Restaurant Final Project IT

Michael Rains, Samousa Fofana, Alec Dudognon

11/21/2021

Contents

1 Introduction

Running a thriving local restaurant is not always as charming as first impressions appear. There are often all sorts of unexpected troubles popping up that could hurt business. One common predicament is that restaurants need to know how many customers to expect each day to effectively purchase ingredients and schedule staff members. This forecast is not easy to make because many unpredictable factors affect restaurant attendance, like weather and local competition. It is even harder for newer restaurants with little historical data. In this task, you are challenged to use reservation and visitation data to predict the total number of visitors to a restaurant for future dates. This information will help restaurants be much more efficient and allow them to focus on creating an enjoyable dining experience for their customers.

1.1 Problem Statement

In this task, we are challenged to use reservation and visitation data to predict the total number of visitors to a restaurant for future dates.

We have a certain amount of information about these restaurants like reservations, dates with the number of customers and reservations , the location of the restaurant etc…

Our main objective will be to create different model that we will use to predict, (i.e. linear, decision tree and random forest model), and choose the model that seems the best one.

2 Setup the software

The software used for the development of the study and the writing of the report is R[1]. The first step is to load the libraries using during the development of the report: tidyverse[2], lubridate[3], leaflet[4], tidymodels[5], rpart[6], rpart.plot[7], caTools [8], RMySQL[9] and mapdata[10].

2.1 Load relevant packages for analysis

```
library(tidyverse)
library(lubridate)
library(leaflet)
library(tidymodels)
library(rpart)
library(rpart.plot)
library(caTools)
library(RMySQL)
library(mapdata)
```

3 Analysis

3.1 Importing data from MySQL Workbench 8.0 CE

Before being able to deal with the data set in Rstudio in order to perform an exploratory data analysis and try to find the correct prediction method, the data must be correctly extracted from the mySQL database.

For this purpose the "RMySQL" package is used, with which it is possible to implement queries in RStudio.

All the following command will be just shown and not run in the chunk of report, they were used in a separate file. We will report them here to let you know how we import the table of the database in Rstudio.

First we connect Rstudio to our database through the use of the command **dbConnect**

mydb <- dbConnect(MySQL(), user="newuser", password="will-20", dbname="ITDAproject", host="127.0.0.1")

dbListTables(mydb) #will return the different names of the table in the database

Now we will crate data frames in R, by extracting the tables from the database, it will be done by writing down some queries through the use of the command **dbGetQuery**.

- Creation of *air_reserve*, it consist in the left join between the tables air_reserve and restaurant_info

air_reserve <- dbGetQuery(mydb, " SELECT R.*, RI.* FROM air_reserve R left JOIN restaurant_info RI USING(ID); ")

- Creation of *air_visit*, it consist in the left join between the tables air_visit and restaurant_info

air_visit <- dbGetQuery(mydb, " SELECT V.*, RI.* FROM air_visit V left JOIN restaurant_info RI USING(ID); ")

- Creation of *date_info*, creation of a data frame with the information present in date_info table

date_info <- dbGetQuery(mydb, " SELECT * FROM date_info; ")

- Creation of *restaurant_info*, creation of a data frame with the information present in restaurant_info table

restaurant_info <- dbGetQuery(mydb, " SELECT * FROM restaurant_info; ")

dbDisconnect(mydb) # It's polite to let the database know when you're done

- We ran the command **glimpse** to take a look to how many observations(rows) we have in each data frames and how many variables(columns) we have for each data frames.

glimpse(air_reserve) glimpse(air_visit) glimpse(date_info) glimpse(restaurant_info)

- The last step done consist in convert the data frames in csv files that we will use in the rest of the analysis

write.csv(air_reserve,"reservation_data.csv") write.csv(air_visit,"visit_data.csv") write.csv(date_info,"date_info.csv") write.csv(restaurant_info,"restaurant_info.csv")

3.2 Step 1 - read in the dataset

```
reservations <- read.csv("reservation_data.csv")
visits <- read.csv("visit_data.csv")
date_info <- read.csv("date_info.csv")
restaurant_info <- read.csv("restaurant_info.csv")
```

All the relevant data is read in - we started from a point where we ran SQL queries to get the restaurant information into the reservations and visits tables but need to do some variable manipulation to align the dates to get the date information to a point where we can join as some data sets store the date in different ways. For this we will use the **lubridate** package in R.

3.3 Step2- Take a look on the variable in the data set

In this part we will become aware of the different data set we have at our disposal :

- **Variable in reservations**

```
names(reservations)
```

- ID: id of the restaurant
- reserve_datetime: the date and time the reservation was made
- visit_datetime: the date and time of the reservation
- reserve_visitors: the number of visitors for that reservation

We will explain the meaning of the other variable in the section of restaurant_info, cause they are the result of the left join used before.

- **Variable in reservations**

```
names(visits)
```

- ID: id of the restaurant
- visit_date: the date of the visit
- visitors: the number of visitors to the restaurant on the date

We will explain the meaning of the other variable in the section of restaurant_info, cause they are the result of the left join used before.

- **Variable in date_info**

```
names(date_info)
```

- calendar_date: represent the date in the format year-month-day
- day_of_week: represent the day of the week in label
- holiday_flg: it tell us if the calendar date is a holiday day in Japan
- **Variable in restaurant_info**

```
names(restaurant_info)
```

- ID: id of the restaurant
- air_genre_name: the genre name of the restaurant
- air_area_name: the area name in Japan of the restaurant
- latitude: the latitude of the position of the restaurant
- longitude: the longitude of the position of the restaurant

3.4 Step 3 - Exploratory data analysis

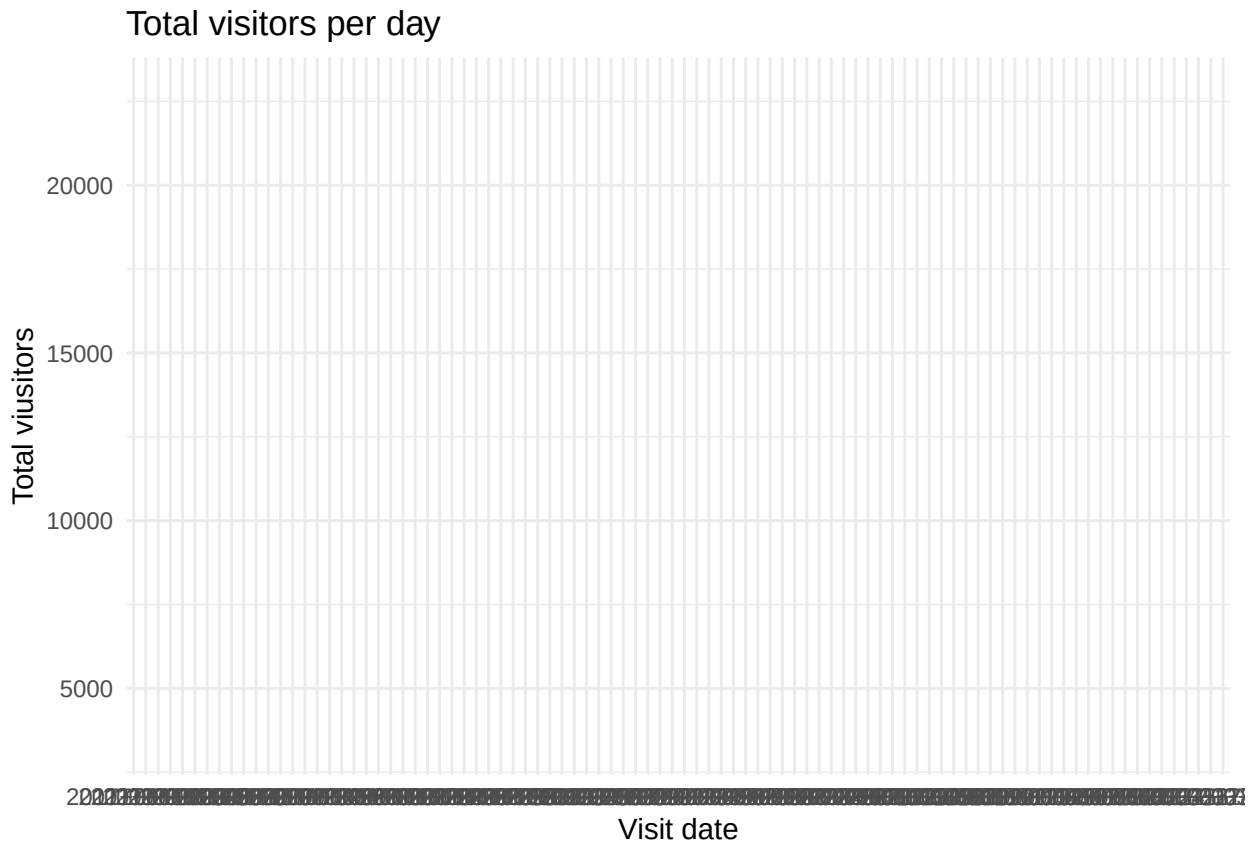
In the following section we will try to understand the main characteristic of the variable that we have in each data sets before proceeding with the combination of them for a more detailed analysis.

The information that we will obtain in this section will be taken in consideration to build the different models for the prediction.

