“Restaurant Visitor Forecasting”

Amr Sarhan, Babak Barghi, Daniel Zöttl

December 20, 2020

Contents

The analysis is carried out in *R 4.0.2*[1] and the packages are used.

```
Final <- read.csv("Final.csv",sep = ",",header = TRUE,dec = ".")
```

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, we 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.

2 Explorative Data Analysis

We decided in this section to explore and visualize the data we have to provide more insights to the restaurants. Our approach is based on certain variables and their correlations, that we believe present valuable insights, which restaurants can use to enhance their dining experience and optimize their operations. In addition to that, we can offer our analysis to investors in the f&b market in Japan.

2.1 Our first pillar - Days

Which days are the busiest and which are the least busy?

```
Final2 <- na.omit(Final)
Final.days <- group_by(Final2, day_of_week) %>%
  summarise(TOTAL.visitors=sum(visitors))
ggplot(Final.days, aes(x=(day_of_week=fct_relevel(day_of_week, "Monday","Tuesday","Wednesday",
"Thursday","Friday")), y=TOTAL.visitors)) +
  geom_bar(stat = "identity",color="black",fill="grey") +
labs(title="Total visitors per weekday", x="Weekday",
y="Total visitors") +
  theme_bw()
```

Friday and the weekend appear to be the most popular days; which is to be expected. Monday and Tuesday have the lowest numbers of average visitors.This finding provides restaurants with the expected number of visitors for each day in the week. Moreover, it will help the restaurants to prepare their supply of ingredients and schedule enough staff in accordance with days from busiest to emptiest.

Does holiday have more visitors than normal days?

```
Final2 <- na.omit(Final)
keymeans <- select(Final2,holiday_flg, visitors)
keymeansObj.1 <- kmeans(keymeans[,1:2],centers = 2,nstart = 25)
names(keymeansObj.1)
```

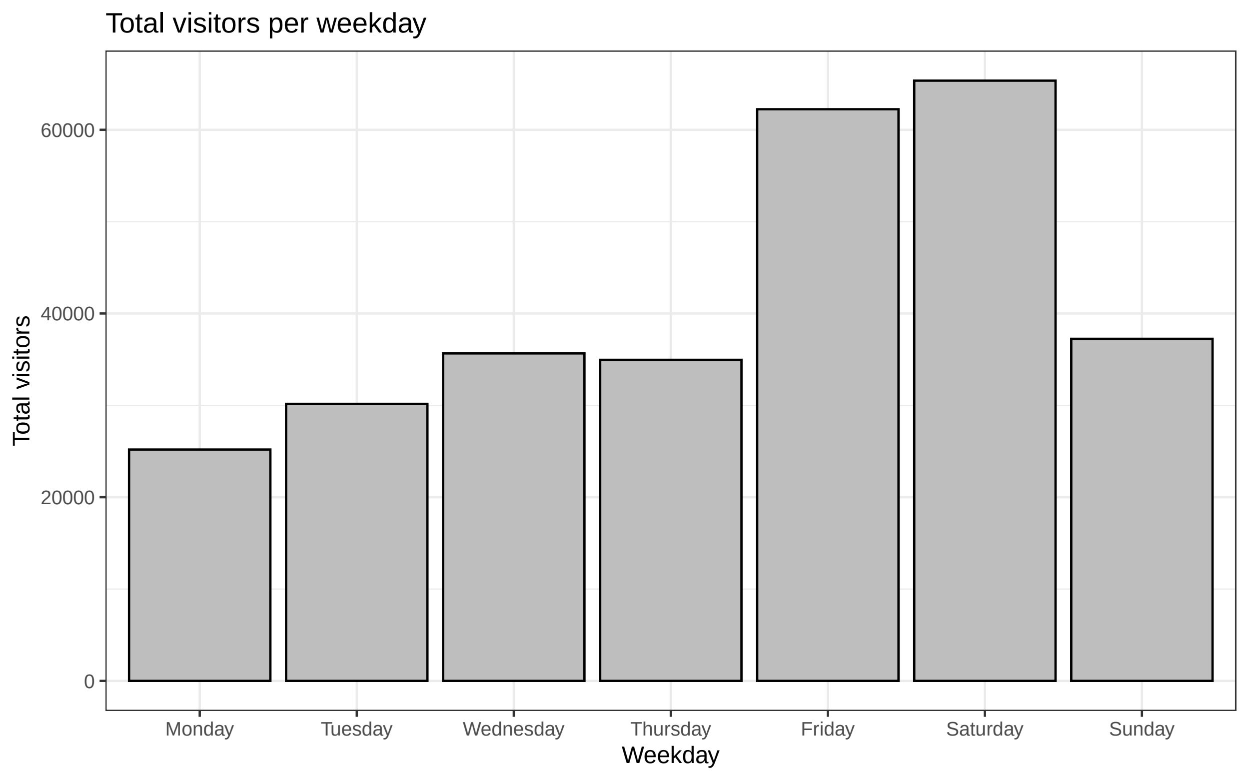


Figure 1: Weekdays correlation with visitors

```
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

```r
Holiday <- c("Holiday","Normal")
NumberHol <- c(0,1)
Holiday.Name <- data.frame(Holiday,NumberHol)
Holiday.Name
```

```r
plot(x=Final2$holiday_flg, y=Final2$reserve_visitors, main="Holidays VS vistors",
xlab="Holiday [Holiday]",ylab="Total visitors [visitors]",
col=keymeansObj.1$cluster,pch=19,cex=1) +
  points(keymeansObj.1$centers, col=1:3, pch=7, cex=4, lwd=5)
```

```
## integer(0)
```

The clustering of visitors shows a tendency of more visitors to the restaurants on holidays than normal days. this finding is logical and we think it's a common knowledge. However, it still serves as a confirmation to this assumption.

2.2 Second pillar- Genre

We asked ourselves the question, **which genres have the most visitors**.

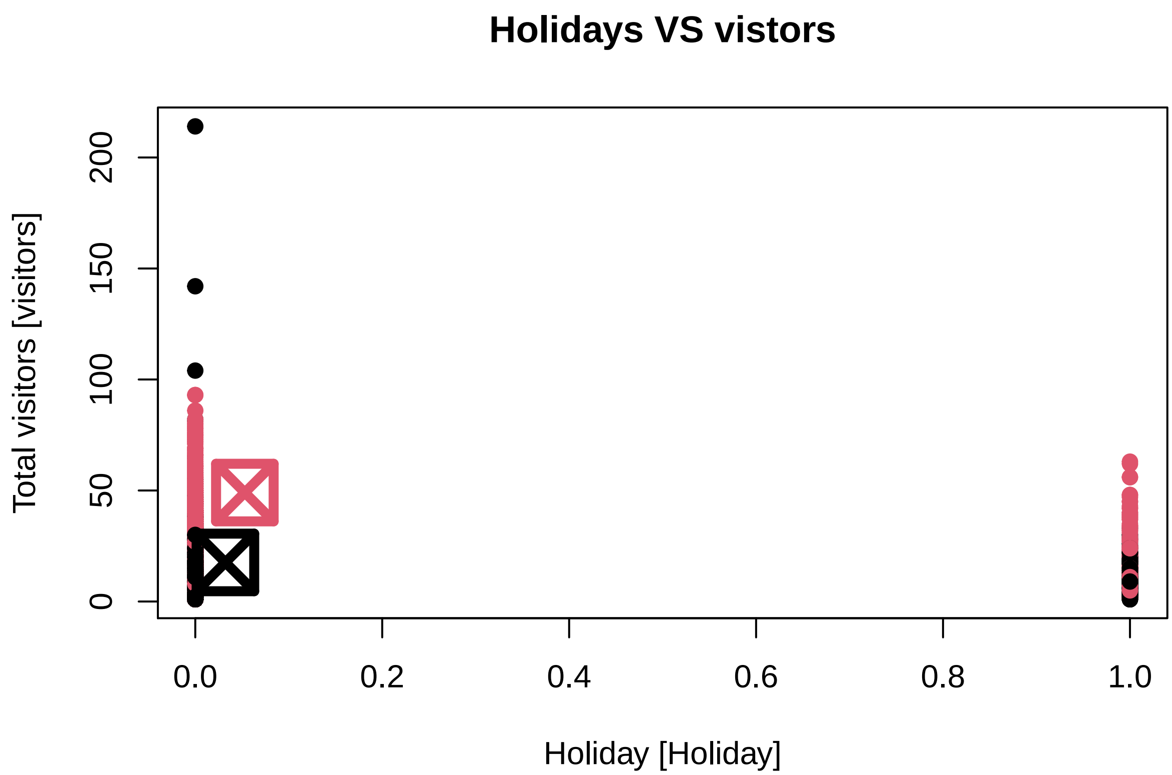


Figure 2: Holidays and normal days clustering with visitors

Certain genres have more visitors than the others. this information is helpful, it can be used in consulting future investors in the same market. moreover, it can be helpful to existing owners, to know the expected trend of their genre and accordingly they can decide how much further they will invest or save.

