ISCTE-IUL

MIG | 2017/2018

Business Data Mining

Group 2:
Alessandro Fragalà N 84080
Anderson Acosta N 81799
Maria Sales N 83748

Introduction

The selected dataset for this project is "Restaurant and consumer data", which was extracted from the UCI Machine Learning Repository. This dataset contains information about consumers, restaurants, and ratings that consumers gave to restaurants, distributed among nine .csv files: five of them about restaurants, three have consumer information and the last one contains the ratings. The main purpose of this project is to analyze the given data and predict the missing ratings. This project follows the CRISP-DM methodology and makes use of the R language.

Library installation

In order to start working and make use of already existing R libraries, it's necessary to import them.

```
install.packages("rminer")
library(rminer)
library(data.table)
install.packages("ggmap")
library(ggmap)
library(caTools)
library(ggplot2)
```

1. Business Understanding

1.1 Setting business objectives

Restaurants of a given location are rated by consumers, being given a grade to the quality of their food, service and overall rating. The goal is to understand what influences the given ratings and predict the missing ones.

1.2 Assess the current situation

A study will be executed starting with a given varied range of characteristics of both restaurants and consumers, such as location, type of cuisine and price, among others. Throughout the project, some of the variables will be excluded or transformed, and others will be created based on existing information, in order to produce the best results.

1.3 Determine data mining goals

The challenge for this project is the analysis and transformation of the information, using several functions and libraries of the R language, and also recurring to external APIs, in order to obtain the necessary information to predict the missing ratings. To achieve this, supervised learning algorithms will be used and adapted to the proposed problem. These algorithms will be compared and the best performing ones will be selected.

1.4 Produce the project plan

This project will follow the CRISP-DM methodology, starting by understanding the provided files in order to identify the relations between them and merge in one data frame. The necessary treatments, such as the exclusion and creation of features, will then be identified. After that, several supervised learning models will be applied and their performance evaluated.

2. Data Understanding

The first step to start analyzing the provided data is to read all the .csv files and store them in separate datasets.

```
# Read datasets
# Restaurants
chefmozaccepts_df <- read.csv("chefmozaccepts.csv")
chefmozcuisine_df <- read.csv("chefmozcuisine.csv")
chefmozhours4_df <- read.csv("chefmozhours4.csv")
chefmozparking_df <- read.csv("chefmozparking.csv")
geoplaces2_df <- read.csv("geoplaces2.csv",encoding='latin-1')
# Consumers
usercuisine_df <- read.csv("usercuisine.csv")
userpayment_df <- read.csv("userpayment.csv")
userprofile_df <- read.csv("userprofile.csv")
# User-Item-Rating
rating_final_df <- read.csv("rating_final.csv")
```

After importing all the required data, it's necessary to analyze it by visualizing some important information and statistics. During this phase, we decided to iterate over the steps "Data gathering", "Descriptive analysis", "Exploratory analysis" and "Data quality verification" for each of the given .csv files.

Dataset 1: chefmozaccepts

This dataset contains information about the payment methods that restaurants accept. It contains the following features:

- placeID: corresponds to the restaurant id;
- Rpayment: corresponds to the type of payment that the restaurant accepts. This is a nominal feature with 12 levels, and no missing values.

```
# Columns
names(chefmozaccepts_df)
```

```
[1] "placeID"  "Rpayment"
```

```
# Number of attributes
ncol(chefmozaccepts_df)
```

```
[1] 2
```

```
# Number of instances
nrow(chefmozaccepts_df)
```

```
[1] 1314
```

```
# Summary
summary(chefmozaccepts_df, maxsum = 10)
```

```
   placeID                     Rpayment
 Min.   :132002   cash                :500
 1st Qu.:132580   MasterCard-Eurocard:194
 Median :132788   VISA                :172
 Mean   :133219   American_Express    :153
 3rd Qu.:133036   bank_debit_cards    :130
 Max.   :135110   Visa                : 83
                  Diners_Club         : 42
                  Discover            : 11
                  checks              : 10
                  (Other)             : 19
```

```
# Head
head(chefmozaccepts_df)
```

	placeID <int>	Rpayment <fctr>
1	135110	cash
2	135110	VISA
3	135110	MasterCard-Eurocard
4	135110	American_Express
5	135110	bank_debit_cards
6	135109	cash

6 rows

```
# Number of nulls
sum(chefmozaccepts_df == "?")
```

```
[1] 0
```

```
# Number of nulls per column
colSums(chefmozaccepts_df == "?")
```

```
 placeID Rpayment
       0        0
```

```
# Number of levels of Rpayment
sprintf("Rpayment has %d levels.", nlevels(chefmozaccepts_df$Rpayment))
```

```
[1] "Rpayment has 12 levels."
```

```
# Number of restaurants
sprintf("There are %d unique placeID's.", length(unique(chefmozaccepts_df$placeID)))
```

```
[1] "There are 615 unique placeID's."
```

From this analysis, we conclude that this dataset has no missing values, but a treatment to Rpayment is necessary to reduce the number of levels of this feature.

Dataset 2: chefmozcuisine

This dataset contains information about the type of cuisine of the restaurants. It contains the following features:

- placeID: corresponds to the restaurant id;
- Rcuisine: corresponds to the type of cuisine of the restaurant. This is a nominal feature with 59 levels.

```
# Columns
names(chefmozcuisine_df)
```

```
[1] "placeID"  "Rcuisine"
```

```r
# Number of attributes
ncol(chefmozcuisine_df)
```

```
[1] 2
```

```r
# Number of instances
nrow(chefmozcuisine_df)
```

```
[1] 916
```

```r
# Summary
summary(chefmozcuisine_df, maxsum = 10)
```

```
   placeID                  Rcuisine
 Min.   :132001   Mexican          :239
 1st Qu.:132323   International    : 62
 Median :132630   American         : 59
 Mean   :132897   Dutch-Belgian    : 55
 3rd Qu.:132907   Italian          : 42
 Max.   :135110   Greek            : 33
                  Bar              : 32
                  French           : 31
                  Cafe-Coffee_Shop: 27
                  (Other)          :336
```

```r
# Head
head(chefmozcuisine_df)
```

	placeID <int>	Rcuisine <fctr>
1	135110	Spanish
2	135109	Italian
3	135107	Latin_American
4	135106	Mexican
5	135105	Fast_Food
6	135104	Mexican

6 rows

```r
# Number of nulls
sum(chefmozcuisine_df == "?")
```

```
[1] 0
```

```r
# Number of nulls per column
colSums(chefmozcuisine_df == "?")
```

```
 placeID Rcuisine
       0        0
```

```
# Number of levels of Rcuisine
sprintf("Rcuisine has %d levels.", nlevels(chefmozcuisine_df$Rcuisine))
```

```
[1] "Rcuisine has 59 levels."
```

```
# Number of restaurants
sprintf("There are %d unique placeID's.", length(unique(chefmozcuisine_df$placeID)))
```

```
[1] "There are 769 unique placeID's."
```

From this analysis, we conclude that this dataset has no missing values, but a treatment to Rcuisine is necessary to reduce the number of levels of this feature.

Dataset 3: chefmozhours4

This dataset contains information about the hours and days that each restaurant is open/closed. It contains the following features:

- placeID: corresponds to the restaurant id;
- hours: corresponds to the hours that the restaurant is open/closed. This is a nominal feature with 273 levels;
- days: corresponds to the day of the week that the restaurant is open/closed. This is a nominal feature with 3 levels.

```
# Columns
names(chefmozhours4_df)
```

```
[1] "placeID" "hours"   "days"
```

```
# Number of attributes
ncol(chefmozhours4_df)
```

```
[1] 3
```

```
# Number of instances
nrow(chefmozhours4_df)
```

```
[1] 2339
```

```
# Summary
summary(chefmozhours4_df, maxsum = 10)
```

```
   placeID               hours                         days
 Min.   :132012   00:00-23:30;: 681   Mon;Tue;Wed;Thu;Fri;:793
 1st Qu.:132574   00:00-00:00;: 100   Sat;                 :783
 Median :132785   17:00-22:00;:  56   Sun;                 :763
 Mean   :133082   14:00-23:30;:  32
 3rd Qu.:132984   09:00-23:30;:  31
 Max.   :135111   11:00-21:00;:  31
                  08:00-23:30;:  30
                  11:00-22:00;:  30
                  12:00-22:00;:  29
                  (Other)     :1319
```

```
# Head
head(chefmozhours4_df)
```

	placeID <int>	hours <fctr>	days <fctr>
1	135111	00:00-23:30;	Mon;Tue;Wed;Thu;Fri;
2	135111	00:00-23:30;	Sat;
3	135111	00:00-23:30;	Sun;
4	135110	08:00-19:00;	Mon;Tue;Wed;Thu;Fri;
5	135110	00:00-00:00;	Sat;
6	135110	00:00-00:00;	Sun;

6 rows

```
# Number of nulls
sum(chefmozhours4_df == "?")
```

```
[1] 0
```

```
# Number of nulls per column
colSums(chefmozhours4_df == "?")
```

```
placeID   hours    days
      0       0       0
```

```
# Number of levels of hours
sprintf("hours has %d levels.", nlevels(chefmozhours4_df$hours))
```

```
[1] "hours has 273 levels."
```

```
# Number of levels of days
sprintf("days has %d levels.", nlevels(chefmozhours4_df$days))
```

```
[1] "days has 3 levels."
```

```
# Number of restaurants
sprintf("There are %d unique placeID's.", length(unique(chefmozhours4_df$placeID)))
```

```
[1] "There are 694 unique placeID's."
```

From this analysis, we conclude that this dataset has no missing values, but a treatment to hours is necessary to reduce the number of levels of this feature.

Dataset 4: chefmozparking

This dataset contains information about the parking that each restaurant offers. It contains the following features:

- placeID: corresponds to the restaurant id;
- parking_lot: corresponds to the type of parking. This is a nominal feature with 7 levels.

```
# Columns
names(chefmozparking_df)
```

```
[1] "placeID"     "parking_lot"
```

```
# Number of attributes
ncol(chefmozparking_df)
```

```
[1] 2
```

```
# Number of instances
nrow(chefmozparking_df)
```

```
[1] 702
```

```
# Summary
summary(chefmozparking_df, maxsum = 10)
```

```
    placeID                 parking_lot
 Min.   :132012   fee               : 22
 1st Qu.:132649   none              :348
 Median :132826   public            :102
 Mean   :133181   street            : 32
 3rd Qu.:133009   valet parking     : 21
 Max.   :135111   validated parking:  3
                  yes               :174
```

```
# Head
head(chefmozparking_df)
```

	placeID <int>	parking_lot <fctr>
1	135111	public
2	135110	none
3	135109	none
4	135108	none
5	135107	none
6	135106	none

6 rows

```
# Number of nulls
sum(chefmozparking_df == "?")
```

```
[1] 0
```

```
# Number of nulls per column
colSums(chefmozparking_df == "?")
```

```
    placeID parking_lot
          0           0
```

```
# Number of levels of parking_lot
sprintf("parking_lot has %d levels.", nlevels(chefmozparking_df$parking_lot))
```

```
[1] "parking_lot has 7 levels."
```

```
# Number of restaurants
sprintf("There are %d unique placeID's.", length(unique(chefmozparking_df$placeID)))
```

```
[1] "There are 675 unique placeID's."
```

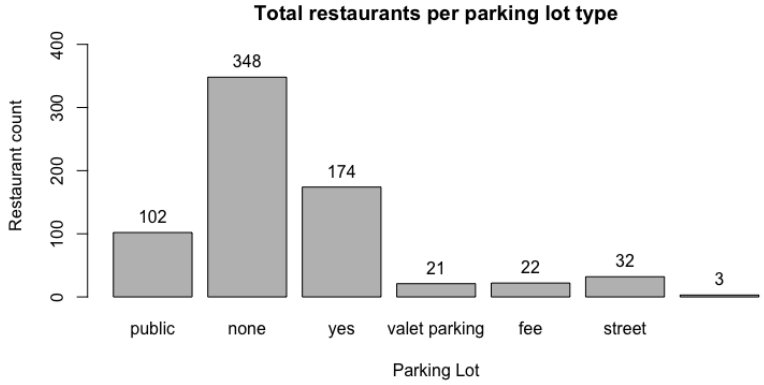
From this analysis, we conclude that this dataset has no missing values, and no treatment is necessary. If we decide to use the feature parking_lot and the data modeling takes too much time, we can consider reducing the levels of this column.

Feature 4.1: parking_lot

```
# Auxiliar function: Draw barplot
drawBarplot <- function(dataframe, type, title, xlab, ylab, feature) {
  count <- c()
  for(s in type) {
    value <- length(dataframe[which(dataframe[,feature] == s),feature])
    count <- c(count, value)
  }
  bb <- barplot(count, main = title, xlab = xlab, ylab = ylab, names.arg = type, ylim = c(0, max(c
ount) + max(count)*0.2)) # las = 2
  text(x = bb, y = count, labels = count, pos = 3)
}
```

A graphical visualization with a barplot shows the distribution of this column.

```
parking_lot <- unique(chefmozparking_df$parking_lot)
title <- "Total restaurants per parking lot type"
xlab <- "Parking Lot"
ylab <- "Restaurant count"
feature <- "parking_lot"
drawBarplot(chefmozparking_df, parking_lot, title, xlab, ylab, feature)
```



Dataset 5: geoplaces2

This dataset contains generic information about each restaurant. It contains the following features:

- placeID: corresponds to the restaurant id;

