Statistical Learning Project

Andrea Ierardi

pdf_document: default html_document: default

Libraries

```
library(knitr)
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 3.6.3
```

```
library(plotly)
```

```
## Warning: package 'plotly' was built under R version 3.6.3
```

```
##
## Attaching package: 'plotly'
```

```
## The following object is masked from 'package:ggplot2':
##
##     last_plot
```

```
## The following object is masked from 'package:stats':
##
##     filter
```

```
## The following object is masked from 'package:graphics':
##
##     layout
```

```
library(tidyr)
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 3.6.3
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(png)
library(ggpubr)
```

```
## Warning: package 'ggpubr' was built under R version 3.6.3
```

```
library(tidyverse)
```

```
## -- Attaching packages ---------------------------------------------------------------------------
## v tibble  3.0.1     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0
## v purrr   0.3.4
## Warning: package 'tibble' was built under R version 3.6.3
## Warning: package 'stringr' was built under R version 3.6.3
## Warning: package 'forcats' was built under R version 3.6.3
## -- Conflicts ------------------------------------------------------------------------------------
## x dplyr::filter() masks plotly::filter(), stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(caTools)
```

```
## Warning: package 'caTools' was built under R version 3.6.3
```

```
library(caret)
```

```
## Warning: package 'caret' was built under R version 3.6.3
## Loading required package: lattice
##
## Attaching package: 'caret'
## The following object is masked from 'package:purrr':
##
##     lift
```

```
library(tree)
```

```
## Warning: package 'tree' was built under R version 3.6.3
## Registered S3 method overwritten by 'tree':
##   method     from
##   print.tree cli
```

```
library(MASS)
```

```
## Warning: package 'MASS' was built under R version 3.6.3
##
## Attaching package: 'MASS'
## The following object is masked from 'package:dplyr':
##
##     select
## The following object is masked from 'package:plotly':
##
##     select
```

```
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 3.6.3
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
##
## Attaching package: 'randomForest'
## The following object is masked from 'package:dplyr':
##
##     combine
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
library(ranger)
```

```
## Warning: package 'ranger' was built under R version 3.6.3
##
## Attaching package: 'ranger'
## The following object is masked from 'package:randomForest':
##
##     importance
```

```
library(tuneRanger)
```

```
## Warning: package 'tuneRanger' was built under R version 3.6.3
## Loading required package: mlrMBO
## Warning: package 'mlrMBO' was built under R version 3.6.3
## Loading required package: mlr
## Warning: package 'mlr' was built under R version 3.6.3
## Loading required package: ParamHelpers
## Warning: package 'ParamHelpers' was built under R version 3.6.3
## 'mlr' is in maintenance mode since July 2019. Future development
## efforts will go into its successor 'mlr3' (<https://mlr3.mlr-org.com>).
##
## Attaching package: 'mlr'
## The following object is masked from 'package:caret':
##
##     train
## Loading required package: smoof
## Warning: package 'smoof' was built under R version 3.6.3
## Loading required package: checkmate
## Loading required package: parallel
## Loading required package: lubridate
## Warning: package 'lubridate' was built under R version 3.6.3
##
## Attaching package: 'lubridate'
```

```
## The following objects are masked from 'package:dplyr':
##
##     intersect, setdiff, union

## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union

## Loading required package: lhs

## Warning: package 'lhs' was built under R version 3.6.3
```

```
library(keras)
```

```
## Warning: package 'keras' was built under R version 3.6.3
```

```
library(formattable)
```

```
## Warning: package 'formattable' was built under R version 3.6.3

##
## Attaching package: 'formattable'

## The following object is masked from 'package:keras':
##
##     normalize

## The following object is masked from 'package:MASS':
##
##     area

## The following object is masked from 'package:plotly':
##
##     style
```

```
library(clustMixType)
```

```
## Warning: package 'clustMixType' was built under R version 3.6.3
```

```
library(cluster)
library(dendextend)
```

```
## Warning: package 'dendextend' was built under R version 3.6.3

##
## ---------------------
## Welcome to dendextend version 1.13.4
## Type citation('dendextend') for how to cite the package.
##
## Type browseVignettes(package = 'dendextend') for the package vignette.
## The github page is: https://github.com/talgalili/dendextend/
##
## Suggestions and bug-reports can be submitted at: https://github.com/talgalili/dendextend/issues
## Or contact: <tal.galili@gmail.com>
##
##  To suppress this message use:  suppressPackageStartupMessages(library(dendextend))
## ---------------------

##
## Attaching package: 'dendextend'
```

```
## The following object is masked from 'package:ggpubr':
##
##     rotate
```

```
## The following object is masked from 'package:stats':
##
##     cutree
```

```
library(readr)

library(factoextra)
```

```
## Warning: package 'factoextra' was built under R version 3.6.3
```

```
## Welcome! Want to learn more? See two factoextra-related books at https://goo.gl/ve3WBa
```

```
library(FactoMineR)
```

```
## Warning: package 'FactoMineR' was built under R version 3.6.3
```

```
library(PCAmixdata)
```

```
## Warning: package 'PCAmixdata' was built under R version 3.6.3
```

Dataset

Link here

```
ds = read.csv("AB_NYC_2019.csv")
```

Data Inspection

```
head(ds)
```

```
##     id                                              name host_id   host_name
## 1 2539               Clean & quiet apt home by the park    2787        John
## 2 2595                            Skylit Midtown Castle    2845    Jennifer
## 3 3647              THE VILLAGE OF HARLEM....NEW YORK !    4632   Elisabeth
## 4 3831                   Cozy Entire Floor of Brownstone    4869 LisaRoxanne
## 5 5022 Entire Apt: Spacious Studio/Loft by central park    7192       Laura
## 6 5099        Large Cozy 1 BR Apartment In Midtown East    7322       Chris
##   neighbourhood_group neighbourhood latitude longitude       room_type price
## 1            Brooklyn    Kensington 40.64749 -73.97237    Private room   149
## 2           Manhattan       Midtown 40.75362 -73.98377 Entire home/apt   225
## 3           Manhattan        Harlem 40.80902 -73.94190    Private room   150
## 4            Brooklyn  Clinton Hill 40.68514 -73.95976 Entire home/apt    89
## 5           Manhattan   East Harlem 40.79851 -73.94399 Entire home/apt    80
## 6           Manhattan   Murray Hill 40.74767 -73.97500 Entire home/apt   200
##   minimum_nights number_of_reviews last_review reviews_per_month
## 1              1                 9  2018-10-19              0.21
## 2              1                45  2019-05-21              0.38
## 3              3                 0                          NA
## 4              1               270  2019-07-05              4.64
## 5             10                 9  2018-11-19              0.10
## 6              3                74  2019-06-22              0.59
##   calculated_host_listings_count availability_365
```

