DMII Restaurant recomnendation

Diogo Fernandes, João Sá

April 24, 2018

Introduction

Recommender systems are one of the most popular machine learning algorithms for internet business in a variety of areas. By prediction the preference or ratings that consumers would give to certain products, recommender systems perform efficient personalized marketing, matching the products to teir prespective customers[1]. Many companies with recommender systems, such as Amazon, Netflix, and Spotify, have successfly boosted ther profit by offering their consumers relevant items at the right time.

The dataset for this assignment is from the UCI Machine Learning Repository[2], which was originally obtained from a recommender system prototype[2]. These data contains the features of 130 restaurants and 128 consumers in Mexico, as well as the ratings given by the consumers to some of the restaurants. The goal of this assignment is to build and compare different recommender systems.

```
library(dplyr)
library(tidyr)
library(ggplot2)
library(corrplot)
library(recommenderlab)
library(Hmisc)
library(DMwR)
library(ggmap)

# capitalize the first letter of a string
capFirst <- function(s) {
  paste(toupper(substring(s, 1, 1)), substring(s, 2), sep = "")
}
```

1 Preliminary exploratory data analysis of the available data.

- The data are nine csv files, including five for the restaurant information, three for the consumer information, and on for ratings.
- 138 consumers, 130 restaurants; each restaurant in the data is identified by a placeID, and each consumer has a userID.
- Three ratings (overall rating, food rating and service rating) are given for a restaurant-consumer pair; there are a total of 1161 ratings available.

Intuitively, the main factors affecting a restaurant rating are: 1. the quality of the food; 2. the quality of the service provided. However, features of the restaurants, such as parking options, or price, may have small influences on the ratings.

```
payment_methods <- read.csv("RCdata/chefmozaccepts.csv")
cuisine_kind <- read.csv("RCdata/chefmozcuisine.csv")
open_hours <- read.csv("RCdata/chefmozhours4.csv")
parking <- read.csv("RCdata/chefmozparking.csv")
res_info <- read.csv("RCdata/geoplaces2.csv")
user_cuisine <- read.csv("RCdata/usercuisine.csv")
```

```
user_payment <- read.csv("RCdata/userpayment.csv")
user_info <- read.csv("RCdata/userprofile.csv")
rating <- read.csv("RCdata/rating_final.csv")

user_info$budget <- factor(user_info$budget, levels=c("low","medium","high"))
user_info$smoker <- factor(user_info$smoker, levels=c("false","true"))
user_info$transport <- factor(user_info$transport, levels=c("on foot", "public","car owner"))
user_info$dress_preference<- factor(user_info$dress_preferenc, levels=c("no preference", "informal","fo
```

Our approach will be always alike:
1. Unknown values checking;
2. Data visualization and analysis.

1.1 Payment methods

- Payment methods accepted by restaurants.

```
# see a few lines of data
head(payment_methods)
```

```
##   placeID            Rpayment
## 1  135110                cash
## 2  135110                VISA
## 3  135110 MasterCard-Eurocard
## 4  135110    American_Express
## 5  135110    bank_debit_cards
## 6  135109                cash
```

```
# see all payment methods and respective relative frequencies
summary(payment_methods$Rpayment)
```

```
##    American_Express    bank_debit_cards       Carte_Blanche
##                 153                 130                   7
##                cash              checks         Diners_Club
##                 500                  10                  42
##            Discover   gift_certificates Japan_Credit_Bureau
##                  11                   7                   5
## MasterCard-Eurocard                Visa                VISA
##                 194                  83                 172
```

As there are many different payment methods we decided to join them into categories, as we don't need that much discrimination for the purpose of our analysis, so there are no lost of valuable information.

```
# the two categories: Debit_Credit_cards and Others
Debit_Credit_cards <- c("VISA", "MasterCard-Eurocard", "American_Express",
                        "bank_debit_cards", "Visa", "Japan_Credit_Bureau",
                        "Discover", "Diners_Club")
Others <- c("Carte_Blanche", "gift_certificates", "checks")

levels(payment_methods$Rpayment)[levels(payment_methods$Rpayment) %in%
                                 Debit_Credit_cards] <- "Debit_Credit_cards"
levels(payment_methods$Rpayment)[levels(payment_methods$Rpayment) %in% Others] <- "Others"
levels(payment_methods$Rpayment) <- capFirst(levels(payment_methods$Rpayment))

payment_methods <- unique(payment_methods[1:nrow(payment_methods),])
```

```
# check the new levels
levels(payment_methods$Rpayment)
```

```
## [1] "Debit_Credit_cards" "Others"             "Cash"
```

```
# check the relative frequencies of the new category-organized data
summary(payment_methods$Rpayment)
```

```
## Debit_Credit_cards             Others               Cash
##                303                 21                500
```

Now we can get the frequency of each payment method. No surprise there, Cash is the prefered method.

1.2 Cuisine kinds

- Contains the cuisine kinds of each restaurant.

```
# see a few lines of data
head(cuisine_kind)
```

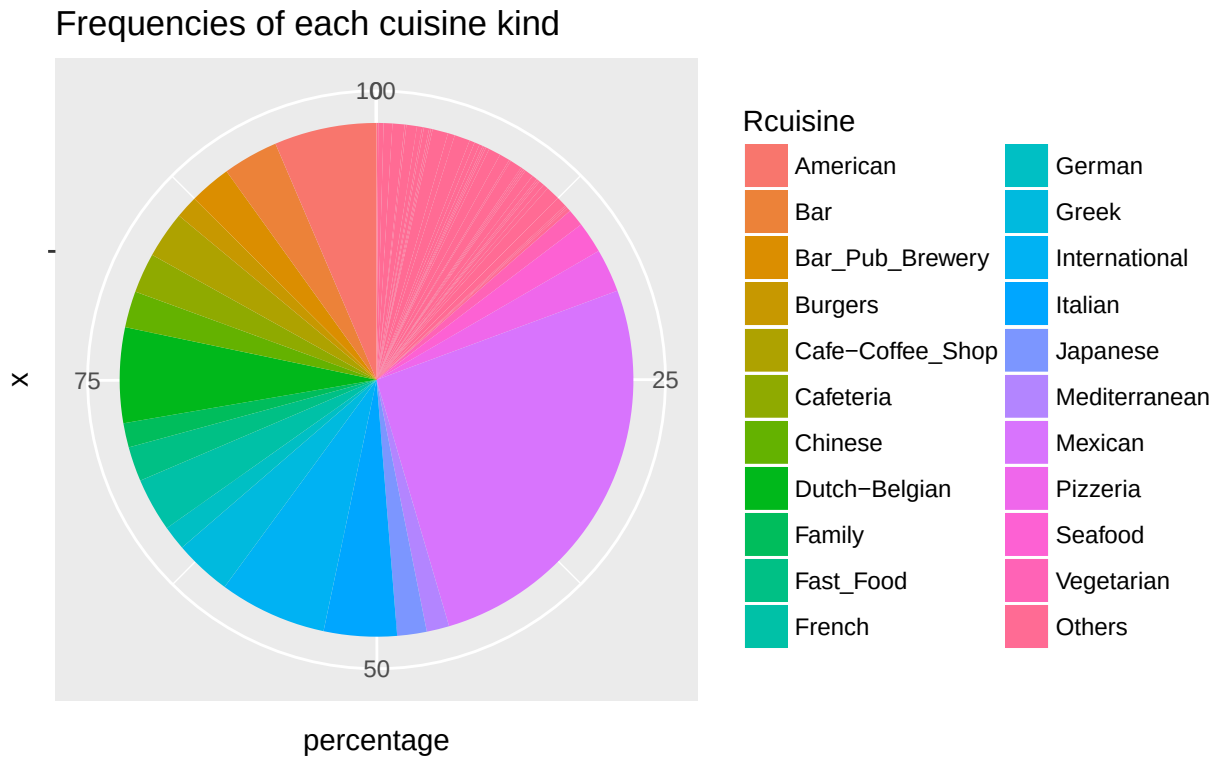
```
##   placeID       Rcuisine
## 1  135110        Spanish
## 2  135109        Italian
## 3  135107 Latin_American
## 4  135106        Mexican
## 5  135105      Fast_Food
## 6  135104        Mexican
```

```
# see all cuisine kinds and respective relative frequencies
summary(cuisine_kind$Rcuisine)
```

```
##               Afghan              African             American
##                    1                    3                   59
##             Armenian                Asian               Bagels
##                    5                    7                    1
##              Bakery                  Bar      Bar_Pub_Brewery
##                    6                   32                   24
##             Barbecue             Brazilian     Breakfast-Brunch
##                    3                    1                    3
##              Burgers     Cafe-Coffee_Shop            Cafeteria
##                   13                   27                   23
##           California            Caribbean              Chinese
##                    1                    1                   21
##         Contemporary Continental-European      Deli-Sandwiches
##                    9                    4                    9
##    Dessert-Ice_Cream               Diner        Dutch-Belgian
##                    3                    3                   55
##     Eastern_European             Ethiopian               Family
##                    2                    1                   14
##            Fast_Food          Fine_Dining               French
##                   20                    1                   31
##                 Game               German                Greek
##                    2                   14                   33
##             Hot_Dogs        International              Italian
##                    7                   62                   42
```

```
##            Japanese               Juice              Korean
##                  17                   6                   1
##       Latin_American       Mediterranean             Mexican
##                   7                  13                 239
##            Mongolian      Organic-Healthy             Persian
##                   1                   1                   1
##            Pizzeria              Polish            Regional
##                  25                   5                   3
##             Seafood               Soup            Southern
##                  18                   1                   1
##        Southwestern             Spanish              Steaks
##                   3                   3                   8
##               Sushi                Thai             Turkish
##                   6                   1                   1
##          Vegetarian          Vietnamese
##                  10                   2
```

We are not sure if it is a good idea to gather some kinds of cuisine into categories, as there may be lost of important information. For now, stays as it is.



No surprise, mexican cuisine is ahead!

1.3 Restaurant schedules

- Contains the schedules of restaurants.

```r
# see a few lines of data
head(cuisine_kind)
```

```
##   placeID       Rcuisine
## 1  135110        Spanish
## 2  135109        Italian
## 3  135107 Latin_American
## 4  135106        Mexican
## 5  135105      Fast_Food
## 6  135104        Mexican
```

```r
# get a summary including only the attributes "hours" and "days"
summary(open_hours[,c("hours", "days")])
```

```
##           hours                          days
##  00:00-23:30;: 681   Mon;Tue;Wed;Thu;Fri;:793
##  00:00-00:00;: 100   Sat;                 :783
##  17:00-22:00;:  56   Sun;                 :763
##  14:00-23:30;:  32
##  09:00-23:30;:  31
##  11:00-21:00;:  31
##  (Other)     :1408
```

```r
# simplify the dataset
levels(open_hours$days)[levels(open_hours$days) == "Mon;Tue;Wed;Thu;Fri;"] <- "Weekdays"
levels(open_hours$days)[levels(open_hours$days) == "Sat;"] <- "Saturday"
levels(open_hours$days)[levels(open_hours$days) == "Sun;"] <- "Sunday"
```

This is a tough one to analyse and organize, as the open hours depend on days... We'll see if we will need this data for anything...

1.4 Parking

- Information about the kind of park available.

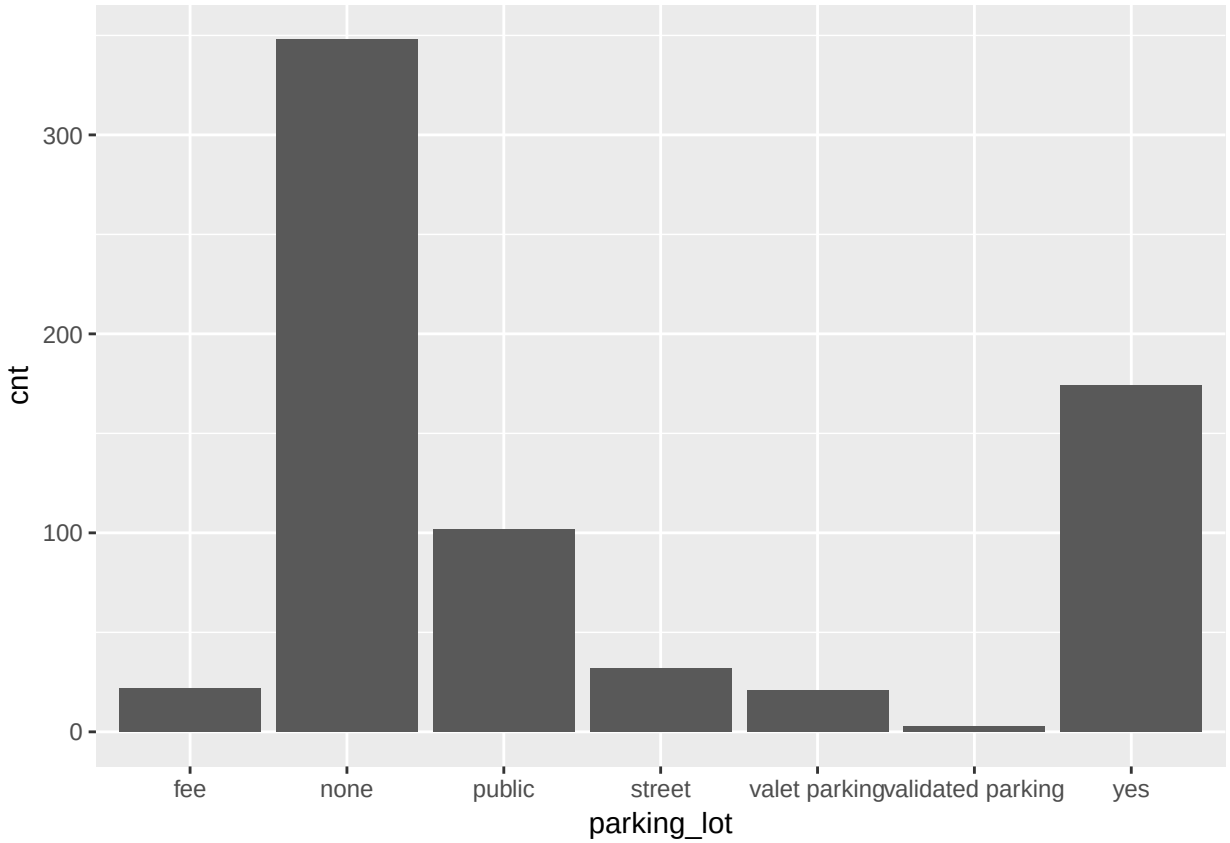
```r
# see a few lines of data
head(parking)
```

```
##   placeID parking_lot
## 1  135111      public
## 2  135110        none
## 3  135109        none
## 4  135108        none
## 5  135107        none
## 6  135106        none
```

```r
# get a summary of the "parking_lot" attribute
summary(parking$parking_lot)
```

```
##               fee              none            public            street
##                22               348               102                32
##     valet parking validated parking               yes
##                21                 3               174
```

```r
# plot the frequencies
parking %>% group_by(parking_lot) %>% summarise(cnt = n()) %>% arrange(desc(cnt)) %>%
  ggplot(aes(x=parking_lot,y=cnt)) + geom_bar(stat="identity")
```



Well, it seems like most of the restaurants doesn't care about where their customers park their cars. This one, we believe, have a slight impact on the rating... We'll see.

1.5 Restaurant features

- A lot of information about restaurants, from location to its features.

As we already know that all restaurants are in Mexico, we start by removing the country attribute. Also, due to a lot of unkown data, and for memory efficiency, we also remove other attributes we consider to not be important.

```
# get a summary of the data
summary(res_info)
```

```
##     placeID          latitude       longitude
##  Min.   :132560   Min.   :18.86   Min.   :-101.03
##  1st Qu.:132831   1st Qu.:22.14   1st Qu.:-100.99
##  Median :134994   Median :22.15   Median :-100.96
##  Mean   :134013   Mean   :21.86   Mean   :-100.34
##  3rd Qu.:135051   3rd Qu.:22.16   3rd Qu.: -99.22
##  Max.   :135109   Max.   :23.76   Max.   : -99.13
##
##                                                the_geom_meter
##  0101000020957F000000DD3546816E5AC119D4BD17FD544A41:  1
##  0101000020957F000003B195E25F8457C1C535BD04614B4941:  1
```

```
##  0101000020957F000004457BB7AA8657C15F10835CD9444941:  1
##  0101000020957F000005810F19B84858C136805B2745A74B41:  1
##  0101000020957F00000B6735CA004858C108FD525CB2A44B41:  1
##  0101000020957F00000F14BF6B2C8657C1963CCB8E5C464941:  1
##  (Other)                                            :124
##                          name                    address
##  Gorditas Dona Tota       :  2   ?                  :27
##  Abondance Restaurante Bar:  1   Ricardo B. Anaya   : 3
##  Arrachela Grill          :  1   Av. V. Carranza    : 2
##  Cabana Huasteca          :  1   venustiano carranza: 2
##  cafe ambar               :  1   16 de Septiembre   : 1
##  Cafe Chaires             :  1   1a. de Lozada 1    : 1
##  (Other)                  :123   (Other)            :94
##               city                state      country   fax
##  San Luis Potosi:64   SLP            :50   ?     :28   ?:130
##  ?              :18   Morelos        :19   mexico:13
##  Cuernavaca     :15   ?              :18   Mexico:89
##  victoria       :10   San Luis Potosi:14
##  san luis potosi: 5   tamaulipas     : 9
##  Jiutepec       : 4   Tamaulipas     : 7
##  (Other)        :14   (Other)        :13
##       zip                  alcohol          smoking_area    dress_code
##  ?      :74   Full_Bar         : 9   none         :70    casual  : 10
##  78000  :13   No_Alcohol_Served:87   not permitted:25    formal  :  2
##  78250  : 3   Wine-Beer        :34   only at bar  : 2    informal:118
##  78269  : 3                          permitted    : 9
##  62290  : 2                          section      :24
##  78210  : 2
##  (Other):33
##           accessibility    price                       url
##  completely      :45    high  :25   ?                    :116
##  no_accessibility:76    low   :45   lacantinaslp.com     :  2
##  partially       : 9    medium:60   carlosandcharlies.com:  1
##                                     chilis.com.mx        :  1
##                                     eloceanodorado.com   :  1
##                                     kikucuernavaca.com.mx:  1
##                                     (Other)              :  8
##     Rambience   franchise     area      other_services
##  familiar:121   f:108     closed:115   Internet:  4
##  quiet   :  9   t: 22     open  : 15   none    :119
##                                       variety :  7
##
##
##
##
```

This is the first troubled dataset as there are a few things missing:
1. address,
2. city,
3. state,
4. country,
5. fax,
6. zip,
7. url.