- latitude and longitude: correspond to the geospatial coordinates of the restaurant. These are both numeric features;
- the_geom_meter: geospatial variable. This is a nominal feature with 130 levels, and no missing values;
- name: name of the restaurant. This is a nominal feature with no missing values;
- address: address of the restaurant. This is a nominal feature with 100 levels and 27 missing values;
- city: city of the restaurant. This is a nominal feature with 17 levels and 18 missing values;
- state: state of the restaurant. This is a nominal feature with 13 levels and 18 missing values;
- country: country of the restaurant. This is a nominal feature with 3 levels and 28 missing values;
- fax: fax of the restaurant. This is a numeric feature with 130 missing values;
- zip: zip code of the restaurant. This is a numeric feature with 35 levels and 74 missing values;
- alcohol: indicates the type of bar service that the restaurant provides. This is a nominal feature with 3 levels, and no missing values;
- smoking_area: indicates the type of smoking area that the restaurant provides. This is a nominal feature with 5 levels, and no missing values;
- dress_code: dress code allowed in the restaurant. This is a nominal feature with 3 levels, and no missing values;
- accessibility: indicates the accessibility that the restaurant provides for disabled people. This is a nominal feature with 3 levels, and no missing values;
- price: budget range of the restaurant. This is a nominal feature with 3 levels, and no missing values;
- url: url of the restaurant. This is a nominal feature with 116 missing values;
- Rambience: type of ambience of the restaurant. This is a nominal feature with 2 levels, and no missing values;
- franchise: indicates if the restaurant is a franchise or not. This is a nominal feature with 2 levels, and no missing values;
- area: indicates the type of the area of the restaurant. This is a nominal feature with 2 levels, and no missing values;
- other_services: indicates if the restaurant offers any additional service or not. This is a nominal feature with 3 levels, and no missing values.

```
# Columns
names(geoplaces2_df)
```

```
 [1] "placeID"        "latitude"       "longitude"      "the_geom_meter" "name"           "addre
ss"        "city"
 [8] "state"          "country"        "fax"            "zip"            "alcohol"        "smok
ing_area"   "dress_code"
[15] "accessibility"  "price"          "url"            "Rambience"      "franchise"      "area"
"other_services"
```

```
# Number of attributes
ncol(geoplaces2_df)
```

```
[1] 21
```

```
# Number of instances
nrow(geoplaces2_df)
```

```
[1] 130
```

```
# Summary
summary(geoplaces2_df)
```

```
    placeID          latitude       longitude
the_geom_meter
 Min.   :132560   Min.   :18.86   Min.   :-101.03   0101000020957F000000DD3546816E5AC119D4BD17FD5
44A41:  1
 1st Qu.:132831   1st Qu.:22.14   1st Qu.:-100.99   0101000020957F000003B195E25F8457C1C535BD04614B
4941:  1
 Median :134994   Median :22.15   Median :-100.96   0101000020957F000004457BB7AA8657C15F10835CD944
4941:  1
 Mean   :134013   Mean   :21.86   Mean   :-100.34   0101000020957F000005810F19B84858C136805B2745A
74B41:  1
 3rd Qu.:135051   3rd Qu.:22.16   3rd Qu.: -99.22   0101000020957F00000B6735CA004858C108FD525CB2A4
4B41:  1
 Max.   :135109   Max.   :23.76   Max.   : -99.13   0101000020957F00000F14BF6B2C8657C1963CCB8E5C4
64941:  1
                                                    (Other)
:124
                      name                     address                city                 sta
te      country   fax
 Gorditas Dona Tota       :  2   ?                  :27   San Luis Potosi:64   SLP            :5
0   ?     :28   ?:130
 Abondance Restaurante Bar:  1   Ricardo B. Anaya   : 3   ?              :18   Morelos        :1
9   mexico:13
 Arrachela Grill          :  1   Av. V. Carranza    : 2   Cuernavaca     :15   ?              :1
8   Mexico:89
 Cabana Huasteca          :  1   venustiano carranza: 2   victoria       :10   San Luis Potosi:14

 cafe ambar               :  1   16 de Septiembre   : 1   san luis potosi: 5   tamaulipas     :
9
 Cafe Chaires             :  1   1a. de Lozada 1    : 1   Jiutepec       : 4   Tamaulipas     :
7
 (Other)                  :123   (Other)            :94   (Other)        :14   (Other)        :
13
      zip                  alcohol          smoking_area    dress_code           accessibility
price
 ?      :74   Full_Bar         : 9   none         :70    casual  : 10   completely      :45    h
igh  :25
 78000  :13   No_Alcohol_Served:87   not permitted:25    formal  :  2   no_accessibility:76    lo
w   :45
 78250  : 3   Wine-Beer        :34   only at bar  : 2    informal:118   partially       : 9    me
dium:60
 78269  : 3                         permitted    : 9

 62290  : 2                         section      :24

 78210  : 2

 (Other):33

                   url         Rambience   franchise     area      other_services
 ?                   :116   familiar:121   f:108     closed:115   Internet:  4
 lacantinaslp.com    :  2   quiet   :  9   t: 22     open  : 15   none    :119
 carlosandcharlies.com:  1                                       variety :  7
 chilis.com.mx       :  1
 eloceanodorado.com  :  1
 kikucuernavaca.com.mx:  1
 (Other)             :  8
```

```r
# Head
head(geoplaces2_df)
```

	placeID <int>	latitude <dbl>	longitude <dbl>	the_geom_meter <fctr>
1	134999	18.91542	-99.18487	0101000020957F000088568DE356715AC138C0A525FC464A41
2	132825	22.14739	-100.98309	0101000020957F00001AD016568C4858C1243261274BA54B41
3	135106	22.14971	-100.97609	0101000020957F0000649D6F21634858C119AE9BF528A34B41
4	132667	23.75270	-99.16336	0101000020957F00005D67BCDDED8157C1222A2DC8D84D4941
5	132613	23.75290	-99.16508	0101000020957F00008EBA2D06DC8157C194E03B7B504E4941
6	135040	22.13562	-100.96971	0101000020957F00001B552189B84A58C15A2AAEFD2CA24B41

6 rows | 1-5 of 21 columns

```
# Number of nulls
sum(geoplaces2_df == "?")
```

```
[1] 411
```

```
# Number of nulls per column
colSums(geoplaces2_df == "?")
```

```
       placeID       latitude      longitude the_geom_meter           name        address
city          state
             0              0              0              0              0             27
18             18
       country            fax            zip        alcohol   smoking_area     dress_code  acce
ssibility          price
            28            130             74              0              0              0
0              0
           url      Rambience      franchise           area other_services
           116              0              0              0              0
```

```
# Number of restaurants
sprintf("There are %d unique placeID's.", length(unique(geoplaces2_df$placeID)))
```

```
[1] "There are 130 unique placeID's."
```

An interesting analysis is to try to understand if any of the given restaurant features affect the ratings. In order to make this analysis easier, we elaborated connected scatterplots which show the relationship between each restaurant feature and the ratings.

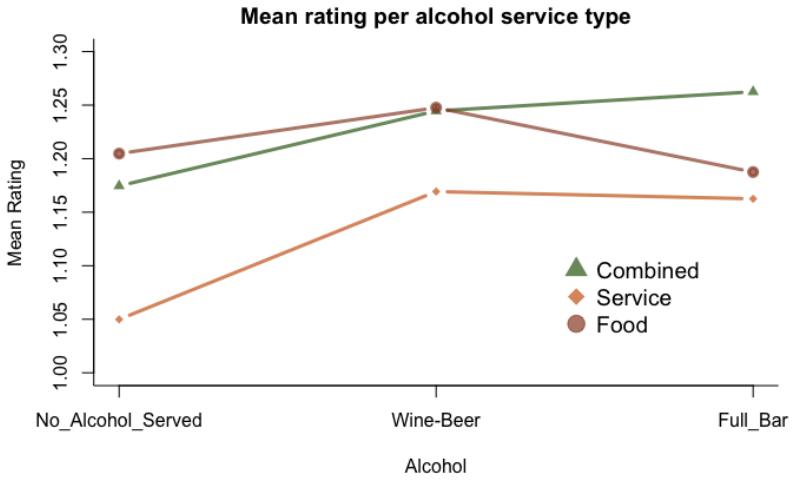
```
# Merge ratings and restaurants
ratings_restaurants <- merge(rating_final_df, geoplaces2_df, by = "placeID")
# Define auxiliar functions:
# Auxiliar function: Draw connected scatterplot
drawConnectedScatterplot <- function(total, rating, service, food, title, xlab, ylim, at, lab) {
  # Make a basic graph
  plot(rating~total, type="b", main=title, bty="l", xlab=xlab, ylab="Mean Rating", col=rgb(0.2,0.4
,0.1,0.7), lwd=3, pch=17, ylim=ylim, xaxt='n')
  lines(service~total , col=rgb(0.8,0.4,0.1,0.7) , lwd=3 , pch=18 , type="b" )
  lines(food~total , col=rgb(0.6,0.3,0.2,0.7) , lwd=3 , pch=19 , type="b" )
  axis(1, at = at, lab = lab)

  # Add a legend
  legend("bottomright", legend = c("Combined", "Service", "Food"), col = c(rgb(0.2,0.4,0.1,0.7), r
gb(0.8,0.4,0.1,0.7), rgb(0.6,0.3,0.2,0.7)), pch = c(17,18,19), bty = "n", pt.cex = 2, cex = 1.2,
text.col = "black", horiz = F , inset = c(0.1, 0.1))
}
# Auxiliar function: Calculate mean rating
calculateMeanRating <- function(feature, value, rating) {
  mean_rating <- mean(ratings_restaurants[which(ratings_restaurants[, feature] == value), rating])
}
```

Feature 5.1: alcohol

From this graph, we can conclude that, in general, the rating of the restaurant increases when some type of alcohol is provided, but the food rating has a slight decrease then the restaurant offers a full bar service.

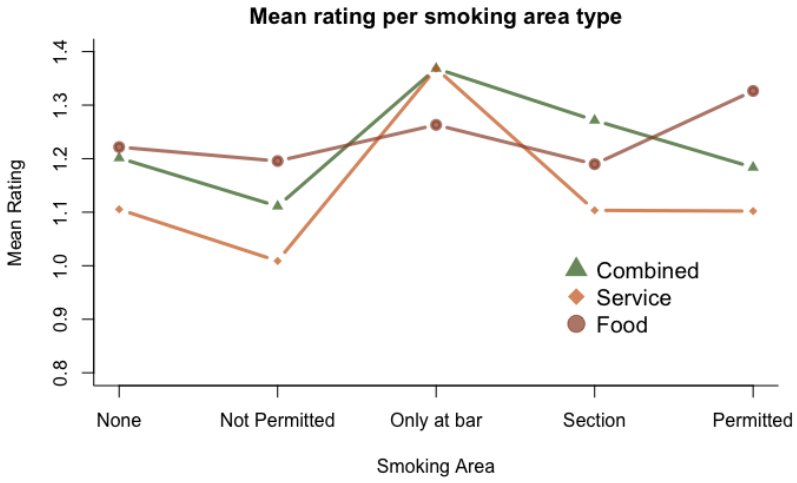
```
# Calculate mean rating per each type of alcohol for general rating
mean_no_alcohol_rating <- calculateMeanRating("alcohol", "No_Alcohol_Served", "rating")
mean_wine_rating <- calculateMeanRating("alcohol", "Wine-Beer", "rating")
mean_full_bar_rating <- calculateMeanRating("alcohol", "Full_Bar", "rating")
# Calculate mean rating per each type of alcohol for service rating
mean_no_alcohol_service <- calculateMeanRating("alcohol", "No_Alcohol_Served", "service_rating")
mean_wine_service <- calculateMeanRating("alcohol", "Wine-Beer", "service_rating")
mean_full_bar_service <- calculateMeanRating("alcohol", "Full_Bar", "service_rating")
# Calculate mean rating per each type of alcohol for food rating
mean_no_alcohol_food <- calculateMeanRating("alcohol", "No_Alcohol_Served", "food_rating")
mean_wine_food <- calculateMeanRating("alcohol", "Wine-Beer", "food_rating")
mean_full_bar_food <- calculateMeanRating("alcohol", "Full_Bar", "food_rating")
# Define variables
total <- c(0, 1, 2)
rating <- c(mean_no_alcohol_rating, mean_wine_rating, mean_full_bar_rating)
service <- c(mean_no_alcohol_service, mean_wine_service, mean_full_bar_service)
food <- c(mean_no_alcohol_food, mean_wine_food, mean_full_bar_food)
title <- "Mean rating per alcohol service type"
xlab <- "Alcohol"
ylim <-c (1.0, 1.3)
lab <- c("No_Alcohol_Served", "Wine-Beer", "Full_Bar")
# Draw graph
drawConnectedScatterplot(total, rating, service, food, title, xlab, ylim, 0:2, lab)
```



Feature 5.2: smoking_area

From this graph, we can observe that food rating reaches the top mean rating when smoking is permitted, but service and combined ratings have the highest value when only at bar smoking is allowed. “Not permitted” smoking option has the lowest ratings overall.

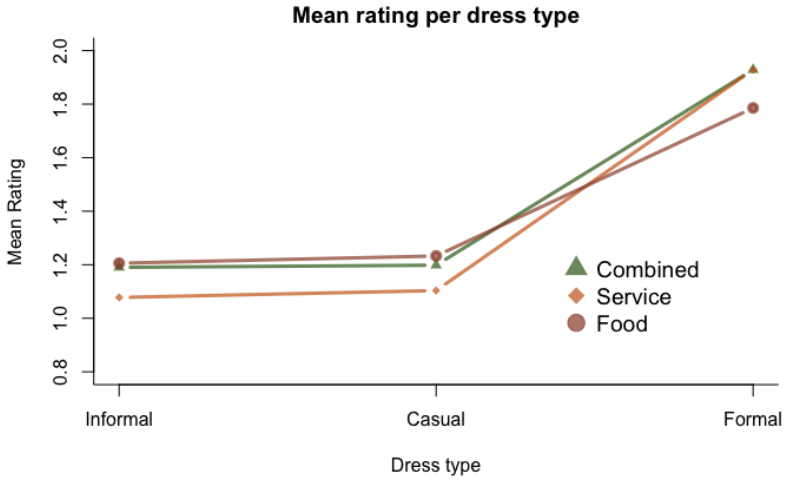
```
# Calculate mean rating per each type of smoking_area for general rating
mean_none_rating <- calculateMeanRating("smoking_area", "none", "rating")
mean_not_rating <- calculateMeanRating("smoking_area", "not permitted", "rating")
mean_bar_rating <- calculateMeanRating("smoking_area", "only at bar", "rating")
mean_section_rating <- calculateMeanRating("smoking_area", "section", "rating")
mean_permitted_rating <- calculateMeanRating("smoking_area", "permitted", "rating")
# Calculate mean rating per each type of smoking_area for service rating
mean_none_service <- calculateMeanRating("smoking_area", "none", "service_rating")
mean_not_service <- calculateMeanRating("smoking_area", "not permitted", "service_rating")
mean_bar_service <- calculateMeanRating("smoking_area", "only at bar", "service_rating")
mean_section_service <- calculateMeanRating("smoking_area", "section", "service_rating")
mean_permitted_service <- calculateMeanRating("smoking_area", "permitted", "service_rating")
# Calculate mean rating per each type of smoking_area for food rating
mean_none_food <- calculateMeanRating("smoking_area", "none", "food_rating")
mean_not_food <- calculateMeanRating("smoking_area", "not permitted", "food_rating")
mean_bar_food <- calculateMeanRating("smoking_area", "only at bar", "food_rating")
mean_section_food <- calculateMeanRating("smoking_area", "section", "food_rating")
mean_permitted_food <- calculateMeanRating("smoking_area", "permitted", "food_rating")
# Define variables
total <- c(0, 1, 2, 3, 4)
rating <- c(mean_none_rating, mean_not_rating, mean_bar_rating, mean_section_rating, mean_permitte
d_rating)
service <- c(mean_none_service, mean_not_service, mean_bar_service, mean_section_service, mean_per
mitted_service)
food <- c(mean_none_food, mean_not_food, mean_bar_food, mean_section_food, mean_permitted_food)
title <- "Mean rating per smoking area type"
xlab <- "Smoking Area"
ylim <-c (0.8, 1.4)
lab <- c("None", "Not Permitted", "Only at bar", "Section", "Permitted")
# Draw graph
drawConnectedScatterplot(total, rating, service, food, title, xlab, ylim, 0:4, lab)
```



Feature 5.3: dress_code

This graph shows a clear relation between formal dress code and high ratings, while casual and informal dress codes have similar, but medium, ratings.