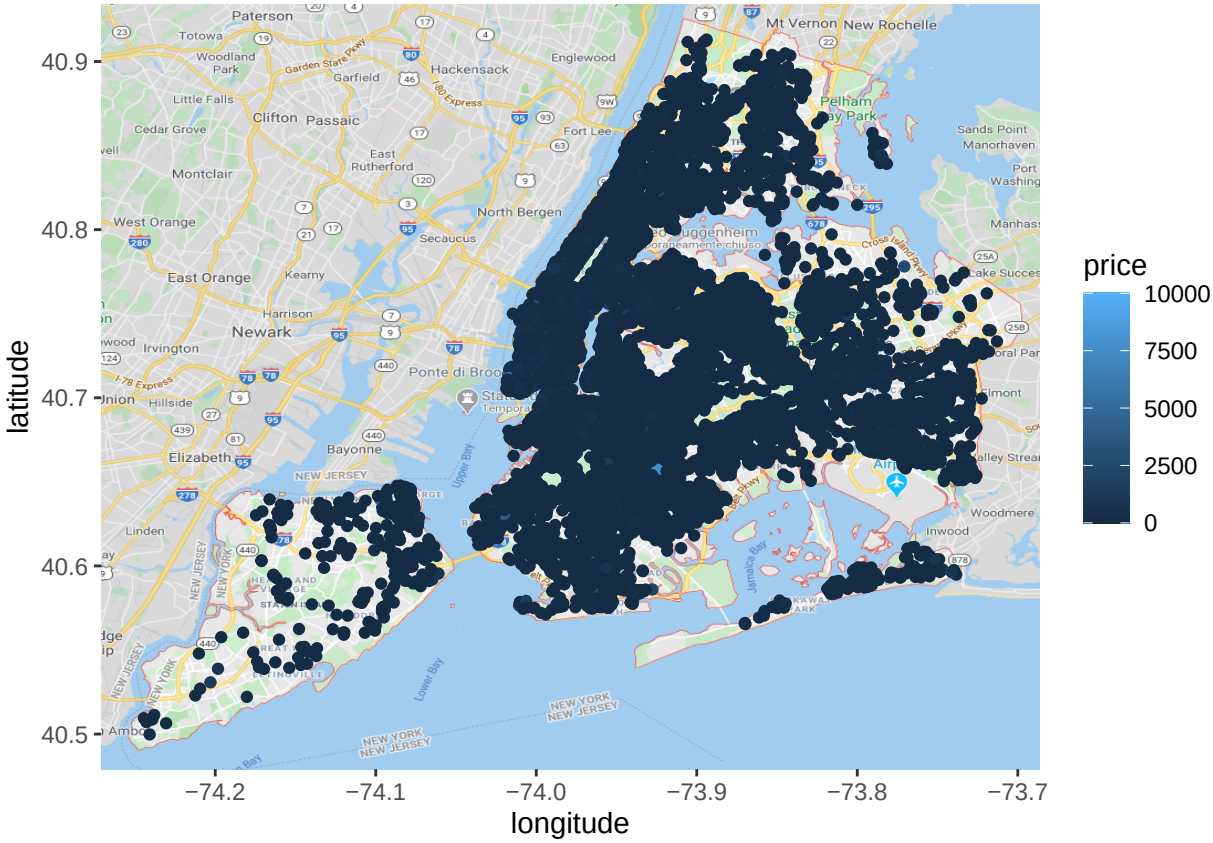
```
## 1                               6             365
## 2                               2             355
## 3                               1             365
## 4                               1             194
## 5                               1               0
## 6                               1             129
```

```
summary(ds)
```

```
##        id                                     name
##  Min.   :    2539   Hillside Hotel                :   18
##  1st Qu.: 9471945   Home away from home           :   17
##  Median :19677284                                 :   16
##  Mean   :19017143   New york Multi-unit building  :   16
##  3rd Qu.:29152178   Brooklyn Apartment            :   12
##  Max.   :36487245   Loft Suite @ The Box House Hotel:   11
##                     (Other)                       :48805
##     host_id                host_name        neighbourhood_group
##  Min.   :     2438   Michael     :  417   Bronx        : 1091
##  1st Qu.:  7822033   David       :  403   Brooklyn     :20104
##  Median : 30793816   Sonder (NYC):  327   Manhattan    :21661
##  Mean   : 67620011   John        :  294   Queens       : 5666
##  3rd Qu.:107434423   Alex        :  279   Staten Island:  373
##  Max.   :274321313   Blueground  :  232
##                      (Other)     :46943
##             neighbourhood      latitude       longitude
##  Williamsburg      : 3920   Min.   :40.50   Min.   :-74.24
##  Bedford-Stuyvesant: 3714   1st Qu.:40.69   1st Qu.:-73.98
##  Harlem            : 2658   Median :40.72   Median :-73.96
##  Bushwick          : 2465   Mean   :40.73   Mean   :-73.95
##  Upper West Side   : 1971   3rd Qu.:40.76   3rd Qu.:-73.94
##  Hell's Kitchen    : 1958   Max.   :40.91   Max.   :-73.71
##  (Other)           :32209
##            room_type         price         minimum_nights    number_of_reviews
##  Entire home/apt:25409   Min.   :    0.0   Min.   :   1.00   Min.   :  0.00
##  Private room   :22326   1st Qu.:   69.0   1st Qu.:   1.00   1st Qu.:  1.00
##  Shared room    : 1160   Median :  106.0   Median :   3.00   Median :  5.00
##                          Mean   :  152.7   Mean   :   7.03   Mean   : 23.27
##                          3rd Qu.:  175.0   3rd Qu.:   5.00   3rd Qu.: 24.00
##                          Max.   :10000.0   Max.   :1250.00   Max.   :629.00
##
##      last_review    reviews_per_month calculated_host_listings_count
##            :10052   Min.   : 0.010    Min.   :  1.000
##  2019-06-23: 1413   1st Qu.: 0.190    1st Qu.:  1.000
##  2019-07-01: 1359   Median : 0.720    Median :  1.000
##  2019-06-30: 1341   Mean   : 1.373    Mean   :  7.144
##  2019-06-24:  875   3rd Qu.: 2.020    3rd Qu.:  2.000
##  2019-07-07:  718   Max.   :58.500    Max.   :327.000
##  (Other)   :33137   NA's   :10052
##  availability_365
##  Min.   :  0.0
##  1st Qu.:  0.0
##  Median : 45.0
##  Mean   :112.8
##  3rd Qu.:227.0
```

```
##  Max.   :365.0
##
```

```
img <- readPNG("map.png")


map_ds = ggplot() + background_image(img)+ geom_point(data = ds,  aes(y=latitude,x = longitude, color =
map_ds
```



Data cleaning

Check for NA and NULL values

```
apply(ds,2,function(x) sum(is.na(x)))
```

```
##                          id                        name
##                           0                           0
##                     host_id                   host_name
##                           0                           0
##           neighbourhood_group               neighbourhood
##                           0                           0
##                    latitude                   longitude
##                           0                           0
##                   room_type                       price
##                           0                           0
##               minimum_nights           number_of_reviews
```

```
##                             0                             0
##                   last_review              reviews_per_month
##                             0                         10052
## calculated_host_listings_count               availability_365
##                             0                             0
```

Variable selection

```
dataset = ds  %>% dplyr::select(neighbourhood_group,latitude, longitude, room_type,price)

head(dataset)
```

```
##   neighbourhood_group latitude longitude       room_type price
## 1            Brooklyn 40.64749 -73.97237    Private room   149
## 2           Manhattan 40.75362 -73.98377 Entire home/apt   225
## 3           Manhattan 40.80902 -73.94190    Private room   150
## 4            Brooklyn 40.68514 -73.95976 Entire home/apt    89
## 5           Manhattan 40.79851 -73.94399 Entire home/apt    80
## 6           Manhattan 40.74767 -73.97500 Entire home/apt   200
```

Variable scaling

```
scale_data = function(df)
{
  df = df %>%filter( price >= 15  & price <= 500)

  numerical = c("price")
  numerical2 = c("latitude","longitude")
  categorical = c("room_type", "neighbourhood_group")

  for( cat in categorical )
  {
    df[cat] = factor(df[[cat]],
                     level = unique(df[[cat]]),
                     labels = c(1:length(unique(df[[cat]])) ))
  }

  df[numerical] = as.numeric(scale(df[numerical]))


  df2 = df
  df2[numerical2] = as.numeric(scale(df2[numerical2]))
  df3 = list()
  df3$df2 = df2
  df3$df = df


  return(df3)


}
dataframe = scale_data(dataset)

dataset = dataframe$df
```

```
data = dataframe$df2

head(dataset)
```