We are not able to fill in the data, using for instance the K-Nearest neighbors imputation, as each missing information if specific a restaurant; it would work well, but it would get inconsistent with the truth- and we speak the truth. So, as we have the power of geolocalization we decided to try to fill in the missing values from each restaurant' coordinates, using Google Maps. However, this service is very limiting in terms of the dayly allowed queries, and we needed to make, at least, 130, which was simply not possible to be deterministic: sometimes it worked, sometimed it did not. Also, this missing information we think isn't that important for our analysis (only might have impact on the analysis refering to the distance between users and restaurants, but even for this we could use the coordinates)- for that reason, we remove these attributes. Intuitively, we don't think that a restaurant is better for having a fax machine either...

```
# attributes removal
res_info <- res_info[, !names(res_info) %in% c("address","city","state","country","zip","fax","url")]
```

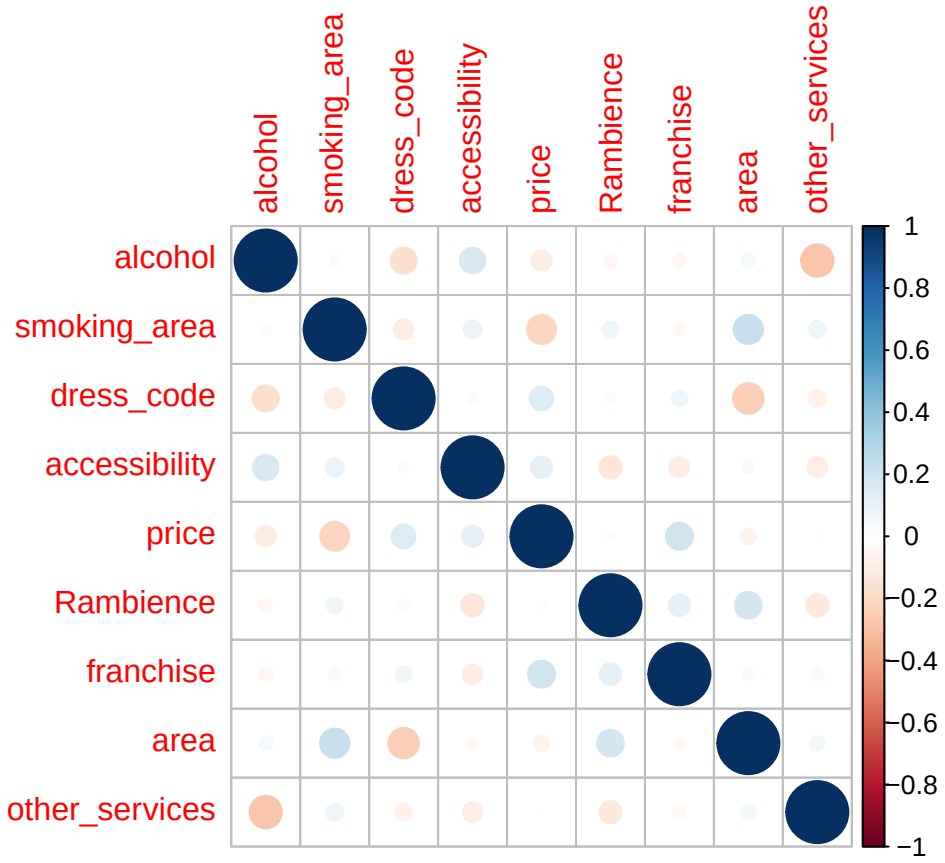
Now for the important features, we try to squeeze its contents, and see the correlation between all of them.

```
# get a summarization of the features the restaurants
describe(res_info[,6:ncol(res_info)])
```

```
## res_info[, 6:ncol(res_info)]
##
##  9  Variables      130  Observations
## ---------------------------------------------------------------------------
## alcohol
##        n  missing distinct
##      130        0        3
##
## Value               Full_Bar No_Alcohol_Served         Wine-Beer
## Frequency                  9                87                34
## Proportion             0.069             0.669             0.262
## ---------------------------------------------------------------------------
## smoking_area
##        n  missing distinct
##      130        0        5
##
## Value               none not permitted   only at bar     permitted
## Frequency             70            25             2             9
## Proportion         0.538         0.192         0.015         0.069
##
## Value            section
## Frequency             24
## Proportion         0.185
## ---------------------------------------------------------------------------
## dress_code
##        n  missing distinct
##      130        0        3
##
## Value        casual   formal informal
## Frequency        10        2      118
## Proportion    0.077    0.015    0.908
## ---------------------------------------------------------------------------
## accessibility
##        n  missing distinct
##      130        0        3
##
## Value            completely no_accessibility        partially
## Frequency                45               76                9
```

```
## Proportion            0.346           0.585           0.069
## ---------------------------------------------------------------------------
## price
##        n  missing distinct
##      130        0        3
##
## Value        high    low medium
## Frequency      25     45     60
## Proportion  0.192  0.346  0.462
## ---------------------------------------------------------------------------
## Rambience
##        n  missing distinct
##      130        0        2
##
## Value      familiar    quiet
## Frequency       121        9
## Proportion    0.931    0.069
## ---------------------------------------------------------------------------
## franchise
##        n  missing distinct
##      130        0        2
##
## Value          f     t
## Frequency    108    22
## Proportion 0.831 0.169
## ---------------------------------------------------------------------------
## area
##        n  missing distinct
##      130        0        2
##
## Value      closed   open
## Frequency     115     15
## Proportion  0.885  0.115
## ---------------------------------------------------------------------------
## other_services
##        n  missing distinct
##      130        0        3
##
## Value      Internet     none  variety
## Frequency         4      119        7
## Proportion    0.031    0.915    0.054
## ---------------------------------------------------------------------------
```

```r
# get a correlation plot to easily observe correlations influences between attributes
res_info_cor_matrix <- cor(data.matrix(res_info[,6:ncol(res_info)]))
corrplot(res_info_cor_matrix)
```



Some of the correlations between the attributes (the connection between variables):
* “area” and “dress_code”: a negative one, meaning, one discourage the other;
* “area” and “smoking_area”: a positive one, they go along;
* “franchise” and “price”: a positive one.

And we also factorize the categorical variables presented in this dataframe, for efficiency:

```
res_info$price <- factor(res_info$price, levels=c("low","medium","high"))
res_info$smoking_area <- factor(res_info$smoking_area, levels=c("none","not permitted",
                                                                "permitted","section","only at bar"))
res_info$accessibility <-factor(res_info$accessibility, levels=c("no_accessibility",
                                                                 "partially","completely"))
```

1.6 User cuisine preferences

- States the preferences of the users for the cuisine kind.

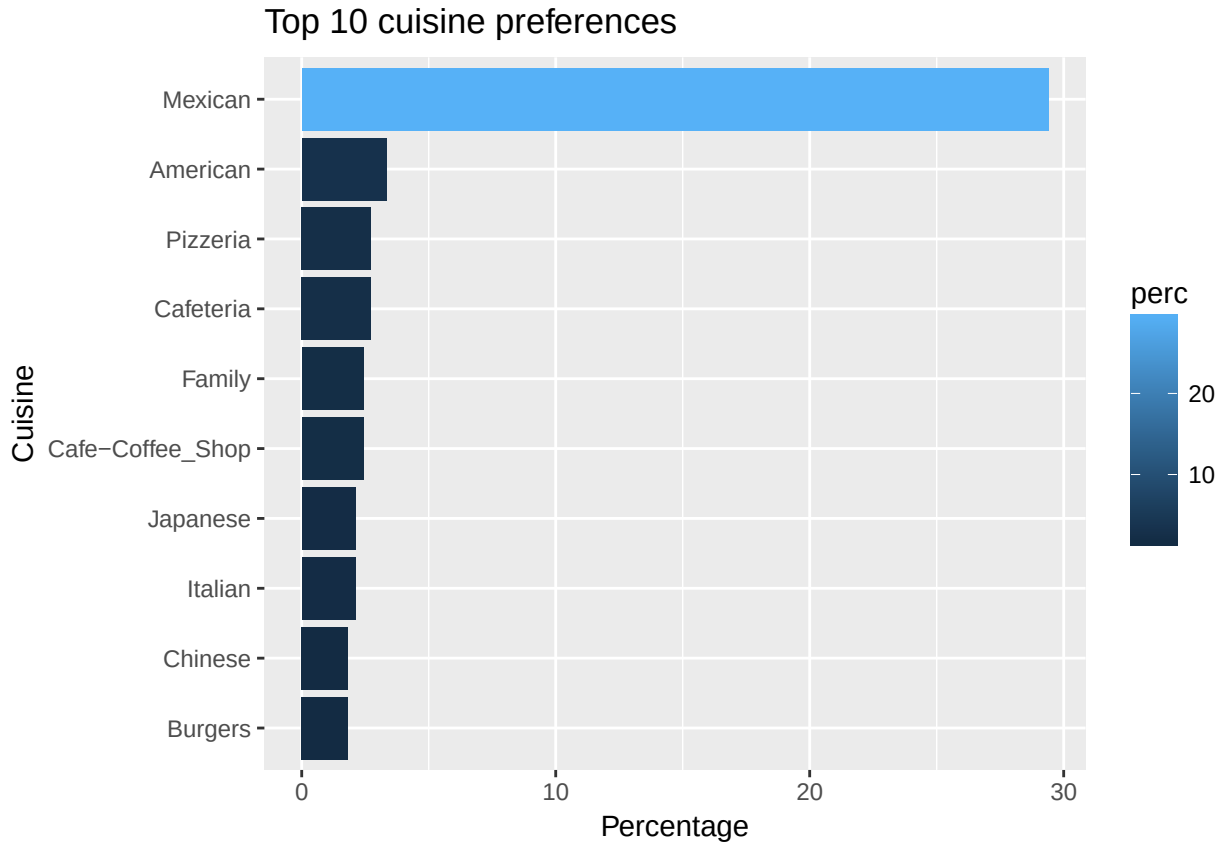
```
# see a few lines
head(user_cuisine$Rcuisine)
```

```
## [1] American         Mexican          Mexican          Bakery
## [5] Breakfast-Brunch Japanese
## 103 Levels: Afghan African American Armenian Asian Australian ... Vietnamese
```

```
# get a summary of the data
summary(user_cuisine)
```

```
##      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
```



The vast majority of users prefer Mexican food.

1.7 User prefered payment method

- Prefered payment method of users.

```
# see a few lines
head(user_payment)
```

```
##   userID         Upayment
## 1  U1001             cash
## 2  U1002             cash
## 3  U1003             cash
## 4  U1004             cash
## 5  U1004 bank_debit_cards
## 6  U1005             cash
```

```
# see a quick summary
summary(user_payment)
```

```
##      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
```

Here, we do the same treatment we did in the payment methods section referring to the restaurants.

```
Debit_Credit_cards<-c("VISA","MasterCard-Eurocard","American_Express","bank_debit_cards",
                      "Visa", "Japan_Credit_Bureau", "Discover", "Diners_Club")
levels(user_payment$Upayment)[levels(user_payment$Upayment) %in%
                                Debit_Credit_cards] <- "Debit_Credit_cards"
levels(user_payment$Upayment) <- capFirst(levels(user_payment$Upayment))

user_payment <- unique(user_payment[1:nrow(user_payment),])

levels(user_payment$Upayment)
```

```
## [1] "Debit_Credit_cards" "Cash"
```

```
# check the relative frequencies of the new category-organized data
user_payment %>% group_by(Upayment) %>% summarise(cnt = n()) %>% arrange(desc(cnt))
```

```
## # A tibble: 2 x 2
##   Upayment             cnt
##   <fct>              <int>
## 1 Cash                 131
## 2 Debit_Credit_cards    35
```

Cash ahead!

1.8 User information

- Similar to what we have on restaurants, we also have a few information on users.

```
# get a quick summary of the raw data
summary(user_info)
```

```
##      userID       latitude       longitude         smoker
##  U1001  :  1   Min.   :18.81   Min.   :-101.05   false:109
##  U1002  :  1   1st Qu.:22.13   1st Qu.:-100.98   true : 26
##  U1003  :  1   Median :22.15   Median :-100.94   NA's :  3
##  U1004  :  1   Mean   :21.81   Mean   :-100.29
##  U1005  :  1   3rd Qu.:22.19   3rd Qu.: -99.18
##  U1006  :  1   Max.   :23.77   Max.   : -99.07
##  (Other):132
##          drink_level      dress_preference     ambience      transport
##  abstemious    :51   no preference:53      ?       : 6   on foot  :14
##  casual drinker:47   informal     :35      family  :70   public   :82
##  social drinker:40   formal       :41      friends :46   car owner:35
##                      elegant      : 4      solitary:16   NA's     : 7
##                      NA's         : 5
##
```

```
##
##  marital_status         hijos       birth_year           interest
##  ?      :  4    ?          : 11   Min.   :1930   eco-friendly:16
##  married: 10    dependent  :  3   1st Qu.:1987   none        :30
##  single :122    independent:113   Median :1989   retro       : 6
##  widow  :  2    kids       : 11   Mean   :1985   technology  :36
##                                   3rd Qu.:1991   variety     :50
##                                   Max.   :1994
##
##              personality      religion           activity       color
##  conformist         : 7   Catholic :99   ?            :  7   blue   :45
##  hard-worker        :61   Christian: 7   professional : 15   black  :21
##  hunter-ostentatious:12   Jewish   : 1   student      :113   green  :19
##  thrifty-protector  :58   Mormon   : 1   unemployed   :  2   red    :15
##                           none     :30   working-class:  1   yellow :12
##                                                             purple :11
##                                                             (Other):15
##      weight          budget       height
##  Min.   : 40.00   low   :35   Min.   :1.200
##  1st Qu.: 53.00   medium:91   1st Qu.:1.600
##  Median : 65.00   high  : 5   Median :1.690
##  Mean   : 64.87   NA's  : 7   Mean   :1.668
##  3rd Qu.: 74.75               3rd Qu.:1.750
##  Max.   :120.00               Max.   :2.000
##
```

Attributes missing information:
1. smoker,
2. dress_preference,
3. ambience,
4. transport,
5. marital_status,
6. hijos,
7. activity,
9. budget.

Oh, man... A lot of missing information. In this case, the situation is different from the restaurants, because the users are our actors, and we need to correlate the available features of the restaurants with the features of the users. For this purpose, we now do a KNN imputation, meaning, we are going to use information available to fill in the missing one, taking into account similar users.

```
#user_info <- read.csv("RCdata/userprofile.csv")

complete.cases(user_info)
```

```
##   [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [12]  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [23]  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE
##  [34]  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [45]  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [56]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [67]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [78]  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE FALSE  TRUE  TRUE  TRUE
##  [89]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [100]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [111]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
```

```
## [122] FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE FALSE
## [133]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
```

```
# tranform all '?' into a NA, so the knnImputation() can identify the missing values
user_info[user_info == '?'] <- NA

levels(user_info$smoker)[match("?",levels(user_info$smoker))] <- "NA"
levels(user_info$dress_preference)[match("?",levels(user_info$dress_preference))] <- "NA"
levels(user_info$ambience)[match("?",levels(user_info$ambience))] <- "NA"
levels(user_info$transport)[match("?",levels(user_info$transport))] <- "NA"
levels(user_info$marital_status)[match("?",levels(user_info$marital_status))] <- "NA"
levels(user_info$hijos)[match("?",levels(user_info$hijos))] <- "NA"
levels(user_info$activity)[match("?",levels(user_info$activity))] <- "NA"
levels(user_info$budget)[match("?",levels(user_info$budget))] <- "NA"

complete.cases(user_info)
```

```
##   [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [12]  TRUE  TRUE FALSE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [23]  TRUE FALSE FALSE  TRUE FALSE  TRUE  TRUE  TRUE FALSE  TRUE FALSE
##  [34]  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [45]  TRUE  TRUE  TRUE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [56]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE
##  [67]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [78]  TRUE FALSE  TRUE  TRUE FALSE FALSE  TRUE FALSE  TRUE  TRUE  TRUE
##  [89]  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [100]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE
## [111]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [122] FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE FALSE
## [133]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
```

```
a <- knnImputation(user_info, k=1)

#voltam a aparecer os '?'
summary(a)
```

```
##      userID       latitude       longitude         smoker
##  U1001  :  1   Min.   :18.81   Min.   :-101.05   false:112
##  U1002  :  1   1st Qu.:22.13   1st Qu.:-100.98   true : 26
##  U1003  :  1   Median :22.15   Median :-100.94
##  U1004  :  1   Mean   :21.81   Mean   :-100.29
##  U1005  :  1   3rd Qu.:22.19   3rd Qu.: -99.18
##  U1006  :  1   Max.   :23.77   Max.   : -99.07
##  (Other):132
##         drink_level      dress_preference     ambience      transport
##  abstemious    :51   no preference:58      NA      : 6   on foot  :21
##  casual drinker:47   informal     :35      family  :70   public   :82
##  social drinker:40   formal       :41      friends :46   car owner:35
##                      elegant      : 4      solitary:16
##
##
##
##  marital_status         hijos       birth_year           interest
##  NA     :  4    NA         : 11   Min.   :1930   eco-friendly:16
##  married: 10    dependent  :  3   1st Qu.:1987   none        :30
```

```
##  single :122    independent:113   Median :1989   retro       : 6
##  widow  :  2    kids       : 11   Mean   :1985   technology  :36
##                                   3rd Qu.:1991   variety     :50
##                                   Max.   :1994
##
##              personality      religion           activity       color
##  conformist         : 7   Catholic :99   NA           :  7   blue   :45
##  hard-worker        :61   Christian: 7   professional : 15   black  :21
##  hunter-ostentatious:12   Jewish   : 1   student      :113   green  :19
##  thrifty-protector  :58   Mormon   : 1   unemployed   :  2   red    :15
##                           none     :30   working-class:  1   yellow :12
##                                                             purple :11
##                                                             (Other):15
##      weight          budget       height
##  Min.   : 40.00   low   :42   Min.   :1.200
##  1st Qu.: 53.00   medium:91   1st Qu.:1.600
##  Median : 65.00   high  : 5   Median :1.690
##  Mean   : 64.87               Mean   :1.668
##  3rd Qu.: 74.75               3rd Qu.:1.750
##  Max.   :120.00               Max.   :2.000
##
```

1.9 Ratings

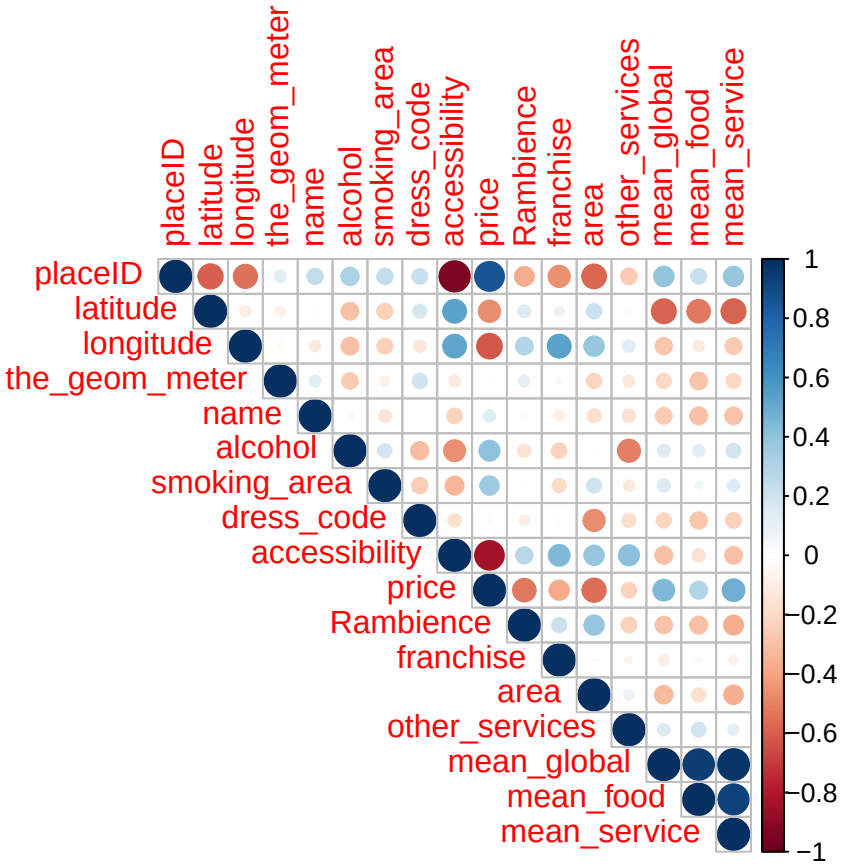
- Contains the evaluations of users on restaurants.

In theory, there should only be one rate for the combinations of user and restaurant.

```
rating %>% group_by(userID, placeID) %>% count() %>% arrange(desc(n)) %>% filter(n>1)
```

```
## # A tibble: 0 x 3
## # Groups:   userID, placeID [0]
## # ... with 3 variables: userID <fct>, placeID <int>, n <int>
```

Now we try to analyse the correlation between ratings and features of restaurants:



There are some strong negative correlations...

Let's now see the numbers on the ratings of the restaurants:

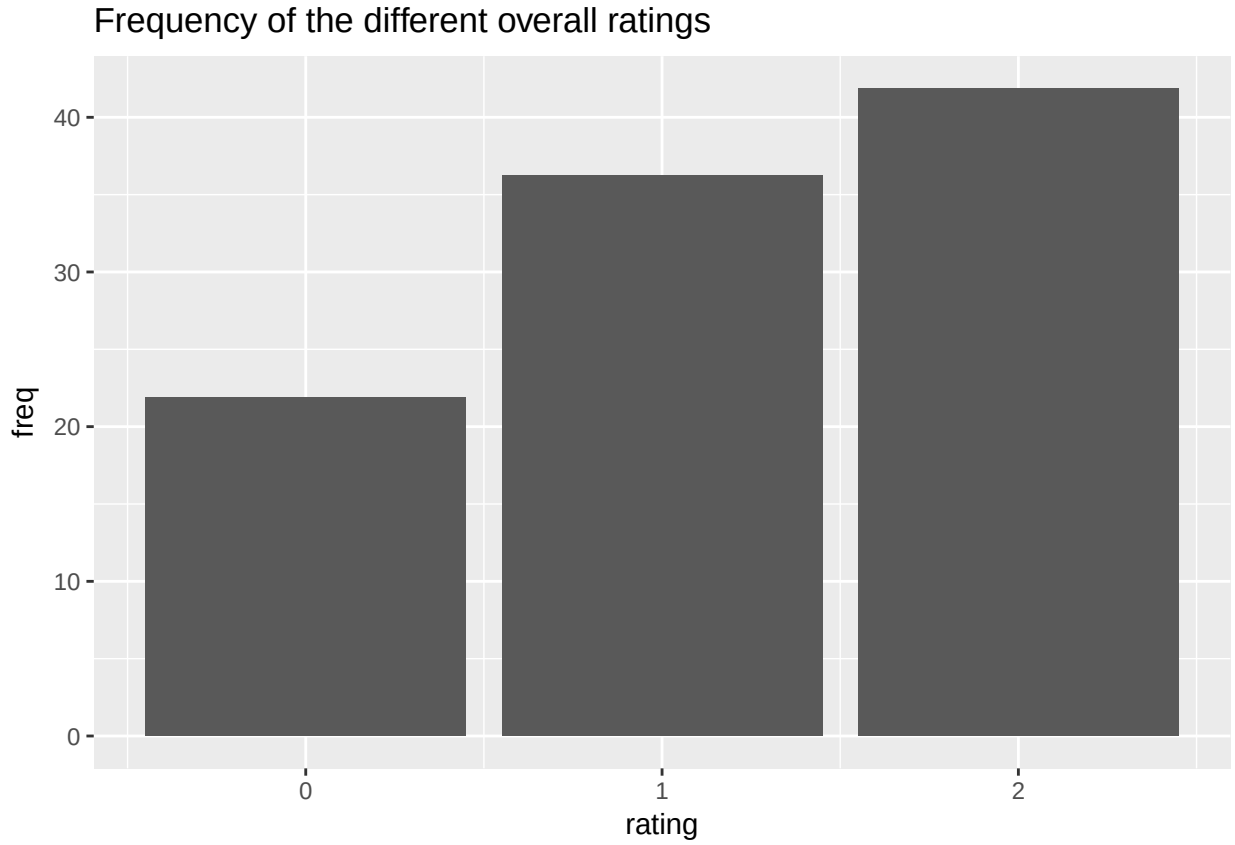
```
# the top 10 most rated
rating %>% group_by(placeID) %>% summarise(cnt = n()) %>% arrange(desc(cnt)) %>% head(10)
```

