# FoodieX

10/20/2020

```
## load packages
library(VIM)
```

```
## Warning: package 'VIM' was built under R version 4.0.2
```

```
## Loading required package: colorspace
```

```
## Loading required package: grid
```

```
## VIM is ready to use.
```

```
## Suggestions and bug-reports can be submitted at: https://github.com/statistikat/VIM/issues
```

```
##
## Attaching package: 'VIM'
```

```
## The following object is masked from 'package:datasets':
##
##     sleep
```

```
library(tidyverse)
```

```
## -- Attaching packages ------------------------------------------------ tidyverse 1.3.0 --
```

```
## v ggplot2 3.3.2     v purrr   0.3.4
## v tibble  3.0.3     v dplyr   1.0.2
## v tidyr   1.1.1     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0
```

```
## Warning: package 'ggplot2' was built under R version 4.0.2
```

```
## Warning: package 'tibble' was built under R version 4.0.2
```

```
## Warning: package 'tidyr' was built under R version 4.0.2
```

```
## Warning: package 'dplyr' was built under R version 4.0.2
```

```
## -- Conflicts --------------------------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library("esquisse")
```

```
## Warning: package 'esquisse' was built under R version 4.0.2
```

```
library(caret)
```

```
## Warning: package 'caret' was built under R version 4.0.2
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:purrr':
##
##     lift
```

```
library(fpc)
```

```
## Warning: package 'fpc' was built under R version 4.0.2
```

## Data Preprocessing

```
## load data
data_raw = read.csv("2020-XTern-DS.csv")
```

```
data_type = data_raw %>%
  mutate(
    Average_Cost = as.numeric(substr(Average_Cost, start = 2, stop = 3)),
    Minimum_Order = as.numeric(substr(Minimum_Order, start = 2, stop = 3)),
    Rating = as.numeric(Rating),
    Votes = as.numeric(Votes),
    Reviews = as.numeric(Reviews),
    Cook_Time = as.numeric(substr(Cook_Time, start = 1, stop = 3))
    )%>%
  mutate(Num_Cuisines = str_count(Cuisines, ','))
```

```
## Warning: Problem with `mutate()` input `Average_Cost`.
## i NAs introduced by coercion
## i Input `Average_Cost` is `as.numeric(substr(Average_Cost, start = 2, stop = 3))`.
```

```
## Warning in mask$eval_all_mutate(dots[[i]]): NAs introduced by coercion
```

```
## Warning: Problem with `mutate()` input `Rating`.
## i NAs introduced by coercion
## i Input `Rating` is `as.numeric(Rating)`.
```

```
## Warning in mask$eval_all_mutate(dots[[i]]): NAs introduced by coercion
```

```
## Warning: Problem with `mutate()` input `Votes`.
## i NAs introduced by coercion
## i Input `Votes` is `as.numeric(Votes)`.
```

```
## Warning in mask$eval_all_mutate(dots[[i]]): NAs introduced by coercion
```

