IST707 Final Project - Customer Segmentation

Paul Strader, Paul Phillips, Samuel Yohannes, Adam Vajdak

2022-12-14

# Introduction

Customer Segmentation is one of the most valuable tools a data literate company has. When deployed effectively, customer segmentation can help marketing departments more efficiently allocate promotional spend and make better decisions regarding which media channels to use in order to advertise to their highest value customers. Segmentation can also help companies make informed decisions about executing price changes at a sub-state level, giving them significant leverage over competitors executing pricing strategies at the state, regional, or national level. Customer segmentation can also be used by R&D departments to develop new products or services to meet the needs of emerging market segments or high value existing segments and can be used by sales planning departments to determine which retail customers should receive a new product first. Other modeling techniques can help organizations make data driven decisions providing high return on investment and large profitability increases, but customers segmentation is the foundation for effective and strategic deployment of resources in a corporate environment.

This analysis will evaluate a data set, sourced from Kaggle, in which an automobile manufacturer has used customer segmentation to group their existing customers into four different customer profiles (Segments A, B, C, D). Our analysis will evaluate several different modeling techniques to better understand the decisions the company made in their existing customer segmentation and to find an accurate way to assign new customers to their existing market profiles. If successful, we will find a modeling technique that segments the existing customers into four groups following a distribution similar to the company’s attempt at segmentation and will apply that model effectively to the new customer database. Our goal is to determine which of the following modeling techniques accurately predicts the existing customer segment: Association Rule Mining, k nearest neighbor, decision trees, random forest, XGBoost, and SVM.

# About the Data

Our data set comes from [Kaggle](https://www.kaggle.com/datasets/kaushiksuresh147/customer-segmentation?select=Train.csv) and contains two eleven dimensional subsets: a Training data set made up of about 8,000 existing customer records, and a test data set made up of around 2,600 new customer records. Both subsets of the data contain the following attributes (data table sourced from Kaggle):

|  |  |
| --- | --- |
| Variable | Definition |
| ID | Unique ID |
| Gender | Gender of the Customer |
| Ever\_Married | Marital Status of the customer |
| Age | Age of the Customer |
| Graduated | Is the customer a graduate? |
| Profession | Profession of the Customer |
| Work\_Experience | Work Experience in years |
| Spending\_Score | Spending score of the customer |
| Family\_Size | Number of family members for the customer (including the customer) |
| Var\_1 | Anonymised Category for the customer |
| Segmentation | (target) Customer Segment of the customer |

# Load Data & Data Cleaning

data <- read.csv('/Users/paulstrader/Desktop/archive-6/Train.csv', na.string = c(""))  
  
#str(data)  
  
summary(data)

## ID Gender Ever\_Married Age   
## Min. :458982 Length:8068 Length:8068 Min. :18.00   
## 1st Qu.:461241 Class :character Class :character 1st Qu.:30.00   
## Median :463472 Mode :character Mode :character Median :40.00   
## Mean :463479 Mean :43.47   
## 3rd Qu.:465744 3rd Qu.:53.00   
## Max. :467974 Max. :89.00   
##   
## Graduated Profession Work\_Experience Spending\_Score   
## Length:8068 Length:8068 Min. : 0.000 Length:8068   
## Class :character Class :character 1st Qu.: 0.000 Class :character   
## Mode :character Mode :character Median : 1.000 Mode :character   
## Mean : 2.642   
## 3rd Qu.: 4.000   
## Max. :14.000   
## NA's :829   
## Family\_Size Var\_1 Segmentation   
## Min. :1.00 Length:8068 Length:8068   
## 1st Qu.:2.00 Class :character Class :character   
## Median :3.00 Mode :character Mode :character   
## Mean :2.85   
## 3rd Qu.:4.00   
## Max. :9.00   
## NA's :335

The majority of the columns are currently either numeric or character. Based on the nature of this dataset, all of these variables will need to be converted to nominal data types. *Age* and *work experience* should furthermore be translated to data type ordinal as order here matters. *Ever Married* could be left as a nominal variable, but will be converted Boolean since the values already consist of *true* and *false* responses (stored in the yes or no format). The *Graduated* column also consists of true or false responses, so this attribute will be converted to type Boolean as well. The *ID* variable should also be removed as this will only create noise in the later analysis. The values of the ID column is unique at the record level; thus, contributes nothing to the underlying trends in the dataset.

# Display missing values by column  
for (col in colnames(data)) {  
 print(  
 sprintf(  
 'Missing value percent for %s: %s%%',   
 col,  
 round((sum(is.na(data[,col]))/nrow(data)),3)\*100  
 )  
 )  
}

## [1] "Missing value percent for ID: 0%"  
## [1] "Missing value percent for Gender: 0%"  
## [1] "Missing value percent for Ever\_Married: 1.7%"  
## [1] "Missing value percent for Age: 0%"  
## [1] "Missing value percent for Graduated: 1%"  
## [1] "Missing value percent for Profession: 1.5%"  
## [1] "Missing value percent for Work\_Experience: 10.3%"  
## [1] "Missing value percent for Spending\_Score: 0%"  
## [1] "Missing value percent for Family\_Size: 4.2%"  
## [1] "Missing value percent for Var\_1: 0.9%"  
## [1] "Missing value percent for Segmentation: 0%"

There are some values missing in family size and work experience. Since the percentage is quite low, it may be worth considering imputing the missing values in order to keep the records.

## Clean Data

### Remove ID column

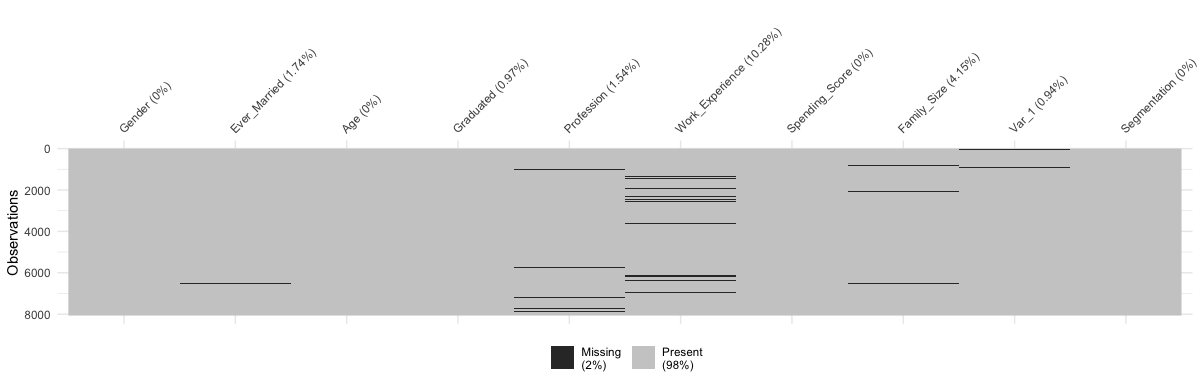
data\_noID <- subset(data, select = -c(ID))

### Explore Relationship of Missing Data

A great first plot for exploring missingness is the *vis\_miss* function from the *naniar* package. This plot leverages the rows, as well as columns impacted by the missing data to yield a 2D summary.

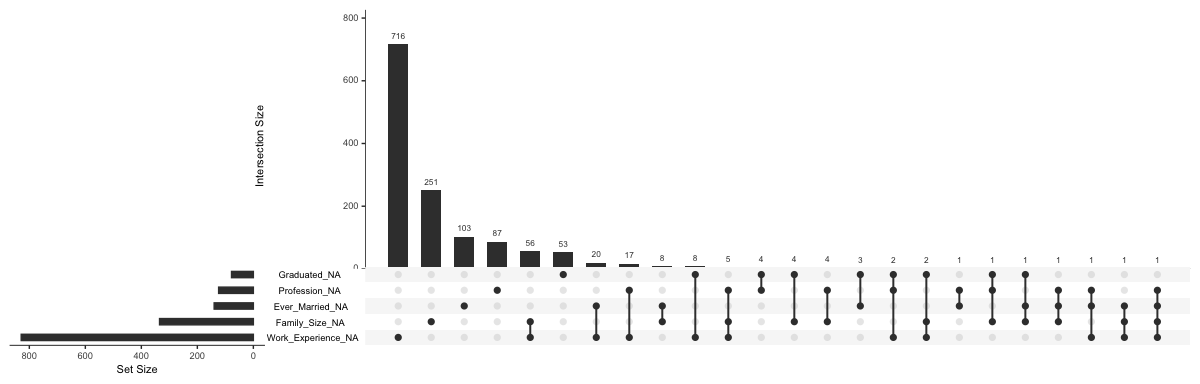
vis\_miss(data\_noID)

## Warning: `gather\_()` was deprecated in tidyr 1.2.0.  
## Please use `gather()` instead.  
## This warning is displayed once every 8 hours.  
## Call `lifecycle::last\_lifecycle\_warnings()` to see where this warning was generated.



It would appear that there could be some correlation between the missing instances in *work experience* and *family size*. For missing data across all other columns it appears they are missing completely at random (MCAR). Next, the missingness will be further explored for possible correlations across the features impacted by such. This will be accomplished using the *UpSetR* package, which visualizes patterns (or combinations) of missingness across customer records.This tool highlights points at which missingess intersects between features at high frequencies. Having such occur would indicate that by simply removing missing values could lead to elimination of present subgroups in the data in the later modeling phase. For example, it could be the younger customers which parents are purchasing the vehicle for that are most commonly missing such information.That is, young adults who are not yet working nor have families of their own would not have information available for such attributes. By eliminating such instances (if such were discovered), the younger population could be under represented (or entirely missed) in the model and analysis phase. Essentially, it is important to verify next whether the missing values of these two features are indeed correlated or not. Cases of correlated missingness between two observed values are referred to as *missing at random* or MAR.

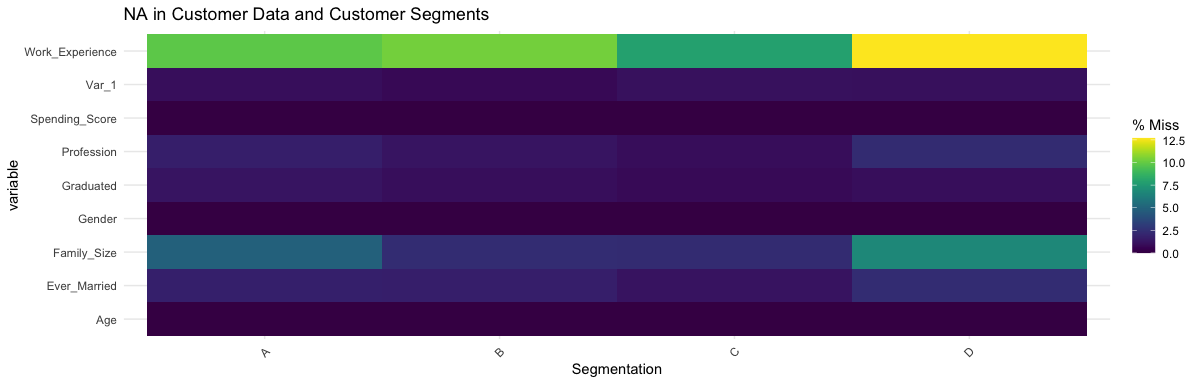
gg\_miss\_upset(data\_noID)

 The following results indicate that there is not strong presence of correlated missing data. The strongest correlation of missing data is indeed between family size and work experience, but this intersection only represent **56** of the total records (e.g. **<1%** of the training data). It would seem missing data across the columns are primarily independent events from one another. The rationale for work experience consisting of the most missing data could be that this is the most frequent question opted out by customers.

### Missing Values Across Customer Segments

The key stakeholders of this project may wish to know how dropping missing data could impact information available for the given customer segments and archetypes collected by their marketing team. This information is expense to collect and explore so it is important these attributes can still be properly represented the later phases of analysis.

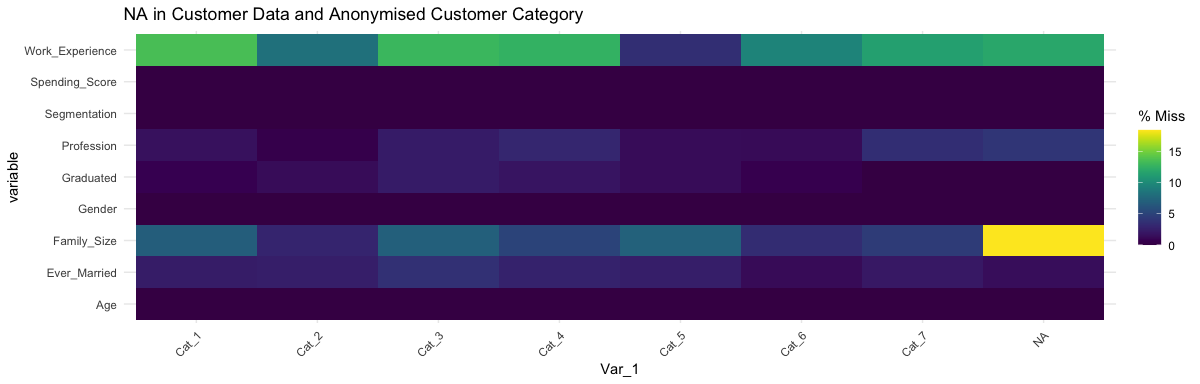
# Explore data mechanisms and relationships with respect to missing data  
gg\_miss\_fct(  
 x=data\_noID,  
 fct=Segmentation  
) +  
labs(  
 title = 'NA in Customer Data and Customer Segments'  
)



Although the *customer segment D* has the most records missing work experience, it is not by much with respect to the other segments. For the most part, work experience is consistently between **5-10%** across the customer segments.

### Missing Values Across Anonymised Categories assigned to Customers

# Explore data mechanisms and relationships with respect to missing data  
gg\_miss\_fct(  
 x=data\_noID,  
 fct=Var\_1  
) +  
labs(  
 title = 'NA in Customer Data and Anonymised Customer Category'  
)



There seems to be less uniformity of missing data across the customer archetypes but not by much (largest differences being roughly 5% in frequency of occurrence). *Category 5* of customers is clearly impacted the least by the missing values.

### Remove Missing Values

For this project we’ll just be removing NA’s.

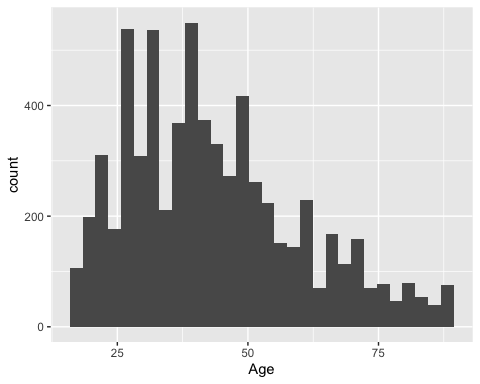
data\_noID <- na.omit(data\_noID)  
print(  
 sprintf(  
 "Lost %s%% of the total dataset.",  
 round(1 - nrow(data\_noID)/nrow(data),3)\*100  
 )  
)

## [1] "Lost 17.4% of the total dataset."

### Explore Distribution of Age present in Dataset

data\_noID %>% dplyr::select(Age) %>% ggplot(aes(x=Age)) + geom\_histogram()

## `stat\_bin()` using `bins = 30`. Pick better value with `binwidth`.

 There appears to be several peaks present in the dataset with respect to disrtibution of age, so quantile ranking may smooth the underlying trends in the dataset too much. Instead, decile ranking will be applied.

### Decile Rank Age Attribute Values

It is important age is represented as an ordinal attribute for EDA aspect of this analysis (visually displays better). Additionally to this, as means of smoothing some of the noise present in the age attribute, a decile ranking will be performed on the attribute. This will cut the column into **10** bins, represented by an integer. From there, min & max ranges will be determined at each quantile level and fitted as a label in the final dataset.