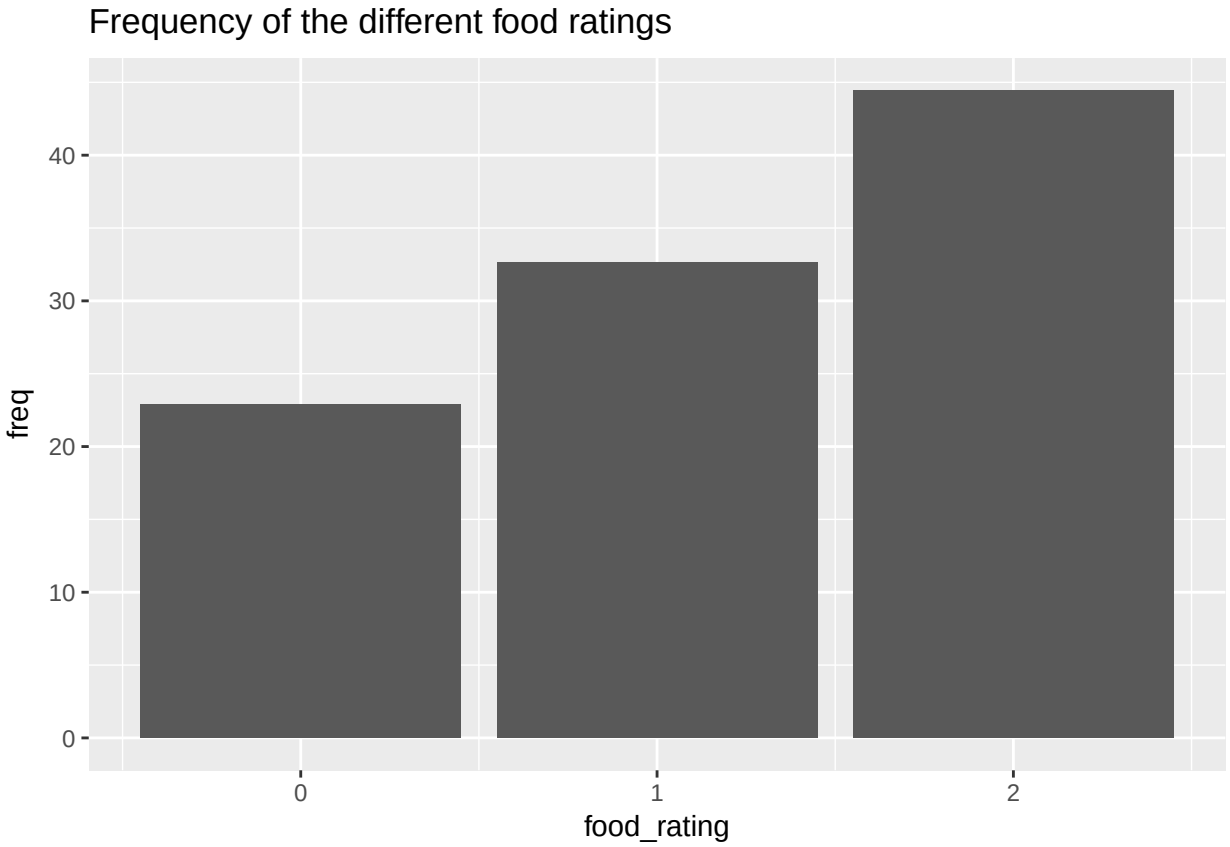
```
## # A tibble: 10 x 2
##    placeID   cnt
##      <int> <int>
##  1  135085    36
##  2  132825    32
##  3  135032    28
##  4  132834    25
##  5  135052    25
##  6  135038    24
##  7  135060    22
##  8  135062    21
##  9  135042    20
## 10  132862    18
```

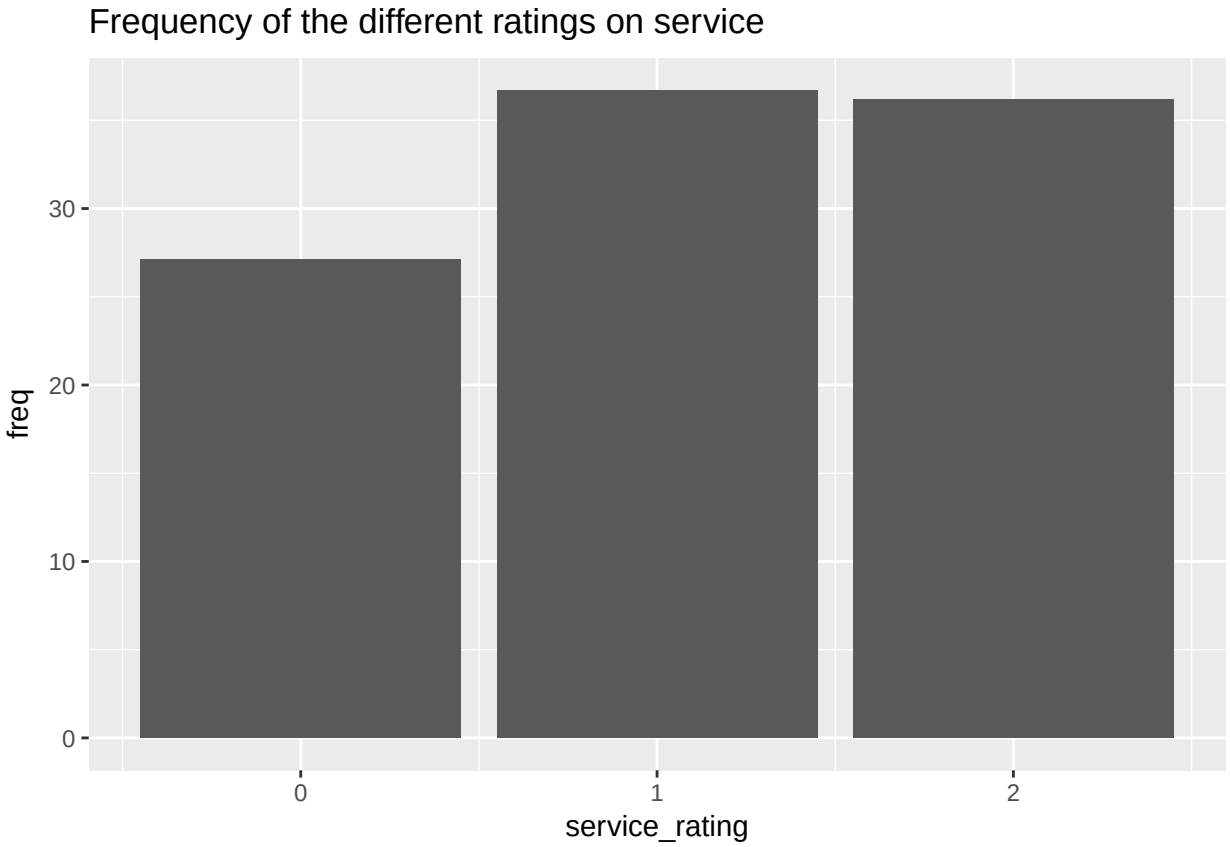
```
rating %>% group_by(placeID) %>% summarise(mean_global = mean(rating), mean_food = mean(food_rating),
                                           mean_service = mean(service_rating)) %>% select(placeID, mean
```

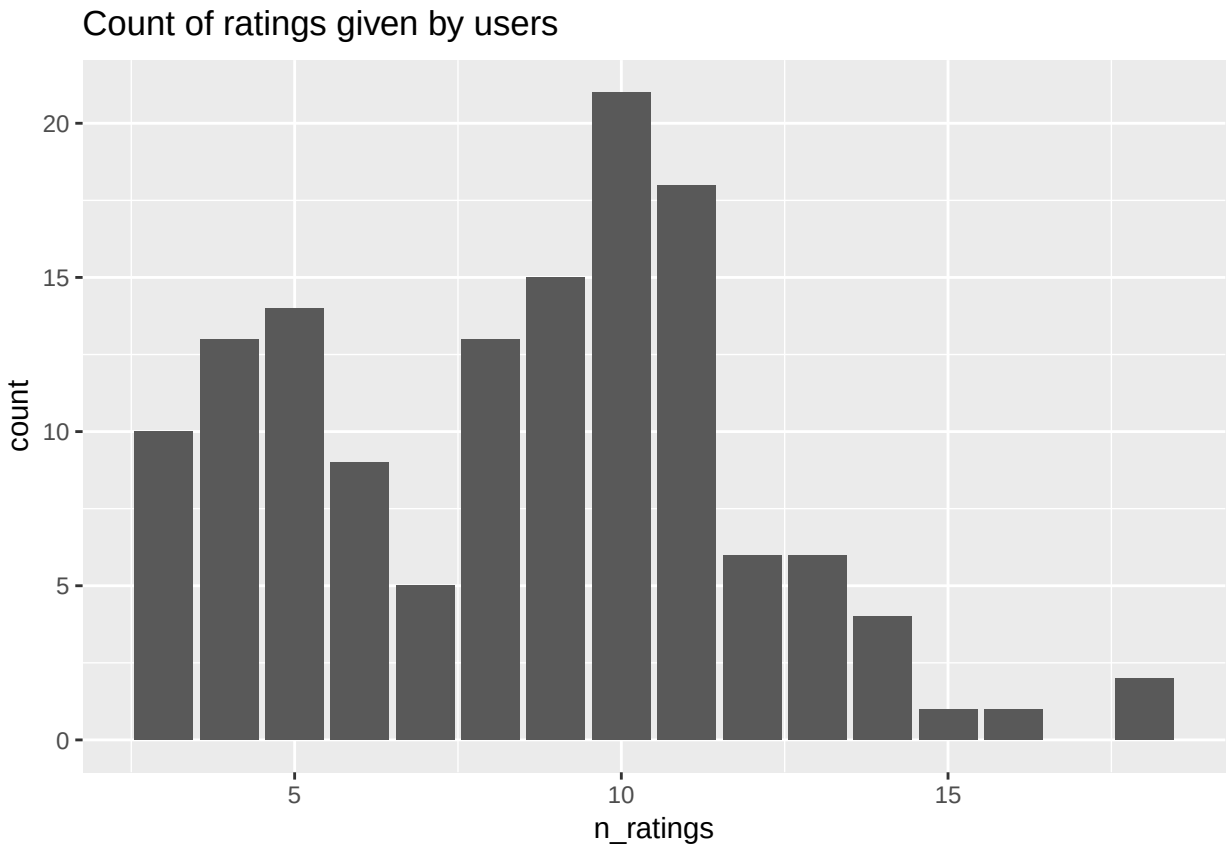
```
## # A tibble: 130 x 4
##    placeID mean_global mean_food mean_service
##      <int>       <dbl>     <dbl>        <dbl>
##  1  135109           1      1.25         0.75
```

```
##  2  135108       1.18      1.18         1
##  3  135106       1.2       1.2          1.2
##  4  135104       0.857     1.43         0.857
##  5  135088       1         1.17         1
##  6  135086       0.8       0.6          0.8
##  7  135085       1.33      1.47         1.17
##  8  135082       0.778     0.778        0.667
##  9  135081       1.18      1.45         1.27
## 10  135080       1         1            1
## # ... with 120 more rows
```




Frequency of the different food ratings
freq
40
30
20
10
0
0
1
2
food_rating


Frequency of the different ratings on service
freq
30
20
10
0
0
1
2
service_rating



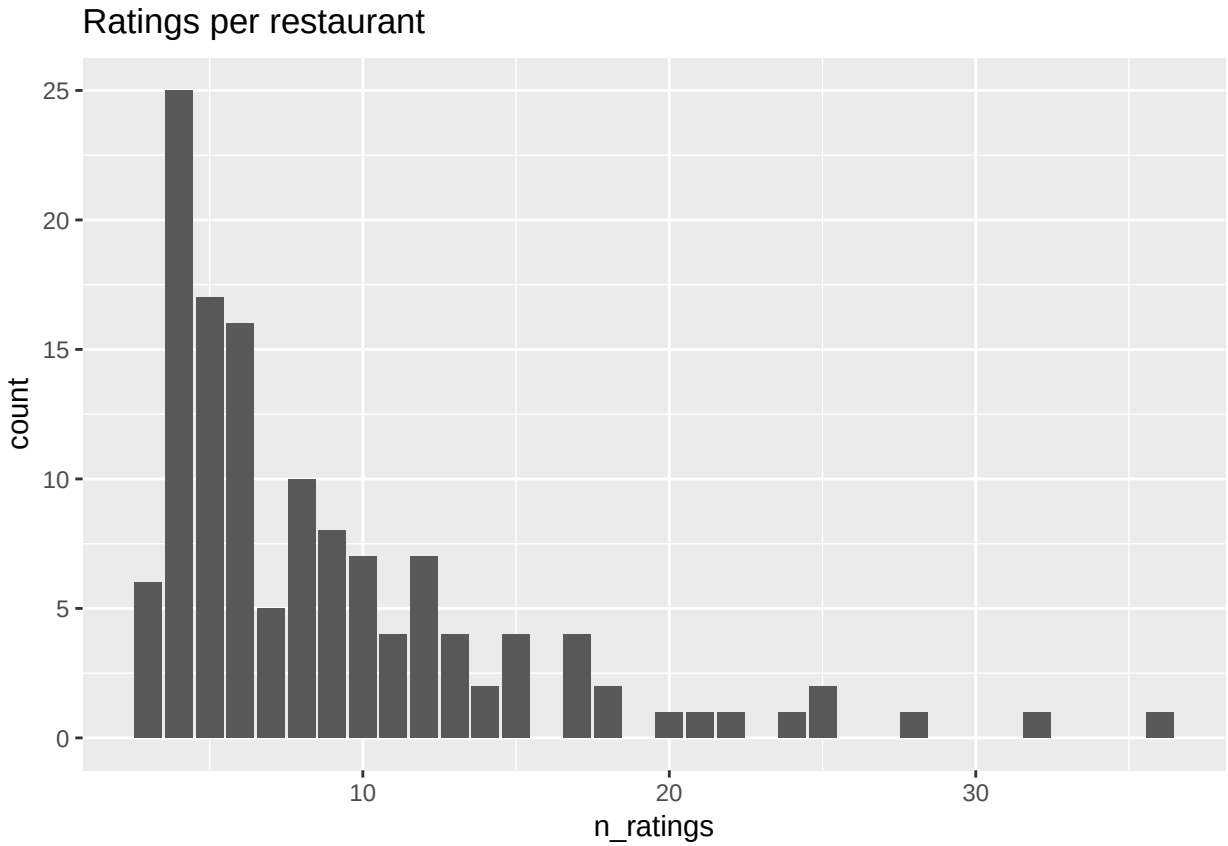
Here we try to find atypical user behaviour (always the same rating).

```
## always 0
atypical_users <- number_ratings_user %>% filter(sd==0)
## always 2
atypical_users %>% filter(mean==2)
```

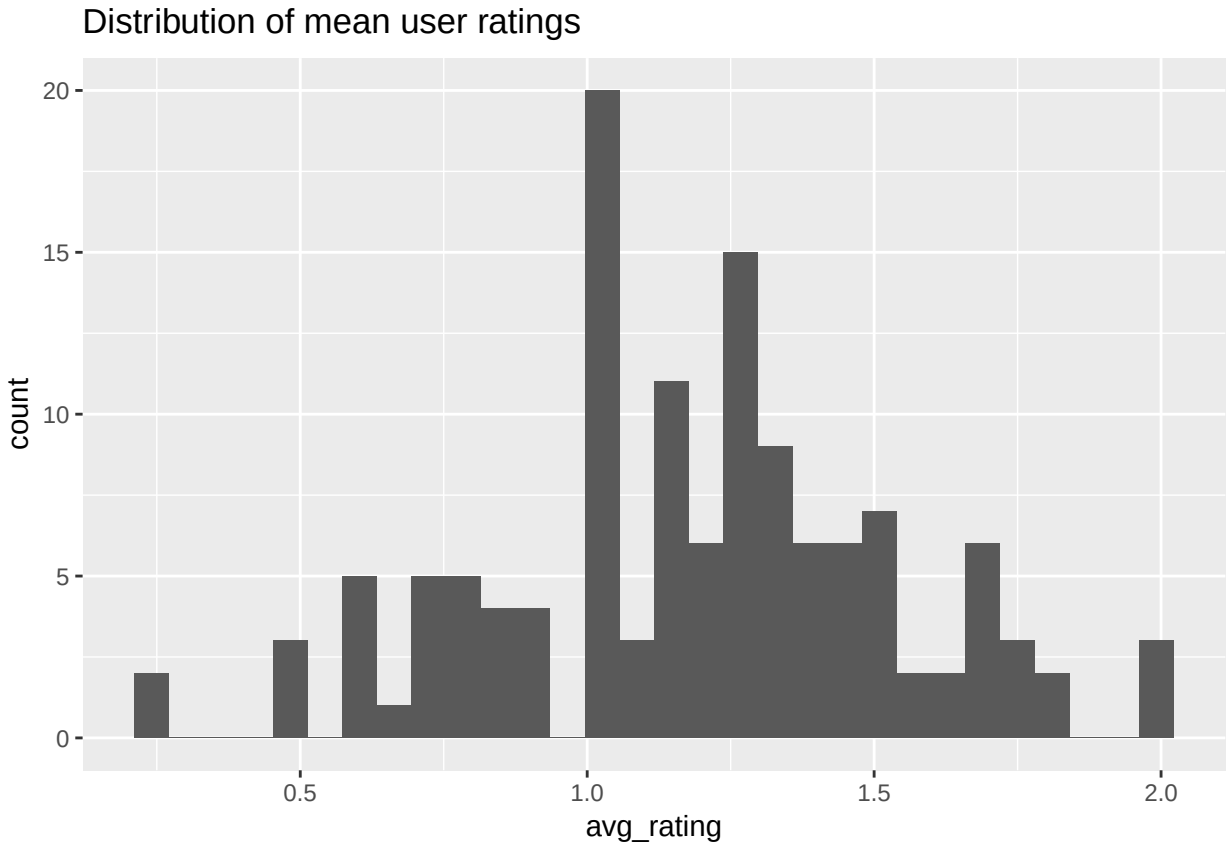
```
## # A tibble: 6 x 6
##   userID n_ratings  mean median    sd   var
##   <fct>      <int> <dbl>  <dbl> <dbl> <dbl>
## 1 U1021          3     2      2     0     0
## 2 U1074          3     2      2     0     0
## 3 U1100          6     2      2     0     0
## 4 U1102          5     2      2     0     0
## 5 U1107          3     2      2     0     0
## 6 U1127          4     2      2     0     0
```

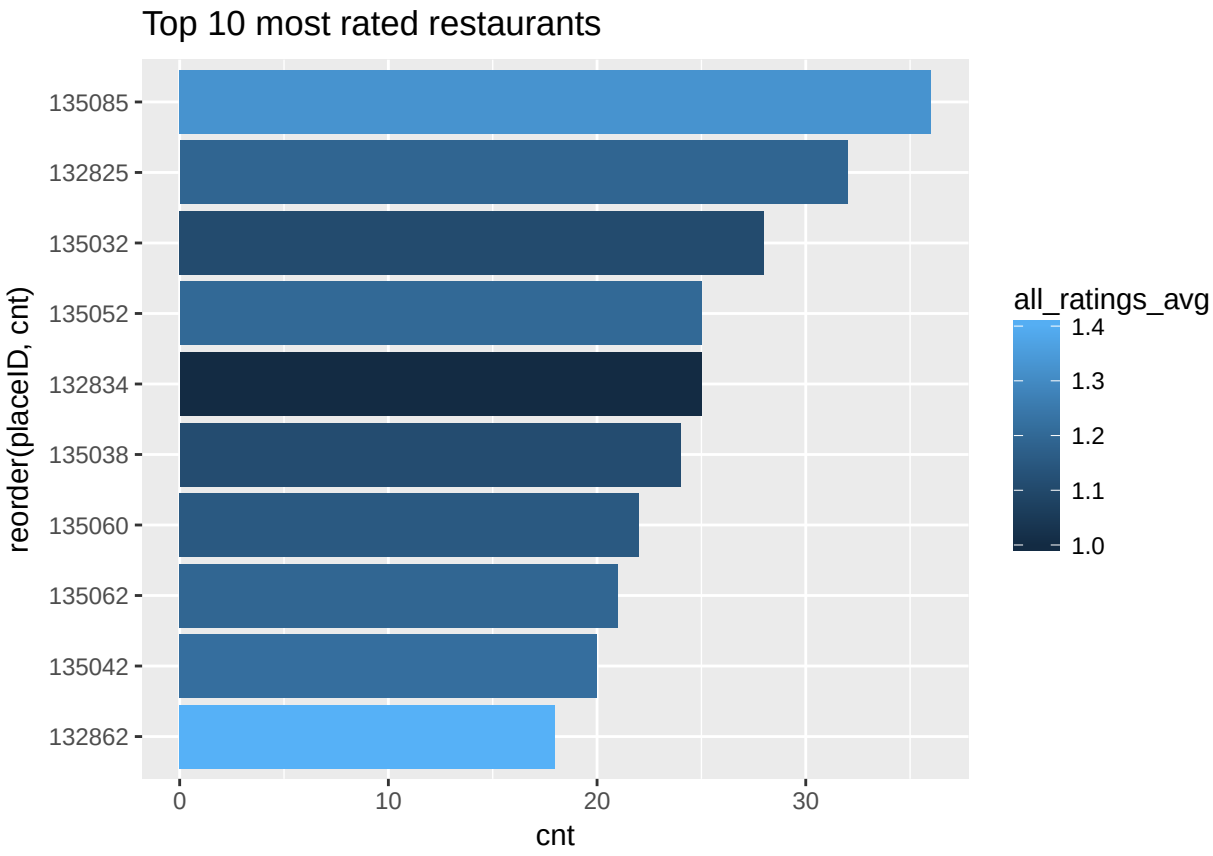
```
## always 1
atypical_users %>% filter(mean==1)
```