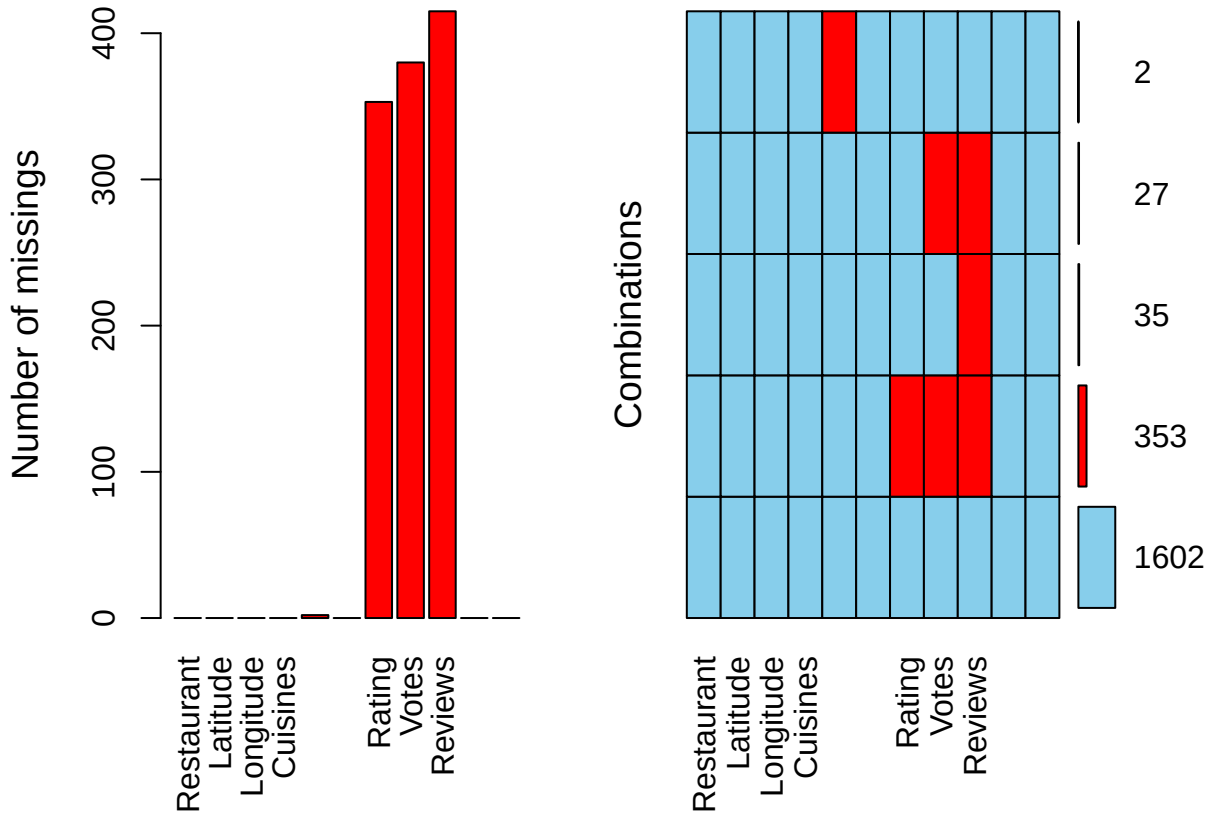
```
## Warning: Problem with `mutate()` input `Reviews`.
## i NAs introduced by coercion
## i Input `Reviews` is `as.numeric(Reviews)`.
```

```
## Warning in mask$eval_all_mutate(dots[[i]]): NAs introduced by coercion
```

```
## Check the columns with missing values
## Calculate missing data percentage of each col
missing_table = sort(colSums(is.na(data_type))/nrow(data_type), decreasing = TRUE)
missing_table
```

```
##       Reviews         Votes        Rating  Average_Cost    Restaurant
##  0.2055473006  0.1882119861  0.1748390292  0.0009905894  0.0000000000
##      Latitude     Longitude      Cuisines Minimum_Order     Cook_Time
##  0.0000000000  0.0000000000  0.0000000000  0.0000000000  0.0000000000
##  Num_Cuisines
##  0.0000000000
```

```
## visualizing the pattern of missing value
aggr(data_type , prop = FALSE, number = TRUE)
```




```
## Remove the column if more than 30% observations are missing.
## Fill the missing value with the median of its column.
data_nona = data_type %>%
  select_if(~sum(is.na(.))/nrow(data_type) < 0.3) %>%
  mutate_if(is.numeric, function(x){x = replace_na(x, median(x, na.rm = TRUE))})
## Check the missing value again
mean(is.na(data_nona)/nrow(data_nona))
```

```
## [1] 0
```

```
data = data_nona %>%
  mutate(Rating_Class = as_factor(case_when(
                              Rating <3.6 ~ "low",
                              Rating >=3.6 ~ "high"))
  )
```