###**Table visits**

- The below graph will show us the **overall total visitors per day**

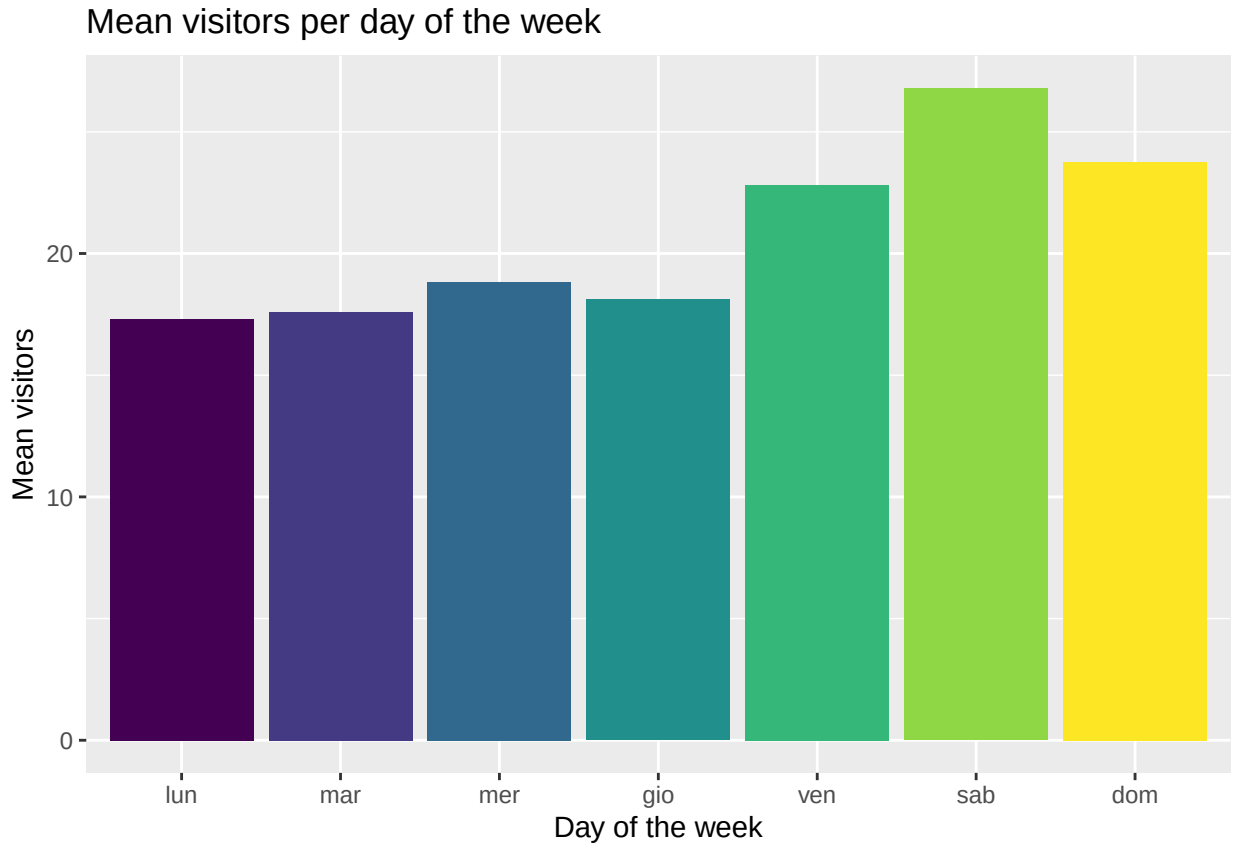
```
visits %>%
  group_by(visit_date) %>%
  summarise(all_visitors = sum(visitors)) %>%
  ggplot(aes(x=visit_date,y=all_visitors)) +
  geom_line(col = "blue") +
  labs(x = "Visit date", y = "Total viusitors",title = "Total visitors per day")+
  theme_minimal()
```



We can easily see that we have an increasing trend during the time, and we can also en-light the fact that during the week we always have a peak which is in correspondence of the weekend.

- The below graph will show the overall mean visitors per day of the week

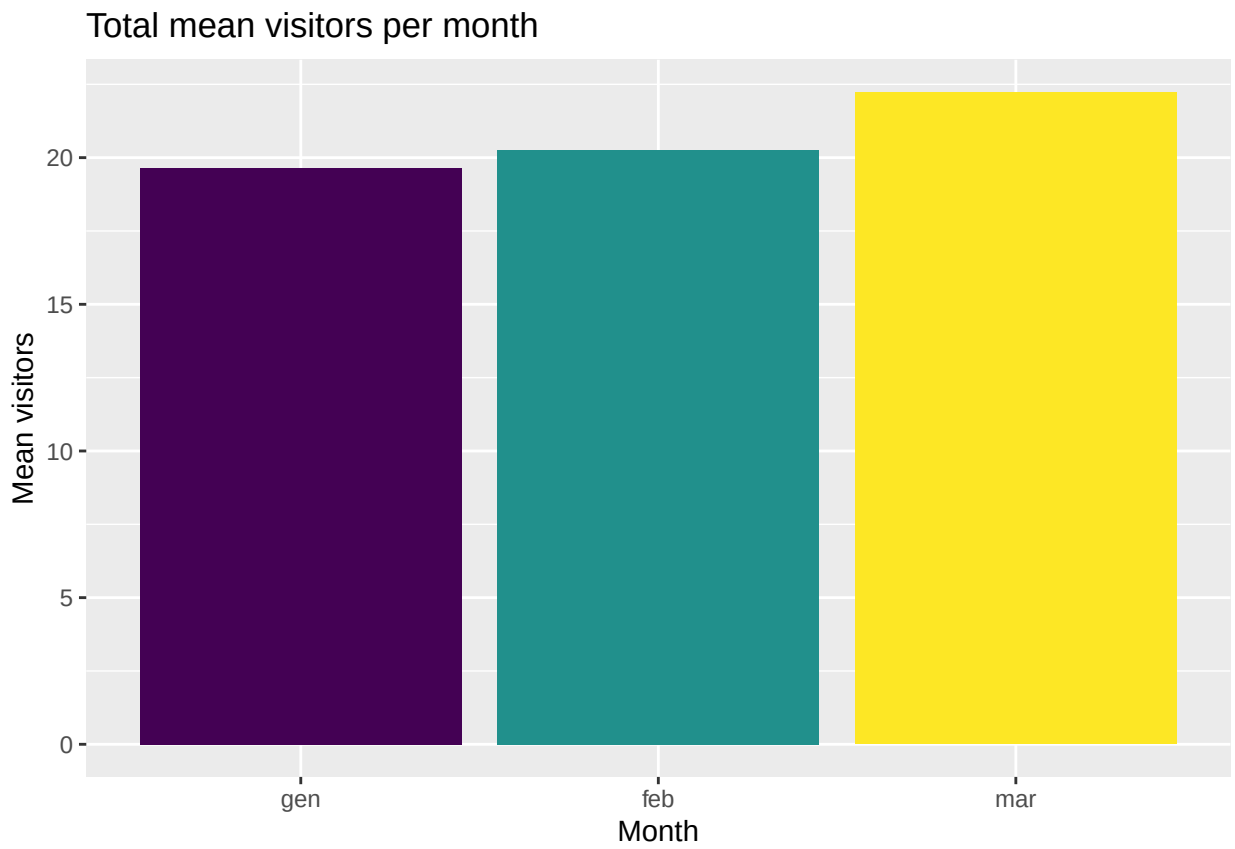
```r
visits %>%
  mutate(week_day = wday(visit_date, label = TRUE, week_start = 1)) %>%
  group_by(week_day) %>%
  drop_na()%>%
  summarise(visits = mean(visitors)) %>%
  ggplot(aes(week_day, visits, fill = week_day)) +
  geom_col() +
  theme(legend.position = "none") +
  labs(x = "Day of the week", y = "Mean visitors", title = "Mean visitors per day of the week")
```



From the result we can easily see that the day with the higher mean fall during the Friday, Saturday and Sunday. While the day with the lower mean are Monday and Tuesday.

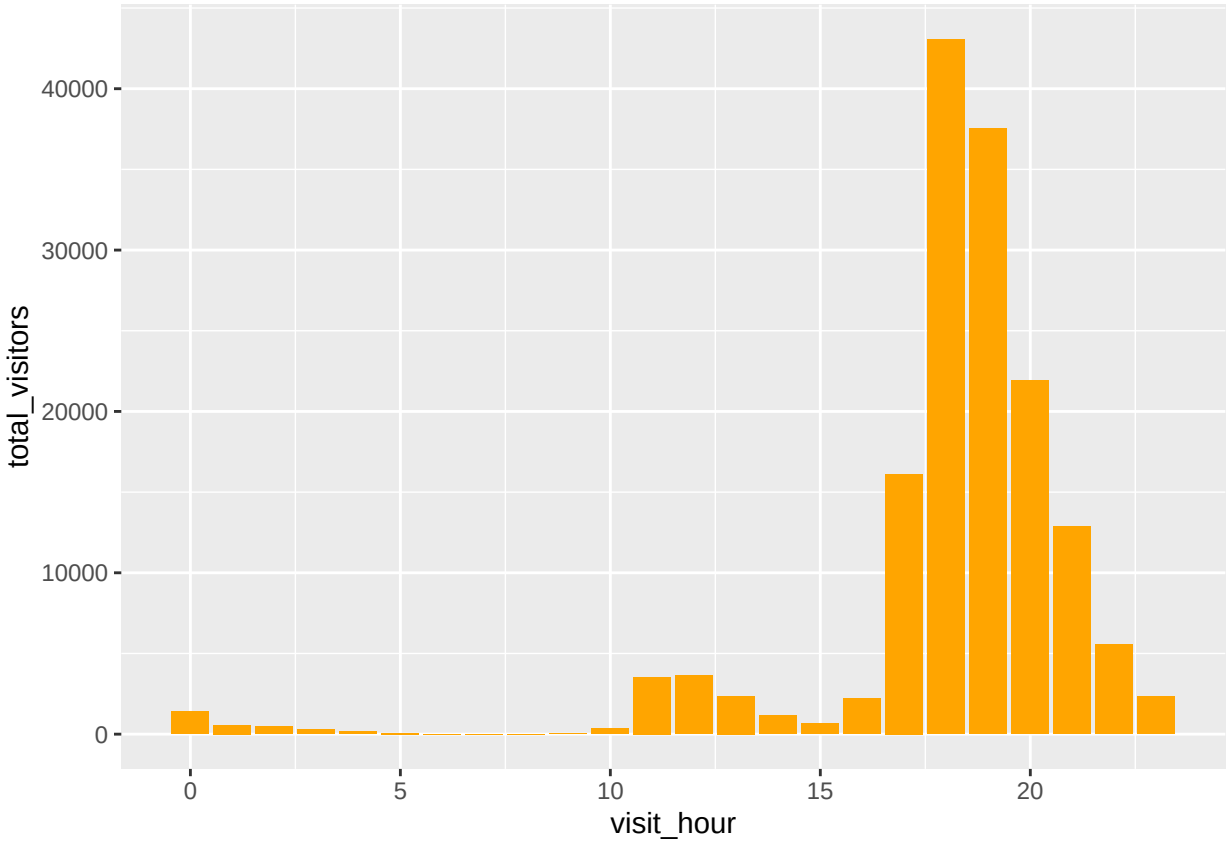
- The below graph will show the total mean visitors per month

```r
visits %>%
  mutate(month = month(visit_date, label = TRUE)) %>%
  group_by(month) %>%
  drop_na()%>%
  summarise(visits = mean(visitors)) %>%
  ggplot(aes(month, visits, fill = month)) +
  geom_col() +
  theme(legend.position = "none") +
  labs(x = "Month", y = "Mean visitors", title= "Total mean visitors per month")
```