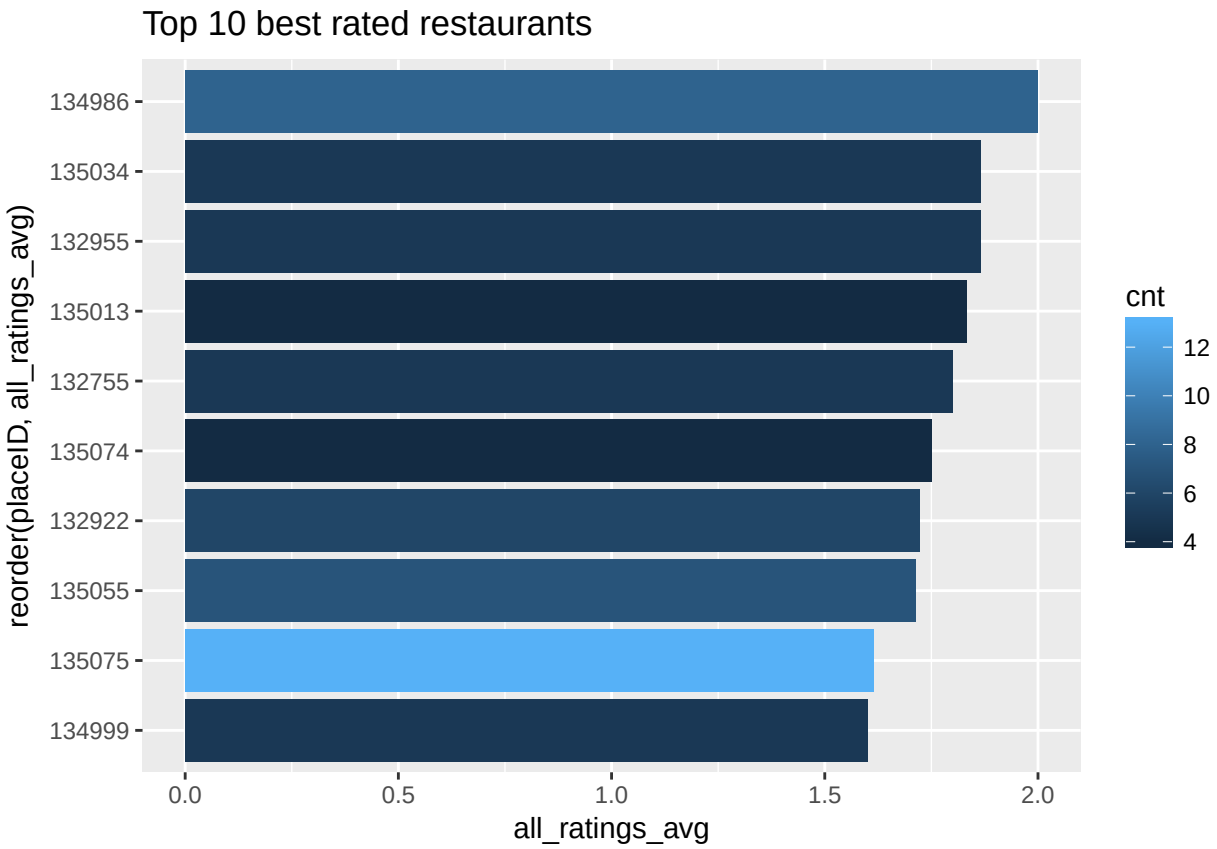
```
## # A tibble: 2 x 6
##   userID n_ratings  mean median    sd   var
##   <fct>      <int> <dbl>  <dbl> <dbl> <dbl>
## 1 U1043          5     1      1     0     0
## 2 U1051          4     1      1     0     0
```

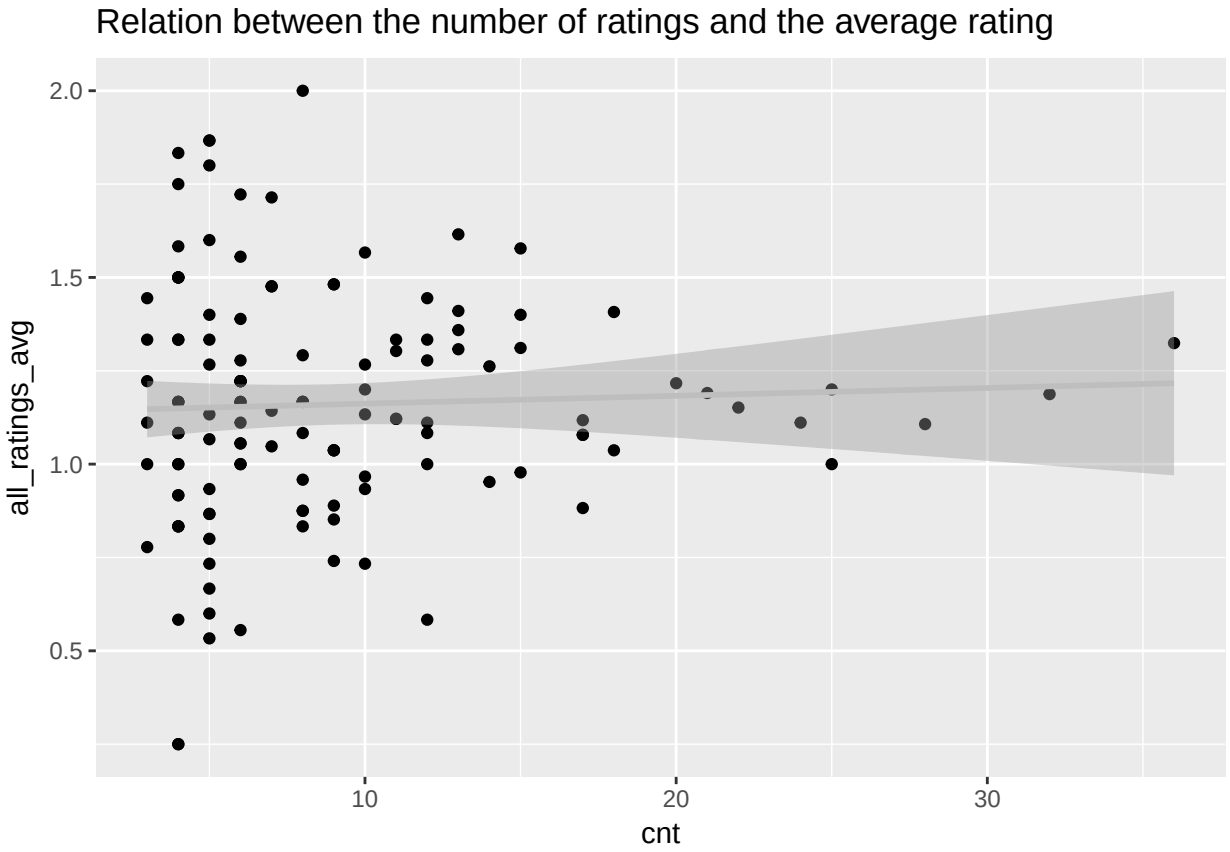



```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```


Distribution of mean user ratings
count
20
15
10
5
0
0.5
1.0
1.5
2.0
avg_rating


Top 10 most rated restaurants
reorder(placeID, cnt)
135085
132825
135032
135052
132834
135038
135060
135062
135042
132862
0
10
20
30
cnt
all_ratings_avg
1.4
1.3
1.2
1.1
1.0


Top 10 best rated restaurants
reorder(placeID, all_ratings_avg)
134986
135034
132955
135013
132755
135074
132922
135055
135075
134999
0.0
0.5
1.0
1.5
2.0
all_ratings_avg
cnt
12
10
8
6
4



It would make sense that the popularity of a restaurant was correlated with its average rating, with popular (more often rated) restaurants having better ratings. But that doesn't seem to be the case.

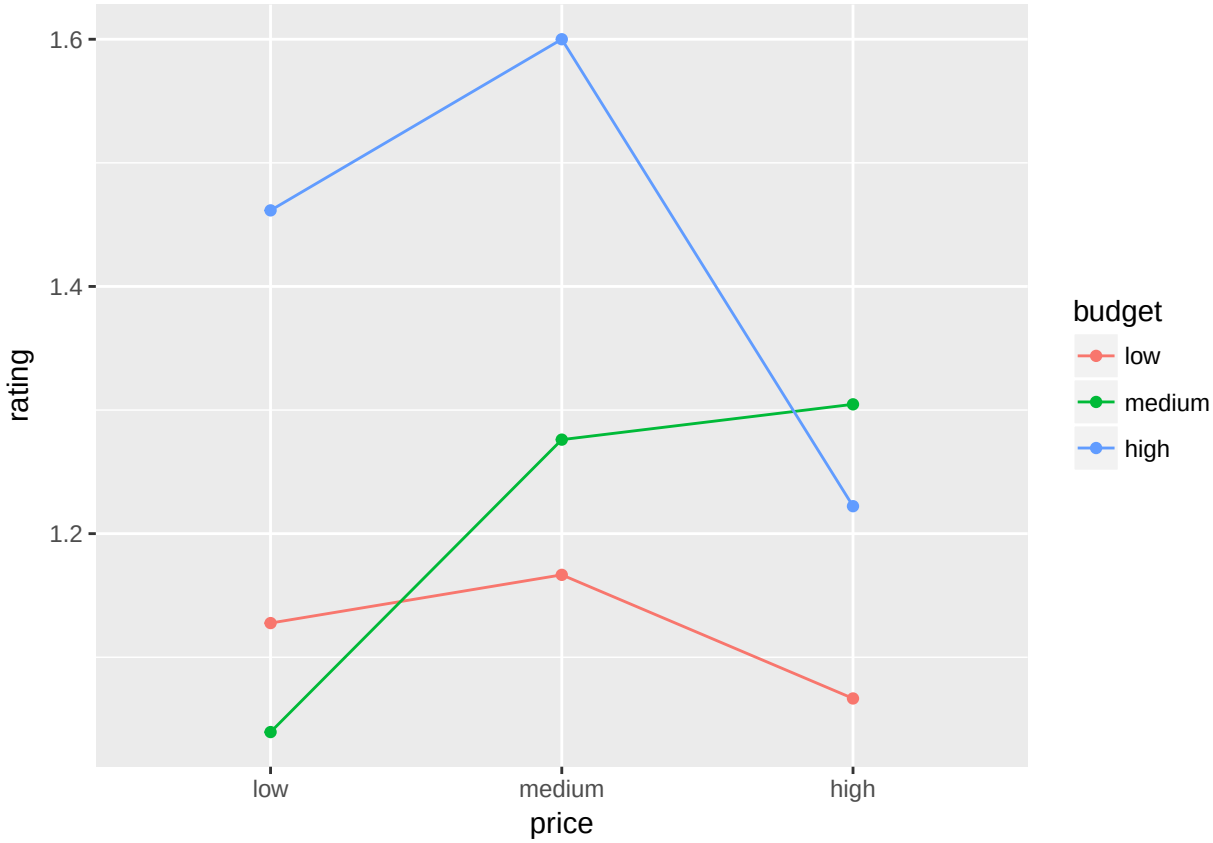
More exepensive restaurants have better ratings.

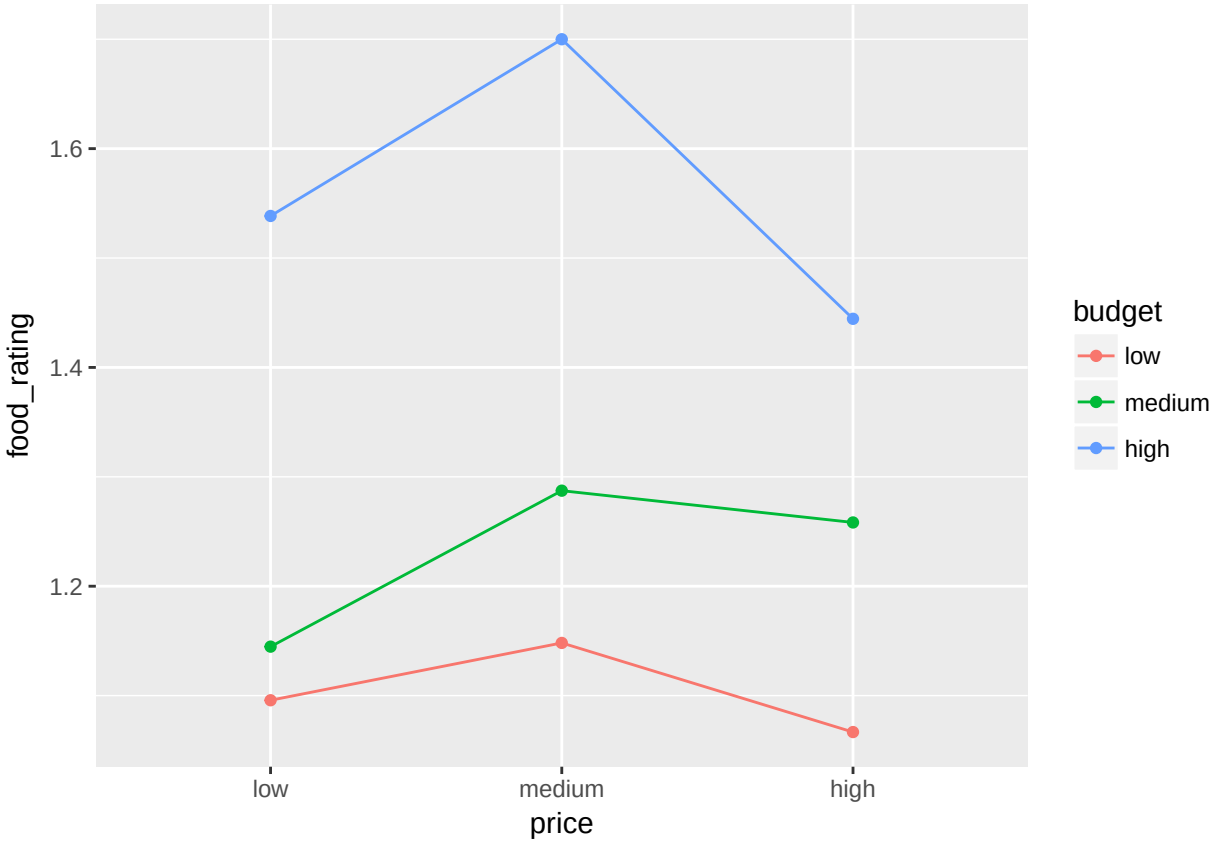
In here we create a function that correlates attributes between users and restaurants, so we are able to see the connection between attributes from the users and their correspondents of the restaurants.

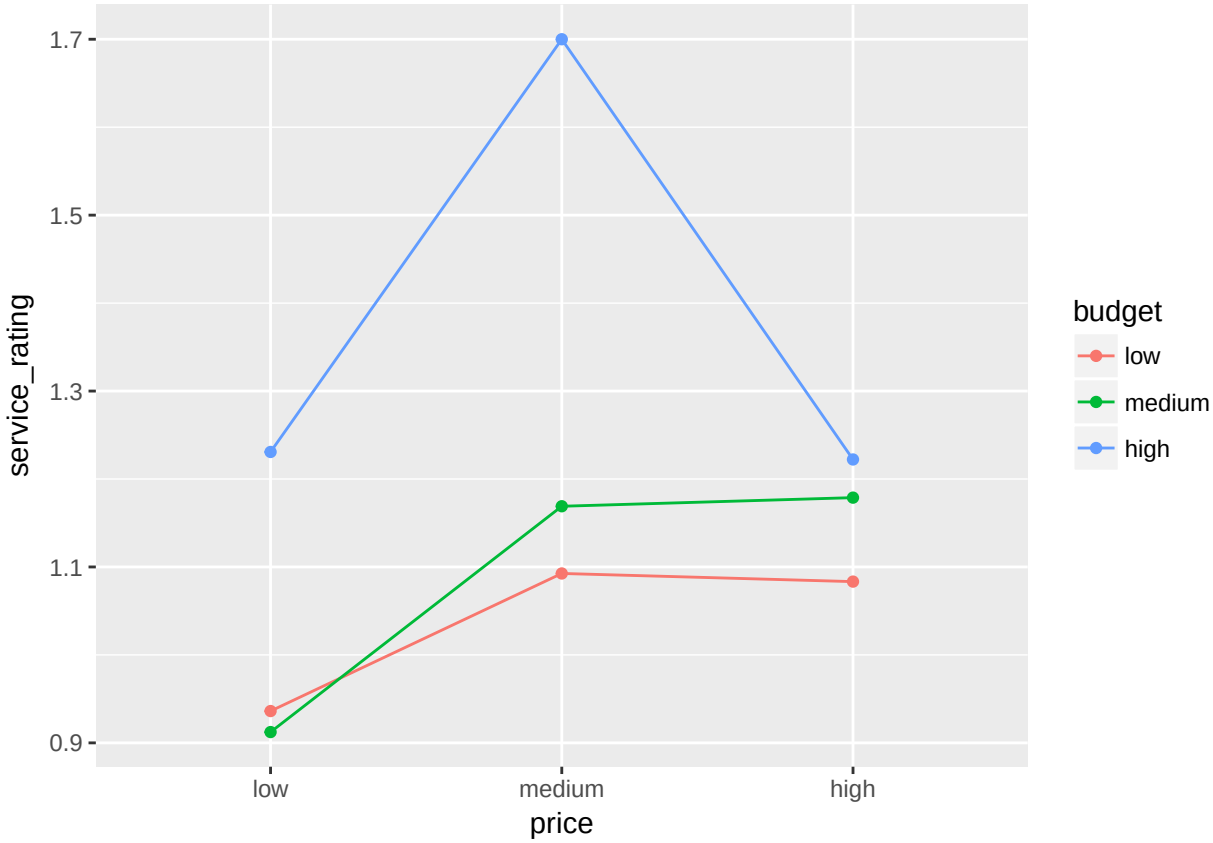
```
ratings_by_groups <- function(res_group, user_group) {
  rating  %>% inner_join(res_info,by="placeID") %>% inner_join(user_info, by="userID") %>%
  select_("userID", "placeID", "rating", "food_rating", "service_rating", res_group, user_group) %>% dr
  group_by_(res_group,user_group) %>% summarise(cnt=n(), rating=mean(rating), food_rating=mean(food_rati
}

