WSMA - Shark Tank Project

Sanju Hyacinth C

20/12/2019

1. Problem Statement:

We are provided with a dataset of Shark Tank episodes containing 495 entrepreneurs making their pitch to the VC sharks. Initially we are asked to only use the description for **text mining** with deal as the dependent variable and develop prediction models. Later we are asked to include a new column "Ratio" calculated by dividing asked for by valuation.

2. Packages Required:

```r
# install.packages('tm')
# install.packages('SnowballC')(
# install.packages('ggplot2')
# install.packages('RColorBrewer')
# install.packages('wordcloud')
# install.packages('topicmodels')
# install.packages('data.table')
# install.packages('stringi')
# install.packages('dplyr')
# install.packages('syuzhet')
# install.packages('plyr')
# install.packages('grid')
# install.packages('caTools')
#install.packages("rpart")
#install.packages("rpart.plot")
#install.packages("rattle")
#install.packages("ROCR")
# install.packages('randomForest')

library(wordcloud)
```

```
## Warning: package 'wordcloud' was built under R version 3.6.1
## Loading required package: RColorBrewer
```

```r
library(RColorBrewer)
library(topicmodels)
```

```
## Warning: package 'topicmodels' was built under R version 3.6.1
```

```r
library(data.table)
```

```
## Warning: package 'data.table' was built under R version 3.6.2
```

```r
library(syuzhet)
```

```
## Warning: package 'syuzhet' was built under R version 3.6.1
```

```r
library(grid)
library(plyr)
```

```
## Warning: package 'plyr' was built under R version 3.6.1
```

```r
library(caTools)
```

```
## Warning: package 'caTools' was built under R version 3.6.1
```

```r
library(rpart)
```

```
## Warning: package 'rpart' was built under R version 3.6.1
```

```r
library(rpart.plot)
```

```
## Warning: package 'rpart.plot' was built under R version 3.6.1
```

```r
library(ROCR)
```

```
## Warning: package 'ROCR' was built under R version 3.6.1
## Loading required package: gplots
## Warning: package 'gplots' was built under R version 3.6.1
##
## Attaching package: 'gplots'
## The following object is masked from 'package:wordcloud':
##
##     textplot
## The following object is masked from 'package:stats':
##
##     lowess
```

```r
library(rattle)
```

```
## Warning: package 'rattle' was built under R version 3.6.1
## Rattle: A free graphical interface for data science with R.
## Version 5.2.0 Copyright (c) 2006-2018 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.
```

```r
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 3.6.1
## randomForest 4.6-14
## Type rfNews() to see new features/changes/bug fixes.
##
## Attaching package: 'randomForest'
## The following object is masked from 'package:rattle':
##
##     importance
```

3. Data Exploration:

```r
# Data loading:

setwd("D:/R Progms")
```

```r
shark.df = read.csv("Dataset (1).csv")
View(shark.df)

# Converting to correct data types:

shark.df$description = as.character(shark.df$description)
shark.df$deal = ifelse(shark.df$deal=="TRUE",1,0)
shark.df$deal = as.factor(shark.df$deal)
```

Data Exploration:

```r
# Structure Exploration:

str(shark.df)
```

```
## 'data.frame':    495 obs. of  19 variables:
##  $ deal                   : Factor w/ 2 levels "0","1": 1 2 2 1 1 2 1 1 1 2 ...
##  $ description            : chr  "Bluetooth device implant for your ear." "Retail and wholesale pie fa
##  $ episode                : int  1 1 1 1 1 2 2 2 2 2 ...
##  $ category               : Factor w/ 54 levels "Alcoholic Beverages",..: 36 45 3 8 8 45 31 43 2 11 .
##  $ entrepreneurs          : Factor w/ 422 levels "","Aaron Lemieux",..: 96 406 402 310 223 388 84 264
##  $ location               : Factor w/ 255 levels "Akron, OH","Alexandria, VA",..: 226 220 8 230 36 154
##  $ website                : Factor w/ 456 levels "","http://180cup.com",..: 1 119 130 21 405 128 25 1
##  $ askedFor               : int  1000000 460000 50000 250000 1200000 500000 200000 100000 500000 2500
##  $ exchangeForStake       : int  15 10 15 25 10 15 20 20 10 10 ...
##  $ valuation              : int  6666667 4600000 333333 1000000 12000000 3333333 1000000 500000 50000
##  $ season                 : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ shark1                 : Factor w/ 2 levels "Barbara Corcoran",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ shark2                 : Factor w/ 4 levels "Barbara Corcoran",..: 3 3 3 3 3 3 3 3 3 3 ...
##  $ shark3                 : Factor w/ 3 levels "Daymond John",..: 2 2 2 2 2 2 2 2 2 2 ...
##  $ shark4                 : Factor w/ 4 levels "Daymond John",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ shark5                 : Factor w/ 5 levels "Daymond John",..: 3 3 3 3 3 3 3 3 3 3 ...
##  $ title                  : Factor w/ 493 levels "180 Cup","50 State Capitals in 50 Minutes",..: 205 
##  $ episode.season         : Factor w/ 122 levels "1-1","1-10","1-11",..: 1 1 1 1 1 7 7 7 7 7 ...
##  $ Multiple.Entreprenuers: logi  FALSE FALSE FALSE FALSE FALSE FALSE ...
```

```r
# Summary of data:

summary(shark.df)
```

```
##  deal    description           episode
##  0:244   Length:495         Min.   : 1.00
##  1:251   Class :character   1st Qu.: 5.00
##          Mode  :character   Median :11.00
##                             Mean   :12.13
##                             3rd Qu.:18.00
##                             Max.   :29.00
##
##                          category          entrepreneurs
##  Specialty Food              : 62                 : 72
##  Novelties                   : 35   Dave Alwan    :  2
##  Baby and Child Care         : 24   James Martin  :  2
##  Online Services             : 22   Aaron Lemieux :  1
##  Personal Care and Cosmetics: 20   Aaron Marino  :  1
```

```
##  Toys and Games             : 19   Aaron McDaniel:  1
##  (Other)                    :313   (Other)       :416
##              location                              website
##  Los Angeles, CA  : 41                                 : 38
##  New York, NY     : 30   http://www.copadivino.com/    :  2
##  San Francisco, CA: 25   http://www.echovalleymeats.com:  2
##  Chicago, IL      : 14   http://180cup.com             :  1
##  Austin, TX       : 13   http://aircork.com/           :  1
##  Atlanta, GA      : 11   http://amangoparty.com        :  1
##  (Other)          :361   (Other)                       :450
##     askedFor       exchangeForStake   valuation            season
##  Min.   :  10000   Min.   :  3.00   Min.   :   40000   Min.   :1.000
##  1st Qu.:  75000   1st Qu.: 10.00   1st Qu.:  440000   1st Qu.:3.000
##  Median : 150000   Median : 15.00   Median : 1000000   Median :4.000
##  Mean   : 258491   Mean   : 17.54   Mean   : 2165615   Mean   :4.048
##  3rd Qu.: 250000   3rd Qu.: 20.00   3rd Qu.: 2000000   3rd Qu.:5.000
##  Max.   :5000000   Max.   :100.00   Max.   :30000000   Max.   :6.000
##
##              shark1                 shark2                shark3
##  Barbara Corcoran:220   Barbara Corcoran:104   Daymond John    : 12
##  Lori Greiner    :275   Kevin O'Leary   : 12   Kevin O'Leary   :379
##                         Robert Herjavec :375   Robert Herjavec:104
##                         Steve Tisch     :  4
##
##
##
##             shark4                  shark5
##  Daymond John   :371   Daymond John     :  8
##  Jeff Foxworthy:  8   John Paul DeJoria:  4
##  Kevin O'Leary :104   Kevin Harrington : 80
##  Mark Cuban    : 12   Mark Cuban       :395
##                        Nick Woodman     :  8
##
##
##                              title     episode.season
##  Copa di Vino                  :  2   1-1     :  5
##  Echo Valley Meats             :  2   1-11    :  5
##  180 Cup                       :  1   1-14    :  5
##  50 State Capitals in 50 Minutes:  1   1-2     :  5
##  A Perfect Pear                :  1   1-3     :  5
##  Addison's Wonderland          :  1   1-4     :  5
##  (Other)                       :487   (Other):465
##  Multiple.Entreprenuers
##  Mode :logical
##  FALSE:334
##  TRUE :161
##
##
##
##
```

```
## Out of the 495 deals, 251 were accepted by the VC Sharks
## Speciality food is the category that is most pitched
## A large number of entrepreneurs from Los Angeles, CA
```

```
## The minimum and maximum asked for value are 10000 and 5000000
## The minimum and maximum valuation are 40000 and 30000000
```

4. Data cleaning - Refining text

4.1 Step 1

4.1.1 Corpus Creation:

```r
# Corpus creation

library(tm)
```

```
## Warning: package 'tm' was built under R version 3.6.1
## Loading required package: NLP
```

```r
library(SnowballC)

sCorpus = Corpus(VectorSource(shark.df$description))
```

Converting the corpus to lowercase:

```r
library(stringi)
sCorpus = tm_map(sCorpus, content_transformer(stri_trans_tolower))
```

```
## Warning in tm_map.SimpleCorpus(sCorpus,
## content_transformer(stri_trans_tolower)): transformation drops documents
```

```r
## Result
writeLines(strwrap(sCorpus[[22]]$content, 100))
```

```
## granola gourmet offers a line of granola bars that diabetics can safely enjoy. unlike most granola
## bars on the market, granola gourmet's bars have a low glycemic index. having a low glycemic index
## means that they are less prone to causing spikes in blood sugar, which aren't good for anybody and
## especially damaging for diabetics. granola gourmet's bars are made with ingredients that naturally
## have a low glycemic index, so they release their carbohydrates slowly into the bloodstream. granola
## gourmet products have been tested by gi labs, which developed the glycemic index concept. they
## ultimate fudge brownie bar has a glycemic index of just 23, well below the threshold to be
## considered low glycemic.
```

Removing Punctuation in the text:

```r
## Removing punctuation
sCorpus = tm_map(sCorpus, removePunctuation)
```

```
## Warning in tm_map.SimpleCorpus(sCorpus, removePunctuation): transformation
## drops documents
```

```r
## Result
writeLines(strwrap(sCorpus[[22]]$content, 100))
```

```
## granola gourmet offers a line of granola bars that diabetics can safely enjoy unlike most granola
## bars on the market granola gourmets bars have a low glycemic index having a low glycemic index
## means that they are less prone to causing spikes in blood sugar which arent good for anybody and
## especially damaging for diabetics granola gourmets bars are made with ingredients that naturally
```

```
## have a low glycemic index so they release their carbohydrates slowly into the bloodstream granola
## gourmet products have been tested by gi labs which developed the glycemic index concept they
## ultimate fudge brownie bar has a glycemic index of just 23 well below the threshold to be
## considered low glycemic
```

Removing extra white spaces:

```r
## White space removal
sCorpus = tm_map(sCorpus, stripWhitespace)
```

```
## Warning in tm_map.SimpleCorpus(sCorpus, stripWhitespace): transformation
## drops documents
```

```r
## Result
writeLines(strwrap(sCorpus[[22]]$content, 100))
```

```
## granola gourmet offers a line of granola bars that diabetics can safely enjoy unlike most granola
## bars on the market granola gourmets bars have a low glycemic index having a low glycemic index
## means that they are less prone to causing spikes in blood sugar which arent good for anybody and
## especially damaging for diabetics granola gourmets bars are made with ingredients that naturally
## have a low glycemic index so they release their carbohydrates slowly into the bloodstream granola
## gourmet products have been tested by gi labs which developed the glycemic index concept they
## ultimate fudge brownie bar has a glycemic index of just 23 well below the threshold to be
## considered low glycemic
```

Remove Stopwords:

```r
## Adding more stop words
moreStopwords = c((stopwords("english")), c("shark", "tank", "also", "can", "just", "use", "products",

## Removing stopwords
sCorpus = tm_map(sCorpus, removeWords, moreStopwords)
```

```
## Warning in tm_map.SimpleCorpus(sCorpus, removeWords, moreStopwords):
## transformation drops documents
```

```r
## Result
writeLines(strwrap(sCorpus[[22]]$content, 100))
```

```
## granola gourmet offers line granola bars diabetics safely enjoy unlike granola bars market granola
## gourmets bars low glycemic index low glycemic index means less prone causing spikes blood sugar
## arent good anybody especially damaging diabetics granola gourmets bars ingredients naturally low
## glycemic index release carbohydrates slowly bloodstream granola gourmet tested gi labs developed
## glycemic index concept ultimate fudge brownie bar glycemic index 23 well threshold considered low
## glycemic
```

Remove Numbers:

```r
## Remove numbers
sCorpus = tm_map(sCorpus, removeNumbers)
```

```
## Warning in tm_map.SimpleCorpus(sCorpus, removeNumbers): transformation
## drops documents
```

```r
## Result
writeLines(strwrap(sCorpus[[22]]$content, 100))
```

```
## granola gourmet offers line granola bars diabetics safely enjoy unlike granola bars market granola
## gourmets bars low glycemic index low glycemic index means less prone causing spikes blood sugar
## arent good anybody especially damaging diabetics granola gourmets bars ingredients naturally low
## glycemic index release carbohydrates slowly bloodstream granola gourmet tested gi labs developed
## glycemic index concept ultimate fudge brownie bar glycemic index well threshold considered low
## glycemic
```

4.1.2 Creating a Document Text Matrix:

```r
## Document Text Matrix
dtm = DocumentTermMatrix(sCorpus)
dtm
```

```
## <<DocumentTermMatrix (documents: 495, terms: 4616)>>
## Non-/sparse entries: 9436/2275484
## Sparsity           : 100%
## Maximal term length: 24
## Weighting          : term frequency (tf)
```

```r
## Removing the least occuring terms (sparse terms) from the text
## Cleaning the data upto 99.7 %
dtm = removeSparseTerms(dtm, 0.997)
dtm
```

```
## <<DocumentTermMatrix (documents: 495, terms: 1562)>>
## Non-/sparse entries: 6382/766808
## Sparsity           : 99%
## Maximal term length: 23
## Weighting          : term frequency (tf)
```

```r
## Converting to a data frame
dataShark = as.data.frame(as.matrix(dtm))

## Include the dependent column to the new data frame
dataShark$deal = shark.df$deal

## dtm contains documents: 495, terms: 1563
```