3.4.1 Reservation

```
reservations_graph <- reservations %>%
  mutate(reserve_date = date(reserve_datetime),
         reserve_hour = hour(reserve_datetime),
         reserve_wday = wday(reserve_datetime, label = TRUE, week_start = 1),
         visit_date = date(visit_datetime),
         visit_hour = hour(visit_datetime),
         visit_wday = wday(visit_datetime, label = TRUE, week_start = 1),

         )

reservations_graph %>%
  group_by(visit_hour) %>%
  drop_na() %>%
  summarise(total_visitors = sum(reserve_visitors)) %>%
  ggplot(aes(visit_hour, total_visitors)) +
  geom_col(fill="orange")
```



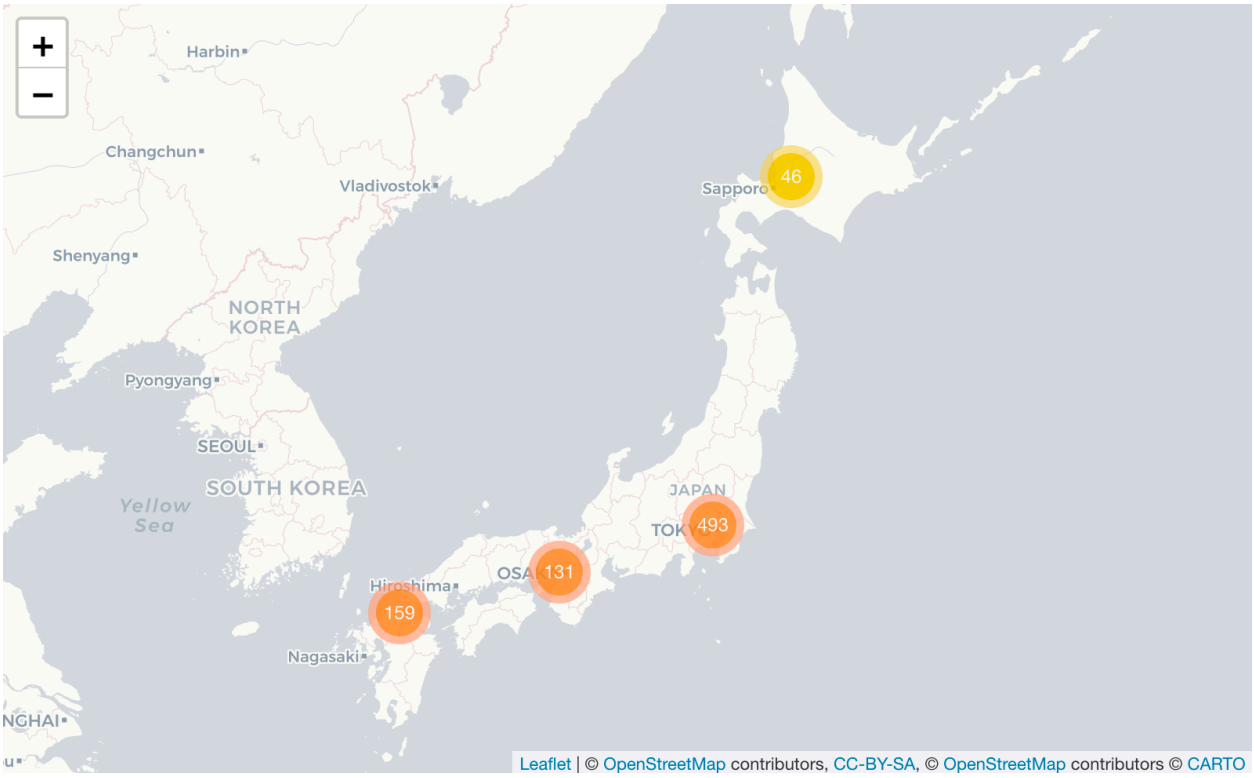
From this graph we can easily see that the majority of the client usually do reservation for the evening. Between 5 AM and 10 AM, we can see that we don't have values principally relate to the fact that restaurant during that time are usually closed.