```
Finalgenre <- select(Final,air_genre_name,visitors)
Finalgenre1 <- arrange(Finalgenre,desc(visitors))
Finalgenre2 <- head(Finalgenre1,10)
Finalgenre3 <- filter(Finalgenre2)
Finalgenre3
```

```
##    air_genre_name visitors
## 1    Bar/Cocktail      877
## 2      Dining bar      348
## 3         Izakaya      335
## 4         Izakaya      262
## 5     Cafe/Sweets      261
## 6     Cafe/Sweets      235
## 7   Karaoke/Party      216
## 8           Other      211
## 9  Italian/French      199
## 10    Cafe/Sweets      181
```

```
p1 <-ggplot(Finalgenre3,aes(x=reorder(as.factor(air_genre_name),
                                      -visitors), y=visitors))+
geom_bar(stat = "identity",color="black",fill="grey") +
  geom_hline(data = Finalgenre3,aes(yintercept=mean(visitors)),
             linetype="dashed",size=1, color="red")+
labs(title="Genre-visitors Correlation",
x="Genre [names]",y="visitors[visitors]")
```

```
FinalID <- select(Final2,ID,visitors)
FinalID1 <- arrange(FinalID,desc(visitors))
FinalID2 <- head(FinalID1,10)
FinalID3 <- filter(FinalID2)
FinalID3
```

```
##                 ID visitors
## 1  restaurant_ 828      216
## 2   restaurant_ 44      199
## 3  restaurant_ 619      166
## 4  restaurant_ 125      164
## 5  restaurant_ 619      152
## 6  restaurant_ 504      145
## 7  restaurant_ 619      144
## 8  restaurant_ 619      140
## 9  restaurant_ 619      139
## 10 restaurant_ 504      137
```

```
p2 <-ggplot(FinalID3,aes(x=reorder(as.factor(ID), -visitors), y=visitors))+
geom_bar(stat = "identity",color="black",fill="grey") +
  geom_hline(data = FinalID3,aes(yintercept=mean(visitors)),
             linetype="dashed",size=1, color="red")+
```

```
  labs(title="ID-visitors Correlation",
x="ID [numeber]",y="visitors[visitors]")

grid.arrange(p1,p2)
```

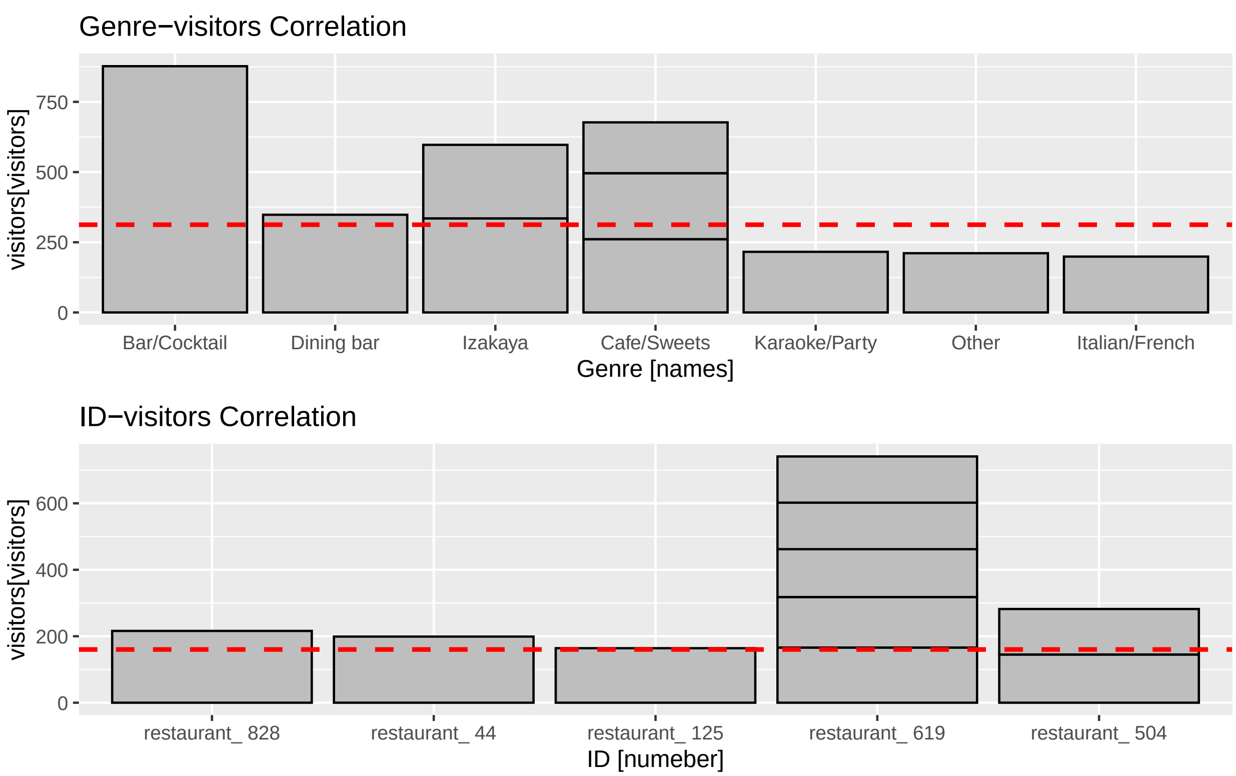


Figure 3: ID correlation with visitors numbers

The figures show the genres and restaurant Ids with most visitors.

```
CafeSweets <- Final[which(Final$air_genre_name=="Cafe/Sweets"), ]
Asian <- Final[which(Final$air_genre_name =="Asian"), ]
BarCocktail <- Final[which(Final$air_genre_name =="Bar/Cocktail"),]
Creativecuisine <- Final[which(Final$air_genre_name =="Creative cuisine"),]
Diningbar <- Final[which(Final$air_genre_name =="Dining bar"),]
Internationalcuisine <- Final[which(Final$air_genre_name =="International cuisine"),]
ItalianFrench <- Final[which(Final$air_genre_name =="Italian/French"),]
Izakaya <- Final [which(Final$air_genre_name =="Izakaya"),]
Japanesefood <- Final[which(Final$air_genre_name =="Japanese food"),]
KaraokeParty <- Final[which(Final$air_genre_name =="Karaoke/Party"),]
Westrenfood <- Final[which(Final$air_genre_name =="Western food"),]
Korean <- Final[which(Final$air_genre_name =="Yakiniuku/Korean food"),]
Oko <- Final[which(Final$air_genre_name =="Okonomiyaki/Monja/Teppanyaki"),]
labels <- c("Cafe/Sweets","Asian","Bar/Cocktail",
            "Creative cuisine","Dining bar","International cuisine",
            "Italian/French","Izakaya","Japanese food","Karaoke/Party",
            "Western food","Yakiniuku/Korean food","Okonomiyaki/Monja/Teppanyaki")
Genre<- c(nrow(CafeSweets)/nrow(Final),nrow(Asian)/nrow(Final),
```

```
          nrow(BarCocktail)/nrow(Final),nrow(Creativecuisine)/nrow(Final),
          nrow(Diningbar)/nrow(Final),nrow(Internationalcuisine)/nrow(Final),
          nrow(ItalianFrench)/nrow(Final),nrow(Izakaya)/nrow(Final),
          nrow(Japanesefood)/nrow(Final),nrow(KaraokeParty)/nrow(Final),
          nrow(Westrenfood)/nrow(Final),nrow(Korean)/nrow(Final),
          nrow(Oko)/nrow(Final))
pct <- round(Genre*100)
labels2 <- paste(labels,pct)
labels3 <- paste(labels2,"%",sep = " ")
```

```
pie(Genre,labels = labels3,col = c("gray 10","gray 20","gray 30","gray 40",
                                   "gray 50","gray 60","gray 70","gray 80",
                                   "gray 90","gray 95","gray 100"),cex=0.7)
```

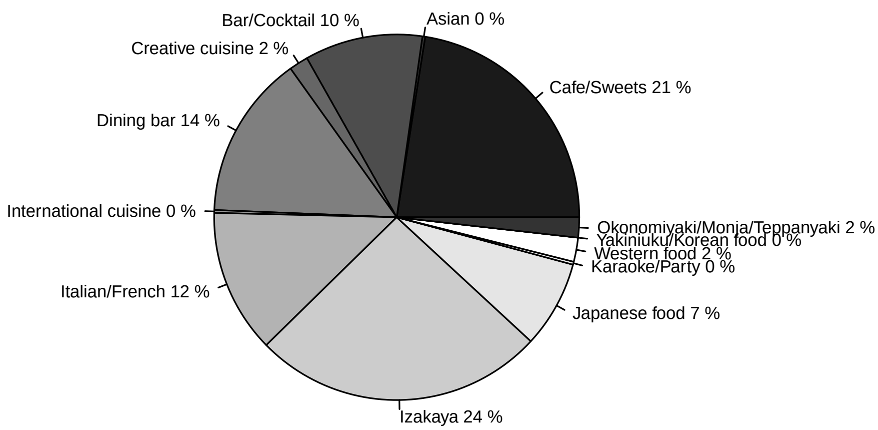


Figure 4: Genre pie chart

The distribution of visitors to each genre and its percentage of total data set is expressed in this pie chart. It shows each genre's share and further help owners and future investors to either invest in a certain genre or reduce future investments.

2.3 Third pillar - Reserve visitors

Our approach here came from the following question.

How often reservations are actually fulfilled to visits

The result will give the restaurants a good index of actual visitors. Restaurants sometimes over-buy ingredients and schedule more people because certain number of reservations are expected. knowing the arithmetic means of reservations vs actual visits give them a better chance to take a more appropriate decision when it comes to prepare for expected reservations.