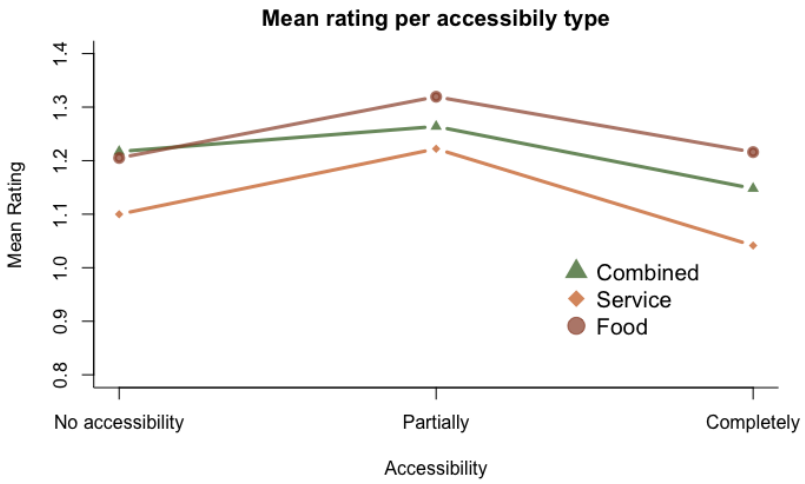
```
# Calculate mean rating per each type of dress_code for general rating
mean_informal_rating <- calculateMeanRating("dress_code", "informal", "rating")
mean_casual_rating <- calculateMeanRating("dress_code", "casual", "rating")
mean_formal_rating <- calculateMeanRating("dress_code", "formal", "rating")
# Calculate mean rating per each type of dress_code for service rating
mean_informal_service <- calculateMeanRating("dress_code", "informal", "service_rating")
mean_casual_service <- calculateMeanRating("dress_code", "casual", "service_rating")
mean_formal_service <- calculateMeanRating("dress_code", "formal", "service_rating")
# Calculate mean rating per each type of dress_code for food rating
mean_informal_food <- calculateMeanRating("dress_code", "informal", "food_rating")
mean_casual_food <- calculateMeanRating("dress_code", "casual", "food_rating")
mean_formal_food <- calculateMeanRating("dress_code", "formal", "food_rating")
# Define variables
total <- c(0, 1, 2)
rating <- c(mean_informal_rating, mean_casual_rating, mean_formal_rating)
service <- c(mean_informal_service, mean_casual_service, mean_formal_service)
food <- c(mean_informal_food, mean_casual_food, mean_formal_food)
title <- "Mean rating per dress type"
xlab <- "Dress type"
ylim <-c (0.8, 2.0)
lab <- c("Informal", "Casual", "Formal")
# Draw graph
drawConnectedScatterplot(total, rating, service, food, title, xlab, ylim, 0:2, lab)
```



Feature 5.4: accessibility

Contrary to common sense, complete accessibility has lower classification comparing to partial accessibility.

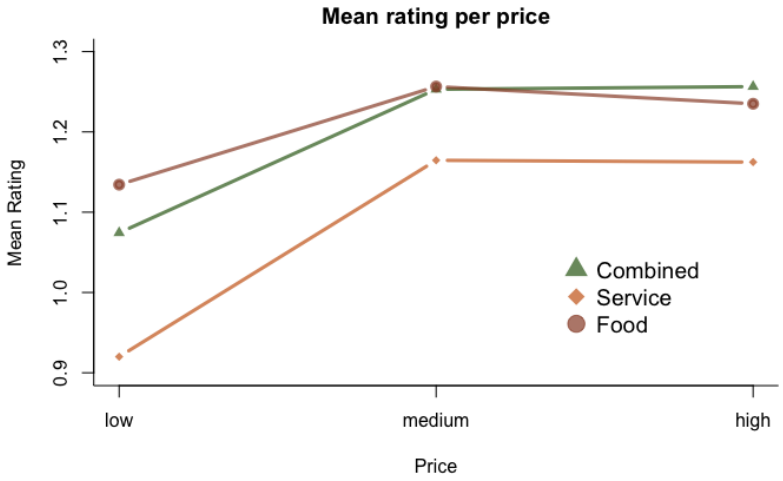
```
# Calculate mean rating per each type of accessibility for general rating
mean_no_accessibility_rating <- calculateMeanRating("accessibility", "no_accessibility", "rating")
mean_partially_rating <- calculateMeanRating("accessibility", "partially", "rating")
mean_completely_rating <- calculateMeanRating("accessibility", "completely", "rating")
# Calculate mean rating per each type of accessibility for service rating
mean_no_accessibility_service <- calculateMeanRating("accessibility", "no_accessibility", "service
_rating")
mean_partially_service <- calculateMeanRating("accessibility", "partially", "service_rating")
mean_completely_service <- calculateMeanRating("accessibility", "completely", "service_rating")
# Calculate mean rating per each type of accessibility for food rating
mean_no_accessibility_food <- calculateMeanRating("accessibility", "no_accessibility", "food_ratin
g")
mean_partially_food <- calculateMeanRating("accessibility", "partially", "food_rating")
mean_completely_food <- calculateMeanRating("accessibility", "completely", "food_rating")
# Define variables
total <- c(0, 1, 2)
rating <- c(mean_no_accessibility_rating, mean_partially_rating, mean_completely_rating)
service <- c(mean_no_accessibility_service, mean_partially_service, mean_completely_service)
food <- c(mean_no_accessibility_food, mean_partially_food, mean_completely_food)
title <- "Mean rating per accessibily type"
xlab <- "Accessibility"
ylim <-c (0.8, 1.4)
lab <- c("No accessibility", "Partially", "Completely")
# Draw graph
drawConnectedScatterplot(total, rating, service, food, title, xlab, ylim, 0:2, lab)
```



Feature 5.5: price

From the following chart, we can conclude that medium and high price restaurants have better ratings than lower price restaurants, but there's no significant difference in ratings between medium and high budget restaurants.

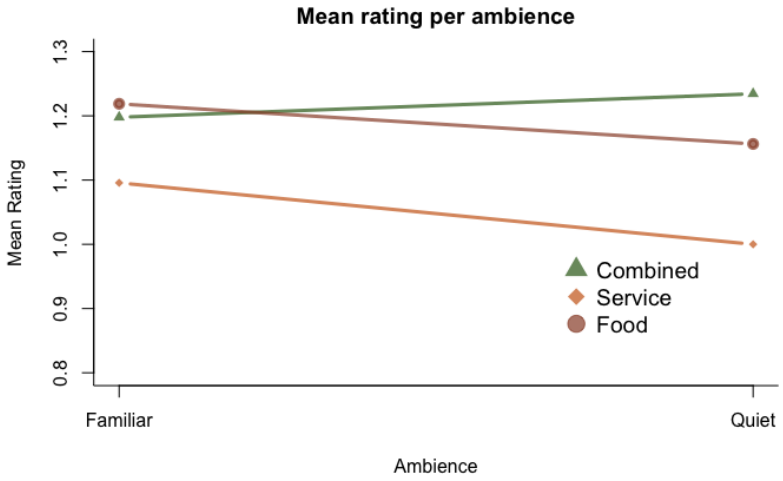
```
# Calculate mean rating per each type of price for general rating
mean_low_rating <- calculateMeanRating("price", "low", "rating")
mean_medium_rating <- calculateMeanRating("price", "medium", "rating")
mean_high_rating <- calculateMeanRating("price", "high", "rating")
# Calculate mean rating per each type of price for service rating
mean_low_service <- calculateMeanRating("price", "low", "service_rating")
mean_medium_service <- calculateMeanRating("price", "medium", "service_rating")
mean_high_service <- calculateMeanRating("price", "high", "service_rating")
# Calculate mean rating per each type of price for food rating
mean_low_food <- calculateMeanRating("price", "low", "food_rating")
mean_medium_food <- calculateMeanRating("price", "medium", "food_rating")
mean_high_food <- calculateMeanRating("price", "high", "food_rating")
# Define variables
total <- c(0, 1, 2)
rating <- c(mean_low_rating, mean_medium_rating, mean_high_rating)
service <- c(mean_low_service, mean_medium_service, mean_high_service)
food <- c(mean_low_food, mean_medium_food, mean_high_food)
title <- "Mean rating per price"
xlab <- "Price"
ylim <-c (0.9, 1.3)
lab <- c("low", "medium", "high")
# Draw graph
drawConnectedScatterplot(total, rating, service, food, title, xlab, ylim, 0:2, lab)
```



Feature 5.6: Rambience

From this graph, we can conclude that food and service ratings decrease from familiar to the quiet ambience, while combined ratings increase.

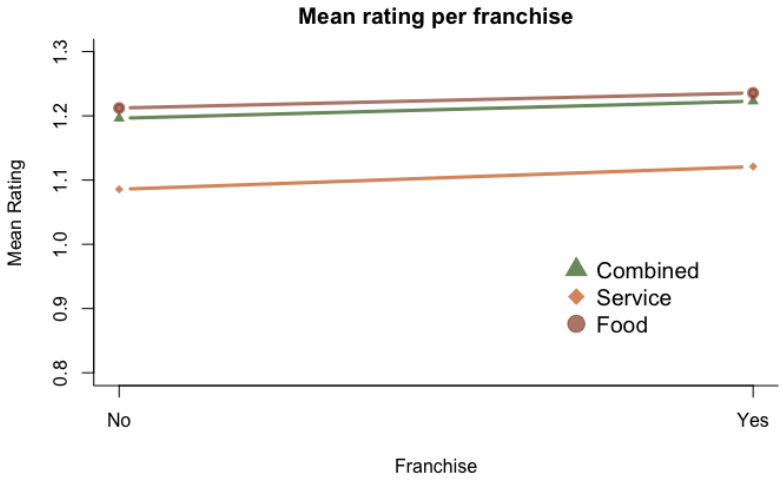
```
# Calculate mean rating per each type of Rambience for general rating
mean_familiar_rating <- calculateMeanRating("Rambience", "familiar", "rating")
mean_quiet_rating <- calculateMeanRating("Rambience", "quiet", "rating")
# Calculate mean rating per each type of Rambience for service rating
mean_familiar_service <- calculateMeanRating("Rambience", "familiar", "service_rating")
mean_quiet_service <- calculateMeanRating("Rambience", "quiet", "service_rating")
# Calculate mean rating per each type of Rambience for food rating
mean_familiar_food <- calculateMeanRating("Rambience", "familiar", "food_rating")
mean_quiet_food <- calculateMeanRating("Rambience", "quiet", "food_rating")
# Define variables
total <- c(0, 1)
rating <- c(mean_familiar_rating, mean_quiet_rating)
service <- c(mean_familiar_service, mean_quiet_service)
food <- c(mean_familiar_food, mean_quiet_food)
title <- "Mean rating per ambience"
xlab <- "Ambience"
ylim <-c (0.8, 1.3)
lab <- c("Familiar", "Quiet")
# Draw graph
drawConnectedScatterplot(total, rating, service, food, title, xlab, ylim, 0:1, lab)
```



Feature 5.7: franchise

This graph shows that there is no significant variance for a franchise or no franchise ratings.

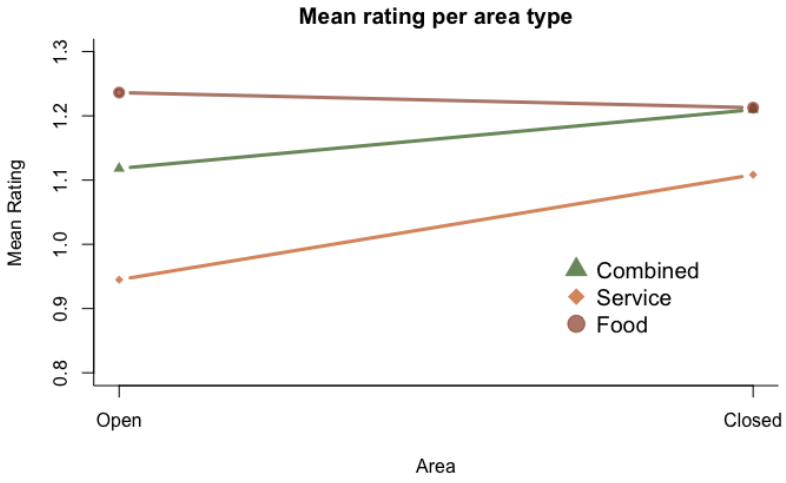
```
# Calculate mean rating per each type of franchise for general rating
mean_f_rating <- calculateMeanRating("franchise", "f", "rating")
mean_t_rating <- calculateMeanRating("franchise", "t", "rating")
# Calculate mean rating per each type of franchise for service rating
mean_f_service <- calculateMeanRating("franchise", "f", "service_rating")
mean_t_service <- calculateMeanRating("franchise", "t", "service_rating")
# Calculate mean rating per each type of franchise for food rating
mean_f_food <- calculateMeanRating("franchise", "f", "food_rating")
mean_t_food <- calculateMeanRating("franchise", "t", "food_rating")
# Define variables
total <- c(0, 1)
rating <- c(mean_f_rating, mean_t_rating)
service <- c(mean_f_service, mean_t_service)
food <- c(mean_f_food, mean_t_food)
title <- "Mean rating per franchise"
xlab <- "Franchise"
ylim <-c (0.8, 1.3)
lab <- c("No", "Yes")
# Draw graph
drawConnectedScatterplot(total, rating, service, food, title, xlab, ylim, 0:1, lab)
```



Feature 5.8: area

The following graph shows that restaurants that offer a closed area have better ratings that restaurants with an open area.

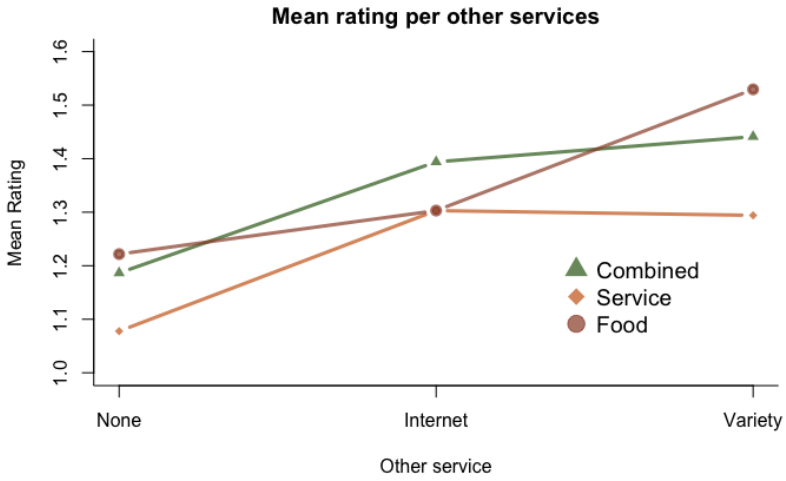
```
# Calculate mean rating per each type of area for general rating
mean_open_rating <- calculateMeanRating("area", "open", "rating")
mean_closed_rating <- calculateMeanRating("area", "closed", "rating")
# Calculate mean rating per each type of area for service rating
mean_open_service <- calculateMeanRating("area", "open", "service_rating")
mean_closed_service <- calculateMeanRating("area", "closed", "service_rating")
# Calculate mean rating per each type of area for food rating
mean_open_food <- calculateMeanRating("area", "open", "food_rating")
mean_closed_food <- calculateMeanRating("area", "closed", "food_rating")
# Define variables
total <- c(0, 1)
rating <- c(mean_open_rating, mean_closed_rating)
service <- c(mean_open_service, mean_closed_service)
food <- c(mean_open_food, mean_closed_food)
title <- "Mean rating per area type"
xlab <- "Area"
ylim <-c (0.8, 1.3)
lab <- c("Open", "Closed")
# Draw graph
drawConnectedScatterplot(total, rating, service, food, title, xlab, ylim, 0:1, lab)
```



Feature 5.9: other_services

This graph shows that restaurants that provide with internet or other services have better ratings comparing to restaurants that provide no additional services.

```
# Calculate mean rating per each type of other_services for general rating
mean_none_rating <- calculateMeanRating("other_services", "none", "rating")
mean_internet_rating <- calculateMeanRating("other_services", "Internet", "rating")
mean_variety_rating <- calculateMeanRating("other_services", "variety", "rating")
# Calculate mean rating per each type of other_services for service rating
mean_none_service <- calculateMeanRating("other_services", "none", "service_rating")
mean_internet_service <- calculateMeanRating("other_services", "Internet", "service_rating")
mean_variety_service <- calculateMeanRating("other_services", "variety", "service_rating")
# Calculate mean rating per each type of other_services for food rating
mean_nonefood <- calculateMeanRating("other_services", "none", "food_rating")
mean_internet_food <- calculateMeanRating("other_services", "Internet", "food_rating")
mean_variety_food <- calculateMeanRating("other_services", "variety", "food_rating")
# Define variables
total <- c(0, 1, 2)
rating <- c(mean_none_rating, mean_internet_rating, mean_variety_rating)
service <- c(mean_none_service, mean_internet_service, mean_variety_service)
food <- c(mean_none_food, mean_internet_food, mean_variety_food)
title <- "Mean rating per other services"
xlab <- "Other service"
ylim <-c (1.0, 1.6)
lab <- c("None", "Internet", "Variety")
# Draw graph
drawConnectedScatterplot(total, rating, service, food, title, xlab, ylim, 0:2, lab)
```



From this analysis we conclude that this dataset needs the following treatments:

- discover the missing cities with the given latitude and longitude, and then drop these coordinates;
- drop the_geom_meter. The location of the restaurant will be given by the city;
- drop the name of the restaurant, because it works as a unique identifier and it shouldn't influence ratings;
- drop address, state, fax, url, and zip, because we decided to focus in the city only to classify the location influence;
- drop country, because all restaurants are from Mexico (which means that this is an irrelevant feature).

