Online Retail Sales Prediction

Identifying and targeting key customers who are inclined to make purchases, while also predicting the timing of their purchases and the potential amount they might spend, is of paramount importance for businesses globally. To address this challenge, I are working with website traffic data sourced from an online retailer, which encompasses a dataset comprising 35 variables and 70,071 observations or rows.

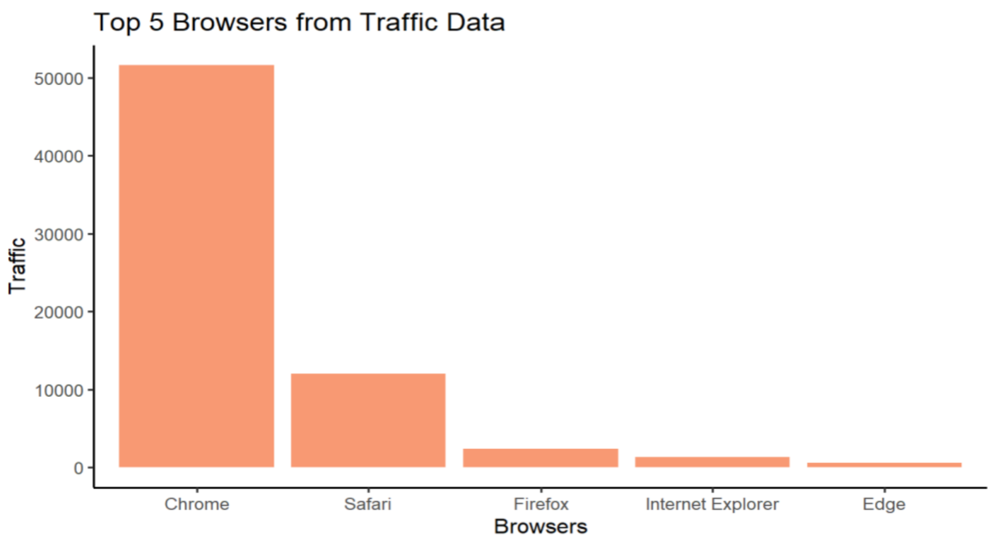
My approach begins with data preparation and modeling. This initial phase sets the foundation for assessing and selecting the most effective model to predict customer purchasing behavior.

The following libraries have been employed in this project:

```r
#LIBRARIES
library(mice)
library(tidyverse)
library(caret)
library(AppliedPredictiveModeling)
library(MASS)
library(dplyr)
library(car)
library(corrr)
library(car)
library(Metrics)
library(ggplot2)
library(earth)
library(rgl)
library(mlbench)
library(lubridate)
library(qpcR)
library(glmnet)
library(e1071)
```

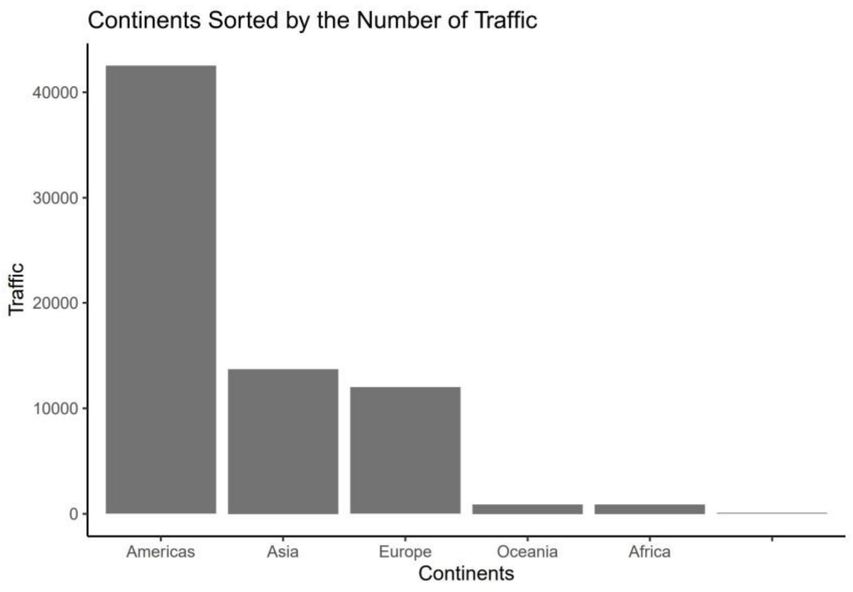
i. Data Visualization

```r
# Traffic data dependency on ‘browser’ column
ggplot(data=Train %>%
         count(browser, sort=TRUE) %>%
         mutate(tier=row_number()) %>%
         filter(tier<6),
       aes(factor(browser, level=browser), n)) +
  geom_col(fill="#f68050", alpha=0.8) +
  labs(title="Top 5 Browsers from Traffic Data", x="Browsers", y="Traffic") +
  theme_classic()
```



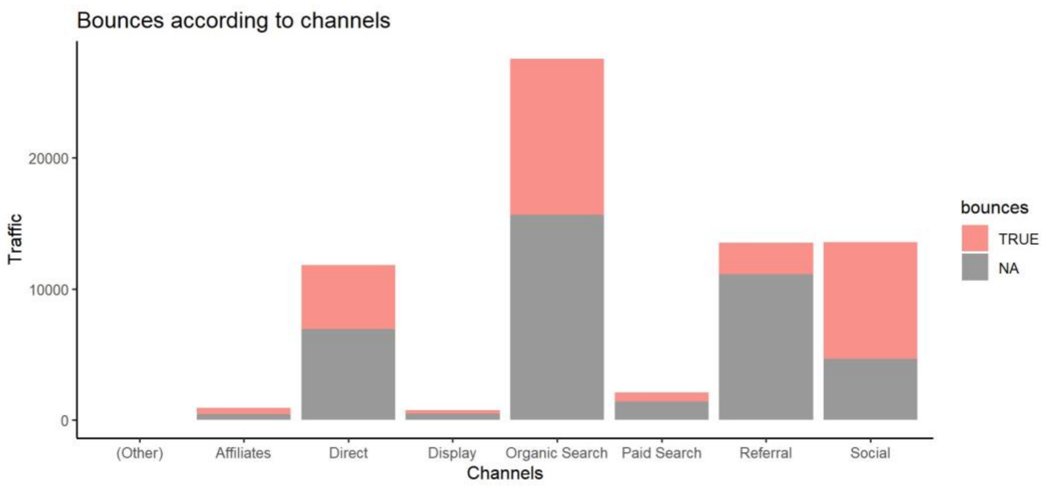
By examining the plot, it becomes evident that the Chrome browser commands the highest volume of web traffic among all browsers, with Safari closely following as the second most popular choice for internet browsing.

```r
#Traffic data dependency on Continent variable
ggplot(data=Train %>% count(continent, sort=TRUE),
       aes(factor(continent,level=continent), n)) +
  geom_col(fill="#515151", alpha=0.8) +
  labs(title="Continents Sorted by the Number of Traffic",
       x="Continents", y="Traffic") +
  theme_classic()
```

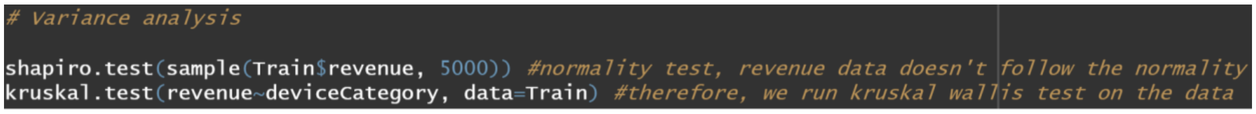


In a similar vein, the data reveals that a significant majority of the web traffic originates from the United States, surpassing the volume from Asia by nearly threefold, making the US the primary region for generating traffic, with Asia emerging as the second-highest contributor.