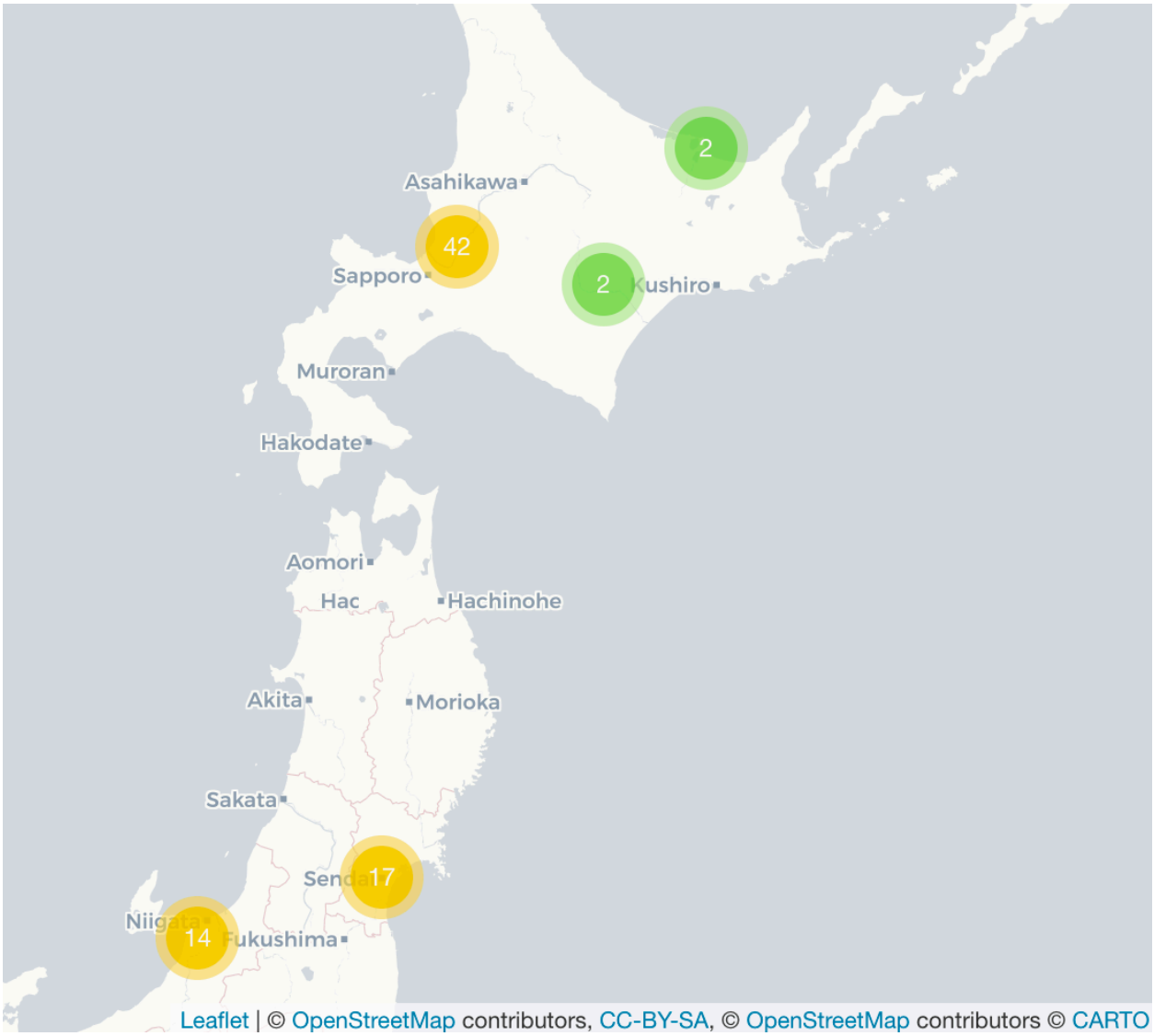
3.4.2 Restaurant Info

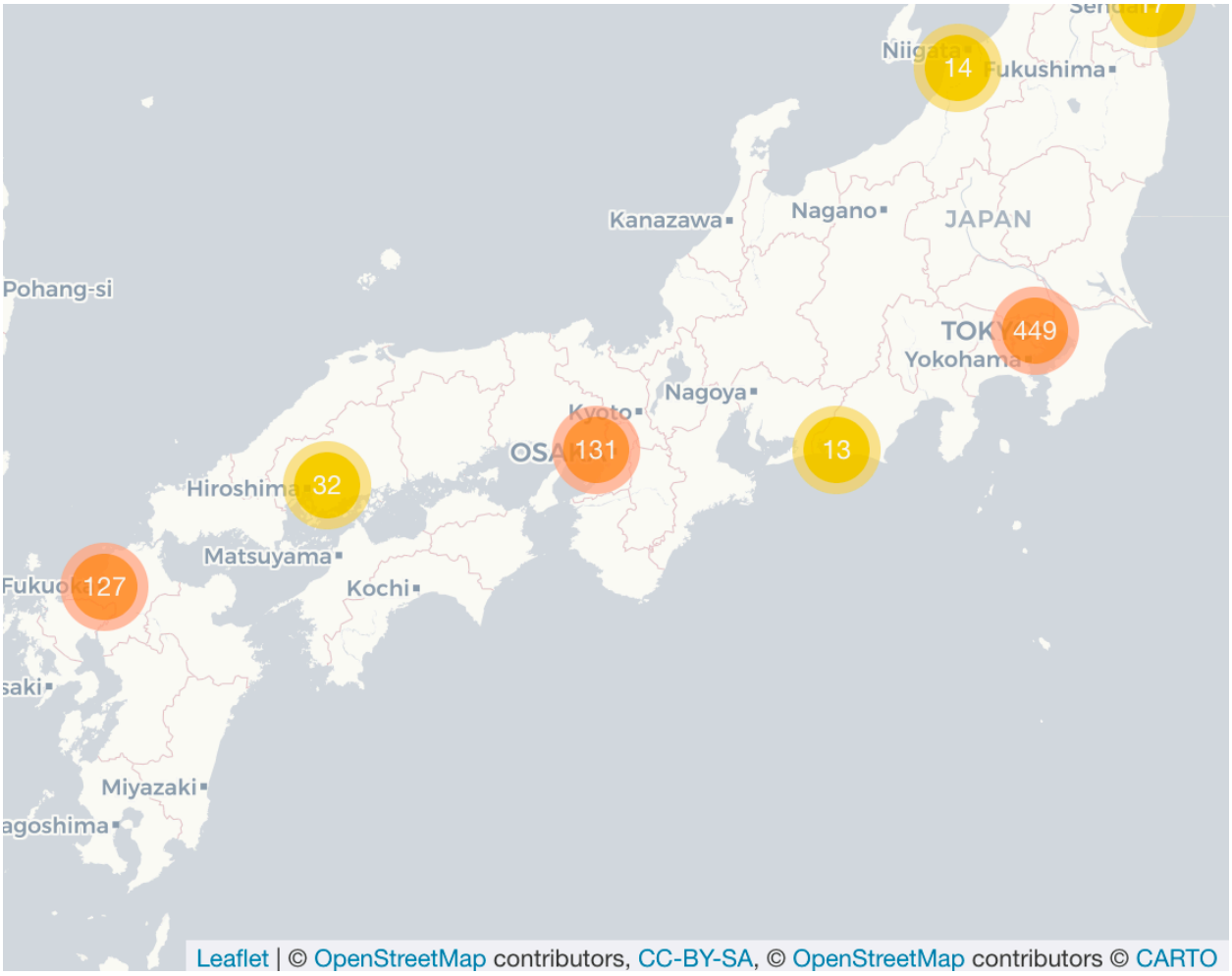
Plotting the position of the different restaurant

```
# leaflet(restaurant_info) %>%
#   addTiles() %>%
#   addProviderTiles("CartoDB.Positron") %>%
#   addMarkers(~longitude, ~latitude,
#              popup = ~ID, label = ~air_genre_name,
#              clusterOptions = markerClusterOptions())
```

The leaflet package only outputs HTML formats thus we will insert images from the interactive map we have generated


Harbin
Changchun
Vladivostok
Sapporo
46
Shenyang
NORTH KOREA
Pyongyang
SEOUL
SOUTH KOREA
Yellow Sea
JAPAN
TOK
493
OSA
131
Hiroshima
159
Nagasaki
NGHAI
Leaflet | © OpenStreetMap contributors, CC-BY-SA, © OpenStreetMap contributors © CARTO

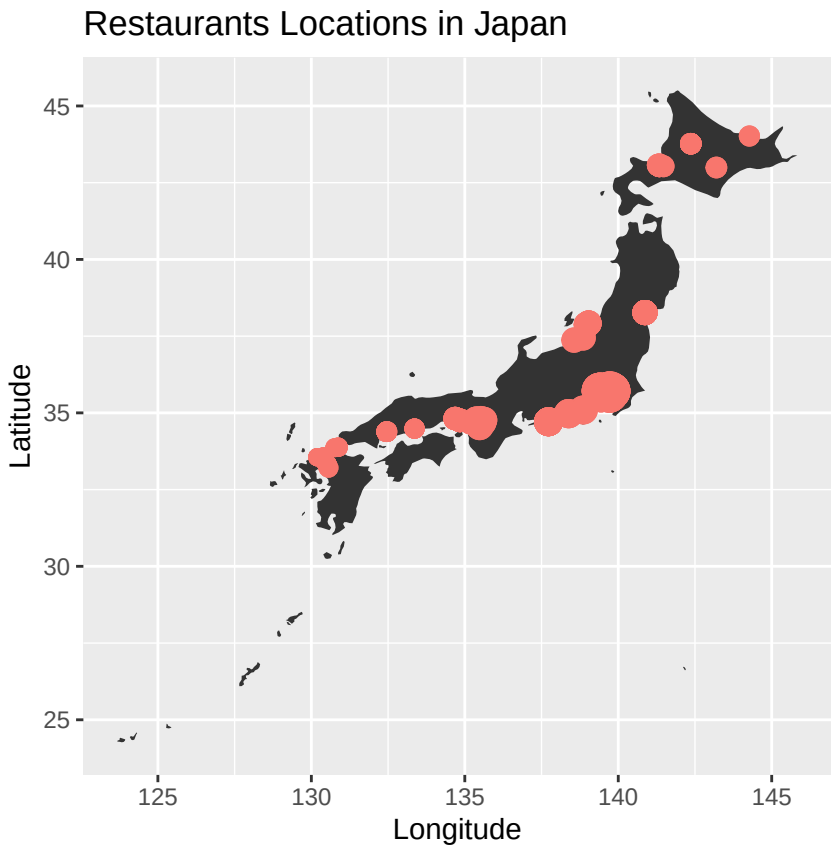

2
Asahikawa
42
Sapporo
2
Kushiro
Muroran
Hakodate
Aomori
Hac
Hachinohe
Akita
Morioka
Sakata
17
14
Fukushima
Leaflet | © OpenStreetMap contributors, CC-BY-SA, © OpenStreetMap contributors © CARTO



The map show us that many restaurants share common coordinates, since those coordinates refer to the area of the restaurant. By clicking on the clusters they will break into smaller clusters and at the end into the individual restaurants, which are labelled by their genre name . If we click on the single markers we will able to visualize the id of each restaurant in the cluster.

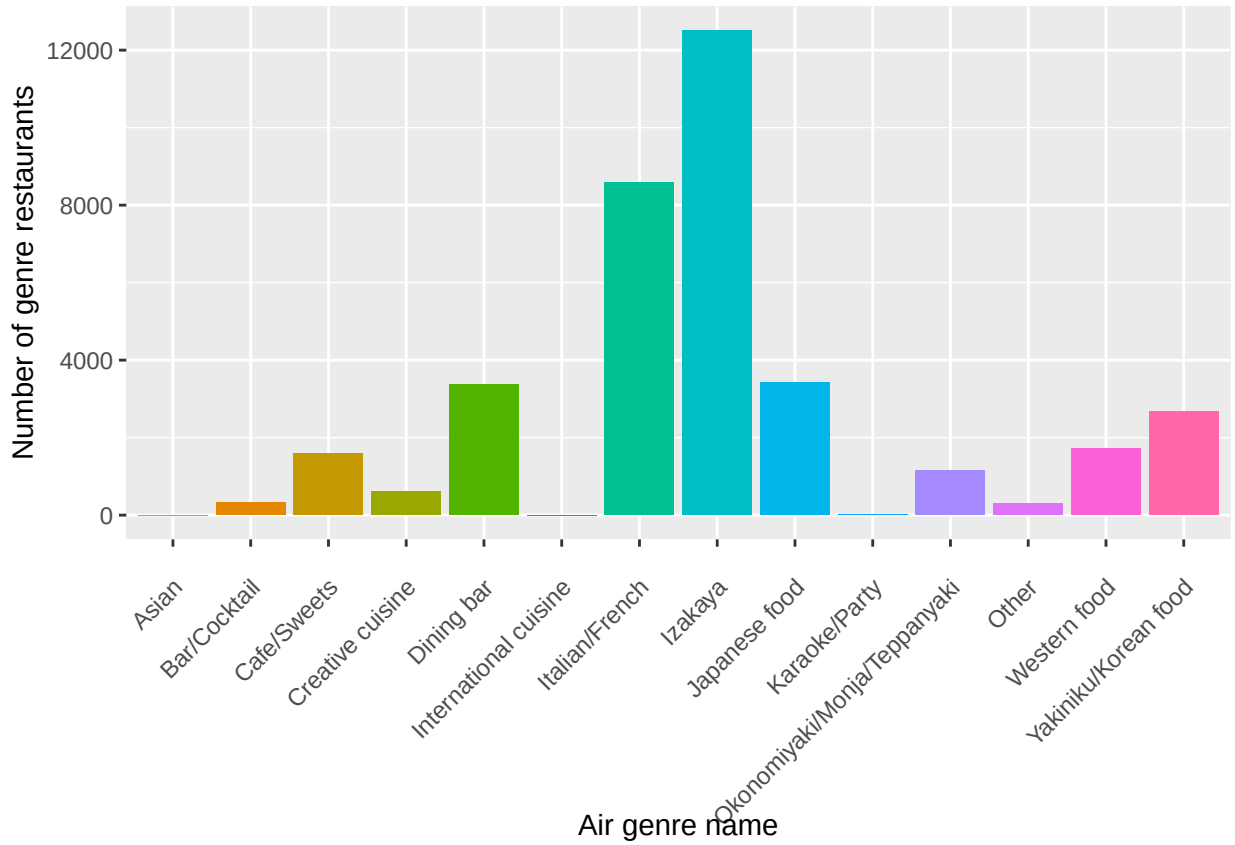
Let's also take a look on the location of the restaurants without the creation of the cluster.

```
japan <- ggplot2::map_data('world2', 'japan')

ggplot()+
  geom_polygon(data=japan, aes(x=long,y=lat, group=group))+
  geom_point(data=restaurant_info, aes(x=longitude, y=latitude, color= "red", size= air_area_name)) +
  coord_equal(ratio=1) +
  theme(legend.position = "none")+
  labs(x= "Longitude",y= "Latitude",title="Restaurants Locations in Japan")
```