Let us find some of the most frequent terms in the text data:

```r
## Minimum frequency of 10 times

termfreq1 = findFreqTerms(dtm, lowfreq = 10)
termfreq1
```

```
##   [1] "device"      "new"         "retail"      "two"         "children"
##   [6] "easy"        "women"       "designed"    "first"       "flavors"
##  [11] "food"        "line"        "many"        "one"         "product"
##  [16] "sold"        "apparel"     "clothing"    "get"         "help"
##  [21] "fit"         "cards"       "fun"         "keep"        "kids"
##  [26] "company"     "childrens"   "designs"     "easier"      "like"
##  [31] "look"        "make"        "offers"      "play"        "protection"
##  [36] "solution"    "time"        "yet"         "coffee"      "back"
##  [41] "sells"       "accessories" "online"      "service"     "users"
##  [46] "bars"        "enjoy"       "ingredients" "market"      "safely"
```

```
##  [51] "well"         "body"         "customers"    "natural"      "bottle"
##  [56] "around"       "featuring"    "live"         "instead"      "three"
##  [61] "fashion"      "unique"       "allows"       "buy"          "sizes"
##  [66] "organic"      "need"         "full"         "allnatural"   "using"
##  [71] "place"        "cleaning"     "easily"       "store"        "way"
##  [76] "business"     "ice"          "mobile"       "making"       "youre"
##  [81] "created"      "design"       "toy"          "usa"          "will"
##  [86] "user"         "want"         "including"    "money"        "plastic"
##  [91] "power"        "without"      "premium"      "wine"         "air"
##  [96] "cover"        "keeps"        "comes"        "provides"     "available"
## [101] "anyone"       "small"        "skin"         "builtin"      "system"
## [106] "training"     "come"         "fire"         "home"         "tools"
## [111] "box"          "patented"     "water"        "people"       "music"
## [116] "real"         "used"         "butter"       "free"         "safe"
## [121] "now"          "better"       "best"         "clothes"      "helps"
## [126] "balm"         "every"        "flavor"       "cup"          "app"
## [131] "take"         "phone"        "dont"         "dog"          "hair"
## [136] "baby"         "simple"       "great"        "fresh"
```

```
## There are about 138 terms that appear at least 10 times in the text data
## These words include plurals and some compound words that lack punctuation marks.
```

```
## Minimum frequency of 25 times

termfreq2 = findFreqTerms(dtm, lowfreq = 25)
termfreq2
```

```
##  [1] "designed" "line"     "one"      "product"  "fun"      "kids"
##  [7] "company"  "like"     "make"     "online"   "way"      "without"
## [13] "system"   "water"
```

```
## There are about 14 terms that appear at least 25 times in the text data
## This tells that only about 10% of the earlier words have made it upto 25 times
```

```
## Reducing the minimum frequency to 30

termfreq4 = findFreqTerms(dtm, lowfreq = 30)
termfreq4
```

```
## [1] "designed" "product"  "company"  "like"     "make"     "online"
## [7] "without"  "system"   "water"
```

```
## We have a few words that actually convey some meaning or an excerpt of the text
## We get a general idea about the pitches relating to establishing a company either for product sales
```

A Wordcloud visualisation is given below:

```
#Creating Wordcloud

palette = brewer.pal(8, "Dark2")
wordcloud(sCorpus, min.freq = 3, max.words = 100, random.order = FALSE, color = palette, rot.per =0.2)
```

```
## Warning in wordcloud(sCorpus, min.freq = 3, max.words = 100, random.order =
## FALSE, : ingredients could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(sCorpus, min.freq = 3, max.words = 100, random.order =
## FALSE, : coffee could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(sCorpus, min.freq = 3, max.words = 100, random.order =
## FALSE, : featuring could not be fit on page. It will not be plotted.

## Warning in wordcloud(sCorpus, min.freq = 3, max.words = 100, random.order =
## FALSE, : business could not be fit on page. It will not be plotted.

## Warning in wordcloud(sCorpus, min.freq = 3, max.words = 100, random.order =
## FALSE, : making could not be fit on page. It will not be plotted.

## Warning in wordcloud(sCorpus, min.freq = 3, max.words = 100, random.order =
## FALSE, : safe could not be fit on page. It will not be plotted.

## Warning in wordcloud(sCorpus, min.freq = 3, max.words = 100, random.order =
## FALSE, : simple could not be fit on page. It will not be plotted.

## Warning in wordcloud(sCorpus, min.freq = 3, max.words = 100, random.order =
## FALSE, : easier could not be fit on page. It will not be plotted.

## Warning in wordcloud(sCorpus, min.freq = 3, max.words = 100, random.order =
## FALSE, : time could not be fit on page. It will not be plotted.

## Warning in wordcloud(sCorpus, min.freq = 3, max.words = 100, random.order =
## FALSE, : around could not be fit on page. It will not be plotted.

## Warning in wordcloud(sCorpus, min.freq = 3, max.words = 100, random.order =
## FALSE, : easily could not be fit on page. It will not be plotted.

## Warning in wordcloud(sCorpus, min.freq = 3, max.words = 100, random.order =
## FALSE, : user could not be fit on page. It will not be plotted.

## Warning in wordcloud(sCorpus, min.freq = 3, max.words = 100, random.order =
## FALSE, : premium could not be fit on page. It will not be plotted.

## Warning in wordcloud(sCorpus, min.freq = 3, max.words = 100, random.order =
## FALSE, : provides could not be fit on page. It will not be plotted.

## Warning in wordcloud(sCorpus, min.freq = 3, max.words = 100, random.order =
## FALSE, : every could not be fit on page. It will not be plotted.

## Warning in wordcloud(sCorpus, min.freq = 3, max.words = 100, random.order =
## FALSE, : money could not be fit on page. It will not be plotted.

## Warning in wordcloud(sCorpus, min.freq = 3, max.words = 100, random.order =
## FALSE, : power could not be fit on page. It will not be plotted.

## Warning in wordcloud(sCorpus, min.freq = 3, max.words = 100, random.order =
## FALSE, : sold could not be fit on page. It will not be plotted.

## Warning in wordcloud(sCorpus, min.freq = 3, max.words = 100, random.order =
## FALSE, : apparel could not be fit on page. It will not be plotted.

## Warning in wordcloud(sCorpus, min.freq = 3, max.words = 100, random.order =
## FALSE, : clothing could not be fit on page. It will not be plotted.

## Warning in wordcloud(sCorpus, min.freq = 3, max.words = 100, random.order =
## FALSE, : bars could not be fit on page. It will not be plotted.

## Warning in wordcloud(sCorpus, min.freq = 3, max.words = 100, random.order =
## FALSE, : enjoy could not be fit on page. It will not be plotted.

## Warning in wordcloud(sCorpus, min.freq = 3, max.words = 100, random.order =
## FALSE, : instead could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(sCorpus, min.freq = 3, max.words = 100, random.order =
## FALSE, : allnatural could not be fit on page. It will not be plotted.

## Warning in wordcloud(sCorpus, min.freq = 3, max.words = 100, random.order =
## FALSE, : mobile could not be fit on page. It will not be plotted.

## Warning in wordcloud(sCorpus, min.freq = 3, max.words = 100, random.order =
## FALSE, : created could not be fit on page. It will not be plotted.

## Warning in wordcloud(sCorpus, min.freq = 3, max.words = 100, random.order =
## FALSE, : usa could not be fit on page. It will not be plotted.

## Warning in wordcloud(sCorpus, min.freq = 3, max.words = 100, random.order =
## FALSE, : dont could not be fit on page. It will not be plotted.

## Warning in wordcloud(sCorpus, min.freq = 3, max.words = 100, random.order =
## FALSE, : hair could not be fit on page. It will not be plotted.

## Warning in wordcloud(sCorpus, min.freq = 3, max.words = 100, random.order =
## FALSE, : women could not be fit on page. It will not be plotted.

## Warning in wordcloud(sCorpus, min.freq = 3, max.words = 100, random.order =
## FALSE, : cards could not be fit on page. It will not be plotted.

## Warning in wordcloud(sCorpus, min.freq = 3, max.words = 100, random.order =
## FALSE, : designs could not be fit on page. It will not be plotted.

## Warning in wordcloud(sCorpus, min.freq = 3, max.words = 100, random.order =
## FALSE, : protection could not be fit on page. It will not be plotted.

## Warning in wordcloud(sCorpus, min.freq = 3, max.words = 100, random.order =
## FALSE, : solution could not be fit on page. It will not be plotted.

## Warning in wordcloud(sCorpus, min.freq = 3, max.words = 100, random.order =
## FALSE, : market could not be fit on page. It will not be plotted.

## Warning in wordcloud(sCorpus, min.freq = 3, max.words = 100, random.order =
## FALSE, : customers could not be fit on page. It will not be plotted.

## Warning in wordcloud(sCorpus, min.freq = 3, max.words = 100, random.order =
## FALSE, : three could not be fit on page. It will not be plotted.

## Warning in wordcloud(sCorpus, min.freq = 3, max.words = 100, random.order =
## FALSE, : youre could not be fit on page. It will not be plotted.

## Warning in wordcloud(sCorpus, min.freq = 3, max.words = 100, random.order =
## FALSE, : want could not be fit on page. It will not be plotted.

## Warning in wordcloud(sCorpus, min.freq = 3, max.words = 100, random.order =
## FALSE, : cover could not be fit on page. It will not be plotted.

## Warning in wordcloud(sCorpus, min.freq = 3, max.words = 100, random.order =
## FALSE, : patented could not be fit on page. It will not be plotted.

## Warning in wordcloud(sCorpus, min.freq = 3, max.words = 100, random.order =
## FALSE, : now could not be fit on page. It will not be plotted.

## Warning in wordcloud(sCorpus, min.freq = 3, max.words = 100, random.order =
## FALSE, : better could not be fit on page. It will not be plotted.

## Warning in wordcloud(sCorpus, min.freq = 3, max.words = 100, random.order =
## FALSE, : flavor could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(sCorpus, min.freq = 3, max.words = 100, random.order =
## FALSE, : phone could not be fit on page. It will not be plotted.

## Warning in wordcloud(sCorpus, min.freq = 3, max.words = 100, random.order =
## FALSE, : great could not be fit on page. It will not be plotted.
```


```
## The words company, designed, make, like and so have appeared more times.
## They do not convey much sentiments but the idea of establishing companies is evident.
```

4.1.3 Model Building :

We use the Document Term Matrix that has been converted to a data frame for building our CART model and arrive at our CART model diagram. We will also calculate the accuracy.

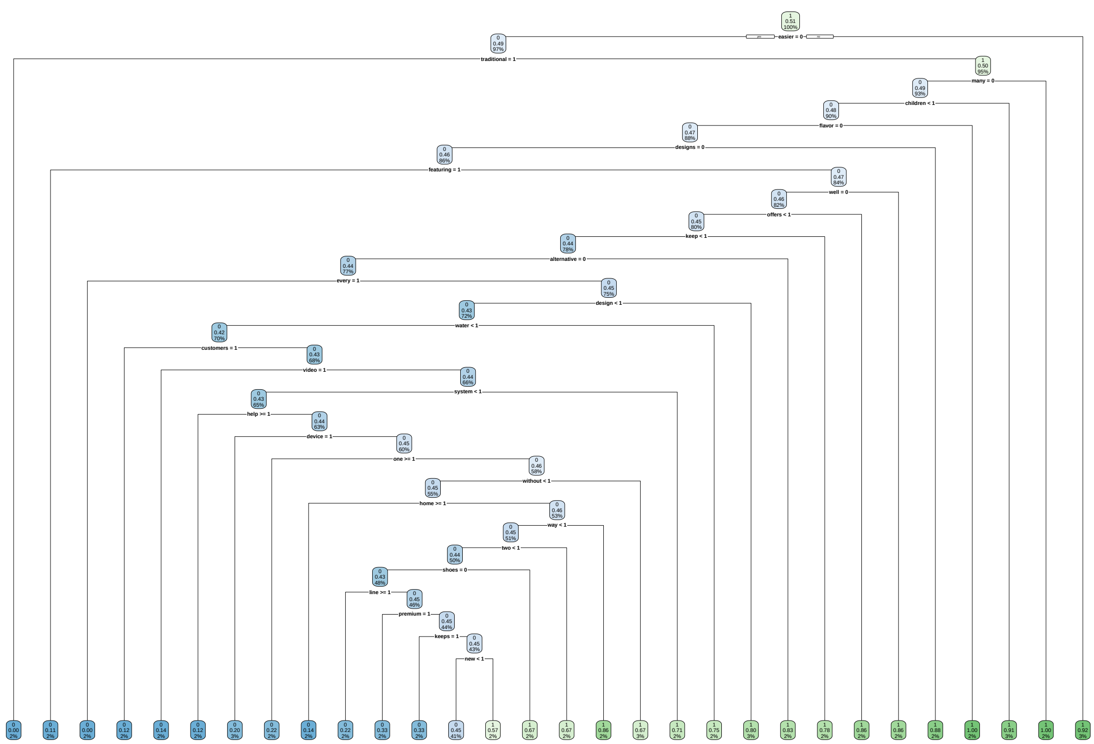
```
## Converting to factor

dataShark$deal = factor(dataShark$deal, levels = c(0,1))
# head(dataShark)

## Set the seed

library(caTools)
set.seed(123)

## Spliting the data to training and testing set

split = sample.split(dataShark$deal, SplitRatio = 0.8)
```

```r
train_set = subset(dataShark, split == TRUE)
test_set = subset(dataShark, split == FALSE)
```

1. CART Model:

```r
## Setting Control Parameter:

cart.ctrl = rpart.control(minsplit = 18, minbucket = 6, cp = 0, xval = 10)

## Model Building:

cart.m1 <- rpart(formula = deal~., data = train_set, method = "class", control = cart.ctrl)
rpart.plot(cart.m1)
```

```
## Warning: labs do not fit even at cex 0.15, there may be some overplotting
```



```r
print(cart.m1)
```

```
## n= 396
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##         1) root 396 195 1 (0.49242424 0.50757576)
##           2) easier< 0.5 383 189 0 (0.50652742 0.49347258)
##             4) traditional>=0.5 8   0 0 (1.00000000 0.00000000) *
```