### Conculsion 1: Identify the trending restaurants with My own scoring algorithm

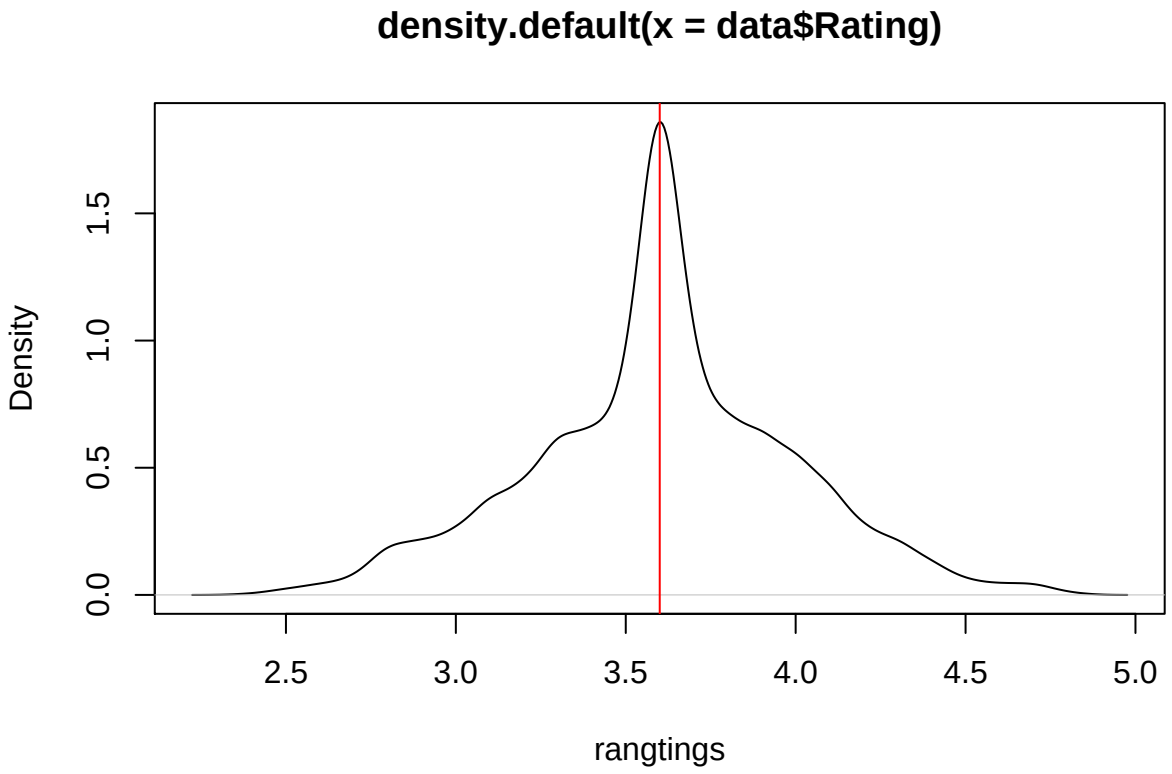
The ID of trending restaurants are as follows: ID_1160, ID_6967, ID_6537, ID_7158, ID_4728, ID_7739, ID_981. The reasons are as follows. The average ratings not less than 4.5 with both the number of votes and reviews are more than the medain. Also, not to let the cutomers to wait too long, the cook time should under 60 minutes. Restaurants should also provided more than 3 types of cuisines and the the average cost should be less or equal to $60.

```
data %>%
  filter(Votes> median(Votes),
```

```
        Reviews > median(Reviews),
        Rating >= 4.5,
        Num_Cuisines > 3,
        Cook_Time <= 60,
        Average_Cost <= 60
        ) %>%
  select(-Latitude, -Longitude)
```

```
##   Restaurant
## 1    ID_1160
## 2    ID_6967
## 3    ID_6537
## 4    ID_7158
## 5    ID_4728
## 6    ID_7739
## 7     ID_981
##                                                                 Cuisines
## 1 Asian, Burmese, Bubble Tea, Desserts, Salad, Tea, Beverages, Ice Cream
## 2             Cafe, European, Continental, Sandwich, Salad, Healthy Food
## 3                           Biryani, North Indian, Mughlai, Kebab, Rolls
## 4                              Ice Cream, Cafe, Pizza, Burger, Beverages
## 5                           North Indian, Mughlai, Biryani, Rolls, Momos
## 6                               Cafe, Desserts, French, Bakery, European
## 7                      Cafe, Salad, Italian, American, Bakery, Beverages
##   Average_Cost Minimum_Order Rating Votes Reviews Cook_Time Num_Cuisines
## 1           60            50    4.7   914     499        45            7
## 2           60            50    4.6   391     174        30            5
## 3           25            99    4.7   706     490        30            4
## 4           20            50    4.5  2805    1457        45            4
## 5           25            99    4.8   650     423        45            4
## 6           25            50    4.6  1502     819        30            4
## 7           40            50    4.5   879     518        45            5
##   Rating_Class
## 1         high
## 2         high
## 3         high
## 4         high
## 5         high
## 6         high
## 7         high
```

```
plot(density(data$Rating), xlab = "rangtings")
abline(v = median(data$Rating), col = "red")
```