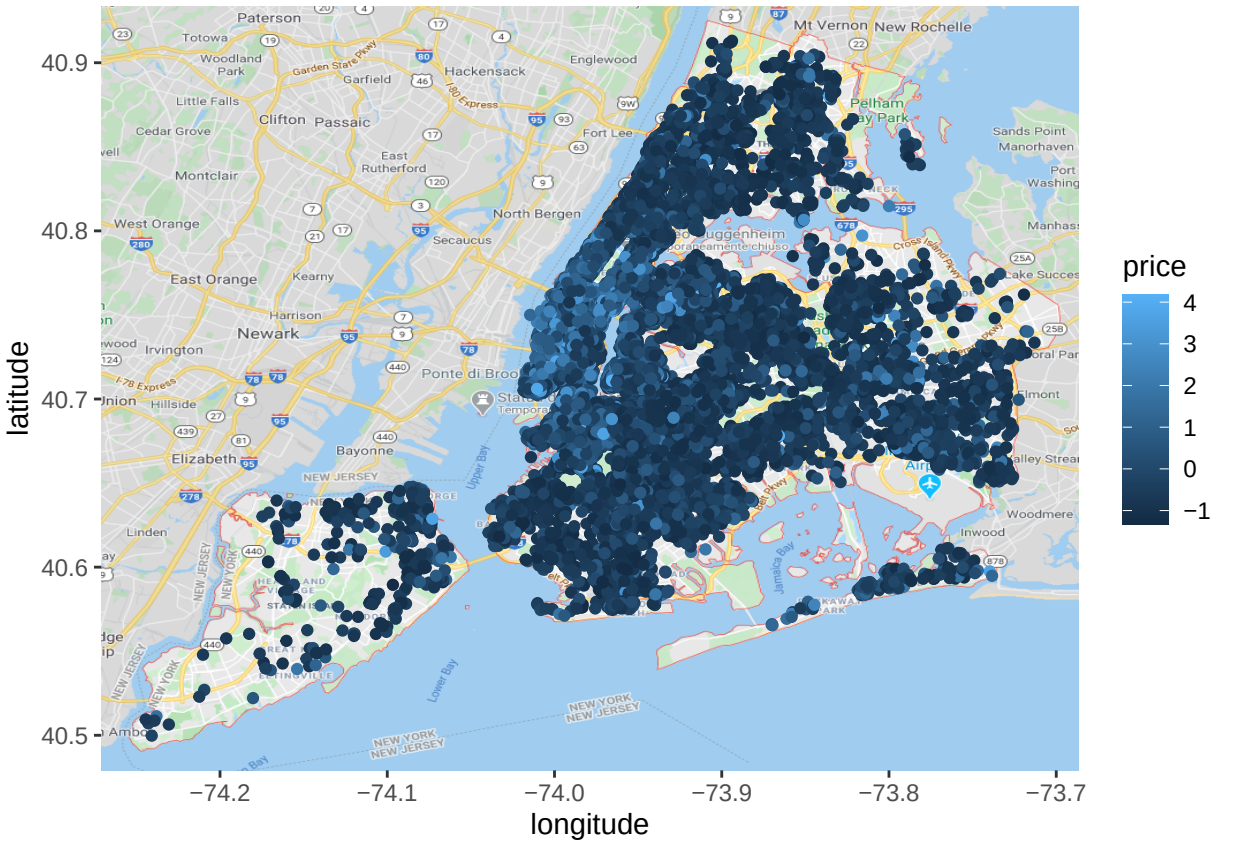
```
##   neighbourhood_group latitude longitude room_type      price
## 1                   1 40.64749 -73.97237         1  0.1973858
## 2                   2 40.75362 -73.98377         2  1.0607252
## 3                   2 40.80902 -73.94190         1  0.2087456
## 4                   1 40.68514 -73.95976         2 -0.4841979
## 5                   2 40.79851 -73.94399         2 -0.5864354
## 6                   2 40.74767 -73.97500         2  0.7767320
```

```
summary(dataset)
```

```
##  neighbourhood_group    latitude       longitude      room_type
##  1:19858             Min.   :40.50   Min.   :-74.24   1:22167
##  2:20877             1st Qu.:40.69   1st Qu.:-73.98   2:24503
##  3: 5632             Median :40.72   Median :-73.96   3: 1145
##  4:  366             Mean   :40.73   Mean   :-73.95
##  5: 1082             3rd Qu.:40.76   3rd Qu.:-73.94
##                      Max.   :40.91   Max.   :-73.71
##      price
##  Min.   :-1.3248
##  1st Qu.:-0.7228
##  Median :-0.3479
##  Mean   : 0.0000
##  3rd Qu.: 0.4587
##  Max.   : 4.1847
```

Data visualisation after the scaling

```
mappa = ggplot() + background_image(img)+ geom_point(data = dataset,  aes(y=latitude,x = longitude, col
mappa
```



Data split

Split data in subsets for each neighbourhood_group and room_type

```
neighbourhoods = unique(dataset$neighbourhood_group)

rooms = unique(dataset$room_type)

clust_data = vector("list")

lis_n = vector("list")

for (n in neighbourhoods)
{
  tmp = dataset %>%filter( neighbourhood_group == n)
  lis_n[[n]] = tmp[-1]

  tmp2 = data %>%filter( neighbourhood_group == n)
  clust_data[[n]] = tmp2[-1]
 # print(clust_data[[n]]$room_type )
  clust_data[[n]]$room_type = factor(clust_data[[n]]$room_type , level = unique(clust_data[[n]]$room_ty
}
```

```
lis_r_n= vector("list")
for (n in neighbourhoods)
{
  for(r in rooms)
  {
    tmp = dataset %>%
      filter( room_type == r  & neighbourhood_group == n)
    lis_r_n[[paste0("n",n,"-","r",r)]]= tmp[-1][-3]


    tmp2 = data %>%
      filter( room_type == r  & neighbourhood_group == n)
    clust_data[[paste0("n",n,"-","r",r)]]= tmp2[-1][-3]

  }
}


data$neighbourhood_group = factor(data$neighbourhood_group  , level = unique(data$neighbourhood_group ) 
data$room_type = factor(data$room_type  , level = unique(data$room_type ) , labels= unique(ds$room_type

clust_data[["all"]] = data
```

SPlit in train and test for each subset

```
trains = vector("list")
tests = vector("list")
datas = vector("list")

for (i in names(lis_n))
{
  sample = sample.split(lis_n[[i]], SplitRatio = .75)
  train = subset(lis_n[[i]], sample == TRUE)
  test  = subset(lis_n[[i]], sample == FALSE)
  trains[[i]]=  train
  tests[[i]] = test
  datas[[i]] = lis_n[[i]]
}

for (i in names(lis_r_n))
{
  sample = sample.split(lis_r_n[[i]], SplitRatio = .75)
  train = subset(lis_r_n[[i]], sample == TRUE)
  test  = subset(lis_r_n[[i]], sample == FALSE)
  trains[[i]]=  train
  tests[[i]] = test
  datas[[i]] = lis_r_n[[i]]

}

sample = sample.split(dataset, SplitRatio = .75)
train = subset(dataset, sample == TRUE)
test  = subset(dataset, sample == FALSE)
```

```
trains[["all"]] = train
tests[["all"]] = test
datas[["all"]] = dataset
```