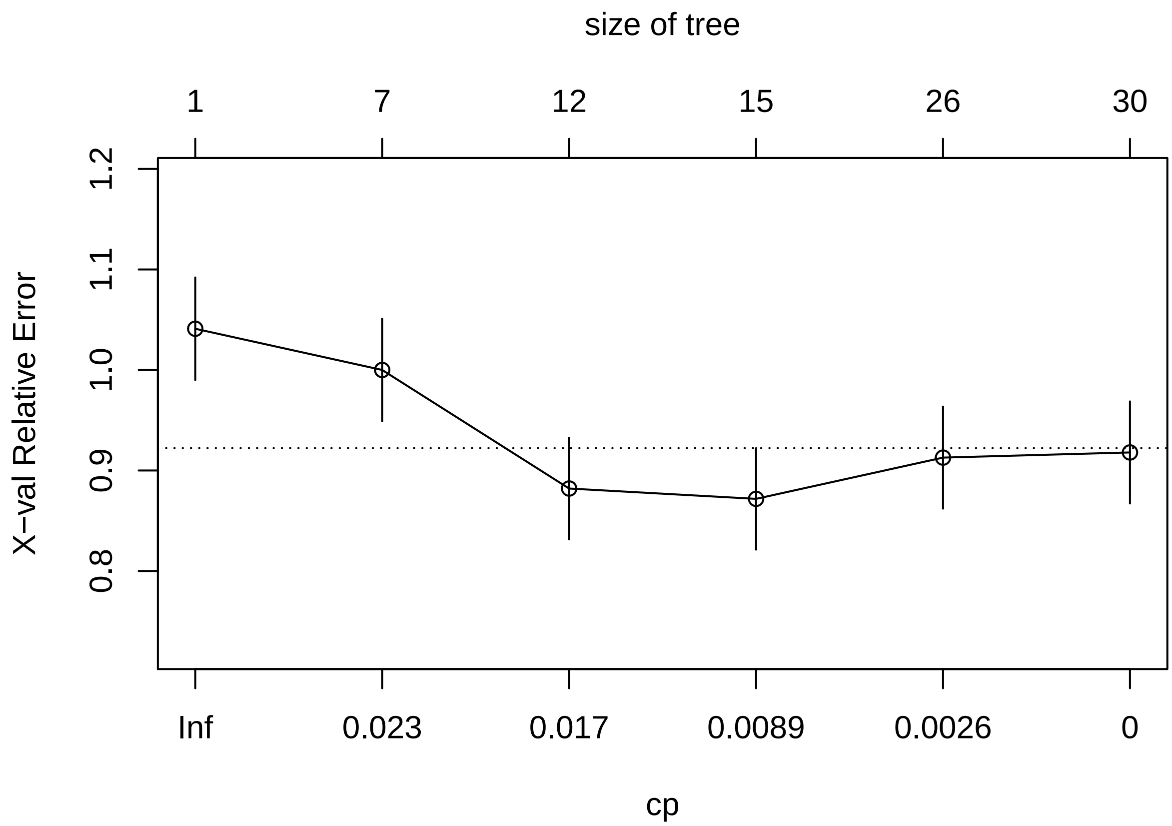
```
##             5) traditional< 0.5 375 186 1 (0.49600000 0.50400000)
##              10) many< 0.5 367 181 0 (0.50681199 0.49318801)
##                20) children< 0.5 356 171 0 (0.51966292 0.48033708)
##                  40) flavor< 0.5 349 164 0 (0.53008596 0.46991404)
##                    80) designs< 0.5 341 157 0 (0.53958944 0.46041056)
##                     160) featuring>=0.5 9   1 0 (0.88888889 0.11111111) *
##                     161) featuring< 0.5 332 156 0 (0.53012048 0.46987952)
##                       322) well< 0.5 325 150 0 (0.53846154 0.46153846)
##                         644) offers< 0.5 318 144 0 (0.54716981 0.45283019)
##                          1288) keep< 0.5 309 137 0 (0.55663430 0.44336570)
##                            2576) alternative< 0.5 303 132 0 (0.56435644 0.43564356)
##                              5152) every>=0.5 7   0 0 (1.00000000 0.00000000) *
##                              5153) every< 0.5 296 132 0 (0.55405405 0.44594595)
##                               10306) design< 0.5 286 124 0 (0.56643357 0.43356643)
##                                 20612) water< 0.5 278 118 0 (0.57553957 0.42446043)
##                                   41224) customers>=0.5 8   1 0 (0.87500000 0.12500000) *
##                                   41225) customers< 0.5 270 117 0 (0.56666667 0.43333333)
##                                     82450) video>=0.5 7   1 0 (0.85714286 0.14285714) *
##                                     82451) video< 0.5 263 116 0 (0.55893536 0.44106464)
##                                      164902) system< 0.5 256 111 0 (0.56640625 0.43359375)
##                                        329804) help>=0.5 8   1 0 (0.87500000 0.12500000) *
##                                        329805) help< 0.5 248 110 0 (0.55645161 0.44354839)
##                                          659610) device>=0.5 10   2 0 (0.80000000 0.20000000) *
##                                          659611) device< 0.5 238 108 0 (0.54621849 0.45378151)
##                                           1319222) one>=0.5 9   2 0 (0.77777778 0.22222222) *
##                                           1319223) one< 0.5 229 106 0 (0.53711790 0.46288210)
##                                             2638446) without< 0.5 217  98 0 (0.54838710 0.45161290)
##                                               5276892) home>=0.5 7   1 0 (0.85714286 0.14285714) *
##                                               5276893) home< 0.5 210  97 0 (0.53809524 0.46190476)
##                                                10553786) way< 0.5 203  91 0 (0.55172414 0.44827586)
##                                                  21107572) two< 0.5 197  87 0 (0.55837563 0.44162437)
##                                                    42215144) shoes< 0.5 191  83 0 (0.56544503 0.4345
##                                                      84430288) line>=0.5 9   2 0 (0.77777778 0.22222
##                                                      84430289) line< 0.5 182  81 0 (0.55494505 0.445
##                                                       168860578) premium>=0.5 6   2 0 (0.66666667 0.
##                                                       168860579) premium< 0.5 176  79 0 (0.55113636
##                                                         337721158) keeps>=0.5 6   2 0 (0.66666667 0.
##                                                         337721159) keeps< 0.5 170  77 0 (0.54705882
##                                                           675442318) new< 0.5 163  73 0 (0.55214724
##                                                           675442319) new>=0.5 7   3 1 (0.42857143 0.
##                                                    42215145) shoes>=0.5 6   2 1 (0.33333333 0.66666
##                                                  21107573) two>=0.5 6   2 1 (0.33333333 0.66666667)
##                                                10553787) way>=0.5 7   1 1 (0.14285714 0.85714286) *
##                                             2638447) without>=0.5 12   4 1 (0.33333333 0.66666667) *
##                                      164903) system>=0.5 7   2 1 (0.28571429 0.71428571) *
##                                 20613) water>=0.5 8   2 1 (0.25000000 0.75000000) *
##                               10307) design>=0.5 10   2 1 (0.20000000 0.80000000) *
##                            2577) alternative>=0.5 6   1 1 (0.16666667 0.83333333) *
##                          1289) keep>=0.5 9   2 1 (0.22222222 0.77777778) *
##                         645) offers>=0.5 7   1 1 (0.14285714 0.85714286) *
##                       323) well>=0.5 7   1 1 (0.14285714 0.85714286) *
##                    81) designs>=0.5 8   1 1 (0.12500000 0.87500000) *
##                  41) flavor>=0.5 7   0 1 (0.00000000 1.00000000) *
##                21) children>=0.5 11   1 1 (0.09090909 0.90909091) *
```

```
##             11) many>=0.5 8   0 1 (0.00000000 1.00000000) *
##          3) easier>=0.5 13   1 1 (0.07692308 0.92307692) *
```

```r
## Pruning the tree

printcp(cart.m1)
```

```
##
## Classification tree:
## rpart(formula = deal ~ ., data = train_set, method = "class",
##     control = cart.ctrl)
##
## Variables actually used in tree construction:
##  [1] alternative children    customers   design      designs
##  [6] device      easier      every       featuring   flavor
## [11] help        home        keep        keeps       line
## [16] many        new         offers      one         premium
## [21] shoes       system      traditional two         video
## [26] water       way         well        without
##
## Root node error: 195/396 = 0.49242
##
## n= 396
##
##          CP nsplit rel error  xerror     xstd
## 1 0.0282051      0   1.00000 1.04103 0.051009
## 2 0.0192308      6   0.82051 1.00000 0.051019
## 3 0.0153846     11   0.72308 0.88205 0.050583
## 4 0.0051282     14   0.67179 0.87179 0.050512
## 5 0.0012821     25   0.58974 0.91282 0.050764
## 6 0.0000000     29   0.58462 0.91795 0.050790
```

```r
plotcp(cart.m1)
```




```
## Extracting the least cpvalue

cart.m1$cptable
```

```
##            CP nsplit rel error    xerror       xstd
## 1 0.028205128      0 1.0000000 1.0410256 0.05100873
## 2 0.019230769      6 0.8205128 1.0000000 0.05101914
## 3 0.015384615     11 0.7230769 0.8820513 0.05058317
## 4 0.005128205     14 0.6717949 0.8717949 0.05051222
## 5 0.001282051     25 0.5897436 0.9128205 0.05076402
## 6 0.000000000     29 0.5846154 0.9179487 0.05078953
```

```
cart.m1$cptable[,"xerror"]
```

```
##         1         2         3         4         5         6
## 1.0410256 1.0000000 0.8820513 0.8717949 0.9128205 0.9179487
```

```
min(cart.m1$cptable[,"xerror"])
```

```
## [1] 0.8717949
```

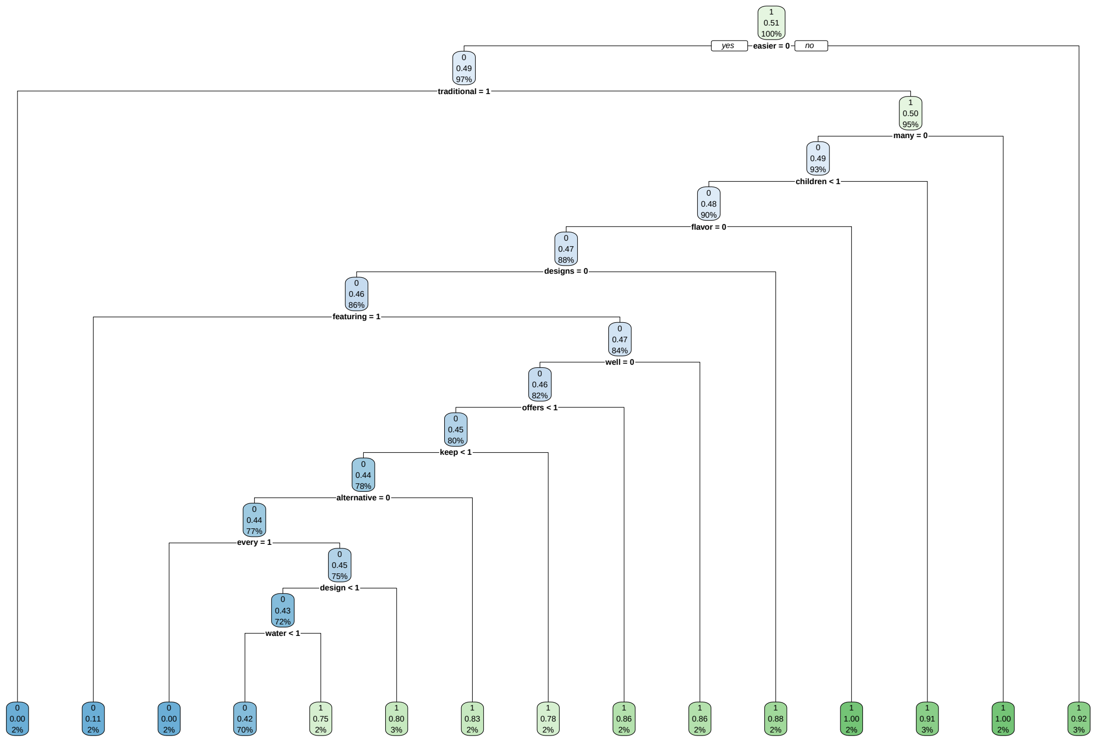
```
## Our least CP value 0.8820513

## Best CP to prune the tree accordingly

cpbest = cart.m1$cptable[which.min(cart.m1$cptable[,"xerror"]), "CP"]
cpbest
```

```
## [1] 0.005128205
```

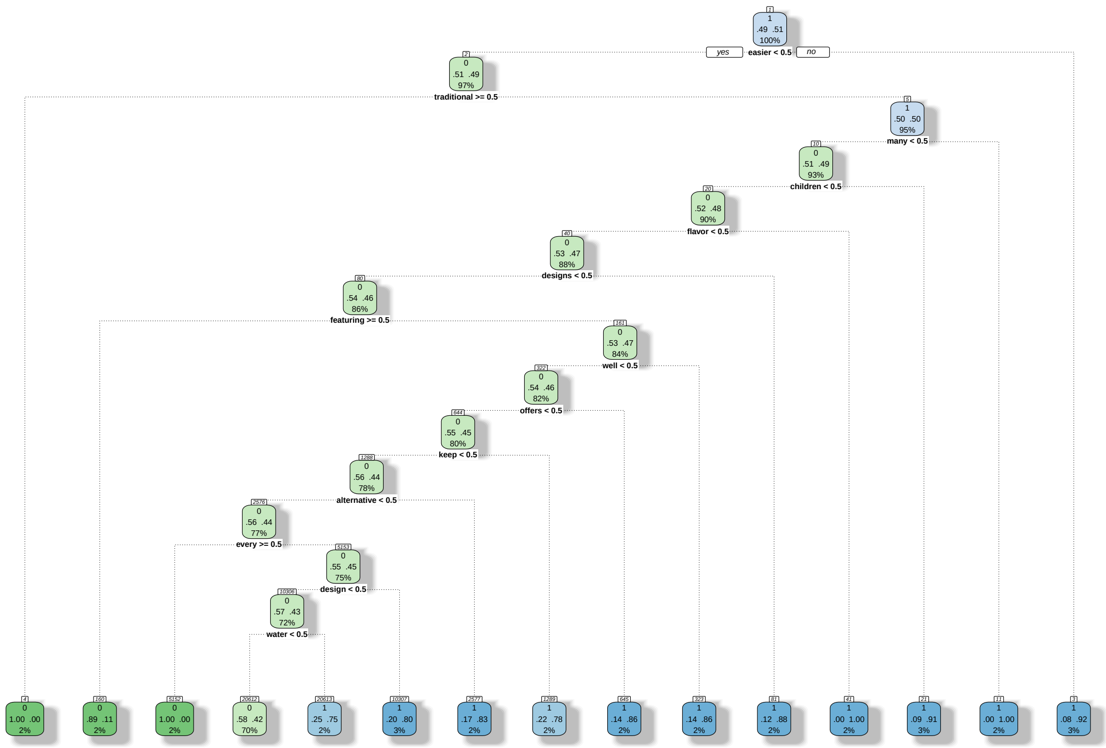
```r
## hence we need to prune the tree at CP = 0.005128205
## PRUNING THE TREE ACCORDINGLY

Pruntree = prune(tree = cart.m1, cp = cpbest)
print(Pruntree)
```

```
## n= 396
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##     1) root 396 195 1 (0.49242424 0.50757576)
##       2) easier< 0.5 383 189 0 (0.50652742 0.49347258)
##         4) traditional>=0.5 8   0 0 (1.00000000 0.00000000) *
##         5) traditional< 0.5 375 186 1 (0.49600000 0.50400000)
##          10) many< 0.5 367 181 0 (0.50681199 0.49318801)
##            20) children< 0.5 356 171 0 (0.51966292 0.48033708)
##              40) flavor< 0.5 349 164 0 (0.53008596 0.46991404)
##                80) designs< 0.5 341 157 0 (0.53958944 0.46041056)
##                 160) featuring>=0.5 9   1 0 (0.88888889 0.11111111) *
##                 161) featuring< 0.5 332 156 0 (0.53012048 0.46987952)
##                   322) well< 0.5 325 150 0 (0.53846154 0.46153846)
##                     644) offers< 0.5 318 144 0 (0.54716981 0.45283019)
##                      1288) keep< 0.5 309 137 0 (0.55663430 0.44336570)
##                        2576) alternative< 0.5 303 132 0 (0.56435644 0.43564356)
##                          5152) every>=0.5 7   0 0 (1.00000000 0.00000000) *
##                          5153) every< 0.5 296 132 0 (0.55405405 0.44594595)
##                           10306) design< 0.5 286 124 0 (0.56643357 0.43356643)
##                             20612) water< 0.5 278 118 0 (0.57553957 0.42446043) *
##                             20613) water>=0.5 8   2 1 (0.25000000 0.75000000) *
##                           10307) design>=0.5 10   2 1 (0.20000000 0.80000000) *
##                        2577) alternative>=0.5 6   1 1 (0.16666667 0.83333333) *
##                      1289) keep>=0.5 9   2 1 (0.22222222 0.77777778) *
##                     645) offers>=0.5 7   1 1 (0.14285714 0.85714286) *
##                   323) well>=0.5 7   1 1 (0.14285714 0.85714286) *
##                81) designs>=0.5 8   1 1 (0.12500000 0.87500000) *
##              41) flavor>=0.5 7   0 1 (0.00000000 1.00000000) *
##            21) children>=0.5 11   1 1 (0.09090909 0.90909091) *
##          11) many>=0.5 8   0 1 (0.00000000 1.00000000) *
##       3) easier>=0.5 13   1 1 (0.07692308 0.92307692) *
```

```r
rpart.plot(Pruntree)
```




```r
library(rattle)

fancyRpartPlot(Pruntree)
```