- Let's have a look on the genre name frequency

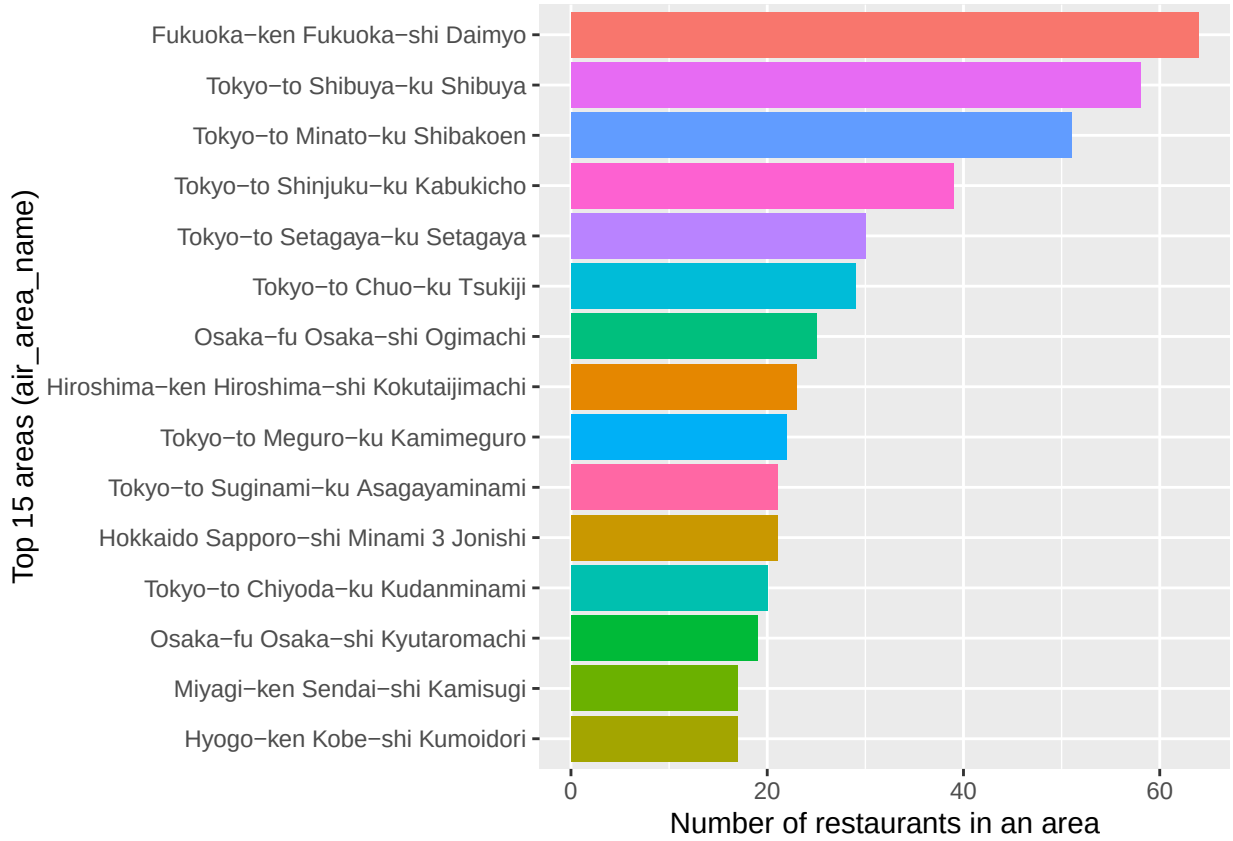
```
reservations %>%
  group_by(air_genre_name) %>%
  count() %>%
  ggplot(aes(air_genre_name, n, fill = air_genre_name)) +
  geom_col() +
  theme(legend.position = "none",axis.text.x  = element_text(angle=45, hjust=1, vjust=0.9))+
  labs(x = "Air genre name", y = "Number of genre restaurants")
```



From the graph, we see that there are lots of **Izakaya** restaurants, (which indicates a typical Japanese place where drinks are served accompanied by food), in our data, followed by **Italian/French** restaurants. We have only 2 Asian restaurants, 3 International cuisine and 24 Karaoke/Party places in the data set.

- Let's have a look on the frequency of restaurant in the different area

```
restaurant_info %>%
  group_by(air_area_name) %>%
  count() %>%
  ungroup() %>%
  top_n(15,n) %>%
  ggplot(aes(reorder(air_area_name, n, FUN = min) ,n, fill = air_area_name)) +
  geom_col() +
  theme(legend.position = "none") +
  coord_flip() +
  labs(x = "Top 15 areas (air_area_name)", y = "Number of restaurants in an area")
```



Fukuoka has the largest number of restaurants per area, followed by many Tokyo areas.

3.5 Step 4 - Examine the variables in the datasets

In order to do that we will use the command **skimr**, is designed to provide summary statistics about variables in data frames.

```
for(data in list(reservations,visits,date_info,restaurant_info)) {

  print(skimr::skim(data))
  print("--------------------------------------")
}
```

```
## -- Data Summary ------------------------
##                            Values
## Name                       data
## Number of rows             36259
## Number of columns          10
## _________________________
## Column type frequency:
##   character                6
##   numeric                  4
## _________________________
## Group variables            None
##
```

```
## -- Variable type: character ---------------------------------------------------
## # A tibble: 6 x 8
##   skim_variable    n_missing complete_rate   min   max empty n_unique whitespace
## * <chr>                <int>         <dbl> <int> <int> <int>    <int>      <int>
## 1 ID                       0             1    13    15     0      275          0
## 2 visit_datetime           0             1    19    19     0     1617          0
## 3 reserve_datetime         0             1    19    19     0     1892          0
## 4 ID..5                    0             1    13    15     0      275          0
## 5 air_genre_name           0             1     5    28     0       14          0
## 6 air_area_name            0             1    21    45     0       64          0
##
## -- Variable type: numeric -----------------------------------------------------
## # A tibble: 4 x 11
##   skim_variable    n_missing complete_rate     mean       sd    p0    p25
## * <chr>                <int>         <dbl>    <dbl>    <dbl> <dbl>  <dbl>
## 1 X                        0             1 18130    10467.     1   9066.
## 2 reserve_visitors         0             1     4.31     4.56   1      2
## 3 latitude                 0             1    36.4      3.10  33.2   34.7
## 4 longitude                0             1   137.       3.78 130.   135.
##       p50     p75    p100 hist
## *   <dbl>   <dbl>   <dbl> <chr>
## 1 18130   27194.  36259   <U+2587><U+2587><U+2587><U+2587><U+2587>
## 2     3       5     100   <U+2587><U+2581><U+2581><U+2581><U+2581>
## 3    35.6    35.7    44.0 <U+2587><U+2586><U+2582><U+2581><U+2583>
## 4   140.    140.    144.  <U+2583><U+2583><U+2581><U+2587><U+2581>
## [1] "--------------------------------------"
## -- Data Summary ------------------------
##                            Values
## Name                       data
## Number of rows             61803
## Number of columns          9
## _________________________
## Column type frequency:
##   character                5
##   numeric                  4
## _________________________
## Group variables            None
##
## -- Variable type: character ---------------------------------------------------
## # A tibble: 5 x 8
##   skim_variable  n_missing complete_rate   min   max empty n_unique whitespace
## * <chr>              <int>         <dbl> <int> <int> <int>    <int>      <int>
## 1 ID                     0             1    13    15     0      825          0
## 2 visit_date             0             1    10    10     0       90          0
## 3 ID..4                  0             1    13    15     0      825          0
## 4 air_genre_name         0             1     5    28     0       14          0
## 5 air_area_name          0             1    21    45     0      103          0
##
## -- Variable type: numeric -----------------------------------------------------
## # A tibble: 4 x 11
##   skim_variable n_missing complete_rate    mean       sd    p0     p25     p50
## * <chr>             <int>         <dbl>   <dbl>    <dbl> <dbl>   <dbl>   <dbl>
## 1 X                     0             1 30902   17841.     1   15452.  30902
## 2 visitors              0             1    20.8    16.7    1       9      17
```

```
## 3 latitude              0             1    35.6     2.06  33.2    34.7    35.7
## 4 longitude             0             1   137.      3.66 130.    135.    140.
##       p75    p100 hist
## *   <dbl>   <dbl> <chr>
## 1 46352.  61803   <U+2587><U+2587><U+2587><U+2587><U+2587>
## 2    29     877   <U+2587><U+2581><U+2581><U+2581><U+2581>
## 3    35.7    44.0 <U+2586><U+2587><U+2581><U+2581><U+2581>
## 4   140.    144.  <U+2582><U+2582><U+2581><U+2587><U+2581>
## [1] "--------------------------------------"
## -- Data Summary ------------------------
##                            Values
## Name                       data
## Number of rows             517
## Number of columns          4
## _______________________
## Column type frequency:
##   character                2
##   numeric                  2
## _______________________
## Group variables            None
##
## -- Variable type: character -------------------------------------------------------
## # A tibble: 2 x 8
##   skim_variable n_missing complete_rate   min   max empty n_unique whitespace
## * <chr>             <int>         <dbl> <int> <int> <int>    <int>      <int>
## 1 calendar_date         0             1    10    10     0      517          0
## 2 day_of_week           0             1     6     9     0        7          0
##
## -- Variable type: numeric ---------------------------------------------------------
## # A tibble: 2 x 11
##   skim_variable n_missing complete_rate     mean      sd    p0   p25   p50   p75
## * <chr>             <int>         <dbl>    <dbl>   <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 X                     0             1 259      149.        1   130   259   388
## 2 holiday_flg           0             1   0.0677   0.251     0     0     0     0
##    p100 hist
## * <dbl> <chr>
## 1   517 <U+2587><U+2587><U+2587><U+2587><U+2587>
## 2     1 <U+2587><U+2581><U+2581><U+2581><U+2581>
## [1] "--------------------------------------"
## -- Data Summary ------------------------
##                            Values
## Name                       data
## Number of rows             829
## Number of columns          6
## _______________________
## Column type frequency:
##   character                3
##   numeric                  3
## _______________________
## Group variables            None
##
## -- Variable type: character -------------------------------------------------------
## # A tibble: 3 x 8
##   skim_variable  n_missing complete_rate   min   max empty n_unique whitespace
```

```
## * <chr>              <int>         <dbl> <int> <int> <int>    <int>      <int>
## 1 ID                     0             1    13    15     0      829          0
## 2 air_genre_name         0             1     5    28     0       14          0
## 3 air_area_name          0             1    21    45     0      103          0
##
## -- Variable type: numeric -------------------------------------------------------
## # A tibble: 3 x 11
##   skim_variable n_missing complete_rate  mean     sd    p0   p25   p50   p75
## * <chr>             <int>         <dbl> <dbl>  <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 X                     0             1 415   239.     1   208   415   622
## 2 latitude              0             1  35.6   2.08  33.2  34.7  35.7  35.7
## 3 longitude             0             1 137.    3.65 130.  135.  140.  140.
##    p100 hist
## * <dbl> <chr>
## 1 829   <U+2587><U+2587><U+2587><U+2587><U+2587>
## 2  44.0 <U+2586><U+2587><U+2581><U+2581><U+2581>
## 3 144.  <U+2582><U+2582><U+2581><U+2587><U+2581>
## [1] "------------------------------------"
```