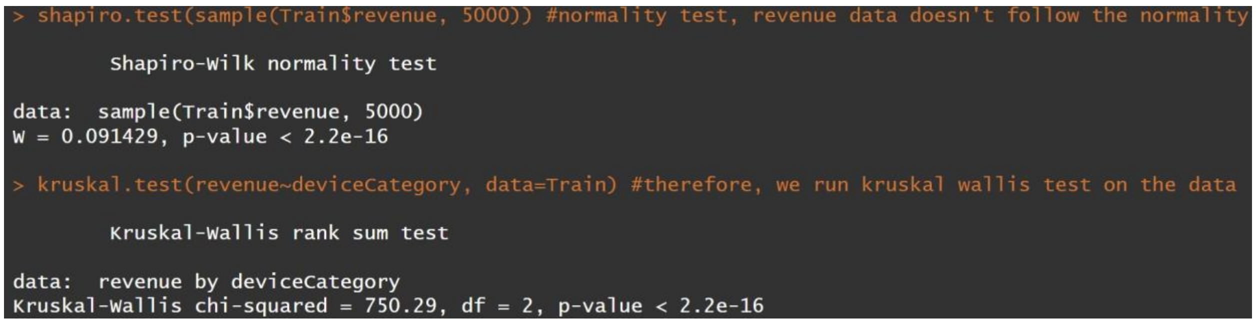
```r
# Traffic data dependency on Bounces
Train$bounces <- as.logical(Train$bounces)

ggplot(data=Train) +
  geom_bar(aes(channelGrouping, fill=bounces), alpha=0.8) +
  labs(title="Bounces according to channels", x="Channels", y="Traffic") +
  theme_classic()
```



I conducted a comprehensive visualization of bounce rates across various customer acquisition channels, encompassing Direct, Social, Organic, Paid Search, and more. A "bounce" denotes when a visitor exits the website without delving deeper after landing on the initial page. This analysis has been pivotal in shedding light on which channels experienced the most bouncing, providing valuable insights for future decision-making. In the provided Figure, "TRUE" signifies the count of bounced traffic, while "NA" designates non-bounced traffic. Notably, Organic search emerged as the channel with the highest total traffic, whereas "Other" recorded the least traffic. However, when examining bounce rates, the "Referral" channel exhibited the lowest percentage of bounced traffic, while the "Social" channels displayed the highest bounce rate compared to non-bounced ("NA") traffic.



```r
# Variance analysis

shapiro.test(sample(Train$revenue, 5000)) #normality test, revenue data doesn't follow the normality
kruskal.test(revenue~deviceCategory, data=Train) #therefore, we run kruskal wallis test on the data
```



```
> shapiro.test(sample(Train$revenue, 5000)) #normality test, revenue data doesn't follow the normality

        Shapiro-Wilk normality test

data:  sample(Train$revenue, 5000)
W = 0.091429, p-value < 2.2e-16

> kruskal.test(revenue~deviceCategory, data=Train) #therefore, we run kruskal wallis test on the data

        Kruskal-Wallis rank sum test

data:  revenue by deviceCategory
Kruskal-Wallis chi-squared = 750.29, df = 2, p-value < 2.2e-16
```

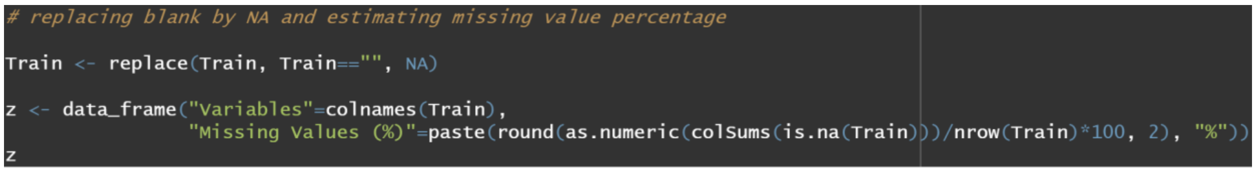
The Shapiro-Wilk normality test yielded a p-value of less than 0.05, indicating that the revenue distribution does not follow a normal pattern. As a result, I proceeded to determine whether different device categories represent identical populations. To assess this, I employed the Kruskal-Wallis rank sum test, which is suitable for non-normally distributed data and is a non-parametric method. The obtained p-value was significantly below 0.05, leading us to conclude that the revenue from different device categories represents non-identical samples.

ii. Data Preparation

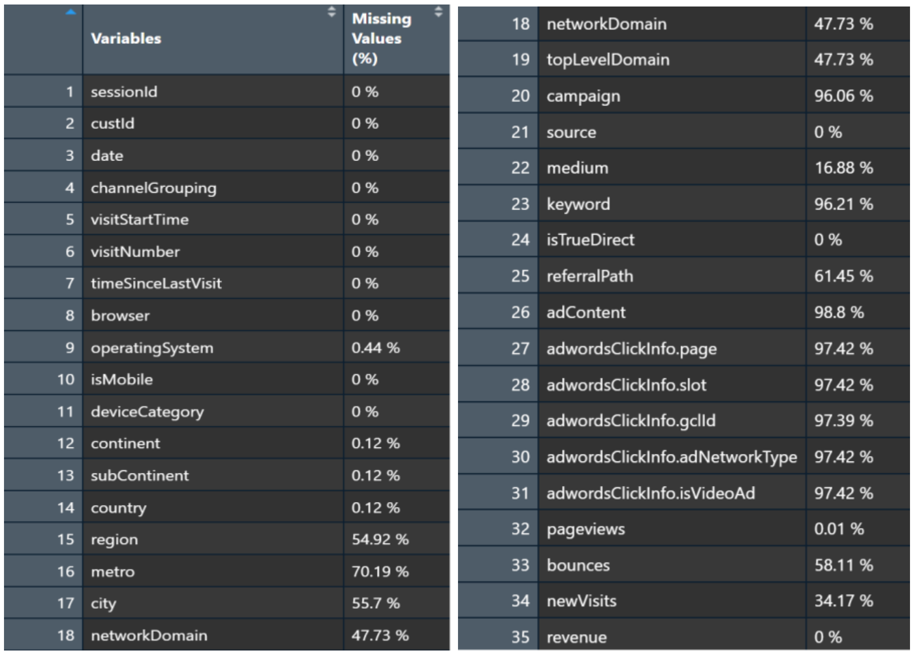
Missing value imputations

Dealing with missing values presents several options, including deletion, indicator variables, and imputation. In our train dataset, I encountered a significant number of missing values, often denoted as 'NA,' within variables such as region, metro, city, networkDomain, among others. Notably, some variables in our dataset contained as much as 98% missing data, rendering them practically unusable. Consequently, I decided to remove these particular variables from our train dataset.