```
reservevisitors <- select(Final2,reserve_visitors,ID)
reservevisitors.1 <- group_by(reservevisitors,ID)
reservevisitors.2 <- summarise(reservevisitors.1,reserve=sum(reserve_visitors))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
reservevisitors.3 <- Final2[which(Final2$holiday_flg ==0),]
reservevisitors.4 <- select(reservevisitors.3,reserve_visitors,ID)
reservevisitors.5 <- group_by(reservevisitors.4,ID)
reservevisitors.6 <- summarise(reservevisitors.5,reserve=sum(reserve_visitors))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
reservevisitors.7 <- Final2[which(Final2$holiday_flg ==1),]
reservevisitors.8 <- select(reservevisitors.7,reserve_visitors,ID)
reservevisitors.9 <- group_by(reservevisitors.8,ID)
reservevisitors.10 <- summarise(reservevisitors.9,reserve=sum(reserve_visitors))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
visitors <- select(Final2,visitors,ID)
visitors.1 <- group_by(visitors,ID)
visitors.2 <- summarise(visitors.1,totvisitors=sum(visitors))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
visitors.3 <- Final2[which(Final2$holiday_flg == 0),]
visitors.4 <- select(visitors.3,visitors,ID)
visitors.5 <- group_by(visitors.4,ID)
visitors.6 <- summarise(visitors.5,totvisitors=sum(visitors))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
visitors.7 <- Final2[which(Final2$holiday_flg == 1),]
visitors.8 <- select(visitors.7,visitors,ID)
visitors.9 <- group_by(visitors.8,ID)
visitors.10 <- summarise(visitors.9,totvisitors=sum(visitors))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
IdealvsActual <- c(nrow(reservevisitors.2),nrow(visitors.2))
IdealvsActual.holiday <- c(nrow(reservevisitors.6),nrow(visitors.6))
IdealvsActual.normal <- c(nrow(reservevisitors.10),nrow(visitors.10))
```

```r
par(mfrow= c(1,2))

barplot( IdealvsActual.holiday ,main = "reservers vs Actual visits in holidays",
col = c("gray 40","gray60"),names.arg = c("reserves","Actual visits"),
ylab = "Instances in numbers",ylim = c(0,300))

barplot( IdealvsActual.normal ,main = "reservers vs Actual visits in normal days",
col = c("gray 40","gray60"),names.arg = c("reserves","Actual visits"),
ylab = "Instances in numbers",ylim = c(0,300))
```

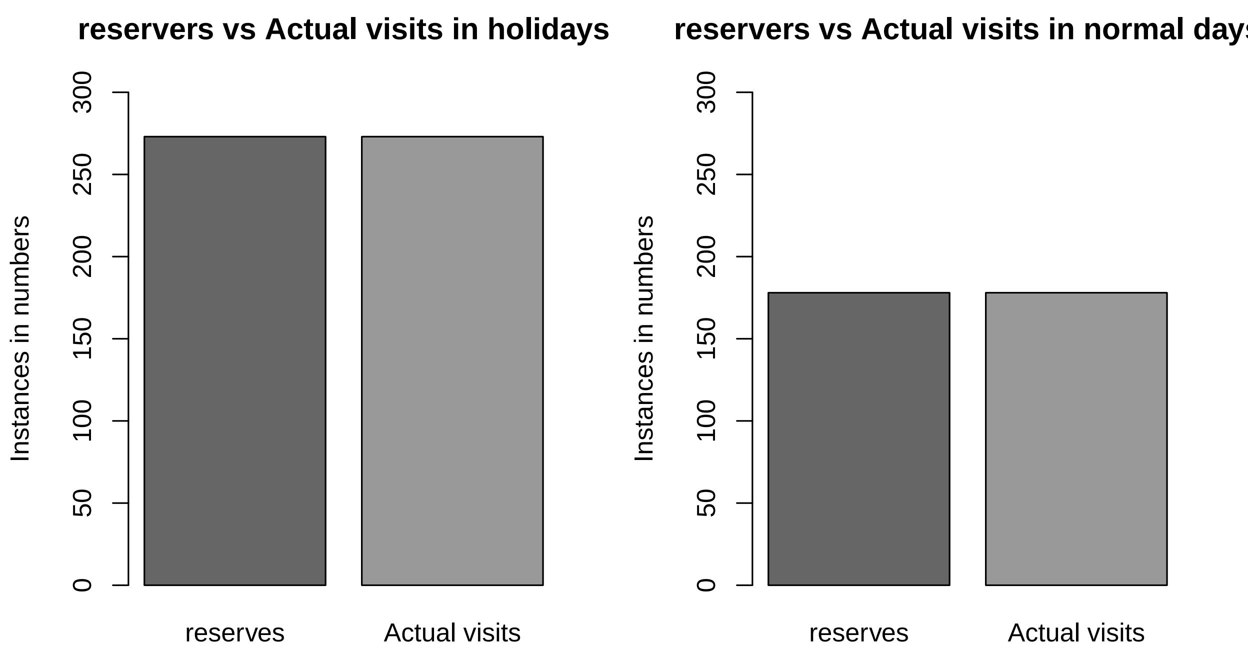


Figure 5: Reservers vs Actual visits Instances in holidays and Normal days per ID

The figures above show same number of instances of visits and reservations in the data set. however these graohs are misleading since not all reservations are actually fulfilled.

Reservations vs Visits

```r
air_reserve <- read_csv("air_reserve.csv")
air_visit <- read_csv("air_visit.csv")
p <- air_reserve %>%
  mutate(visit_date = date(visit_datetime)) %>%
  group_by(ID,visit_date) %>%
  summarise(reserve_visitors = sum(reserve_visitors))

all_reserve <- air_visit %>%
  inner_join(p, by = c("ID", "visit_date"))

p1 <- all_reserve %>%
```

```r
  filter(reserve_visitors < 120) %>% #to remove outliers
  ggplot(aes(reserve_visitors, visitors)) +
  geom_point(color = "black", alpha = 0.5) +
  geom_abline(slope = 1, intercept = 0, color = "blue") +
  geom_smooth(method = "lm", color = "red") +
  theme_bw()
p1
```

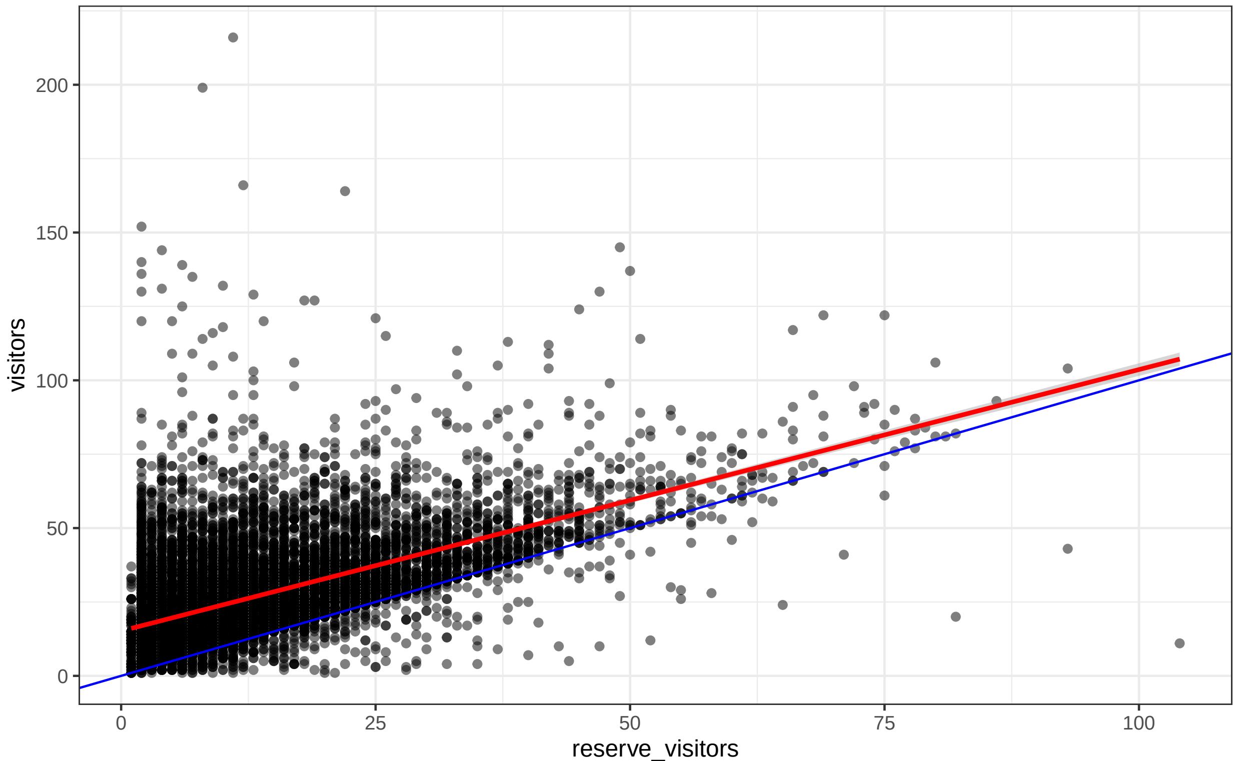


Figure 6: Reserve visitors vs Actual visitors ID

The blue line shows $reserve\_visitors = visitors$ and the red line is a linear fit.

The scatter points fall largely above the line of identity, indicating that there were more visitors that day than had reserved a table. This is not surprising, since a certain number of people will always be walk-in customers.

A notable fraction of the points is below the line, which probably indicates that some people made a reservation but changed their mind and didn't go. That kind of effect is probably to be expected and taking it into account will be one of the challenges in this competition.

The linear fit suggests a trend in which larger numbers of reserve_visitors are more likely to underestimate the eventual visitor numbers. This is not surprising either, since we can imagine that it is more likely that (a) a large reservation is cancelled than (b) a large group of people walk in a restaurant without reservation.

Are reservations more likely to be completed if it's a holiday than in a normal day?