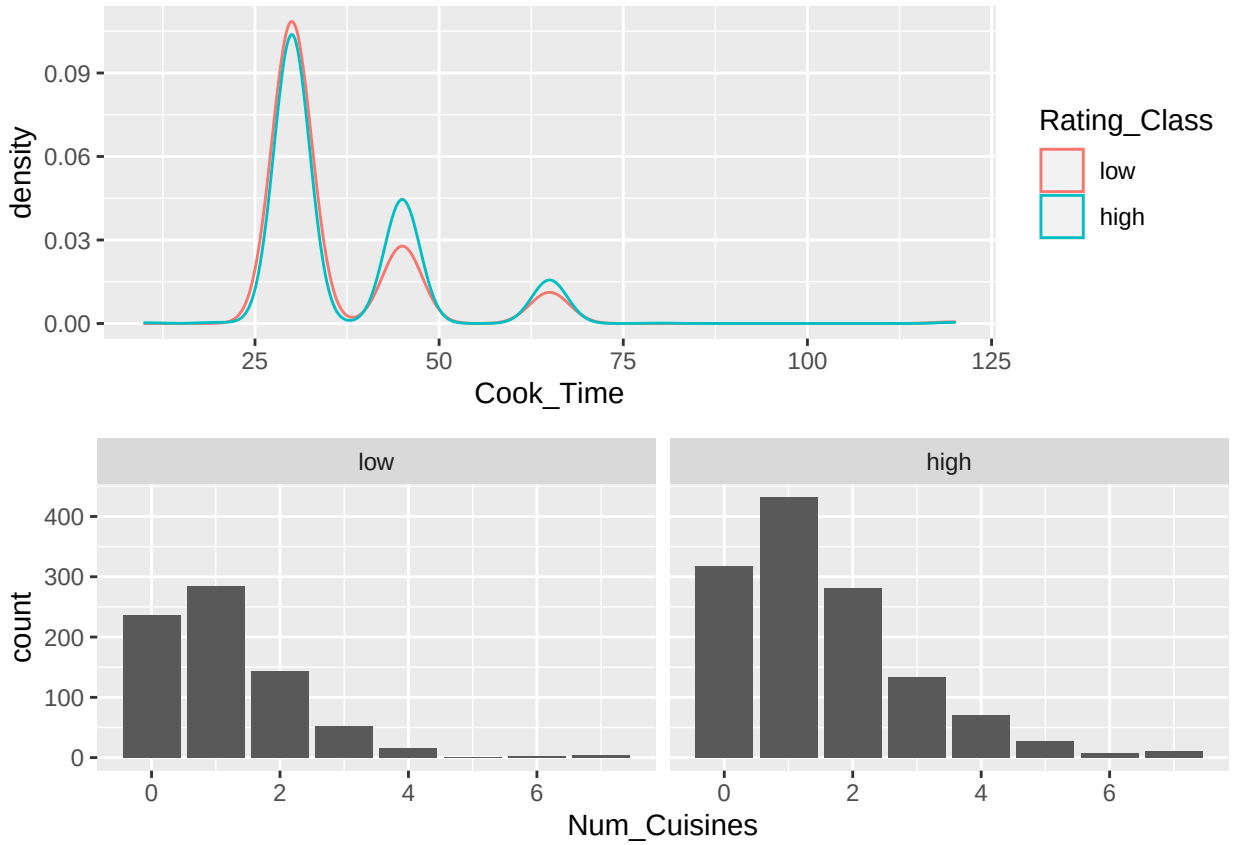



### Conculsion 2 with Data visualizatoins:

Restaurant with high ratings usually yake longer to prepare food. The number of cuisines do not play an important role in restaurant ratings.

```
# cook time vs rating class
p01 = data %>%
  ggplot(aes(x = Cook_Time, col = Rating_Class)) +
  geom_density()

# number of cuisines vs rating class
p02 = data %>%
  ggplot(aes(x = Num_Cuisines, fill = Num_Cuisines)) +
  geom_bar() +
  facet_wrap(~Rating_Class)
gridExtra::grid.arrange(p01, p02, ncol = 1)
```