However, it's worth noting that certain missing values hold valuable insights. Take, for instance, variables like 'bounces' and 'newVisits,' which function more as binary indicators rather than traditional integer values, as described on Kaggle. For 'bounces,' a value of 1 signifies a bounced session, while a null value indicates otherwise (when the visitor leaves the website from the landing page without further browsing). Similarly, for 'newVisits,' a value of 1 denotes the first visit by the customer, with null values representing subsequent visits. As such, I chose to impute these missing values using the following approach:



```
# replacing blank by NA and estimating missing value percentage

Train <- replace(Train, Train=="", NA)

z <- data_frame("Variables"=colnames(Train),
                "Missing Values (%)"=paste(round(as.numeric(colSums(is.na(Train)))/nrow(Train)*100, 2), "%"))
z
```

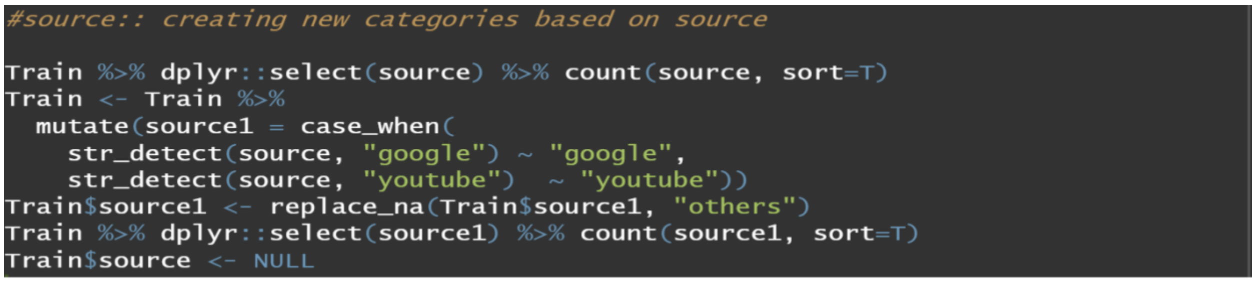


	Variables	Missing Values (%)
1	sessionId	0 %
2	custId	0 %
3	date	0 %
4	channelGrouping	0 %
5	visitStartTime	0 %
6	visitNumber	0 %
7	timeSinceLastVisit	0 %
8	browser	0 %
9	operatingSystem	0.44 %
10	isMobile	0 %
11	deviceCategory	0 %
12	continent	0.12 %
13	subContinent	0.12 %
14	country	0.12 %
15	region	54.92 %
16	metro	70.19 %
17	city	55.7 %
18	networkDomain	47.73 %
19	topLevelDomain	47.73 %
20	campaign	96.06 %
21	source	0 %
22	medium	16.88 %
23	keyword	96.21 %
24	isTrueDirect	0 %
25	referralPath	61.45 %
26	adContent	98.8 %
27	adwordsClickInfo.page	97.42 %
28	adwordsClickInfo.slot	97.42 %
29	adwordsClickInfo.gclId	97.39 %
30	adwordsClickInfo.adNetworkType	97.42 %
31	adwordsClickInfo.isVideoAd	97.42 %
32	pageviews	0.01 %
33	bounces	58.11 %
34	newVisits	34.17 %
35	revenue	0 %

Next, I streamline the "Train" dataset by removing columns that are considered non-essential or contain a high percentage of missing values. This action focuses the dataset on key variables for analysis while simplifying its structure. Columns that have been removed include:

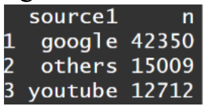
- sessionId
- region
- metro
- city
- networkDomain
- campaign
- keyword
- adContent
- adwordsClickInfo.page
- adwordsClickInfo.slot
- adwordsClickInfo.gclId
- adwordsClickInfo.adNetworkType
- adwordsClickInfo.isVideoAd
- referralPath
- topLevelDomain

Subsequently, I delve into data processing, addressing missing values and transforming variables. I convert "bounces" and "newVisits" columns, handling missing values by assigning "FALSE" and 0, respectively. Additionally, I standardize the representation of time by extracting the hour from the "visitStartTime" column and convert the "date" column into months. Lastly, I adjust the scale of the "timeSinceLastVisit" variable, changing it from seconds to weeks, which aids in making more meaningful interpretations of the data. This comprehensive data preparation process optimizes the "Train" dataset, making it well-prepared for subsequent analysis and modeling activities.

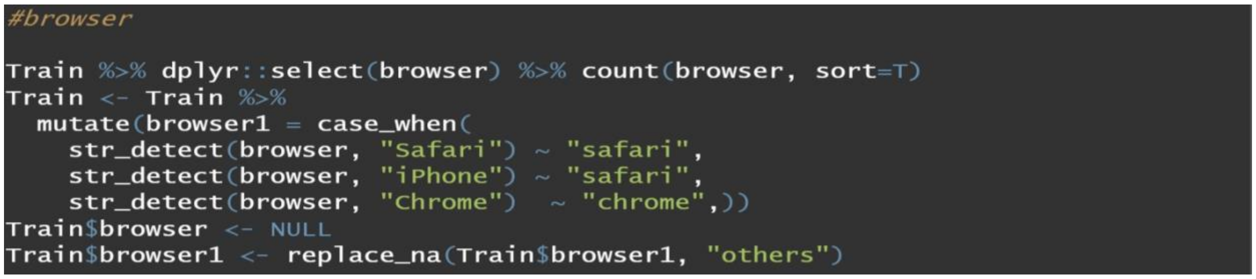


```
#source:: creating new categories based on source

Train %>% dplyr::select(source) %>% count(source, sort=T)
Train <- Train %>%
  mutate(source1 = case_when(
    str_detect(source, "google") ~ "google",
    str_detect(source, "youtube")  ~ "youtube"))
Train$source1 <- replace_na(Train$source1, "others")
Train %>% dplyr::select(source1) %>% count(source1, sort=T)
Train$source <- NULL
```