MODELS TRAIN

```
model_lis = vector("list")
```

LINEAR REGRESSION

```
lin_reg = vector("list")

for (sub in names(trains))
{
  lin_reg[[sub]]$fit =lm.fit = lm(price~., data = trains[[sub]])
  lin_reg[[sub]]$summary = summary(lm.fit)

  lin_reg[[sub]]$pred  = pr.lm = predict(lm.fit,tests[[sub]])

  lin_reg[[sub]]$MSE = sum((pr.lm - tests[[sub]]$price)^2)/nrow(tests[[sub]])
  #lin_reg[[sub]]$plt=   plot(lm.fit)
}

lin_reg$name = "Linear Regression"
model_lis$linear_regression=  lin_reg
```

DECISION TREE

```
dec_tree = vector("list")

for (sub in names(trains))
{
  dec_tree[[sub]]$fit = tree_res=tree(price~., data = trains[[sub]])
  dec_tree[[sub]]$summary = summary(tree_res)

  dec_tree[[sub]]$pred  = pred = predict(tree_res,tests[[sub]])

  dec_tree[[sub]]$MSE = sum((pred - tests[[sub]]$price)^2)/nrow(tests[[sub]])
  #dec_tree[[sub]]$plt=   plot(tree_res)+text(tree_res,pretty=0)

}

dec_tree$name = "Decision Tree"
model_lis$decision_tree = dec_tree
```

RANDOM FOREST

```
rf = vector("list")

for (sub in names(trains))
```

```
{
  rf[[sub]]$fit = res =  randomForest(  price ~ . , data=trains[[sub]])

  rf[[sub]]$pred = predt = predict(res,tests[[sub]])

  rf[[sub]]$MSE = sum((predt - tests[[sub]]$price)^2)/nrow(tests[[sub]])
}
```

```
## Warning in randomForest.default(m, y, ...): The response has five or fewer
## unique values. Are you sure you want to do regression?
```

```
rf$name = "Random Forest"
model_lis$random_forest = rf
```

RANGER RANDOM FOREST

```
ranger_rf = vector("list")

for (sub in names(trains))
{

  ranger_rf[[sub]]$fit = res = ranger( price~ ., data = trains[[sub]], write.forest = TRUE, classificati

  ranger_rf[[sub]]$pred = predt = predict(res,tests[[sub]])
  ranger_rf[[sub]]$MSE = sum((predt$predictions - tests[[sub]]$price)^2)/nrow(tests[[sub]])




}

ranger_rf$name = "Ranger Random Forest"
model_lis$ranger = ranger_rf
```