Rattle 2019−Dec−22 20:29:55 DELL

```r
## Summary
summary(Pruntree)
```

```
## Call:
## rpart(formula = deal ~ ., data = train_set, method = "class",
##     control = cart.ctrl)
##   n= 396
##
##            CP nsplit rel error    xerror       xstd
## 1 0.028205128      0 1.0000000 1.0410256 0.05100873
## 2 0.019230769      6 0.8205128 1.0000000 0.05101914
## 3 0.015384615     11 0.7230769 0.8820513 0.05058317
## 4 0.005128205     14 0.6717949 0.8717949 0.05051222
##
## Variable importance
##      easier        many traditional    children      flavor       every
##           8           7           7           6           6           4
##     designs      design   featuring      offers        well        keep
##           4           4           4           4           4           3
## alternative      peanut       water     catalog        gift     pumpkin
##           3           3           3           2           2           2
##       three  overlooked     remotes       serve       nylon    struggle
##           2           2           2           2           2           1
##        user  dishwasher    clothing        cant      giving      follow
##           1           1           1           1           1           1
##         low       outer   efficient      pieces sustainable        home
```

```
##          1           1           1           1           1           1
##       onto       books        ones       young       beach     started
##          1           1           1           1           1           1
##
## Node number 1: 396 observations,    complexity param=0.02820513
##   predicted class=1  expected loss=0.4924242  P(node) =1
##     class counts:   195   201
##    probabilities: 0.492 0.508
##   left son=2 (383 obs) right son=3 (13 obs)
##   Primary splits:
##       easier      < 0.5 to the left,  improve=4.641029, (0 missing)
##       traditional < 0.5 to the right, improve=4.207123, (0 missing)
##       children    < 0.5 to the left,  improve=4.142045, (0 missing)
##       flavor      < 0.5 to the left,  improve=3.959700, (0 missing)
##       three       < 0.5 to the left,  improve=3.455831, (0 missing)
##   Surrogate splits:
##       user < 1.5 to the left,  agree=0.972, adj=0.154, (0 split)
##       onto < 0.5 to the left,  agree=0.970, adj=0.077, (0 split)
##       home < 1.5 to the left,  agree=0.970, adj=0.077, (0 split)
##
## Node number 2: 383 observations,    complexity param=0.02820513
##   predicted class=0  expected loss=0.4934726  P(node) =0.9671717
##     class counts:   194   189
##    probabilities: 0.507 0.493
##   left son=4 (8 obs) right son=5 (375 obs)
##   Primary splits:
##       traditional < 0.5 to the right, improve=3.979363, (0 missing)
##       children    < 0.5 to the left,  improve=3.912622, (0 missing)
##       flavor      < 0.5 to the left,  improve=3.658852, (0 missing)
##       three       < 0.5 to the left,  improve=3.127840, (0 missing)
##       materials   < 0.5 to the left,  improve=3.127840, (0 missing)
##   Surrogate splits:
##       nylon < 0.5 to the right, agree=0.984, adj=0.25, (0 split)
##
## Node number 3: 13 observations
##   predicted class=1  expected loss=0.07692308  P(node) =0.03282828
##     class counts:     1    12
##    probabilities: 0.077 0.923
##
## Node number 4: 8 observations
##   predicted class=0  expected loss=0  P(node) =0.02020202
##     class counts:     8     0
##    probabilities: 1.000 0.000
##
## Node number 5: 375 observations,    complexity param=0.02820513
##   predicted class=1  expected loss=0.496  P(node) =0.9469697
##     class counts:   186   189
##    probabilities: 0.496 0.504
##   left son=10 (367 obs) right son=11 (8 obs)
##   Primary splits:
##       many      < 0.5 to the left,  improve=4.022060, (0 missing)
##       children  < 0.5 to the left,  improve=3.719269, (0 missing)
##       flavor    < 0.5 to the left,  improve=3.509739, (0 missing)
##       three     < 0.5 to the left,  improve=3.000195, (0 missing)
```

```
##       materials < 0.5 to the left,  improve=3.000195, (0 missing)
##   Surrogate splits:
##       serve      < 0.5 to the left,  agree=0.984, adj=0.25, (0 split)
##       overlooked < 0.5 to the left,  agree=0.984, adj=0.25, (0 split)
##       remotes    < 0.5 to the left,  agree=0.984, adj=0.25, (0 split)
##
## Node number 10: 367 observations,    complexity param=0.02820513
##   predicted class=0  expected loss=0.493188  P(node) =0.9267677
##     class counts:   186   181
##    probabilities: 0.507 0.493
##   left son=20 (356 obs) right son=21 (11 obs)
##   Primary splits:
##       children < 0.5 to the left,  improve=3.923039, (0 missing)
##       flavor   < 0.5 to the left,  improve=3.665940, (0 missing)
##       three    < 0.5 to the left,  improve=3.133530, (0 missing)
##       started  < 0.5 to the left,  improve=3.133530, (0 missing)
##       well     < 0.5 to the left,  improve=2.889280, (0 missing)
##   Surrogate splits:
##       dishwasher < 0.5 to the left,  agree=0.975, adj=0.182, (0 split)
##       ones       < 0.5 to the left,  agree=0.973, adj=0.091, (0 split)
##       young      < 0.5 to the left,  agree=0.973, adj=0.091, (0 split)
##       books      < 0.5 to the left,  agree=0.973, adj=0.091, (0 split)
##
## Node number 11: 8 observations
##   predicted class=1  expected loss=0  P(node) =0.02020202
##     class counts:     0     8
##    probabilities: 0.000 1.000
##
## Node number 20: 356 observations,    complexity param=0.02820513
##   predicted class=0  expected loss=0.4803371  P(node) =0.8989899
##     class counts:   185   171
##    probabilities: 0.520 0.480
##   left son=40 (349 obs) right son=41 (7 obs)
##   Primary splits:
##       flavor  < 0.5 to the left,  improve=3.856524, (0 missing)
##       three   < 0.5 to the left,  improve=3.296148, (0 missing)
##       started < 0.5 to the left,  improve=3.296148, (0 missing)
##       well    < 0.5 to the left,  improve=3.082388, (0 missing)
##       designs < 0.5 to the left,  improve=2.549432, (0 missing)
##   Surrogate splits:
##       peanut  < 0.5 to the left,  agree=0.989, adj=0.429, (0 split)
##       gift    < 1.5 to the left,  agree=0.986, adj=0.286, (0 split)
##       three   < 0.5 to the left,  agree=0.986, adj=0.286, (0 split)
##       pumpkin < 0.5 to the left,  agree=0.986, adj=0.286, (0 split)
##       catalog < 0.5 to the left,  agree=0.986, adj=0.286, (0 split)
##
## Node number 21: 11 observations
##   predicted class=1  expected loss=0.09090909  P(node) =0.02777778
##     class counts:     1    10
##    probabilities: 0.091 0.909
##
## Node number 40: 349 observations,    complexity param=0.02820513
##   predicted class=0  expected loss=0.469914  P(node) =0.8813131
##     class counts:   185   164
```

```
##    probabilities: 0.530 0.470
##   left son=80 (341 obs) right son=81 (8 obs)
##   Primary splits:
##       designs < 0.5 to the left,  improve=2.687110, (0 missing)
##       well    < 0.5 to the left,  improve=2.687110, (0 missing)
##       dont    < 0.5 to the left,  improve=2.687110, (0 missing)
##       every   < 0.5 to the right, improve=2.378652, (0 missing)
##       keep    < 0.5 to the left,  improve=2.243416, (0 missing)
##   Surrogate splits:
##       clothing < 1.5 to the left,  agree=0.983, adj=0.25, (0 split)
##
## Node number 41: 7 observations
##   predicted class=1  expected loss=0  P(node) =0.01767677
##     class counts:     0     7
##    probabilities: 0.000 1.000
##
## Node number 80: 341 observations,    complexity param=0.01923077
##   predicted class=0  expected loss=0.4604106  P(node) =0.8611111
##     class counts:   184   157
##    probabilities: 0.540 0.460
##   left son=160 (9 obs) right son=161 (332 obs)
##   Primary splits:
##       featuring < 0.5 to the right, improve=2.255717, (0 missing)
##       every     < 0.5 to the right, improve=2.255717, (0 missing)
##       well      < 0.5 to the left,  improve=2.249733, (0 missing)
##       range     < 0.5 to the left,  improve=2.249733, (0 missing)
##       dont      < 0.5 to the left,  improve=2.249733, (0 missing)
##
## Node number 81: 8 observations
##   predicted class=1  expected loss=0.125  P(node) =0.02020202
##     class counts:     1     7
##    probabilities: 0.125 0.875
##
## Node number 160: 9 observations
##   predicted class=0  expected loss=0.1111111  P(node) =0.02272727
##     class counts:     8     1
##    probabilities: 0.889 0.111
##
## Node number 161: 332 observations,    complexity param=0.01923077
##   predicted class=0  expected loss=0.4698795  P(node) =0.8383838
##     class counts:   176   156
##    probabilities: 0.530 0.470
##   left son=322 (325 obs) right son=323 (7 obs)
##   Primary splits:
##       well  < 0.5 to the left,  improve=2.144843, (0 missing)
##       range < 0.5 to the left,  improve=2.144843, (0 missing)
##       dont  < 0.5 to the left,  improve=2.144843, (0 missing)
##       every < 0.5 to the right, improve=1.950059, (0 missing)
##       keep  < 0.5 to the left,  improve=1.753972, (0 missing)
##   Surrogate splits:
##       giving  < 0.5 to the left,  agree=0.985, adj=0.286, (0 split)
##       started < 0.5 to the left,  agree=0.982, adj=0.143, (0 split)
##       beach   < 0.5 to the left,  agree=0.982, adj=0.143, (0 split)
##
```

```
## Node number 322: 325 observations,    complexity param=0.01923077
##   predicted class=0  expected loss=0.4615385  P(node) =0.8207071
##     class counts:   175   150
##    probabilities: 0.538 0.462
##   left son=644 (318 obs) right son=645 (7 obs)
##   Primary splits:
##       offers   < 0.5 to the left,  improve=2.239270, (0 missing)
##       every    < 0.5 to the right, improve=1.857862, (0 missing)
##       keep     < 0.5 to the left,  improve=1.851401, (0 missing)
##       solution < 0.5 to the left,  improve=1.851401, (0 missing)
##       usa      < 0.5 to the left,  improve=1.689977, (0 missing)
##
## Node number 323: 7 observations
##   predicted class=1  expected loss=0.1428571  P(node) =0.01767677
##     class counts:     1     6
##    probabilities: 0.143 0.857
##
## Node number 644: 318 observations,    complexity param=0.01923077
##   predicted class=0  expected loss=0.4528302  P(node) =0.8030303
##     class counts:   174   144
##    probabilities: 0.547 0.453
##   left son=1288 (309 obs) right son=1289 (9 obs)
##   Primary splits:
##       keep        < 0.5 to the left,  improve=1.955995, (0 missing)
##       solution    < 0.5 to the left,  improve=1.955995, (0 missing)
##       travel      < 0.5 to the left,  improve=1.770803, (0 missing)
##       alternative < 0.5 to the left,  improve=1.770803, (0 missing)
##       built       < 0.5 to the left,  improve=1.770803, (0 missing)
##   Surrogate splits:
##       cant      < 0.5 to the left,  agree=0.981, adj=0.333, (0 split)
##       efficient < 0.5 to the left,  agree=0.978, adj=0.222, (0 split)
##       pieces    < 0.5 to the left,  agree=0.978, adj=0.222, (0 split)
##       school    < 0.5 to the left,  agree=0.975, adj=0.111, (0 split)
##       place     < 0.5 to the left,  agree=0.975, adj=0.111, (0 split)
##
## Node number 645: 7 observations
##   predicted class=1  expected loss=0.1428571  P(node) =0.01767677
##     class counts:     1     6
##    probabilities: 0.143 0.857
##
## Node number 1288: 309 observations,    complexity param=0.01923077
##   predicted class=0  expected loss=0.4433657  P(node) =0.780303
##     class counts:   172   137
##    probabilities: 0.557 0.443
##   left son=2576 (303 obs) right son=2577 (6 obs)
##   Primary splits:
##       alternative < 0.5 to the left,  improve=1.861034, (0 missing)
##       built       < 0.5 to the left,  improve=1.861034, (0 missing)
##       dont        < 0.5 to the left,  improve=1.861034, (0 missing)
##       design      < 0.5 to the left,  improve=1.838727, (0 missing)
##       every       < 0.5 to the right, improve=1.664809, (0 missing)
##
## Node number 1289: 9 observations
##   predicted class=1  expected loss=0.2222222  P(node) =0.02272727
```