# Note: age will be ranked in ascending order, that way the higher age levels will be represented by higher quantile values  
data\_noID <- mutate(  
 data\_noID,  
 ageRanked = ntile(data\_noID$Age, 10)  
)  
  
# Convert to ordered nominal  
data\_noID$ageRanked <- ordered(data\_noID$ageRanked)  
  
# Explore distribution of ageRanked -> determine age ranges each represents  
data\_noID %>%   
 dplyr::select(ageRanked, Age) %>%   
 group\_by(ageRanked) %>%   
 dplyr::summarize(  
 RowCount = n(),   
 MinAge=round(min(Age)),  
 MaxAge=round(max(Age))  
 )

## # A tibble: 10 × 4  
## ageRanked RowCount MinAge MaxAge  
## <ord> <int> <dbl> <dbl>  
## 1 1 667 18 25  
## 2 2 667 25 29  
## 3 3 667 29 33  
## 4 4 667 33 37  
## 5 5 667 37 41  
## 6 6 666 41 45  
## 7 7 666 45 50  
## 8 8 666 50 57  
## 9 9 666 57 68  
## 10 10 666 68 89

# Note: the created categories will be easier to understand by actual rank values  
# update labels to represent these ranges  
  
# Create min & max ranges of each decile range  
age\_dRange\_labels <-   
 data\_noID %>%   
 dplyr::select(ageRanked, Age) %>%   
 group\_by(ageRanked) %>%   
 dplyr::summarize(  
 RowCount = n(),   
 MinAge=round(min(Age)),  
 MaxAge=round(max(Age))  
 ) %>%   
 dplyr::select(MinAge, MaxAge) %>%   
 as.data.frame(c(MinAge, MaxAge))  
  
# Create new column for eventual label  
age\_dRange\_labels$ageLabel <-   
 paste(  
 as.character(age\_dRange\_labels[,1]),  
 '-',  
 as.character(age\_dRange\_labels[,2]),   
 sep=''  
 )  
  
# Display results  
age\_dRange\_labels$ageLabel

## [1] "18-25" "25-29" "29-33" "33-37" "37-41" "41-45" "45-50" "50-57" "57-68"  
## [10] "68-89"

# Map new labels to existing age ranked column  
# That is, modify the existing ordinal column  
data\_noID$age\_dRanked\_Bins <- factor(  
 data\_noID$ageRanked,  
 labels=age\_dRange\_labels$ageLabel,  
 ordered=TRUE  
)  
  
# Note: the ordinal column created using decile ranking is still preserved,  
# but now the user-facing values are represented by the actual age range  
# within each decile represents in the dataset  
min(data\_noID$age\_dRanked\_Bins)

## [1] 18-25  
## 10 Levels: 18-25 < 25-29 < 29-33 < 33-37 < 37-41 < 41-45 < 45-50 < ... < 68-89

# Drop Age  
# Need ageRanked for conversion back to ordinal upon import later on!  
data\_noID <- subset(data\_noID, select = -c(Age))

### Convert Graduated & Ever Married to Type Boolean

# Convert Graduated  
data\_noID$IsGraduated <- FALSE  
  
# Convert missing values back to blanks -> apparently can't apply joint filters on columns with NA's in R?  
data\_noID[  
 is.na(data\_noID$Graduated),  
 "Graduated"  
] <- ''  
  
# Note: NULL's are present; convert those to FALSE as well  
# since we don't know the answer  
data\_noID[  
 (data\_noID$Graduated == 'Yes' & data\_noID$Graduated != '' ),  
 'IsGraduated'  
] <- TRUE  
  
# Convert Ever Married  
data\_noID$EverMarriedTRUE <- FALSE  
  
# Convert missing values back to blanks -> apparently can't apply joint filters on columns with NA's in R?  
data\_noID[  
 is.na(data\_noID$Ever\_Married),  
 "Ever\_Married"  
] <- ''  
  
# Note: NULL's are present; convert those to FALSE as well  
# since we don't know the answer  
data\_noID[  
 (data\_noID$Ever\_Married == 'Yes' & data\_noID$Ever\_Married != '' ),  
 'EverMarriedTRUE'  
] <- TRUE  
  
# Drop originals  
data\_noID <- subset(data\_noID, select = -c(Graduated, Ever\_Married))

### Convert Work Experience & Family Size to Ordinal

# Convert to Ordinal  
data\_noID$Work\_Experience <- ordered(data\_noID$Work\_Experience)  
  
data\_noID$Family\_Size <- ordered(data\_noID$Family\_Size)

### Convert Everything Else to Nominal

# First grab all colnames NOT including the ones that have just been modified  
target\_cols <- names(  
 subset(  
 data\_noID,  
 select = -c(  
 age\_dRanked\_Bins,   
 Work\_Experience,  
 EverMarriedTRUE,  
 ageRanked,  
 IsGraduated,  
 Family\_Size  
 )  
 )  
)  
  
# Iterate through columns  
for (col in target\_cols) {  
 # Convert column to nominal  
 data\_noID[,col] <- factor(  
 data\_noID[,col]  
 )  
}  
  
# Inspect outcome  
glimpse(data\_noID)

## Rows: 6,665  
## Columns: 11  
## $ Gender <fct> Male, Female, Male, Male, Male, Female, Female, Femal…  
## $ Profession <fct> Healthcare, Engineer, Lawyer, Artist, Healthcare, Hea…  
## $ Work\_Experience <ord> 1, 1, 0, 0, 1, 1, 0, 1, 1, 4, 0, 1, 9, 1, 1, 0, 12, 3…  
## $ Spending\_Score <fct> Low, Low, High, Average, Low, Low, Low, Average, Low,…  
## $ Family\_Size <ord> 4, 1, 2, 2, 3, 3, 3, 4, 3, 4, 1, 2, 5, 6, 4, 1, 1, 4,…  
## $ Var\_1 <fct> Cat\_4, Cat\_6, Cat\_6, Cat\_6, Cat\_6, Cat\_6, Cat\_7, Cat\_…  
## $ Segmentation <fct> D, B, B, C, C, D, D, C, A, D, B, C, D, B, B, C, A, D,…  
## $ ageRanked <ord> 1, 9, 9, 8, 3, 3, 9, 8, 2, 1, 9, 5, 3, 3, 9, 10, 7, 1…  
## $ age\_dRanked\_Bins <ord> 18-25, 57-68, 57-68, 50-57, 29-33, 29-33, 57-68, 50-5…  
## $ IsGraduated <lgl> FALSE, TRUE, TRUE, FALSE, TRUE, TRUE, TRUE, TRUE, TRU…  
## $ EverMarriedTRUE <lgl> FALSE, TRUE, TRUE, TRUE, FALSE, FALSE, TRUE, TRUE, FA…

Now that we are confident in the cleanliness of our data set, we will begin our exploratory analysis.

## Exploratory Analysis

We begin by creating a user defined function that applies some additional cleaning steps to the data to facilitate our exploratory analysis.

clean\_data\_cols <- function(data) {  
   
 # Clean Age Bins  
 data[,'age\_dRanked\_Bins'] <- factor(  
 data[,'ageRanked'],  
 ordered = TRUE,  
 labels = c(  
 "18-25",   
 "25-29",   
 "29-33",   
 "33-37",   
 "37-41",   
 "41-45",  
 "45-50",  
 "50-57",  
 "57-68",  
 "68-89"  
 )  
 )  
   
 # Clean Other Ordinal Columns  
 data[,'Work\_Experience'] <- ordered(data[,'Work\_Experience'])  
 data[,'Family\_Size'] <- ordered(data[,'Family\_Size'])  
   
 # Note: for exploratory work treat Spending Score as ordinal  
 data[,'Spending\_Score'] <- factor(  
 data[,'Spending\_Score'],  
 levels=c('Low','Average','High'),  
 ordered=TRUE  
 )  
   
 # Clean All Nominal Columns  
   
 # First grab all colnames NOT including the ones that have just been modified  
 target\_cols <- names(  
 subset(  
 data,  
 select = -c(  
 age\_dRanked\_Bins,   
 Work\_Experience,  
 EverMarriedTRUE,  
 ageRanked,  
 IsGraduated,  
 Family\_Size,  
 Spending\_Score  
 )  
 )  
 )  
   
 # Iterate through columns  
 for (col in target\_cols) {  
 # Convert column to nominal  
 data[,col] <- factor(  
 data[,col]  
 )  
 }  
   
 return(data)  
}  
  
# Convert columns to expected types  
edaDF <- clean\_data\_cols(data\_noID)  
  
glimpse(edaDF)  
summary(edaDF)

### Visualize Spending Scores Across Customer Segments & Categories

Here we want to see where the most valuable customers reside with respect to the provided segmentation and customer categories. If the concentrations are skewed, we will likely need to perform stratified sampling on such for k-Fold cross validation (e.g. evaluating modeling approaches).

plot <- edaDF %>%   
 dplyr::select(Spending\_Score, Var\_1, Segmentation) %>%  
 group\_by(Spending\_Score, Var\_1, Segmentation) %>%   
 dplyr::summarize(ClientCount=n()) %>%   
 ggplot(  
 aes(  
 x=Var\_1,  
 y=ClientCount,  
 fill=Spending\_Score)) +  
 geom\_bar(stat="identity") +   
 labs(  
 x='Customer Categories',  
 y='Customer Count',  
 fill='Spending Score',  
 title='Distribution of Customers by Spending Score & Category Across Segments'  
 ) +  
 facet\_grid(rows = vars(Segmentation))

## `summarise()` has grouped output by 'Spending\_Score', 'Var\_1'. You can override  
## using the `.groups` argument.

# Display plot  
print(plot)

 It would seem the data is heavily skewed to the *Cat\_6* level of the categories. This means this feature will likely not be a good contributor for predictions of customer segmentation. If it is to still be used as a predictor, it will be important to utilized stratified sampling on customer category column in k-Fold cross validation evaluation to ensure samples are being drawn proportionally with each fold iteration.

Additionally, *segment D* and *segment A* seem to contain predominantly low spending score customers. This is important for marketers to know as this will help them dictate how to advertise for these different segments with respect to what the customers are most likely willing to spend. Instead, as seen in the plot *segment C* and *segment B* contain the majority of the present *average* and *high* scoring customers.

### General Understanding of Row Counts Across Categories

After seeing the results from the first plot it is a good idea to understand just how many customers fall in each of the categories in general. Here, a quick summary will be performed to obtain such. The same will be conducted for the segmentation column as well.

# Groupby RowCount Summation by Customer Categories  
edaDF %>%   
 dplyr::select(Var\_1) %>%   
 group\_by(Var\_1) %>%   
 dplyr::summarise(RowCount=n())

## # A tibble: 7 × 2  
## Var\_1 RowCount  
## <fct> <int>  
## 1 Cat\_1 104  
## 2 Cat\_2 362  
## 3 Cat\_3 634  
## 4 Cat\_4 849  
## 5 Cat\_5 74  
## 6 Cat\_6 4476  
## 7 Cat\_7 166

# Groupby RowCount Summation by Customer Segmentation  
edaDF %>%   
 dplyr::select(Segmentation) %>%   
 group\_by(Segmentation) %>%   
 dplyr::summarise(RowCount=n())

## # A tibble: 4 × 2  
## Segmentation RowCount  
## <fct> <int>  
## 1 A 1616  
## 2 B 1572  
## 3 C 1720  
## 4 D 1757

What is the row count sum of all non-category 6 levels?

edaDF %>%   
 dplyr::select(Var\_1) %>%   
 filter(Var\_1 != 'Cat\_6') %>%   
 dplyr::summarise(RowCount=n())

## RowCount  
## 1 2189

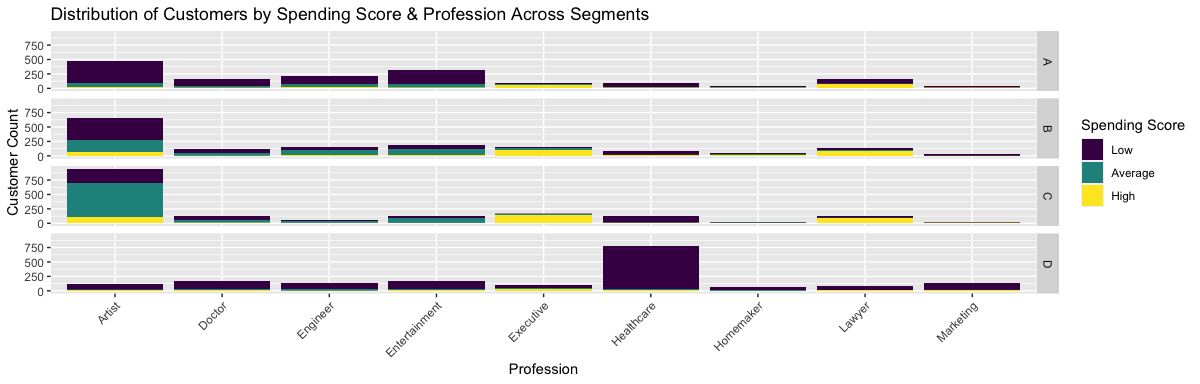
From a predictor standpoint, it may be worth considering creating a new feature that represents a weighted distribution of customers across the categories which penalizes categories containing higher concentrations of customers. Doing so would allow for all the category levels to still be represented in the analysis and considered by the modeling approaches. The trade off with exploring such would be the loss of the interpretation aspect in the modeling results. That is, the client would have to decide if model predictive performance is more important than model interpretation or vice versa.

### Profession and Spending Score Across Segments

plot <- edaDF %>%   
 dplyr::select(Spending\_Score, Profession, Segmentation) %>%  
 group\_by(Spending\_Score, Profession, Segmentation) %>%   
 dplyr::summarize(ClientCount=n()) %>%   
 ggplot(  
 aes(  
 x=Profession,  
 y=ClientCount,  
 fill=Spending\_Score)) +  
 geom\_bar(stat="identity") +   
 labs(  
 x='Profession',  
 y='Customer Count',  
 fill='Spending Score',  
 title='Distribution of Customers by Spending Score & Profession Across Segments'  
 ) +  
 facet\_grid(rows = vars(Segmentation)) +   
 theme(axis.text.x = element\_text(angle = 45, vjust = 1, hjust=1))

## `summarise()` has grouped output by 'Spending\_Score', 'Profession'. You can  
## override using the `.groups` argument.

# Display plot  
print(plot)



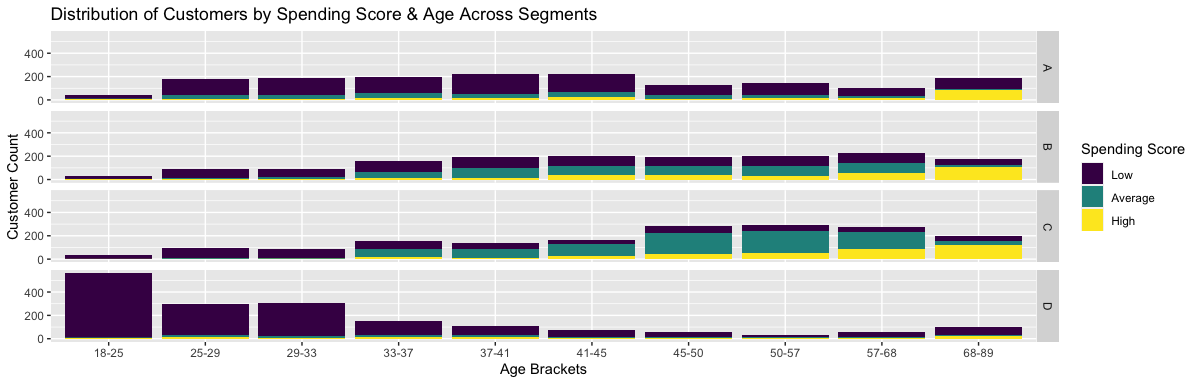
The first detail that stands out here is that across the segments the majority of customers are artists and healthcare workers. Next it clear that segment D is primarily comprised of healthcare workers specifically. This may also serve as useful information with respect to advertising strategies. While the artist occupation category makes up most of the average spending score customers in segment C, the high spending customers are distributed across executive and lawyer occupations in addition to artists. Segment B shows an even stronger concentration of high spending customers in these two occupations. One other interesting trait that can be seen in this plot with respect to the trends observed in the previous plot are how occupations differ between segment A & D, which were previously seen to be the segments containing the lower scored customers. Segment A shows a scarce presence of healthcare workers. Instead, there are primarily artists, doctors, engineers, and entertainment workers found in this segment.

### Age and Spending Score Across Segments

plot <- edaDF %>%   
 dplyr::select(Spending\_Score, age\_dRanked\_Bins, Segmentation) %>%  
 group\_by(Spending\_Score, age\_dRanked\_Bins, Segmentation) %>%   
 dplyr::summarize(ClientCount=n()) %>%   
 ggplot(  
 aes(  
 x=age\_dRanked\_Bins,  
 y=ClientCount,  
 fill=Spending\_Score)) +  
 geom\_bar(stat="identity") +   
 labs(  
 x='Age Brackets',  
 y='Customer Count',  
 fill='Spending Score',  
 title='Distribution of Customers by Spending Score & Age Across Segments'  
 ) +  
 facet\_grid(rows = vars(Segmentation))

## `summarise()` has grouped output by 'Spending\_Score', 'age\_dRanked\_Bins'. You  
## can override using the `.groups` argument.