NEURAL NETWORKS

```
build_model <- function(dimension) {

  model <- keras::keras_model_sequential() %>%
    layer_dense(units = 32, activation = "relu",
                input_shape = dimension) %>%
    layer_dense(units = 16, activation = "relu") %>%
    layer_dense(units = 1,activation="linear")

  model %>% compile(
    loss = "mse",
    optimizer = optimizer_rmsprop(),
    metrics = list("mean_absolute_error")
  )

  return(model)
}
```

```
print_dot_callback <- callback_lambda(
  on_epoch_end = function(epoch, logs) {
    if (epoch %% 1 == 0)
      cat(".")
  }
)


nn = vector("list")

for (sub in names(trains))
{
  d = trains[[sub]]
  d2 = tests[[sub]]
  len = length(d)
  len2 = length(d2)

  if(!is.null(d$room_type))
  {
    d$room_type = keras::to_categorical(d$room_type)
    d2$room_type = keras::to_categorical(d2$room_type)

  }

  if(!is.null(d$neighbourhood_group))
  {
    d$neighbourhood_group = keras::to_categorical(d$neighbourhood_group)
    d2$neighbourhood_group = keras::to_categorical(d2$neighbourhood_group)
  }


  target = as.vector(d$price)
  features = as.matrix(as_tibble(d[-len]))

  target_test = as.vector(d2$price)
  features_test = as.matrix(as_tibble(d2[-len]))


  nn[[sub]]$epochs = epochs =  30

  nn[[sub]]$model = model =  build_model(dim(features)[2])

  nn[[sub]]$summary = model %>% summary()
  nn[[sub]]$history <- model %>% fit(
    x = features,
    y = target,
    epochs = epochs,
    validation_split = 0.2,
    verbose = 0,
    callbacks = list(print_dot_callback)
  )
  eva = model %>% evaluate(features_test,target_test, verbose = 0)
```

```
  nn[[sub]]$mae = eva[1]
  nn[[sub]]$loss = eva[2]

  nn[[sub]]$pred = pred = model %>% predict(features_test)
  nn[[sub]]$MSE = sum((pred - target_test)^2)/length(target_test)

}
```

```
## Model: "sequential"
## _________________________________________________________________________
## Layer (type)                 Output Shape              Param #
## =========================================================================
## dense (Dense)                (None, 32)                224
## _________________________________________________________________________
## dense_1 (Dense)              (None, 16)                528
## _________________________________________________________________________
## dense_2 (Dense)              (None, 1)                 17
## =========================================================================
## Total params: 769
## Trainable params: 769
## Non-trainable params: 0
## _________________________________________________________________________
## ............................Model: "sequential_1"
## _________________________________________________________________________
## Layer (type)                 Output Shape              Param #
## =========================================================================
## dense_3 (Dense)              (None, 32)                224
## _________________________________________________________________________
## dense_4 (Dense)              (None, 16)                528
## _________________________________________________________________________
## dense_5 (Dense)              (None, 1)                 17
## =========================================================================
## Total params: 769
## Trainable params: 769
## Non-trainable params: 0
## _________________________________________________________________________
## ............................Model: "sequential_2"
## _________________________________________________________________________
## Layer (type)                 Output Shape              Param #
## =========================================================================
## dense_6 (Dense)              (None, 32)                224
## _________________________________________________________________________
## dense_7 (Dense)              (None, 16)                528
## _________________________________________________________________________
## dense_8 (Dense)              (None, 1)                 17
## =========================================================================
## Total params: 769
## Trainable params: 769
## Non-trainable params: 0
## _________________________________________________________________________
## ............................Model: "sequential_3"
## _________________________________________________________________________
## Layer (type)                 Output Shape              Param #
## =========================================================================
```

```
## dense_9 (Dense)                     (None, 32)                    224
## _________________________________________________________________________
## dense_10 (Dense)                    (None, 16)                    528
## _________________________________________________________________________
## dense_11 (Dense)                    (None, 1)                     17
## =========================================================================
## Total params: 769
## Trainable params: 769
## Non-trainable params: 0
## _________________________________________________________________________
## ............................Model: "sequential_4"
## _________________________________________________________________________
## Layer (type)                        Output Shape                  Param #
## =========================================================================
## dense_12 (Dense)                    (None, 32)                    224
## _________________________________________________________________________
## dense_13 (Dense)                    (None, 16)                    528
## _________________________________________________________________________
## dense_14 (Dense)                    (None, 1)                     17
## =========================================================================
## Total params: 769
## Trainable params: 769
## Non-trainable params: 0
## _________________________________________________________________________
## ............................Model: "sequential_5"
## _________________________________________________________________________
## Layer (type)                        Output Shape                  Param #
## =========================================================================
## dense_15 (Dense)                    (None, 32)                    96
## _________________________________________________________________________
## dense_16 (Dense)                    (None, 16)                    528
## _________________________________________________________________________
## dense_17 (Dense)                    (None, 1)                     17
## =========================================================================
## Total params: 641
## Trainable params: 641
## Non-trainable params: 0
## _________________________________________________________________________
## ............................Model: "sequential_6"
## _________________________________________________________________________
## Layer (type)                        Output Shape                  Param #
## =========================================================================
## dense_18 (Dense)                    (None, 32)                    96
## _________________________________________________________________________
## dense_19 (Dense)                    (None, 16)                    528
## _________________________________________________________________________
## dense_20 (Dense)                    (None, 1)                     17
## =========================================================================
## Total params: 641
## Trainable params: 641
## Non-trainable params: 0
## _________________________________________________________________________
## ............................Model: "sequential_7"
## _________________________________________________________________________
```

```
## Layer (type)                 Output Shape              Param #
## =================================================================
## dense_21 (Dense)             (None, 32)                96
## _________________________________________________________________
## dense_22 (Dense)             (None, 16)                528
## _________________________________________________________________
## dense_23 (Dense)             (None, 1)                 17
## =================================================================
## Total params: 641
## Trainable params: 641
## Non-trainable params: 0
## _________________________________________________________________
## ............................Model: "sequential_8"
## _________________________________________________________________
## Layer (type)                 Output Shape              Param #
## =================================================================
## dense_24 (Dense)             (None, 32)                96
## _________________________________________________________________
## dense_25 (Dense)             (None, 16)                528
## _________________________________________________________________
## dense_26 (Dense)             (None, 1)                 17
## =================================================================
## Total params: 641
## Trainable params: 641
## Non-trainable params: 0
## _________________________________________________________________
## ............................Model: "sequential_9"
## _________________________________________________________________
## Layer (type)                 Output Shape              Param #
## =================================================================
## dense_27 (Dense)             (None, 32)                96
## _________________________________________________________________
## dense_28 (Dense)             (None, 16)                528
## _________________________________________________________________
## dense_29 (Dense)             (None, 1)                 17
## =================================================================
## Total params: 641
## Trainable params: 641
## Non-trainable params: 0
## _________________________________________________________________
## ............................Model: "sequential_10"
## _________________________________________________________________
## Layer (type)                 Output Shape              Param #
## =================================================================
## dense_30 (Dense)             (None, 32)                96
## _________________________________________________________________
## dense_31 (Dense)             (None, 16)                528
## _________________________________________________________________
## dense_32 (Dense)             (None, 1)                 17
## =================================================================
## Total params: 641
## Trainable params: 641
## Non-trainable params: 0
## _________________________________________________________________
```

```
## .............................Model: "sequential_11"
## _________________________________________________________________
## Layer (type)                 Output Shape              Param #
## =================================================================
## dense_33 (Dense)             (None, 32)                96
## _________________________________________________________________
## dense_34 (Dense)             (None, 16)                528
## _________________________________________________________________
## dense_35 (Dense)             (None, 1)                 17
## =================================================================
## Total params: 641
## Trainable params: 641
## Non-trainable params: 0
## _________________________________________________________________
## .............................Model: "sequential_12"
## _________________________________________________________________
## Layer (type)                 Output Shape              Param #
## =================================================================
## dense_36 (Dense)             (None, 32)                96
## _________________________________________________________________
## dense_37 (Dense)             (None, 16)                528
## _________________________________________________________________
## dense_38 (Dense)             (None, 1)                 17
## =================================================================
## Total params: 641
## Trainable params: 641
## Non-trainable params: 0
## _________________________________________________________________
## .............................Model: "sequential_13"
## _________________________________________________________________
## Layer (type)                 Output Shape              Param #
## =================================================================
## dense_39 (Dense)             (None, 32)                96
## _________________________________________________________________
## dense_40 (Dense)             (None, 16)                528
## _________________________________________________________________
## dense_41 (Dense)             (None, 1)                 17
## =================================================================
## Total params: 641
## Trainable params: 641
## Non-trainable params: 0
## _________________________________________________________________
## .............................Model: "sequential_14"
## _________________________________________________________________
## Layer (type)                 Output Shape              Param #
## =================================================================
## dense_42 (Dense)             (None, 32)                96
## _________________________________________________________________
## dense_43 (Dense)             (None, 16)                528
## _________________________________________________________________
## dense_44 (Dense)             (None, 1)                 17
## =================================================================
## Total params: 641
## Trainable params: 641
```

```
## Non-trainable params: 0
## _________________________________________________________________
## .........................Model: "sequential_15"
## _________________________________________________________________
## Layer (type)                 Output Shape              Param #
## =================================================================
## dense_45 (Dense)             (None, 32)                96
## _________________________________________________________________
## dense_46 (Dense)             (None, 16)                528
## _________________________________________________________________
## dense_47 (Dense)             (None, 1)                 17
## =================================================================
## Total params: 641
## Trainable params: 641
## Non-trainable params: 0
## _________________________________________________________________
## .........................Model: "sequential_16"
## _________________________________________________________________
## Layer (type)                 Output Shape              Param #
## =================================================================
## dense_48 (Dense)             (None, 32)                96
## _________________________________________________________________
## dense_49 (Dense)             (None, 16)                528
## _________________________________________________________________
## dense_50 (Dense)             (None, 1)                 17
## =================================================================
## Total params: 641
## Trainable params: 641
## Non-trainable params: 0
## _________________________________________________________________
## .........................Model: "sequential_17"
## _________________________________________________________________
## Layer (type)                 Output Shape              Param #
## =================================================================
## dense_51 (Dense)             (None, 32)                96
## _________________________________________________________________
## dense_52 (Dense)             (None, 16)                528
## _________________________________________________________________
## dense_53 (Dense)             (None, 1)                 17
## =================================================================
## Total params: 641
## Trainable params: 641
## Non-trainable params: 0
## _________________________________________________________________
## .........................Model: "sequential_18"
## _________________________________________________________________
## Layer (type)                 Output Shape              Param #
## =================================================================
## dense_54 (Dense)             (None, 32)                96
## _________________________________________________________________
## dense_55 (Dense)             (None, 16)                528
## _________________________________________________________________
## dense_56 (Dense)             (None, 1)                 17
## =================================================================
```

```
## Total params: 641
## Trainable params: 641
## Non-trainable params: 0
## _________________________________________________________________
## ..........................Model: "sequential_19"
## _________________________________________________________________
## Layer (type)                 Output Shape              Param #
## =================================================================
## dense_57 (Dense)             (None, 32)                96
## _________________________________________________________________
## dense_58 (Dense)             (None, 16)                528
## _________________________________________________________________
## dense_59 (Dense)             (None, 1)                 17
## =================================================================
## Total params: 641
## Trainable params: 641
## Non-trainable params: 0
## _________________________________________________________________
## ..........................Model: "sequential_20"
## _________________________________________________________________
## Layer (type)                 Output Shape              Param #
## =================================================================
## dense_60 (Dense)             (None, 32)                416
## _________________________________________________________________
## dense_61 (Dense)             (None, 16)                528
## _________________________________________________________________
## dense_62 (Dense)             (None, 1)                 17
## =================================================================
## Total params: 961
## Trainable params: 961
## Non-trainable params: 0
## _________________________________________________________________
## ............................
```

```
nn$name = "Neural Networks"
model_lis$neural_networks = nn
```

Comparison between the models

```
concat = c()

for (n in unique(ds$neighbourhood_group))
{
  concat = c(concat,n)
}
for (n in unique(ds$neighbourhood_group))
{
  for(r in unique(ds$room_type))
  {
    concat = c(concat, paste0(n,"/",r))

  }
}
```

```
concat = c(concat,"All")

n = names(nn)



cols = vector("list")
cols[["Subset"]] = concat
for( m in model_lis)
{
  col = c()
  for( nam in n)
  {

    if( nam != "name")
    {
      col = c(col,m[[nam]]$MSE)

    }

  }
  cols[[m$name]] = col

}
mse_df = as.data.frame(cols)
formattable(mse_df)
```