Upon categorization, it was evident that Google had the highest count, with 42,350 observations, closely followed by YouTube.com with 12,712 observations. To streamline our analysis, I made the strategic decision to condense the number of categories from 131 down to just three, which I have labeled as "Google," "YouTube," and "Others." Following this reclassification, the category counts are as follows:

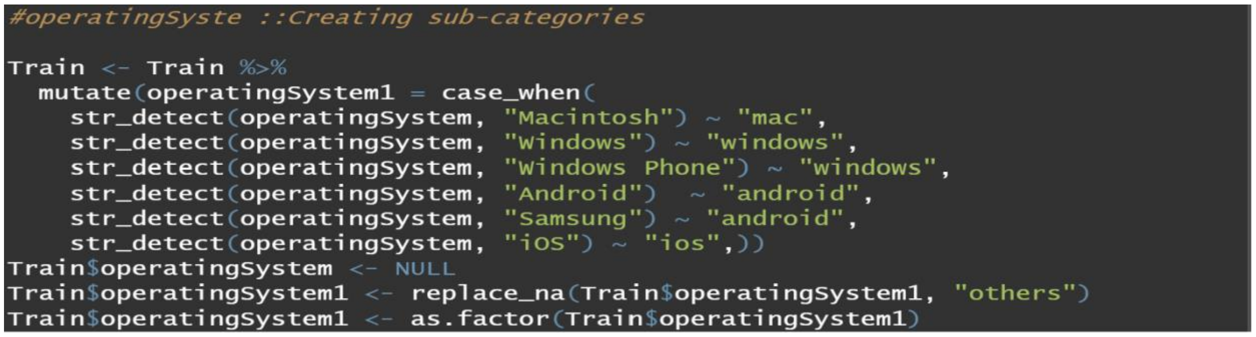


```
  source1     n
1  google 42350
2  others 15009
3 youtube 12712
```



```r
#browser

Train %>% dplyr::select(browser) %>% count(browser, sort=T)
Train <- Train %>%
  mutate(browser1 = case_when(
    str_detect(browser, "Safari") ~ "safari",
    str_detect(browser, "iPhone") ~ "safari",
    str_detect(browser, "Chrome")  ~ "chrome",))
Train$browser <- NULL
Train$browser1 <- replace_na(Train$browser1, "others")
```

The browser variables initially featured 28 distinct categories. Upon examining the distribution of observations, it became evident that Chrome and Safari stood out with the highest counts. To streamline our analysis, I opted to condense the number of categories from 28 to just three. The newly defined categories consist of 'Chrome,' 'Safari,' and 'Others,' where the 'Others' category encompasses all browsers except for Chrome and Safari.



```r
#operatingSyste ::Creating sub-categories

Train <- Train %>%
  mutate(operatingSystem1 = case_when(
    str_detect(operatingSystem, "Macintosh") ~ "mac",
    str_detect(operatingSystem, "Windows") ~ "windows",
    str_detect(operatingSystem, "Windows Phone") ~ "windows",
    str_detect(operatingSystem, "Android")  ~ "android",
    str_detect(operatingSystem, "Samsung") ~ "android",
    str_detect(operatingSystem, "iOS") ~ "ios",))
Train$operatingSystem <- NULL
Train$operatingSystem1 <- replace_na(Train$operatingSystem1, "others")
Train$operatingSystem1 <- as.factor(Train$operatingSystem1)
```

The 'operatingSystem' variable initially included 16 distinct categories. In order to simplify our analysis, I chose to reduce the number of categories from 16 to 5. Among these, the four most frequently occurring operating systems are 'mac,' 'windows,' 'android,' and 'iOS.' The following variables: channelGrouping, browser1, operatingSystem, deviceCategory, continent, source1, and medium were converted into factors.

Some further data processing:

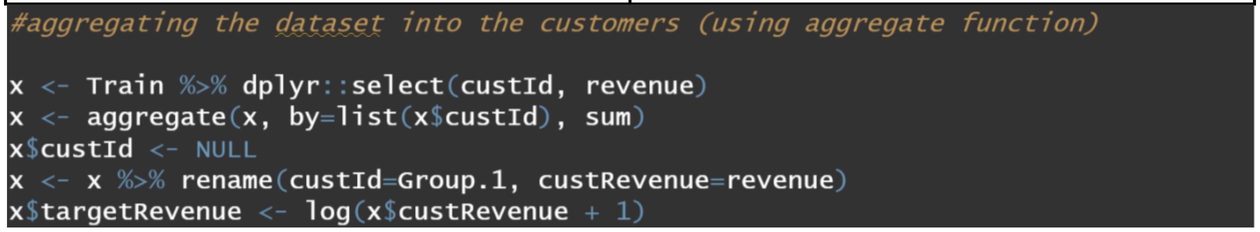
Variable	**Action**
pageviews	Calculate median and replace missing values
country, subContinent, continent, medium	Remove columns due to NA values
channelGrouping	Convert to a numeric format
browser1, deviceCategory, source1, operatingSystem1	Remove duplicate entries
logRevenue	Create a new column representing the logarithm of revenue (logRevenue)

revenue	Remove this column from the dataset
logRevenue (Again)	Replace infinite values with 0

Aggregations

The dataset provided contains customer information at the session level, distinguished by the unique session IDs (given by sessionId). In essence, each customer, identified by custId, can have multiple sessions with the company. To gain insights at the customer level, I made the choice to aggregate the dataset based on individual customers. This aggregation process involved summarizing and consolidating several variables, including 'visitNumber,' 'browser,' 'operatingSystem,' 'continent,' 'source,' 'IsTrueDirect,' 'bounces,' and 'newVisits,' to obtain a comprehensive view of the total revenue earned from each customer.

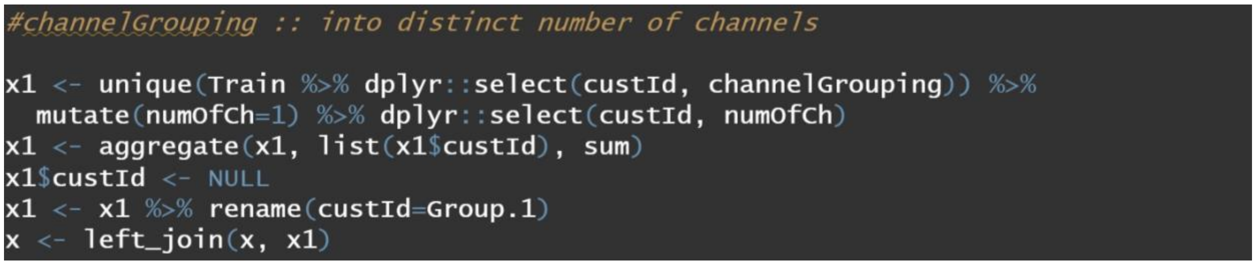
Numeric/Logical variable	**Process**
visitNumber	Max visitNumber for each customer
TrueDirect	Total number of direct for each customer
newVisits	Average of the newVisits
Character variable	**Process**
continent	First occurrence of that customer
Operating system	
browser	



```
#aggregating the dataset into the customers (using aggregate function)

x <- Train %>% dplyr::select(custId, revenue)
x <- aggregate(x, by=list(x$custId), sum)
x$custId <- NULL
x <- x %>% rename(custId=Group.1, custRevenue=revenue)
x$targetRevenue <- log(x$custRevenue + 1)
```

Processing: total revenue for each customer based on different sessionId

Given that each customer ID can have multiple rows in the original dataset, the aggregation process has transformed the data into a new dataset where each distinct customer ID is represented by a single row. Consequently, the total number of rows has decreased significantly, from the initial 71,000 to approximately 42,000 in the aggregated dataset.