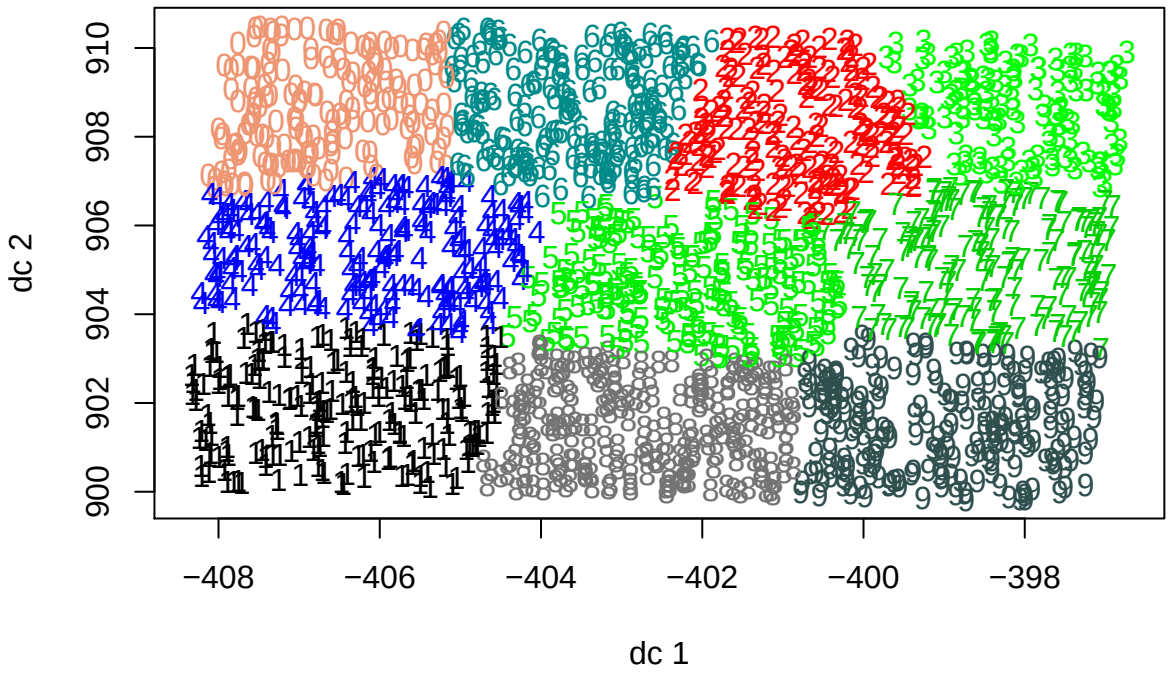

### Conculsion 3:clustering restaurant locations to figure out the optimized FoodieX pick up zones

The location (Latitude, Longitude) of 10 best pick up zones are list as follows.

```
pickup = kmeans(data %>% select(Latitude, Longitude),
                        centers = 10, nstart = 5)
pickup$centers
```

```
##    Latitude Longitude
## 1  39.82833 -85.17466
## 2  39.35278 -85.79657
## 3  39.11337 -85.85663
## 4  39.81337 -85.51485
## 5  39.45878 -85.45549
## 6  39.60321 -85.83044
## 7  39.13848 -85.51959
## 8  39.49552 -85.14734
## 9  39.16606 -85.16829
## 10 39.87896 -85.82055
```

```
plotcluster(data %>% select(Latitude, Longitude), pickup$cluster)
```




### Conculsion 4: Estimating cook time based on restaurant info

Average_Cost, Minimum_Order, Votes, Reviews are significant factors that will affact cook time.

#### Linear model

```
cooktime_lm_model = lm(Cook_Time ~ .-Restaurant-Cuisines -Latitude-Longitude -Rating_Class,data = data)
summary(cooktime_lm_model)
```

```
##
## Call:
## lm(formula = Cook_Time ~ . - Restaurant - Cuisines - Latitude -
##     Longitude - Rating_Class, data = data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -35.630  -5.645  -4.342   5.650  84.081
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)   22.892182   2.684303   8.528  < 2e-16 ***
## Average_Cost   0.142292   0.024203   5.879 4.82e-09 ***
## Minimum_Order  0.176991   0.018380   9.629  < 2e-16 ***
## Rating         0.098188   0.727183   0.135   0.8926
## Votes          0.009994   0.001858   5.378 8.41e-08 ***
## Reviews       -0.012640   0.003203  -3.946 8.21e-05 ***
## Num_Cuisines   0.515438   0.206788   2.493   0.0128 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 11.07 on 2012 degrees of freedom
## Multiple R-squared:  0.1231, Adjusted R-squared:  0.1205
```

```
## F-statistic: 47.06 on 6 and 2012 DF,  p-value: < 2.2e-16
```

### Random Forest Model

```
set.seed(42)
cooktime_rf_model = train(Cook_Time ~ .-Restaurant-Cuisines -Latitude-Longitude -Rating_Class,
      data = data,
      trControl = trainControl(method = "oob"),
      method = "rf")
cooktime_rf_model$finalModel
```

```
##
## Call:
##  randomForest(x = x, y = y, mtry = param$mtry)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##           Mean of squared residuals: 103.2375
##                     % Var explained: 25.87
```

```
cooktime_rf_model$results
```

```
##       RMSE  Rsquared mtry
## 1 10.17692 0.2563582    2
## 2 10.27958 0.2412797    4
## 3 10.35339 0.2303456    6
```