```r
par(mfrow=c(3,2))

#holidays
```

```
boxplot(reservevisitors.6$reserve,xlab= "reserve visitors (holidays)",
        ylab="count [Numbers]",ylim=c(0,2000))
boxplot(visitors.6$totvisitors,xlab= "Actual visitors (holidays)",
        ylab="count [Numbers]",ylim=c(0,2000))

#normal days
boxplot(reservevisitors.10$reserve,xlab= "reserve visitors (normal days)",
        ylab="count [Numbers]",ylim=c(0,100))
boxplot(visitors.10$totvisitors,xlab= "Actual visitors (normal days)",
        ylab="count [Numbers]",ylim=c(0,100))
```

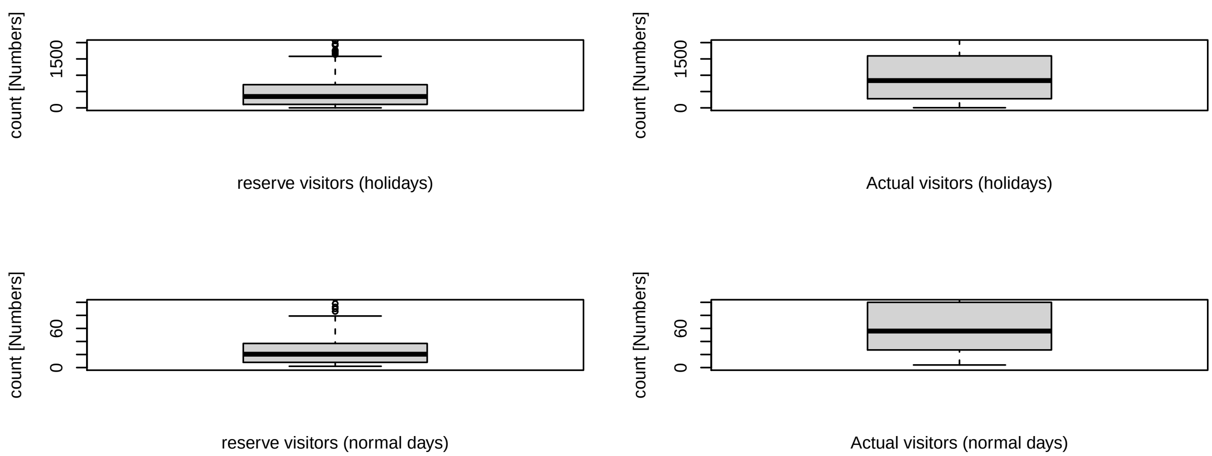


Figure 7: Reservers vs Actual visits mean in holidays and Normal days per ID

The figures above show the means of reservations and actual visitors in holidays and normal days. It doessn't suggest any strong correlation.

2.4 fourth pillar - Geographical distribution

Does a certain place have more visitors than the others?

The answer to that question might be more helpful to future investors than existing owners. Nonetheless, it still provides a valuable correlation, which helps to understand the cap of visitors in your area and helps you to take a better decision in case of expansion or even moving from yur area to another.

```
ggplot(Final, aes(longitude,latitude)) +
geom_point(aes(colour = visitors))+ scale_colour_gradient(low = "darkblue", high = "lightblue")+
```

```
labs(title="visitors considering LAT and LON") +
  theme_bw()
```

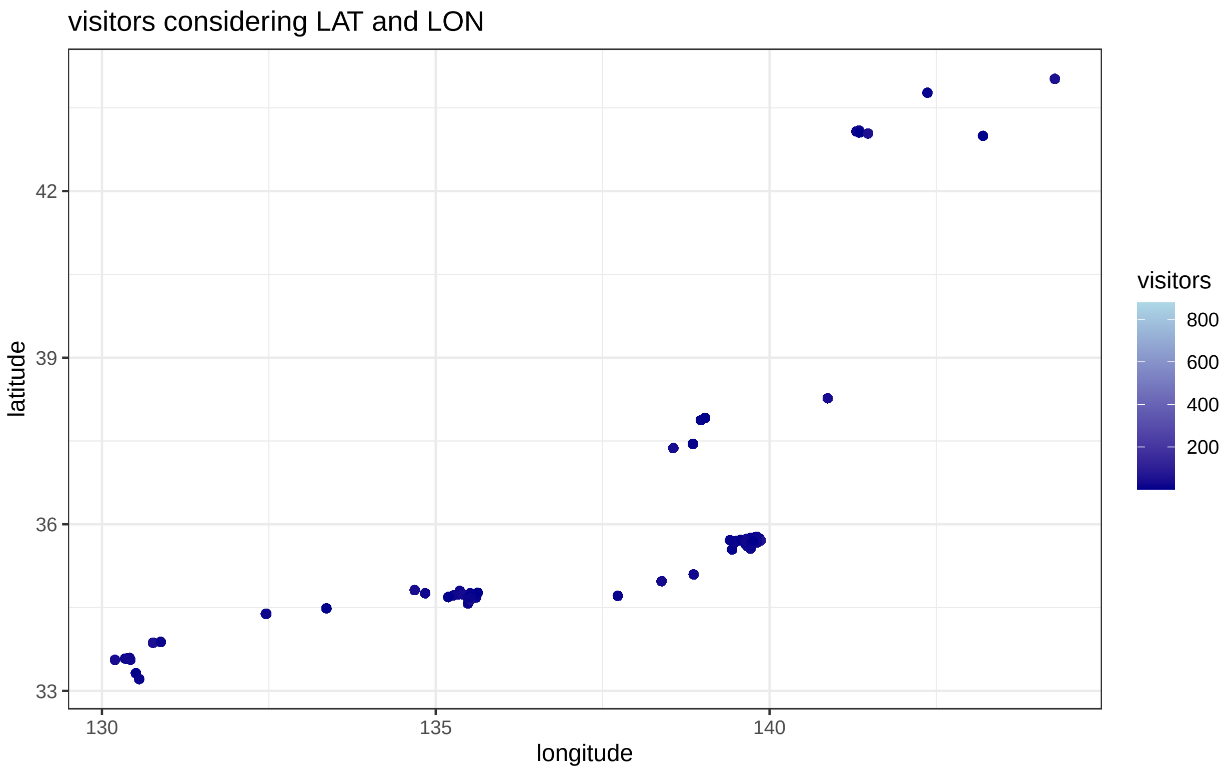


Figure 8: Visits Distribution on coordinates

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

The previous figures show the distribution of visitors according to coordinates and represent them visually on the real map of the country. this illustration is helpful in order to take better decision. for instance, to move your store from an area to another or in case of expansion and opening another branch.

3 Analysis

3.1 Data preparing

Before the models for the prediction can be created, the data must be correctly extracted from the mySQL database. For this purpose the "RMariaDB" package is used, with which it is possible to implement queries in RStudio. First the calendar information and the submission file that has to be submitted at the end are created.

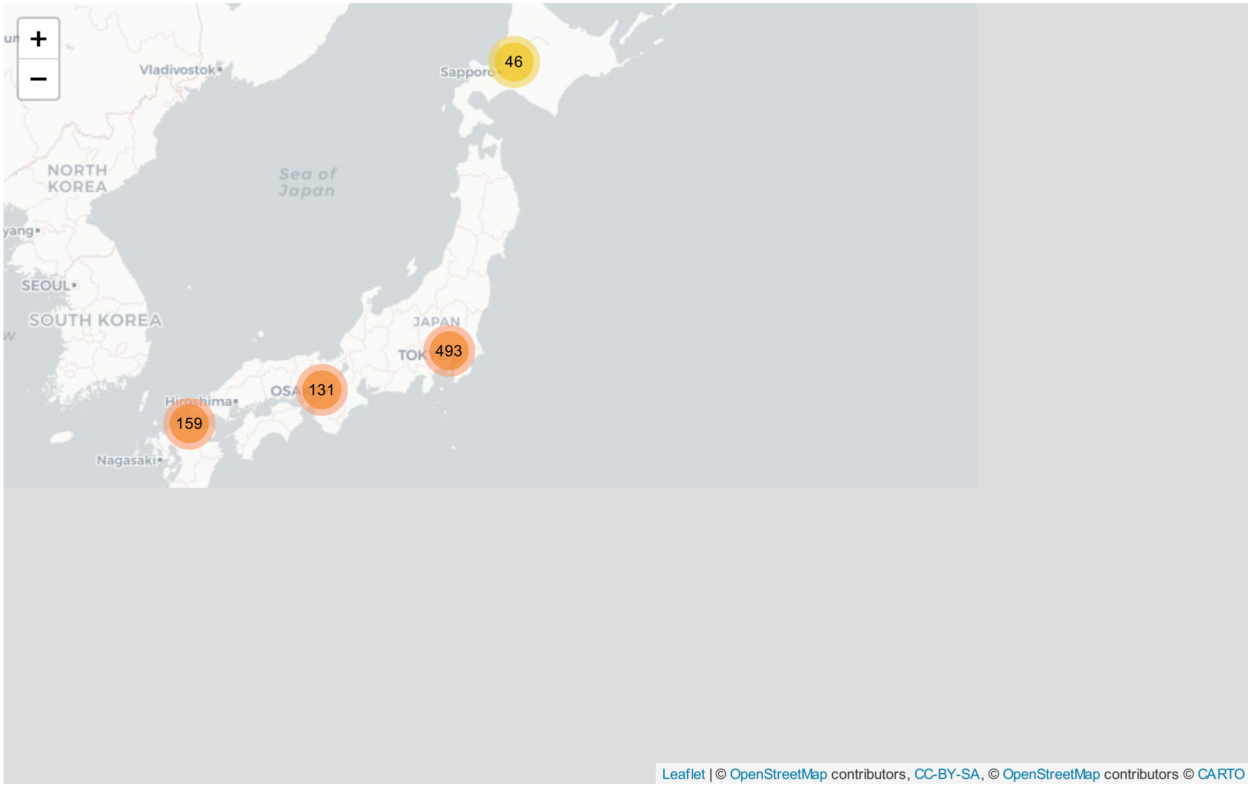


Figure 9: Visits Distribution on map of japan

```
# create date info table
date_info <- read.csv("date_info.csv")
date_info$calendar_date <- as.Date(date_info$calendar_date)

# Import Submission file
submission <- read.csv("project_submission.csv", sep=",", header=TRUE)

# Separate ID
submission <- submission %>%
  tidyr::separate(ID, sep=" _" ,
                c("ID", "visit_date"))

# Change data style in submission
submission$visit_date <- as.Date(submission$visit_date)
submission$visitors <- NA
```

After that, a table is created that contains all relevant data and variables of the restaurants from January to March. In addition, all reservations for the respective days are summed up and connected to the table.