# Display plot  
print(plot)



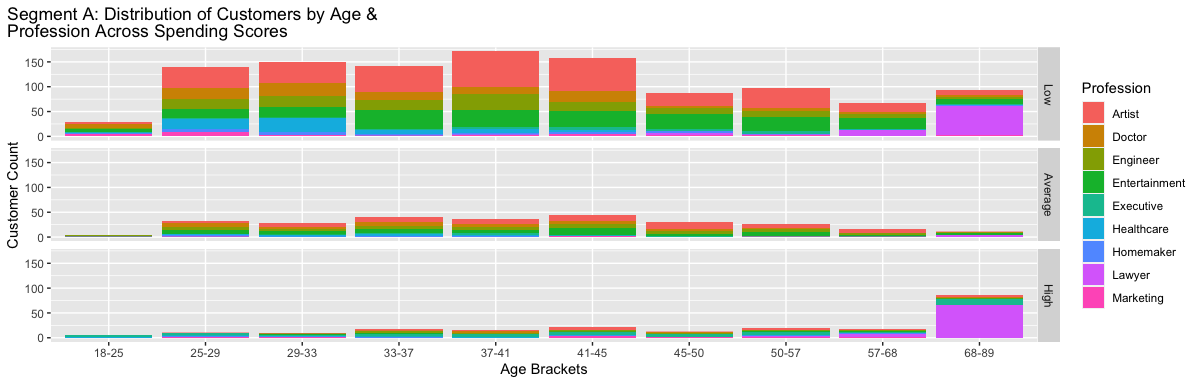
This plot shows it is clear that customers with the highest spending scores fall into the *68-89* age range consistently across segments A, B, and C. Segment D is primarily comprised of the younger demographic with respect to customers. This is probably one of the characteristics used by the organization to build this segment. Segment A has a more uniform distribution across the brackets 25-29 all the way up to 41-45. This is another distinguishing factor as to how the low-scored customer segments differ from one another. It is clear for segments B & C that middle-aged individuals and above should be focused on for advertising efforts as these not only contain the majority of the customers but are also made up primarily of average scored spenders.

### Segment A: Age Brackets and Profession Across Scores

plot <- edaDF %>%   
 dplyr::select(  
 Spending\_Score,   
 age\_dRanked\_Bins,   
 Profession,  
 Segmentation,  
 Var\_1  
 ) %>%  
 filter(Segmentation == 'A') %>%   
 group\_by(  
 Spending\_Score,   
 age\_dRanked\_Bins,   
 Profession,  
 ) %>%   
 dplyr::summarize(ClientCount=n()) %>%   
 ggplot(  
 aes(  
 x=age\_dRanked\_Bins,  
 y=ClientCount,  
 fill=Profession)) +  
 geom\_bar(stat="identity") +   
 labs(  
 x='Age Brackets',  
 y='Customer Count',  
 fill='Profession',  
 title='Segment A: Distribution of Customers by Age &<br>Profession Across Spending Scores'  
 ) +  
 facet\_grid(rows = vars(Spending\_Score)) +  
 theme(  
 plot.title.position = 'plot',  
 plot.title = element\_markdown()  
 )

## `summarise()` has grouped output by 'Spending\_Score', 'age\_dRanked\_Bins'. You  
## can override using the `.groups` argument.

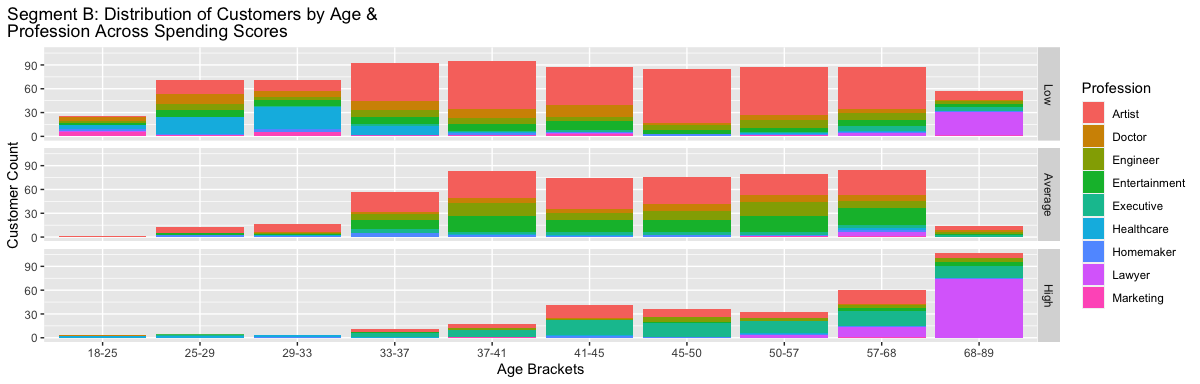
# Display plot  
print(plot)

 ### Segment B: Age Brackets and Profession Across Scores

plot <- edaDF %>%   
 dplyr::select(  
 Spending\_Score,   
 age\_dRanked\_Bins,   
 Profession,  
 Segmentation,  
 Var\_1  
 ) %>%  
 filter(Segmentation == 'B') %>%   
 group\_by(  
 Spending\_Score,   
 age\_dRanked\_Bins,   
 Profession,  
 ) %>%   
 dplyr::summarize(ClientCount=n()) %>%   
 ggplot(  
 aes(  
 x=age\_dRanked\_Bins,  
 y=ClientCount,  
 fill=Profession)) +  
 geom\_bar(stat="identity") +   
 labs(  
 x='Age Brackets',  
 y='Customer Count',  
 fill='Profession',  
 title='Segment B: Distribution of Customers by Age &<br>Profession Across Spending Scores'  
 ) +  
 facet\_grid(rows = vars(Spending\_Score)) +  
 theme(  
 plot.title.position = 'plot',  
 plot.title = element\_markdown()  
 )

## `summarise()` has grouped output by 'Spending\_Score', 'age\_dRanked\_Bins'. You  
## can override using the `.groups` argument.

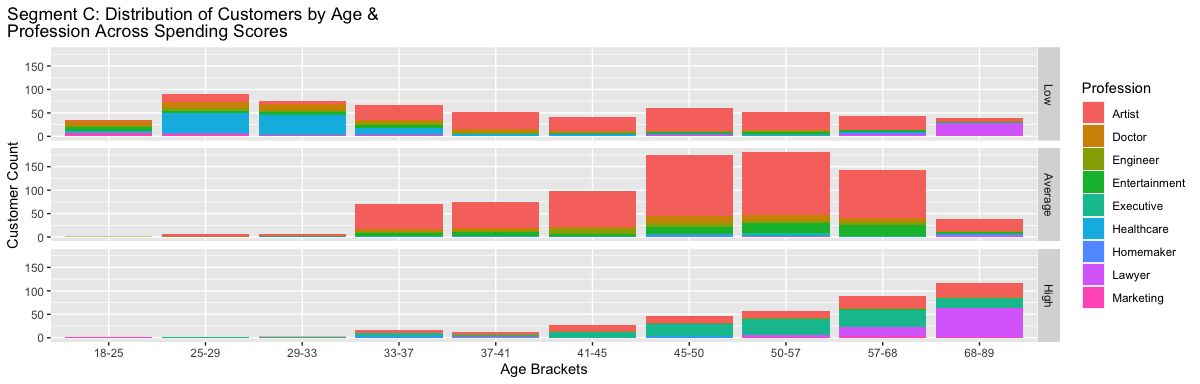
# Display plot  
print(plot)

 ### Segment C: Age Brackets and Profession Across Scores

plot <- edaDF %>%   
 dplyr::select(  
 Spending\_Score,   
 age\_dRanked\_Bins,   
 Profession,  
 Segmentation,  
 Var\_1  
 ) %>%  
 filter(Segmentation == 'C') %>%   
 group\_by(  
 Spending\_Score,   
 age\_dRanked\_Bins,   
 Profession,  
 ) %>%   
 dplyr::summarize(ClientCount=n()) %>%   
 ggplot(  
 aes(  
 x=age\_dRanked\_Bins,  
 y=ClientCount,  
 fill=Profession)) +  
 geom\_bar(stat="identity") +   
 labs(  
 x='Age Brackets',  
 y='Customer Count',  
 fill='Profession',  
 title='Segment C: Distribution of Customers by Age &<br>Profession Across Spending Scores'  
 ) +  
 facet\_grid(rows = vars(Spending\_Score)) +  
 theme(  
 plot.title.position = 'plot',  
 plot.title = element\_markdown()  
 )

## `summarise()` has grouped output by 'Spending\_Score', 'age\_dRanked\_Bins'. You  
## can override using the `.groups` argument.

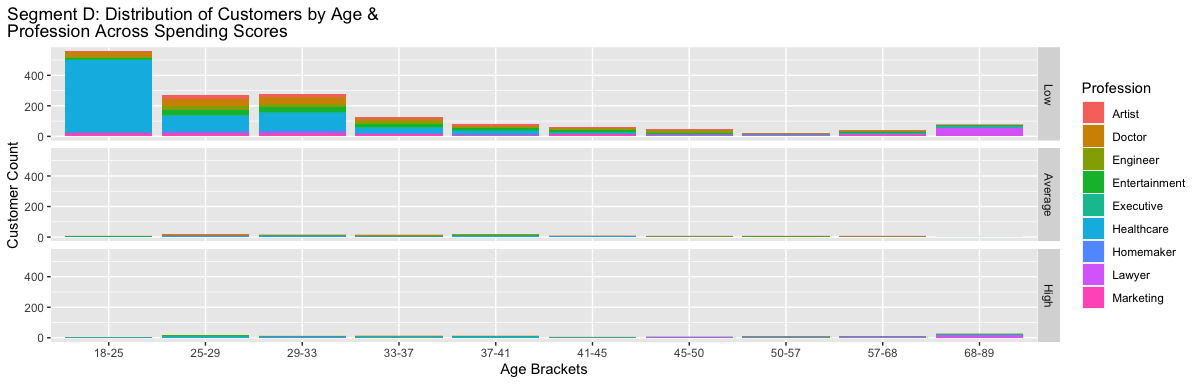
# Display plot  
print(plot)

 ### Segment D: Age Brackets and Profession Across Scores

plot <- edaDF %>%   
 dplyr::select(  
 Spending\_Score,   
 age\_dRanked\_Bins,   
 Profession,  
 Segmentation,  
 Var\_1  
 ) %>%  
 filter(Segmentation == 'D') %>%   
 group\_by(  
 Spending\_Score,   
 age\_dRanked\_Bins,   
 Profession,  
 ) %>%   
 dplyr::summarize(ClientCount=n()) %>%   
 ggplot(  
 aes(  
 x=age\_dRanked\_Bins,  
 y=ClientCount,  
 fill=Profession)) +  
 geom\_bar(stat="identity") +   
 labs(  
 x='Age Brackets',  
 y='Customer Count',  
 fill='Profession',  
 title='Segment D: Distribution of Customers by Age &<br>Profession Across Spending Scores'  
 ) +  
 facet\_grid(rows = vars(Spending\_Score)) +  
 theme(  
 plot.title.position = 'plot',  
 plot.title = element\_markdown()  
 )

## `summarise()` has grouped output by 'Spending\_Score', 'age\_dRanked\_Bins'. You  
## can override using the `.groups` argument.

# Display plot  
print(plot)



## Optimal Value for K

Since this dataset is a customer segmentation data set with pre-defined segments, we want to evaluate the validity of the number of segments chosen in the original clustering performed on the existing customer database. If this automobile manufacturer used either kMeans or kModes clustering, they would have chosen a desired number for k, the number of clusters they wanted the algorithm to create. Since there are four segments, they must have chosen k=4, but is this the true optimal value based on the data? We want to understand this before moving on to futher modeling.

Given our data is primarily categorical data after the cleaning process, we will utilize kModes, which works better with categorical data than kMeans, and we will utilize the elbow method for determining the optimal value for k.

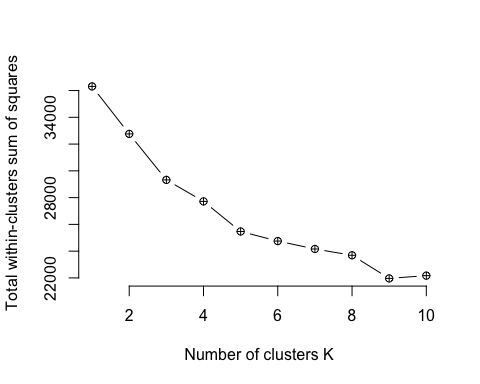
library(klaR)

## Loading required package: MASS

##   
## Attaching package: 'MASS'

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

# remove pre-defined cluster attributes  
kmodesDF<- edaDF[,-c(6,7)]  
  
set.seed(10)  
  
# max number of clusters to check  
k.max<-10  
  
kTest<- sapply(1:k.max,  
 function(k){set.seed(10)  
 sum(kmodes(kmodesDF, k, iter.max = 100, weighted=FALSE)$withindiff)})  
#kTest  
  
plot(1:k.max, kTest, type = "b", pch=10, frame=FALSE, xlab = "Number of clusters K", ylab="Total within-clusters sum of squares")

 While the elbow plot above does not display a perfect elbow, it is reasonable to infer that an elbow exists between k=3 and k=5. It could be that “better” clusters would be derived at k=3 or k=5 than those derived at k=4, however it is not obvious that 3 or 5 is a better value for k based on the elbow method, and there may have been internal business rules applied that required the automobile manufacturer to chose 4 clusters. Had this process convincingly shown that the optimal value for k is significantly different than k=4, for example k=10, we would consider re-clustering the data using the optimal k value and perform our modeling on the re-clustered data to hopefully derive more accurate and useful results. That is not the case, however, so we feel comfortable proceeding with modeling the data set using the original segmentation as the target variable in training and testing our models.

# Models and Methods

The first modeling technique we will utilize in our analysis is Association Rule Mining. While we do not expect to use ARM as our final model, ARM can provide useful information to better understand how each attribute interacts in the process of segmenting customers into segments A, B, C, and D. Our goal is to find strong rules that begin to paint a data profile of each segment.

## Association Rule Mining using Apiori algorithm

We look at the segments using ARM to see if we can find some patterns in the segments that can help us with making some models

#creating ARM specific working dataframe based on cleaned data frame used for EDA  
bd<-edaDF  
  
# removing ageRanked   
bd<-bd[,-8]  
  
#Running A Rules on Segment A using apriori sort by support First  
  
RP1 <-apriori(bd, parameter = list(supp = 0.01, conf = 0.4, minlen= 4), appearance = list(default="lhs",rhs="Segmentation=A"), control = list (verbose=F))  
  
RP1 <- sort(RP1,decreasing= TRUE,by="support")  
#inspect(RP1[1:50,])  
arules::inspect(RP1[1:10,])

## lhs rhs support confidence coverage lift count  
## [1] {Gender=Male,   
## Profession=Entertainment,   
## Spending\_Score=Low} => {Segmentation=A} 0.02640660 0.5086705 0.05191298 2.097951 176  
## [2] {Work\_Experience=1,   
## Spending\_Score=Low,   
## EverMarriedTRUE} => {Segmentation=A} 0.02610653 0.4084507 0.06391598 1.684606 174  
## [3] {Profession=Entertainment,   
## Spending\_Score=Low,   
## IsGraduated} => {Segmentation=A} 0.02445611 0.5174603 0.04726182 2.134204 163  
## [4] {Profession=Entertainment,   
## Spending\_Score=Low,   
## Var\_1=Cat\_6} => {Segmentation=A} 0.02415604 0.5111111 0.04726182 2.108017 161  
## [5] {Work\_Experience=1,   
## Spending\_Score=Low,   
## Family\_Size=1} => {Segmentation=A} 0.02340585 0.4406780 0.05311328 1.817524 156  
## [6] {Spending\_Score=Low,   
## age\_dRanked\_Bins=37-41,   
## IsGraduated} => {Segmentation=A} 0.02040510 0.4276730 0.04771193 1.763886 136  
## [7] {Gender=Female,   
## Profession=Engineer,   
## Spending\_Score=Low} => {Segmentation=A} 0.01995499 0.4554795 0.04381095 1.878571 133  
## [8] {Spending\_Score=Low,   
## age\_dRanked\_Bins=41-45,   
## IsGraduated} => {Segmentation=A} 0.01845461 0.4522059 0.04081020 1.865069 123  
## [9] {Gender=Male,   
## Work\_Experience=1,   
## Spending\_Score=Low,   
## EverMarriedTRUE} => {Segmentation=A} 0.01815454 0.4245614 0.04276069 1.751053 121  
## [10] {Gender=Male,   
## Profession=Entertainment,   
## Spending\_Score=Low,   
## IsGraduated} => {Segmentation=A} 0.01680420 0.5161290 0.03255814 2.128713 112

Sorting by Lift

RP1 <- sort(RP1,decreasing= TRUE,by="lift")  
#inspect(RP1[1:50,])  
arules::inspect(RP1[1:10,])