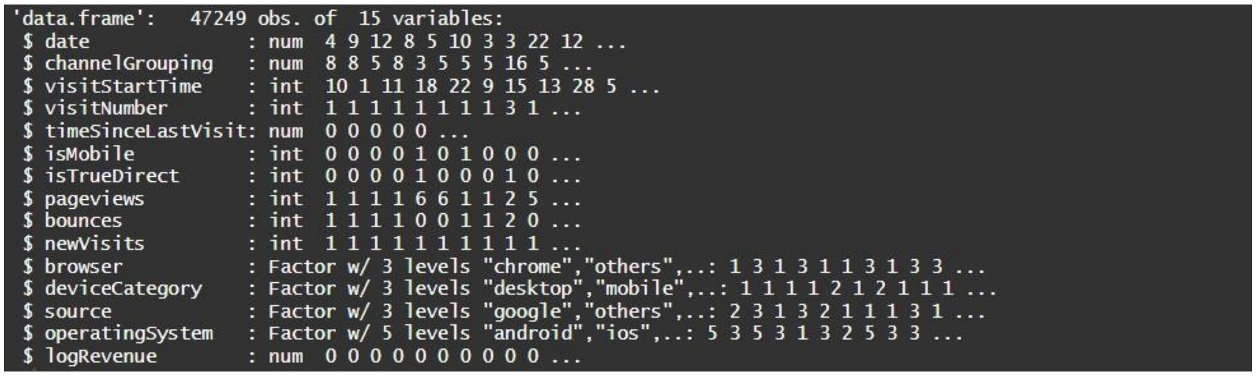
```r
#channelGrouping :: into distinct number of channels

x1 <- unique(Train %>% dplyr::select(custId, channelGrouping)) %>%
  mutate(numOfCh=1) %>% dplyr::select(custId, numOfCh)
x1 <- aggregate(x1, list(x1$custId), sum)
x1$custId <- NULL
x1 <- x1 %>% rename(custId=Group.1)
x <- left_join(x, x1)
```

In this context, a single customer ID can interact with various channels, and the same customer may engage with the same channel multiple times. To gain insights into the diversity of channels used by each customer, a new dataset has been constructed. This dataset provides information on the number of distinct channels that a single customer ID has interacted with, allowing us to understand the range of channels each customer engages with. For aggregation I performed the following data manipulations on both Train and Test.

In the next phase of data preparation it begins by addressing missing values, identifying columns with NA values and calculating the median value for the "pageviews" column to replace NA values. Subsequently, certain columns with missing data are deselected from the dataset, improving its focus and clarity. Data transformation is then carried out, converting "pageviews" into integers and "channelGrouping" into a numeric format, aligning the data for modeling.

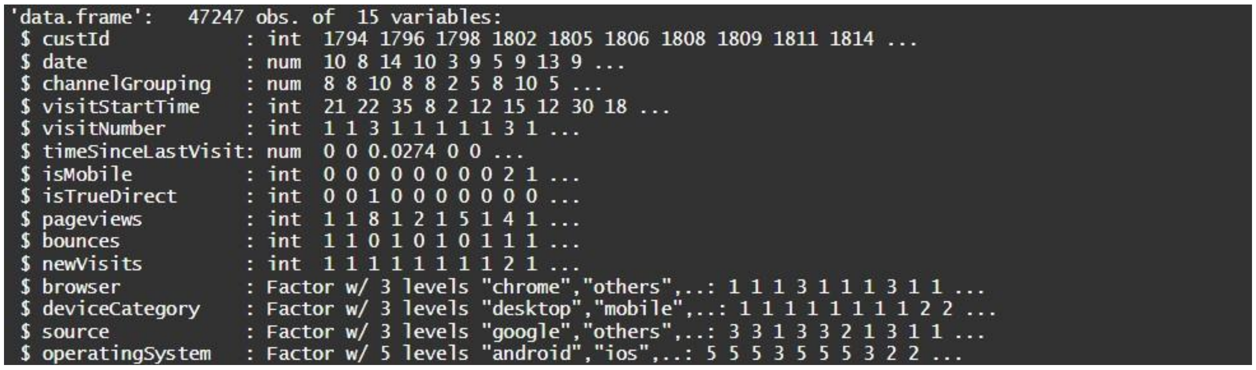
The process continues by handling duplicate data based on customer ID (custId) for various categorical variables like "browser1," "deviceCategory," "source1," and "operatingSystem1." Duplicates are removed, leaving only unique values. These cleaned data sets are then integrated into the primary "Train" dataset. Further, variable names are standardized, and a new variable, "logRevenue," is introduced by taking the logarithm of revenue plus 1, addressing potential issues with Inf values.

To streamline the data for modeling, extraneous columns like "custId" and "Group.1" are deselected, ensuring the dataset is concise and optimized for the subsequent phases of analysis. Lastly, the "revenue" column is deselected, and any Inf values are changed to 0, enhancing the dataset's compatibility with revenue prediction models. This comprehensive data preparation process substantially improves the dataset's quality and structure, setting the stage for effective revenue forecasting. The figures below present the outcomes of the data preparation process and highlight the characteristics of datasets "TrainNew" and "TestNew," which were generated from the provided Train and Test datasets.



```
'data.frame':   47249 obs. of  15 variables:
 $ date              : num  4 9 12 8 5 10 3 3 22 12 ...
 $ channelGrouping   : num  8 8 5 8 3 5 5 5 16 5 ...
 $ visitStartTime    : int  10 1 11 18 22 9 15 13 28 5 ...
 $ visitNumber       : int  1 1 1 1 1 1 1 1 3 1 ...
 $ timeSinceLastVisit: num  0 0 0 0 0 ...
 $ isMobile          : int  0 0 0 0 1 0 1 0 0 0 ...
 $ isTrueDirect      : int  0 0 0 0 1 0 0 0 1 0 ...
 $ pageviews         : int  1 1 1 1 6 6 1 1 2 5 ...
 $ bounces           : int  1 1 1 1 0 0 1 1 2 0 ...
 $ newVisits         : int  1 1 1 1 1 1 1 1 1 1 ...
 $ browser           : Factor w/ 3 levels "chrome","others",..: 1 3 1 3 1 1 3 1 3 3 ...
 $ deviceCategory    : Factor w/ 3 levels "desktop","mobile",..: 1 1 1 1 2 1 2 1 1 1 ...
 $ source            : Factor w/ 3 levels "google","others",..: 2 3 1 3 2 1 1 1 3 1 ...
 $ operatingSystem   : Factor w/ 5 levels "android","ios",..: 5 3 5 3 1 3 2 5 3 3 ...
 $ logRevenue        : num  0 0 0 0 0 0 0 0 0 0 ...
```

Fig 1. Properties of data frame TrainNew, which was derived from the 'Train' dataset.