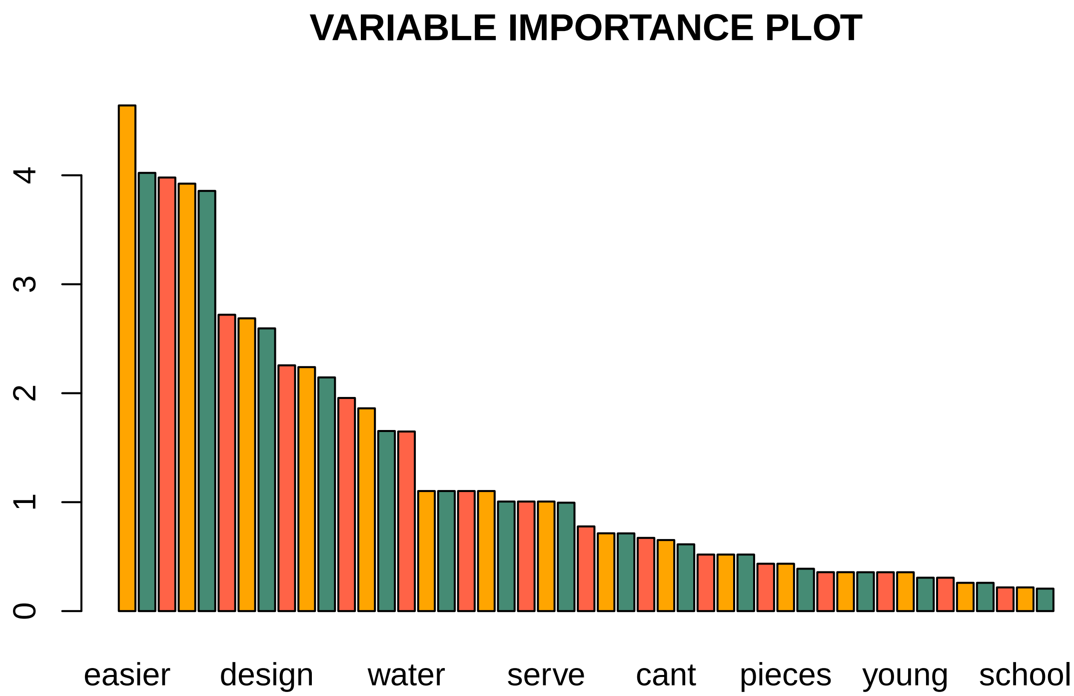
```
##     class counts:     2     7
##    probabilities: 0.222 0.778
##
## Node number 2576: 303 observations,    complexity param=0.01538462
##   predicted class=0  expected loss=0.4356436  P(node) =0.7651515
##     class counts:   171   132
##    probabilities: 0.564 0.436
##   left son=5152 (7 obs) right son=5153 (296 obs)
##   Primary splits:
##       every  < 0.5 to the right, improve=2.719829, (0 missing)
##       design < 0.5 to the left,  improve=1.941531, (0 missing)
##       built  < 0.5 to the left,  improve=1.936227, (0 missing)
##       video  < 0.5 to the right, improve=1.228516, (0 missing)
##       one    < 0.5 to the right, improve=1.140232, (0 missing)
##   Surrogate splits:
##       struggle    < 0.5 to the right, agree=0.983, adj=0.286, (0 split)
##       sustainable < 0.5 to the right, agree=0.980, adj=0.143, (0 split)
##
## Node number 2577: 6 observations
##   predicted class=1  expected loss=0.1666667  P(node) =0.01515152
##     class counts:     1     5
##    probabilities: 0.167 0.833
##
## Node number 5152: 7 observations
##   predicted class=0  expected loss=0  P(node) =0.01767677
##     class counts:     7     0
##    probabilities: 1.000 0.000
##
## Node number 5153: 296 observations,    complexity param=0.01538462
##   predicted class=0  expected loss=0.4459459  P(node) =0.7474747
##     class counts:   164   132
##    probabilities: 0.554 0.446
##   left son=10306 (286 obs) right son=10307 (10 obs)
##   Primary splits:
##       design   < 0.5 to the left,  improve=2.594746, (0 missing)
##       solution < 0.5 to the left,  improve=1.838086, (0 missing)
##       created  < 0.5 to the left,  improve=1.838086, (0 missing)
##       built    < 0.5 to the left,  improve=1.838086, (0 missing)
##       video    < 0.5 to the right, improve=1.317230, (0 missing)
##   Surrogate splits:
##       low          < 0.5 to the left,  agree=0.973, adj=0.2, (0 split)
##       follow       < 0.5 to the left,  agree=0.973, adj=0.2, (0 split)
##       outer        < 0.5 to the left,  agree=0.973, adj=0.2, (0 split)
##       toys         < 0.5 to the left,  agree=0.970, adj=0.1, (0 split)
##       construction < 0.5 to the left,  agree=0.970, adj=0.1, (0 split)
##
## Node number 10306: 286 observations,    complexity param=0.01538462
##   predicted class=0  expected loss=0.4335664  P(node) =0.7222222
##     class counts:   162   124
##    probabilities: 0.566 0.434
##   left son=20612 (278 obs) right son=20613 (8 obs)
##   Primary splits:
##       water     < 0.5 to the left,  improve=1.648186, (0 missing)
##       customers < 0.5 to the right, improve=1.567251, (0 missing)
```

```
##       video     < 0.5 to the right, improve=1.212852, (0 missing)
##       shoes     < 0.5 to the left,  improve=1.130926, (0 missing)
##       fun       < 0.5 to the right, improve=1.130597, (0 missing)
##   Surrogate splits:
##       companion < 0.5 to the left,  agree=0.976, adj=0.125, (0 split)
##
## Node number 10307: 10 observations
##   predicted class=1  expected loss=0.2  P(node) =0.02525253
##     class counts:     2     8
##    probabilities: 0.200 0.800
##
## Node number 20612: 278 observations
##   predicted class=0  expected loss=0.4244604  P(node) =0.7020202
##     class counts:   160   118
##    probabilities: 0.576 0.424
##
## Node number 20613: 8 observations
##   predicted class=1  expected loss=0.25  P(node) =0.02020202
##     class counts:     2     6
##    probabilities: 0.250 0.750
```

```
print(Pruntree$variable.importance)
```

```
##       easier         many  traditional     children       flavor
##    4.6410287    4.0220599    3.9793629    3.9230391    3.8565243
##        every      designs       design    featuring       offers
##    2.7198287    2.6871098    2.5947458    2.2557169    2.2392702
##         well         keep  alternative       peanut        water
##    2.1448431    1.9559952    1.8610337    1.6527961    1.6481863
##      catalog         gift      pumpkin        three   overlooked
##    1.1018641    1.1018641    1.1018641    1.1018641    1.0055150
##      remotes        serve        nylon     struggle         user
##    1.0055150    1.0055150    0.9948407    0.7770939    0.7140044
##   dishwasher     clothing         cant       giving       follow
##    0.7132798    0.6717774    0.6519984    0.6128123    0.5189492
##          low        outer    efficient       pieces  sustainable
##    0.5189492    0.5189492    0.4346656    0.4346656    0.3885470
##         home         onto        books         ones        young
##    0.3570022    0.3570022    0.3566399    0.3566399    0.3566399
##        beach      started construction         toys        place
##    0.3064062    0.3064062    0.2594746    0.2594746    0.2173328
##       school    companion
##    0.2173328    0.2060233
```

```
barplot(sort(Pruntree$variable.importance, decreasing = TRUE),main = "VARIABLE IMPORTANCE PLOT", col = 
```




```
## From the variable importance, some of the significant words in getting the deal are below
## easier, traditional, children, flavor, strong, solution, cleaning, etc...
```

```
#Scoring/Predicting the training and test dataset

train_set$predict.class = predict(Pruntree, data = train_set, type="class")
train_set$predict.score = predict(Pruntree, data = train_set)

#head(train_set)

test_set$predict.class = predict(Pruntree, newdata = test_set, type="class")
test_set$predict.score = predict(Pruntree, newdata = test_set)

#head(b_test)
```

```
## Confusion matrix for CART model

conf.tr = with(train_set,table(deal,predict.class))
conf.tr
```

```
##     predict.class
## deal   0   1
##    0 183  12
##    1 119  82
```

```
conf.te = with(test_set,table(deal,predict.class))
conf.te
```

```
##     predict.class
## deal  0  1
##    0 40  9
##    1 38 12
```

```
##  Accuracy:

accuracy.tr = (conf.tr[1,1]+conf.tr[2,2])/(conf.tr[1,1]+conf.tr[1,2]+conf.tr[2,1]+conf.tr[2,2])

accuracy.te = (conf.te[1,1]+conf.te[2,2])/(conf.te[1,1]+conf.te[1,2]+conf.te[2,1]+conf.te[2,2])

accuracy.tr
```

```
## [1] 0.6691919
```

```
accuracy.te
```

```
## [1] 0.5252525
```

```
## CART is not considered a great model as the accuracy is quite low (train = 67%, test = 52%)
## Moreover, though the model has predicted the non deals better
## It has a very large number of mis-predictions on the deals
```

2. Random Forest Model:

```
# Data spliting
library(caTools)

# Setting seed
set.seed(123)

## Splitting to train and test
split1 = sample.split(dataShark$deal, SplitRatio = 0.8)
train_2 = subset(dataShark, split1 == TRUE)
test_2 = subset(dataShark, split1 == FALSE)
```

Model Building:

```
## Fitting Random Forest Model:

classifier = randomForest(x = train_2[-1563], y = train_2$deal, ntree = 5)

# Predicting the Test set results

y_pred = predict(classifier, newdata = test_2[-1563])
# y_pred
```

Confusion Matrix and Accuracy Evaluation:

```
# Making the Confusion Matrix

cm_rf = with(test_2, table(deal, y_pred))
cm_rf
```

```
##     y_pred
```

```
## deal  0  1
##    0 37 12
##    1 32 18
```

```r
## We observe about 75% correct prediction of no deals
## But the mispredictions on the number of deals is more, which makes it not a good working model.

# Accuracy:

accuracy.rf = (cm_rf[1,1]+cm_rf[2,2])/(cm_rf[1,1]+cm_rf[1,2]+cm_rf[2,1]+cm_rf[2,2])
accuracy.rf
```

```
## [1] 0.5555556
```

```r
## The random forest model gives an accuracy of 55.5% which is an average performance
```

3. Logistic Regression Model:

```r
## Using the same data split for the random forest model:

# Data spliting
# library(caTools)

# Setting seed
# set.seed(123)

## Splitting to train and test
# split1 = sample.split(dataShark$deal, SplitRatio = 0.8)
# train_2 = subset(dataShark, split1 == TRUE)
# test_2 = subset(dataShark, split1 == FALSE)
```

Building a Logit Model:

Logistic Regression Model 1:

```r
## Logistic Regression Model 1:

Logit1 = glm(formula = deal~., data = train_2, family = binomial)
predLog1 = predict(Logit1, newdata = test_2, type = "response")
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type
## == : prediction from a rank-deficient fit may be misleading
```

```r
cmLogit1 = table(test_2$deal, predLog1 > 0.3)
cmLogit1
```

```
##
##     FALSE TRUE
##   0    23   26
##   1    24   26
```

```r
## Accuracy:
acc.log1 = (cmLogit1[1,1]+cmLogit1[2,2])/(cmLogit1[1,1]+cmLogit1[1,2]+cmLogit1[2,1]+cmLogit1[2,2])
acc.log1
```

```
## [1] 0.4949495
```

```
## The accuracy here falls to 49.5% which is lower compared to the RF model
```

Logistic Regression Model 2:

```r
# Tweaking up the threshold to 0.8 to review the change in accuracy

## Logistic Regression Model 2:

Logit2 = glm(formula = deal~., data = train_2, family = binomial)
predLog2 = predict(Logit2, newdata = test_2, type = "response")
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type
## == : prediction from a rank-deficient fit may be misleading
```

```r
cmLogit2 = table(test_2$deal, predLog2 > 0.9)
cmLogit2
```

```
##
##     FALSE TRUE
##   0    28   21
##   1    25   25
```

```r
## Accuracy:

acc.log2 = (cmLogit2[1,1]+cmLogit2[2,2])/(cmLogit2[1,1]+cmLogit2[1,2]+cmLogit2[2,1]+cmLogit2[2,2])
acc.log2
```

```
## [1] 0.5353535
```

```r
# This has proved a 4 point increase in accuracy from 49.5%
# This is not a great model for the given data, when compared to our random forest model
```

4.2 Step 2:

```r
# Data loading:

# shark.df = read.csv("Dataset (1).csv")
# View(shark.df)

# Converting to correct data types:

# shark.df$description = as.character(shark.df$description)
# shark.df$deal = ifelse(shark.df$deal=="TRUE",1,0)
# shark.df$deal = as.factor(shark.df$deal)
```

4.2.1 Creating a new variable called "Ratio"

```r
## Ratio column:
shark.df$ratio = shark.df$askedFor/shark.df$valuation

head(shark.df)
```

```
##   deal
## 1    0
## 2    1
```

```
## 3    1
## 4    0
## 5    0
## 6    1
##
## 1
## 2
## 3
## 4
## 5
## 6 One of the first entrepreneurs to pitch on Shark Tank, Susan Knapp presented A Perfect Pear, her li
##   episode            category               entrepreneurs        location
## 1       1           Novelties              Darrin Johnson    St. Paul, MN
## 2       1      Specialty Food                  Tod Wilson    Somerset, NJ
## 3       1 Baby and Child Care             Tiffany Krumins     Atlanta, GA
## 4       1   Consumer Services Nick Friedman, Omar Soliman       Tampa, FL
## 5       1   Consumer Services              Kevin Flannery        Cary, NC
## 6       2      Specialty Food                 Susan Knapp Napa Valley, CA
##                              website askedFor exchangeForStake valuation
## 1                                     1000000               15   6666667
## 2                http://whybake.com/   460000               10   4600000
## 3      http://www.avatheelephant.com/    50000               15    333333
## 4 http://collegehunkshaulingjunk.com/   250000               25   1000000
## 5            http://www.wispots.com/  1200000               10  12000000
## 6         http://www.aperfectpear.com   500000               15   3333333
##   season           shark1          shark2       shark3       shark4
## 1      1 Barbara Corcoran Robert Herjavec Kevin O'Leary Daymond John
## 2      1 Barbara Corcoran Robert Herjavec Kevin O'Leary Daymond John
## 3      1 Barbara Corcoran Robert Herjavec Kevin O'Leary Daymond John
## 4      1 Barbara Corcoran Robert Herjavec Kevin O'Leary Daymond John
## 5      1 Barbara Corcoran Robert Herjavec Kevin O'Leary Daymond John
## 6      1 Barbara Corcoran Robert Herjavec Kevin O'Leary Daymond John
##             shark5                       title episode.season
## 1 Kevin Harrington                   Ionic Ear            1-1
## 2 Kevin Harrington       Mr. Tod's Pie Factory            1-1
## 3 Kevin Harrington            Ava the Elephant            1-1
## 4 Kevin Harrington College Foxes Packing Boxes            1-1
## 5 Kevin Harrington                     Wispots            1-1
## 6 Kevin Harrington              A Perfect Pear            1-2
##   Multiple.Entreprenuers     ratio
## 1                  FALSE 0.1500000
## 2                  FALSE 0.1000000
## 3                  FALSE 0.1500002
## 4                  FALSE 0.2500000
## 5                  FALSE 0.1000000
## 6                  FALSE 0.1500000
```