```
# Send the query to open the table

dbSendQuery(con, "DROP TEMPORARY TABLE IF EXISTS myData;")
```

```
## <MariaDBResult>
##   SQL  DROP TEMPORARY TABLE IF EXISTS myData;
```

```
##   ROWS Fetched: 0 [complete]
##        Changed: 0
```

```r
dbSendQuery(con, "
CREATE TEMPORARY TABLE myData
SELECT V.ID, V.visit_date, V.visitors,
R.air_area_name, R.air_genre_name, R.latitude, R.longitude,
 D.calendar_date, D.day_of_week, D.holiday_flg
FROM air_visit V JOIN restaurant_info R USING (ID)
LEFT OUTER JOIN date_info D on V.visit_date = D.calendar_date
LEFT OUTER JOIN air_reserve Re on V.ID = Re.ID AND V.visit_date = Re.visit_datetime
GROUP BY ID, visit_date;")
```

```
## <MariaDBResult>
##   SQL
## CREATE TEMPORARY TABLE myData
## SELECT V.ID, V.visit_date, V.visitors,
## R.air_area_name, R.air_genre_name, R.latitude, R.longitude,
##  D.calendar_date, D.day_of_week, D.holiday_flg
## FROM air_visit V JOIN restaurant_info R USING (ID)
## LEFT OUTER JOIN date_info D on V.visit_date = D.calendar_date
## LEFT OUTER JOIN air_reserve Re on V.ID = Re.ID AND V.visit_date = Re.visit_datetime
## GROUP BY ID, visit_date;
##   ROWS Fetched: 0 [complete]
##        Changed: 61803
```

```r
# Send query
res <- dbSendQuery(con, "SELECT * FROM myData")
# Create database
myData <- dbFetch(res, n = -1)

# Create sum of reserve_visitors

Reserve_Visitors <- dbSendQuery(con, "SELECT RI.ID, AR.visit_datetime, AR.reserve_visitors
FROM restaurant_info RI, air_reserve AR
WHERE RI.ID = AR.ID
ORDER BY RI.ID;")
resvisitors <- dbFetch(Reserve_Visitors, n=-1)
str(resvisitors)
```

```
## 'data.frame':    36259 obs. of  3 variables:
##  $ ID              : chr  "restaurant_ 1" "restaurant_ 1" "restaurant_ 1" "restaurant_ 1" ...
##  $ visit_datetime  : Date, format: "2017-04-21" "2017-04-16" ...
##  $ reserve_visitors: int  8 12 3 2 4 3 2 52 2 5 ...
```

```r
resvisitors$visit_date <- as.Date(resvisitors$visit_datetime,"%Y-%m-%d")

SumResVisitors <- resvisitors %>% group_by(ID,visit_date) %>%
  summarise(reserve_visitors=sum(reserve_visitors))
```

```
## `summarise()` regrouping output by 'ID' (override with `.groups` argument)
```

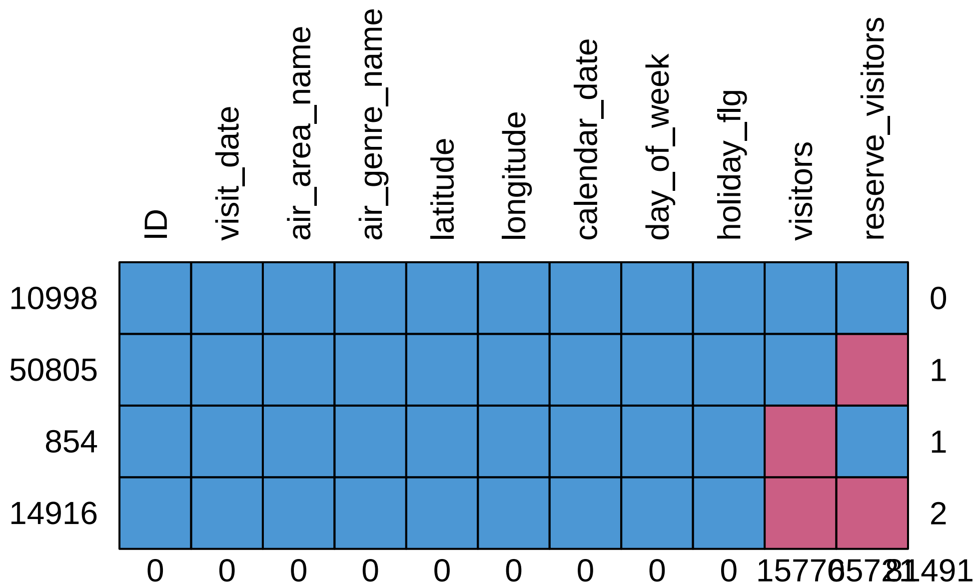
```
# Merge myData and SumResVisitors and name it Final

myData$visit_date <- as.Date(myData$visit_date)
Final <- merge(myData, SumResVisitors,by=c("ID", "visit_date"), all.x = TRUE, sort = FALSE)
```

Finally, all necessary variables are assigned to the Submission file and the table is linked to Final. This way we get a complete table containing all relevant variables from January to April.

```
# Merge Submission with some variables of Final file

# create a database for restaurant info

res <- dbSendQuery(con, "SELECT * FROM restaurant_info")
restaurant_info <- dbFetch(res, n=-1)

# merge the  submission_2 file with restaurant info

submission_2 <- merge(submission, restaurant_info,
                      by ="ID", all.x =TRUE, sort =F)

# merge submission file with sum of reservation

submission_3 <- merge(submission_2, SumResVisitors,
                      by=c("ID","visit_date"), all.x = TRUE, sort = FALSE)

# merge submission file with date info

date_info$visit_date <- as.Date(date_info$calendar_date)
date_info$calendar_date <- as.Date(date_info$calendar_date)

submission_4 <- merge(submission_3,date_info, by=c("visit_date"),
                      all.x = TRUE, sort= FALSE)

# merge Final with submission_4 to get a full table from January to April

Final$calendar_date <- as.Date(Final$calendar_date)
Final_table <- rbind(Final, submission_4)
str(Final_table)
```

```
## 'data.frame':    77573 obs. of  11 variables:
##  $ ID              : chr  "restaurant_ 319" "restaurant_ 319" "restaurant_ 319" "restaurant_ 623" ..
##  $ visit_date      : Date, format: "2017-01-06" "2017-01-07" ...
##  $ visitors        : int  13 37 40 21 41 27 29 19 27 40 ...
##  $ air_area_name   : chr  "Shizuoka-ken Hamamatsu-shi Motoshirocho" "Shizuoka-ken Hamamatsu-shi Motos
##  $ air_genre_name  : chr  "Cafe/Sweets" "Cafe/Sweets" "Cafe/Sweets" "Okonomiyaki/Monja/Teppanyaki" .
##  $ latitude        : num  34.7 34.7 34.7 34.4 34.4 ...
##  $ longitude       : num  138 138 138 132 132 ...
##  $ calendar_date   : Date, format: "2017-01-06" "2017-01-07" ...
##  $ day_of_week     : chr  "Friday" "Saturday" "Sunday" "Friday" ...
##  $ holiday_flg     : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ reserve_visitors: int  2 12 5 2 8 16 4 4 13 8 ...
```

```r
md.pattern(Final_table,rotate.names = TRUE)
```




```
##       ID visit_date air_area_name air_genre_name latitude longitude
## 10998  1          1             1              1        1         1
## 50805  1          1             1              1        1         1
## 854    1          1             1              1        1         1
## 14916  1          1             1              1        1         1
##        0          0             0              0        0         0
##       calendar_date day_of_week holiday_flg visitors reserve_visitors
## 10998             1           1           1        1                1     0
## 50805             1           1           1        1                0     1
## 854               1           1           1        0                1     1
## 14916             1           1           1        0                0     2
##                   0           0           0    15770            65721 81491
```

Finally, we analyze how many missing values the table contains. As we can see about 650000 entries for the reservations are missing. These values must either be imputed or replaced by 0 to use this variable. However, since there are 65000 missing values, imputation makes little sense for us. Therefore we replace the NA values by 0.

Moreover, to get a better model, in the next section we adapt the data and create new variables.

3.2 Data Modeling

First, the days of the week are converted into numbers to be able to work better with the model. In addition, the days of the weekend are assigned a variable 1, all others a 0.

```
# Data Modeling

#Day of the week
Final_table1 <- Final_table %>%
  mutate(day_week = case_when(day_of_week == "Saturday"  ~ "6",
                              day_of_week == "Sunday"  ~ "7",
                              day_of_week == "Monday"  ~ "1",
                              day_of_week == "Tuesday"  ~ "2",
                              day_of_week == "Wednesday"  ~ "3",
                              day_of_week == "Thursday"  ~ "4",
                              day_of_week == "Friday"  ~ "5"))
#Weekend or not
Final_table1 <- Final_table1 %>%
  mutate(is_weekend = case_when(day_of_week == "Saturday"  ~ "1",
                              day_of_week == "Sunday"  ~ "1",
                              day_of_week == "Monday"  ~ "0",
                              day_of_week == "Tuesday"  ~ "0",
                              day_of_week == "Wednesday"  ~ "0",
                              day_of_week == "Thursday"  ~ "0",
                              day_of_week == "Friday"  ~ "1"))
```

Furthermore, it is also interesting for us whether the day before or after the restaurant visit is a holiday or not. Therefore, these variables are also created.

```
#Previous and next day are holiday or not

holiday <- Final_table1 %>%
  select(ID, visit_date) %>%
  mutate(dayplus = visit_date + 1)

date_info <- date_info[,1:3]

holiday <- holiday %>%
  left_join(date_info, by = c("dayplus" = "calendar_date")) %>%
  mutate(next_holiday = holiday_flg) %>%
  select(-holiday_flg, - day_of_week, - dayplus)

Final_table1 <- left_join(holiday,Final_table1,
                          by = c("visit_date" = "visit_date", "ID" = "ID"))

holiday <- Final_table1 %>%
  select(ID, visit_date) %>%
  mutate(dayminus = visit_date - 1)
holiday <- holiday %>%
  left_join(date_info, by = c("dayminus" = "calendar_date")) %>%
  mutate(prev_holiday = holiday_flg) %>%
  select(-holiday_flg, - day_of_week, - dayminus)
Final_table1 <- left_join(holiday,Final_table1,
                          by = c("visit_date" = "visit_date", "ID" = "ID"))
```

Finally, the mean number of visitors for each row is added. For this, a for loop is used that matches the following variables for each row:

- ID
- day_week
- is_weekend
- holiday_flg
- prev_holiday
- next_holiday

After that, all entries of the table are filtered out whose variables match those of row [i] and the mean value of visiors is calculated. In this way, we want to calculate the mean value as specifically as possible.