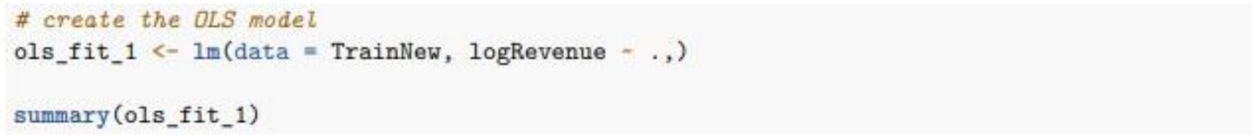


```
'data.frame':   47247 obs. of  15 variables:
 $ custId           : int  1794 1796 1798 1802 1805 1806 1808 1809 1811 1814 ...
 $ date             : num  10 8 14 10 3 9 5 9 13 9 ...
 $ channelGrouping  : num  8 8 10 8 8 2 5 8 10 5 ...
 $ visitStartTime   : int  21 22 35 8 2 12 15 12 30 18 ...
 $ visitNumber      : int  1 1 3 1 1 1 1 1 3 1 ...
 $ timeSinceLastVisit: num  0 0 0.0274 0 0 ...
 $ isMobile         : int  0 0 0 0 0 0 0 0 2 1 ...
 $ isTrueDirect     : int  0 0 1 0 0 0 0 0 0 0 ...
 $ pageviews        : int  1 1 8 1 2 1 5 1 4 1 ...
 $ bounces          : int  1 1 0 1 0 1 0 1 1 1 ...
 $ newVisits        : int  1 1 1 1 1 1 1 1 2 1 ...
 $ browser          : Factor w/ 3 levels "chrome","others",..: 1 1 1 3 1 1 1 3 1 1 ...
 $ deviceCategory   : Factor w/ 3 levels "desktop","mobile",..: 1 1 1 1 1 1 1 1 2 2 ...
 $ source           : Factor w/ 3 levels "google","others",..: 3 3 1 3 3 2 1 3 1 1 ...
 $ operatingSystem  : Factor w/ 5 levels "android","ios",..: 5 5 5 3 5 5 5 3 2 2 ...
```

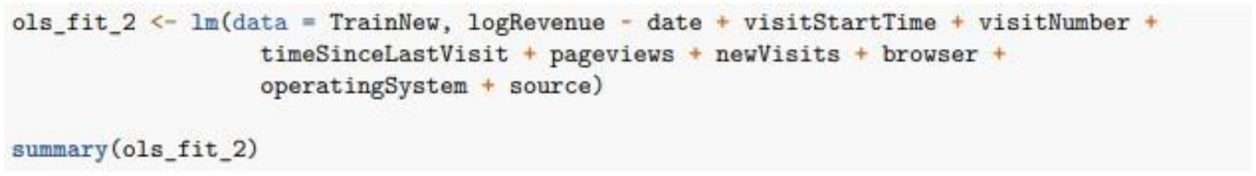
Fig 2. Properties of data frame TestNew, which was derived from the 'Test' dataset.

iii. Modeling

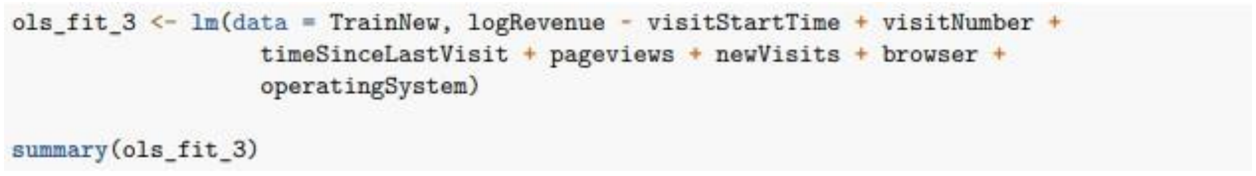
For our modeling, I used various models to try and obtain and I evaluated them based on the RMSE (root-mean-square deviation) value they produced. The lower the RMSE value the better the model. The first model I used was the ordinary least square (OLS) model. At first, I fitted all the Train data with the variables obtained as a function of logRevenue from the data preparation. When looking at the p-value I excluded some of the variables with a higher p-value and ran the same model again to see if I could get better results.



```r
# create the OLS model
ols_fit_1 <- lm(data = TrainNew, logRevenue ~ .,)

summary(ols_fit_1)
```

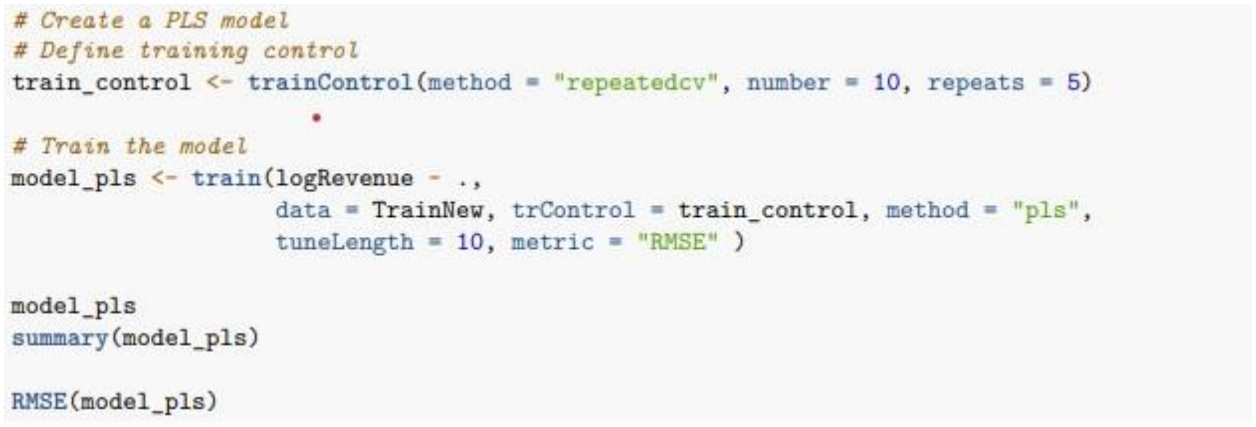


```r
ols_fit_2 <- lm(data = TrainNew, logRevenue ~ date + visitStartTime + visitNumber +
                   timeSinceLastVisit + pageviews + newVisits + browser +
                   operatingSystem + source)

summary(ols_fit_2)
```