4.2.2 Including column to dataframe:

```r
## Checking our corpus if it is cleaned:
writeLines(strwrap(sCorpus[[71]]$content, 100))
```

```
## allergy season bringing maybe time give first defense nasal screens try screens ward allergies
## without medication first defense nasal screens selfadhesive strips attach nose cover nasal passage
```

```
## filter air coming nose keeps allergens product tested significantly reduce particles microns sub
## micron levels shown effective
```

```r
## Document Text Matrix
# dtm = DocumentTermMatrix(sCorpus)

## Removing the least occuring terms (sparse terms) from the text
## Cleaning the data upto 99.7 %
# dtm = removeSparseTerms(dtm, 0.997)
## dtm contains documents: 495, terms: 1562


## Converting to a data frame
dataShark2 = as.data.frame(as.matrix(dtm))

## Include the dependent column to the new data frame
dataShark2$deal = shark.df$deal
dataShark2$ratio = shark.df$ratio

## Now we have both the columns (deal and ratio) included to the data frame
```

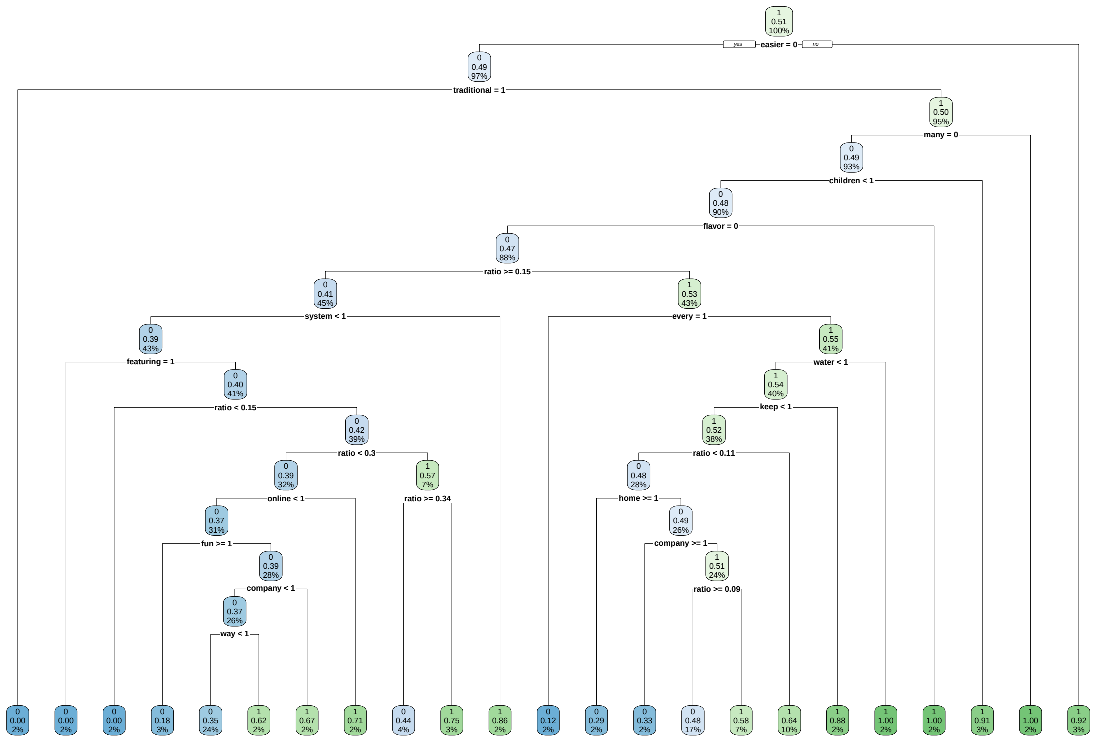
4.2.3 Model Building:

Now we are building models on the data frame two which contains the new column **ratio** along with the dependent variable deal. Let us see what difference does this bring to the model and their accuracy. We will build a CART, Random Forest and a Logistic Regression model, and evaluate the accuracy.

```r
## Converting to factor

dataShark2$deal = factor(dataShark2$deal, levels = c(0,1))
# head(dataShark)

## Set the seed

library(caTools)
set.seed(123)

## Spliting the data to training and testing set

split3 = sample.split(dataShark2$deal, SplitRatio = 0.8)
train_c2 = subset(dataShark2, split3 == TRUE)
test_c2 = subset(dataShark2, split3 == FALSE)
```

1. CART model (New):

```r
## Setting Control Parameter:
# cart.ctrl = rpart.control(minsplit = 18, minbucket = 6, cp = 0, xval = 10)

## Model Building:

cart.m2 <- rpart(formula = deal~., data = train_c2, method = "class", control = cart.ctrl)
rpart.plot(cart.m2)
```



```r
print(cart.m2)
```

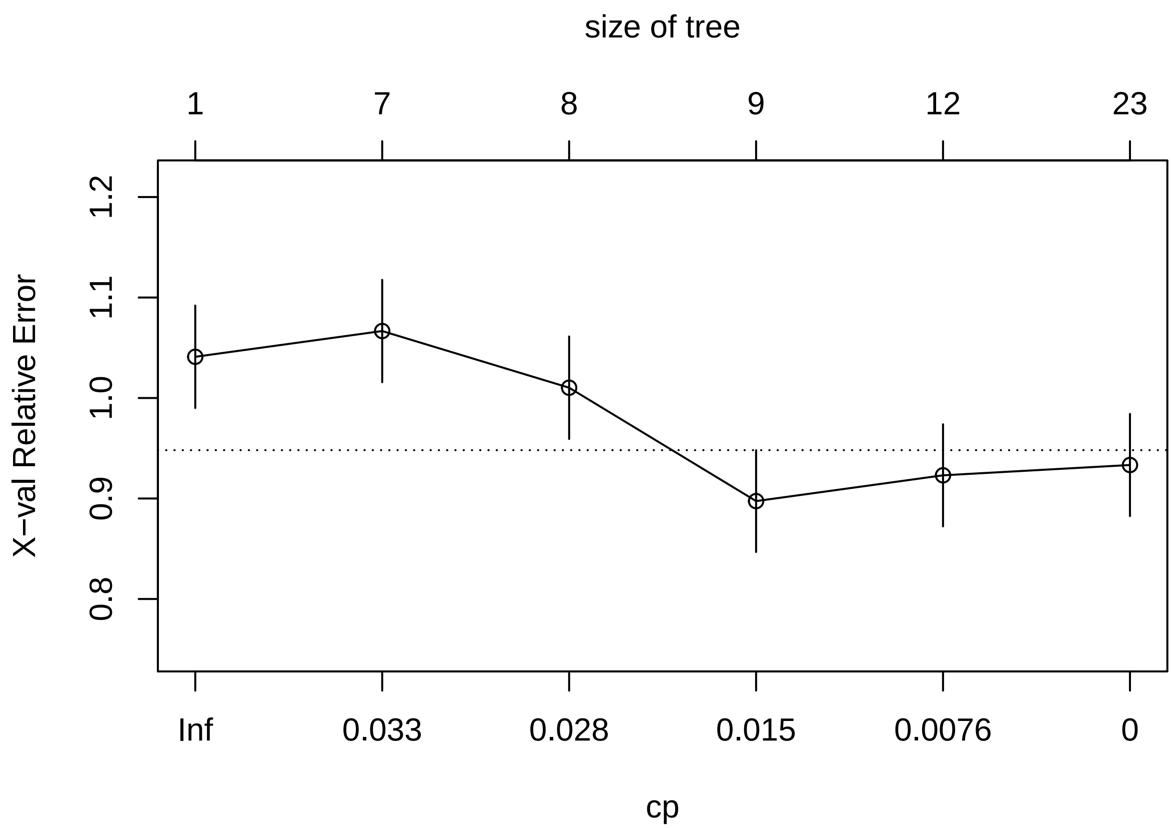
```
## n= 396
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##     1) root 396 195 1 (0.49242424 0.50757576)
##       2) easier< 0.5 383 189 0 (0.50652742 0.49347258)
##         4) traditional>=0.5 8   0 0 (1.00000000 0.00000000) *
##         5) traditional< 0.5 375 186 1 (0.49600000 0.50400000)
##          10) many< 0.5 367 181 0 (0.50681199 0.49318801)
##            20) children< 0.5 356 171 0 (0.51966292 0.48033708)
##              40) flavor< 0.5 349 164 0 (0.53008596 0.46991404)
##                80) ratio>=0.15 177  72 0 (0.59322034 0.40677966)
##                 160) system< 0.5 170  66 0 (0.61176471 0.38823529)
##                   320) featuring>=0.5 7   0 0 (1.00000000 0.00000000) *
##                   321) featuring< 0.5 163  66 0 (0.59509202 0.40490798)
##                     642) ratio< 0.1500001 7   0 0 (1.00000000 0.00000000) *
##                     643) ratio>=0.1500001 156  66 0 (0.57692308 0.42307692)
##                      1286) ratio< 0.3000002 128  50 0 (0.60937500 0.39062500)
##                        2572) online< 0.5 121  45 0 (0.62809917 0.37190083)
##                          5144) fun>=0.5 11   2 0 (0.81818182 0.18181818) *
##                          5145) fun< 0.5 110  43 0 (0.60909091 0.39090909)
##                           10290) company< 0.5 104  39 0 (0.62500000 0.37500000)
##                             20580) way< 0.5 96  34 0 (0.64583333 0.35416667) *
```

```
##                             20581) way>=0.5 8   3 1 (0.37500000 0.62500000) *
##                           10291) company>=0.5 6   2 1 (0.33333333 0.66666667) *
##                        2573) online>=0.5 7   2 1 (0.28571429 0.71428571) *
##                      1287) ratio>=0.3000002 28  12 1 (0.42857143 0.57142857)
##                        2574) ratio>=0.3350002 16   7 0 (0.56250000 0.43750000) *
##                        2575) ratio< 0.3350002 12   3 1 (0.25000000 0.75000000) *
##                   161) system>=0.5 7   1 1 (0.14285714 0.85714286) *
##                  81) ratio< 0.15 172  80 1 (0.46511628 0.53488372)
##                   162) every>=0.5 8   1 0 (0.87500000 0.12500000) *
##                   163) every< 0.5 164  73 1 (0.44512195 0.55487805)
##                     326) water< 0.5 158  73 1 (0.46202532 0.53797468)
##                       652) keep< 0.5 150  72 1 (0.48000000 0.52000000)
##                        1304) ratio< 0.105 111  53 0 (0.52252252 0.47747748)
##                          2608) home>=0.5 7   2 0 (0.71428571 0.28571429) *
##                          2609) home< 0.5 104  51 0 (0.50961538 0.49038462)
##                            5218) company>=0.5 9   3 0 (0.66666667 0.33333333) *
##                            5219) company< 0.5 95  47 1 (0.49473684 0.50526316)
##                             10438) ratio>=0.09 69  33 0 (0.52173913 0.47826087) *
##                             10439) ratio< 0.09 26  11 1 (0.42307692 0.57692308) *
##                        1305) ratio>=0.105 39  14 1 (0.35897436 0.64102564) *
##                       653) keep>=0.5 8   1 1 (0.12500000 0.87500000) *
##                     327) water>=0.5 6   0 1 (0.00000000 1.00000000) *
##               41) flavor>=0.5 7   0 1 (0.00000000 1.00000000) *
##             21) children>=0.5 11   1 1 (0.09090909 0.90909091) *
##           11) many>=0.5 8   0 1 (0.00000000 1.00000000) *
##        3) easier>=0.5 13   1 1 (0.07692308 0.92307692) *
```

```r
## Pruning the tree:
printcp(cart.m2)
```

```
##
## Classification tree:
## rpart(formula = deal ~ ., data = train_c2, method = "class",
##     control = cart.ctrl)
##
## Variables actually used in tree construction:
##  [1] children    company     easier      every       featuring
##  [6] flavor      fun         home        keep        many
## [11] online      ratio       system      traditional water
## [16] way
##
## Root node error: 195/396 = 0.49242
##
## n= 396
##
##          CP nsplit rel error  xerror     xstd
## 1 0.0350427      0   1.00000 1.04103 0.051009
## 2 0.0307692      6   0.78974 1.06667 0.050960
## 3 0.0256410      7   0.75897 1.01026 0.051024
## 4 0.0085470      8   0.73333 0.89744 0.050680
## 5 0.0068376     11   0.70769 0.92308 0.050814
## 6 0.0000000     22   0.62051 0.93333 0.050858
```

```r
plotcp(cart.m2)
```




```r
## Extracting the least cpvalue
cart.m2$cptable
```

```
##            CP nsplit rel error    xerror       xstd
## 1 0.035042735      0 1.0000000 1.0410256 0.05100873
## 2 0.030769231      6 0.7897436 1.0666667 0.05095988
## 3 0.025641026      7 0.7589744 1.0102564 0.05102435
## 4 0.008547009      8 0.7333333 0.8974359 0.05067957
## 5 0.006837607     11 0.7076923 0.9230769 0.05081371
## 6 0.000000000     22 0.6205128 0.9333333 0.05085813
```

```r
cart.m2$cptable[,"xerror"]
```

```
##         1         2         3         4         5         6
## 1.0410256 1.0666667 1.0102564 0.8974359 0.9230769 0.9333333
```

```r
min(cart.m2$cptable[,"xerror"])
```

```
## [1] 0.8974359
```