```
# mean visits by ID


for (i in 1:nrow(Final_table1))
          {
            mean_vis <- filter(Final_table1,Final_table1$ID == Final_table1$ID[i] &
                                 Final_table1$is_weekend == Final_table1$is_weekend[i] &
                                 Final_table1$holiday_flg == Final_table1$holiday_flg[i] &
                                 Final_table1$prev_holiday == Final_table1$prev_holiday[i] &
                                 Final_table1$next_holiday == Final_table1$next_holiday[i] &
                                 Final_table1$day_week == Final_table1$day_week[i])

            Final_table1$Mean_Visitors[i] <- mean(mean_vis$visitors,na.rm = TRUE)
}
```

Finally, we get a table with 77573 rows and 18 variables that can be considered for the model. Before that, however, the remaining NAs have to be imputed and, as already mentioned, the NAs for reservations have to be replaced by 0.

```
# In order to avoid running everytime through the for loop, adaptions are made on a new table
Final_table2 <- Final_table1

# Substitute reserve_visitors NA by 0
Final_table2$reserve_visitors[is.na(Final_table2$reserve_visitors)] <- 0

# Substitute NAs of Mean_visitors through Imputation
imp <- mice(Final_table2,c("","","","","","","","","","","","","","","","","pmm"),
            m = 1, seed=123)
```

```
##
##  iter imp variable
##   1   1  Mean_Visitors
##   2   1  Mean_Visitors
##   3   1  Mean_Visitors
##   4   1  Mean_Visitors
##   5   1  Mean_Visitors
```

```r
Final_table2 <- complete(imp)
```

3.3 Linear Regression Model

Now different models can be created. However, since we have already filtered by ID when calculating the mean value and this already includes variables such as holiday_flg, week_day, prev_holiday, next_holiday,is_weekend,longitude, latitude, area and genre, these are no longer taken into account.

For the trainset all entries of the months January and February are used. For the testset those from March.

```r
# Create train and testset for model
Trainset = subset(Final_table2, month(visit_date) %in% c("1","2"))
Testset = subset(Final_table2, month(visit_date) %in% c("3"))
```

The following models were created:

```r
# Create models for prediction

Predictmodel_vis1 <- lm(visitors ~ Mean_Visitors, data=Trainset)
summary(Predictmodel_vis1)
```

```
##
## Call:
## lm(formula = visitors ~ Mean_Visitors, data = Trainset)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -115.558   -3.833    0.040    3.246  297.190
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)   0.246885   0.074415   3.318 0.000909 ***
## Mean_Visitors 0.952219   0.002995 317.944  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 8.188 on 39509 degrees of freedom
## Multiple R-squared:  0.719,  Adjusted R-squared:  0.719
## F-statistic: 1.011e+05 on 1 and 39509 DF,  p-value: < 2.2e-16
```

```r
Predictmodel_vis2 <- lm(visitors ~ Mean_Visitors + reserve_visitors,
                        data=Trainset)
summary(Predictmodel_vis2)
```

```
##
## Call:
## lm(formula = visitors ~ Mean_Visitors + reserve_visitors, data = Trainset)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -113.294   -3.799   -0.044    3.308  298.327
```

```
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)     0.297824   0.073572   4.048 5.17e-05 ***
## Mean_Visitors   0.929853   0.003050 304.920  < 2e-16 ***
## reserve_visitors 0.204276  0.006689  30.537  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 8.093 on 39508 degrees of freedom
## Multiple R-squared:  0.7255, Adjusted R-squared:  0.7255
## F-statistic: 5.22e+04 on 2 and 39508 DF,  p-value: < 2.2e-16
```

```
Predictmodel_vis3 <- lm(visitors ~ Mean_Visitors + reserve_visitors +
                          calendar_date, data=Trainset)
summary(Predictmodel_vis3)
```

```
##
## Call:
## lm(formula = visitors ~ Mean_Visitors + reserve_visitors + calendar_date,
##     data = Trainset)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -112.562   -3.808   -0.093    3.337  298.631
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)     -7.930e+02  4.301e+01  -18.44   <2e-16 ***
## Mean_Visitors    9.321e-01  3.039e-03  306.72   <2e-16 ***
## reserve_visitors  1.977e-01  6.671e-03   29.64   <2e-16 ***
## calendar_date    4.613e-02  2.501e-03   18.45   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 8.058 on 39507 degrees of freedom
## Multiple R-squared:  0.7278, Adjusted R-squared:  0.7278
## F-statistic: 3.521e+04 on 3 and 39507 DF,  p-value: < 2.2e-16
```

```
# Create Testset for prediction

PredictTest_vis1 <- predict(Predictmodel_vis1, newdata=Testset)

SSE <- sum((Testset$visitors-PredictTest_vis1)^2)
SST <- sum((Testset$visitors-mean(PredictTest_vis1))^2)
R2_1<- 1- SSE/SST
R2_1
```

```
## [1] 0.6517834
```

```
PredictTest_vis2 <- predict(Predictmodel_vis2, newdata=Testset)
```

```
SSE <- sum((Testset$visitors-PredictTest_vis2)^2)
SST <- sum((Testset$visitors-mean(PredictTest_vis2))^2)
R2_2<- 1- SSE/SST
R2_2
```

```
## [1] 0.6607563
```

```
PredictTest_vis3 <- predict(Predictmodel_vis3, newdata=Testset)

SSE <- sum((Testset$visitors-PredictTest_vis3)^2)
SST <- sum((Testset$visitors-mean(PredictTest_vis3))^2)
R2_3<- 1- SSE/SST
R2_3
```

```
## [1] 0.6698731
```

It can be seen that Model 3 has the best R2 value, so this model is used for prediction.

3.4 Prediciton

For the final prediction, the trainset contains all entries from January to March. The test set will of course contain the entries from April.

```
# Create new train and testsets for model
Trainset_final = subset(Final_table2, month(visit_date) %in% c("1","2","3"))
Testset_final = subset(Final_table2, month(visit_date) %in% c("4"))
```

To be on the safe side, the R2 of the models will be checked again, since we now have a larger trainset, at least on the summary of the model.

```
# Final Prediction model
Predictmodel_final <- lm(visitors ~ Mean_Visitors + reserve_visitors +
                           calendar_date, data=Trainset_final)
summary(Predictmodel_final)
```

```
##
## Call:
## lm(formula = visitors ~ Mean_Visitors + reserve_visitors + calendar_date,
##     data = Trainset_final)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -117.64   -4.06   -0.17    3.43  790.16
##
## Coefficients:
##                    Estimate Std. Error t value Pr(>|t|)
## (Intercept)      -8.253e+02  2.501e+01  -33.00   <2e-16 ***
## Mean_Visitors     9.755e-01  2.716e-03  359.20   <2e-16 ***
## reserve_visitors  2.022e-01  5.446e-03   37.12   <2e-16 ***
## calendar_date     4.795e-02  1.453e-03   33.00   <2e-16 ***
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 9.083 on 61799 degrees of freedom
## Multiple R-squared:  0.7043, Adjusted R-squared:  0.7043
## F-statistic: 4.906e+04 on 3 and 61799 DF,  p-value: < 2.2e-16
```

Since there is still a good R2 value of about 0.7, we can use this model for the prediction.

```r
# Final Prediction Test
PredictTest_final <- predict(Predictmodel_final, newdata=Testset_final)
PredictTest_final_frame <- as.data.frame(PredictTest_final)

NAs <- sum(is.na(PredictTest_final_frame))

Testset_final2 <- Testset_final
Testset_final2$visitors <- PredictTest_final

# Imputation of the 74 NAs of the prediction
imp_final <- mice(Testset_final2,m=1,seed=123)
```

```
##
##  iter imp variable
##   1   1  visitors  Mean_Visitors
##   2   1  visitors  Mean_Visitors
##   3   1  visitors  Mean_Visitors
##   4   1  visitors  Mean_Visitors
##   5   1  visitors  Mean_Visitors
```

```r
Testset_final2 <- complete(imp_final)
Testset_final2$visitors <- round(Testset_final2$visitors)
```

At last with the predict values for visitors for March, we get the output by *write.csv* function.

```r
project_submission <- read_csv("project_submission.csv")
```

```
## Parsed with column specification:
## cols(
##   ID = col_character(),
##   visitors = col_double()
## )
```

```r
Testset_predicition <- Testset_final2[,c(1,2,5)]

Predict_submission <- submission[,c(1,2)]

Predict_submission <- merge(Predict_submission,
                            Testset_predicition,by=c("ID", "visit_date"),
                            all.x = TRUE, sort = FALSE)

NA_submission <- sum(is.na(Predict_submission))
```

```r
final_submission <- project_submission
final_submission$visitors <- Predict_submission$visitors

write.csv(final_submission,"~/R\\project2_data_submission.csv",
          row.names = FALSE)
```

4 Results

Since we also include the visitors of the test set for the calculation of the mean value of the visitors and these are clearly not available for April, we have to assume that the R2 will be worse with the real test set. Nevertheless, with this and all the other variables added, we have hopefully improved the model.

```r
#plot the result
air_visit <- read_csv("air_visit.csv")
# Separate ID
submission_data <- final_submission %>%
  tidyr::separate(ID, sep=" _" ,
                  c("ID", "visit_date"))

# Change data style in submission
submission_data$visit_date <- as.Date(submission_data$visit_date)

#combine all the visits
combined_data <- rbind(air_visit,submission_data)
combined_data <- as.data.frame(combined_data)
combined_data_plot <- combined_data %>%  group_by(visit_date) %>%
  summarise(all_visitors = sum(visitors)) %>%
  as.data.frame()
```

For better understanding we plot all the visitors data considering the provided dates with the predicted values.