rating_by_price_and_budget <- ratings_by_groups("price","budget")
rating_by_smokingArea_and_smokers <- ratings_by_groups("smoking_area", "smoker")
rating_by_alcohol_and_drinkLevel <- ratings_by_groups("alcohol","drink_level")
rating_by_parkingOptions_and_transport <- ratings_by_groups("accessibility","transport")
rating_by_dressCode_and_dressPreference <- ratings_by_groups("dress_code","dress_preference")
```

Ratings on price splited into three types of users

- high budget and low budget users give better ratings to average priced restaurants
- average budget users give better ratings to high priced restaurants

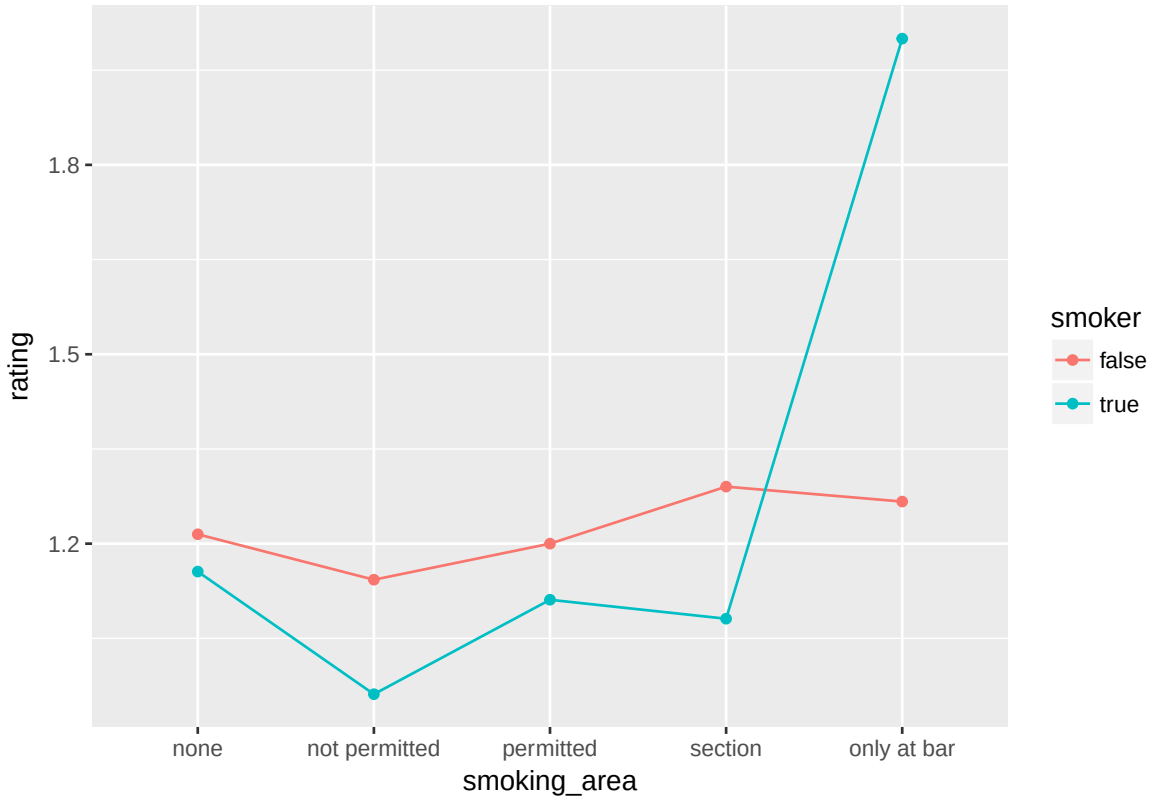

1.6
1.4
1.2
rating
low
medium
high
price
budget
low
medium
high


food_rating
1.6
1.4
1.2
low
medium
high
price
budget
low
medium
high

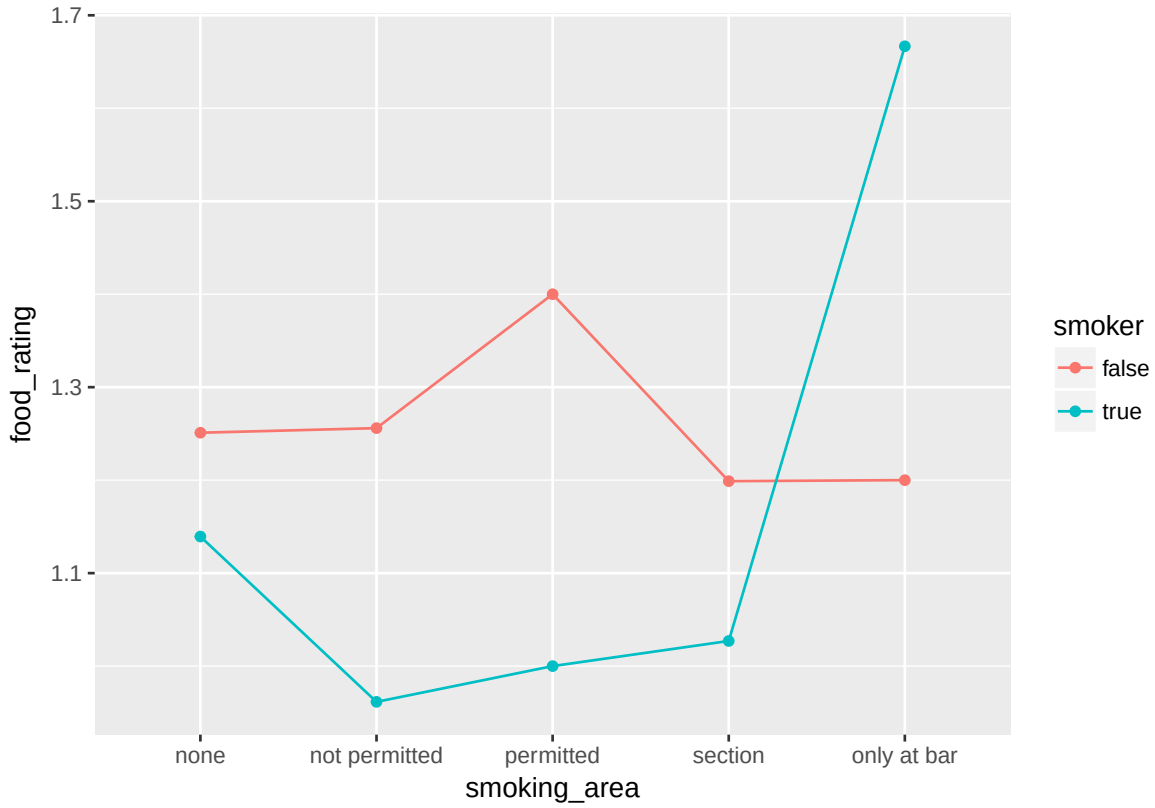


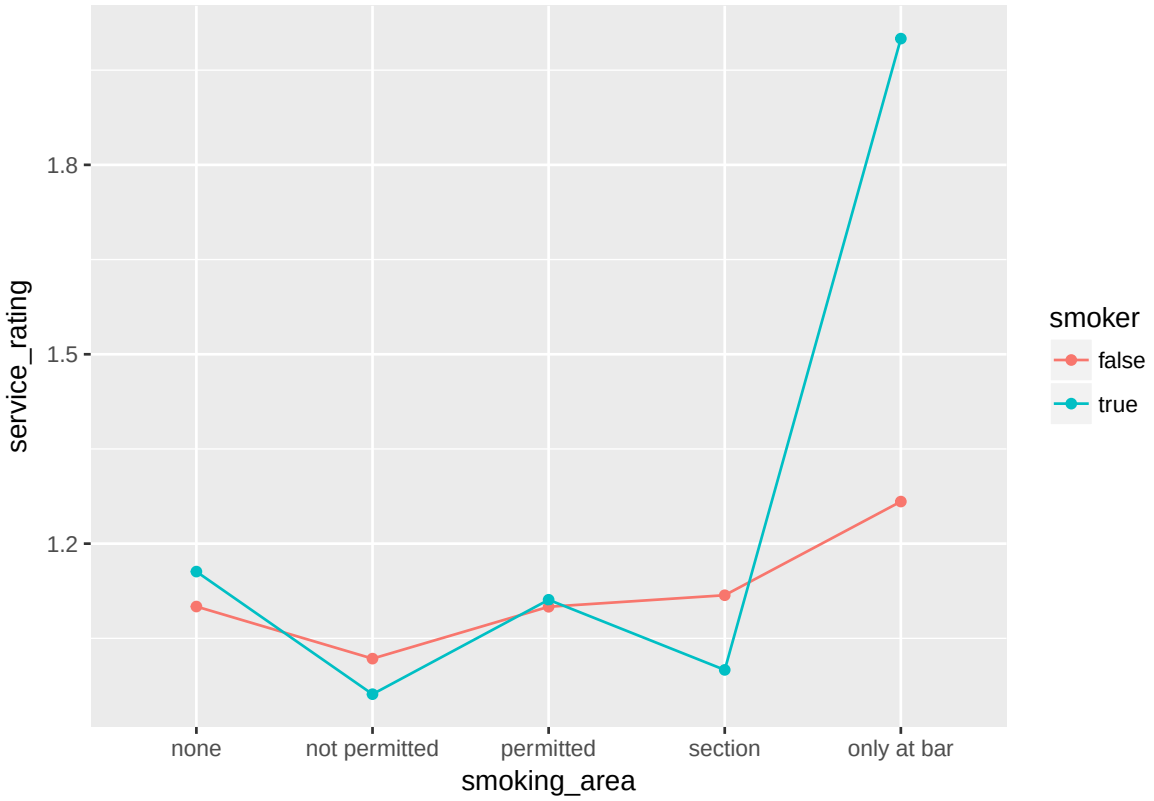
Ratings on smoking area split into two types of users

- Smokers seems to prefer restaurants where they can smoke at the bar.
- Smokers give worse ratings to restaurants where they can't smoke.
- Non-smokers don't seem to change their ratings according to whatever kind of smoking areas restaurants


1.8
1.5
1.2
rating
none
not permitted
permitted
section
only at bar
smoking_area
smoker
false
true

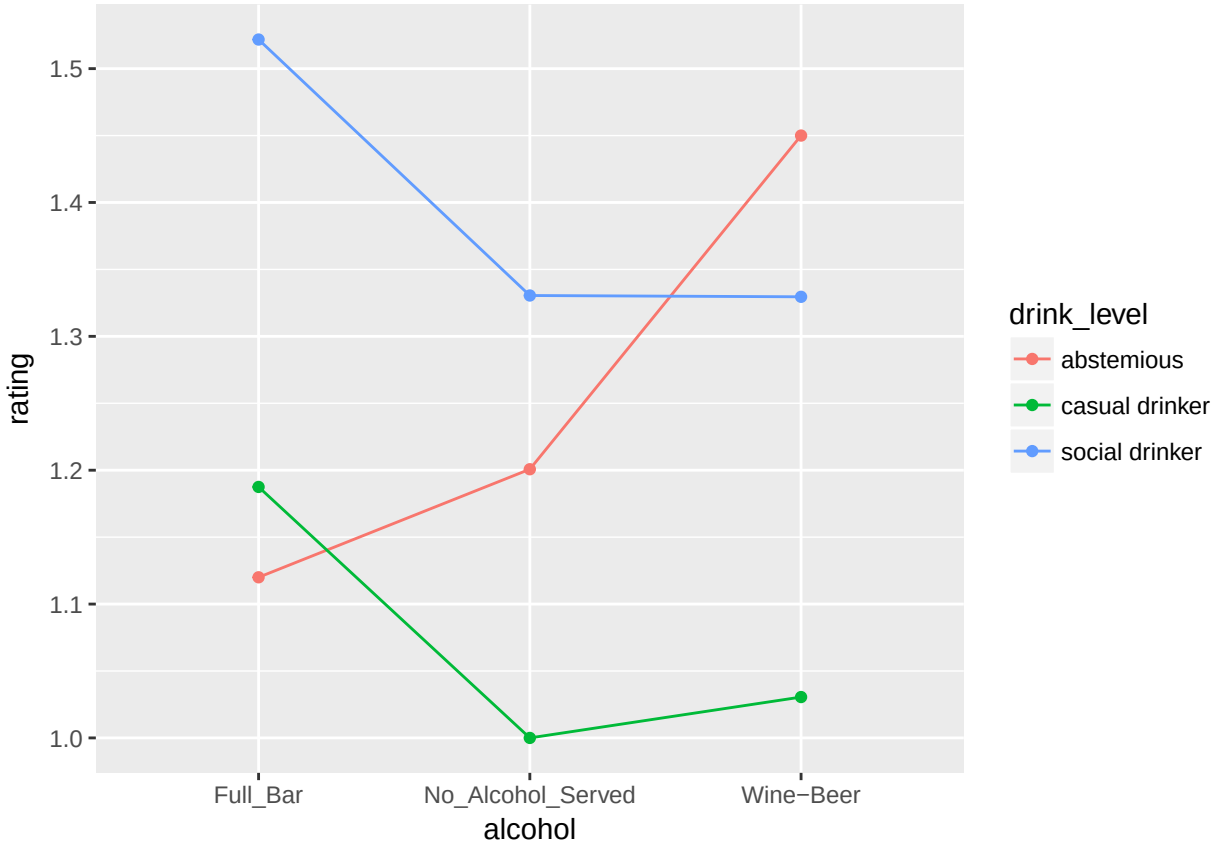
have.

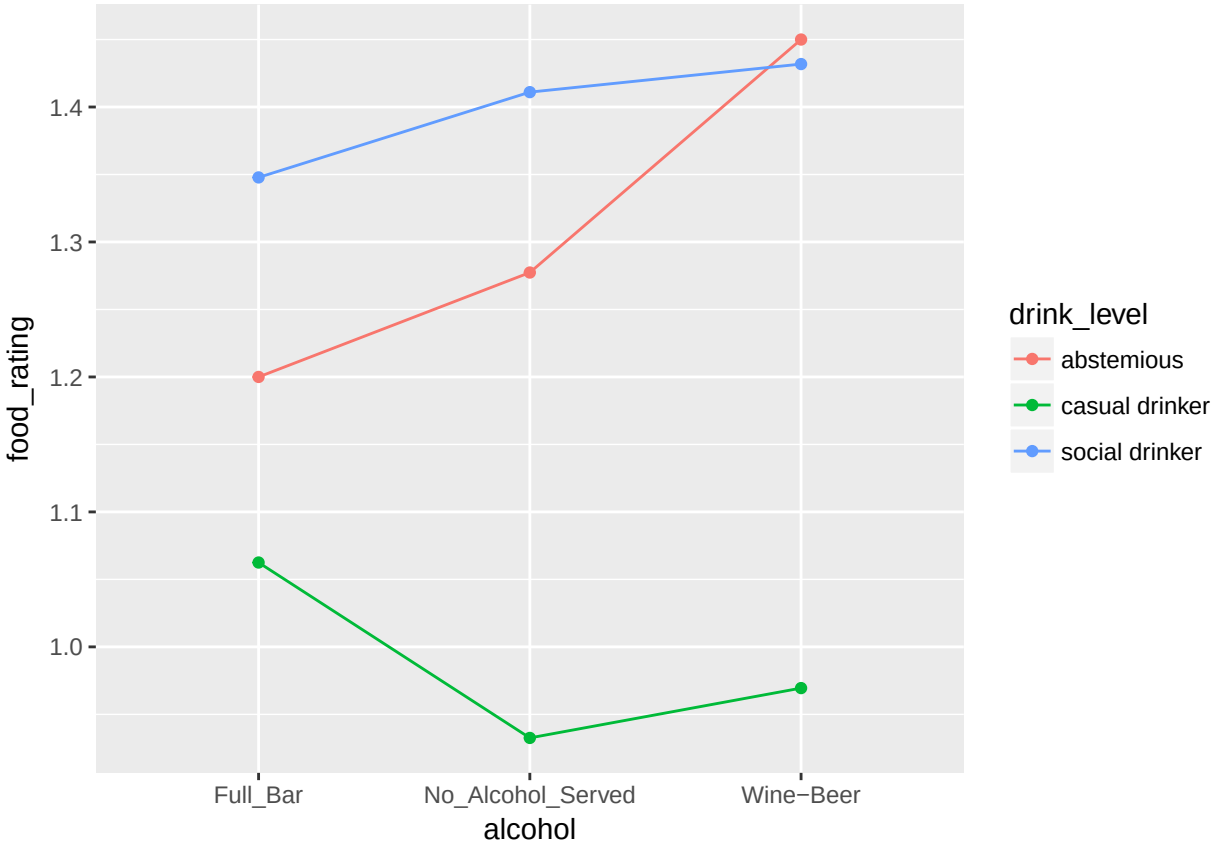

1.7
1.5
1.3
1.1
food_rating
none
not permitted
permitted
section
only at bar
smoking_area
smoker
false
true

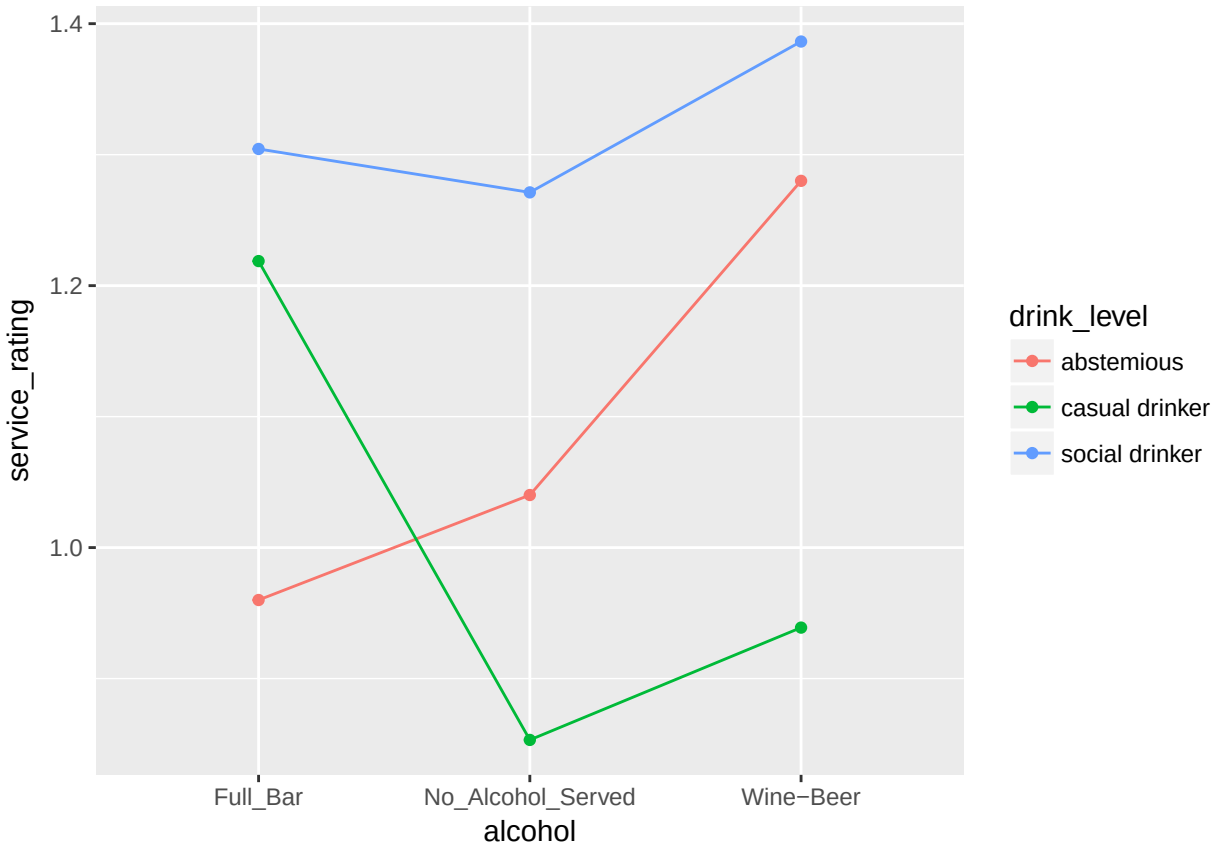


Ratings on alcohol split into three types of users

- Users who do not drink alcohol give better ratings to restaurants that serves beer/wine.
- Users who drink casualy give better ratings to restaurants with full bar.

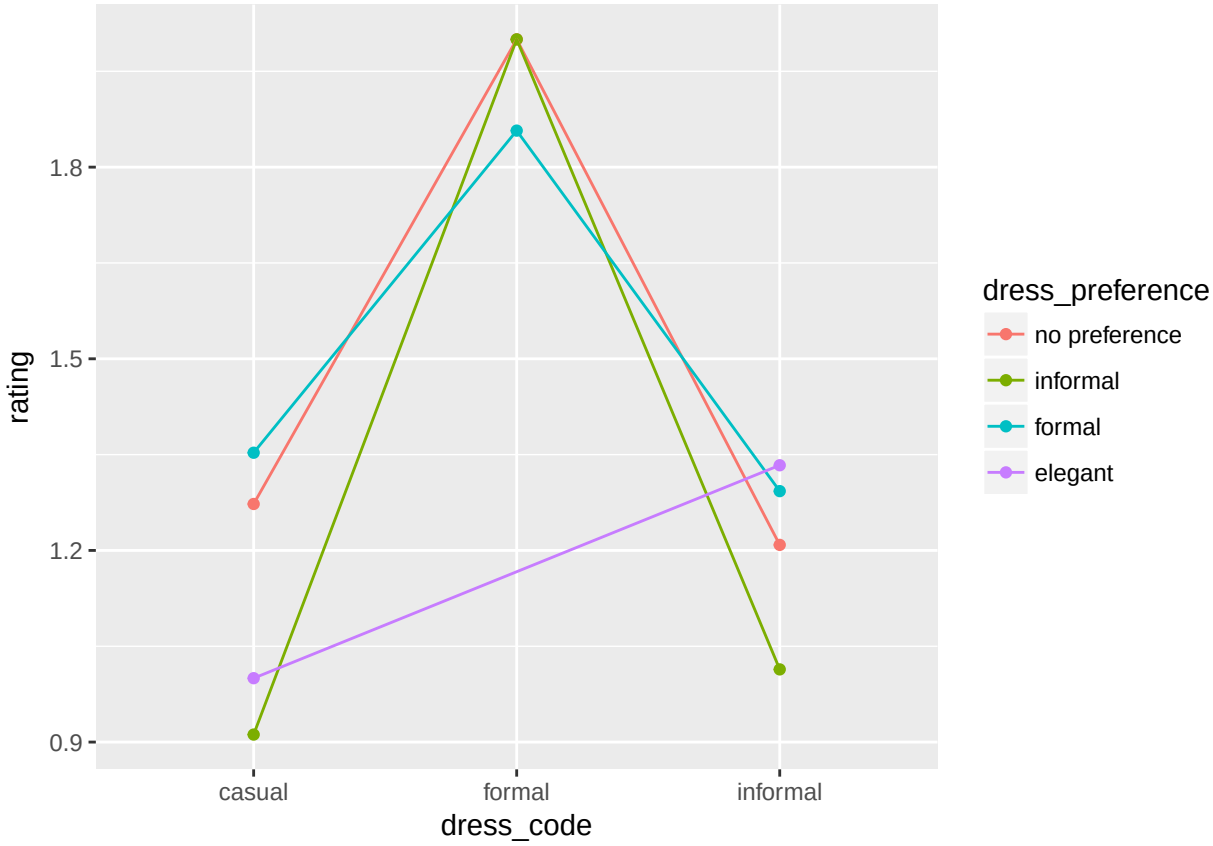

rating
1.5
1.4
1.3
1.2
1.1
1.0
Full_Bar
No_Alcohol_Served
Wine−Beer
alcohol
drink_level
abstemious
casual drinker
social drinker

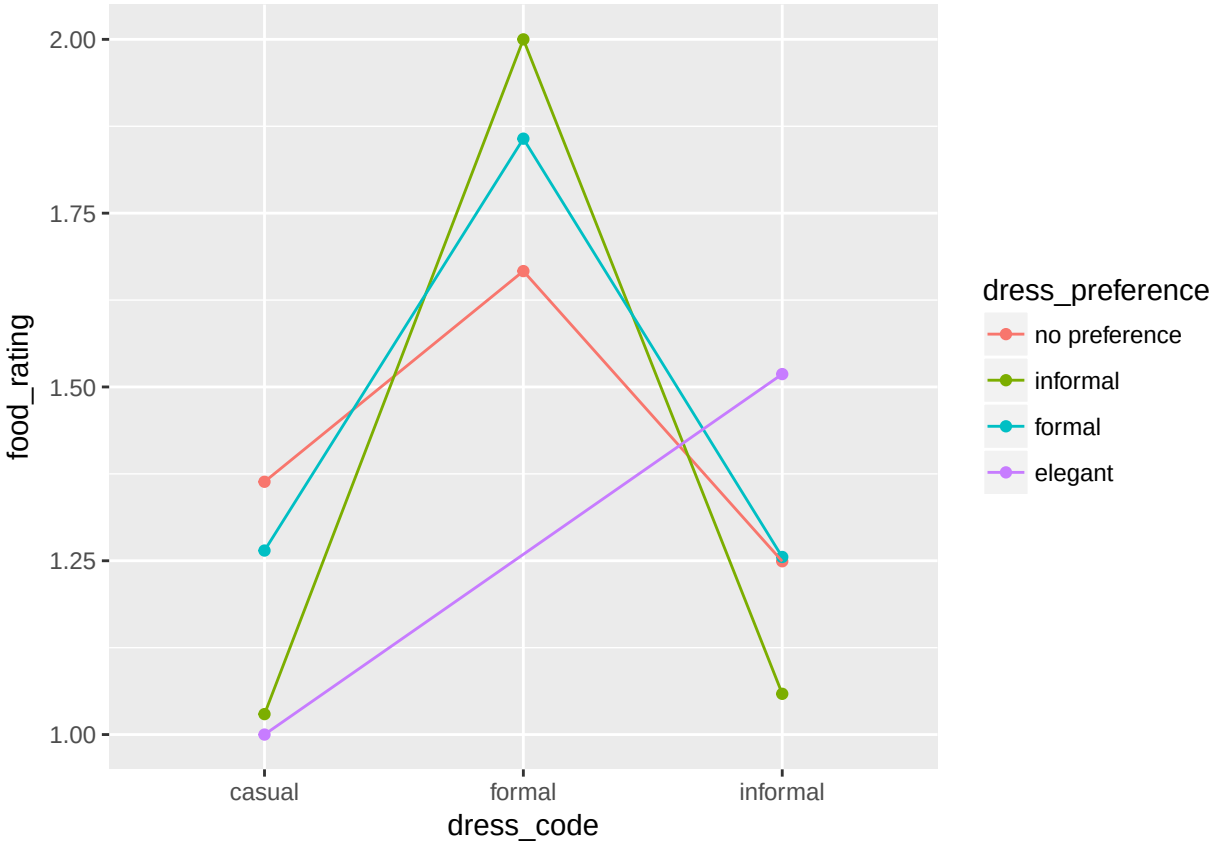

food_rating
1.4
1.3
1.2
1.1
1.0
Full_Bar
No_Alcohol_Served
Wine−Beer
alcohol
drink_level
abstemious
casual drinker
social drinker

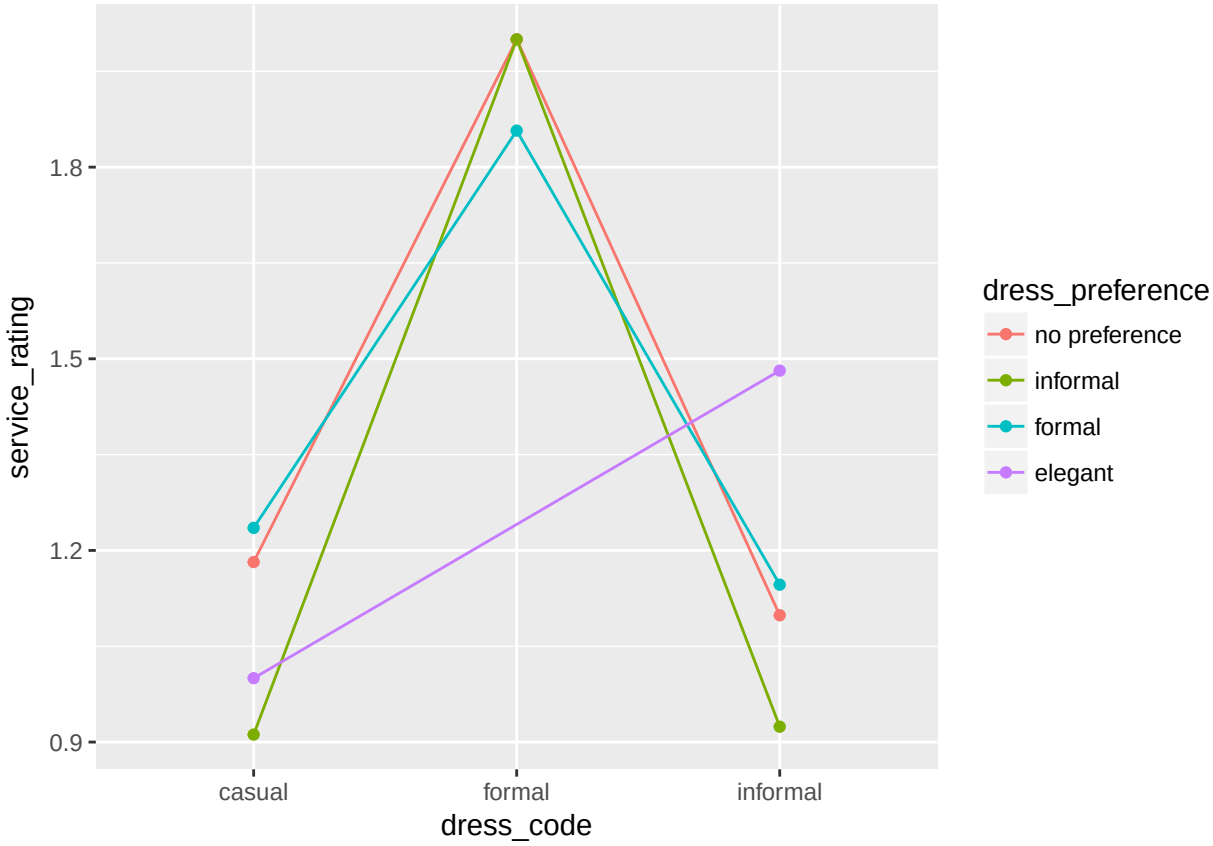


Ratings on dress code split on four types of users

- Users give better ratings to restaurants with a formal dress code independently of their dressing preferences.

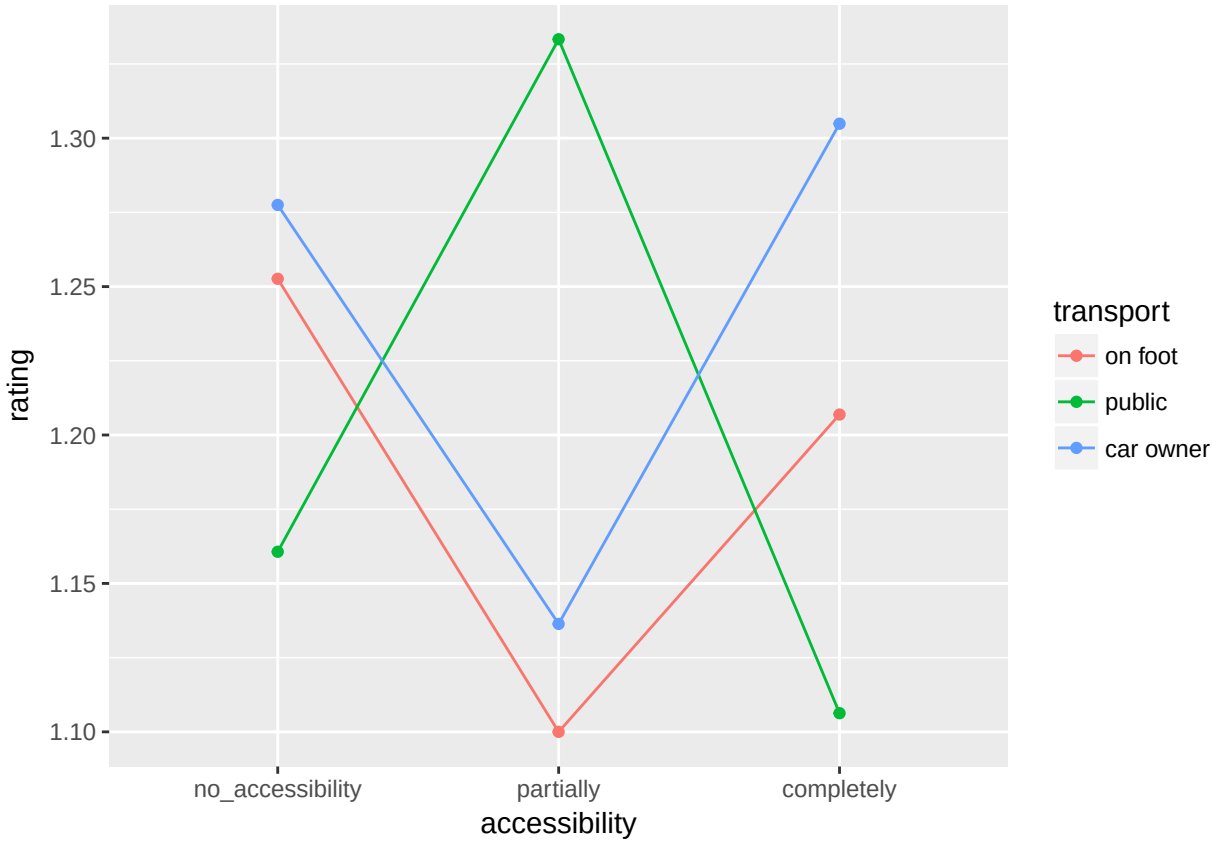

rating
1.8
1.5
1.2
0.9
casual
formal
informal
dress_code
dress_preference
no preference
informal
formal
elegant

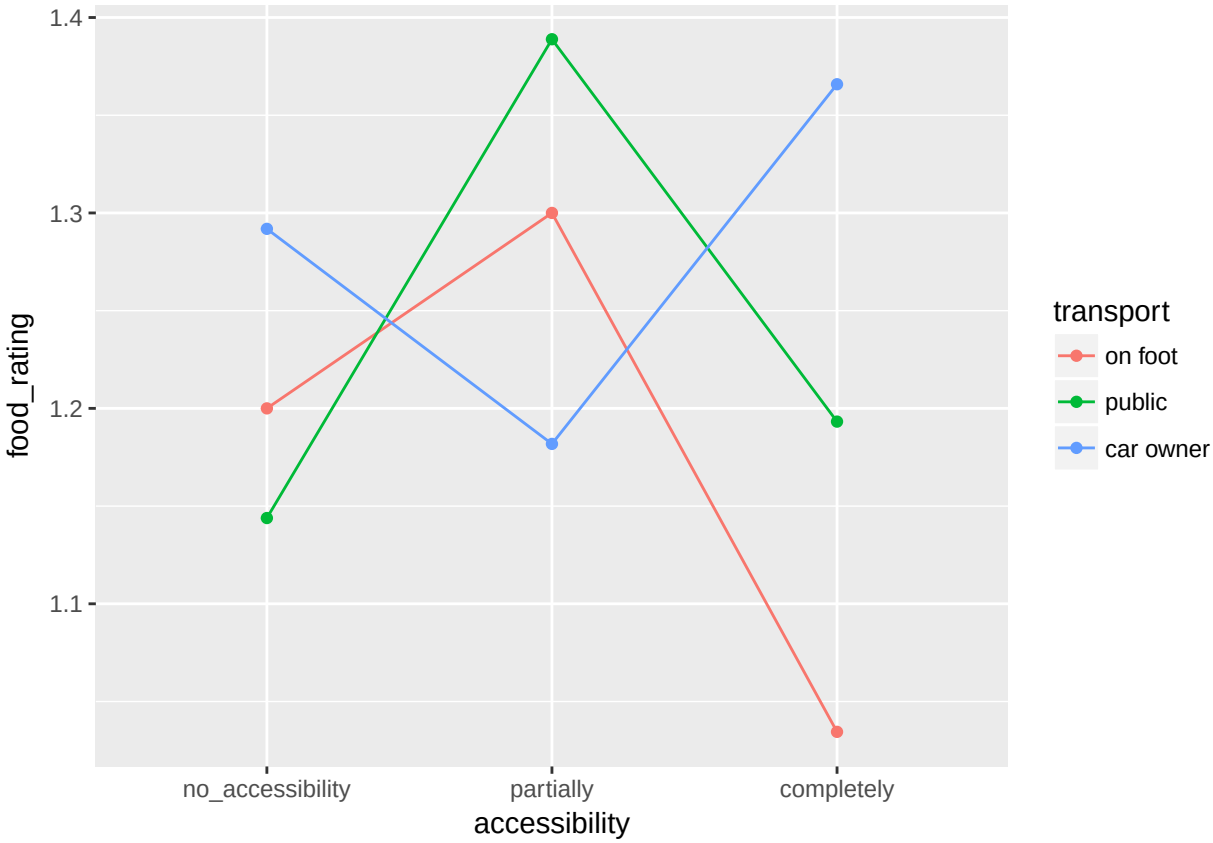

2.00
1.75
1.50
1.25
1.00
food_rating
casual
formal
informal
dress_code
dress_preference
no preference
informal
formal
elegant

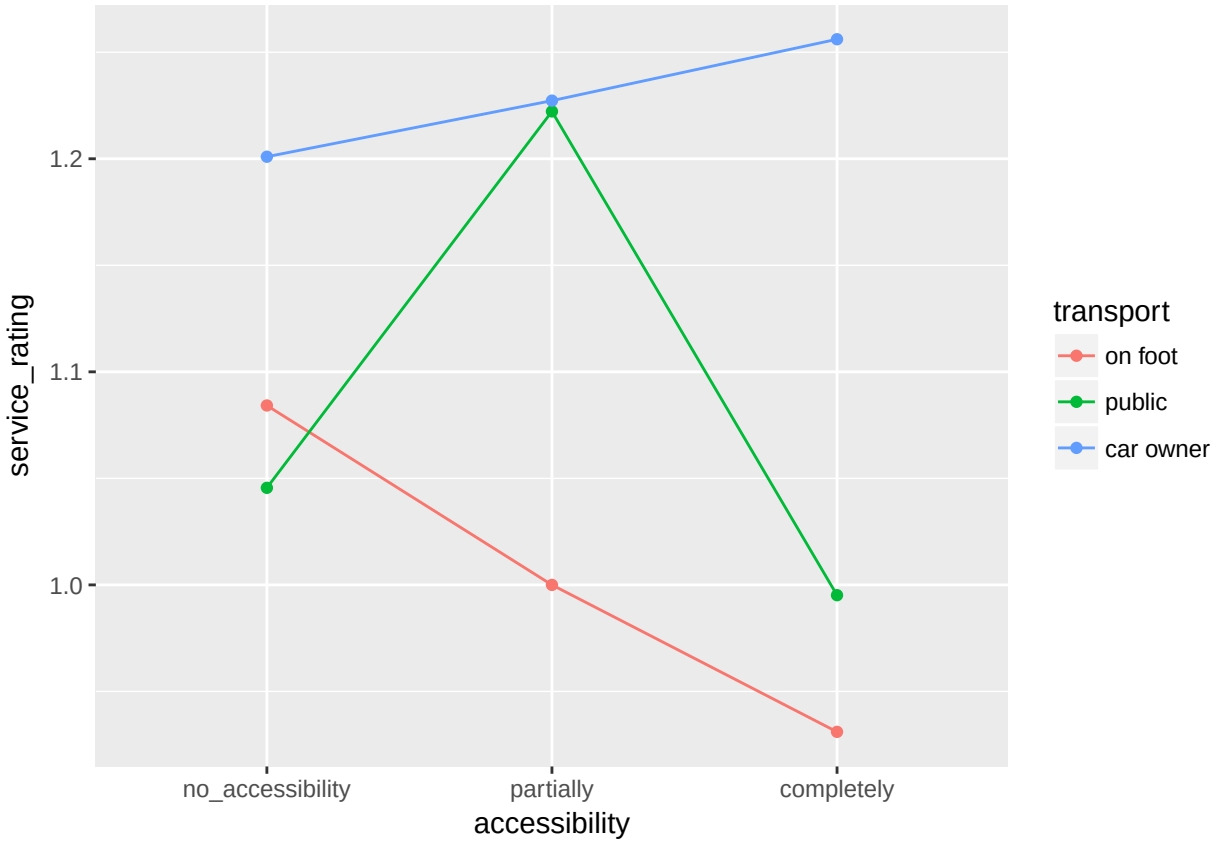


Ratings on acessibility split on three types of users

- Acessibility doesn't seem to affect the ratings.


1.30
1.25
1.20
1.15
1.10
rating
no_accessibility
partially
completely
accessibility
transport
on foot
public
car owner


1.4
1.3
1.2
1.1
food_rating
no_accessibility
partially
completely
accessibility
transport
on foot
public
car owner



2 Recommender systems