Dataset 6: usercuisine

This dataset contains information about the preferred cuisine(s) type(s) of each user. It contains the following features:

- userID: corresponds to the user id;
- Rcuisine: corresponds to the type(s) of cuisine(s) that the user prefers. This is a nominal feature with 103 levels.

```
# Columns
names(usercuisine_df)
```

```
[1] "userID"   "Rcuisine"
```

```
# Number of attributes
ncol(usercuisine_df)
```

```
[1] 2
```

```
# Number of instances
nrow(usercuisine_df)
```

```
[1] 330
```

```
# Summary
summary(usercuisine_df)
```

```
     userID                Rcuisine
 U1135  :103   Mexican          : 97
 U1108  : 18   American         : 11
 U1101  : 15   Cafeteria        :  9
 U1016  : 14   Pizzeria         :  9
 U1060  : 13   Cafe-Coffee_Shop:  8
 U1008  : 10   Family           :  8
 (Other):157   (Other)          :188
```

```
# Head
head(usercuisine_df)
```

	userID <fctr>	Rcuisine <fctr>
1	U1001	American
2	U1002	Mexican
3	U1003	Mexican
4	U1004	Bakery
5	U1004	Breakfast-Brunch
6	U1004	Japanese

6 rows

```
# Number of nulls
sum(usercuisine_df == "?")
```

```
[1] 0
```

```
# Number of nulls per column
colSums(usercuisine_df == "?")
```

```
  userID Rcuisine
       0        0
```

```
# Number of levels of Rcuisine
sprintf("Rcuisine has %d levels.", nlevels(usercuisine_df$Rcuisine))
```

```
[1] "Rcuisine has 103 levels."
```

```
# Number of users
sprintf("There are %d unique userID's.", length(unique(usercuisine_df$userID)))
```

```
[1] "There are 138 unique userID's."
```

From this analysis, we conclude that this dataset has no missing values, but a treatment to Rcuisine is necessary to reduce the number of levels of this feature. This treatment should match the one performed to Rcuisine in chefmozcuisine.

Dataset 7: userpayment

This dataset contains information about the payment methods that users prefer. It contains the following features:

- userID: corresponds to the user id;
- Upayment: corresponds to the type of payment that the user prefers. This is a nominal feature with 5 levels.

```
# Columns
names(userpayment_df)
```

```
[1] "userID"   "Upayment"
```

```
# Number of attributes
ncol(userpayment_df)
```

```
[1] 2
```

```
# Number of instances
nrow(userpayment_df)
```

```
[1] 177
```

```
# Summary
summary(userpayment_df)
```

```
     userID                   Upayment
 U1041  :  4   American_Express   :  3
 U1044  :  4   bank_debit_cards   : 22
 U1076  :  3   cash               :131
 U1077  :  3   MasterCard-Eurocard:  4
 U1078  :  3   VISA               : 17
 U1086  :  3
 (Other):157
```

```
# Head
head(userpayment_df)
```

	userID <fctr>	Upayment <fctr>
1	U1001	cash
2	U1002	cash
3	U1003	cash
4	U1004	cash
5	U1004	bank_debit_cards
6	U1005	cash

6 rows

```
# Number of nulls
sum(userpayment_df == "?")
```

```
[1] 0
```

```
# Number of nulls per column
colSums(userpayment_df == "?")
```

```
  userID Upayment
       0        0
```

```
# Number of levels of Rpayment
sprintf("Upayment has %d levels.", nlevels(userpayment_df$Upayment))
```

```
[1] "Upayment has 5 levels."
```

```
# Number of users
sprintf("There are %d unique userID's.", length(unique(userpayment_df$userID)))
```

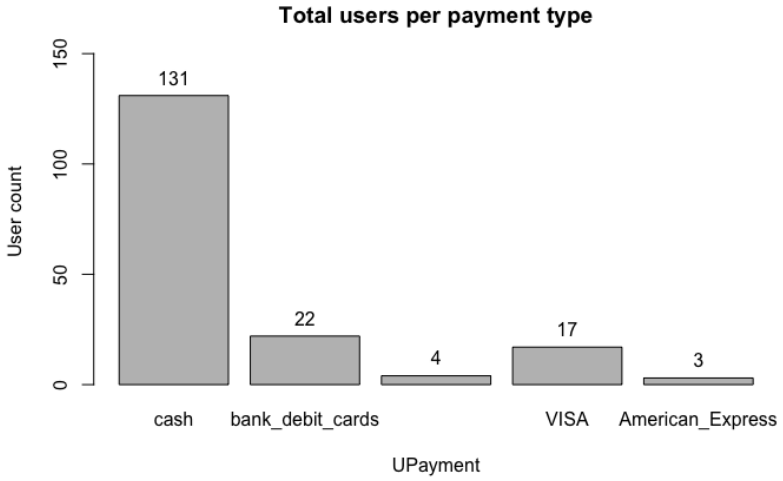
```
[1] "There are 133 unique userID's."
```

From this analysis, we conclude that this dataset has no missing values, but a treatment to Upayment is necessary to reduce the number of levels of this feature, and it should match the treatment performed to chefmozaccepts.

Feature 7.1: Upayment

A graphical visualization with a barplot shows the distribution of this feature.

```
upayment <- unique(userpayment_df$Upayment)
title <- "Total users per payment type"
xlab <- "UPayment"
ylab <- "User count"
feature <- "Upayment"
drawBarplot(userpayment_df, upayment, title, xlab, ylab, feature)
```



Dataset 8: userprofile

This dataset contains information about the users that rated the restaurants. It contains the following features:

- userID: corresponds to the user id;
- latitude and longitude: correspond to the geospatial coordinates of the user. These are both numeric features;
- smoker: indicates whether the user is smoker or not. This is a nominal feature with 2 levels and 3 missing values;
- drink_level: indicates the drink level of the user. This is a nominal feature with 3 levels, and no missing values;
- dress_preference: indicates the user's dress preference. This is a nominal feature with 4 levels and 5 missing values;
- ambience: indicates the ambience that the user prefers. This is a nominal feature with 3 levels and 6 missing values;
- transport: indicates the type of transportation that the user prefers. This is a nominal feature with 3 levels and 7 missing values;
- marital_status: indicates the marital status of the user. This is a nominal feature with 3 levels and 4 missing values;
- hijos: indicates wheter the user has kids or not. This is a nominal feature with 3 levels and 11 missing values;
- birth_year: user's birth year. This is a numeric feature with no missing values;
- interest: indicates user's interest. This is a nominal feature with 5 levels, and no missing values;
- personality: indicates user's personality. This is a nominal feature with 4 levels, and no missing values;
- religion: indicates user's religion. This is a nominal feature with 5 levels, and no missing values;
- activity: indicates user's activity. This is a nominal feature with 4 levels and 7 missing values;
- color: indicates user's color preference. This is a nominal feature with 8 levels, and no missing values;
- weight: user's weight (numeric). This feature has no missing values;
- budget: indicates user's budget. This is a nominal feature with 3 levels and 7 missing values;
- height: user's height (numeric). This feature has no missing values

```
# Columns
names(userprofile_df)
```

```
 [1] "userID"           "latitude"         "longitude"        "smoker"           "drink_level"
"dress_preference"
 [7] "ambience"         "transport"        "marital_status"   "hijos"            "birth_year"
"interest"
[13] "personality"      "religion"         "activity"         "color"            "weight"
"budget"
[19] "height"
```

```
# Number of attributes
ncol(userprofile_df)
```

```
[1] 19
```

```
# Number of instances
nrow(userprofile_df)
```

```
[1] 138
```

```
# Summary
summary(userprofile_df)
```

```
     userID       latitude       longitude         smoker            drink_level      dress_pref
erence     ambience
 U1001  :  1   Min.   :18.81   Min.   :-101.05   ?    :  3   abstemious    :51   ?            :
5      ?       : 6
 U1002  :  1   1st Qu.:22.13   1st Qu.:-100.98   false:109   casual drinker:47   elegant      : 4
family  :70
 U1003  :  1   Median :22.15   Median :-100.94   true : 26   social drinker:40   formal       :41
friends :46
 U1004  :  1   Mean   :21.81   Mean   :-100.29                                   informal     :
35      solitary:16
 U1005  :  1   3rd Qu.:22.19   3rd Qu.: -99.18                                   no preference:5
3
 U1006  :  1   Max.   :23.77   Max.   : -99.07

 (Other):132

     transport  marital_status         hijos       birth_year           interest
personality      religion
 ?        : 7   ?      :  4    ?          : 11   Min.   :1930   eco-friendly:16   conformist
: 7   Catholic :99
 car owner:35   married: 10    dependent  :  3   1st Qu.:1987   none        :30   hard-worker
:61   Christian: 7
 on foot  :14   single :122    independent:113   Median :1989   retro       : 6   hunter-ostentat
ious:12   Jewish   : 1
 public   :82   widow  :  2    kids       : 11   Mean   :1985   technology  :36   thrifty-protect
or  :58   Mormon   : 1
                                                 3rd Qu.:1991   variety     :50
none     :30
                                                 Max.   :1994



         activity       color        weight          budget       height
 ?            :  7   blue   :45   Min.   : 40.00   ?     : 7   Min.   :1.200
 professional : 15   black  :21   1st Qu.: 53.00   high  : 5   1st Qu.:1.600
 student      :113   green  :19   Median : 65.00   low   :35   Median :1.690
 unemployed   :  2   red    :15   Mean   : 64.87   medium:91   Mean   :1.668
 working-class:  1   yellow :12   3rd Qu.: 74.75               3rd Qu.:1.750
                     purple :11   Max.   :120.00               Max.   :2.000
                     (Other):15
```

```
# Head
head(userprofile_df)
```

userID <fctr>	latitude <dbl>	longitude <dbl>	smoker <fctr>	drink_level <fctr>	dress_preference <fctr>	ambience <fctr>	transpo <fctr>
1 U1001	22.14000	-100.9788	false	abstemious	informal	family	on foot
2 U1002	22.15009	-100.9833	false	abstemious	informal	family	public
3 U1003	22.11985	-100.9465	false	social drinker	formal	family	public
4 U1004	18.86700	-99.1830	false	abstemious	informal	family	public
5 U1005	22.18348	-100.9599	false	abstemious	no preference	family	public
6 U1006	22.15000	-100.9830	true	social drinker	no preference	friends	car owne

6 rows | 1-9 of 19 columns

```
# Number of nulls
sum(userprofile_df == "?")
```

```
[1] 50
```

```
# Number of nulls per column
colSums(userprofile_df == "?")
```

```
          userID         latitude        longitude           smoker      drink_level dress_prefe
rence          ambience
               0                0                0                3                0
5                6
       transport   marital_status            hijos       birth_year         interest      person
ality          religion
               7                4               11                0                0
0                0
        activity            color           weight           budget           height
               7                0                0                7                0
```

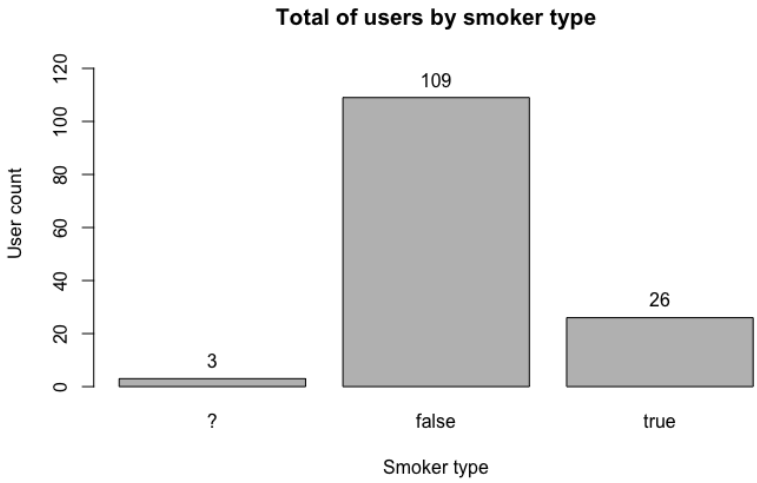
```
# Number of users
sprintf("There are %d unique userID's.", length(unique(userprofile_df$userID)))
```

```
[1] "There are 138 unique userID's."
```

The elaboration of some barplots allowed us to understand the distribution of these nominal features.