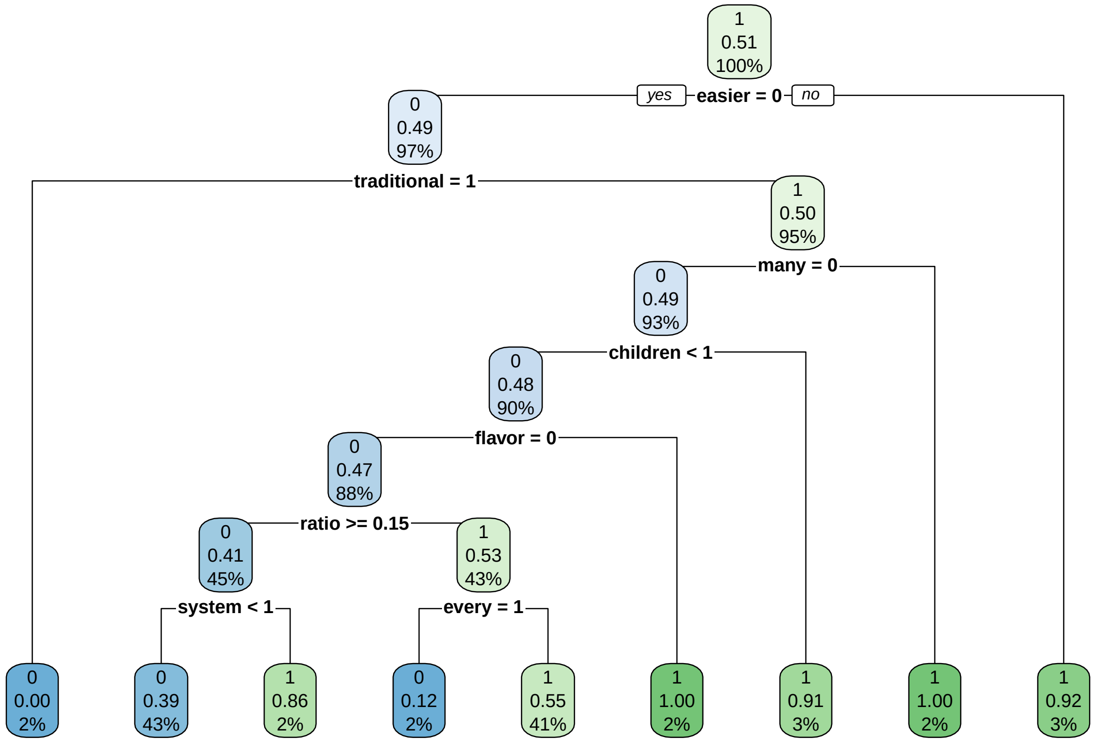
```r
## Our least CP value 0.8923077

## Best CP to prune the tree accordingly
cpbest2 = cart.m2$cptable[which.min(cart.m2$cptable[,"xerror"]), "CP"]
cpbest2
```

```
## [1] 0.008547009
```

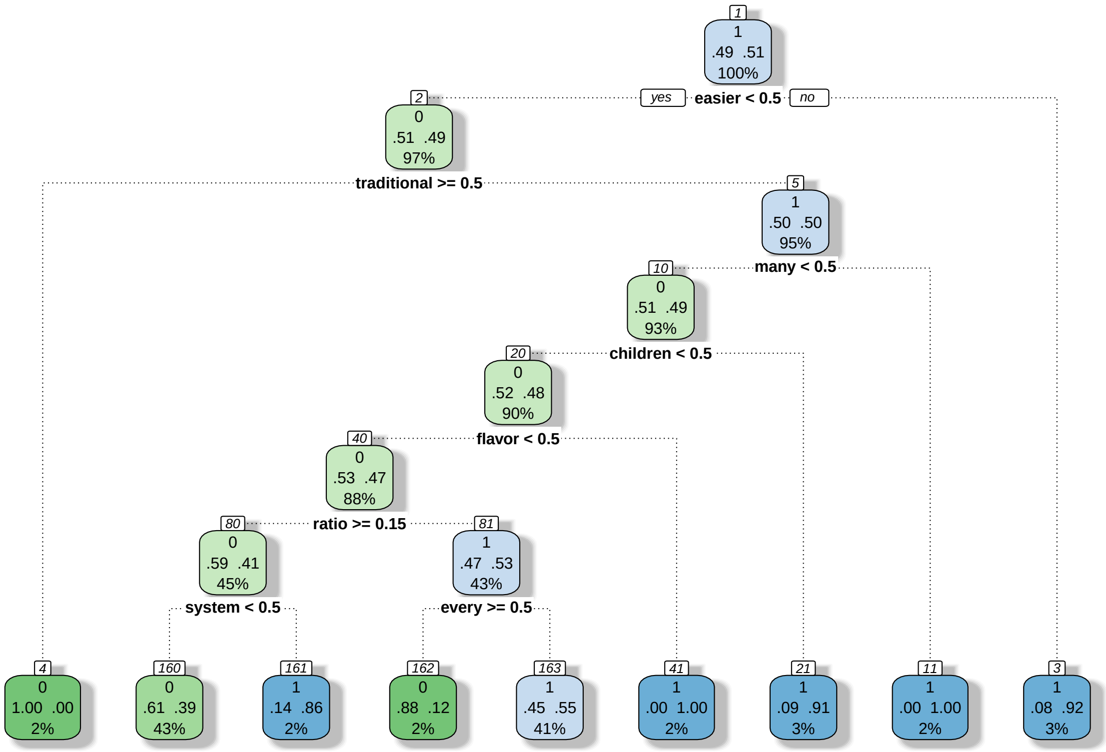
```r
## hence we need to prune the tree at CP = 0.00
## PRUNING THE TREE ACCORDINGLY

Pruntree2 = prune(tree = cart.m2, cp = cpbest2)
print(Pruntree2)
```

```
## n= 396
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##   1) root 396 195 1 (0.49242424 0.50757576)
##     2) easier< 0.5 383 189 0 (0.50652742 0.49347258)
##       4) traditional>=0.5 8   0 0 (1.00000000 0.00000000) *
##       5) traditional< 0.5 375 186 1 (0.49600000 0.50400000)
##        10) many< 0.5 367 181 0 (0.50681199 0.49318801)
##          20) children< 0.5 356 171 0 (0.51966292 0.48033708)
##            40) flavor< 0.5 349 164 0 (0.53008596 0.46991404)
##              80) ratio>=0.15 177  72 0 (0.59322034 0.40677966)
##               160) system< 0.5 170  66 0 (0.61176471 0.38823529) *
##               161) system>=0.5 7   1 1 (0.14285714 0.85714286) *
##              81) ratio< 0.15 172  80 1 (0.46511628 0.53488372)
##               162) every>=0.5 8   1 0 (0.87500000 0.12500000) *
##               163) every< 0.5 164  73 1 (0.44512195 0.55487805) *
##            41) flavor>=0.5 7   0 1 (0.00000000 1.00000000) *
##          21) children>=0.5 11   1 1 (0.09090909 0.90909091) *
##        11) many>=0.5 8   0 1 (0.00000000 1.00000000) *
##     3) easier>=0.5 13   1 1 (0.07692308 0.92307692) *
```

```r
rpart.plot(Pruntree2)
```




```r
library(rattle)

fancyRpartPlot(Pruntree2)
```



Rattle 2019−Dec−22 20:30:34 DELL

```r
## Summary

summary(Pruntree2)
```

```
## Call:
## rpart(formula = deal ~ ., data = train_c2, method = "class",
##     control = cart.ctrl)
##   n= 396
##
##            CP nsplit rel error    xerror       xstd
## 1 0.035042735      0 1.0000000 1.0410256 0.05100873
## 2 0.030769231      6 0.7897436 1.0666667 0.05095988
## 3 0.025641026      7 0.7589744 1.0102564 0.05102435
## 4 0.008547009      8 0.7333333 0.8974359 0.05067957
##
## Variable importance
##      easier        many traditional    children      flavor      system
##          10           8           8           8           8           6
##       ratio       every     peanut    shoulder     catalog        gift
##           6           6           3           3           2           2
##     pumpkin       three   overlooked     remotes       serve       nylon
##           2           2           2           2           2           2
##      anyone       relax        user  dishwasher        year        home
##           2           2           1           1           1           1
##        hold lightweight        onto       books        ones       young
##           1           1           1           1           1           1
```

```
##     mission sustainable
##           1           1
##
## Node number 1: 396 observations,    complexity param=0.03504274
##   predicted class=1  expected loss=0.4924242  P(node) =1
##     class counts:   195   201
##    probabilities: 0.492 0.508
##   left son=2 (383 obs) right son=3 (13 obs)
##   Primary splits:
##       easier      < 0.5       to the left,  improve=4.641029, (0 missing)
##       traditional < 0.5       to the right, improve=4.207123, (0 missing)
##       children    < 0.5       to the left,  improve=4.142045, (0 missing)
##       flavor      < 0.5       to the left,  improve=3.959700, (0 missing)
##       three       < 0.5       to the left,  improve=3.455831, (0 missing)
##   Surrogate splits:
##       user < 1.5       to the left,  agree=0.972, adj=0.154, (0 split)
##       onto < 0.5       to the left,  agree=0.970, adj=0.077, (0 split)
##       home < 1.5       to the left,  agree=0.970, adj=0.077, (0 split)
##
## Node number 2: 383 observations,    complexity param=0.03504274
##   predicted class=0  expected loss=0.4934726  P(node) =0.9671717
##     class counts:   194   189
##    probabilities: 0.507 0.493
##   left son=4 (8 obs) right son=5 (375 obs)
##   Primary splits:
##       traditional < 0.5       to the right, improve=3.979363, (0 missing)
##       children    < 0.5       to the left,  improve=3.912622, (0 missing)
##       flavor      < 0.5       to the left,  improve=3.658852, (0 missing)
##       three       < 0.5       to the left,  improve=3.127840, (0 missing)
##       materials   < 0.5       to the left,  improve=3.127840, (0 missing)
##   Surrogate splits:
##       nylon < 0.5       to the right, agree=0.984, adj=0.25, (0 split)
##
## Node number 3: 13 observations
##   predicted class=1  expected loss=0.07692308  P(node) =0.03282828
##     class counts:     1    12
##    probabilities: 0.077 0.923
##
## Node number 4: 8 observations
##   predicted class=0  expected loss=0  P(node) =0.02020202
##     class counts:     8     0
##    probabilities: 1.000 0.000
##
## Node number 5: 375 observations,    complexity param=0.03504274
##   predicted class=1  expected loss=0.496  P(node) =0.9469697
##     class counts:   186   189
##    probabilities: 0.496 0.504
##   left son=10 (367 obs) right son=11 (8 obs)
##   Primary splits:
##       many      < 0.5       to the left,  improve=4.022060, (0 missing)
##       children  < 0.5       to the left,  improve=3.719269, (0 missing)
##       flavor    < 0.5       to the left,  improve=3.509739, (0 missing)
##       three     < 0.5       to the left,  improve=3.000195, (0 missing)
##       materials < 0.5       to the left,  improve=3.000195, (0 missing)
```

```
##   Surrogate splits:
##       serve      < 0.5       to the left,  agree=0.984, adj=0.25, (0 split)
##       overlooked < 0.5       to the left,  agree=0.984, adj=0.25, (0 split)
##       remotes    < 0.5       to the left,  agree=0.984, adj=0.25, (0 split)
##
## Node number 10: 367 observations,    complexity param=0.03504274
##   predicted class=0  expected loss=0.493188  P(node) =0.9267677
##     class counts:   186   181
##    probabilities: 0.507 0.493
##   left son=20 (356 obs) right son=21 (11 obs)
##   Primary splits:
##       children < 0.5       to the left,  improve=3.923039, (0 missing)
##       flavor   < 0.5       to the left,  improve=3.665940, (0 missing)
##       three    < 0.5       to the left,  improve=3.133530, (0 missing)
##       started  < 0.5       to the left,  improve=3.133530, (0 missing)
##       well     < 0.5       to the left,  improve=2.889280, (0 missing)
##   Surrogate splits:
##       dishwasher < 0.5       to the left,  agree=0.975, adj=0.182, (0 split)
##       ones       < 0.5       to the left,  agree=0.973, adj=0.091, (0 split)
##       young      < 0.5       to the left,  agree=0.973, adj=0.091, (0 split)
##       books      < 0.5       to the left,  agree=0.973, adj=0.091, (0 split)
##
## Node number 11: 8 observations
##   predicted class=1  expected loss=0  P(node) =0.02020202
##     class counts:     0     8
##    probabilities: 0.000 1.000
##
## Node number 20: 356 observations,    complexity param=0.03504274
##   predicted class=0  expected loss=0.4803371  P(node) =0.8989899
##     class counts:   185   171
##    probabilities: 0.520 0.480
##   left son=40 (349 obs) right son=41 (7 obs)
##   Primary splits:
##       flavor  < 0.5       to the left,  improve=3.856524, (0 missing)
##       three   < 0.5       to the left,  improve=3.296148, (0 missing)
##       started < 0.5       to the left,  improve=3.296148, (0 missing)
##       well    < 0.5       to the left,  improve=3.082388, (0 missing)
##       designs < 0.5       to the left,  improve=2.549432, (0 missing)
##   Surrogate splits:
##       peanut  < 0.5       to the left,  agree=0.989, adj=0.429, (0 split)
##       gift    < 1.5       to the left,  agree=0.986, adj=0.286, (0 split)
##       three   < 0.5       to the left,  agree=0.986, adj=0.286, (0 split)
##       pumpkin < 0.5       to the left,  agree=0.986, adj=0.286, (0 split)
##       catalog < 0.5       to the left,  agree=0.986, adj=0.286, (0 split)
##
## Node number 21: 11 observations
##   predicted class=1  expected loss=0.09090909  P(node) =0.02777778
##     class counts:     1    10
##    probabilities: 0.091 0.909
##
## Node number 40: 349 observations,    complexity param=0.03504274
##   predicted class=0  expected loss=0.469914  P(node) =0.8813131
##     class counts:   185   164
##    probabilities: 0.530 0.470
```