3.6 Step 5 - Convert the date variables with lubridate

In the following chunk through the use of the package **lubridate** we modify the type of the variable of visit_datetime and reserve_datetime from a character to a date with hour, minute and seconds.

```
reservations$visit_datetime <- ymd_hms(reservations$visit_datetime)
reservations$reserve_datetime <- ymd_hms(reservations$reserve_datetime)

visits$visit_date <- ymd(visits$visit_date)

date_info$calendar_date <- ymd(date_info$calendar_date)
```

3.7 Step 6 - Examine the dates in each dataset

To understand in which time horizon we are working we create the following chunk.

```
tribble(
  ~"variable",                      ~"Reservation",          ~"Visits", ~"Date-Info",
  "visit min", min(reservations$visit_datetime), min(visits$visit_date), NA,
  "visit max", max(reservations$visit_datetime), max(visits$visit_date), NA,
  "reserve min", min(reservations$reserve_datetime), NA, NA,
  "reserve max", max(reservations$reserve_datetime), NA, NA,
  "Other min", NA                                  , NA, min(date_info$calendar_date),
  "Other max", NA                                  , NA, max(date_info$calendar_date)

)
```

```
## # A tibble: 6 x 4
##   variable    Reservation         Visits     `Date-Info`
##   <chr>       <dttm>              <date>     <date>
## 1 visit min   2017-01-01 10:00:00 2017-01-01 NA
## 2 visit max   2017-05-28 23:00:00 2017-03-31 NA
```

```
## 3 reserve min 2017-01-01 00:00:00 NA         NA
## 4 reserve max 2017-03-31 23:00:00 NA         NA
## 5 Other min   NA                  NA         2016-01-01
## 6 Other max   NA                  NA         2017-05-31
```

From this output we can see that the first visit that we have inside the data_set reservation is 2017-01-01 at 10:00 while the last visit that we have is at on 2017-05-28 at 23:00. The first call that we receive for the reservation of a table is on the date 2017-01-01 at 00:00 while the last call for a reservation is on the date 2017-03-31 23:00.

Looking at the data set Visits also the first visit that we have inside the data set is on 2017-01-01 while the last visit that we receive is different from the one of the reservation cause it is on 2017-03-31.

We also evaluate some information relate to the data_set date_info and we get that the first date that we have is 2016-01-01 while the last one is 2017-05-31.

3.8 Step 7 - Data Modelling

3.8.1 Now we will summarize by day.

We perform this process, in order to have the observations of all data sets relate to only the date and not also the hour. It will also help us in being align with the prediction that we have to perform, cause the prediction will be relate to a day of the calendar and not also to the different hour of a day.

```
reserve_data_day_summed <- reservations %>% mutate(reserveDay = day(reserve_datetime))

reservations<- reservations%>%mutate(day_reservation = day(reservations$reserve_datetime))
reservations<- reservations%>%mutate(month_reservation = month(reservations$reserve_datetime))
reservations<- reservations%>%mutate(year_reservation = 2017)
reservations<- reservations%>%mutate(date = '')

reservations$date <- as.Date(with(reservations, paste(year_reservation, month_reservation, day_reservati

reservation_per_day <- reservations %>%
  mutate(day_mon_rest=paste(day_reservation,month_reservation,ID))%>%
  group_by(ID,day_mon_rest,date) %>%
  summarise(Tot_visit_day_from_Reservations = sum(reserve_visitors),
            count_of_reservations = n())

head(reservation_per_day)
```

```
## # A tibble: 6 x 5
## # Groups:   ID, day_mon_rest [6]
##   ID            day_mon_rest       date       Tot_visit_day_fr~ count_of_reserv~
##   <chr>         <chr>              <date>                 <int>            <int>
## 1 restaurant_ 1 1 2 restaurant_ 1  2017-02-01                14                5
## 2 restaurant_ 1 1 3 restaurant_ 1  2017-03-01                63                7
## 3 restaurant_ 1 10 2 restaurant_ 1 2017-02-10                 2                1
## 4 restaurant_ 1 10 3 restaurant_ 1 2017-03-10                30                1
## 5 restaurant_ 1 11 1 restaurant_ 1 2017-01-11                 4                1
## 6 restaurant_ 1 11 3 restaurant_ 1 2017-03-11                20                7
```

We also create a new data set called **reservation_per_day**, where we create two new column that would be usefull in carrying out the analysis:

- Tot_visit_day_from_Reservations : the total number of visitors that a restaurant receive per day from the reservations
- count_of_reservations : number of reservation that a restaurant have for a date

3.8.2 Forming modeling dataset

Now we've created a summarized version of the reservation data set and we need to join this into the visits data set to form our total data set with all of the variables we will use.

```
# visits <- visits %>% add_row(ID="restaurant_ 292",air_genre_name="Cafe/Sweets",air_area_name="T?ky?-t

date_info$calendar_date=ymd(date_info$calendar_date)

combinedDataset <- visits %>% mutate(date = visit_date) %>%  left_join(reservation_per_day,by = c('date
  mutate(calendar_date=date) %>% left_join(date_info,by='calendar_date')

combinedDataset <- combinedDataset %>% left_join(
  y=(combinedDataset %>% group_by(air_genre_name) %>% summarise(avgDailyVisitsGenre = mean(visitors))),
by="air_genre_name")
```

The result is the **combineDataset** with all the variable that we mentioned before.

In the last part of the previous chunk we also create a new column wich represent the mean daily visitor per restaurant genre name.

3.9 Step 8 - Linear regression models

3.9.1 Create Training and test

We will create the training and test set that we will use. Specifically we use the train set to create the model and then use the test set to validate our model and to check if the different model present over fit.

The approach utilized in order to split the data set is the following:

- The **trainset** will have all the variable relate to a date lower or equal to the 10 March of 2017
- The **testset** will have all the variable relate to a date greater or equal to the 10 March of 2017

We adopt this criteria to have a testset that last 3 weeks, so the same length of the final prediction that we will deliver.

```
trainset <- subset(combinedDataset,visit_date <= as.Date("2017-03-10"))
testset <- subset(combinedDataset,visit_date > as.Date("2017-03-10"))
```

3.9.2 Now we will begin to fit with linear regression models to start.

This is a simpler modeling technique and we want to start here to see if by chance it is the best fitting type of model.

```
#mod1<-lm(visitors ~ date + air_genre_name+latitude+longitude+count_of_reservations, data=trainset)
#
#summary(mod1)$r.squared
```

3.9.2.1 Model 1 The first trial show us that this model is not so good, the first information that support this sentence is relate to the R-sqaured, (0.03958), which is very low.Also the most of the coefficient are not significant taking a look to the p-values.