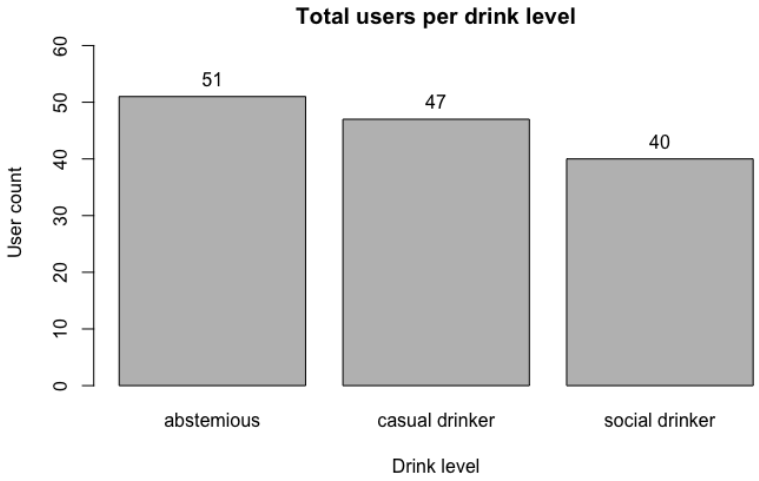
Feature 8.1: smoker

```
smoker_type <- unique(levels(userprofile_df$smoker))
title <- "Total of users by smoker type"
xlab <- "Smoker type"
ylab <- "User count"
feature <- "smoker"
drawBarplot(userprofile_df, smoker_type, title, xlab, ylab, feature)
```



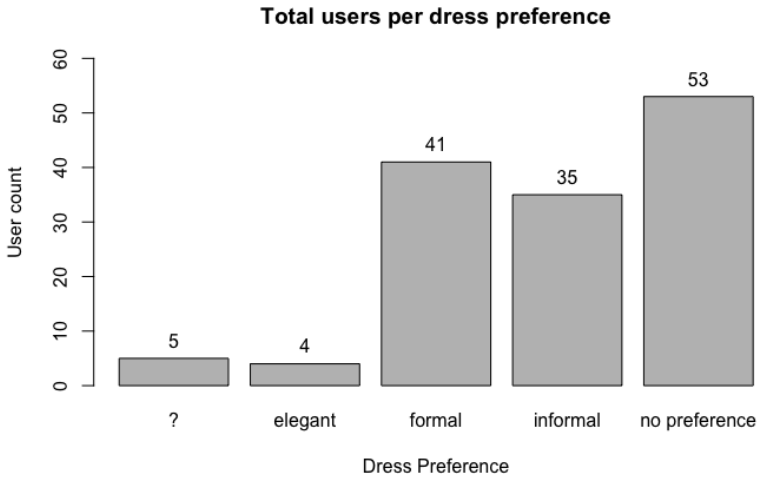
Feature 8.2: drink_level

```
drink_level_type <- unique(levels(userprofile_df$drink_level))
title <- "Total users per drink level"
xlab <- "Drink level"
ylab <- "User count"
feature <- "drink_level"
drawBarplot(userprofile_df, drink_level_type, title, xlab, ylab, feature)
```



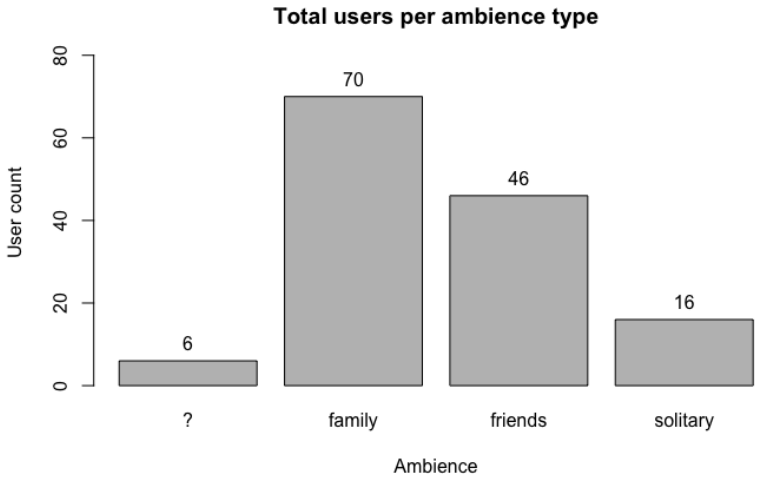
Feature 8.3: dress_preference

```
dress_preference_type <- unique(levels(userprofile_df$dress_preference))
title <- "Total users per dress preference"
xlab <- "Dress Preference"
ylab <- "User count"
feature <- "dress_preference"
drawBarplot(userprofile_df, dress_preference_type, title, xlab, ylab, feature)
```



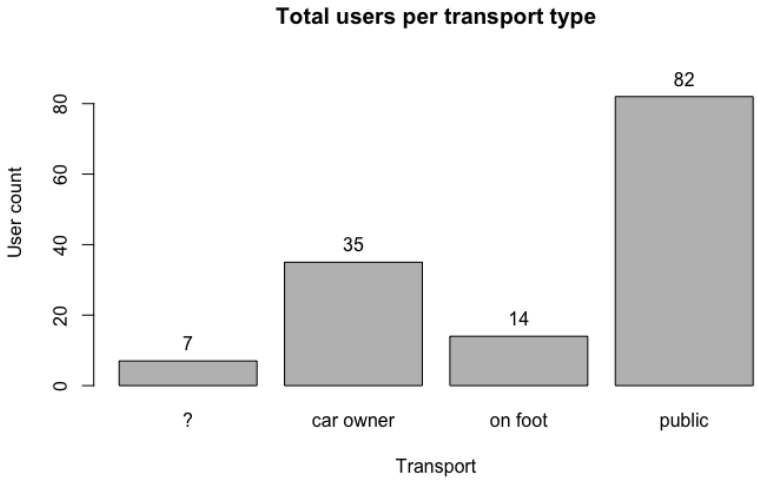
Feature 8.4: ambience

```
ambience_type <- unique(levels(userprofile_df$ambience))
title <- "Total users per ambience type"
xlab <- "Ambience"
ylab <- "User count"
feature <- "ambience"
drawBarplot(userprofile_df, ambience_type, title, xlab, ylab, feature)
```



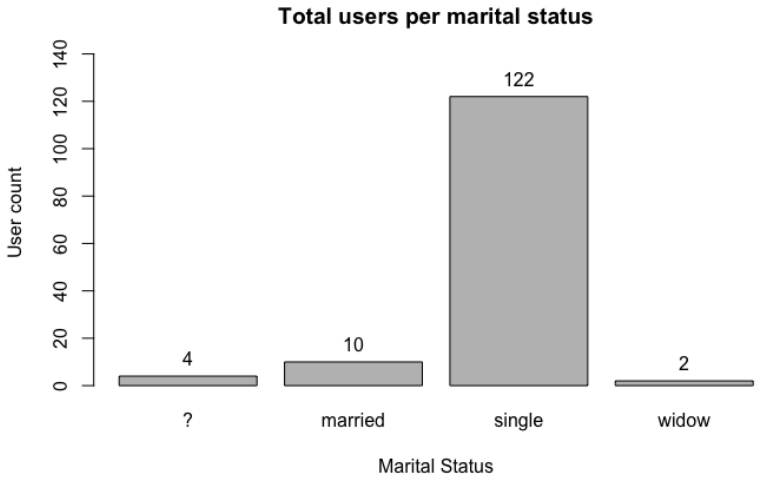
Feature 8.5: transport

```
transport_type <- unique(levels(userprofile_df$transport))
title <- "Total users per transport type"
xlab <- "Transport"
ylab <- "User count"
feature <- "transport"
drawBarplot(userprofile_df, transport_type, title, xlab, ylab, feature)
```



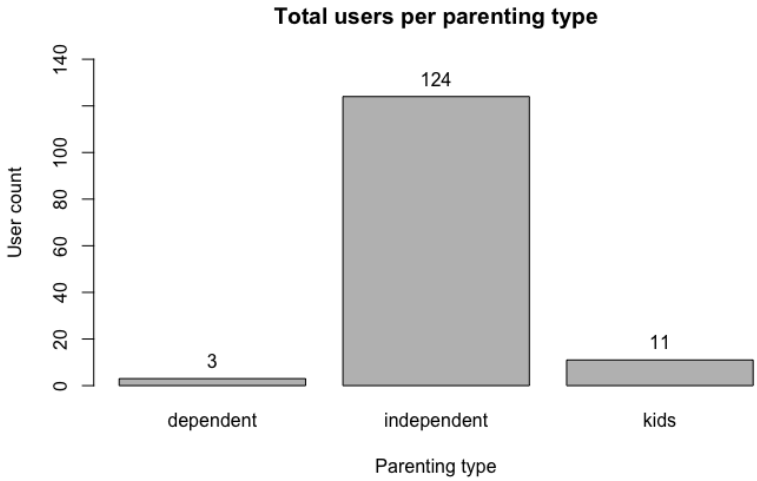
Feature 8.6: marital_status

```
marital_status <- unique(levels(userprofile_df$marital_status))
title <- "Total users per marital status"
xlab <- "Marital Status"
ylab <- "User count"
feature <- "marital_status"
drawBarplot(userprofile_df, marital_status, title, xlab, ylab, feature)
```



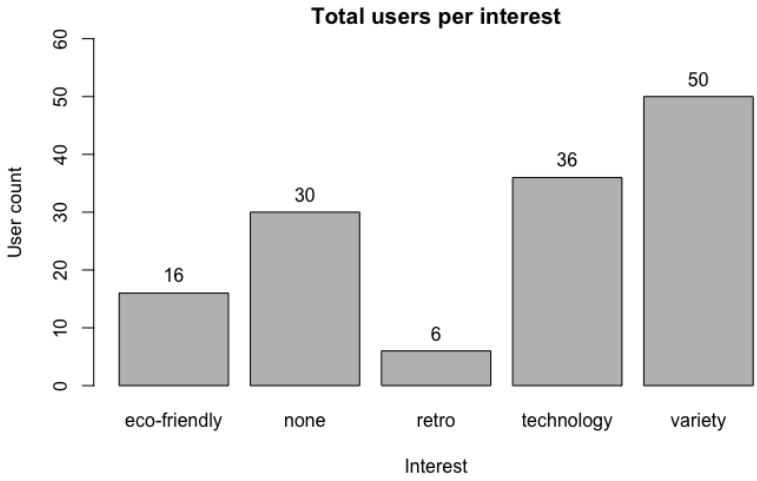
Feature 8.7: hijos

```
hijos <- unique(levels(userprofile_df$hijos))
title <- "Total users per parenting type"
xlab <- "Parenting type"
ylab <- "User count"
feature <- "hijos"
drawBarplot(userprofile_df, hijos, title, xlab, ylab, feature)
```



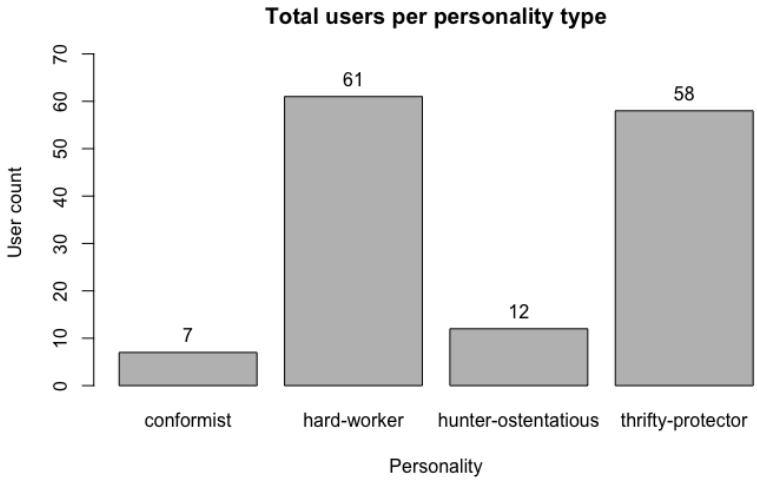
Feature 8.8: interest

```
interest <- unique(levels(userprofile_df$interest))
title <- "Total users per interest"
xlab <- "Interest"
ylab <- "User count"
feature <- "interest"
drawBarplot(userprofile_df, interest, title, xlab, ylab, feature)
```



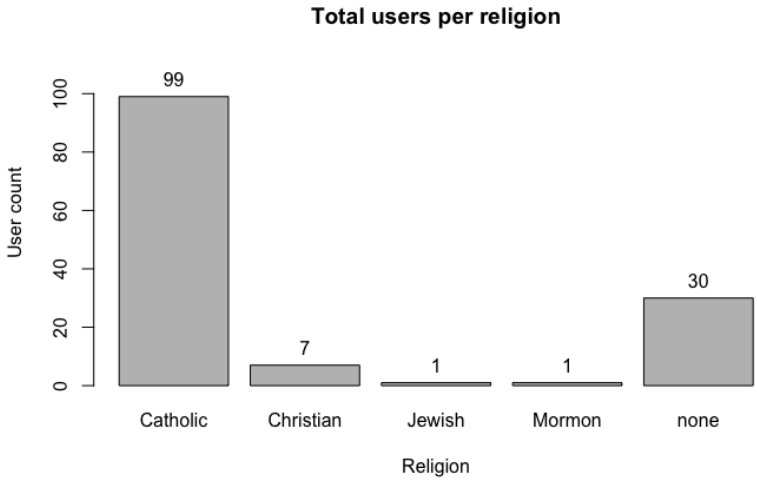
Feature 8.9: personality

```
personality <- unique(levels(userprofile_df$personality))
title <- "Total users per personality type"
xlab <- "Personality"
ylab <- "User count"
feature <- "personality"
drawBarplot(userprofile_df, personality, title, xlab, ylab, feature)
```



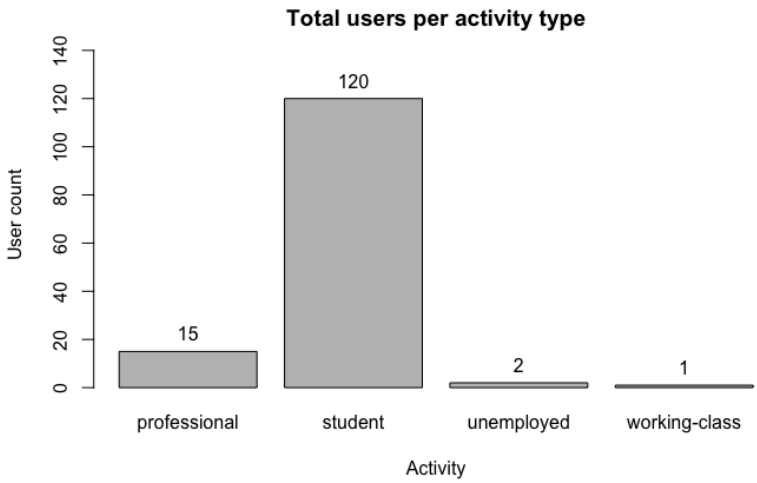
Feature 8.10: religion

```
religion <- unique(levels(userprofile_df$religion))
title <- "Total users per religion"
xlab <- "Religion"
ylab <- "User count"
feature <- "religion"
drawBarplot(userprofile_df, religion, title, xlab, ylab, feature)
```



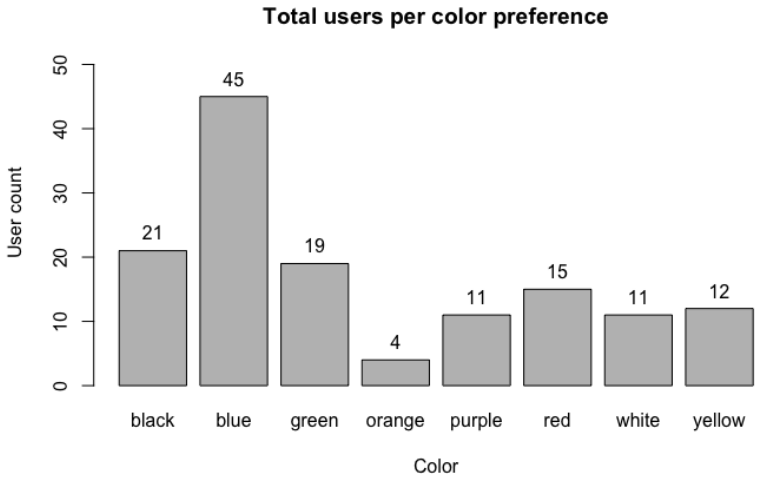
Feature 8.11: activity

```
activity <- unique(levels(userprofile_df$activity))
title <- "Total users per activity type"
xlab <- "Activity"
ylab <- "User count"
feature <- "activity"
drawBarplot(userprofile_df, activity, title, xlab, ylab, feature)
```