Subset

Linear.Regression

Decision.Tree

Random.Forest

Ranger.Random.Forest

Neural.Networks

Brooklyn

0.47479223

0.47887198

0.45421326

0.45179894

0.51721578

Manhattan

0.77767609

0.78511656

0.73475525

0.73767252

0.89479765

Queens

0.38466653

0.38645428

0.35873769

0.36031453

0.38615614

Staten Island

0.49181862

0.48483713

0.45351507

0.45254907

0.61070680

Bronx

0.34294047

0.34207516

0.32907841

0.33123837

1.70587840

Brooklyn/Private room

0.18581380

0.18061163

0.19337990

0.19377044

0.34404182

Brooklyn/Entire home/apt

0.75987316

0.75033084

0.80584461

0.80405287

0.81320227

Brooklyn/Shared room

0.27729515

0.30432942

0.25182944

0.25153853

0.29992881

Manhattan/Private room

0.44456114

0.37531131

0.37897160

0.38065764

0.49837404

Manhattan/Entire home/apt

1.01147065

1.01744901

1.05385997

1.05263409

1.11773552

Manhattan/Shared room

0.71469919

0.73805775

0.74621468

0.74026531

0.74561266

Queens/Private room

0.16162900

0.15515885

0.15180124

0.15189433

0.17062233

Queens/Entire home/apt

0.68463027

0.65912884

0.65756147

0.65368981

0.71756037

Queens/Shared room

0.07098971

0.29897867

0.23001859

0.21842252

0.11834837

Staten Island/Private room

0.26401002

0.23393266

0.22679115

0.23006554

0.25781255

Staten Island/Entire home/apt

0.43196942

0.43680752

0.49157730

0.47372337

1.06202156

Staten Island/Shared room

0.11991315

0.22418801

0.08410882

0.22060135

0.34163296

Bronx/Private room

0.15183296

0.20281265

0.18504129

0.18393881

0.16079765

Bronx/Entire home/apt

0.82524505

0.88600404

0.87054705

0.87293908

0.91599822

Bronx/Shared room

0.06903888

0.04892947

0.05345524

0.05415259

0.06863484

All

0.61349693

0.61042296

0.57989682

0.55205834

0.63652349

Clustering and Groups

Clustering for Mixed type of data

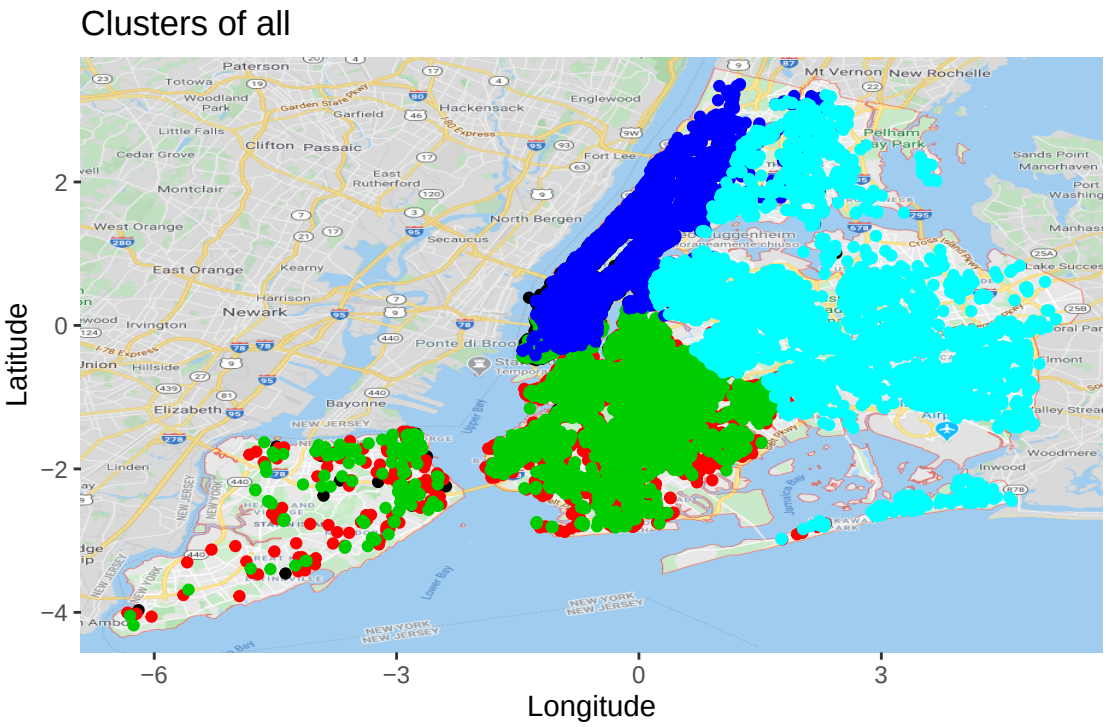
```
clust_num = 5

get_clusters = function(dts, num, dim_plot,name)
{
  l = list()
  if(is.null(dts$room_type))
  {
    l$cl = kmeans(dts,num)
  }
  else
  {
    l$cl = kproto(dts,num)

  }

  clust = list()

  for (i in 1:num)
  {
    indexes = l$cl$cluster == i
    clust[[i]] = dts[indexes,]
  }

  min_lat =  dim_plot[1]
  max_lat = dim_plot[2]
  min_long = dim_plot[3]
  max_long= dim_plot[4]


  myplot= ggplot() +  background_image(img)+  xlab('Longitude') +  ylab('Latitude')+ theme(plot.margin =


  count = 1
  l$clust_plot = vector("list")
  for(el in clust)
  {
    myplot = myplot+ geom_point(data = el, aes(y = latitude, x = longitude),color= count)

    p =ggplot()+ background_image(img)+geom_point(data = el, aes(y = latitude, x = longitude),color= cou
```

```
    l$clust_plot[[as.character(count)]] = p
    #print(paste0("=== clust: ",count,"==="))
    #print(summary(el))
    #print("==========")

    count= count+1
  }
  l$myplot = myplot
  return(l)
}

get_all_cluster = function(clust_data, clust_num, dim)
{
  lis = vector("list")
  plots = vector("list")

  cnt= 1
  for(sub in names(clust_data))
  {

    lis[[sub]] = get_clusters(clust_data[[sub]], clust_num, dim, sub)
    plots[[cnt]] = lis[[sub]]$myplot
    cnt = cnt+1

  }
  lis[["all_plots"]]= plots
  return(lis)
}

borders = c(min(clust_data$all$latitude) ,max(clust_data$all$latitude), min(clust_data$all$longitude), m
all_cluster = get_all_cluster(clust_data,5, borders)
```

```
## # NAs in variables:
##  latitude longitude room_type     price
##         0         0         0         0
## 0 observation(s) with NAs.
##
## Estimated lambda: 0.8287906
##
## # NAs in variables:
##  latitude longitude room_type     price
##         0         0         0         0
## 0 observation(s) with NAs.
##
## Estimated lambda: 1.272312
##
## # NAs in variables:
##  latitude longitude room_type     price
##         0         0         0         0
## 0 observation(s) with NAs.
##
## Estimated lambda: 1.701065
##
## # NAs in variables:
```