3.9.3 Model 2

```
#mod2<- lm( visitors ~ date + Tot_visit_day_from_Reservations + day_of_week + holiday_flg + air_genre_n
#
#summary(mod2)$r.squared
```

With the second trial we can see that we are improving the R-squared increas to 0.1721. We reach this improvement by removing the variable **count_of_reservations**, and adding the variables **holiday_flg** and **Tot_visit_day_from_Reservations**.

3.9.4 Model 3

```
# mod3<- lm( visitors ~ visit_date + Tot_visit_day_from_Reservations + day_of_week + holiday_flg + air_
# summary(mod3)$r.squared
# max rsq achieved to this point
print(.252)
```

```
## [1] 0.252
```

date + Tot_visit_day_from_Reservations + day_of_week + holiday_flg + air_genre_name + latitude + longitude

In the third trial we improve again the R-squared increas to 0.252. We reach this improvement by removing the variable **longitude** and **latitude**, and adding the variables **air_area_name** and substituting the **date**,(relate to the date in which reservation was made), with **visit_date** ,(day for which the reservation is made).

3.9.4.1 Post Model 3 Conclusions To this point we have tried a few different iterations to fairly poor results. **After this we will include ID** as we think that it might be important.

Ultimately in this sort of modeling situation, the restaurant themselves will have an effect on the final visitor total. Think of this similar to a sports team or predicting attendance for La Liga, the team being predicted will no doubt be one of the largest predictors.

3.9.5 Model 4

```
# mod4<- lm( visitors ~  ID + visit_date + Tot_visit_day_from_Reservations + day_of_week + holiday_flg
# summary(mod4)$r.squared
```

Let's comment a bit the fourth model, as we supposed taking in consideration the ID of the restaurant in the model we increase a lot the value of our R-squared reaching the value 0.5017, and a lot of levels of the ID result significant in the model.

3.9.6 Model 5

Another few iterations - we realized that visit date wouldn't be of any use for the dataset we are predicting (and would make the model not usable), thus we've swapped it for **day__of__week**.

```
#mod5<- lm( visitors ~  ID + day_of_week + holiday_flg + air_genre_name + air_area_name, data=trainset)
#summary(mod5)$r.squared
```

After running the model the R-squared decrease a bit to 0.4978, but we decide to perform the validation with this model, cause the coefficient in this model are more significant.

3.9.7 Validation of the model

In the previous chunck we performed the validation of the mod 5, we create a prediction through this model relate to the visitors in the test set and than we calculate the R-squared taking in consideration the values predicted and the real values available. We obtained a result of 0.4823 which is good because is close and not higher to R-squared obtained in the creation of the model.

3.9.8 Filling the submission file

In the following chunk we will do the prediction for the restaurant and date provide in the submission file through the use of the mod5.

Initially we use some command to extract the the date and the respective restaurant. After, we join the data set with other data set that contain the information needed to perform the prediction.

Then we launch the prediction and attached the results in the column of the visitors in the submission file, we reset the order of the rows as it was initially, and finally we write a csv file with the result.

3.10 Step 9 - Decision tree

3.10.1 Now let's explore decision tree fits

We will now use a very powerful supervised learning algorithm **random forest** to try and fit to our model and see how it performs. It is an extrapolation of a decision tree algorithm.

```
combinedDataset=combinedDataset %>% left_join(combinedDataset%>%group_by(ID,day_of_week)%>%summarise(av
```

```
set.seed(42)
```

```
combinedDataset= combinedDataset%>%select(-ID..4,-X.y)%>%rename(X=X.x)
```

```
data_split <- initial_time_split(combinedDataset, prop = 0.75)

tm_train <- training(data_split)
tm_test <- testing(data_split)

tm_rec <- tm_train %>%
  recipe(visitors ~ . ) %>%
  step_normalize(all_predictors())

# Show the result of our recipe
tm_rec
```

```
## Recipe
##
## Inputs:
##
##       role #variables
##    outcome          1
##  predictor         16
##
## Operations:
##
## Centering and scaling for all_predictors()
```

```
rf_spec <- rand_forest(mode = "regression") %>%
  set_engine("ranger")

rf_spec
```

```
## Random Forest Model Specification (regression)
##
## Computational engine: ranger
```

```
## Random Forest Model Specification (regression)
##
## Computational engine: ranger
rf_fit <- rf_spec %>%
  fit(visitors ~ .,
    data = tm_train
  )

print("RF R-Squared")
```

```
## [1] "RF R-Squared"
```

```
rf_fit$fit$r.squared
```

```
## [1] 0.6422863
```

We started from the splitting of the data set in the train and test set by the use of the function **initial_time_split**. We perform the random forest taking in consideration all the variable available, and we obtained a R-squared of 0.639.

3.10.2 Creation of the regression tree using rpart

We invoke 3 new packages to be able to run the decision tree method :

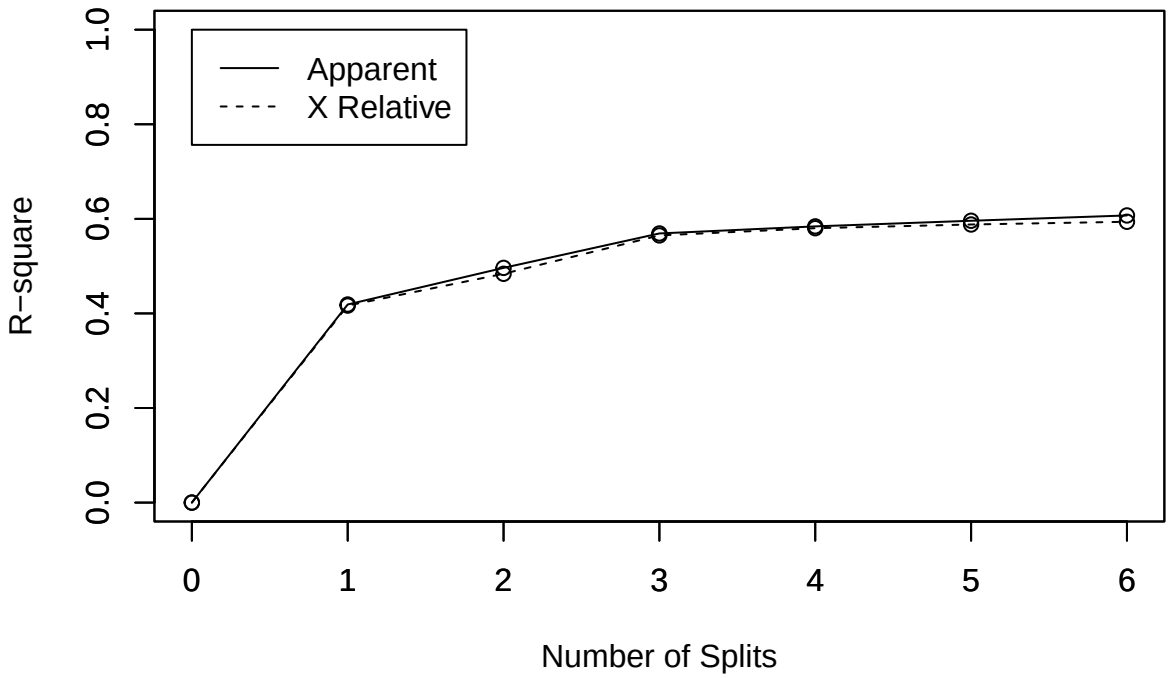
```
set.seed(122)

combinedDataset$ID=as.factor(combinedDataset$ID)
combinedDataset$air_genre_name=as.factor(combinedDataset$air_genre_name)
combinedDataset$air_area_name=as.factor(combinedDataset$air_area_name)
combinedDataset=combinedDataset%>%select(-day_mon_rest)

combinedDataset$count_of_reservations[is.na(combinedDataset$count_of_reservations)] <- 0
combinedDataset$Tot_visit_day_from_Reservations[is.na(combinedDataset$Tot_visit_day_from_Reservations)]

trainset2 <- subset(combinedDataset,visit_date <= as.Date("2017-03-10"))
testset2 <- subset(combinedDataset,visit_date > as.Date("2017-03-10"))



firsttree <- rpart(
  formula = visitors ~ ID + air_area_name + latitude + longitude + air_genre_name+ avgDailyVisitsGenre +
  data    = trainset2,
  method  = "anova"
  )
firsttree
```