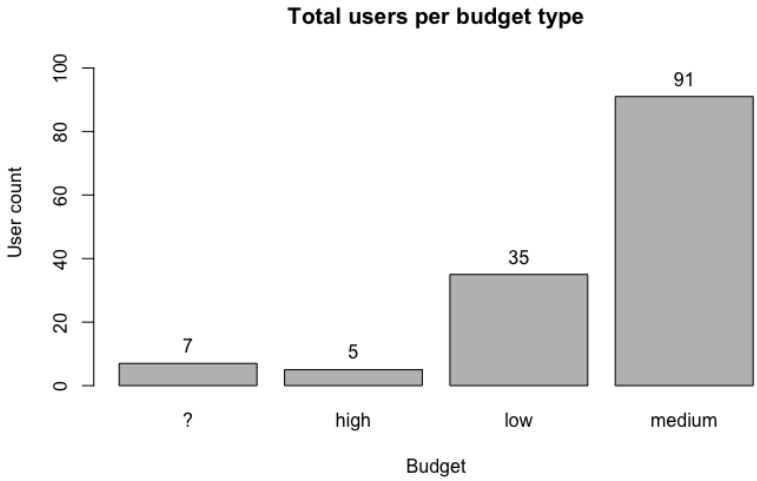
Feature 8.12: color

```
color <- unique(levels(userprofile_df$color))
title <- "Total users per color preference"
xlab <- "Color"
ylab <- "User count"
feature <- "color"
drawBarplot(userprofile_df, color, title, xlab, ylab, feature)
```



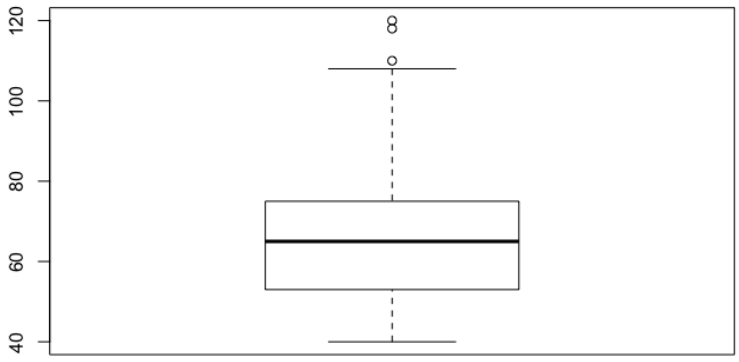
Feature 8.13: budget

```
budget_type <- unique(levels(userprofile_df$budget))
title <- "Total users per budget type"
xlab <- "Budget"
ylab <- "User count"
feature <- "budget"
drawBarplot(userprofile_df, budget_type, title, xlab, ylab, feature)
```



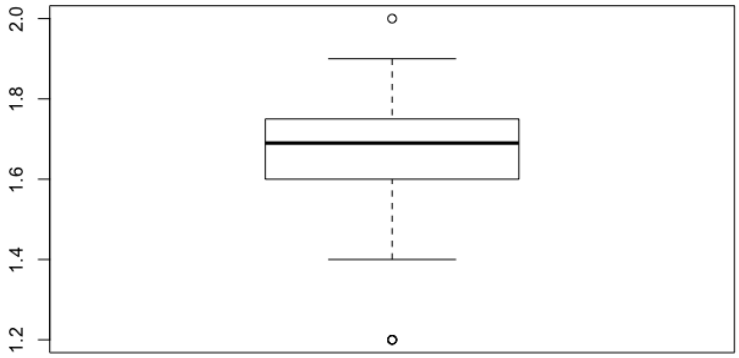
Feature 8.14: weight

We decided to analyze the distribution of the feature weight using a boxplot, but these values shouldn't be relevant to predict the ratings.



Feature 8.15: height

We decided to analyze the distribution of the feature height using a boxplot, but these values shouldn't be relevant to predict the ratings.



From the previous analysis we conclude that this dataset needs the following treatments:

- discover the user cities to match the restaurant cities with the given latitude and longitude, and then drop these coordinates;
- drop the_geom_meter, like we did in geoplaces;
- discover smoker missing values;
- discover dress_preference missing values;
- discover ambience missing values;
- discover transport missing values;
- discover marital_status missing values;
- discover hijos missing values;
- drop birth_year;
- discover activity missing values;
- discover budget missing values;
- drop weight and height, since these features shouldn't influence the ratings.

Dataset 9: rating_final

The final dataset contains information about the ratings that users gave to restaurants. It contains the following features:

- userID: corresponds to the user id;
- placeID: corresponds to the restaurant id;
- rating: combined rating of the restaurant. This is a numerical feature;
- food_rating: rating that corresponds to the quality of the food of the restaurant. This is a numerical feature;
- service_rating: rating that corresponds to the quality of the service of the restaurant. This is a numerical feature.

It's interesting to notice that even though rating, food_rating, and service_rating are presented as numerical features, they only have 3 possible values: 0, 1 or 2. This means that the problem of predicting the ratings can either be treated as a regression or classification problem.

```
# Columns
names(rating_final_df)
```

```
[1] "userID"  "placeID" "rating"
```

```
# Number of attributes
ncol(rating_final_df)
```

```
[1] 3
```

```
# Number of instances
nrow(rating_final_df)
```

```
[1] 1161
```

```
# Summary
summary(rating_final_df)
```

```
     userID        placeID           rating
 U1061  :  18   Min.   :132560   Min.   :0.0
 U1106  :  18   1st Qu.:132856   1st Qu.:1.0
 U1134  :  16   Median :135030   Median :1.0
 U1024  :  15   Mean   :134192   Mean   :1.2
 U1022  :  14   3rd Qu.:135059   3rd Qu.:2.0
 U1089  :  14   Max.   :135109   Max.   :2.0
 (Other):1066
```

```
# Head
head(rating_final_df)
```

	userID <fctr>	placeID <int>	rating <int>
1	U1077	135085	2
2	U1077	135038	2
3	U1077	132825	2
4	U1077	135060	1
5	U1068	135104	1
6	U1068	132740	0

6 rows

```
# Number of nulls
sum(rating_final_df == "?")
```

```
[1] 0
```

```
# Number of nulls per column
colSums(rating_final_df == "?")
```

```
 userID placeID  rating
      0       0       0
```

```
# Number of users
sprintf("There are %d unique userID's.", length(unique(rating_final_df$userID)))
```

```
[1] "There are 138 unique userID's."
```

```
# Number of resstaurant
sprintf("There are %d unique placeID's.", length(unique(rating_final_df$placeID)))
```

```
[1] "There are 130 unique placeID's."
```

Conclusion

In this phase, we analyzed the available data presented in the given dataset and decided which treatments are necessary to prepare data for the modeling phase. In total, there are 938 restaurants and 138 users. Users provided 1161 ratings, but only to 130 restaurants different restaurants. Each restaurant and each user are characterized by a set of features, which will be used to predict the missing ratings.

```
total_restaurants <- length(unique(c(chefmozaccepts_df$placeID, chefmozcuisine_df$placeID, chefmoz
hours4_df$placeID, chefmozparking_df$placeID, geoplaces2_df$placeID)))
total_users <- length(unique(c(usercuisine_df$userID, userpayment_df$userID, userprofile_df$userID
)))
sprintf("There are %d restaurants and %d users.", total_restaurants, total_users)
```

3. Data Preparation

Similarly to the approach used in Data Understanding, we execute each of the steps of this new CRISP-DM phase to all of the datasets

Dataset 1: chefmozaccepts

As observed in Data Understanding, this dataset has an attribute (Rpayment) that has 12 levels. An important treatment for this feature is to reduce the number of levels. The same problem exists in userpayment dataset with Upayment, but instead, this one has 5 levels. To accommodate the treatment for both datasets, the following levels will be created (instead of the originals presented in the datasets):

1. Card
   - Instead of: American_Express, bank_debit_cards, Carte_Blanche, Visa, VISA, MasterCard-Eurocard, Diners_Club, Discover, and Japan_Credit_Bureau
2. Cash
   - Instead of: cash
3. Other
   - Instead of: checks, gift_certificates

```
# Auxiliar function: Calculate new payment feature
calculatePayment <- function(dataset) {

  dataset$payment <- c("")

  # Compute new payment types
  for(i in seq_len(nrow(dataset))) {
    payment <- as.character(dataset[i,2])
    if(payment == "American_Express" || payment == "bank_debit_cards" || payment == "Carte_Blanch
e" || payment == "Visa" || payment == "VISA" || payment == "MasterCard-Eurocard" || payment == "D
iners_Club" || payment == "Discover" || payment == "Japan_Credit_Bureau") {
      dataset[i,3] <- "Card"
    } else if(payment == "cash") {
      dataset[i,3] <- "Cash"
    } else {
      dataset[i,3] <- "Other"
    }
  }

  # Transform Rpayment to calculated values as factor
  dataset[,2] <- as.factor(dataset$payment)

  # Drop payment column
  drops <- c("payment")
  dataset <- dataset[ , !(names(dataset) %in% drops)]

  # Return only unique rows
  return(unique(dataset))
}
```

```
# Calculate new Rpayment types
chefmozaccepts_df <- calculatePayment(chefmozaccepts_df)
summary(chefmozaccepts_df$Rpayment)
```

```
 Card  Cash Other
  303   500    16
```

Datset 2: chefmozcuisine

As observed in Data Understanding, this dataset has an attribute (Rcuisine) that has 59 levels. An important treatment for this feature is to reduce the number of levels. The same problem exists in usercuisine dataset, whereas this one has 103 levels. To accommodate the treatment for both datasets, the following levels will be created (instead of the originals presented in the datasets):

1. Bar/Cafeteria
   - Instead of: Bagels, Bakery, Bar, Bar_Pub_Brewery, Breakfast-Brunch, Cafe-Coffee_Shop, Cafeteria, Deli-Sandwiches, Dessert-Ice_Cream, Doughnuts, and Juice
2. National
   - Instead of: Mexican, Regional, and Tex-Mex
3. Oriental
   - Instead of: Asian, Cambodian, Chinese, Dim_Sum, Filipino, Japanese, Korean, Malaysian, Mongolian, Sushi, Thai, Tibetan, Vietnamese
4. American
   - Since the total of American cuisine is bigger than Asian cuisine, we decided to keep it as a level
5. Dutch-Belgian

- Since the total of Dutch-Belgian cuisine is bigger than Asian cuisine, we decided to keep it as a level

6. International (others)

- Instead of: Afghan, African, Armenian, Barbecue, Brazilian, Burgers, California, Caribbean, Contemporary, Continental-European, Diner, Dutch-Belgian, Eastern_European, Ethiopian, Family, Fast_Food, Fine_Dining, French, Game, German, Greek, Hot_Dogs, International, Italian, Latin_American, Mediterranean, Organic-Healthy, Persian, Pizzeria, Polish, Seafood, Soup, Southern, Southwestern, Spanish, Steaks, Turkish and Vegetarian

```
# Auxiliar function: Calculate new cuisine feature
calculateCuisine <- function(dataset) {

  dataset$cuisine <- c("")

  # Compute new cuisine types
  for(i in seq_len(nrow(dataset))) {
    cuisine <- as.character(dataset[i,2])
    if(cuisine == "Bagels" || cuisine == "Bakery" || cuisine == "Bar" || cuisine == "Bar_Pub_Brew
ery" || cuisine == "Breakfast-Brunch" || cuisine == "Cafe-Coffee_Shop" || cuisine == "Cafeteria" |
| cuisine == "Deli-Sandwiches" || cuisine == "Dessert-Ice_Cream" || cuisine == "Doughnuts" || cui
sine == "Juice") {
      dataset[i,3] <- "Bar/Cafeteria"
    } else if(cuisine == "Mexican" || cuisine == "Regional" || cuisine == "Tex-Mex") {
      dataset[i,3] <- "National"
    } else if(cuisine == "Asian" || cuisine == "Cambodian" || cuisine == "Chinese" || cuisine ==
"Dim_Sum" || cuisine == "Filipino" || cuisine == "Japanese" || cuisine == "Korean" || cuisine ==
"Malaysian" || cuisine == "Mongolian" || cuisine == "Sushi" || cuisine == "Thai" || cuisine == "T
ibetan" || cuisine == "Vietnamese") {
      dataset[i,3] <- "Oriental"
    } else if(cuisine == "American") {
      dataset[i,3] <- "American"
    } else if(cuisine == "Dutch-Belgian") {
      dataset[i,3] <- "Dutch-Belgian"
    } else {
      dataset[i,3] <- "International"
    }
  }

  # Transform Rcuisine to calculated values as factor
  dataset[,2] <- as.factor(dataset$cuisine)

  # Drop cuisine column
  drops <- c("cuisine")
  dataset <- dataset[ , !(names(dataset) %in% drops)]

  # Return only unique rows
  return(unique(dataset))
}
```

```
# Calculate new Rcuisine types
chefmozcuisine_df <- calculateCuisine(chefmozcuisine_df)
summary(chefmozcuisine_df$Rcuisine)
```

```
     American Bar/Cafeteria Dutch-Belgian International      National      Oriental
           59           107            55           316           242            47
```

Dataset 3: chefmozhours4

As discussed in Data Understanding, it's necessary to reduce the number of levels presented in hours feature. In order to achieve this, we decided to create three new features: saturday, sunday and weekday. By analyzing the data in this dataset we can verify if the restaurant is open on these respective days, and therefore fill their values (1 if open, 0 if closed).

```
# Select saturday rows
restaurant_saturday_df <- chefmozhours4_df[which(chefmozhours4_df$days == "Sat;"),]
# Select sunday rows
restaurant_sunday_df <- chefmozhours4_df[which(chefmozhours4_df$days == "Sun;"),]
# Select weekday rows
restaurant_weekday_df <- chefmozhours4_df[which(chefmozhours4_df$days == "Mon;Tue;Wed;Thu;Fri;"),]
# Merge the three types in one dataframe
restaurant_days_df <- merge(restaurant_saturday_df, restaurant_sunday_df, by = "placeID")
restaurant_days_df <- merge(restaurant_days_df, restaurant_weekday_df, by = "placeID")
# Rename columns
setnames(restaurant_days_df, old=c("hours.x","hours.y","hours"), new=c("saturday", "sunday", "week
day"))
# Removed unnecessary columns and drop duplicated rows
restaurant_days_df <- restaurant_days_df[,c("placeID", "saturday", "sunday", "weekday")]
restaurant_days_df <- restaurant_days_df[!duplicated(restaurant_days_df[c("placeID", "saturday", "
sunday", "weekday")]),]
# Transform factor into string
restaurant_days_df <- data.frame(lapply(restaurant_days_df, as.character), stringsAsFactors = FALS
E)
# Verify if restaurant is open on saturday
for(placeID in unique(restaurant_days_df$placeID)) {
  test <- restaurant_days_df[which(restaurant_days_df$placeID == placeID),]
  if(is.element("00:00-00:00;", test$saturday)){
    #Closed restaurant
    restaurant_days_df[which(restaurant_days_df$placeID == placeID),]$saturday <- 0
  } else {
    #Open restaurant (the Na will considered "open")
    restaurant_days_df[which(restaurant_days_df$placeID == placeID),]$saturday <- 1
  }
}
# Verify if restaurant is open on sunday
for(placeID in unique(restaurant_days_df$placeID)) {
  test <- restaurant_days_df[which(restaurant_days_df$placeID == placeID),]
  if(is.element("00:00-00:00;", test$sunday)){
    #Closed restaurant
    restaurant_days_df[which(restaurant_days_df$placeID == placeID),]$sunday <- 0
  } else {
    #Open restaurant (the Na will considered "open")
    restaurant_days_df[which(restaurant_days_df$placeID == placeID),]$sunday <- 1
  }
}
# Verify if restaurant is open on weekdays
for(placeID in unique(restaurant_days_df$placeID)) {
  test <- restaurant_days_df[which(restaurant_days_df$placeID == placeID),]
  if(is.element("00:00-00:00;", test$weekday)){
    #Closed restaurant
    restaurant_days_df[which(restaurant_days_df$placeID == placeID),]$weekday <- 0
  } else {
    #Open restaurant (the Na will considered "open")
    restaurant_days_df[which(restaurant_days_df$placeID == placeID),]$weekday <- 1
  }
}
chefmozhours4_df <- restaurant_days_df
chefmozhours4_df[,2] <- as.factor(chefmozhours4_df$saturday)
chefmozhours4_df[,3] <- as.factor(chefmozhours4_df$sunday)
chefmozhours4_df[,4] <- as.factor(chefmozhours4_df$weekday)
summary(chefmozhours4_df)
```

```
   placeID          saturday sunday  weekday
 Length:688         0: 27    0: 49   0: 24
 Class :character   1:661    1:639   1:664
 Mode  :character
```

Dataset 4: chefmozparking

Dataset 4: chefmozparking

As discussed in Data Understanding, no treatment will be applied to this dataset.