## lhs rhs support confidence coverage lift count  
## [1] {Profession=Entertainment,   
## Spending\_Score=Low,   
## Family\_Size=1} => {Segmentation=A} 0.01290323 0.6466165 0.01995499 2.666893 86  
## [2] {Gender=Male,   
## Profession=Entertainment,   
## Spending\_Score=Low,   
## EverMarriedTRUE} => {Segmentation=A} 0.01380345 0.5859873 0.02355589 2.416835 92  
## [3] {Gender=Male,   
## Profession=Entertainment,   
## Work\_Experience=1,   
## Spending\_Score=Low} => {Segmentation=A} 0.01155289 0.5620438 0.02055514 2.318083 77  
## [4] {Profession=Engineer,   
## Spending\_Score=Low,   
## Family\_Size=1} => {Segmentation=A} 0.01080270 0.5581395 0.01935484 2.301980 72  
## [5] {Profession=Entertainment,   
## Spending\_Score=Low,   
## EverMarriedTRUE} => {Segmentation=A} 0.01560390 0.5473684 0.02850713 2.257556 104  
## [6] {Profession=Entertainment,   
## Work\_Experience=1,   
## Spending\_Score=Low} => {Segmentation=A} 0.01470368 0.5297297 0.02775694 2.184807 98  
## [7] {Profession=Entertainment,   
## Spending\_Score=Low,   
## Var\_1=Cat\_6,   
## IsGraduated} => {Segmentation=A} 0.01680420 0.5283019 0.03180795 2.178918 112  
## [8] {Gender=Male,   
## Profession=Entertainment,   
## Spending\_Score=Low,   
## Var\_1=Cat\_6,   
## IsGraduated} => {Segmentation=A} 0.01095274 0.5214286 0.02100525 2.150570 73  
## [9] {Gender=Male,   
## Profession=Entertainment,   
## Spending\_Score=Low,   
## Var\_1=Cat\_6} => {Segmentation=A} 0.01665416 0.5186916 0.03210803 2.139282 111  
## [10] {Profession=Entertainment,   
## Spending\_Score=Low,   
## Family\_Size=2} => {Segmentation=A} 0.01305326 0.5178571 0.02520630 2.135840 87

Sorting by confidence

RP1 <- sort(RP1,decreasing= TRUE,by="confidence")  
#inspect(RP1[1:50,])  
arules::inspect(RP1[1:10,])

## lhs rhs support confidence coverage lift count  
## [1] {Profession=Entertainment,   
## Spending\_Score=Low,   
## Family\_Size=1} => {Segmentation=A} 0.01290323 0.6466165 0.01995499 2.666893 86  
## [2] {Gender=Male,   
## Profession=Entertainment,   
## Spending\_Score=Low,   
## EverMarriedTRUE} => {Segmentation=A} 0.01380345 0.5859873 0.02355589 2.416835 92  
## [3] {Gender=Male,   
## Profession=Entertainment,   
## Work\_Experience=1,   
## Spending\_Score=Low} => {Segmentation=A} 0.01155289 0.5620438 0.02055514 2.318083 77  
## [4] {Profession=Engineer,   
## Spending\_Score=Low,   
## Family\_Size=1} => {Segmentation=A} 0.01080270 0.5581395 0.01935484 2.301980 72  
## [5] {Profession=Entertainment,   
## Spending\_Score=Low,   
## EverMarriedTRUE} => {Segmentation=A} 0.01560390 0.5473684 0.02850713 2.257556 104  
## [6] {Profession=Entertainment,   
## Work\_Experience=1,   
## Spending\_Score=Low} => {Segmentation=A} 0.01470368 0.5297297 0.02775694 2.184807 98  
## [7] {Profession=Entertainment,   
## Spending\_Score=Low,   
## Var\_1=Cat\_6,   
## IsGraduated} => {Segmentation=A} 0.01680420 0.5283019 0.03180795 2.178918 112  
## [8] {Gender=Male,   
## Profession=Entertainment,   
## Spending\_Score=Low,   
## Var\_1=Cat\_6,   
## IsGraduated} => {Segmentation=A} 0.01095274 0.5214286 0.02100525 2.150570 73  
## [9] {Gender=Male,   
## Profession=Entertainment,   
## Spending\_Score=Low,   
## Var\_1=Cat\_6} => {Segmentation=A} 0.01665416 0.5186916 0.03210803 2.139282 111  
## [10] {Profession=Entertainment,   
## Spending\_Score=Low,   
## Family\_Size=2} => {Segmentation=A} 0.01305326 0.5178571 0.02520630 2.135840 87

Running Arules on Segment B sorting by Confidence first

RP2 <-apriori(bd, parameter = list(supp = 0.005, conf = 0.4), appearance = list(default="lhs",rhs="Segmentation=B"), control = list (verbose=F))  
  
RP2 <- sort(RP2,decreasing= TRUE,by="confidence")  
#inspect(RP2[1:50])  
arules::inspect(RP2[1:10])

## lhs rhs support confidence coverage lift count  
## [1] {Profession=Artist,   
## Spending\_Score=Low,   
## Family\_Size=1,   
## Var\_1=Cat\_6,   
## age\_dRanked\_Bins=50-57} => {Segmentation=B} 0.005101275 0.5151515 0.009902476 2.184151 34  
## [2] {Profession=Artist,   
## Spending\_Score=Low,   
## Family\_Size=1,   
## age\_dRanked\_Bins=50-57,   
## IsGraduated} => {Segmentation=B} 0.005401350 0.5142857 0.010502626 2.180480 36  
## [3] {Profession=Artist,   
## Family\_Size=1,   
## Var\_1=Cat\_6,   
## age\_dRanked\_Bins=50-57} => {Segmentation=B} 0.005101275 0.5074627 0.010052513 2.151551 34  
## [4] {Profession=Artist,   
## Spending\_Score=Low,   
## Family\_Size=1,   
## age\_dRanked\_Bins=50-57} => {Segmentation=B} 0.005701425 0.5000000 0.011402851 2.119911 38  
## [5] {Profession=Artist,   
## Family\_Size=1,   
## age\_dRanked\_Bins=50-57,   
## IsGraduated} => {Segmentation=B} 0.005401350 0.5000000 0.010802701 2.119911 36  
## [6] {Gender=Male,   
## Profession=Artist,   
## Work\_Experience=0,   
## Family\_Size=1} => {Segmentation=B} 0.005401350 0.5000000 0.010802701 2.119911 36  
## [7] {Gender=Male,   
## Profession=Artist,   
## Work\_Experience=0,   
## Spending\_Score=Low,   
## Family\_Size=1} => {Segmentation=B} 0.005401350 0.5000000 0.010802701 2.119911 36  
## [8] {Profession=Artist,   
## Family\_Size=1,   
## age\_dRanked\_Bins=50-57} => {Segmentation=B} 0.005701425 0.4871795 0.011702926 2.065554 38  
## [9] {Work\_Experience=1,   
## Var\_1=Cat\_4,   
## IsGraduated,   
## EverMarriedTRUE} => {Segmentation=B} 0.005401350 0.4864865 0.011102776 2.062616 36  
## [10] {Gender=Female,   
## Profession=Artist,   
## Spending\_Score=Low,   
## age\_dRanked\_Bins=45-50} => {Segmentation=B} 0.005101275 0.4857143 0.010502626 2.059342 34

Sort by Lift

RP2 <- sort(RP2,decreasing= TRUE,by="lift")  
#inspect(RP2[1:50])  
arules::inspect(RP2[1:10])

## lhs rhs support confidence coverage lift count  
## [1] {Profession=Artist,   
## Spending\_Score=Low,   
## Family\_Size=1,   
## Var\_1=Cat\_6,   
## age\_dRanked\_Bins=50-57} => {Segmentation=B} 0.005101275 0.5151515 0.009902476 2.184151 34  
## [2] {Profession=Artist,   
## Spending\_Score=Low,   
## Family\_Size=1,   
## age\_dRanked\_Bins=50-57,   
## IsGraduated} => {Segmentation=B} 0.005401350 0.5142857 0.010502626 2.180480 36  
## [3] {Profession=Artist,   
## Family\_Size=1,   
## Var\_1=Cat\_6,   
## age\_dRanked\_Bins=50-57} => {Segmentation=B} 0.005101275 0.5074627 0.010052513 2.151551 34  
## [4] {Profession=Artist,   
## Spending\_Score=Low,   
## Family\_Size=1,   
## age\_dRanked\_Bins=50-57} => {Segmentation=B} 0.005701425 0.5000000 0.011402851 2.119911 38  
## [5] {Profession=Artist,   
## Family\_Size=1,   
## age\_dRanked\_Bins=50-57,   
## IsGraduated} => {Segmentation=B} 0.005401350 0.5000000 0.010802701 2.119911 36  
## [6] {Gender=Male,   
## Profession=Artist,   
## Work\_Experience=0,   
## Family\_Size=1} => {Segmentation=B} 0.005401350 0.5000000 0.010802701 2.119911 36  
## [7] {Gender=Male,   
## Profession=Artist,   
## Work\_Experience=0,   
## Spending\_Score=Low,   
## Family\_Size=1} => {Segmentation=B} 0.005401350 0.5000000 0.010802701 2.119911 36  
## [8] {Profession=Artist,   
## Family\_Size=1,   
## age\_dRanked\_Bins=50-57} => {Segmentation=B} 0.005701425 0.4871795 0.011702926 2.065554 38  
## [9] {Work\_Experience=1,   
## Var\_1=Cat\_4,   
## IsGraduated,   
## EverMarriedTRUE} => {Segmentation=B} 0.005401350 0.4864865 0.011102776 2.062616 36  
## [10] {Gender=Female,   
## Profession=Artist,   
## Spending\_Score=Low,   
## age\_dRanked\_Bins=45-50} => {Segmentation=B} 0.005101275 0.4857143 0.010502626 2.059342 34

Sort by Support

RP2 <- sort(RP2,decreasing= TRUE,by="support")  
#inspect(RP2[1:50])  
arules::inspect(RP2[1:10])

## lhs rhs support confidence coverage lift count  
## [1] {Profession=Artist,   
## Spending\_Score=Low,   
## Family\_Size=1} => {Segmentation=B} 0.03120780 0.4086444 0.07636909 1.732579 208  
## [2] {Profession=Artist,   
## Family\_Size=1} => {Segmentation=B} 0.03120780 0.4054581 0.07696924 1.719070 208  
## [3] {Profession=Artist,   
## Spending\_Score=Low,   
## Family\_Size=1,   
## IsGraduated} => {Segmentation=B} 0.02865716 0.4116379 0.06961740 1.745272 191  
## [4] {Profession=Artist,   
## Family\_Size=1,   
## IsGraduated} => {Segmentation=B} 0.02865716 0.4081197 0.07021755 1.730355 191  
## [5] {Profession=Artist,   
## Spending\_Score=Low,   
## Family\_Size=1,   
## Var\_1=Cat\_6} => {Segmentation=B} 0.02385596 0.4015152 0.05941485 1.702353 159  
## [6] {Profession=Artist,   
## Spending\_Score=Low,   
## Family\_Size=1,   
## Var\_1=Cat\_6,   
## IsGraduated} => {Segmentation=B} 0.02175544 0.4016620 0.05416354 1.702976 145  
## [7] {Gender=Male,   
## Profession=Artist,   
## Spending\_Score=Low,   
## Family\_Size=1} => {Segmentation=B} 0.01530383 0.4232365 0.03615904 1.794447 102  
## [8] {Gender=Male,   
## Profession=Artist,   
## Family\_Size=1} => {Segmentation=B} 0.01530383 0.4180328 0.03660915 1.772385 102  
## [9] {Gender=Female,   
## Profession=Artist,   
## Spending\_Score=Low,   
## Family\_Size=1,   
## IsGraduated} => {Segmentation=B} 0.01470368 0.4000000 0.03675919 1.695929 98  
## [10] {Gender=Male,   
## Profession=Artist,   
## Spending\_Score=Low,   
## Family\_Size=1,   
## IsGraduated} => {Segmentation=B} 0.01395349 0.4246575 0.03285821 1.800472 93

A rules on Segment C sorting by Support

RP3 <-apriori(bd, parameter = list(supp = 0.02, conf = 0.6, minlen=5), appearance = list(default="lhs",rhs="Segmentation=C"), control = list (verbose=F))  
  
RP3 <- sort(RP3,decreasing= TRUE,by="support")  
#inspect(RP3[1:50,])  
arules::inspect(RP3[1:10,])

## lhs rhs support confidence coverage lift count  
## [1] {Profession=Artist,   
## Spending\_Score=Average,   
## IsGraduated,   
## EverMarriedTRUE} => {Segmentation=C} 0.08222056 0.6867168 0.11972993 2.661028 548  
## [2] {Profession=Artist,   
## Spending\_Score=Average,   
## Var\_1=Cat\_6,   
## EverMarriedTRUE} => {Segmentation=C} 0.06826707 0.6862745 0.09947487 2.659314 455  
## [3] {Profession=Artist,   
## Spending\_Score=Average,   
## Var\_1=Cat\_6,   
## IsGraduated} => {Segmentation=C} 0.06406602 0.6965742 0.09197299 2.699225 427  
## [4] {Profession=Artist,   
## Spending\_Score=Average,   
## Var\_1=Cat\_6,   
## IsGraduated,   
## EverMarriedTRUE} => {Segmentation=C} 0.06406602 0.6965742 0.09197299 2.699225 427  
## [5] {Gender=Female,   
## Profession=Artist,   
## IsGraduated,   
## EverMarriedTRUE} => {Segmentation=C} 0.06166542 0.6372093 0.09677419 2.469186 411  
## [6] {Gender=Female,   
## Profession=Artist,   
## Var\_1=Cat\_6,   
## EverMarriedTRUE} => {Segmentation=C} 0.04966242 0.6365385 0.07801950 2.466587 331  
## [7] {Gender=Female,   
## Profession=Artist,   
## Var\_1=Cat\_6,   
## IsGraduated,   
## EverMarriedTRUE} => {Segmentation=C} 0.04756189 0.6549587 0.07261815 2.537965 317  
## [8] {Profession=Artist,   
## Family\_Size=2,   
## Var\_1=Cat\_6,   
## IsGraduated,   
## EverMarriedTRUE} => {Segmentation=C} 0.04636159 0.6143141 0.07546887 2.380467 309  
## [9] {Gender=Male,   
## Profession=Artist,   
## Spending\_Score=Average,   
## EverMarriedTRUE} => {Segmentation=C} 0.04621155 0.6209677 0.07441860 2.406250 308  
## [10] {Gender=Male,   
## Profession=Artist,   
## Spending\_Score=Average,   
## IsGraduated} => {Segmentation=C} 0.04231058 0.6394558 0.06616654 2.477891 282

Sorted by Confidence

RP3 <- sort(RP3,decreasing= TRUE,by="confidence")  
#inspect(RP3[1:50,])  
arules::inspect(RP3[1:10,])

## lhs rhs support confidence coverage lift count  
## [1] {Profession=Artist,   
## Spending\_Score=Average,   
## age\_dRanked\_Bins=50-57,   
## EverMarriedTRUE} => {Segmentation=C} 0.02010503 0.7928994 0.02535634 3.072485 134  
## [2] {Gender=Female,   
## Profession=Artist,   
## Spending\_Score=Average,   
## Var\_1=Cat\_6,   
## IsGraduated} => {Segmentation=C} 0.03015754 0.7528090 0.04006002 2.917135 201  
## [3] {Gender=Female,   
## Profession=Artist,   
## Spending\_Score=Average,   
## Var\_1=Cat\_6,   
## IsGraduated,   
## EverMarriedTRUE} => {Segmentation=C} 0.03015754 0.7528090 0.04006002 2.917135 201  
## [4] {Gender=Female,   
## Profession=Artist,   
## Spending\_Score=Average,   
## IsGraduated} => {Segmentation=C} 0.03990998 0.7450980 0.05356339 2.887255 266  
## [5] {Gender=Female,   
## Profession=Artist,   
## Spending\_Score=Average,   
## IsGraduated,   
## EverMarriedTRUE} => {Segmentation=C} 0.03990998 0.7450980 0.05356339 2.887255 266  
## [6] {Gender=Female,   
## Profession=Artist,   
## Spending\_Score=Average,   
## Var\_1=Cat\_6} => {Segmentation=C} 0.03090773 0.7357143 0.04201050 2.850893 206  
## [7] {Gender=Female,   
## Profession=Artist,   
## Spending\_Score=Average,   
## Var\_1=Cat\_6,   
## EverMarriedTRUE} => {Segmentation=C} 0.03090773 0.7357143 0.04201050 2.850893 206  
## [8] {Profession=Artist,   
## Work\_Experience=1,   
## Spending\_Score=Average,   
## Var\_1=Cat\_6,   
## IsGraduated} => {Segmentation=C} 0.02325581 0.7276995 0.03195799 2.819836 155  
## [9] {Profession=Artist,   
## Work\_Experience=1,   
## Spending\_Score=Average,   
## Var\_1=Cat\_6,   
## IsGraduated,   
## EverMarriedTRUE} => {Segmentation=C} 0.02325581 0.7276995 0.03195799 2.819836 155  
## [10] {Profession=Artist,   
## Work\_Experience=0,   
## Spending\_Score=Average,   
## Var\_1=Cat\_6,   
## IsGraduated} => {Segmentation=C} 0.02010503 0.7243243 0.02775694 2.806757 134

Sort by lift

RP3 <- sort(RP3,decreasing= TRUE,by="lift")  
#inspect(RP3[1:50,])  
arules::inspect(RP3[1:10,])