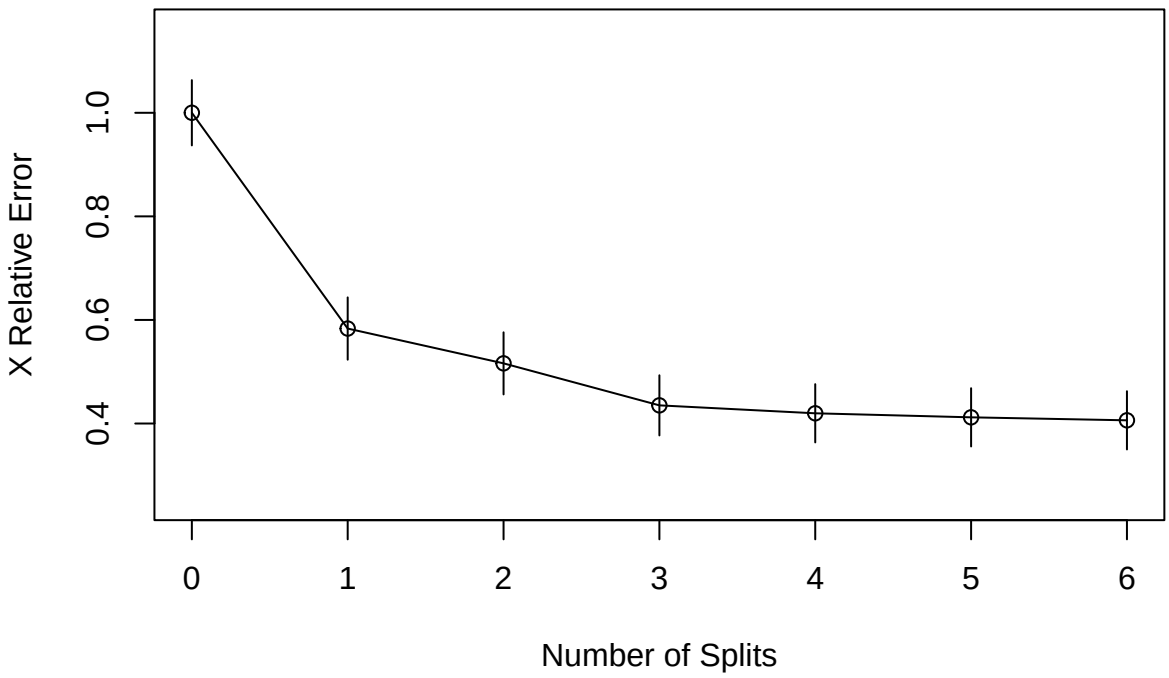
```
## n= 46763
##
## node), split, n, deviance, yval
##       * denotes terminal node
##
##  1) root 46763 11971780.0 20.044030
##    2) avg_of_day_id< 25.60769 32958  2674783.0 13.341860
##      4) avg_of_day_id< 14.83974 18881   664618.8  8.901753
##        8) avg_of_day_id< 8.630682 8283   127685.3  5.881806 *
##        9) avg_of_day_id>=8.630682 10598   402351.3 11.262030 *
##      5) avg_of_day_id>=14.83974 14077  1138674.0 19.297220 *
##    3) avg_of_day_id>=25.60769 13805  4282150.0 36.044770
##      6) avg_of_day_id< 41.96154 10340  1558884.0 31.300580
##       12) avg_of_day_id< 32.64103 5742   726847.9 27.976840 *
##       13) avg_of_day_id>=32.64103 4598   689386.6 35.451280 *
##      7) avg_of_day_id>=41.96154 3465  1796059.0 50.202020
##       14) avg_of_day_id< 56.96154 2725   635130.6 46.458720 *
##       15) avg_of_day_id>=56.96154 740   982135.9 63.986490 *
```

```
rsq.rpart(firsttree)
```

```
##
## Regression tree:
## rpart(formula = visitors ~ ID + air_area_name + latitude + longitude +
##     air_genre_name + avgDailyVisitsGenre + count_of_reservations +
##     avg_of_day_id, data = trainset2, method = "anova")
##
```

```
## Variables actually used in tree construction:
## [1] avg_of_day_id
##
## Root node error: 11971782/46763 = 256.01
##
## n= 46763
##
##         CP nsplit rel error  xerror     xstd
## 1 0.418889      0   1.00000 1.00004 0.063021
## 2 0.077449      1   0.58111 0.58339 0.060155
## 3 0.072795      2   0.50366 0.51617 0.059964
## 4 0.014934      3   0.43087 0.43506 0.058083
## 5 0.011915      4   0.41593 0.41974 0.056259
## 6 0.011242      5   0.40402 0.41193 0.056254
## 7 0.010000      6   0.39277 0.40617 0.056249
```





3.10.3 Validation with the test set

We created a function **rsqUPC** that will help us in the process of the validation to easily calculate the R-squared. The result return us 0.6.

3.11 Final model choose

3.11.1 Now we will consider again LM with all of the variables that we've now developed

Considering visit date time instead of reserve date time to not lose information relate to the future, cause in the reservation data set we also have information relate to the number of reservation that we have per day and how many visitors we will have from the reservations for the month of April.

```
mod6=lm( visitors ~  ID + day_of_week + holiday_flg + air_genre_name + air_area_name + count_of_reserva
summary(mod6)$r.squared
```

3.11.1.1 Creation of the model with trainset

```
## [1] 0.6436243
```

```
#We will print the rsq instead of printing the summary to conciseness of the report

predmod6train <- predict(firsttree, data=trainset2)
#rsq
rsqUPC(trainset2,predmod6train)
```

```
## [1] 0.6072254
```

In the previous chunk after building the mod 6, we took a look on the **R-squared which is 0.6436** the highest one that we reach during all the analysis.

```
#mod 6
predictTest2<-predict(mod6, newdata = testset2)
rsqUPC(testset2,predictTest2)
```

3.11.1.2 Test the new fits with rsq on the test set as the above r-squared was on the train set

```
## [1] 0.6252271
```

Here we see quite a close result to the original meaning that we can expect a similar performance in the wild if you will. As a part of the next steps we will retrain the model inclusive of the test set data as we would otherwise lose some levels otherwise based on our setup.

3.11.2 Refitting LM model with full dataset

Now that we've tested with a train test split we want to reload the best LM model with the full data set so that we don't drop levels when we predict the submission data set.

We also undergo a series of steps here to get the variables that we need into the submission data set to achieve an accurate prediction. We also ensure that we maintain the original sequence which seems entirely random.

```
submission2 <- read.csv("project_submission (1).csv")

submission2<-submission2%>%mutate(date1 = str_sub(ID,-10,-1))
submission2<-submission2%>%mutate(id = str_sub(ID,1,-13))
submission2$date1<-ymd(submission2$date1)
submission2$id <- str_trim(submission2$id)

submission2<-rename(submission2,id=ID,date=date1,ID=id)

submission2$counter <- seq(from=1,to=15770,by=1)

submission2$ID=as.factor(submission2$ID)

submission2 <- submission2 %>% left_join(restaurant_info%>%select(-X), by="ID") %>% left_join(date_info

reservations<- reservations%>%mutate(day_reservation = day(reservations$visit_datetime))
reservations<- reservations%>%mutate(month_reservation = month(reservations$visit_datetime))
```

```
reservations<- reservations%>%mutate(year_reservation = 2017)
reservations<- reservations%>%mutate(date = '')

reservations$date <- as.Date(with(reservations, paste(year_reservation, month_reservation, day_reservati

reservation_per_day <- reservations %>%
  group_by(ID,date) %>%
  summarise(Tot_visit_day_from_Reservations = sum(reserve_visitors),
            count_of_reservations = n())

reservation_per_day$ID=as.factor(reservation_per_day$ID)

mod6<- lm( visitors ~  ID + day_of_week + holiday_flg + air_genre_name + air_area_name + count_of_reserv
predictTest2<-predict(mod6, newdata = combinedDataset)
##quick rsq calculation on model
rsqUPC(combinedDataset,predictTest2)
```

```
## [1] 0.6479203
```

```
submission2 <- submission2 %>% select(-date)%>%mutate(date=ymd(calendar_date)) %>% select(-calendar_date

submission2$count_of_reservations[is.na(submission2$count_of_reservations)] <- 0
submission2$Tot_visit_day_from_Reservations[is.na(submission2$Tot_visit_day_from_Reservations)] <- 0
submission2$avg_of_day_id[is.na(submission2$avg_of_day_id)] <- 0

submission_zeroes2 <- submission2 %>% filter(ID=="restaurant_ 292"|ID=="restaurant_ 325")

submission2 <- submission2 %>% filter(ID!="restaurant_ 292"&ID!="restaurant_ 325")


submission2$visitors <- predict(mod6, newdata = submission2)
submission2$visitors <- round(submission2$visitors,0)
submission_zeroes2 <- submission_zeroes2 %>% select(id,ID,visitors,counter)
submission2 <- submission2 %>% select(id,ID,visitors,counter)

submission2 <- bind_rows(submission2,submission_zeroes2) %>%arrange(counter)%>%select(-counter)

submission2$visitors <- if_else(submission2$visitors<0,0,submission2$visitors)

submission2=submission2%>%select(-ID)
submission2 <-rename(submission2,ID=id)

submission2 %>% write_csv("project6_data_submission.csv")
```

#Conclusion

4 References

[1] R Core Team (2016). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

[2] Wickham et al., (2019). Welcome to the tidyverse. Journal of Open Source Software, 4(43), 1686, https://doi.org/10.21105/joss.01686

[3] Garrett Grolemund, Hadley Wickham (2011). Dates and Times Made Easy with lubridate. Journal of Statistical Software, 40(3), 1-25. URL https://www.jstatsoft.org/v40/i03/.

[4] Joe Cheng, Bhaskar Karambelkar and Yihui Xie (2021). leaflet: Create Interactive Web Maps with the JavaScript 'Leaflet' Library. R package version 2.0.4.1. https://CRAN.R-project.org/package=leaflet

[5] Kuhn et al., (2020). Tidymodels: a collection of packages for modeling and machine learning using tidyverse principles. https://www.tidymodels.org

[6] Terry Therneau and Beth Atkinson (2019). rpart: Recursive Partitioning and Regression Trees. R package version 4.1-15. https://CRAN.R-project.org/package=rpart

[7] Stephen Milborrow (2021). rpart.plot: Plot 'rpart' Models: An Enhanced Version of 'plot.rpart'. R package version 3.1.0. https://CRAN.R-project.org/package=rpart.plot

[8] Jarek Tuszynski (2021). caTools: Tools: Moving Window Statistics, GIF, Base64, ROC AUC, etc. R package version 1.18.2. https://CRAN.R-project.org/package=caTools

[9] Jeroen Ooms, David James, Saikat DebRoy, Hadley Wickham and Jeffrey Horner (2021). RMySQL: Database Interface and 'MySQL' Driver for R. R package version 0.10.22. https://CRAN.R-project.org/package=RMySQL

[10] Original S code by Richard A. Becker and Allan R. Wilks. R version by Ray Brownrigg. (2018). mapdata: Extra Map Databases. R package version 2.3.0. https://CRAN.R-project.org/package=mapdata