Dataset 5: geoplaces2

5.1 Discover missing cities

As the original dataset has 18 missing cities, we will use restaurant's latitude and longitude to call a geo-information API (maps.googleapis) in order to fill the missing values.

```
# Auxiliar function: calculate city
calculateCity <- function(dataset, filename) {

  # Select latitude and longitude from dataset
  coordinates_numeric <- subset(dataset, select=c(latitude, longitude))
  coordinates <- data.frame(id=numeric(), name=character(), stringsAsFactors=FALSE)

  for(i in seq_len(nrow(dataset))) {
    coordinates[i,]$id <- i
    coordinates[i,]$name <- NA
  }

  # As Google API responds with random results (doesn't always return a city to the provided coord
inates), it's necessary to make the same calls several, and merge the results
  for(i in 1:5) {

    # API call
    result <- lapply(with(coordinates_numeric, paste(latitude, longitude, sep = ",")), geocode, ou
tput = "more")

    # Extract city
    city = sapply(result, "[[", "locality")

    # Create auxiliar dataframe
    df <- data.frame(id=numeric(), name=character(), stringsAsFactors=FALSE)

    # Replace "character(0)" with NA
    for(i in seq_len(length(city))) {
      if((sapply(city, as.character)[i])=="character(0)") {
        df[i,]$id <- i
        df[i,]$name <- NA
      } else {
        df[i,]$id <- i
        df[i,]$name <- sapply(city, as.character)[i]
      }
    }

    df$name <- sapply(df$name,as.character)

    # Merge coordinates with the discovered cities
    coordinates <- merge(coordinates, df, by = "id")

  }

  # Merge all results into one column
  for(i in seq_len(nrow(coordinates))){
    for(j in seq_len(ncol(coordinates))){
      if (j > 1) {
        if(!is.na(coordinates[i,j])) {
          coordinates[i,2] <- coordinates[i,j]
          break
        }

      }
    }
  }

  # Extact only id and first city column
  coordinates <- coordinates[, c("id", "name.x")]

  # Rename name.x to city
  names(coordinates) <- c("id", "city")

  # Save calculated cities
  write.csv(coordinates, file = filename, encoding="UTF-8")
}
```

```
calculateCity(geoplaces, "restaurant_cities.csv")

# Read cities
restaurant_cities <- read.csv("restaurant_cities.csv")

# Substitute missing city with the correct value
restaurant_cities[95,]$city <- "Ciudad Victoria"

# Drop old city
drops <- c("city")
geoplaces2_df <- geoplaces2_df[ , !(names(geoplaces2_df) %in% drops)]

# Merge city into geoplaces
geoplaces2_df$id<-NA
for(i in seq_len(nrow(geoplaces2_df))) {
    geoplaces2_df[i,]$id <- i
}
geoplaces2_df <- merge(restaurant_cities, geoplaces2_df, by = "id")

# Drop id
drops <- c("id", "X")
geoplaces2_df <- geoplaces2_df[ , !(names(geoplaces2_df) %in% drops)]

# Transform city into factor
geoplaces2_df$city <- as.factor(geoplaces2_df$city)
```

During this process, we were able to find 17/18 missing cities. For the last missing value, we decided to go directly to Google Maps and verify which was the corresponding city. Additionally, we decided to drop the initial city column and use only the values retrieved from the API since they had more quality. City ended up as a nominal feature with 7 levels.

```
summary(geoplaces2_df$city)
```

```
           Ciudad Victoria                 Cuernavaca                    Jiutepec
San Luis Potosí
                        23                         15                           5
79
Soledad de Graciano Sánchez                   Temixco               Tres de Mayo
                          5                          1                           2
```

5.2 Select relevant features

Since this dataset contains some irrelevant features to this project, we need to select only the relevant ones (as identified in section Data Understanding).

```
geoplaces2_df <- geoplaces2_df[,c("placeID", "city", "alcohol",
                          "smoking_area", "dress_code", "accessibility", "price", "Rambience",
                          "franchise", "area", "other_services")]
```

Dataset 6: usercuisine

As discussed in Data Understanding section, a treatment to Ucuisine will be applied to reduce the number of levels. This treatment matches the one performed to chefmozcuisine dataset.

```
summary(usercuisine_df$Rcuisine)
```

```
    American Bar/Cafeteria Dutch-Belgian International      National      Oriental
          11            13             1            33           102            14
```

Dataset 7: userpayment

As discussed in Data Understanding section, a treatment to Upayment will be applied to reduce the number of levels of this feature. This treatment matches the one performed to chefmozaccepts dataset.

```
# Calculate new Upayment types
userpayment_df <- calculatePayment(userpayment_df)

summary(userpayment_df$Upayment)
```

Dataset 8: userprofile

As seen in Data Understanding phase, userprofile dataset has 50 missing values. The columns with missing values are 'smoker', 'ambience', 'transport', 'marital_status', 'hijos', 'activity' and 'budget'. These are all categorical features, so we decided to replace the missing values with the corresponding mode of that column.

```
# Auxiliar function: Calculate mode
calculateMode <- function(v) {
   uniqv <- unique(v)
   uniqv[which.max(tabulate(match(v, uniqv)))]
}
```

8.1 Treat smoker missing values

This column has 50 missing values, where 26 are 'true' (smoker) and 112 are 'false' (non.smoker). This means that the mode of this column is 'false'. In this case, and because it's preferable to put a smoker person in a non-smoking restaurant than a non-smoker in a smoker restaurant, we will impute the missing values with 'false' (non-smoker).

```
summary(userprofile_df$smoker)
```

```
    ? false  true
    3   109    26
```

```
# Force NA's
userprofile_df[which(userprofile_df$smoker=="?"),]$smoker <- NA
length(which(is.na(userprofile_df$smoker)))
```

```
[1] 3
```

```
# Calculate mode
smoker_mode <- calculateMode(na.omit(userprofile_df$smoker))
sprintf("Mode: %s", smoker_mode)
```

```
[1] "Mode: false"
```

```
# Substitute
userprofile_df <- imputation("value", userprofile_df, "smoker", Value=smoker_mode)
print(summary(userprofile_df$smoker))
```

```
    ? false  true
    0   112    26
```

8.2 Treat dress_preference missing values

This column has 5 missing values. The mode of this column is ‘no preference’.

```
summary(userprofile_df$dress_preference)
```

```
           ?       elegant        formal      informal no preference 
           5             4            41            35            53 
```

```
# Force NA's
userprofile_df[which(userprofile_df$dress_preference=="?"),]$dress_preference <- NA
length(which(is.na(userprofile_df$dress_preference)))
```

```
[1] 5
```

```
# Calculate mode
dress_preference_mode <- calculateMode(na.omit(userprofile_df$dress_preference))
sprintf("Mode: %s", dress_preference_mode)
```

```
[1] "Mode: no preference"
```

```
# Substitute
userprofile_df <- imputation("value", userprofile_df, "dress_preference", Value=dress_preference_m
ode)
print(summary(userprofile_df$dress_preference))
```

```
           ?       elegant        formal      informal no preference 
           0             4            41            35            58 
```

8.3 Treat ambience missing values

This column has 6 missing values. The mode of this column is ‘family’.

```
summary(userprofile_df$ambience)
```

```
      ?   family  friends solitary 
      6       70       46       16 
```

```
# Force NA's
userprofile_df[which(userprofile_df$ambience=="?"),]$ambience <- NA
length(which(is.na(userprofile_df$ambience)))
```

```
[1] 6
```

```
# Calculate mode
ambience_mode <- calculateMode(na.omit(userprofile_df$ambience))
sprintf("Mode: %s", ambience_mode)
```

```
[1] "Mode: family"
```

```
# Substitute
userprofile_df <- imputation("value", userprofile_df, "ambience", Value=ambience_mode)
print(summary(userprofile_df$ambience))
```

```
       ?   family  friends solitary
       0       76       46       16
```

8.4 Treat transport missing values

This column has 7 missing values, and the mode is ‘public’.

```
summary(userprofile_df$transport_mode)
```

```
Length  Class   Mode
     0   NULL   NULL
```

```
# Force NA's
userprofile_df[which(userprofile_df$transport=="?"),]$transport <- NA
length(which(is.na(userprofile_df$transport)))
```

```
[1] 7
```

```
# Calculate mode
transport_mode <- calculateMode(na.omit(userprofile_df$transport))
sprintf("Mode: %s", transport_mode)
```

```
[1] "Mode: public"
```

```
# Substitute
userprofile_df <- imputation("value", userprofile_df, "transport", Value=transport_mode)
print(summary(userprofile_df$transport))
```

```
        ? car owner   on foot    public
        0        35        14        89
```

8.5 Treat marital_status missing values

This column has 4 missing values, and the mode is ‘single’.

```
summary(userprofile_df$marital_status)
```

```
      ? married  single   widow
      4      10     122       2
```

```
# Force NA's
userprofile_df[which(userprofile_df$marital_status=="?"),]$marital_status <- NA
length(which(is.na(userprofile_df$marital_status)))
```

```
[1] 4
```

```
# Calculate mode
marital_status_mode <- calculateMode(na.omit(userprofile_df$marital_status))
sprintf("Mode: %s", marital_status_mode)
```

```
[1] "Mode: single"
```

```
# Substitute
userprofile_df <- imputation("value", userprofile_df, "marital_status", Value=marital_status_mode)
print(summary(userprofile_df$marital_status))
```

```
      ? married  single   widow
      0      10     126       2
```

8.6 Treat hijos missing values

This column has 11 missing values, and the mode is ‘independent’.

```
summary(userprofile_df$hijos)
```

```
          ?   dependent independent        kids
         11           3         113          11
```

```
# Force NA's
userprofile_df[which(userprofile_df$hijos=="?"),]$hijos <- NA
length(which(is.na(userprofile_df$hijos)))
```

```
[1] 11
```

```
# Calculate mode
hijos_mode <- calculateMode(na.omit(userprofile_df$hijos))
sprintf("Mode: %s", hijos_mode)
```

```
[1] "Mode: independent"
```

```
# Substitute
userprofile_df <- imputation("value", userprofile_df, "hijos", Value=hijos_mode)
print(summary(userprofile_df$hijos))
```

```
          ?   dependent independent        kids
          0           3         124          11
```

8.7 Treat activity missing values

This column has 7 missing values, and the mode is ‘student’.

```
summary(userprofile_df$activity)
```

```
           ?  professional       student    unemployed working-class
           7            15           113             2             1
```

```
# Force NA's
userprofile_df[which(userprofile_df$activity=="?"),]$activity <- NA
length(which(is.na(userprofile_df$activity)))
```

```
[1] 7
```

```
# Calculate mode
activity_mode <- calculateMode(na.omit(userprofile_df$activity))
sprintf("Mode: %s", activity_mode)
```

```
[1] "Mode: student"
```

```
# Substitute
userprofile_df <- imputation("value", userprofile_df, "activity", Value=activity_mode)
print(summary(userprofile_df$activity))
```

```
           ?  professional       student    unemployed working-class
           0            15           120             2             1
```

8.8 Treat budget missing values

This column has 7 missing values, and the mode is ‘medium’.

```
summary(userprofile_df$budget)
```

```
    ?   high    low medium
    7      5     35     91
```

```
# Force NA's
userprofile_df[which(userprofile_df$budget=="?"),]$budget <- NA
length(which(is.na(userprofile_df$budget)))
```

```
[1] 7
```

```
# Calculate mode
budget_mode <- calculateMode(na.omit(userprofile_df$budget))
sprintf("Mode: %s", budget_mode)
```

```
[1] "Mode: medium"
```

```
# Substitute
userprofile_df <- imputation("value", userprofile_df, "budget", Value=budget_mode)
print(summary(userprofile_df$budget))
```

```
    ?   high    low medium
    0      5     35     98
```

8.9 Drop unused levels

After removing all NA’s, we need to remove that level from all of the columns.

```
# Drop unused levels
userprofile_df <- droplevels(userprofile_df, as.factor("?"))
```

8.10 Discover missing cities

```
# Calculate missing cities and store them into a csv
calculateCity(userprofile_df, "user_cities.csv")

# Read cities
user_cities <- read.csv("user_cities.csv")

# Verify is there is any missing city
sum(is.na(user_cities$city))

# Substitute missing cities with the correct value
user_cities[19,]$city <- "San Luis Potosí"
user_cities[74,]$city <- "Cuernavaca"
user_cities[79,]$city <- "Cuernavaca"
user_cities[81,]$city <- "San Luis Potosí"
user_cities[105,]$city <- "San Luis Potosí"
user_cities[109,]$city <- "San Luis Potosí"

# Drop old city
drops <- c("city")
userprofile_df <- userprofile_df[ , !(names(userprofile_df) %in% drops)]

# Merge city into userprofile
userprofile_df$id<-NA
for(i in seq_len(nrow(userprofile_df))) {
    userprofile_df[i,]$id <- i
}
userprofile_df <- merge(user_cities, userprofile_df, by = "id")

# Drop id
drops <- c("id", "X")
userprofile_df <- userprofile_df[ , !(names(userprofile_df) %in% drops)]

# Transform city into factor
userprofile_df$city <- as.factor(userprofile_df$city)

# Rename city to Ucity
setnames(userprofile_df, old=c("city"), new=c("Ucity"))
```

During this process, we were able to find all missing cities expect 6. For the last missing values, we decided to go directly to Google Maps and verify which was the corresponding city. Additionally, we decided to drop the initial city column and use only the values retrieved from the API since they had more quality. City ended up as a nominal feature with 7 levels.

```
summary(userprofile_df$Ucity)
```

```
              Calera Chica            Ciudad de México             Ciudad Victoria
Cuernavaca
                         1                           1                          25
10
            Emiliano Zapata                    Jiutepec                    La Joya
Lazaro Cardenas
                         1                           6                           1
2
            San Luis Potosí Soledad de Graciano Sánchez                     Temixco
Tepoztlán
                        72                          14                           1
1
               Tres de Mayo                  Xochitepec                   Yautepec
                         1                           1                           1
```

8.11 Select relevant features

Finally, we select only the relevant features identified in Data Understanding section.

```
userprofile_df <- userprofile_df[,c("userID", "Ucity", "smoker",
                            "drink_level", "dress_preference", "ambience", "transport", "marital
_status",
                            "hijos", "interest", "personality", "religion", "activity", "color",
"budget")]
```

Dataset 9: rating_final

As discussed in Data Understanding, this dataset doesn't need any treatment because it contains the ratings that users gave to the restaurants. The only step we need to do is to extract the combined rating since it's only rating that we will try to predict.

```
rating_final_df <- rating_final_df[,c("userID", "placeID", "rating")]
```

Modeling dataset

The dataset that will be used for modeling should include all the treatments applied during this phase and the relevant features that were identified.

```
# Join dataframes
final_df <- merge(rating_final_df, chefmozaccepts_df, by = "placeID")
final_df <- merge(final_df, chefmozcuisine_df, by = "placeID")
final_df <- merge(final_df, chefmozhours4_df, by = "placeID")
final_df <- merge(final_df, chefmozparking_df, by = "placeID")
final_df <- merge(final_df, geoplaces2_df, by = "placeID")
final_df <- merge(final_df, usercuisine_df, by = "userID")
final_df <- merge(final_df, userpayment_df, by = "userID")
final_df <- merge(final_df, userprofile_df, by = "userID")
```

```
# Analyze final dataframe
# Columns
names(final_df)
# Number of attributes
ncol(final_df)
# Number of instances
nrow(final_df)
# Summary
summary(final_df)
# Head
head(final_df)
# Number of nulls
sum(final_df == "?")
sum(is.na(final_df))
# Number of nulls per column
colSums(final_df == "?")
colSums(is.na(final_df))

# Drop userID and placeID
drops <- c("userID", "placeID")
final_df <- final_df[ , !(names(final_df) %in% drops)]

# Save in file
write.csv(final_df, file = "final_df.csv")
```

4. Modeling: 1st iteration

The main goal of this project is to predict the ratings that would be given by each customer for the restaurants that he/she didn't rate. In Data Understanding we saw that ratings are numerical values, but only three, so predicting their values can be treated as a regression or classification problem. As already discussed, we will focus on predicting only the combined ratings, and not food rating or service rating, using regression models.