```
bm <- as(rating[,1:2], "binaryRatingMatrix") #visited/not visited
rm <- as(rating[,1:3], "realRatingMatrix")   #global rating

# rm <- normalize(rm) --already done

#(for checking purposes)
#b <- as(bm, "matrix")
#r <- as(rm, "matrix")
```

Our data is in a non-binary format as we have the ratings given to each restaurant by the users: we tried to do the transformation, counting having a rate as a visit, into a 0-1 matrix using binarize() from recommenderlab package, but we noticed that this transformation is done when coercing the data frame to a binaryRatingMatrix (when converting back from "binaryRatingMatrix" to "matrix"). That's why we do not do this transformation explicity.

For all the models to be generated we are going to need the rating data in binary and non-binary format. We start by doing it, by coercing the data frame with user-restaurant rating information to a binaryRatingMatrix for recommendation with binary info, and realRatingMatrix for recommendation with non-binary info - only non-NA values are stored explicitly, wich makes it more efficient to work with.

We split the analysis into two sections: considering binary(bm) and non-binary(rm) information. For the binary, we count each restaurant evaluation as a visit from a user; unrated restaurants by a certain user are assumed to not have been visited by him. The non-binary information corresponds to the ratings given by

users to restaurants (so, a missing rating corresponds to a NA value).

An important operation for rating matrices is to normalize the entries to, e.g., centering to remove rating bias by subtracting the row mean from all ratings in the row- this could be done with normalize(). However, this data is already normalized, as warned by this function, so no need for that.

All recommendation models are obtained using the package recommenderlab; a recommender is created using the creator function Recommender().

The top-N kind of recommendations are generated by predict(). So, in order to predict the top 2, we set the parameters "type" and "n" to "topNlist" and 2, respectively. In this way, the function returns a list of the 2 predicted values (top 2 recommendations); this list contains recommendations for all the users, so we need to select wich one we would like to see.

2.1 Popularity

2.1.1 Using simple recommendation strategies

At the begining of the assignment, we assumed that one of the intentions was to make a recommender systems ourselves, so we could compare to some of the recommender systems that we have talked about during classes. Later one we found out this was not the purpose. Even though, it counts as a good learning experience, which we would like to share.

2.1.1.1 The simplest one

```
#inspect how many times each restaurant was rated/visited
table(rating$placeID)
```

```
##
## 132560 132561 132564 132572 132583 132584 132594 132608 132609 132613
##      4      4      4     15      4      6      5      6      5      6
## 132626 132630 132654 132660 132663 132665 132667 132668 132706 132715
##      4      6      4      5      6      5      4      3      4      4
## 132717 132723 132732 132733 132740 132754 132755 132766 132767 132768
##      3     12      8     10      8     13      5      3      6     10
## 132773 132825 132830 132834 132845 132846 132847 132851 132854 132856
##      4     32     12     25      5      5      6      7      6     14
## 132858 132861 132862 132866 132869 132870 132872 132875 132877 132884
##      5      7     18      5      6      5     12      8      4      6
## 132885 132921 132922 132925 132937 132951 132954 132955 132958 134975
##      5     17      6      5      4     10      9      5      6      3
## 134976 134983 134986 134987 134992 134996 134999 135000 135001 135011
##      4      5      8      4      4      9      5      4      7      3
## 135013 135016 135018 135019 135021 135025 135026 135027 135028 135030
##      4      3      4      6      9     15     11      8     15     12
## 135032 135033 135034 135035 135038 135039 135040 135041 135042 135043
##     28      4      5      4     24     12      4     17     20      9
## 135044 135045 135046 135047 135048 135049 135050 135051 135052 135053
##      4     13     11     10      6      5      8     14     25      8
## 135054 135055 135057 135058 135059 135060 135062 135063 135064 135065
##     10      7     15     18      9     22     21      8     17      9
## 135066 135069 135070 135071 135072 135073 135074 135075 135076 135079
##     12     12      8      9      8      4      4     13     13     17
## 135080 135081 135082 135085 135086 135088 135104 135106 135108 135109
##      6     11      9     36     10      6      7     10     11      4
```

```r
#sort the restaurants by decreasing number of visits
sr <- sort(table(rating$placeID), decreasing = TRUE)
#obtain the top 3 restaurants for recommendation
names(sr)[1:3]
```

```
## [1] "135085" "132825" "135032"
```

2.1.1.2 Top-N function based on the number of evaluations

We created a function that takes as inpute an user id, and a value specifying the number of items to present as the top ones. This function only takes into account the number of evaluations, so it gives the most evaluated restaurants.