```r
#plot all the visits
combined_data_plot %>%
  ggplot()+
  geom_line(aes(visit_date,all_visitors), col = "blue") +
  geom_line(data = combined_data_plot[combined_data_plot$visit_date >= as.Date("2017-01-01") &
                                      combined_data_plot$visit_date <= "2017-04-01",],
            aes(x = visit_date, y = all_visitors), color = "red", size =1) +
  labs(x = "Date", y = "All visitors")+
  theme_bw()
```

In the figure above, the red line indicate the presented visitors by the data set and the blue line show the predicted visitors for April. It is clear that the, historical data have a very important impact on the model.

5 Further Suggestions

What we have now is definitely not the endpoint of this analysis. A data scientist could continue work forever to improve the performance of the models. Moving Forward, there are some areas that we would like to address and further explore to improve the analysis:

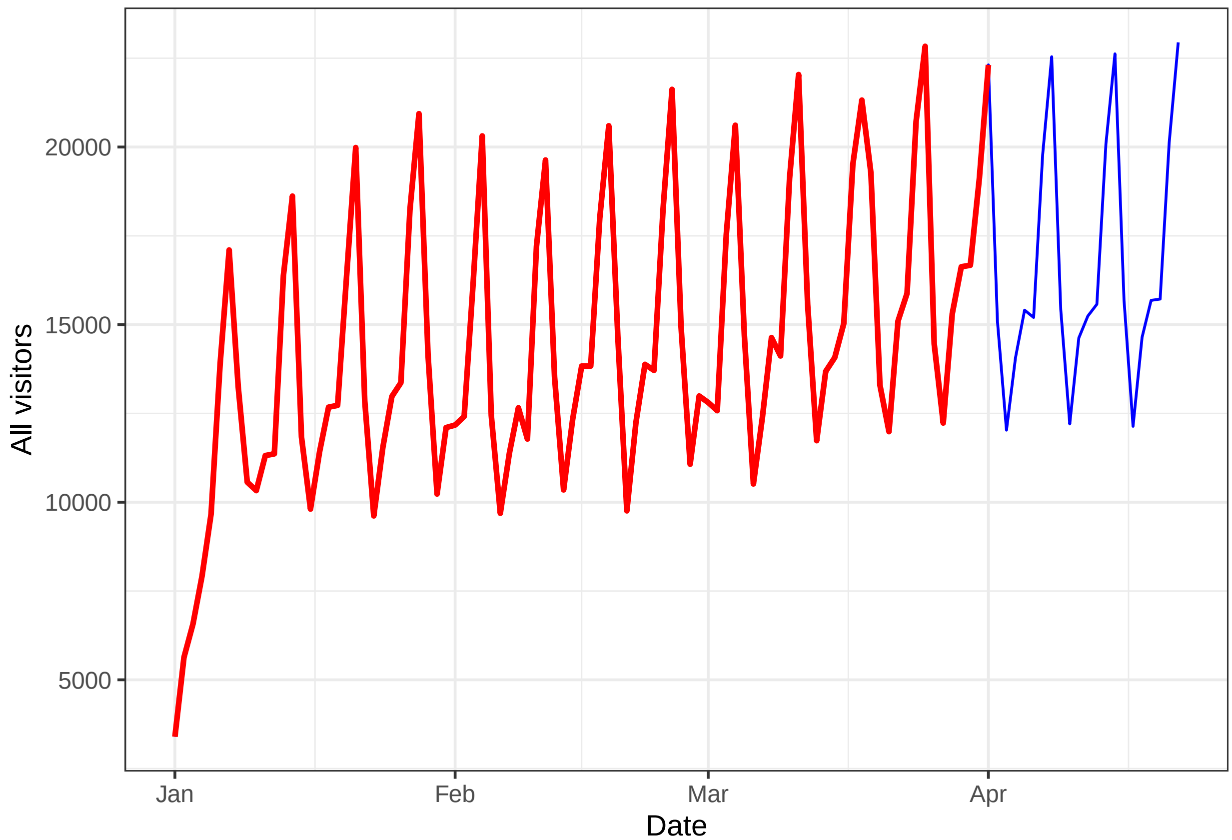


Figure 10: Provided Visits vs Predicted Visitors

- More Feature Explorations

In our analysis, we used features according to the time series of data sets. This limits our choice of features. Number of visitors to a restaurant certainly depends more things on its reservations, genre or area. There could also be a correlation between the social-economic conditions of the neighborhood, other attributes like the vibe of the restaurant itself and the number of visitors. Therefore, it would be valuable for us to also explore these other features as well.

- Times Series Models

What is presented is a time series case. In this context, it could also be valuable to explore deeper time series models with the dataset. In this report, we chose linear regression approach to start with because we did observe a slight pattern from the exploratory analysis. However, moving forward, it still worth trying other methods out and comparing it with the several models we built.

By now the visitor's data is available on a daily basis, but if it will be available on an hourly basis then we can further reduce the scale of prediction to next hour visitors prediction. It will give more control to restaurants in terms of being prepared for the next hour rush and thus further improvement of productivity and food delivery time.

At the end, the *timetk* package is briefly used in order to get a time series based analysis.

6 Use timetk package

The timetk package is a tool kit for time series analysis. It integrates well into the tidyverse ecosystem and is specifically designed to work with the tidy "tibble" data frames. Here we briefly describe the timetk approach and how we can apply it to our data.

First, we will create the train and validation data frames in the same way as for previous linear approach:

```
ID = "restaurant_ 319"

pred_len <- submission %>%
  distinct(visit_date) %>%
  nrow()


max_date <- max(air_visit$visit_date)
split_date <- max_date - pred_len
all_visits <- tibble(visit_date = seq(min(air_visit$visit_date),
                                      max(air_visit$visit_date), 1))


visits <- air_visit %>%
    right_join(all_visits, by = "visit_date") %>%
    mutate(visitors = log1p(visitors)) %>%
    rownames_to_column() %>%
    select(y = visitors,
          ds = visit_date)

visits_train <- visits %>% filter(ds <= split_date)
visits_test <- visits %>% filter(ds > split_date)
```

Then, we use the tk_augment_timeseries_signature tool to augment our data frames with time series characteristics. This means that we will add comprehensive time series properties that have been extracted from the date. Those new features include for instance the month, day and week of the year, half and quarter of the year. Here we show a str of the augmented training data:

```
visits_train_aug <- visits_train %>%
  tk_augment_timeseries_signature()
```

```
## tk_augment_timeseries_signature(): Using the following .date_var variable: ds
```

```
visits_test_aug <- visits_test %>%
  .$ds %>%
  tk_get_timeseries_signature()

str(visits_train_aug)
```

```
## tibble [45,982 x 30] (S3: tbl_df/tbl/data.frame)
##  $ y        : num [1:45982] 3.22 2.08 2.48 2.3 4.47 ...
##  $ ds       : Date[1:45982], format: "2017-01-01" "2017-01-01" ...
##  $ index.num: num [1:45982] 1.48e+09 1.48e+09 1.48e+09 1.48e+09 1.48e+09 ...
##  $ diff     : num [1:45982] NA 0 0 0 0 0 0 0 0 0 ...
##  $ year     : int [1:45982] 2017 2017 2017 2017 2017 2017 2017 2017 2017 2017 ...
##  $ year.iso : int [1:45982] 2016 2016 2016 2016 2016 2016 2016 2016 2016 2016 ...
##  $ half     : int [1:45982] 1 1 1 1 1 1 1 1 1 1 ...
##  $ quarter  : int [1:45982] 1 1 1 1 1 1 1 1 1 1 ...
##  $ month    : int [1:45982] 1 1 1 1 1 1 1 1 1 1 ...
##  $ month.xts: int [1:45982] 0 0 0 0 0 0 0 0 0 0 ...
##  $ month.lbl: Ord.factor w/ 12 levels "January"<"February"<..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ day      : int [1:45982] 1 1 1 1 1 1 1 1 1 1 ...
##  $ hour     : int [1:45982] 0 0 0 0 0 0 0 0 0 0 ...
##  $ minute   : int [1:45982] 0 0 0 0 0 0 0 0 0 0 ...
##  $ second   : int [1:45982] 0 0 0 0 0 0 0 0 0 0 ...
##  $ hour12   : int [1:45982] 0 0 0 0 0 0 0 0 0 0 ...
##  $ am.pm    : int [1:45982] 1 1 1 1 1 1 1 1 1 1 ...
##  $ wday     : int [1:45982] 1 1 1 1 1 1 1 1 1 1 ...
##  $ wday.xts : int [1:45982] 0 0 0 0 0 0 0 0 0 0 ...
##  $ wday.lbl : Ord.factor w/ 7 levels "Sunday"<"Monday"<..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ mday     : int [1:45982] 1 1 1 1 1 1 1 1 1 1 ...
##  $ qday     : int [1:45982] 1 1 1 1 1 1 1 1 1 1 ...
##  $ yday     : int [1:45982] 1 1 1 1 1 1 1 1 1 1 ...
##  $ mweek    : int [1:45982] 1 1 1 1 1 1 1 1 1 1 ...
##  $ week     : int [1:45982] 1 1 1 1 1 1 1 1 1 1 ...
##  $ week.iso : int [1:45982] 52 52 52 52 52 52 52 52 52 52 ...
##  $ week2    : int [1:45982] 1 1 1 1 1 1 1 1 1 1 ...
##  $ week3    : int [1:45982] 1 1 1 1 1 1 1 1 1 1 ...
##  $ week4    : int [1:45982] 1 1 1 1 1 1 1 1 1 1 ...
##  $ mday7    : int [1:45982] 1 1 1 1 1 1 1 1 1 1 ...
```

Now, the idea behind timetk is to use these new features to make predictions based on a regression or classification approach; with standard tools such as linear/logistic regression. For this approach, we will use a simple linear model. This analysis can easily be extended to more sophisticated methods.