For the first approach, we will select four different modeling techniques, and use k-fold method (with k = 2) to partition the training and datasets.

4.1 Linear regression #1

```
# Start the clock!
ptm <- proc.time()
# Create model
linear_model_1 <- mining(rating~., final_df, model = "lm", method = c("kfold", 2))
# Analyze model
summary(linear_model_1)
```

```
           Length Class  Mode
time       1      -none- numeric
test       1      -none- list
pred       1      -none- list
error      1      -none- numeric
mpar       1      -none- list
model      1      -none- character
task       1      -none- character
method     6      -none- list
sen        0      -none- NULL
sresponses 0      -none- NULL
runs       1      -none- numeric
attributes 0      -none- NULL
feature    1      -none- character
```

```
# Stop the clock
proc.time() - ptm
```

```
   user  system elapsed
  0.170   0.039   0.210
```

4.2 C&R Tree #1

```
# Start the clock!
ptm <- proc.time()
# Create model
crt_model_1 <- mining(rating~., final_df, model = "ctree", method = c("kfold", 2))
# Analyze model
summary(crt_model_1)
```

```
           Length Class  Mode
time       1      -none- numeric
test       1      -none- list
pred       1      -none- list
error      1      -none- numeric
mpar       1      -none- list
model      1      -none- character
task       1      -none- character
method     6      -none- list
sen        0      -none- NULL
sresponses 0      -none- NULL
runs       1      -none- numeric
attributes 0      -none- NULL
feature    1      -none- character
```

```
# Stop the clock
proc.time() - ptm
```

```
   user  system elapsed
  0.549   0.106   0.682
```

4.3 Decision Tree #1

```
# Start the clock!
ptm <- proc.time()
# Create model
dt_model_1 <- mining(rating~., final_df, model = "dt", method = c("kfold", 2))
# Analyze model
summary(dt_model_1)
```

```
           Length Class  Mode
time       1      -none- numeric
test       1      -none- list
pred       1      -none- list
error      1      -none- numeric
mpar       1      -none- list
model      1      -none- character
task       1      -none- character
method     6      -none- list
sen        0      -none- NULL
sresponses 0      -none- NULL
runs       1      -none- numeric
attributes 0      -none- NULL
feature    1      -none- character
```

```
# Stop the clock
proc.time() - ptm
```

```
   user  system elapsed
  0.270   0.045   0.321
```

4.4 Random Forest #1

```
# Start the clock!
ptm <- proc.time()
# Create model
dt_model_1 <- mining(rating~., final_df, model = "randomforest", method = c("kfold", 2))
```

```
The response has five or fewer unique values.  Are you sure you want to do regression?The response has five or fewer unique values.  Are you sure you want to do regression?
```

```
# Analyze model
summary(dt_model_1)
```

```
           Length Class  Mode
time       1      -none- numeric
test       1      -none- list
pred       1      -none- list
error      1      -none- numeric
mpar       1      -none- list
model      1      -none- character
task       1      -none- character
method     6      -none- list
sen        0      -none- NULL
sresponses 0      -none- NULL
runs       1      -none- numeric
attributes 0      -none- NULL
feature    1      -none- character
```

```
# Stop the clock
proc.time() - ptm
```

```
   user  system elapsed
 20.682   0.301  21.295
```

5. Evaluation: 1st iteration

In the first modeling iteration, we used four regression models (Linear Regression, C&R Tree, Decision Tree and Random Forest). All models had similar MAE (close to 0.5), except for Decision Tree and Random Forest which, with MAE close to 0.3, showed a clear performance increase. All the models took almost the same amount of time to compute (close to 1 second, since the amount of data that we have is small). It's possible to notice that Random Forest accuses 23 seconds, but this only happened because the algorithm kept asking if we wanted to use regression (after skipping that step, this algorithm is as fast as the others).

5.1 Linear regression #1

```
# Calculate metrics
mae <- mmetric(linear_model_1, metric = "MAE")
nmae <- mmetric(linear_model_1, metric = "NMAE")
sprintf("MAE = %f | NMAE = %f", mae, nmae)
```

```
[1] "MAE = 0.521249 | NMAE = 26.062443"
```

5.2 C&R Tree #1

```
# Calculate metrics
mae <- mmetric(crt_model_1, metric = "MAE")
nmae <- mmetric(crt_model_1, metric = "NMAE")
sprintf("MAE = %f | NMAE = %f", mae, nmae)
```

```
[1] "MAE = 0.465768 | NMAE = 23.288420"
```

5.3 Decision Tree #1

```
# Calculate metrics
mae <- mmetric(dt_model_1, metric = "MAE")
nmae <- mmetric(dt_model_1, metric = "NMAE")
sprintf("MAE = %f | NMAE = %f", mae, nmae)
```

```
[1] "MAE = 0.315163 | NMAE = 15.758173"
```

5.4 Random Forest #1

```
# Calculate metrics
mae <- mmetric(dt_model_1, metric = "MAE")
nmae <- mmetric(dt_model_1, metric = "NMAE")
sprintf("MAE = %f | NMAE = %f", mae, nmae)
```

```
[1] "MAE = 0.307608 | NMAE = 15.380375"
```

4. Modeling: 2nd iteration

The performance of the previous models doesn't reach reasonably acceptable levels. For this reason, we decided to try another approach using an increasing number of random forest trees. This modeling will be executed using Gradient Boosting,

which generates an increasing number of models on training data and uses their combined power to increase the accuracy of the final model which is formed by combining them.

```
# Start the clock!
ptm <- proc.time()
# Split training and test data
split = sample.split(final_df$rating, SplitRatio = 0.8)
training_set = subset(final_df, split == TRUE)
test_set = subset(final_df, split == FALSE)
# Create model
regressorBoost.boost=gbm(rating ~ . , data = training_set, distribution = "gaussian", n.trees = 10
000,
                 shrinkage = 0.01, interaction.depth = 4)
regressorBoost.boost
```

```
gbm(formula = rating ~ ., distribution = "gaussian", data = training_set,
    n.trees = 10000, interaction.depth = 4, shrinkage = 0.01)
A gradient boosted model with gaussian loss function.
10000 iterations were performed.
There were 32 predictors of which 32 had non-zero influence.
```

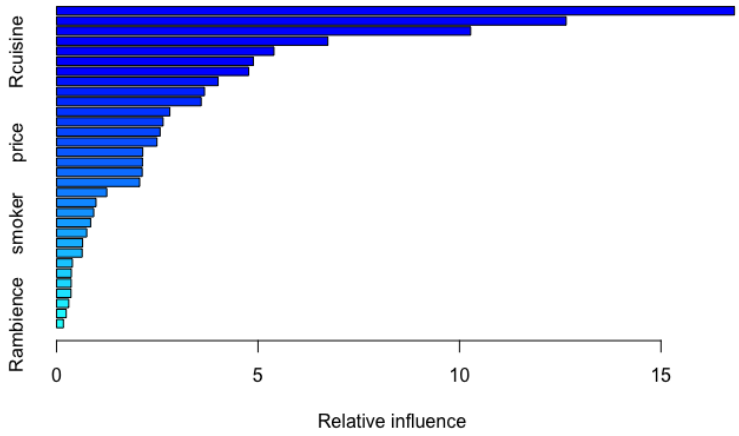
```
# Stop the clock
proc.time() - ptm
```

```
   user  system elapsed
 31.949   0.861  33.327
```

An important aspect to notice is that this model indicated that all of the 32 predictors (features) had some influence on the rating. The following graph shows the importance of all of the features, sorted from the most important to the least.

```
# Variable importance plot
summary(regressorBoost.boost) #Summary gives a table of Variable Importance and a plot of Variable
Importance
```

	var <fctr>	**rel.inf** <dbl>
color	color	16.8241536
Ucity	Ucity	12.6473885
interest	interest	10.2767015
drink_level	drink_level	6.7326869
Rcuisine	Rcuisine	5.3957670
smoking_area	smoking_area	4.8838385
dress_preference	dress_preference	4.7683908
ambience	ambience	4.0089794
parking_lot	parking_lot	3.6721951
alcohol	alcohol	3.5893015



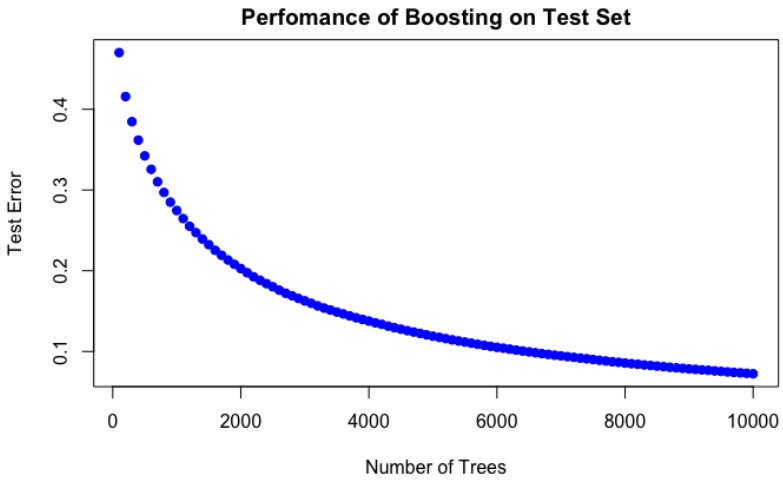
5. Evaluation: 2nd iteration

Finally, we plot the evolution of the mean squared error with the increase of trees.

```
# Create a sequence with the number of tree to use
n.trees = seq(from=100, to=10000, by=100)
# Calculate prediction matrix
predmatrix <- predict(regressorBoost.boost, training_set, n.trees = n.trees)
# Calculate mean squared error for the evolution of each 100 trees
test.error <- with(training_set, apply( (predmatrix-rating)^2, 2, mean))
head(test.error)
```

```
      100       200       300       400       500       600 
0.4701868 0.4157582 0.3846151 0.3617174 0.3422619 0.3256238 
```

```
# Plot the evolution of test error vs number of trees
plot(n.trees, test.error , pch=19, col="blue", xlab="Number of Trees", ylab="Test Error", main = "
Perfomance of Boosting on Test Set")
```



The number of necessary trees to reach a mean squared error of less than 0.1 is 6500.

```
head(test.error[which(test.error < 0.1)],1)
```

```
     6500 
0.09952152 
```

4. Modeling: 3rd iteration

As a final approach we decided to select the top 17 most influential features to analyze if the model performs better or faster, since it takes 35 seconds to compute all forests when using all features.

```
# Start the clock!
ptm <- proc.time()
# Create model
regressorBoost2.boost=gbm(rating ~ color + Ucity + interest + smoking_area + Rcuisine + drink_leve
l + dress_preference + alcohol + parking_lot + price + personality + ambience + religion + city +
transport + Ucuisine + accessibility , data = training_set, distribution = "gaussian", n.trees = 1
0000,
                 shrinkage = 0.01, interaction.depth = 4)
```

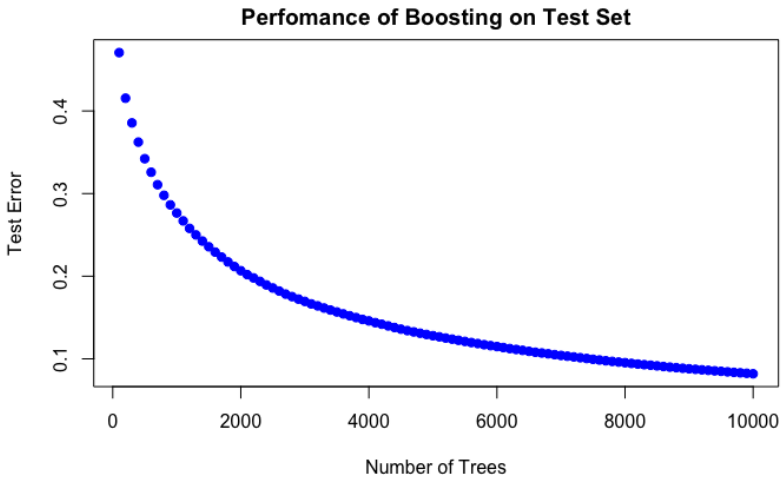
5. Evaluation: 3rd iteration

By removing the least important variables, the model needs to perform more iterations to achieve better results.

```
# Create a sequence with the number of tree to use
n.trees = seq(from=100, to=10000, by=100)
# Calculate prediction matrix
predmatrix <- predict(regressorBoost2.boost, training_set, n.trees = n.trees)
# Calculate mean squared error for the evolution of each 100 trees
test.error <- with(training_set, apply( (predmatrix-rating)^2, 2, mean))
head(test.error)
```

```
      100       200       300       400       500       600
0.4706265 0.4154545 0.3855391 0.3623002 0.3422111 0.3258519
```

```
# Plot the evolution of test error vs number of trees
plot(n.trees, test.error , pch=19, col="blue", xlab="Number of Trees", ylab="Test Error", main = "
Perfomance of Boosting on Test Set")
```



The number of necessary trees to reach a mean squared error of less than 0.1 is 7500.

```
head(test.error[which(test.error < 0.1)],1)
```

```
     7500
0.09945048
```

6. Deployment

This phase is not covered by this project.

7. Conclusion

During the research to execute this project we found many ways to implement regression models, each one with their own peculiarities. After evaluating the performance of Linear Regression, C&R Trees, Decision Trees and Random Forest, we decided to choose the Gradient Boosting technique, which makes use of several Random Trees, to achieve the desired predictions and a better understanding of our dataset. For the implementation of this technique, we selected and treated 32 features of the datasets, and ended up with a mean squared error of less than 0.1 by combining 6500 trees (Random Forest).

Bibliography

Restaurant & consumer data Data Set : https://archive.ics.uci.edu/ml/datasets/Restaurant+%26+consumer+data#

Content based recommenders: https://www.kaggle.com/liyenhsu/simple-content-based-recommenders

Mode calculation: https://www.tutorialspoint.com/r/r_mean_median_mode.htm

CRISP-DM: https://www.sv-europe.com/crisp-dm-methodology/

Gradient Boosting: https://datascienceplus.com/gradient-boosting-in-r/

Gradient Boosting: https://www.analyticsvidhya.com/blog/2016/04/complete-tutorial-tree-based-modeling-scratch-in-python/