```r
get_topN_popularity <- function(user, n_top) {
  rated_restaurants <- tbl_df(rating)
  rated_restaurants_all <- group_by(rated_restaurants, placeID) %>% count %>% arrange(desc(n)) %>% sele
  rated_restaurants_user <- filter(rating, userID==user) %>% select(placeID)
  top <- setdiff(rated_restaurants_all,rated_restaurants_user) %>% head(n_top)
  return(top)
}
```

Lets say we want the get the top 3 restaurants recommendation for the user with id U1111:

```r
top10 <- get_topN_popularity("U1111", 3)
(top10 <- left_join(top10, res_info, by="placeID") %>% select(placeID))
```

```
## # A tibble: 3 x 1
## # Groups:   placeID [3]
##   placeID
##     <int>
## 1  135085
## 2  132825
## 3  135032
```

2.1.1.2 Top-N function based on the quality of the evaluations

This function take the same parameters as the previous one.

```r
get_topN_popularity <- function(user, n_top) {
  rated_restaurants <- tbl_df(rating)
  rated_restaurants_all <- mutate(rated_restaurants, avg=(rated_restaurants$rating+rated_restaurants$fo
  rated_restaurants_all <- rated_restaurants_all %>% group_by(placeID) %>% summarise(cnt=n(), average_r
  rated_restaurants_user <- filter(rating, userID==user) %>% select(placeID)
  top <- setdiff(rated_restaurants_all$placeID,rated_restaurants_user$placeID) %>%  head(n_top)
  top <- tibble(placeID=top)
  return (top)
}
```

And if we execute:

```
## # A tibble: 10 x 1
##    placeID
##      <int>
##  1  134986
##  2  132955
##  3  135034
```

```
##  4  135013
##  5  132755
##  6  135074
##  7  132922
##  8  135055
##  9  135075
## 10  134999
```

2.1.2 Using recommenderlabs package

We want to know restaurants recommendation based on popularity. As basis, we only take into account the data containing all evaluations made to each restaurant by the users. The point, of course, is to be able to recommend good rated restaurants to a user from whom we may know nothing about.

```
# create the models
modelPOPb <- Recommender(bm, method="POPULAR")
modelPOPr <- Recommender(rm, method="POPULAR")

# generate 2 recommendations
predb <- predict(modelPOPb, bm, type="topNList", n=2)
predr <- predict(modelPOPr, rm, type="topNList", n=2)
```

```
## [1] "Predicted values using binary information"
```

```
##       1      2
## U1001 135032 132834
## U1002 132834 135085
## U1003 135032 135032
```

```
## [1] "Predicted values using non-binary information"
```

```
##       1      2
## U1001 134986 135057
## U1002 135057 134986
## U1003 134986 135057
## U1004 135057 134986
## U1005 134986 135052
```

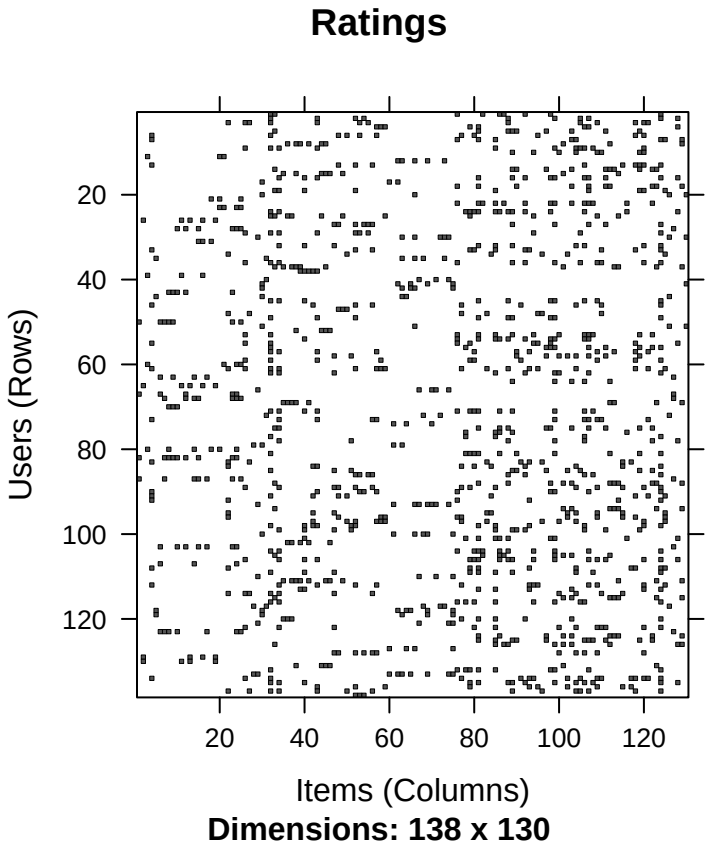
Intuitively, as this kind of analysis takes no information other than evaluations, one would expect the same kind of results for every user: only the best rated restaurants. In fact, this does not occur as it makes no sense to recommend a restaurant to a user who has already visited it. For instance, as it is possible to observe from the outputs above, users U1001 and U1002 have the same restaurant recommendations: 135032 and 132834; on the other hand, the recommendations for user U1003 only have one restaurant in common with U1001 and U1002 (135032), as he has already visited 132834. The same logic follows for the non-binary information.

2.2 Association Rules

Task: we want to be able to predict which restaurants the visitor is most interested at some point (in this case, in the final of our analysis, but this could be implemented as a live recommender system which would recommend restaurants at the time of the query).

Strategy: look for restaurants that were visited and look for sets of restaurants that predict other sets of restaurants using association rule discovery. So, the model is a set of discovered rules, from which we make queries to get the best ones depending on the previouly visited restaurants.

An easy way to see the who rated which restaurant, and to visually find patterns is with image() function.



To construct our association rule based recommendation model, some parameters are required; the better these parameters, the better is our model. For instance, a value for confidence must be chosen: if we set the bar to high we may get into a point where we do not get any recommendation (or few, although really strong ones); if we set it too low we get too many recommendations (being the majority weak ones).

In order to archieve good results, we need to evaluate different combinations of values- we do so using the k-fold cross validation method: the key idea is to split the data into k parts, and, for each run, k-1 parts are used for training and the remaining part is used for testing, repeating this process a total of k times. We choose our k=5.

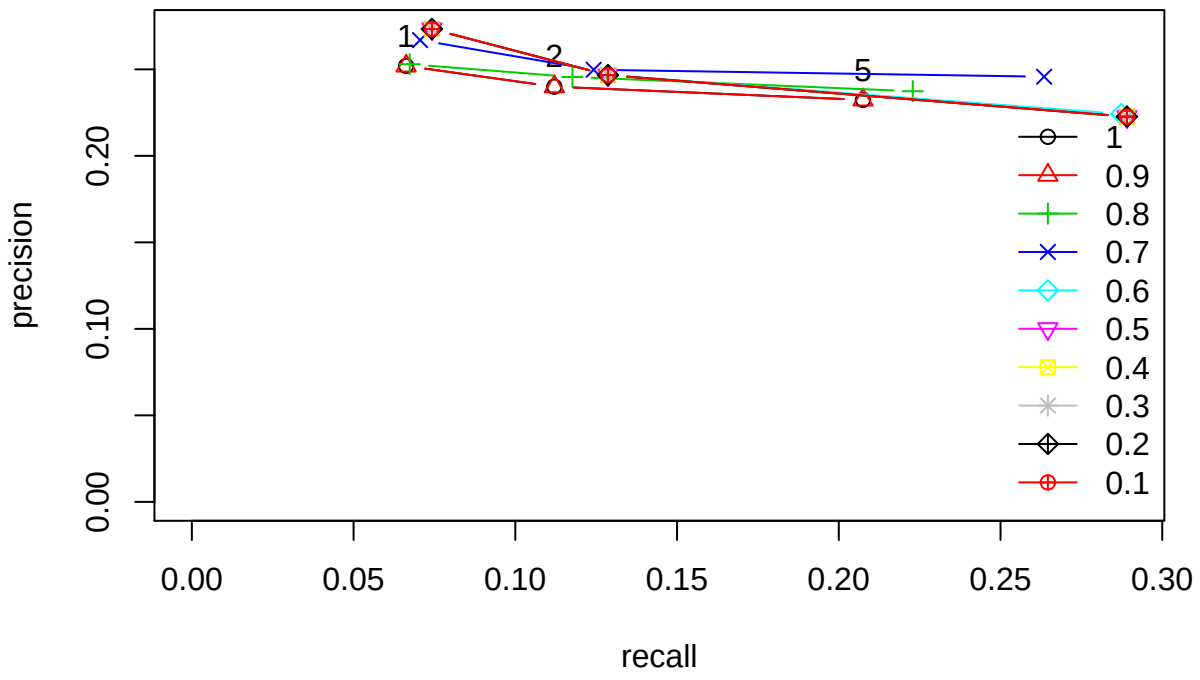
We set some values for the paramaters "support", "confidence" and "maxlen". Some short explanations on these parameters:
-support: the minimum support of a rule, how many transactions support the condition;
-confidence: the minimum confidence of a rule, the significance of a rule;
-maxlen: the maximum number of restaurants we want to be included (correlated) in a rule.

To evaluate recommender algorithms, package recommenderlab provides the infrastructure to create and maintain evaluation schemes stored as an object of class evaluationScheme. The function evaluate() is then used to evaluate several recommender algorithms using an evaluation scheme resulting in a evaluation result list with one entry per algorithm. We use this way to figure out wich parameters are the best according to some error measures. As stated in the slides[5], we set a very low support (10/|DB|), and we vary the confidence - even though there are values we know for sure that will be bad, we want to see their impact graphically. Also, following the phrase "one is bad, two are bad luck, three are not a coincidence", we set the maximum number of restaurants to be used in a rule as 3- long rules have the danger of overfitting data.

```
set.seed(1234)
scheme <- evaluationScheme(bm, method="cross-validation", k=5, given=3)
```

```
algorithms <- list(
  "1"   = list(name="AR", param=list(support = 10/nrow(rating), conf = 1,   maxlen = 3)),
  "0.9" = list(name="AR", param=list(support = 10/nrow(rating), conf = 0.9, maxlen = 3)),
  "0.8" = list(name="AR", param=list(support = 10/nrow(rating), conf = 0.8, maxlen = 3)),
  "0.7" = list(name="AR", param=list(support = 10/nrow(rating), conf = 0.7, maxlen = 3)),
  "0.6" = list(name="AR", param=list(support = 10/nrow(rating), conf = 0.6, maxlen = 3)),
  "0.5" = list(name="AR", param=list(support = 10/nrow(rating), conf = 0.5, maxlen = 3)),
  "0.4" = list(name="AR", param=list(support = 10/nrow(rating), conf = 0.4, maxlen = 3)),
  "0.3" = list(name="AR", param=list(support = 10/nrow(rating), conf = 0.3, maxlen = 3)),
  "0.2" = list(name="AR", param=list(support = 10/nrow(rating), conf = 0.2, maxlen = 3)),
  "0.1" = list(name="AR", param=list(support = 10/nrow(rating), conf = 0.1, maxlen = 3))
)

results <- evaluate(scheme, algorithms, type = "topNList", n=c(1,2,5))
```

```
plot(results, "prec/rec", annotate=TRUE)
```



This is the tough part: we would want higher values of precision and recall (highly specialised, very picky), so the ones that go closer to the right superior corner would be the better parameters for our model - sometimes it's more important to correctly classify the samples than getting all of them (high precision); sometimes it's more important to get all of them (high recall) than getting all of them correctly. Even though, we don't want to overstep here, and we prefer a higher precision model, with medium-to-low sensitivity (recall). From the graph, and assuming only the parameters tested, we choose to go with **confidence=0.7**.

```
#get the model
modelAR <- Recommender(bm, method="AR",
```

```
                        parameter=list(support = 10/nrow(rating), conf = 0.7, maxlen = 3))

# object containing all the rules
rules <- getModel(modelAR)$rule_base

# generate a prediction based on the model obtained above
pred<- predict(modelAR, bm, n = 2)
```

```
## Predicted values based on the rules available in the model

## U1001

## [1] "135040" "135033"

## U1002

## [1] "132825" "135062"

## U1003

## [1] "132723" "132825"

## U1127

## character(0)

## U1138

## [1] "132921"
```

Well, it appears that with a 0.7 confidence some users do not get a recommendation, or only one, such as U1127 and U1138. This means that there's no rule in our model having the left hand side with the restaurants they have visited. For instance, if we drop the confidence, more rules are retained and eventually they will end up to having recomendations, even though theoretically not very strong ones.

```
# get a new model with confidence 0.6
modelAR <- Recommender(bm, method="AR",
                        parameter=list(support = 10/nrow(rating), conf = 0.6, maxlen = 3))

# object containing all the rules
rules <- getModel(modelAR)$rule_base

# generate a prediction based on the model obtained above
pred<- predict(modelAR, bm, n = 2)
```

```
## Predicted values based on the rules available in the model

## U1001

## [1] "135040" "135033"

## U1002

## [1] "132825" "135062"

## U1003

## [1] "132723" "132825"

## U1127

## [1] "134983" "135021"

## U1138
```

```
## [1] "132921" "132937"
```

With the drop of confidence to 0.6 all users have now recomendations, as expected. A note to the unchanged recommendations for the rest of the users. As we were expecting, this change doesn't affect the strength of the recommendations that were part of the 0.7 confidence model- those were pretty good; the less value for confidence only makes sure more rules are possible to remain as we set the low bar even lower.

2.3 Collaborative Filtering

Collaborative filtering uses given rating data by many users for many items as the basis for predicting missing ratings and/or for creating a top-N recommendation list for a given user, called the active user.

In collaborative filtering, and taking into account our data, the customers' preference data for restaurants are collected and a customer is recommended restaurants that they are likely to enjoy based on their preferences.

The user will be recommended restaurants that people with similar tastes and preferences liked in the past; we do not need to now more information on the restaurants.

For the collaborative filtering it is better to have more ratings per user. So as a rule of tumb our idea was to remove users who have had rated fewer than 3 restaurants- this does not occur, which is nice.

```
rating %>% group_by(userID) %>% mutate(n = n()) %>% filter(n<3)
```

We could calculate the distance between users and the distance between restaurants so we could see whose users are similar, and which restaurants are alike, using the similarity function - this takes a method arguments to specify which one we would like to use (we are using "cosine", but we could use "pearson", "jaccard", etc). Although, given the big dimensions of the data, even though it would be nice to print it in, it would take a few pages...

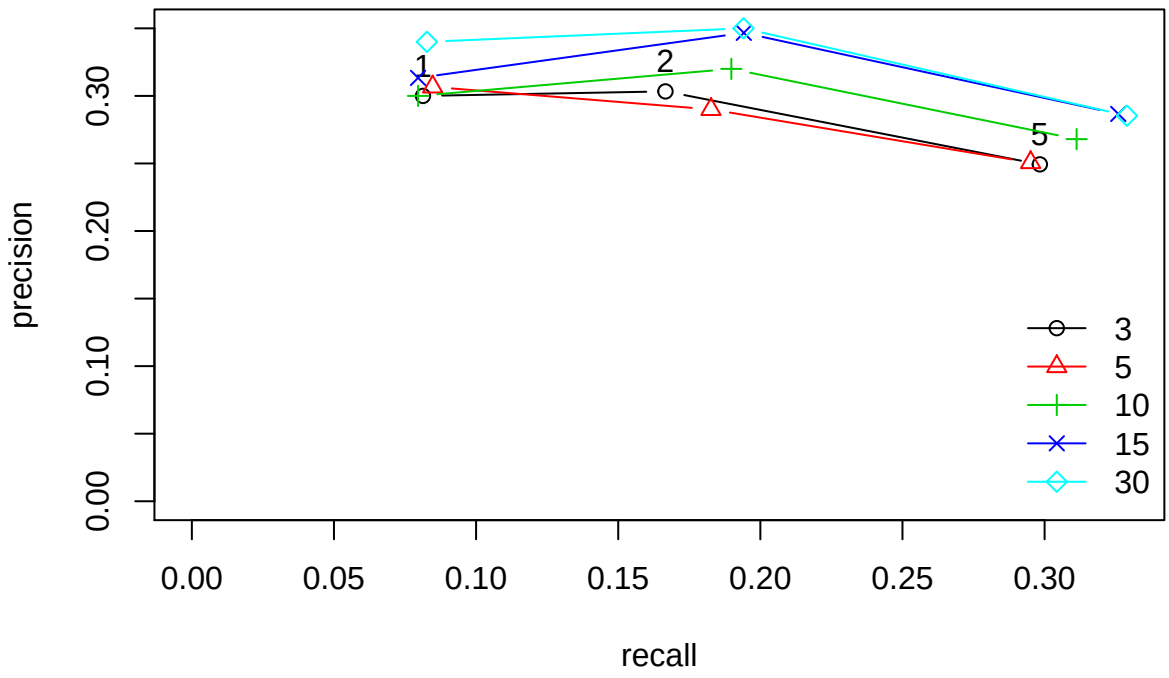
```
simCos_users <- similarity(bm, method = "cosine")
simCos_items <- similarity(bm, method = "cosine", which = "items")
```

So, taking int account the binary information, once again we start by using Recommender() to generate our model, this time specifying "UBCF" and "IBCF" for user-based and item-based collaborative filtering. Once again we need to calibrate our model, as it takes parameters: the nearest neighbours (neighborhood size)- we do another 5-fold cross validation. Another parameter we set is the similarity method to be used: cosine.

```
set.seed(1234)
scheme <- evaluationScheme(bm, method="cross-validation", k=5, given=3)
```

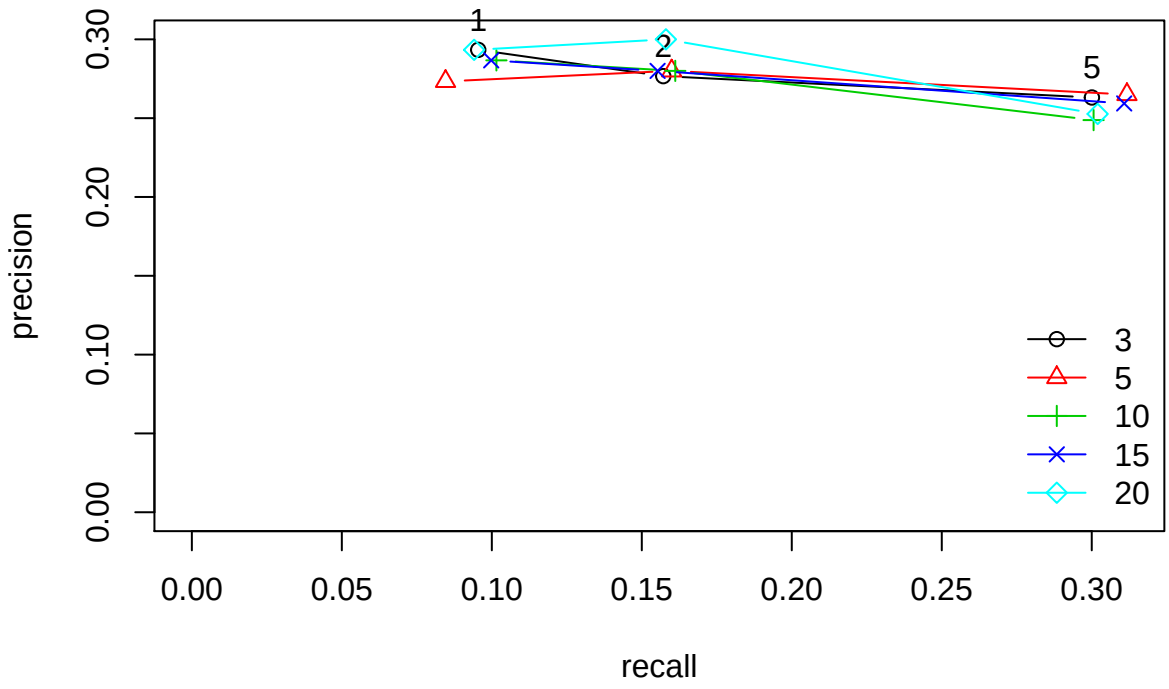
```
algorithms <- list(
  "3" = list(name="UBCF", parameter = list(method = "cosine", nn = 3)),
  "5" = list(name="UBCF", parameter = list(method = "cosine", nn = 5)),
  "10" = list(name="UBCF", parameter = list(method = "cosine", nn = 10)),
  "15" = list(name="UBCF", parameter = list(method = "cosine", nn = 15)),
  "30" = list(name="UBCF", parameter = list(method = "cosine", nn = 20))
)

results <- evaluate(scheme, algorithms, type = "topNList", n=c(1,2,5))
```

```
plot(results, "prec/rec", main="UBCF", annotate=TRUE)
```




```
algorithms <- list(
  "3" = list(name="IBCF", parameter = list(method = "cosine", k = 3)),
  "5" = list(name="IBCF", parameter = list(method = "cosine", k = 5)),
  "10" = list(name="IBCF", parameter = list(method = "cosine", k = 10)),
  "15" = list(name="IBCF", parameter = list(method = "cosine", k = 15)),
  "20" = list(name="IBCF", parameter = list(method = "cosine", k = 20))
)

results <- evaluate(scheme, algorithms, type = "topNList", n=c(1,2,5))
```

```
plot(results, "prec/rec", main="IBCF", annotate=TRUE)
```



We go for nn=10 and k=5, using the same criteria we used in the Association rule based recommendation system, and set n=2 to get the top 2 for the user U1001.