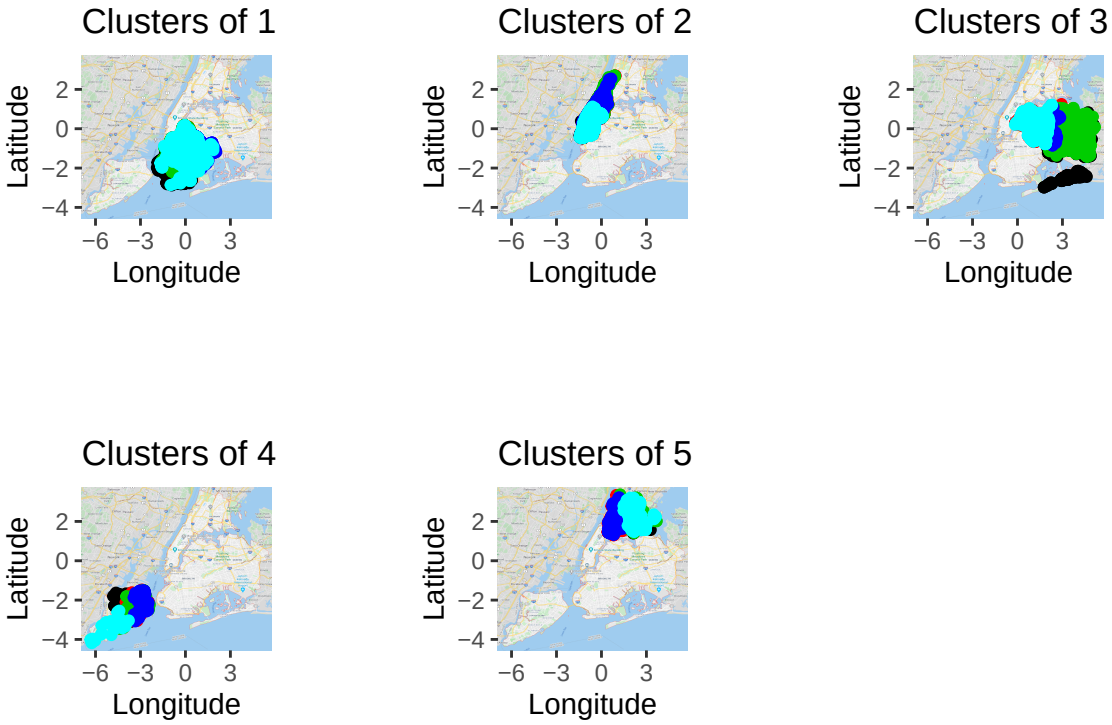
```
##  latitude longitude room_type     price
##         0         0         0         0
## 0 observation(s) with NAs.
##
## Estimated lambda: 0.935192
##
## # NAs in variables:
##  latitude longitude room_type     price
##         0         0         0         0
## 0 observation(s) with NAs.
##
## Estimated lambda: 0.7781472
##
## # NAs in variables:
## neighbourhood_group            latitude           longitude           room_type
##                   0                   0                   0                   0
##               price
##                   0
## 0 observation(s) with NAs.
##
## Estimated lambda: 1.747748
```

```
multiplots <- function(plotlist, file=NULL,cols = 2, layout = NULL) {
  require(grid)

  plots <- c(plotlist)

  numPlots = length(plots)

  if (is.null(layout)) {
    layout <- matrix(seq(1, cols * ceiling(numPlots/cols)),
                     ncol = cols, nrow = ceiling(numPlots/cols),byrow =T)
  }

  if (numPlots == 1) {
    print(plots[[1]])

  } else {
    grid.newpage()
    pushViewport(viewport(layout = grid.layout(nrow(layout), ncol(layout))))

    for (i in 1:numPlots) {
      matchidx <- as.data.frame(which(layout == i, arr.ind = TRUE))

      print(plots[[i]], vp = viewport(layout.pos.row = matchidx$row,
                                      layout.pos.col = matchidx$col))
    }
  }
}

all_cluster$all_plots[[21]]
```

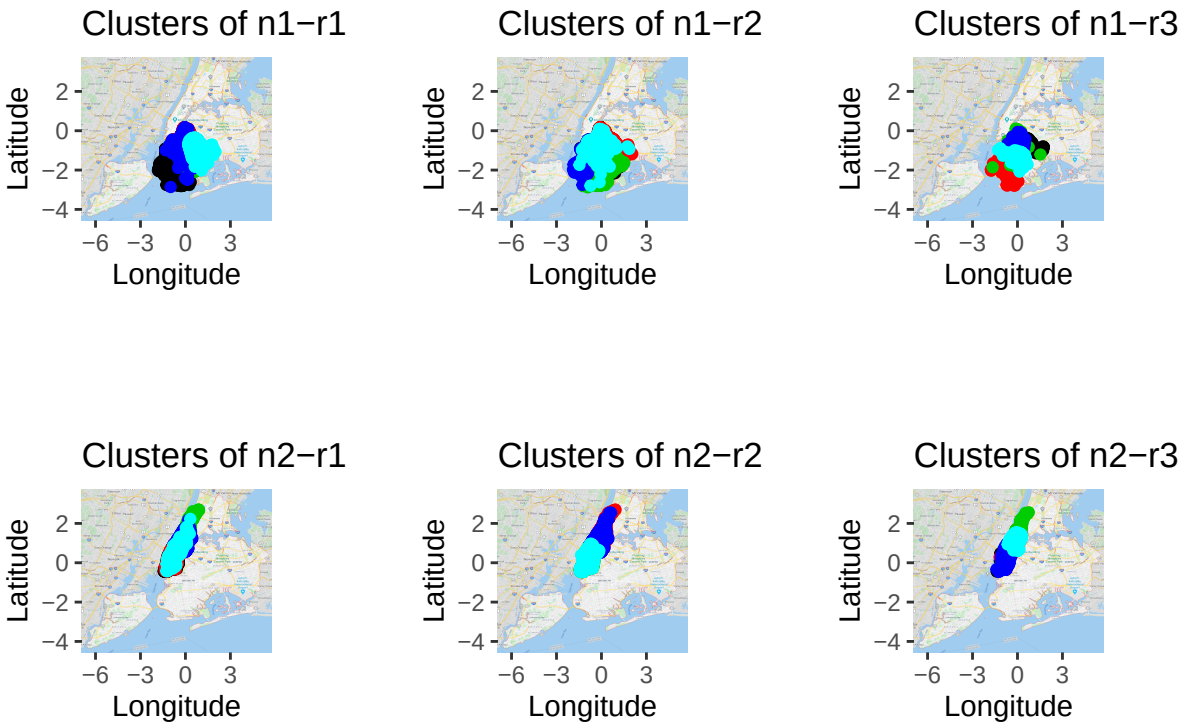



```
multiplots(all_cluster$all_plot[1:5], cols=3)
```