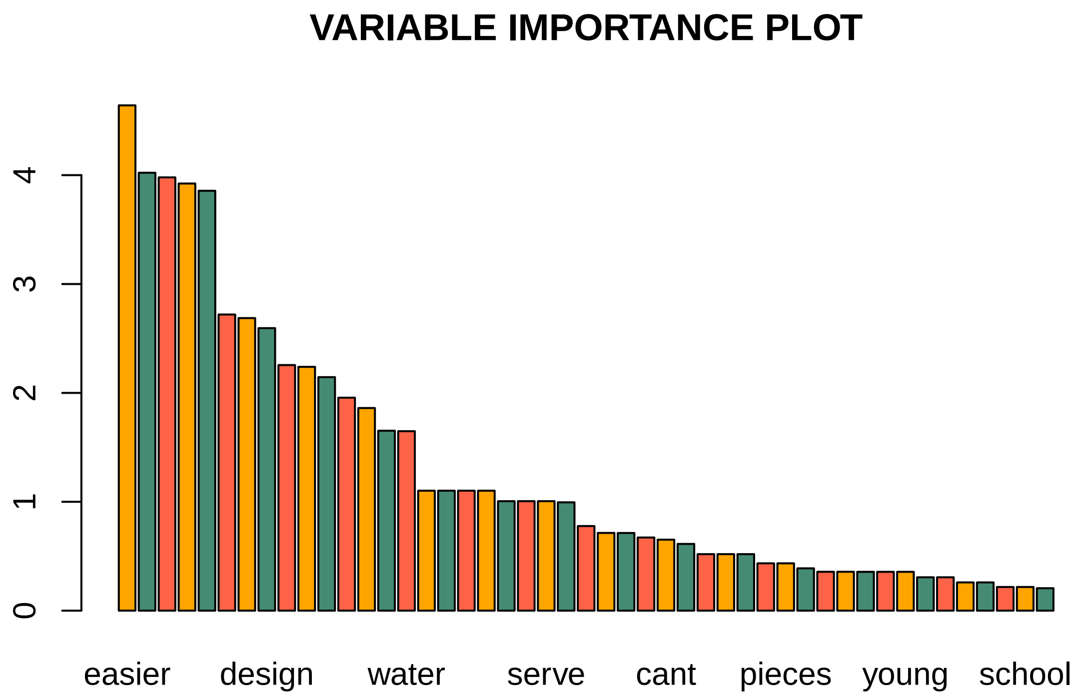
```
##   left son=80 (177 obs) right son=81 (172 obs)
##   Primary splits:
##       ratio   < 0.15      to the right, improve=2.863071, (0 missing)
##       designs < 0.5       to the left,  improve=2.687110, (0 missing)
##       well    < 0.5       to the left,  improve=2.687110, (0 missing)
##       dont    < 0.5       to the left,  improve=2.687110, (0 missing)
##       every   < 0.5       to the right, improve=2.378652, (0 missing)
##   Surrogate splits:
##       product < 0.5       to the left,  agree=0.530, adj=0.047, (0 split)
##       keep    < 0.5       to the left,  agree=0.530, adj=0.047, (0 split)
##       like    < 0.5       to the left,  agree=0.530, adj=0.047, (0 split)
##       fun     < 0.5       to the right, agree=0.527, adj=0.041, (0 split)
##       home    < 0.5       to the left,  agree=0.527, adj=0.041, (0 split)
##
## Node number 41: 7 observations
##   predicted class=1  expected loss=0  P(node) =0.01767677
##     class counts:     0     7
##    probabilities: 0.000 1.000
##
## Node number 80: 177 observations,    complexity param=0.02564103
##   predicted class=0  expected loss=0.4067797  P(node) =0.4469697
##     class counts:   105    72
##    probabilities: 0.593 0.407
##   left son=160 (170 obs) right son=161 (7 obs)
##   Primary splits:
##       system    < 0.5       to the left,  improve=2.956502, (0 missing)
##       design    < 0.5       to the left,  improve=1.822531, (0 missing)
##       easy      < 0.5       to the left,  improve=1.378351, (0 missing)
##       featuring < 0.5       to the right, improve=1.330534, (0 missing)
##       ratio     < 0.1500001 to the left,  improve=1.330534, (0 missing)
##   Surrogate splits:
##       shoulder    < 0.5       to the left,  agree=0.977, adj=0.429, (0 split)
##       anyone      < 0.5       to the left,  agree=0.972, adj=0.286, (0 split)
##       relax       < 0.5       to the left,  agree=0.972, adj=0.286, (0 split)
##       lightweight < 0.5       to the left,  agree=0.966, adj=0.143, (0 split)
##       hold        < 0.5       to the left,  agree=0.966, adj=0.143, (0 split)
##
## Node number 81: 172 observations,    complexity param=0.03076923
##   predicted class=1  expected loss=0.4651163  P(node) =0.4343434
##     class counts:    80    92
##    probabilities: 0.465 0.535
##   left son=162 (8 obs) right son=163 (164 obs)
##   Primary splits:
##       every     < 0.5       to the right, improve=2.819200, (0 missing)
##       keep      < 0.5       to the left,  improve=2.380305, (0 missing)
##       like      < 0.5       to the left,  improve=1.917743, (0 missing)
##       home      < 0.5       to the right, improve=1.856787, (0 missing)
##       available < 0.5       to the right, improve=1.685813, (0 missing)
##   Surrogate splits:
##       year        < 0.5       to the right, agree=0.965, adj=0.250, (0 split)
##       sustainable < 0.5       to the right, agree=0.959, adj=0.125, (0 split)
##       mission     < 0.5       to the right, agree=0.959, adj=0.125, (0 split)
##
## Node number 160: 170 observations
```

```
##   predicted class=0  expected loss=0.3882353  P(node) =0.4292929
##     class counts:   104    66
##    probabilities: 0.612 0.388
##
## Node number 161: 7 observations
##   predicted class=1  expected loss=0.1428571  P(node) =0.01767677
##     class counts:     1     6
##    probabilities: 0.143 0.857
##
## Node number 162: 8 observations
##   predicted class=0  expected loss=0.125  P(node) =0.02020202
##     class counts:     7     1
##    probabilities: 0.875 0.125
##
## Node number 163: 164 observations
##   predicted class=1  expected loss=0.445122  P(node) =0.4141414
##     class counts:    73    91
##    probabilities: 0.445 0.555
```

```
print(Pruntree2$variable.importance)
```

```
##      easier        many traditional    children      flavor      system
##   4.6410287   4.0220599   3.9793629   3.9230391   3.8565243   2.9565019
##       ratio       every      peanut    shoulder     catalog        gift
##   2.8630707   2.8192002   1.6527961   1.2670723   1.1018641   1.1018641
##     pumpkin       three  overlooked     remotes       serve       nylon
##   1.1018641   1.1018641   1.0055150   1.0055150   1.0055150   0.9948407
##      anyone       relax        user  dishwasher        year        home
##   0.8447148   0.8447148   0.7140044   0.7132798   0.7048001   0.4735225
##        hold lightweight        onto       books        ones       young
##   0.4223574   0.4223574   0.3570022   0.3566399   0.3566399   0.3566399
##     mission sustainable        keep        like     product         fun
##   0.3524000   0.3524000   0.1331661   0.1331661   0.1331661   0.1165203
```

```
barplot(sort(Pruntree$variable.importance, decreasing = TRUE),main = "VARIABLE IMPORTANCE PLOT", col = 
```




```
## The words have changed from the previous model.
## The words easier, traditional, children, flavor remain with ratio as an addition
```

```
#Scoring/Predicting the training and test dataset

train_c2$predict.class = predict(Pruntree2, data = train_c2, type="class")
train_c2$predict.score = predict(Pruntree2, data = train_c2)

#head(train_set)

test_c2$predict.class = predict(Pruntree2, newdata = test_c2, type="class")
test_c2$predict.score = predict(Pruntree2, newdata = test_c2)
```

Confusion matrix and Accuracy:

```
## Confusion matrix for CART model2

conf.tr2 = with(train_c2,table(deal,predict.class))
conf.tr2
```

```
##     predict.class
## deal   0   1
##    0 119  76
##    1  67 134
```

```
conf.te2 = with(test_c2,table(deal,predict.class))
conf.te2
```

```
##     predict.class
## deal  0  1
##    0 25 24
##    1 19 31
```

```r
## Accuracy

accuracy.tr2 = (conf.tr2[1,1]+conf.tr2[2,2])/(conf.tr2[1,1]+conf.tr2[1,2]+conf.tr2[2,1]+conf.tr2[2,2])

accuracy.te2 = (conf.te2[1,1]+conf.te2[2,2])/(conf.te2[1,1]+conf.te2[1,2]+conf.te2[2,1]+conf.te2[2,2])

accuracy.tr2
```

```
## [1] 0.6388889
```

```r
accuracy.te2
```

```
## [1] 0.5656566
```

```r
## CART model 2 has brought in accuracies (train = 65%, test = 48%) a bit lower than our CART model 1
## Even here, the model has predicted the non deals better than the deals.
```

2. Random Forest Model (New):

```r
# Data Partitioning
library(caTools)

# Setting seed
set.seed(123)

## Splitting the dataShark2 dataset to train and test

split4 = sample.split(dataShark2$deal, SplitRatio = 0.8)
train_rf2 = subset(dataShark2, split4 == TRUE)
test_rf2 = subset(dataShark2, split4 == FALSE)
```

Model Building:

```r
## Fitting Random Forest Model:

classifier2 = randomForest(x = train_rf2[-1563], y = train_rf2$deal, ntree = 5)

# Predicting the Test set results

y_pred2 = predict(classifier2, newdata = test_rf2[-1563])
# y_pred2
```

Confusion Matrix and Accuracy Evaluation:

```r
# Making the Confusion Matrix

cm_rf2 = with(test_rf2, table(deal, y_pred2))
cm_rf2
```

```
##     y_pred2
```

```
## deal  0  1
##    0 27 22
##    1 22 28
```

```r
## We observe 27 correct predictions on no deals out of 49
## Also the correct prediction of deals is 28 out of 50
## Though the accuracies are the same for both rf models, this is slightly better as we have more corre

# Accuracy:

accuracy.rf2 = (cm_rf2[1,1]+cm_rf2[2,2])/(cm_rf2[1,1]+cm_rf2[1,2]+cm_rf2[2,1]+cm_rf2[2,2])
accuracy.rf2
```

```
## [1] 0.5555556
```

```r
## The random forest model gives an accuracy of 55.5% which is an average performance
```

3. Logistic Regression Model (New):

```r
## Using the same data split for the random forest model:

# Data spliting
# library(caTools)

# Setting seed
# set.seed(123)

## Splitting to train and test
# split4 = sample.split(dataShark2$deal, SplitRatio = 0.8)
# train_rf2 = subset(dataShark2, split4 == TRUE)
# test_rf2 = subset(dataShark2, split4 == FALSE)
```

Building a Logit Model (New):

Logistic Regression Model 1:

```r
## Logistic Regression Model 1:

Logitn1 = glm(formula = deal~., data = train_rf2, family = binomial)
```

```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```r
predLogn1 = predict(Logitn1, newdata = test_rf2, type = "response")
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type
## == : prediction from a rank-deficient fit may be misleading
```

```r
cmLogitn1 = table(test_rf2$deal, predLogn1 > 0.3)
cmLogitn1
```

```
##
##     FALSE TRUE
##   0    27   22
##   1    27   23
```

```
## Accuracy:
acc.logn1 = (cmLogitn1[1,1]+cmLogitn1[2,2])/(cmLogitn1[1,1]+cmLogitn1[1,2]+cmLogitn1[2,1]+cmLogitn1[2,2
acc.logn1
```

```
## [1] 0.5050505
```

```
## The accuracy here is 50.5% which is lower compared to the RF model
## When comparing to our earlier Logit model 1, it is 1% higher.
## The number of no deals have been predicted better than last logit model which was 23 and this model
```

Logistic Regression Model 2 (New):

```
# Tweaking up the threshold to 0.9 to review any change in accuracy

## Logistic Regression Model 2:

Logitn2 = glm(formula = deal~., data = train_rf2, family = binomial)
```

```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
predLogn2 = predict(Logitn2, newdata = test_rf2, type = "response")
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type
## == : prediction from a rank-deficient fit may be misleading
```

```
cmLogitn2 = table(test_rf2$deal, predLogn2 > 0.9)
cmLogitn2
```

```
##
##     FALSE TRUE
##   0    27   22
##   1    27   23
```

```
## Accuracy:

acc.logn2 = (cmLogitn2[1,1]+cmLogitn2[2,2])/(cmLogitn2[1,1]+cmLogitn2[1,2]+cmLogitn2[2,1]+cmLogitn2[2,2
acc.logn2
```

```
## [1] 0.5050505
```

```
## We find there is no increase in the accuracy or the prediction scores even after the tweak
## Our similar model logit2 had a 4 point increase in accuracy to 53%
```

Interpretation of the Models:

The data provided was that of the pitches made to the VC sharks on the Shark Tank show. We were asked to use only the description part for text mining. The insights from the text (after cleanup) gave an idea about what the contestants wanted (Some words like company, product, design, etc were some of the highly targeted words or frequent words from the text). We then developed some models (namely CART, Random Forest and Logistic Regression) based on the text data with **deal** as the dependent variable. And have recorded the results. Later, we were asked to include another column **ratio** formulated by the division of **askedFor** from **valuation**. Then we build another set of models with ration included as one of the independent variables in predicting the acceptance or rejection of deals.

CART MODEL (OLD VS NEW): The first model built with the data was the CART model. The control parameter was the same for botht he models (with and without the ratio column). The first model, with both the train and test data, was efficientin predicting the number of non deals. But it failed to correctly predict the number of deals that were accepted. The accuracy of train and test data was 66% and 52%. The new model (with ratio) actually **under performed** when compared to the one without ratio. This resulted in a decrease in the number of correct predictions of both the deals and non deals rate. The accuracy fell low (train - 65% and test - 48%)

RANDOM FOREST MODEL (OLD VS NEW): The random forest model on both the cases have resulted in the same accuracy rate **(of 55.5%)**. The first model we framed, like the cart model, did a good job in predicting the true deal predictions (about 75% correct predictions). But when it came to identifying the deal rates, it failed. Same goes with the new model with the ratio factor, but this time, the model predicted a higher number of deal entries and a bit lower number of non deal entries. Maybe that is why, we find the accuracies remaining a constant.

LOGISTIC REGRESSION MODELS (OLD VS NEW): The logistic models were done using the glm function, with binomail family to predict the deals. We did two models for each of the with-and-without-ratio data. The first model with the threshold as 0.3, the accuracy was close to 50%. This time the deal rates were predicted better than the non deal rates. But the second model upon tweaking the threshold to 0.9 resulted in the non deal entries being correctly predicted with only one entry missing a dominating deal entry. Hence the accuracy rose to a 53%. The new model (1) remained the same nonetheless with an accuracy of 50.05%. It predicted the non deal rates better than the deal entries.

One thing to remember is that, we have arriced at results for these models, mostly just by doing text mining which is difficult to interpret when compared to entirely or partially numberical data. We have mostly used only the description to analyse the category, importance and deal chances of the pitch to predict. Moreover the data provided was limited (495 entries) for which training and testing would not be that effective. Hence, when we consider in an overall level, we cannot point blank say that including the ratio column has increased or decreased the accuracy of the models. More larger data would maybe result in better models