```
#user based
bmodelUB <- Recommender(bm, "UBCF", parameter = list(method = "cosine", nn = 10))
#item based
bmodelIB <- Recommender(bm, "IBCF", parameter = list(method = "cosine", k = 5))

#top 2 prediction
bpredub <- predict(bmodelUB, bm, n = 2)
bpredib <- predict(bmodelIB, bm, n = 2)
```

```
#user based
nbmodelUB<- Recommender(rm, "UBCF", parameter = list(method = "cosine", nn=2))

#top2 prediction
nbpredub <- predict(nbmodelUB, rm, n=2)
```

```
## [1] "Predicted values using binary information UBCF"
```

```
##       1      2
## U1001 135052 135042
## U1002 135032 132834
```

```
## [1] "Predicted values using binary information IBCF"
```

```
##       1      2
## U1001 135057 135032
## U1002 135054 135066
```

```
## [1] "Predicted values using non-binary information UBCF"
```

```
##       1      2
## U1001 132846 135032
## U1002 132847 135042
```

2.4 Recommendation systems evaluation: comparing top-N recommendations

To evaluate recommender algorithms, package recommenderlab provides the infrastructure to create and maintain evaluation schemes stored as an object of class evaluationScheme from rating data. The function evaluate() is then used to evaluate several recommender algorithms using an evaluation scheme resulting in a evaluation result list with one entry per algorithm. Each object of evaluationResult contains one or several object of confusionMatrix depending on the number of evaluations specified in the evaluationScheme. With this infrastructure several recommender algorithms can be compared on a data set with a single line of code.

We use the same function we used to evaluate all the other recommendation systems, but now with a list of algorithms together with their parameters. In the following we use the evaluation scheme created to compare the four recommender algorithms: popular restaurants, association rules, user-based CF, item-based CF.

Next we evaluate each recommender algorithm: we start by creating a 5-fold cross validation scheme, specifying its parameters; only then we evaluate top-1, top-2 and top-5 recommendation lists.

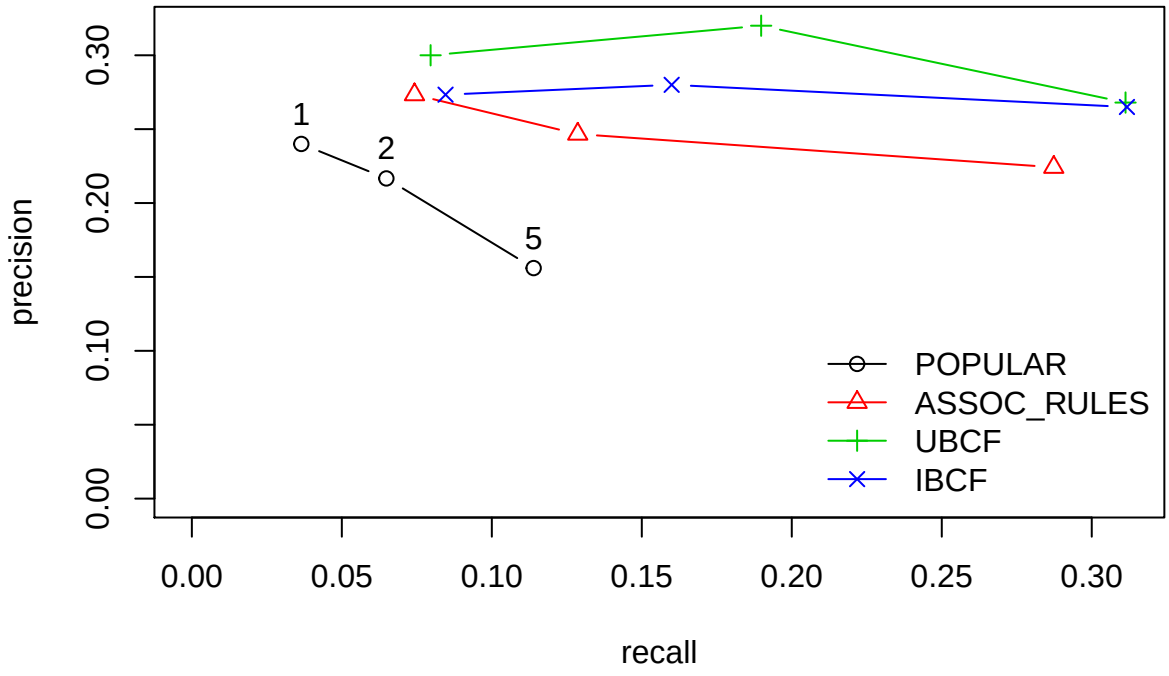
```
set.seed(1234)
# create an evaluation scheme (5-fold cross validation, given-3 scheme)
scheme <- evaluationScheme(bm, method="cross-validation", k=5, given=3)

algorithms <- list(
  POPULAR = list(name="POPULAR"),
  ASSOC_RULES = list(name="AR",
                     parameter=list(support = 10/nrow(rating), conf = 0.6, maxlen = 3)),
  UBCF = list(name="UBCF", param=list(method = "cosine", nn=10)),
  IBCF = list(name="IBCF", param=list(method = "cosine", k=5))
)

results <- evaluate(scheme, algorithms, type = "topNList", n=c(1,2,5))
```

```
# average confusion matrices for the 5 runs
avg(results)
```

```
## $POPULAR
##          TP       FP       FN       TN precision     recall        TPR
## 1 0.2400000 0.760000 5.220000 120.7800 0.2400000 0.03648897 0.03648897
## 2 0.4333333 1.566667 5.026667 119.9733 0.2166667 0.06484671 0.06484671
## 5 0.7800000 4.220000 4.680000 117.3200 0.1560000 0.11394432 0.11394432
##           FPR
## 1 0.006213067
## 2 0.012815245
## 5 0.034596177
##
## $ASSOC_RULES
##          TP        FP       FN       TN precision     recall        TPR
## 1 0.2733333 0.7266667 5.186667 120.8133 0.2733333 0.07423227 0.07423227
## 2 0.4933333 1.5066667 4.966667 120.0333 0.2466667 0.12864099 0.12864099
## 5 1.1000000 3.7400000 4.360000 117.8000 0.2242222 0.28733291 0.28733291
##           FPR
## 1 0.005977628
```

```
## 2 0.012380690
## 5 0.030736594
##
## $UBCF
##     TP   FP   FN     TN precision    recall       TPR         FPR
## 1 0.30 0.70 5.16 120.84     0.300 0.0795960 0.0795960 0.005751388
## 2 0.64 1.36 4.82 120.18     0.320 0.1897914 0.1897914 0.011171681
## 5 1.34 3.66 4.12 117.88     0.268 0.3112613 0.3112613 0.030015124
##
## $IBCF
##          TP        FP       FN       TN precision     recall        TPR
## 1 0.2733333 0.7266667 5.186667 120.8133 0.2733333 0.08457987 0.08457987
## 2 0.5600000 1.4400000 4.900000 120.1000 0.2800000 0.15995780 0.15995780
## 5 1.2866667 3.5866667 4.173333 117.9533 0.2650000 0.31173374 0.31173374
##           FPR
## 1 0.005978101
## 2 0.011843283
## 5 0.029466108
```

```
plot(results, "prec/rec", annotate=TRUE)
```



- Looking only for each average results table, we see that “Popular”" is the worst recommender system taking into account the true positives, meaning a low performance as a recommender system, and looking to the whole picture, in the graph, is far from the rest of the recommendation systems.
- Looking only to the graph, the closer to the superior right corner the best: this means Collaborative filtering outperforms the other recommenders (at least the ones analysed here). On paper this is more

robust than the other two, and so can be seen here.

3 Context-aware recommendation

The previous models don't take into account the context of the recommendations, being it the customer's budget, smoking habits or drinking level. As we've seen previously in the exploratory analysis, there are a number of features that seems to be correlated with the rates given to the restaurants. Namely, the price, the smoking area, the alcohol service and the dress code. In this chapter We'll test a couple of different scenarios and build some context aware recommendations customized for those situations. For each scenario, we applied two steps:

1. Pre-filtering the data according to the context in hand.
2. Comparing the different recommender systems using 5-fold cross-validation for the top 1, 2 and 5 recommendations with binary and/or non binary information.

Scenario 1: Returns cheaper restaurants to low budget users

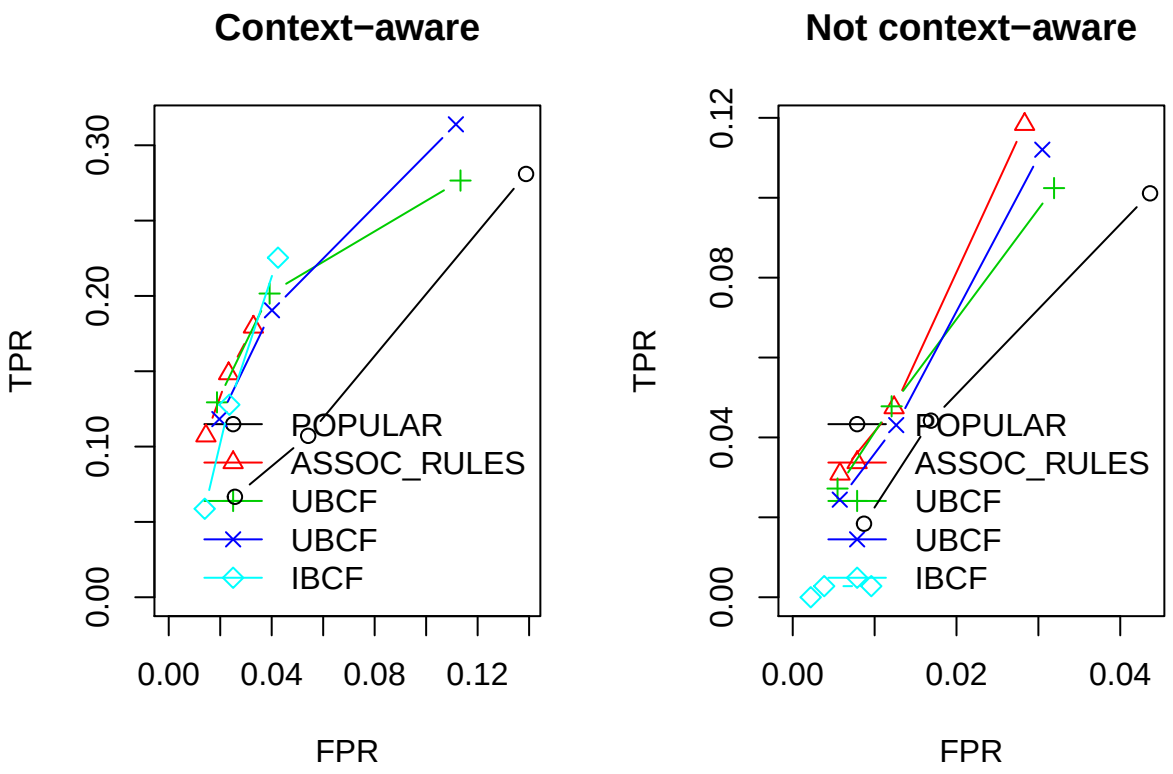
Let's compare the performance of context aware models returning cheaper restaurants for users with lower budgets to non-context aware models.

Filtering low budget users and building the rating matrices:

```
context_ratings <- rating  %>% inner_join(res_info,by="placeID") %>% inner_join(user_info, by="userID")

all_ratings <- rating  %>% inner_join(res_info,by="placeID") %>% inner_join(user_info, by="userID") %>%
  select(userID, placeID, rating, food_rating, service_rating)

bm_context <- as(context_ratings[,1:2], "binaryRatingMatrix")
bm_all <- as(all_ratings[,1:2], "binaryRatingMatrix")
```

Comparing the results using binary information:

```
set.seed(1234)
scheme_context <- evaluationScheme(bm_context, method="cross-validation", k=5, given=1)
scheme_all <- evaluationScheme(bm_all, method="cross-validation", k=5, given=1)

algorithms <- list(
  POPULAR = list(name="POPULAR", param=NULL),
  ASSOC_RULES = list(name="AR", param=list(support = 0.0005, conf = 0.5, maxlen = 5)),
  UBCF = list(name="UBCF", param=list(method = "cosine", nn=3)),
  UBCF = list(name="UBCF", param=NULL),
  IBCF = list(name="IBCF", param=list(method = "cosine", k=3))
)

results <- evaluate(scheme_context, algorithms, type = "topNList", n=c(1,2,5))
results2 <- evaluate(scheme_all, algorithms, type = "topNList", n=c(1,2,5))
```



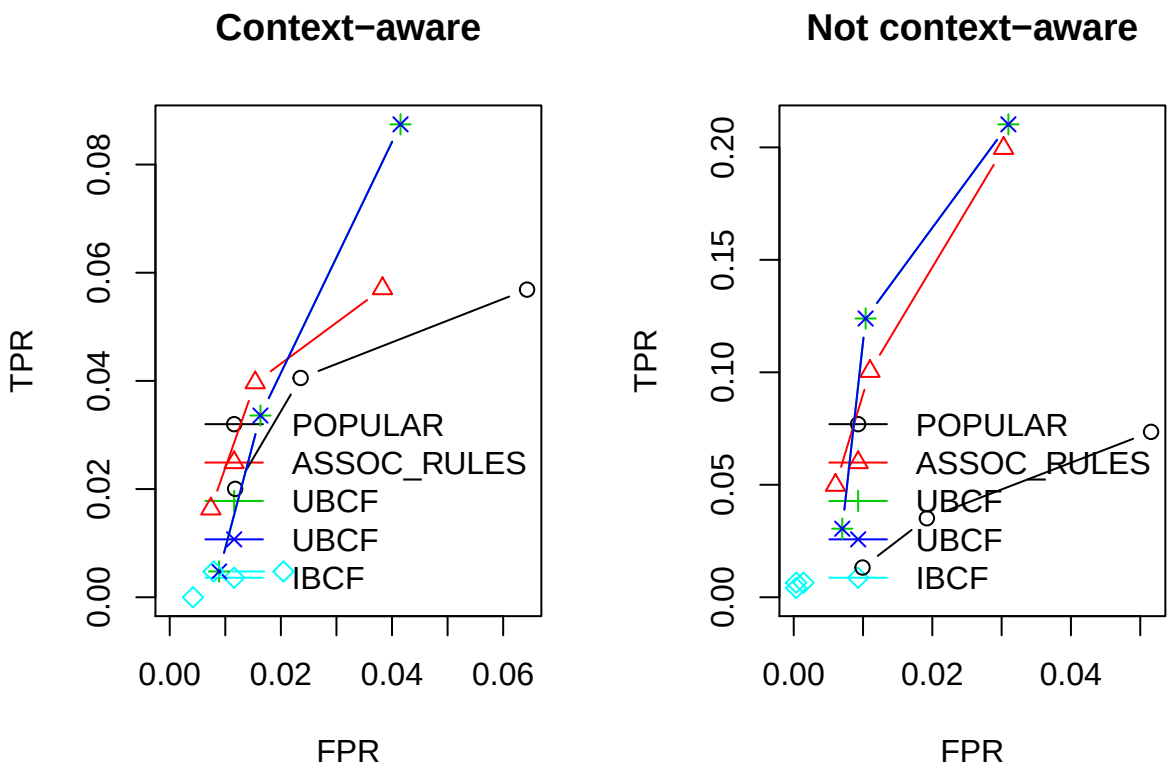
Scenario 2: returns restaurants where smoking is permitted to smokers.

Filtering smokers from users, and building the rating matrices:

```
context_ratings <- rating  %>% inner_join(res_info,by="placeID") %>% inner_join(user_info, by="userID")
  select(userID, placeID, rating, food_rating, service_rating)
all_ratings <- rating  %>% inner_join(res_info,by="placeID") %>% inner_join(user_info, by="userID") %>%
  select(userID, placeID, rating, food_rating, service_rating)

bm_context <- as(context_ratings[,1:2], "binaryRatingMatrix")
bm_all <- as(all_ratings[,1:2], "binaryRatingMatrix")
```

Comparing the results using binary information:

```
set.seed(1234)
scheme_context <- evaluationScheme(bm_context, method="cross-validation", k=5, given=1)
scheme_all <- evaluationScheme(bm_all, method="cross-validation", k=5, given=1)

results <- evaluate(scheme_context, algorithms, type = "topNList", n=c(1,2,5))
results2 <- evaluate(scheme_all, algorithms, type = "topNList", n=c(1,2,5))
```



Scenario 3: returns restaurants with alcohol service to casual/social alcohol consummers

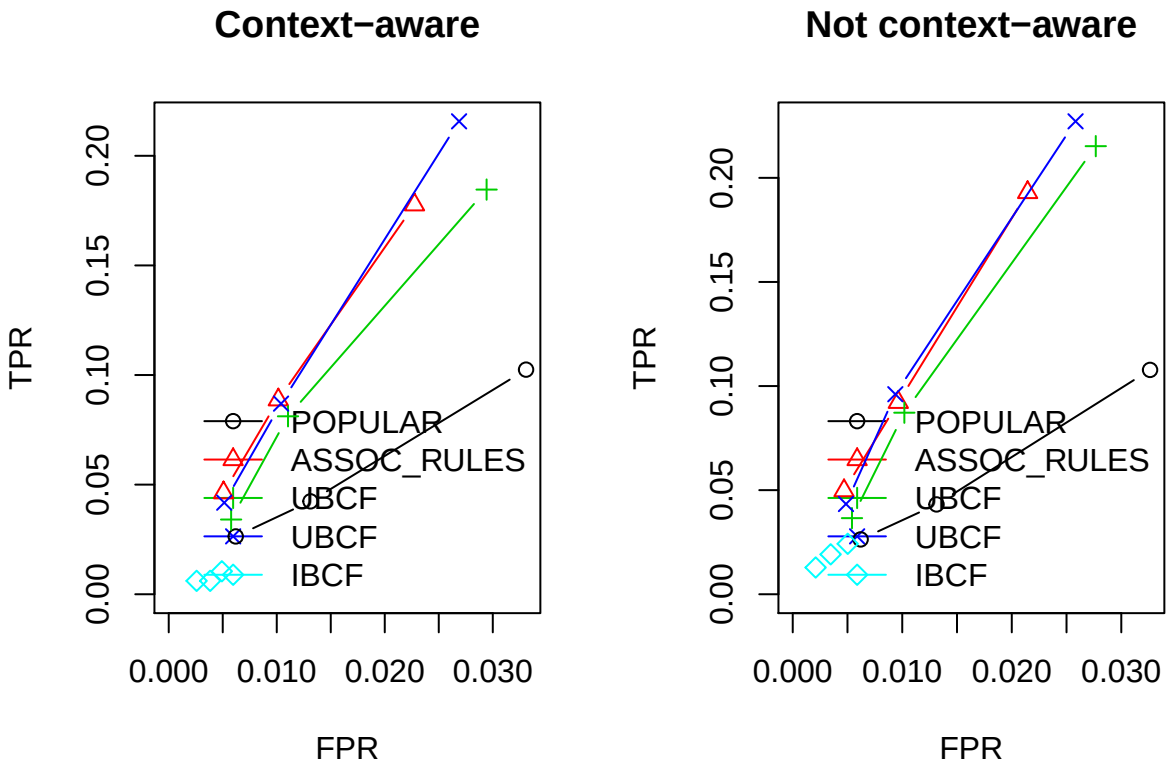
Alcohol consumption might be relevant, as users who drink alcohol might prefere restaurants with alcohol service.

Filtering users who consume alcohol:

```
context_ratings <- rating  %>% inner_join(res_info,by="placeID") %>% inner_join(user_info, by="userID")
  select(userID, placeID, rating, food_rating, service_rating)

all_ratings <- rating  %>% inner_join(res_info,by="placeID") %>% inner_join(user_info, by="userID") %>%
  select(userID, placeID, rating, food_rating, service_rating)

bm_context <- as(context_ratings[,1:2], "binaryRatingMatrix")
bm_all <- as(all_ratings[,1:2], "binaryRatingMatrix")
```

Comparing the results:

```
set.seed(1234)
scheme_context <- evaluationScheme(bm_context, method="cross-validation", k=5, given=1)
scheme_all <- evaluationScheme(bm_all, method="cross-validation", k=5, given=1)

results <- evaluate(scheme_context, algorithms, type = "topNList", n=c(1,2,5))
results2 <- evaluate(scheme_all, algorithms, type = "topNList", n=c(1,2,5))
```



Improvements

Some work that can possibly improve the predictions for this restaurant-consumer dataset is summarised as follows:
* Use a weigth average of the three ratings, instead of just looking for the overall.
* Use clustering recommendation.
* Build an hybrid model: another type of recommender systems are hybrid models which combine both the content-based and collaborative filtering algorithms.
* Incorporate more consumer features into the content-based model. We only have taken into account few attributes of the consumers- more information about the consumer may lead to a improved recommendation system.

References

[1] https://en.wikipedia.org/wiki/Recommender_system
[2] https://archive.ics.uci.edu/ml/datasets/Restaurant+%26+consumer+data
[3] Mooney RJ, Roy L (1999) Content-based book recommendation using learning for text categorization. In: Workshop on recommender systems: algorithms and evaluation
[4] M.B. Vivek et al., Proceedings of International Conference on Cognition and Recognition, Lecture Notes in Networks and Systems 14, 2018
[5] recommenderlab: A Framework for Developing and Testing Recommendation Algorithms, Michael Hahsler, SMU
[6] Course slides on Recommendation Systems