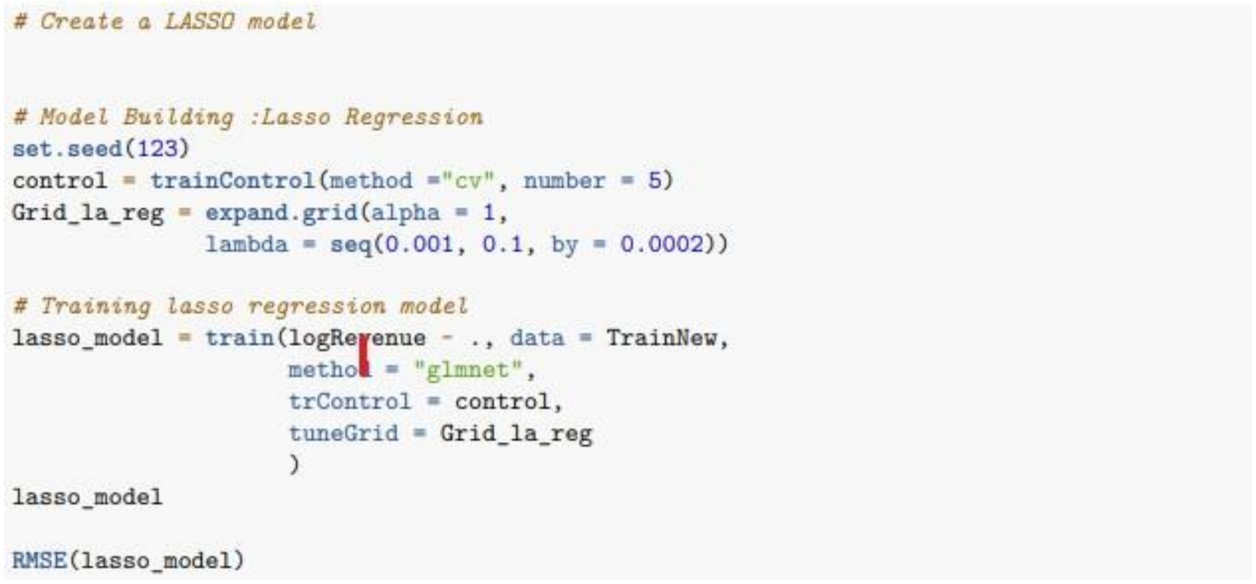


```r
ols_fit_3 <- lm(data = TrainNew, logRevenue ~ visitStartTime + visitNumber +
                   timeSinceLastVisit + pageviews + newVisits + browser +
                   operatingSystem)

summary(ols_fit_3)
```

Our group then moved on the the PLS (partial least square) model and LASSO model. In PLS model, I introduced a train control. For the control, I focused on the repeated cross-validation method. The tunelength was set to 10 and the metric used was RMSE. For the LASSO model, I had the same trainControl and I added a grid that has a range of lambda values. The model was set to make predictions using different lambda values are pick the best one that produced the lowest RMSE score. I used all the variables from the data preparation in modeling.

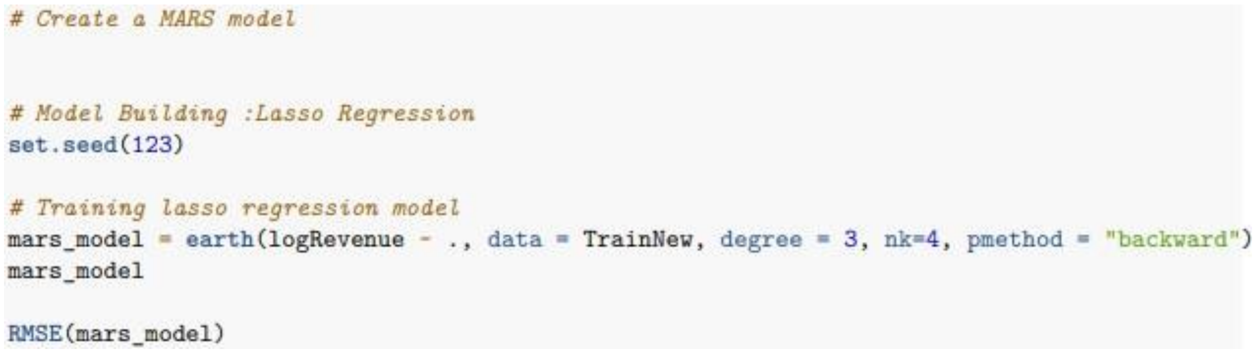


```
# Create a PLS model
# Define training control
train_control <- trainControl(method = "repeatedcv", number = 10, repeats = 5)

# Train the model
model_pls <- train(logRevenue ~ .,
                   data = TrainNew, trControl = train_control, method = "pls",
                   tuneLength = 10, metric = "RMSE" )

model_pls
summary(model_pls)

RMSE(model_pls)
```