## lhs rhs support confidence coverage lift count  
## [1] {Profession=Artist,   
## Spending\_Score=Average,   
## age\_dRanked\_Bins=50-57,   
## EverMarriedTRUE} => {Segmentation=C} 0.02010503 0.7928994 0.02535634 3.072485 134  
## [2] {Gender=Female,   
## Profession=Artist,   
## Spending\_Score=Average,   
## Var\_1=Cat\_6,   
## IsGraduated} => {Segmentation=C} 0.03015754 0.7528090 0.04006002 2.917135 201  
## [3] {Gender=Female,   
## Profession=Artist,   
## Spending\_Score=Average,   
## Var\_1=Cat\_6,   
## IsGraduated,   
## EverMarriedTRUE} => {Segmentation=C} 0.03015754 0.7528090 0.04006002 2.917135 201  
## [4] {Gender=Female,   
## Profession=Artist,   
## Spending\_Score=Average,   
## IsGraduated} => {Segmentation=C} 0.03990998 0.7450980 0.05356339 2.887255 266  
## [5] {Gender=Female,   
## Profession=Artist,   
## Spending\_Score=Average,   
## IsGraduated,   
## EverMarriedTRUE} => {Segmentation=C} 0.03990998 0.7450980 0.05356339 2.887255 266  
## [6] {Gender=Female,   
## Profession=Artist,   
## Spending\_Score=Average,   
## Var\_1=Cat\_6} => {Segmentation=C} 0.03090773 0.7357143 0.04201050 2.850893 206  
## [7] {Gender=Female,   
## Profession=Artist,   
## Spending\_Score=Average,   
## Var\_1=Cat\_6,   
## EverMarriedTRUE} => {Segmentation=C} 0.03090773 0.7357143 0.04201050 2.850893 206  
## [8] {Profession=Artist,   
## Work\_Experience=1,   
## Spending\_Score=Average,   
## Var\_1=Cat\_6,   
## IsGraduated} => {Segmentation=C} 0.02325581 0.7276995 0.03195799 2.819836 155  
## [9] {Profession=Artist,   
## Work\_Experience=1,   
## Spending\_Score=Average,   
## Var\_1=Cat\_6,   
## IsGraduated,   
## EverMarriedTRUE} => {Segmentation=C} 0.02325581 0.7276995 0.03195799 2.819836 155  
## [10] {Profession=Artist,   
## Work\_Experience=0,   
## Spending\_Score=Average,   
## Var\_1=Cat\_6,   
## IsGraduated} => {Segmentation=C} 0.02010503 0.7243243 0.02775694 2.806757 134

A Rules on Segment D Sorting by support

RP4 <-apriori(bd, parameter = list(supp = 0.02, conf = 0.7,minlen=4), appearance = list(default="lhs",rhs="Segmentation=D"), control = list (verbose=F))  
  
RP4 <- sort(RP4,decreasing= TRUE,by="support")  
#inspect(RP4[1:10,])  
arules::inspect(RP4[1:10,])

## lhs rhs support confidence coverage lift count  
## [1] {Profession=Healthcare,   
## Spending\_Score=Low,   
## age\_dRanked\_Bins=18-25} => {Segmentation=D} 0.07096774 0.9792961 0.07246812 3.714860 473  
## [2] {Gender=Male,   
## Profession=Healthcare,   
## Spending\_Score=Low} => {Segmentation=D} 0.07021755 0.8041237 0.08732183 3.050361 468  
## [3] {Profession=Healthcare,   
## Spending\_Score=Low,   
## Var\_1=Cat\_6} => {Segmentation=D} 0.06496624 0.7530435 0.08627157 2.856594 433  
## [4] {Gender=Male,   
## Spending\_Score=Low,   
## age\_dRanked\_Bins=18-25} => {Segmentation=D} 0.05266317 0.8841310 0.05956489 3.353861 351  
## [5] {Gender=Male,   
## Profession=Healthcare,   
## age\_dRanked\_Bins=18-25} => {Segmentation=D} 0.04636159 0.9809524 0.04726182 3.721143 309  
## [6] {Gender=Male,   
## Profession=Healthcare,   
## Spending\_Score=Low,   
## age\_dRanked\_Bins=18-25} => {Segmentation=D} 0.04606152 0.9839744 0.04681170 3.732606 307  
## [7] {Spending\_Score=Low,   
## Var\_1=Cat\_6,   
## age\_dRanked\_Bins=18-25} => {Segmentation=D} 0.04201050 0.8588957 0.04891223 3.258133 280  
## [8] {Gender=Male,   
## Profession=Healthcare,   
## Var\_1=Cat\_6} => {Segmentation=D} 0.04186047 0.7664835 0.05461365 2.907577 279  
## [9] {Gender=Male,   
## Profession=Healthcare,   
## Spending\_Score=Low,   
## Var\_1=Cat\_6} => {Segmentation=D} 0.04066017 0.8065476 0.05041260 3.059556 271  
## [10] {Profession=Healthcare,   
## Var\_1=Cat\_6,   
## age\_dRanked\_Bins=18-25} => {Segmentation=D} 0.03585896 0.9795082 0.03660915 3.715664 239

Support by Lift

RP4 <- sort(RP4,decreasing= TRUE,by="lift")  
arules::inspect(RP4[1:10,])

## lhs rhs support confidence coverage lift count  
## [1] {Gender=Male,   
## Profession=Healthcare,   
## Var\_1=Cat\_6,   
## age\_dRanked\_Bins=18-25} => {Segmentation=D} 0.02325581 0.9872611 0.02355589 3.745074 155  
## [2] {Gender=Male,   
## Profession=Healthcare,   
## Spending\_Score=Low,   
## Var\_1=Cat\_6,   
## age\_dRanked\_Bins=18-25} => {Segmentation=D} 0.02325581 0.9872611 0.02355589 3.745074 155  
## [3] {Gender=Male,   
## Profession=Healthcare,   
## Spending\_Score=Low,   
## age\_dRanked\_Bins=18-25} => {Segmentation=D} 0.04606152 0.9839744 0.04681170 3.732606 307  
## [4] {Profession=Healthcare,   
## Spending\_Score=Low,   
## Var\_1=Cat\_6,   
## age\_dRanked\_Bins=18-25} => {Segmentation=D} 0.03585896 0.9835391 0.03645911 3.730955 239  
## [5] {Profession=Healthcare,   
## Work\_Experience=1,   
## Spending\_Score=Low,   
## age\_dRanked\_Bins=18-25} => {Segmentation=D} 0.02355589 0.9812500 0.02400600 3.722272 157  
## [6] {Gender=Male,   
## Profession=Healthcare,   
## age\_dRanked\_Bins=18-25} => {Segmentation=D} 0.04636159 0.9809524 0.04726182 3.721143 309  
## [7] {Profession=Healthcare,   
## Var\_1=Cat\_6,   
## age\_dRanked\_Bins=18-25} => {Segmentation=D} 0.03585896 0.9795082 0.03660915 3.715664 239  
## [8] {Profession=Healthcare,   
## Spending\_Score=Low,   
## age\_dRanked\_Bins=18-25} => {Segmentation=D} 0.07096774 0.9792961 0.07246812 3.714860 473  
## [9] {Profession=Healthcare,   
## Spending\_Score=Low,   
## Family\_Size=4,   
## age\_dRanked\_Bins=18-25} => {Segmentation=D} 0.02580645 0.9772727 0.02640660 3.707184 172  
## [10] {Profession=Healthcare,   
## Family\_Size=4,   
## age\_dRanked\_Bins=18-25} => {Segmentation=D} 0.02610653 0.9720670 0.02685671 3.687437 174

Sort by Confidence

RP4 <- sort(RP4,decreasing= TRUE,by="confidence")  
#inspect(RP4[1:50,])  
arules::inspect(RP4[1:10,])

## lhs rhs support confidence coverage lift count  
## [1] {Gender=Male,   
## Profession=Healthcare,   
## Var\_1=Cat\_6,   
## age\_dRanked\_Bins=18-25} => {Segmentation=D} 0.02325581 0.9872611 0.02355589 3.745074 155  
## [2] {Gender=Male,   
## Profession=Healthcare,   
## Spending\_Score=Low,   
## Var\_1=Cat\_6,   
## age\_dRanked\_Bins=18-25} => {Segmentation=D} 0.02325581 0.9872611 0.02355589 3.745074 155  
## [3] {Gender=Male,   
## Profession=Healthcare,   
## Spending\_Score=Low,   
## age\_dRanked\_Bins=18-25} => {Segmentation=D} 0.04606152 0.9839744 0.04681170 3.732606 307  
## [4] {Profession=Healthcare,   
## Spending\_Score=Low,   
## Var\_1=Cat\_6,   
## age\_dRanked\_Bins=18-25} => {Segmentation=D} 0.03585896 0.9835391 0.03645911 3.730955 239  
## [5] {Profession=Healthcare,   
## Work\_Experience=1,   
## Spending\_Score=Low,   
## age\_dRanked\_Bins=18-25} => {Segmentation=D} 0.02355589 0.9812500 0.02400600 3.722272 157  
## [6] {Gender=Male,   
## Profession=Healthcare,   
## age\_dRanked\_Bins=18-25} => {Segmentation=D} 0.04636159 0.9809524 0.04726182 3.721143 309  
## [7] {Profession=Healthcare,   
## Var\_1=Cat\_6,   
## age\_dRanked\_Bins=18-25} => {Segmentation=D} 0.03585896 0.9795082 0.03660915 3.715664 239  
## [8] {Profession=Healthcare,   
## Spending\_Score=Low,   
## age\_dRanked\_Bins=18-25} => {Segmentation=D} 0.07096774 0.9792961 0.07246812 3.714860 473  
## [9] {Profession=Healthcare,   
## Spending\_Score=Low,   
## Family\_Size=4,   
## age\_dRanked\_Bins=18-25} => {Segmentation=D} 0.02580645 0.9772727 0.02640660 3.707184 172  
## [10] {Profession=Healthcare,   
## Family\_Size=4,   
## age\_dRanked\_Bins=18-25} => {Segmentation=D} 0.02610653 0.9720670 0.02685671 3.687437 174

### Findings-Interesting rules

Segment A All rules have a low confidence and low support no definitive rules to really follow -

Segment B. All rules have a low confidence and low support no definitive rules to really follow

artists with low spending score-

Segment C

artists family size 2 graduated and have been married seems to be the predominant group here

Segment D

healthcare workers are in this segment in a lot of rules with very high confidence along with family size=4 age 18-25

Overall some interesting finds but Nothing definite as support overall is too low for all rules to be considered important.

## k-Nearest Neighbor, Decision Tree, Deep Learning

This section contains all modeling efforts for basic decision tree, kNN, and basic deep learning networks that were explored. In the first part of this script a the *Customer Category* will be omitted from being one of the features used as model input. Based on the previously observed summaries from EDA, the majority of the sample set resides in *Category 6*, , while less than 100 observations reside in the category level with the next largest amounts of customers. Instead, the best model configuration yielded from Part 1 will be applied to Part 2, where this feature will be considered and stratified sampling will be applied to the Customer Category feature in order to see if its consideration improves the model or not.

#==Variables==  
  
# For passing list of columns to remove to subset()  
`%ni%` <- Negate(`%in%`)  
  
# For now, remove Var\_1 & age\_dRanked\_Bins  
# the bins were only really needed for EDA, the ranked values themselves serve  
# the same purpose  
data\_clean <- subset(edaDF, select = -c(Var\_1, ageRanked))  
  
glimpse(data\_clean)

summary(data\_clean)

## Gender Profession Work\_Experience Spending\_Score  
## Female:2988 Artist :2192 1 :2187 Low :3999   
## Male :3677 Healthcare :1077 0 :2133 Average:1662   
## Entertainment: 809 9 : 443 High :1004   
## Doctor : 592 8 : 397   
## Engineer : 582 2 : 259   
## Executive : 505 3 : 235   
## (Other) : 908 (Other):1011   
## Family\_Size Segmentation age\_dRanked\_Bins IsGraduated EverMarriedTRUE  
## 2 :2093 A:1616 18-25 : 667 Mode :logical Mode :logical   
## 3 :1292 B:1572 25-29 : 667 FALSE:2416 FALSE:2721   
## 1 :1243 C:1720 29-33 : 667 TRUE :4249 TRUE :3944   
## 4 :1174 D:1757 33-37 : 667   
## 5 : 522 37-41 : 667   
## 6 : 180 41-45 : 666   
## (Other): 161 (Other):2664

# Determine Majority Vote  
data\_clean %>%   
 dplyr::select(Segmentation) %>%   
 group\_by(Segmentation) %>%   
 dplyr::summarise(RowPct = round(n()/nrow(data\_clean), 3)\*100)

## # A tibble: 4 × 2  
## Segmentation RowPct  
## <fct> <dbl>  
## 1 A 24.2  
## 2 B 23.6  
## 3 C 25.8  
## 4 D 26.4

Based on the results above, the baseline performance for classification in this setting would be **26.4%**.

## Stage Data for Modeling

As mentioned k-Fold Cross Validation will be used for evaluating the performance of each model. The folds that will be used for evaluation will be created in this step.

# Set seed for reproducibility  
set.seed(101)  
  
# Number of observations  
N <- nrow(data\_clean)  
  
# Define number of folds  
kfolds <- 5  
  
# Generate indices for reference of rows to holdout (for each fold)  
holdout <- split(sample(1:N), 1:kfolds)

## Part 1: Determine Configuration for Classifier

### Decision Tree

In this section ranges of *maxdepth* and *minsplit* will be explored for determining best configuration. *GridSearchCV* from the *caret* package will be used to explore tuning of maxdepth, while minsplit will be tuned manually.

**Caret Grid Search Approach**

# Configure control  
train\_control <- trainControl(  
 method = 'cv',  
 number = 5, # number of folds  
 search = 'grid'  
)  
  
# Set seed for reproducibility!  
set.seed(101)  
  
# Configure tuning grid  
# For now, accept all other default values  
dtGrid <- expand.grid(  
 maxdepth = c(3, 5, 10, 15, 20, 25)  
)  
  
# train decision tree while tuning parameters  
# Modeling options: names(getModelInfo())  
model <- train(  
 Segmentation ~ .,  
 data = data\_clean,  
 method = 'rpart2',  
 trControl = train\_control,  
 tuneGrid = dtGrid  
)  
  
# Display summary  
print(model)

## CART   
##   
## 6665 samples  
## 8 predictor  
## 4 classes: 'A', 'B', 'C', 'D'   
##   
## No pre-processing  
## Resampling: Cross-Validated (5 fold)   
## Summary of sample sizes: 5331, 5331, 5332, 5333, 5333   
## Resampling results across tuning parameters:  
##   
## maxdepth Accuracy Kappa   
## 3 0.4840205 0.3097248  
## 5 0.4864226 0.3127296  
## 10 0.4864226 0.3127296  
## 15 0.4864226 0.3127296  
## 20 0.4864226 0.3127296  
## 25 0.4864226 0.3127296  
##   
## Accuracy was used to select the optimal model using the largest value.  
## The final value used for the model was maxdepth = 5.

**Test Minimum Row Count Threshold for Node Split**

# Minimum Number of Samples Required by A Node for Decision Tree  
options <- seq(10, 250, 10) # minimum number of samples required for a node (default 1)  
  
# Perform 5-Fold CV for Decision Tree  
  
# Initialize final output  
cv\_results <- c()  
  
# Iterate over k-Folds  
for (k in 1:kfolds) {  
 # Create test & training set for given fold  
 test <- data\_clean[holdout[[k]], ]  
 train <- data\_clean[-holdout[[k]], ]  
   
 # Remove label from test  
 test\_nolabel <- subset(test, select = -c(Segmentation))  
   
 # Store test labels in separate variable  
 test\_label <- test$Segmentation  
   
 # Initialize object to store option results  
 temp\_results <- c()  
   
 # Iterate over k\_options  
 for (option in options) {  
   
 # Make predictions  
 model <-rpart::rpart(  
 Segmentation ~ .,  
 data=train,  
 control = rpart.control(  
 minsplit=option,  
 maxdepth = 5  
 ),  
 na.action=na.pass  
 )  
   
 # build confusion matrix  
 confusion\_mat <- caret::confusionMatrix(  
 data=predict(  
 model,  
 test\_nolabel,  
 type='class'  
 ),  
 reference= test\_label,  
 mode = 'everything'  
 )  
   
 # Append kth iteration's results in final output objects  
 temp\_results <- append(temp\_results, as.numeric(confusion\_mat$overall['Accuracy'])) # Return evaluation criteria  
   
 }  
   
 # Label outcome  
 names(temp\_results) <- paste0('minsplit\_', options)  
   
 # Add results to final df  
 cv\_results <- rbind(cv\_results, temp\_results)  
}  
  
# Update rownames  
rownames(cv\_results) <- paste0('Fold\_', 1:kfolds)  
  
# Display CV Results  
# Transpose so that folds are columns  
round(rowMeans(t(cv\_results)), digits = 3) # Average results over k-Folds

## minsplit\_10 minsplit\_20 minsplit\_30 minsplit\_40 minsplit\_50 minsplit\_60   
## 0.501 0.501 0.501 0.501 0.501 0.501   
## minsplit\_70 minsplit\_80 minsplit\_90 minsplit\_100 minsplit\_110 minsplit\_120   
## 0.501 0.501 0.501 0.501 0.501 0.501   
## minsplit\_130 minsplit\_140 minsplit\_150 minsplit\_160 minsplit\_170 minsplit\_180   
## 0.501 0.501 0.501 0.501 0.501 0.501   
## minsplit\_190 minsplit\_200 minsplit\_210 minsplit\_220 minsplit\_230 minsplit\_240   
## 0.501 0.501 0.501 0.501 0.501 0.501   
## minsplit\_250   
## 0.501

The minsplit option seems to have yielded insignificant results across the options test; thus, value **250** would be selected for in order to improve model performance with respect to processing (e.g. reduces overall model flexibility).

