and accurate results.

Observe the ways in which a Random Forest functions (Koehrsen, 2020):



Decision trees do work similarly to logic gates, although we have the option to fine tune the interactions between how deep we want the machine to go in to each branch, and even how many trees we want to utilize in our predictive modeling. (Koehrsen, 2020)

```
Accuracy_rf = round((cm_proj$overall[1]), 2)

Others_rf = round(rbind(cm_proj$byClass[c(1:4,7)]), 2)

tab = cbind(Accuracy_rf, Others_rf)
rownames(tab) = NULL

kbl(tab, booktabs = T)
```

Accuracy_rf	Sensitivity	Specificity	Pos Pred Value	Neg Pred Value	F1
0.97	0.99	0.37	0.98	0.71	0.98

Through these analysis problems, I learned that I still have a lot left to learn. Machine Learning is not an easy concept but it seems to be a little more understandable.

My findings in this project were surprising, but not by much. In the 'Auto' dataset, I expected that any simple linear regression model would perform very poorly and I was right. In Data Science we have a process of model selection. We list out the models that perform the best, and from those models we select the best one overall. There is however, a model called the Null-Model which is a model that uses *none* of the predictors in a dataset. Incidentally, the model with the $p-value$ that was the most significant, was the null-model. This of course implies that the predictor variables selected performed so poorly that they tend to skew the results of the linear model in unwanted manners and generate inaccurate results, as exemplified when pulling

the coefficients and using them to generate a simple model.

My findings for the classification problem, in the ‘Default‘ dataset were unsurprising to say the least. The Random Forest prediction model correctly predicted the right classification 97% of the time! This was astonishing! Of course it was able to generate this high of an accuracy with the testing data that was generated from the original dataset as well. A truly remarkable feat would be to see repeated accuracy of that model with an entirely new dataset!

Programming Language Used:

- R - 4.1.2
- RStudio - 1.4

Packages Used:

- dplyr - 0.7.8 allows the user to wrangle/prepare data and also allows for use of the pipe operator for passing an object to a function.
- tidyverse - 1.3.1 allows the user to use packages such as ggplot2 which is an excellent data visualization tool.
- caret - 9.0 allows the user to build confusion matrices that can report all important model metrics such as accuracy and specificity, etc.
- kableExtra - 1.3.4 allows the user to generate html or latex tables.
- GGally - 1.5.0 allows the user to generate the correlation plots of all predictors and the response variables in a dataset
- ggplot2 - 3.3.5 allows the user to finely control the methods of data visualization as well as fine-tuning graphs to make them look better and allows the user to assign labels, text, titles, select color based on variables, etc.
- randomForest - 4.6-14 allows the user to use the randomForest function but with high levels of hyperparameter tuning such as interaction depth, number of trees, types of outputs, etc.
- ISLR2 - 1.3 Contains certain functions and datasets curated by the team of authors that published "Into to Statistical Learning in R vol 2," which happens to be the required textbook for this course.
- car - 3.0-12 contains the function vif that allows the user to calculate the variable inflation factor of variables in a linear model.
- patchwork - 1.1.1 allows the user to combine multiple ggplot objects into a single output, making side by side comparison a lot simpler. This package also gives the user the ability to generate different types of plots and compare them directly.
- caretEnsemble - 2.0.1 allows the user to use the same re-sampling techniques for multiple caret models.

Works Cited:

Abhigyan. (2020, August 2). Understanding Polynomial Regression!!! Medium. https://medium.com/analytics-vidhya/understanding-polynomial-regression-5ac25b970e18

Chatterjee, D. R. (2019, August 26). All the Annoying Assumptions. Medium. https://towardsdatascience.com/all-the-annoying-assumptions-31b55df246c3

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). An introduction to statistical learning : with applications in R (second). Springer. (Original work published 2021)

Koehrsen, W. (2020, August 18). Random Forest Simple Explanation. Medium. https://williamkoehrsen.medium.com/random-forest-simple-explanation-377895a60d2d

Machine Learning: Linear Regression and its applications. (2021, September 7). The Data Science Portal. https://thedatascienceportal.com/posts/linear-regression-and-its-applications/

Zhang, Z., & Wang, L. (n.d.). Regression with categorical predictors – Advanced Statistics using R. Advstats.psychstat.org. Retrieved December 9, 2021, from https://advstats.psychstat.org/book/mregression/catpredictor.php