```r
fit_lm <- lm(y ~ ., data = select(visits_train_aug,
                                  -c(ds, diff, wday.xts, wday.lbl, year.iso)))
summary(fit_lm)
```

```
##
## Call:
## lm(formula = y ~ ., data = select(visits_train_aug, -c(ds, diff,
##     wday.xts, wday.lbl, year.iso)))
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.2561 -0.5059  0.0860  0.5973  4.0296
##
## Coefficients: (16 not defined because of singularities)
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -5.379e+01  3.381e+01  -1.591   0.1116
## index.num    3.800e-08  2.284e-08   1.664   0.0961 .
## year                NA         NA      NA       NA
## half                NA         NA      NA       NA
## quarter             NA         NA      NA       NA
## month       -7.160e-02  5.739e-02  -1.248   0.2122
## month.xts           NA         NA      NA       NA
## month.lbl.L         NA         NA      NA       NA
## month.lbl.Q -1.176e-02  7.467e-03  -1.575   0.1153
## day                 NA         NA      NA       NA
## hour                NA         NA      NA       NA
## minute              NA         NA      NA       NA
## second              NA         NA      NA       NA
## hour12              NA         NA      NA       NA
## am.pm               NA         NA      NA       NA
## wday         4.939e-02  1.946e-03  25.380   <2e-16 ***
## mday                NA         NA      NA       NA
## qday                NA         NA      NA       NA
## yday                NA         NA      NA       NA
## mweek               NA         NA      NA       NA
## week                NA         NA      NA       NA
## week.iso     6.916e-07  1.307e-03   0.001   0.9996
## week2        1.295e-02  8.109e-03   1.597   0.1104
## week3        2.599e-03  5.319e-03   0.489   0.6251
## week4       -2.980e-03  4.214e-03  -0.707   0.4794
## mday7       -2.722e-03  1.328e-02  -0.205   0.8376
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.7987 on 45972 degrees of freedom
## Multiple R-squared:  0.01524,    Adjusted R-squared:  0.01505
## F-statistic: 79.05 on 9 and 45972 DF,  p-value: < 2.2e-16
```

Then we use the predict tool to apply the model to our prediction range. Make sure to include the same features as in the training data frame.

```r
pred <- predict(fit_lm, newdata = select(visits_test_aug,
                                         -c(index, diff, wday.xts, wday.lbl, year.iso)))
```

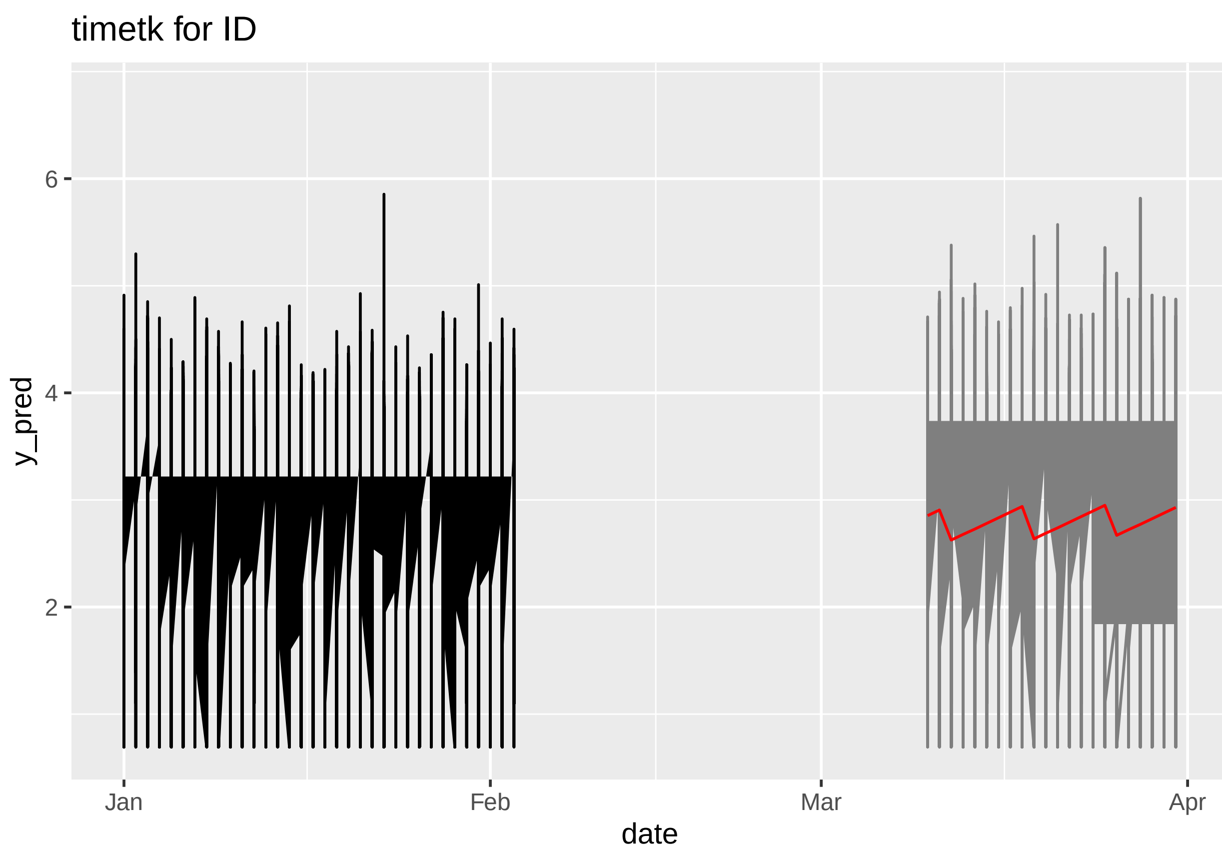
```
## Warning in predict.lm(fit_lm, newdata = select(visits_test_aug, -c(index, :
## prediction from a rank-deficient fit may be misleading
```

```r
pred_tk <- tibble(
    date  = visits_test$ds,
    value = pred
    )
```

6.1 Loop for all restaurants IDs

```r
plot_tk_lm_ID <- function(ID){

pred_len <- submission %>%
  distinct(visit_date) %>%
  nrow()

max_date <- max(air_visit$visit_date)
  split_date <- max_date - pred_len
  all_visits <- tibble(visit_date = seq(min(air_visit$visit_date),
                                        max(air_visit$visit_date), 1))


visits <- air_visit %>%
    right_join(all_visits, by = "visit_date") %>%
    mutate(visitors = log1p(visitors)) %>%
    rownames_to_column() %>%
    select(y = visitors,
          ds = visit_date)

visits_train <- visits %>% filter(ds <= split_date)
visits_test <- visits %>% filter(ds > split_date)

#augment train with ts info
visits_train_aug <- visits_train %>%
    tk_augment_timeseries_signature()

# fit lm
fit_lm <- lm(y ~ ., data = select(visits_train_aug,
                                  -c(ds, diff, wday.xts, wday.lbl, year.iso)))

# augment valid with ts info
visits_test_aug <- visits_test %>%
    .$ds %>%
    tk_get_timeseries_signature()

# predict from lm
pred <- predict(fit_lm, newdata = select(visits_test_aug,
                                         -c(index, diff, wday.xts, wday.lbl, year.iso)))
```

```
pred_tk <- tibble(
      date   = visits_test$ds,
      y_pred = pred
      )

# plot
p <- pred_tk %>%
 ggplot(aes(date, y_pred)) +
 geom_line(data = visits_train, aes(ds, y), colour = "black") +
 geom_line(data = visits_test, aes(ds, y), colour = "grey50") +
 geom_line(colour = "red") +
 labs(title = str_c("timetk for ", ID))

  return(p)
}

plot_tk_lm_ID("ID")
```



According to the result, it seems that the outcome is quite sensible. Particularly considering that we only used a simply linear model for our prediction. More elaborate methods for time series forecasting are likely to give a better result.

7 References

[1] R Core Team (2019). 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] Hao Zhu (2020). kableExtra: Construct Complex Table with 'kable' and Pipe Syntax. R package version 1.2.1. https://CRAN.R-project.org/package=kableExtra

[4] Erich Neuwirth (2014). RColorBrewer: ColorBrewer Palettes. R package version 1.1-2. https://CRAN.R-project.org/package=RColorBrewer

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

[6] Maechler, M., Rousseeuw, P., Struyf, A., Hubert, M., Hornik, K.(2019). cluster: Cluster Analysis Basics and Extensions. R package version 2.1.0.

[7] Diethelm Wuertz, Tobias Setz, Yohan Chalabi, Martin Maechler and Joe W. Byers (2018). timeDate: Rmetrics - Chronological and Calendar Objects. R package version 3043.102. https://CRAN.R-project.org/package=timeDate

[8] Stef van Buuren, Karin Groothuis-Oudshoorn (2011). mice: Multivariate Imputation by Chained Equations in R. Journal of Statistical Software, 45(3), 1-67. URL https://www.jstatsoft.org/v45/i03/.

[9] Baptiste Auguie (2017). gridExtra: Miscellaneous Functions for "Grid" Graphics. R package version 2.3. https://CRAN.R-project.org/package=gridExtra

[10] Hadley Wickham (2011). The Split-Apply-Combine Strategy for Data Analysis. Journal of Statistical Software, 40(1), 1-29. URL http://www.jstatsoft.org/v40/i01/.

[11] Kirill Müller, Jeroen Ooms, David James, Saikat DebRoy, Hadley Wickham and Jeffrey Horner (2020). RMariaDB: Database Interface and 'MariaDB' Driver. R package version 1.0.10. https://CRAN.R-project.org/package=RMariaDB

[12] Joe Cheng, Bhaskar Karambelkar and Yihui Xie (2019). leaflet: Create Interactive Web Maps with the JavaScript 'Leaflet' Library. R package version 2.0.3. https://CRAN.R-project.org/package=leaflet

[13] Matt Dancho and Davis Vaughan (2020). timetk: A Tool Kit for Working with Time Series in R. R package version 2.6.0. https://CRAN.R-project.org/package=timetk