**Yield Confusion Matrix of Best Performing Model**

# Perform 5-Fold CV for Decision Tree  
  
# Initialize final output  
final\_cm <- data.frame(  
 fold=c(),  
 orig=c(),   
 pred=c()  
)  
cv\_labels <- list()  
cv\_pred <- list()  
  
# Iterate over k-Folds  
for (k in 1:kfolds) {  
 # Create test & training set for given fold  
 test <- data\_clean[holdout[[k]], ]  
 train <- data\_clean[-holdout[[k]], ]  
   
 # Remove label from test  
 test\_nolabel <- subset(test, select = -c(Segmentation))  
   
 # Store test labels in separate variable  
 test\_label <- test$Segmentation  
   
 # Make predictions  
 model <-rpart::rpart(  
 Segmentation ~ .,  
 data=train,  
 control = rpart.control(  
 minsplit = 240,  
 maxdepth = 5  
 ),  
 na.action=na.pass  
 )  
   
 # Predict on test labels  
 predictions <- predict(model, test\_nolabel, type='class')  
   
 # Append kth iteration's confusion matrix to final output dataframe  
 final\_cm <- rbind(  
 final\_cm,  
 data.frame(  
 fold=k,  
 orig=test\_label,  
 pred=predictions  
 )  
 )  
}  
  
# Display final confusion matrix  
table(final\_cm$orig, final\_cm$pred)

##   
## A B C D  
## A 598 397 228 393  
## B 303 494 553 222  
## C 161 353 965 241  
## D 287 138 51 1281

confusionMatrix(final\_cm$orig, final\_cm$pred)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction A B C D  
## A 598 397 228 393  
## B 303 494 553 222  
## C 161 353 965 241  
## D 287 138 51 1281  
##   
## Overall Statistics  
##   
## Accuracy : 0.5008   
## 95% CI : (0.4887, 0.5129)  
## No Information Rate : 0.3206   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.3326   
##   
## Mcnemar's Test P-Value : < 2.2e-16   
##   
## Statistics by Class:  
##   
## Class: A Class: B Class: C Class: D  
## Sensitivity 0.44329 0.35745 0.5370 0.5994  
## Specificity 0.80850 0.79595 0.8449 0.8949  
## Pos Pred Value 0.37005 0.31425 0.5610 0.7291  
## Neg Pred Value 0.85126 0.82564 0.8317 0.8256  
## Prevalence 0.20240 0.20735 0.2696 0.3206  
## Detection Rate 0.08972 0.07412 0.1448 0.1922  
## Detection Prevalence 0.24246 0.23586 0.2581 0.2636  
## Balanced Accuracy 0.62590 0.57670 0.6910 0.7472

Above illustrates the final 5-fold confusion matrix for the best performing decision tree yielded.

# Train on entire dataset and inspect summary of model  
model <-rpart::rpart(  
 Segmentation ~ .,  
 data=data\_clean,  
 control = rpart.control(  
 minsplit = 240,  
 maxdepth = 5  
 ),  
 na.action=na.pass  
)  
  
summary(model)

## Call:  
## rpart::rpart(formula = Segmentation ~ ., data = data\_clean, na.action = na.pass,   
## control = rpart.control(minsplit = 240, maxdepth = 5))  
## n= 6665   
##   
## CP nsplit rel error xerror xstd  
## 1 0.14751426 0 1.0000000 1.0000000 0.007328811  
## 2 0.12408313 1 0.8524857 0.8445395 0.008065994  
## 3 0.01657158 2 0.7284026 0.7326813 0.008290938  
## 4 0.01324368 5 0.6786879 0.6976365 0.008313842  
## 5 0.01000000 6 0.6654442 0.6839853 0.008316753  
##   
## Variable importance  
## Profession age\_dRanked\_Bins Spending\_Score EverMarriedTRUE   
## 47 18 17 10   
## Family\_Size IsGraduated Gender   
## 6 1 1   
##   
## Node number 1: 6665 observations, complexity param=0.1475143  
## predicted class=D expected loss=0.7363841 P(node) =1  
## class counts: 1616 1572 1720 1757  
## probabilities: 0.242 0.236 0.258 0.264   
## left son=2 (5355 obs) right son=3 (1310 obs)  
## Primary splits:  
## Profession splits as LLLLLRLLR, improve=409.8108, (0 missing)  
## age\_dRanked\_Bins splits as RRRLLLLLLL, improve=409.3950, (0 missing)  
## Spending\_Score splits as RLL, improve=336.3352, (0 missing)  
## EverMarriedTRUE < 0.5 to the right, improve=315.6414, (0 missing)  
## IsGraduated < 0.5 to the right, improve=238.6249, (0 missing)  
## Surrogate splits:  
## age\_dRanked\_Bins splits as RLLLLLLLLL, agree=0.863, adj=0.302, (0 split)  
##   
## Node number 2: 5355 observations, complexity param=0.1240831  
## predicted class=C expected loss=0.7073763 P(node) =0.8034509  
## class counts: 1482 1463 1567 843  
## probabilities: 0.277 0.273 0.293 0.157   
## left son=4 (2795 obs) right son=5 (2560 obs)  
## Primary splits:  
## Spending\_Score splits as LRR, improve=216.0664, (0 missing)  
## age\_dRanked\_Bins splits as RRRLLLLLLL, improve=143.5284, (0 missing)  
## Profession splits as RLLLR-LL-, improve=140.5360, (0 missing)  
## EverMarriedTRUE < 0.5 to the right, improve=130.5216, (0 missing)  
## IsGraduated < 0.5 to the right, improve=116.5752, (0 missing)  
## Surrogate splits:  
## EverMarriedTRUE < 0.5 to the left, agree=0.780, adj=0.540, (0 split)  
## Family\_Size splits as LRRRRRRRR, agree=0.684, adj=0.340, (0 split)  
## age\_dRanked\_Bins splits as LLLLLLRRRR, agree=0.635, adj=0.237, (0 split)  
## Profession splits as LLLLR-LR-, agree=0.592, adj=0.146, (0 split)  
## Gender splits as LR, agree=0.538, adj=0.033, (0 split)  
##   
## Node number 3: 1310 observations  
## predicted class=D expected loss=0.3022901 P(node) =0.1965491  
## class counts: 134 109 153 914  
## probabilities: 0.102 0.083 0.117 0.698   
##   
## Node number 4: 2795 observations, complexity param=0.01657158  
## predicted class=A expected loss=0.6318426 P(node) =0.4193548  
## class counts: 1029 669 420 677  
## probabilities: 0.368 0.239 0.150 0.242   
## left son=8 (1101 obs) right son=9 (1694 obs)  
## Primary splits:  
## Profession splits as LRRRR-RR-, improve=74.84454, (0 missing)  
## age\_dRanked\_Bins splits as RRRLLLLLLL, improve=65.67042, (0 missing)  
## IsGraduated < 0.5 to the right, improve=50.11079, (0 missing)  
## Family\_Size splits as LLRRRRRRR, improve=27.18341, (0 missing)  
## EverMarriedTRUE < 0.5 to the right, improve=15.17358, (0 missing)  
## Surrogate splits:  
## IsGraduated < 0.5 to the right, agree=0.621, adj=0.037, (0 split)  
##   
## Node number 5: 2560 observations, complexity param=0.01324368  
## predicted class=C expected loss=0.5519531 P(node) =0.384096  
## class counts: 453 794 1147 166  
## probabilities: 0.177 0.310 0.448 0.065   
## left son=10 (1469 obs) right son=11 (1091 obs)  
## Primary splits:  
## Profession splits as RLLLL-LL-, improve=94.734150, (0 missing)  
## IsGraduated < 0.5 to the left, improve=88.789450, (0 missing)  
## age\_dRanked\_Bins splits as LLLRRRRRRR, improve=48.019680, (0 missing)  
## Spending\_Score splits as RRL, improve= 9.688014, (0 missing)  
## Work\_Experience splits as RRRRRRRRRLLLLLL, improve= 5.755071, (0 missing)  
## Surrogate splits:  
## Spending\_Score splits as RRL, agree=0.631, adj=0.135, (0 split)  
## IsGraduated < 0.5 to the left, agree=0.610, adj=0.085, (0 split)  
##   
## Node number 8: 1101 observations, complexity param=0.01657158  
## predicted class=B expected loss=0.6548592 P(node) =0.1651913  
## class counts: 372 380 250 99  
## probabilities: 0.338 0.345 0.227 0.090   
## left son=16 (670 obs) right son=17 (431 obs)  
## Primary splits:  
## age\_dRanked\_Bins splits as LLLLLLRRRR, improve=21.081180, (0 missing)  
## Family\_Size splits as LLRRRRRRR, improve=12.581220, (0 missing)  
## EverMarriedTRUE < 0.5 to the left, improve= 7.128975, (0 missing)  
## Gender splits as RL, improve= 5.092826, (0 missing)  
## IsGraduated < 0.5 to the right, improve= 3.133601, (0 missing)  
## Surrogate splits:  
## EverMarriedTRUE < 0.5 to the left, agree=0.713, adj=0.267, (0 split)  
##   
## Node number 9: 1694 observations, complexity param=0.01657158  
## predicted class=A expected loss=0.6121606 P(node) =0.2541635  
## class counts: 657 289 170 578  
## probabilities: 0.388 0.171 0.100 0.341   
## left son=18 (1078 obs) right son=19 (616 obs)  
## Primary splits:  
## age\_dRanked\_Bins splits as RRRLLLLLLL, improve=44.29891, (0 missing)  
## Family\_Size splits as LLRRRRRRR, improve=20.79223, (0 missing)  
## IsGraduated < 0.5 to the right, improve=17.62032, (0 missing)  
## EverMarriedTRUE < 0.5 to the right, improve=15.11807, (0 missing)  
## Profession splits as -RLLR-RR-, improve=14.55819, (0 missing)  
## Surrogate splits:  
## Family\_Size splits as LLLRRRRRR, agree=0.699, adj=0.172, (0 split)  
## EverMarriedTRUE < 0.5 to the right, agree=0.684, adj=0.130, (0 split)  
## Profession splits as -RLLL-RL-, agree=0.678, adj=0.115, (0 split)  
## Work\_Experience splits as LLLLLLLLLLRRRRR, agree=0.640, adj=0.010, (0 split)  
## IsGraduated < 0.5 to the right, agree=0.639, adj=0.008, (0 split)  
##   
## Node number 10: 1469 observations  
## predicted class=B expected loss=0.6501021 P(node) =0.2204051  
## class counts: 355 514 449 151  
## probabilities: 0.242 0.350 0.306 0.103   
##   
## Node number 11: 1091 observations  
## predicted class=C expected loss=0.36022 P(node) =0.1636909  
## class counts: 98 280 698 15  
## probabilities: 0.090 0.257 0.640 0.014   
##   
## Node number 16: 670 observations  
## predicted class=A expected loss=0.5865672 P(node) =0.1005251  
## class counts: 277 189 124 80  
## probabilities: 0.413 0.282 0.185 0.119   
##   
## Node number 17: 431 observations  
## predicted class=B expected loss=0.5568445 P(node) =0.06466617  
## class counts: 95 191 126 19  
## probabilities: 0.220 0.443 0.292 0.044   
##   
## Node number 18: 1078 observations  
## predicted class=A expected loss=0.5426716 P(node) =0.1617404  
## class counts: 493 220 99 266  
## probabilities: 0.457 0.204 0.092 0.247   
##   
## Node number 19: 616 observations  
## predicted class=D expected loss=0.4935065 P(node) =0.09242311  
## class counts: 164 69 71 312  
## probabilities: 0.266 0.112 0.115 0.506

### k-Nearest Neighbors

Note:  
kNN requires separate data preparation than Decision Tree as the model is expecting columns to be formatted as numeric values. In this case, values will be mapped to numeric.

# Read in csv file -> use File Explorer to choose  
data <- data\_noID  
  
# Convert columns to expected types  
data <- clean\_data\_cols(data)  
  
# For now, remove Var\_1 & age\_dRanked\_Bins  
# the bins were only really needed for EDA, the ranked values themselves serve  
# the same purpose  
data\_knn <- subset(data, select = -c(Var\_1, age\_dRanked\_Bins))  
  
glimpse(data\_knn)

## Rows: 6,665  
## Columns: 9  
## $ Gender <fct> Male, Female, Male, Male, Male, Female, Female, Female…  
## $ Profession <fct> Healthcare, Engineer, Lawyer, Artist, Healthcare, Heal…  
## $ Work\_Experience <ord> 1, 1, 0, 0, 1, 1, 0, 1, 1, 4, 0, 1, 9, 1, 1, 0, 12, 3,…  
## $ Spending\_Score <ord> Low, Low, High, Average, Low, Low, Low, Average, Low, …  
## $ Family\_Size <ord> 4, 1, 2, 2, 3, 3, 3, 4, 3, 4, 1, 2, 5, 6, 4, 1, 1, 4, …  
## $ Segmentation <fct> D, B, B, C, C, D, D, C, A, D, B, C, D, B, B, C, A, D, …  
## $ ageRanked <ord> 1, 9, 9, 8, 3, 3, 9, 8, 2, 1, 9, 5, 3, 3, 9, 10, 7, 1,…  
## $ IsGraduated <lgl> FALSE, TRUE, TRUE, FALSE, TRUE, TRUE, TRUE, TRUE, TRUE…  
## $ EverMarriedTRUE <lgl> FALSE, TRUE, TRUE, TRUE, FALSE, FALSE, TRUE, TRUE, FAL…

**Convert Character Columns to Integer Representations**

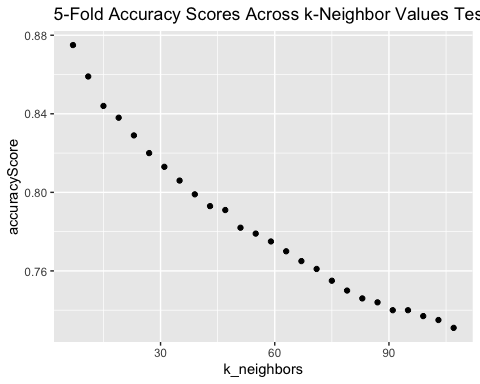
# Select columns to be converted  
target\_cols <- c(  
 "Gender",  
 "Profession",  
 "Spending\_Score",  
 "Work\_Experience",  
 "Family\_Size",  
 "Segmentation",  
 "ageRanked"  
)  
  
# Iterate through columns, create new columns in data for integer representation  
for (col in target\_cols) {  
 data\_knn[, paste(col, '\_int', sep='')] <- unclass(edaDF[,col])  
}  
  
# Inspect results  
glimpse(data\_knn)

## Rows: 6,665  
## Columns: 16  
## $ Gender <fct> Male, Female, Male, Male, Male, Female, Female, Fe…  
## $ Profession <fct> Healthcare, Engineer, Lawyer, Artist, Healthcare, …  
## $ Work\_Experience <ord> 1, 1, 0, 0, 1, 1, 0, 1, 1, 4, 0, 1, 9, 1, 1, 0, 12…  
## $ Spending\_Score <ord> Low, Low, High, Average, Low, Low, Low, Average, L…  
## $ Family\_Size <ord> 4, 1, 2, 2, 3, 3, 3, 4, 3, 4, 1, 2, 5, 6, 4, 1, 1,…  
## $ Segmentation <fct> D, B, B, C, C, D, D, C, A, D, B, C, D, B, B, C, A,…  
## $ ageRanked <ord> 1, 9, 9, 8, 3, 3, 9, 8, 2, 1, 9, 5, 3, 3, 9, 10, 7…  
## $ IsGraduated <lgl> FALSE, TRUE, TRUE, FALSE, TRUE, TRUE, TRUE, TRUE, …  
## $ EverMarriedTRUE <lgl> FALSE, TRUE, TRUE, TRUE, FALSE, FALSE, TRUE, TRUE,…  
## $ Gender\_int <int> 2, 1, 2, 2, 2, 1, 1, 1, 1, 2, 1, 1, 1, 2, 2, 1, 2,…  
## $ Profession\_int <int> 6, 3, 8, 1, 6, 6, 3, 1, 3, 6, 2, 6, 7, 6, 4, 1, 7,…  
## $ Spending\_Score\_int <int> 1, 1, 3, 2, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 2, 3, 1,…  
## $ Work\_Experience\_int <int> 2, 2, 1, 1, 2, 2, 1, 2, 2, 5, 1, 2, 10, 2, 2, 1, 1…  
## $ Family\_Size\_int <int> 4, 1, 2, 2, 3, 3, 3, 4, 3, 4, 1, 2, 5, 6, 4, 1, 1,…  
## $ Segmentation\_int <int> 4, 2, 2, 3, 3, 4, 4, 3, 1, 4, 2, 3, 4, 2, 2, 3, 1,…  
## $ ageRanked\_int <int> 1, 9, 9, 8, 3, 3, 9, 8, 2, 1, 9, 5, 3, 3, 9, 10, 7…