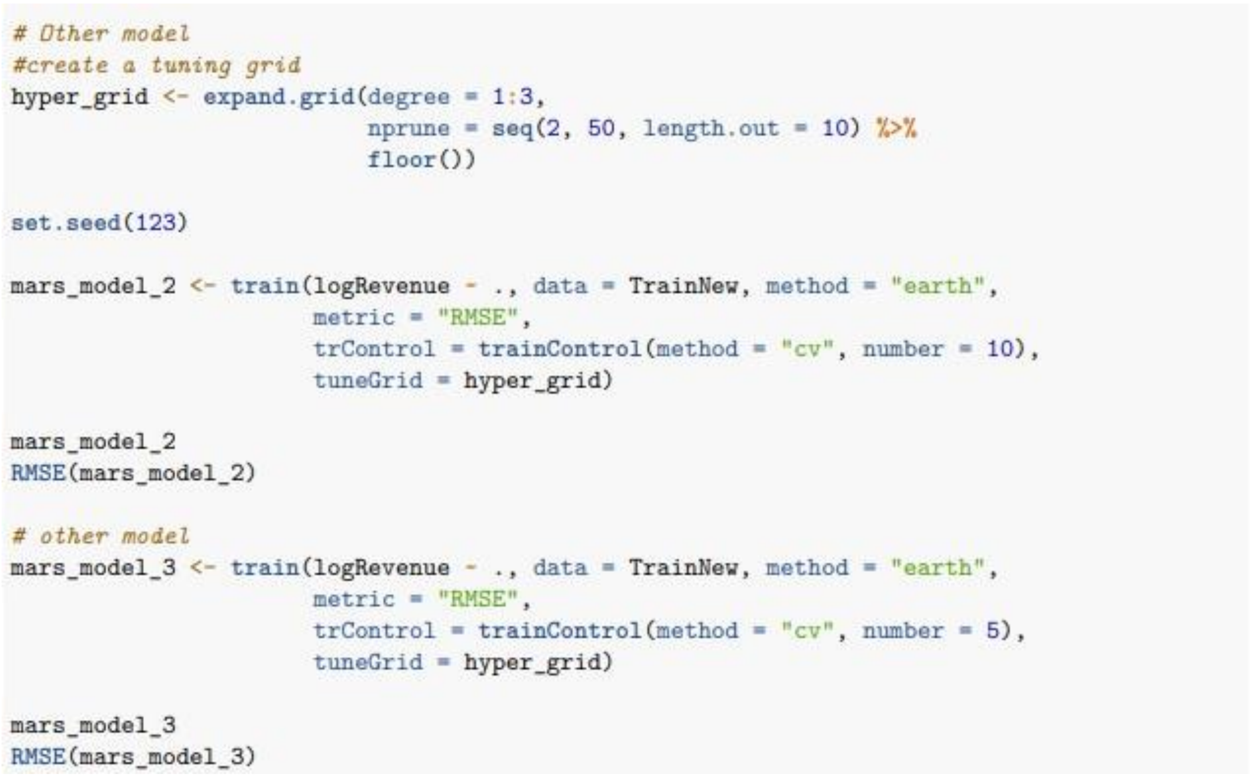


```
# Create a LASSO model


# Model Building :Lasso Regression
set.seed(123)
control = trainControl(method ="cv", number = 5)
Grid_la_reg = expand.grid(alpha = 1,
              lambda = seq(0.001, 0.1, by = 0.0002))

# Training lasso regression model
lasso_model = train(logRevenue ~ ., data = TrainNew,
                    method = "glmnet",
                    trControl = control,
                    tuneGrid = Grid_la_reg
                    )
lasso_model

RMSE(lasso_model)
```

In our modeling, I also used the MARS model. When executing this took the longest time but produced the lowest RMSE. I had a grid of hyperparameters for degree, nprune with a cross-validation method.



```
# Create a MARS model


# Model Building :Lasso Regression
set.seed(123)

# Training lasso regression model
mars_model = earth(logRevenue ~ ., data = TrainNew, degree = 3, nk=4, pmethod = "backward")
mars_model

RMSE(mars_model)
```

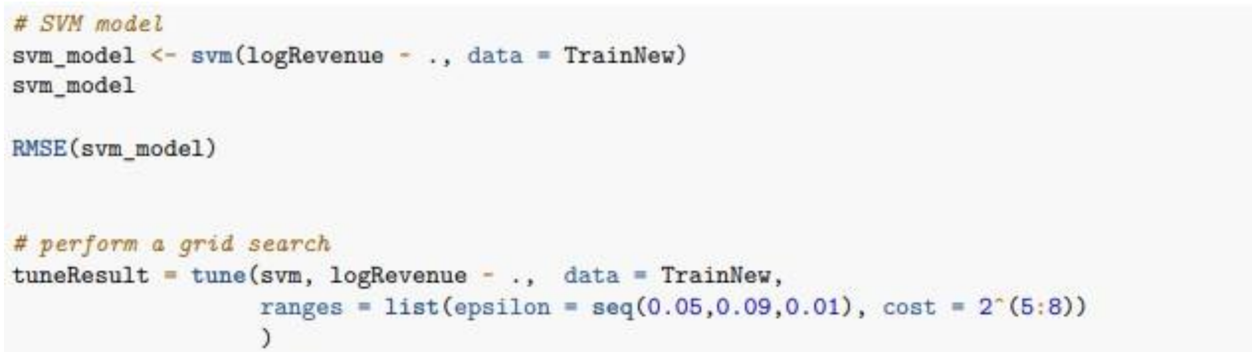


```
# Other model
#create a tuning grid
hyper_grid <- expand.grid(degree = 1:3,
                          nprune = seq(2, 50, length.out = 10) %>%
                          floor())

set.seed(123)

mars_model_2 <- train(logRevenue ~ ., data = TrainNew, method = "earth",
                      metric = "RMSE",
                      trControl = trainControl(method = "cv", number = 10),
                      tuneGrid = hyper_grid)

mars_model_2
RMSE(mars_model_2)

# other model
mars_model_3 <- train(logRevenue ~ ., data = TrainNew, method = "earth",
                      metric = "RMSE",
                      trControl = trainControl(method = "cv", number = 5),
                      tuneGrid = hyper_grid)

mars_model_3
RMSE(mars_model_3)
```

Model	Method	Package	Hyperparameters	Value	RMSE	Rsquared
OLS	lm	stats	N/A	N/A	0.963	0.459
OLS	lm	stats	N/A	N/A	0.968	0.453
PLS	pls	caret	tuneLength, ncomp	10, 10	1.01	0.45
LASSO	glmnet	caret	Lambda, alpha	0.0056, 1	0.96	0.43
MARS	earth	earth	Degree, nk	3, 4	0.78	0.63
MARS	earth	earth	Degree, nprune	3, 28	0.73	0.67
SVM	svm	e1071				

iv. Debrief

The MARS model was our best model because it gave us the lowest RMSE score in both the R script (value of 0.73) prediction and in the final evaluation from the real predicted data in Kaggle submission (value of 0.744). In this model, I started with all our prepared data from the Train that had the variables I wanted to include. Previous models have proven that all the columns selected from data prep I relevant in the model prediction. In the MARS model, I had a grid sequence of hyperparameters for degree, nprune with a cross-validation method. In the sequence I set the model to evaluate with degree ranging from 1:3 and nprune from 2 - 50. This took about 10 minutes to run and after the model was complete it found degree = 3 and nprune = 28 to yield the best predictions. I also changed up the method from "cv" to "backward" trying to see if I yielded before results. Our group also tried using the svm model and it did seem to yield better results before tuning but it took up several hours to hyper-tune the parameters. This was very unfortunate because if I had advanced computers with faster processing speeds I would be able to find the best hyperparameters from a series of parameter lengths that would optimize our model predictions to a more desired RMSE score. I could have also done more data preparation for outliers. It was very difficult to deal with a lot of the variables having missing data and having categorical data that was missing too.



```r
# SVM model
svm_model <- svm(logRevenue ~ ., data = TrainNew)
svm_model

RMSE(svm_model)


# perform a grid search
tuneResult = tune(svm, logRevenue ~ .,  data = TrainNew,
                  ranges = list(epsilon = seq(0.05,0.09,0.01), cost = 2^(5:8))
                  )
```