# Initial guess for k-Nearest Neighbors Approach  
k\_guess <- round(sqrt(nrow(data\_knn)))  
options <- seq(k\_guess-75, k\_guess+25, 4)  
  
# Perform 5-Fold CV for k-Nearest Neighbors Approach  
  
# Initialize final output  
cv\_results <- c()  
  
# Iterate over k-Folds  
for (k in 1:kfolds) {  
 # Create test & training set for given fold  
 test <- data\_knn[holdout[[k]],]  
 train <- data\_knn[-holdout[[k]],]  
   
 # Remove original columns from temporary dataframe  
 train <- subset(train, select = names(train) %ni% target\_cols)  
 test <- subset(test, select = names(test) %ni% target\_cols)  
   
 # Remove label from test  
 test\_nolabel <- subset(test, select = -c(Segmentation\_int))  
   
 # Store test labels in separate variable  
 test\_label <- test$Segmentation\_int  
   
 # Initialize object to store option results  
 temp\_results <- c()  
   
 # Iterate over k\_options  
 for (option in options) {  
   
 # Make predictions  
 pred <- knn(  
 train=train,  
 test=test,  
 cl=train$Segmentation\_int,  
 k=option,  
 prob=FALSE  
 )  
   
 confusion\_mat <- caret::confusionMatrix(  
 pred,  
 factor(test\_label) # predictions returning back as factors, test labels are integers; quick fix  
 )  
   
 # Append kth iteration's results in final output objects  
 temp\_results <- append(temp\_results, as.numeric(confusion\_mat$overall['Accuracy'])) # Return evaluation criteria  
   
 }  
   
 # Label outcome  
 names(temp\_results) <- paste0('kValue\_', options)  
   
 # Add results to final df  
 cv\_results <- rbind(cv\_results, temp\_results)  
}  
  
# Update rownames  
rownames(cv\_results) <- paste0('Fold\_', 1:kfolds)  
  
# Display CV Results  
# Transpose so that folds are columns  
round(rowMeans(t(cv\_results)), digits = 3) # Average results over k-Folds

## kValue\_7 kValue\_11 kValue\_15 kValue\_19 kValue\_23 kValue\_27 kValue\_31   
## 0.875 0.859 0.844 0.838 0.829 0.820 0.813   
## kValue\_35 kValue\_39 kValue\_43 kValue\_47 kValue\_51 kValue\_55 kValue\_59   
## 0.806 0.799 0.793 0.791 0.782 0.779 0.775   
## kValue\_63 kValue\_67 kValue\_71 kValue\_75 kValue\_79 kValue\_83 kValue\_87   
## 0.770 0.765 0.761 0.755 0.750 0.746 0.744   
## kValue\_91 kValue\_95 kValue\_99 kValue\_103 kValue\_107   
## 0.740 0.740 0.737 0.735 0.731

data.frame(  
 k\_neighbors=options,  
 accuracyScore=round(rowMeans(t(cv\_results)), digits = 3)  
) %>%   
dplyr::select(k\_neighbors, accuracyScore) %>%   
ggplot( aes(x=k\_neighbors, y=accuracyScore) ) +   
 geom\_point() +  
 labs(  
 title='5-Fold Accuracy Scores Across k-Neighbor Values Tested'  
 )



**Yield Confusion Matrix of Best Performing Model**

# Perform 5-Fold CV for k-Nearest Neighbors Approach  
  
# Initialize final output  
final\_cm <- data.frame(  
 fold=c(),  
 orig=c(),   
 pred=c()  
)  
cv\_labels <- list()  
cv\_pred <- list()  
  
# Iterate over k-Folds  
for (k in 1:kfolds) {  
 # Create test & training set for given fold  
 test <- data\_knn[holdout[[k]], ]  
 train <- data\_knn[-holdout[[k]], ]  
   
 # Remove original columns from temporary dataframe  
 train <- subset(train, select = names(train) %ni% target\_cols)  
 test <- subset(test, select = names(test) %ni% target\_cols)  
   
 # Remove label from test  
 test\_nolabel <- subset(test, select = -c(Segmentation\_int))  
   
 # Store test labels in separate variable  
 test\_label <- test$Segmentation\_int  
   
 # Make predictions  
 predictions <- knn(  
 train=train,  
 test=test,  
 cl=train$Segmentation\_int,  
 k=7,  
 prob=FALSE  
 )  
   
 # Append kth iteration's confusion matrix to final output dataframe  
 final\_cm <- rbind(  
 final\_cm,  
 data.frame(  
 fold=k,  
 orig=factor(test\_label), # predictions returning back as factors, test labels are integers; quick fix  
 pred=predictions  
 )  
 )  
}  
  
# Display final confusion matrix  
# Map   
#table(final\_cm$orig, final\_cm$pred)  
confusionMatrix(final\_cm$orig, final\_cm$pred)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 1 2 3 4  
## 1 1436 176 4 0  
## 2 152 1298 115 7  
## 3 1 87 1488 144  
## 4 1 6 139 1611  
##   
## Overall Statistics  
##   
## Accuracy : 0.8752   
## 95% CI : (0.867, 0.883)  
## No Information Rate : 0.2644   
## P-Value [Acc > NIR] : <2e-16   
##   
## Kappa : 0.8334   
##   
## Mcnemar's Test P-Value : 0.1972   
##   
## Statistics by Class:  
##   
## Class: 1 Class: 2 Class: 3 Class: 4  
## Sensitivity 0.9031 0.8283 0.8522 0.9143  
## Specificity 0.9645 0.9463 0.9528 0.9702  
## Pos Pred Value 0.8886 0.8257 0.8651 0.9169  
## Neg Pred Value 0.9695 0.9472 0.9478 0.9692  
## Prevalence 0.2386 0.2351 0.2620 0.2644  
## Detection Rate 0.2155 0.1947 0.2233 0.2417  
## Detection Prevalence 0.2425 0.2359 0.2581 0.2636  
## Balanced Accuracy 0.9338 0.8873 0.9025 0.9423

### Deep Learning Neural Network

Note: Several approaches were tested for training a basic deep learning network, but each resulted in some form of internal (dimension-related issue). While testing number of layers in the deep learning network and threshold values (in a similar fashion seen for k-fold Neighbors), we kept running into convergence error messages. Even after simplifying the k-fold approach (e.g. fixing all hyper parameters of the neural network) we still resulted in an error message. For these reasons, this modeling approach has been shelved for this project.

## Naive Bayes and SVM

Our analysis will now test Naive Bayes and SVM to determine if we can find better accuracy than that which we found using previous modeling techniques.

### NB and SVM Cleaning

set.seed(12345)  
  
Train<-bd  
  
dt<- sort(sample(nrow(Train),nrow(Train)\*.9))  
Rtrain<-Train[dt,]  
RtestL<-Train[-dt,]  
Rtest <- RtestL[,-1]  
Rlabel <- RtestL$Segmentation

# Function for model evaluation for our 4 by 4 table  
  
get\_accuracy\_rate <- function(results\_table, total\_cases) {  
 diagonal\_sum <- sum(c(results\_table[[1]], results\_table[[6]], results\_table[[11]], results\_table[[16]]))   
 ((diagonal\_sum / total\_cases)\*100)   
 }

#### Naive Bayes

#training  
NB<- naiveBayes(Segmentation~., data=Rtrain, na.action = na.pass)  
#testing  
NBP <-predict(NB,RtestL)

NBC <- confusionMatrix(NBP,Rlabel, mode = "everything")  
NBC

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction A B C D  
## A 83 25 21 34  
## B 21 37 10 14  
## C 36 56 121 5  
## D 23 19 30 132  
##   
## Overall Statistics  
##   
## Accuracy : 0.5592   
## 95% CI : (0.5206, 0.5973)  
## No Information Rate : 0.2774   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.4052   
##   
## Mcnemar's Test P-Value : 1.749e-10   
##   
## Statistics by Class:  
##   
## Class: A Class: B Class: C Class: D  
## Sensitivity 0.5092 0.27007 0.6648 0.7135  
## Specificity 0.8413 0.91509 0.8000 0.8506  
## Pos Pred Value 0.5092 0.45122 0.5550 0.6471  
## Neg Pred Value 0.8413 0.82906 0.8641 0.8855  
## Precision 0.5092 0.45122 0.5550 0.6471  
## Recall 0.5092 0.27007 0.6648 0.7135  
## F1 0.5092 0.33790 0.6050 0.6787  
## Prevalence 0.2444 0.20540 0.2729 0.2774  
## Detection Rate 0.1244 0.05547 0.1814 0.1979  
## Detection Prevalence 0.2444 0.12294 0.3268 0.3058  
## Balanced Accuracy 0.6752 0.59258 0.7324 0.7821

Naive Bayes returned an accuracy of 56%, which is less than some of our other modeling techniques, so our analysis will proceed with Support Vector Machines.

#### SVM

get data ready for K folds

N <- nrow(Train)  
kfolds <- 5  
set.seed(30)  
holdout <- split(sample(1:N), 1:kfolds)

# Baseline SVM - no changes to data  
all\_results <- data.frame(orig=c(), pred=c())  
for (k in 1:kfolds) {  
 new\_test <- Train[holdout[[k]], ]  
 new\_train <- Train[-holdout[[k]], ]  
  
 new\_test\_no\_label <- new\_test[-c(7)]  
 new\_test\_just\_label <- new\_test[c(7)]  
  
 test\_model <- svm(Segmentation ~ ., new\_train, na.action=na.pass)  
 pred <- predict(test\_model, new\_test\_no\_label, type=c("class"))  
  
 all\_results <- rbind(all\_results,  
data.frame(orig=new\_test\_just\_label$Segmentation, pred=pred))  
}  
  
  
table(all\_results$orig, all\_results$pred)

##   
## A B C D  
## A 844 163 323 286  
## B 528 226 661 157  
## C 250 132 1123 215  
## D 389 92 78 1198

get\_accuracy\_rate(table(all\_results$orig, all\_results$pred), length(all\_results$pred))

## [1] 50.87772

Base SVM model returned accuracy of 50.88%.

### Pluggin into this variable to make it work with old code, is not binarized

binarized\_svm\_trainset <- Train

#### Polynomial Kernel

all\_results <- data.frame(orig=c(), pred=c())  
for (k in 1:kfolds) {  
 new\_test <- binarized\_svm\_trainset[holdout[[k]], ]  
 new\_train <- binarized\_svm\_trainset[-holdout[[k]], ]  
   
 new\_test\_no\_label <- new\_test[-c(7)]  
 new\_test\_just\_label <- new\_test[c(7)]  
   
 test\_model <- svm(Segmentation ~ ., new\_train, kernel="polynomial", na.action=na.pass)  
 pred <- predict(test\_model, new\_test\_no\_label, type=c("class"))  
   
 all\_results <- rbind(all\_results, data.frame(orig=new\_test\_just\_label$Segmentation, pred=pred))  
}  
table(all\_results$orig, all\_results$pred)

##   
## A B C D  
## A 0 0 319 1297  
## B 0 0 322 1250  
## C 0 0 327 1393  
## D 0 0 365 1392

get\_accuracy\_rate(table(all\_results$orig, all\_results$pred), length(all\_results$pred))

## [1] 25.79145

Polynomial Kernel returned accuracy of 25.79%.

#### Radial Kernel

all\_results <- data.frame(orig=c(), pred=c())  
for (k in 1:kfolds) {  
 new\_test <- binarized\_svm\_trainset[holdout[[k]], ]  
 new\_train <- binarized\_svm\_trainset[-holdout[[k]], ]  
   
 new\_test\_no\_label <- new\_test[-c(7)]  
 new\_test\_just\_label <- new\_test[c(7)]  
   
 test\_model <- svm(Segmentation ~ ., new\_train, kernel="radial", na.action=na.pass)  
 pred <- predict(test\_model, new\_test\_no\_label, type=c("class"))  
   
 all\_results <- rbind(all\_results, data.frame(orig=new\_test\_just\_label$Segmentation, pred=pred))  
}  
table(all\_results$orig, all\_results$pred)

##   
## A B C D  
## A 844 163 323 286  
## B 528 226 661 157  
## C 250 132 1123 215  
## D 389 92 78 1198

get\_accuracy\_rate(table(all\_results$orig, all\_results$pred), length(all\_results$pred))

## [1] 50.87772

Radial Kernel returned accuracy of 50.88%

#### Sigmoid Kernel

all\_results <- data.frame(orig=c(), pred=c())  
for (k in 1:kfolds) {  
 new\_test <- binarized\_svm\_trainset[holdout[[k]], ]  
 new\_train <- binarized\_svm\_trainset[-holdout[[k]], ]  
   
 new\_test\_no\_label <- new\_test[-c(7)]  
 new\_test\_just\_label <- new\_test[c(7)]  
   
 test\_model <- svm(Segmentation ~ ., new\_train, kernel="sigmoid", na.action=na.pass)  
 pred <- predict(test\_model, new\_test\_no\_label, type=c("class"))  
   
 all\_results <- rbind(all\_results, data.frame(orig=new\_test\_just\_label$Segmentation, pred=pred))  
}  
table(all\_results$orig, all\_results$pred)

##   
## A B C D  
## A 841 126 354 295  
## B 543 147 724 158  
## C 268 72 1163 217  
## D 390 79 101 1187

get\_accuracy\_rate(table(all\_results$orig, all\_results$pred), length(all\_results$pred))

## [1] 50.08252

Sigmoid kernel accuracy was 50.08%

Highest accuracy returned with SVM was 50.88%, still less than other modeling techniques tested.

### Random Forest

# Set seed for reproducibility  
set.seed(101)  
  
# Number of observations  
N <- nrow(data\_clean)  
  
# Define number of folds  
kfolds <- 5  
  
# Generate indices for reference of rows to holdout (for each fold)  
holdout <- split(sample(1:N), 1:kfolds)

## Part 1: Determine Configuration for Classifier

### Random Forest

**Test Number of Trees**

# Number of Trees for RandomForest  
ntree\_options <- c(250, 500, 750, 1000) # (default 500)  
  
# Perform 5-Fold CV for Random Forest Approach  
  
# Initialize final output  
cv\_results <- c()  
  
# Iterate over k-Folds  
for (k in 1:kfolds) {  
 # Create test & training set for given fold  
 test <- data\_clean[holdout[[k]], ]  
 train <- data\_clean[-holdout[[k]], ]  
   
 # Remove label from test  
 test\_nolabel <- subset(test, select = -c(Segmentation))  
   
 # Store test labels in separate variable  
 test\_label <- test$Segmentation  
   
 # Initialize object to store option results  
 temp\_results <- c()  
   
 # Iterate over ntree\_options  
 for (option in ntree\_options) {  
   
 # Make predictions  
 model <- randomForest(  
 Segmentation ~ .,  
 data=train,  
 ntree=option,  
 na.action=na.pass  
 )  
   
 # build confusion matrix  
 confusion\_mat <- caret::confusionMatrix(  
 data=predict(  
 model,  
 test\_nolabel,  
 type='class'  
 ),  
 reference= test\_label,  
 mode = 'everything'  
 )  
   
 # Append kth iteration's results in final output objects  
 temp\_results <- append(temp\_results, as.numeric(confusion\_mat$overall['Accuracy'])) # Return evaluation criteria  
   
 }  
   
 # Label outcome  
 names(temp\_results) <- paste0('ntree\_', ntree\_options)  
   
 # Add results to final df  
 cv\_results <- rbind(cv\_results, temp\_results)  
}  
  
# Update rownames  
rownames(cv\_results) <- paste0('Fold\_', 1:kfolds)  
  
# Display CV Results  
# Transpose so that folds are columns  
round(rowMeans(t(cv\_results)), digits = 3) # Average results over k-Folds

## ntree\_250 ntree\_500 ntree\_750 ntree\_1000   
## 0.529 0.531 0.534 0.532

**Yield Confusion Matrix of Best Performing Model**

The following process should be a 5-fold run using the final configured model in order to yield a confusion matrix for such.

# Perform 5-Fold CV for Decision Tree  
  
# Initialize final output  
final\_cm <- data.frame(  
 fold=c(),  
 orig=c(),   
 pred=c()  
)  
cv\_labels <- list()  
cv\_pred <- list()  
  
# Iterate over k-Folds  
for (k in 1:kfolds) {  
 # Create test & training set for given fold  
 test <- data\_clean[holdout[[k]], ]  
 train <- data\_clean[-holdout[[k]], ]  
   
 # Remove label from test  
 test\_nolabel <- subset(test, select = -c(Segmentation))  
   
 # Store test labels in separate variable  
 test\_label <- test$Segmentation  
   
 # Make predictions  
 model <- randomForest(  
 Segmentation ~ .,  
 data=train,  
 ntree=500, # change this & any other param to what you yield!  
 na.action=na.pass  
 )  
   
 # Predict on test labels  
 predictions <- predict(model, test\_nolabel, type='class')  
   
 # Append kth iteration's confusion matrix to final output dataframe  
 final\_cm <- rbind(  
 final\_cm,  
 data.frame(  
 fold=k,  
 orig=test\_label,  
 pred=predictions  
 )  
 )  
}  
  
# Display final confusion matrix  
#table(final\_cm$orig, final\_cm$pred)  
confusionMatrix(final\_cm$orig, final\_cm$pred)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction A B C D  
## A 810 285 206 315  
## B 407 470 519 176  
## C 175 286 1029 230  
## D 372 101 36 1248  
##   
## Overall Statistics  
##   
## Accuracy : 0.5337   
## 95% CI : (0.5216, 0.5457)  
## No Information Rate : 0.2954   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.3768   
##   
## Mcnemar's Test P-Value : < 2.2e-16   
##   
## Statistics by Class:  
##   
## Class: A Class: B Class: C Class: D  
## Sensitivity 0.4592 0.41156 0.5749 0.6338  
## Specificity 0.8355 0.80047 0.8583 0.8916  
## Pos Pred Value 0.5012 0.29898 0.5983 0.7103  
## Neg Pred Value 0.8111 0.86805 0.8461 0.8531  
## Prevalence 0.2647 0.17134 0.2686 0.2954  
## Detection Rate 0.1215 0.07052 0.1544 0.1872  
## Detection Prevalence 0.2425 0.23586 0.2581 0.2636  
## Balanced Accuracy 0.6474 0.60601 0.7166 0.7627

### XGBoost

Extreme Gradient Boosting is an ensemble machine learning technique that utilizes a large number of decision trees. XGBoost is similar to Random Forest, another ensemble classifier used earlier in our analysis, but Random Forest utilizes tree bagging while XGBoost utilizes boosted trees. The primary difference between bagging (Random Forest) and boosting (XGBoost) is that bagging grows a large number of decision trees, then takes the average result across all of the trees. Boosting looks to learn from the first trees created, and evolves subsequent trees to minimize error detected in the prior trees. While XGBoost is sensitive to overfitting, when parameters are carefully tuned it can produce outstanding predictive accuracy.

Our initial goal in utilizing XGBoost was to improve on the poor accuracy results from our Random Forest model, however in researching XGBoost we learned that XGBoost does not work well with multinomial categorical data like our automobile customer data. XGBoost works best with binomial numeric data because it requires transforming the data into a sparse matrix. When there are multiple predictor classes, this exponentially increases the dimensionality of the data set, making it difficult to derive informative predictions from its output. For those reasons, we decided to shelve XGBoost as a predictor for this project.

Further examination however, suggested that we could still utilize XGBoost to better understand how the original segments were created. ARM did not yield definitive results, perhaps XGBoost could improve our understanding of the profiles of each existing customer segment. To accomplish this task, we need to convert our multinomial data set into four separate binomial predicts representing inclusion or exclusion from each of the individual segments.

library(klaR)  
#create a copy of eda dataframe for xgb without age\_dRanked\_bins (which was added for eda viz purposes)  
xgDF<-edaDF[,-9]  
  
#convert to data table  
df<- data.table(xgDF, keep.rownames = FALSE)  
  
  
# convert to sparse matrix  
sparse\_matrix<-sparse.model.matrix(Segmentation~.,data=xgDF)[,-1]  
head(sparse\_matrix)

## 6 x 50 sparse Matrix of class "dgCMatrix"

## [[ suppressing 50 column names 'GenderMale', 'ProfessionDoctor', 'ProfessionEngineer' ... ]]

##   
## 1 1 . . . . 1 . . . -0.3585686 0.2698687 -0.06517949 -0.1687035 0.3516842  
## 3 . . 1 . . . . . . -0.3585686 0.2698687 -0.06517949 -0.1687035 0.3516842  
## 4 1 . . . . . . 1 . -0.4183300 0.4722703 -0.45625644 0.3936414 -0.3077236  
## 6 1 . . . . . . . . -0.4183300 0.4722703 -0.45625644 0.3936414 -0.3077236  
## 7 1 . . . . 1 . . . -0.3585686 0.2698687 -0.06517949 -0.1687035 0.3516842  
## 8 . . . . . 1 . . . -0.3585686 0.2698687 -0.06517949 -0.1687035 0.3516842  
##   
## 1 -0.4380037 0.4255756 -0.34515908 0.23907119 -0.14198182 0.071637612  
## 3 -0.4380037 0.4255756 -0.34515908 0.23907119 -0.14198182 0.071637612  
## 4 0.2190019 -0.1418585 0.08331426 -0.04403943 0.02070568 -0.008499378  
## 6 0.2190019 -0.1418585 0.08331426 -0.04403943 0.02070568 -0.008499378  
## 7 -0.4380037 0.4255756 -0.34515908 0.23907119 -0.14198182 0.071637612  
## 8 -0.4380037 0.4255756 -0.34515908 0.23907119 -0.14198182 0.071637612  
##   
## 1 -0.029959342 0.009844668 -0.0022103751 -7.071068e-01 0.4082483 -0.1290994  
## 3 -0.029959342 0.009844668 -0.0022103751 -7.071068e-01 0.4082483 -0.5163978  
## 4 0.002953738 -0.000820389 0.0001578839 7.071068e-01 0.4082483 -0.3872983  
## 6 0.002953738 -0.000820389 0.0001578839 -9.073800e-17 -0.8164966 -0.3872983  
## 7 -0.029959342 0.009844668 -0.0022103751 -7.071068e-01 0.4082483 -0.2581989  
## 8 -0.029959342 0.009844668 -0.0022103751 -7.071068e-01 0.4082483 -0.2581989  
##   
## 1 -0.3228883 0.2860388 0.2011456 -0.4160251 0.02247333 0.47795212  
## 3 0.5318160 -0.4449492 0.3128931 -0.1849001 0.08989331 -0.03413944  
## 4 0.1329540 0.2224746 -0.4693397 0.5084752 -0.38204659 0.20483662  
## 6 0.1329540 0.2224746 -0.4693397 0.5084752 -0.38204659 0.20483662  
## 7 -0.1519474 0.4131671 -0.2458446 -0.1849001 0.49441323 -0.47795212  
## 8 -0.1519474 0.4131671 -0.2458446 -0.1849001 0.49441323 -0.47795212  
##   
## 1 -0.493626826 . . 1 . . . -0.4954337 0.52223297 -0.4534252 0.3365809  
## 3 0.008814765 . . . . 1 . 0.3853373 0.17407766 -0.1511417 -0.4113767  
## 4 -0.070518118 . . . . 1 . 0.3853373 0.17407766 -0.1511417 -0.4113767  
## 6 -0.070518118 . . . . 1 . 0.2752409 -0.08703883 -0.3778543 -0.3178820  
## 7 0.246813413 . . . . 1 . -0.2752409 -0.08703883 0.3778543 -0.3178820  
## 8 0.246813413 . . . . 1 . -0.2752409 -0.08703883 0.3778543 -0.3178820  
##   
## 1 -0.21483446 0.1167748 -0.05269379 0.01869894 -0.004535159 . .  
## 3 -0.50128041 -0.4281744 -0.27517866 -0.13089258 -0.040816431 1 1  
## 4 -0.50128041 -0.4281744 -0.27517866 -0.13089258 -0.040816431 1 1  
## 6 0.03580574 0.3892495 0.50351840 0.37397880 0.163265726 . 1  
## 7 -0.03580574 0.3892495 -0.50351840 0.37397880 -0.163265726 1 .  
## 8 -0.03580574 0.3892495 -0.50351840 0.37397880 -0.163265726 1 .

With data converted to sparse matrix, the multinomial data can be converted to four separate output\_vectors with a binary decision, included or excluded from the given customer segment. Four individual xgboost models will be created (one for each segment) then feature importance will be measured and plotted for each segment.

#create 4 output vectors  
outA = df[,Segmentation]=='A'  
outB<-df[,Segmentation]=='B'  
outC<-df[,Segmentation]=='C'  
outD<-df[,Segmentation]=='D'  
  
#create 4 xgb models  
xgA<-xgboost::xgboost(data=sparse\_matrix, label=outA, max\_depth=4, eta=1, nthread=2, nrounds=10, objective = "binary:logistic")

## [1] train-logloss:0.506226   
## [2] train-logloss:0.484087   
## [3] train-logloss:0.473800   
## [4] train-logloss:0.463234   
## [5] train-logloss:0.459558   
## [6] train-logloss:0.456045   
## [7] train-logloss:0.450836   
## [8] train-logloss:0.444917   
## [9] train-logloss:0.440810   
## [10] train-logloss:0.437281

xgB<-xgboost::xgboost(data=sparse\_matrix, label=outB, max\_depth=4, eta=1, nthread=2, nrounds=10, objective = "binary:logistic")

## [1] train-logloss:0.519200   
## [2] train-logloss:0.499360   
## [3] train-logloss:0.491884   
## [4] train-logloss:0.487746   
## [5] train-logloss:0.482355   
## [6] train-logloss:0.476116   
## [7] train-logloss:0.472686   
## [8] train-logloss:0.467270   
## [9] train-logloss:0.463412   
## [10] train-logloss:0.458738

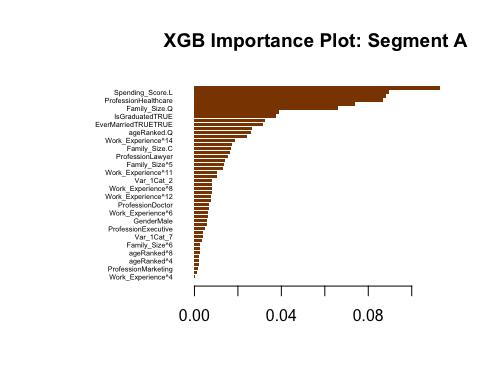
xgC<-xgboost::xgboost(data=sparse\_matrix, label=outC, max\_depth=4, eta=1, nthread=2, nrounds=10, objective = "binary:logistic")

## [1] train-logloss:0.474441   
## [2] train-logloss:0.449817   
## [3] train-logloss:0.435770   
## [4] train-logloss:0.427154   
## [5] train-logloss:0.421269   
## [6] train-logloss:0.416001   
## [7] train-logloss:0.409655   
## [8] train-logloss:0.403573   
## [9] train-logloss:0.400615   
## [10] train-logloss:0.396220

xgD<-xgboost::xgboost(data=sparse\_matrix, label=outD, max\_depth=4, eta=1, nthread=2, nrounds=10, objective = "binary:logistic")

## [1] train-logloss:0.409088   
## [2] train-logloss:0.369601   
## [3] train-logloss:0.355739   
## [4] train-logloss:0.347163   
## [5] train-logloss:0.338728   
## [6] train-logloss:0.330762   
## [7] train-logloss:0.325187   
## [8] train-logloss:0.319417   
## [9] train-logloss:0.315222   
## [10] train-logloss:0.311537

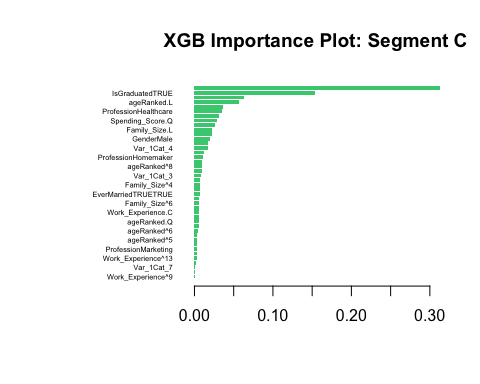
#evaluate feature importance for all 4 models  
importanceA<-xgb.importance(feature\_names = colnames(sparse\_matrix), model = xgA)  
importanceB<-xgb.importance(feature\_names = colnames(sparse\_matrix), model = xgB)  
importanceC<-xgb.importance(feature\_names = colnames(sparse\_matrix), model = xgC)  
importanceD<-xgb.importance(feature\_names = colnames(sparse\_matrix), model = xgD)  
  
#plot each feature importance table  
xgb.plot.importance(importance\_matrix=importanceA, col='darkorange4',main='XGB Importance Plot: Segment A')



xgb.plot.importance(importance\_matrix=importanceB, col='cyan4',main='XGB Importance Plot: Segment B')



xgb.plot.importance(importance\_matrix=importanceC, col='seagreen3',main='XGB Importance Plot: Segment C')



xgb.plot.importance(importance\_matrix=importanceD, col='orchid3',main='XGB Importance Plot: Segment D')

 #### XGBoost for Feature Importance Findings

\*NB: These findings are slightly different than those we presented during our final presentation. In evaluation of those results, we realized we had unnecessarily further partitioned the training data set into smaller test and train data sets. We decided to use the full training data set in our final modeling to generate a more accurate picture of each segment.

* Segment A: Age, Spending Score, and Profession = Healthcare were all important factors in determining inclusion in segment A
* Segment B: Age, IsGraduated = TRUE, Profession = Entertainment were all important factors in determining inclusion in segment B
* Segment C: Spending Score, IsGraduated = TRUE, Age were all important factors in determining inclusion in segment C
* Segment D: Age, Profession = Healthcare, Spending Score were all important factors in determining inclusion in segment D

Clearly Age, education status, spending score, and profession are important in evaluating the segmentation process through XGBoost. Unfortunately the output is not as clear and definitive as we hoped and the results mirror what we saw in ARM. Perhaps future analysis could re-evaluate the parameters or find more true numeric data on these customers to improve results, but that is out of scope for this analysis.

# Conclusions

models<- c("Decision Tree", "kNN", "Naive Bayes", "SVM - Radial", "Random Forest")  
accuracies <- c(50.08, 87.52, 56, 50.88, 53.37)  
model\_comp <- data.frame(models, accuracies)  
print(dplyr::arrange(model\_comp, -accuracies))

## models accuracies  
## 1 kNN 87.52  
## 2 Naive Bayes 56.00  
## 3 Random Forest 53.37  
## 4 SVM - Radial 50.88  
## 5 Decision Tree 50.08

The dataframe output above highlights the different modeling techniques we performed in this analysis and their best accuracy. These accuracies reflect significant tuning, but are a great way to compare algorithm effectiveness in predicting the correct customer segmentation for this automobile company. Most of the models performed comprably, with accuracies around 50%, but kNN significantly outperformed the others with 87.52% predictive accuracy. In our analysis, kNN was the winner.

## Final Thoughts

Our analysis set out to determine which of several modeling techniques would be best suited for predicting customer segments given a pre-clustered training data set by an automobile manufacturer. We tested various iterations of kNN, Naive Bayes, Random Forest, SVM, Decision Trees, XGBoost, kModes clustering, and Association Rule mining. Across those eight modeling techniques, we received actionable results from five (kNN, Naive Bayes, Random Forest, SVM, Decision Tree). kModes, XGBoost, and Association Rule Mining gave us some insight into the profiles of each customer segment, but these profiles were not definitive and further analysis is needed to better understand how each segment is unique. However, our goal was finding the best predictive model, which turned out to be kNN.

This data set presented unique challenges, most notably its prevalence of categorical data. Our recommendation to this company would be to start collecting a broader array of data, especially numeric data, as it would open up more options for modeling their consumers and may yield more accurate segmentation. This is an important next step for this automobile manufacturer because efficient and accurate segmentation is critical for success in an increasingly data literate marketplace.

Customer segmentation is a vital pillar of data modeling-based decision making for large corporations. Accurate segments help companies advertise, launch new products, and execute pricing strategies. An excellent pricing strategy for a great new product is only successful if targeted to the right consumer, in the right place, at the right time. We advise this company to expand its data collection so future analysis can be more insightful, creating more accurate customer segments, and unlocking potential for higher ROI and profitability.

**Resources**

Exploring Missing Data  
<https://cran.r-project.org/web/packages/naniar/vignettes/naniar-visualisation.html>

How to Write DataFrame to CSV in R  
<https://datatofish.com/export-dataframe-to-csv-in-r/>

Use File Explorer from R  
<https://statisticsglobe.com/file-choose-function-r>

Difference Between Customer Segments and Customer Categories  
<https://commence.com/blog/2020/11/12/customer-segments-vs-customer-archetypes/>

**rpart** Decision Tree Parameters  
<https://www.learnbymarketing.com/tutorials/rpart-decision-trees-in-r/>

**caret** GridSearchCV Hyperparameter Tuning Approach  
<https://www.projectpro.io/recipes/tune-hyper-parameters-grid-search-r>

MinSplit in R  
<https://medium.com/talking-with-data/minsplit-and-minbucket-a49ff56026c8>

Configure **rpart** While Using GridSearchCV  
<https://stackoverflow.com/questions/36781755/how-to-specify-minbucket-in-caret-train-for>

Overview of Deep Learning for Multiclass in R  
<https://www.thearmchaircritic.org/tech-journal/build-a-multi-class-classification-neural-network-in-r-in-fifty-lines-of-code>

**neuralnet** Help Page  
<https://www.rdocumentation.org/packages/neuralnet/versions/1.44.2/topics/neuralnet>