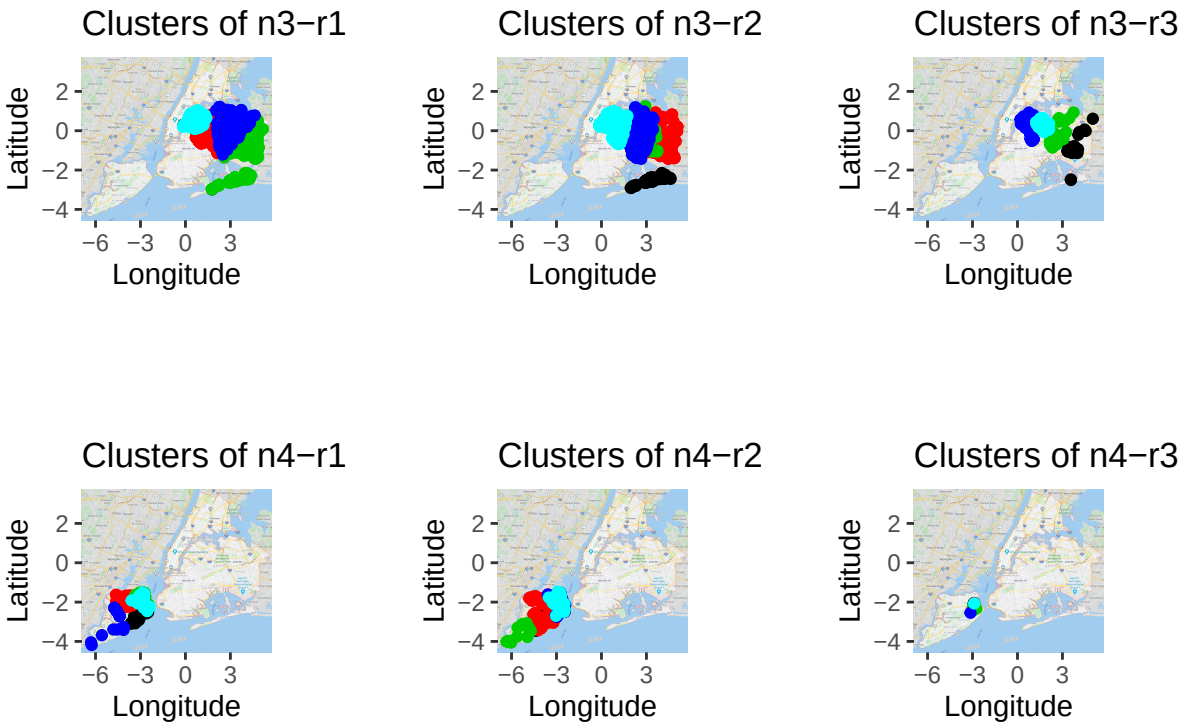
```
## Loading required package: grid
```



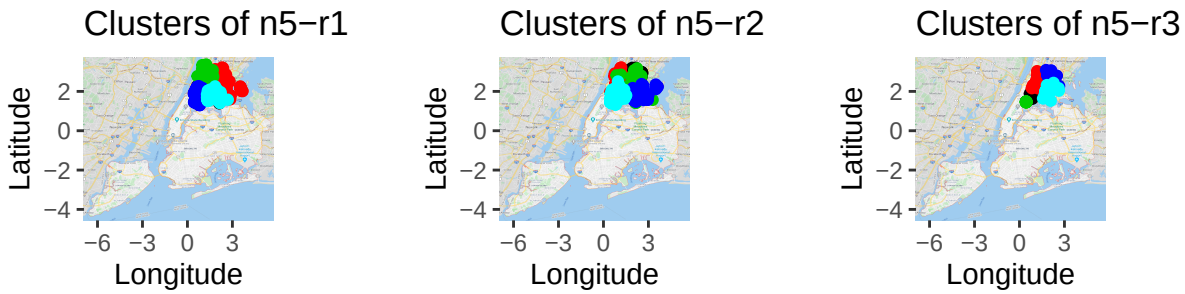

```
multiplots(all_cluster$all_plot[6:11], cols=3)
```




```
multiplots(all_cluster$all_plot[12:17], cols=3)
```




```
multiplots(all_cluster$all_plot[c(18:20,22,23)], cols=3)
```




```
## NULL
## NULL
```

PCAmixdata

```
## Split mixed dataset into quantitative and qualitative variables
## For now excluding the target variable "Churn", which will be added later as a supplementary variable
#split <- splitmix(dataset[1:5])

split = splitmix(clust_data$all)
## PCA
res.pcamix <- PCAmix(X.quanti=split$X.quanti,
                     X.quali=split$X.quali,
                     rename.level=TRUE,
                     graph=FALSE,
                     ndim=25)

res.pcamix
```

```
##
## Call:
## PCAmix(X.quanti = split$X.quanti, X.quali = split$X.quali, ndim = 25,     rename.level = TRUE, graph
##
## Method = Principal Component of mixed data (PCAmix)
##
```

```
##
## "name" "description"
## "$eig" "eigenvalues of the principal components (PC) "
## "$ind" "results for the individuals (coord,contrib,cos2)"
## "$quanti" "results for the quantitative variables (coord,contrib,cos2)"
## "$levels" "results for the levels of the qualitative variables (coord,contrib,cos2)"
## "$quali" "results for the qualitative variables (contrib,relative contrib)"
## "$sqload" "squared loadings"
## "$coef" "coef of the linear combinations defining the PC"
```

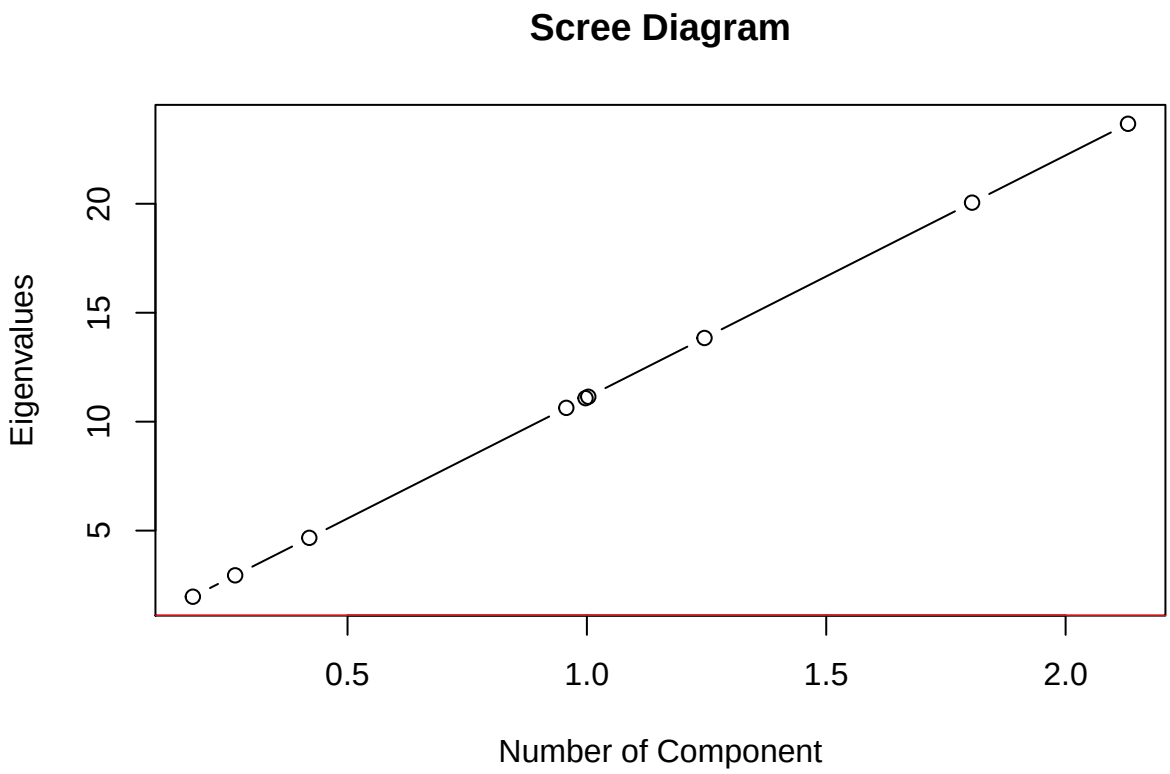
eig.val <- get_eigenvalue(res.famd) head(eig.val)

fviz_screeplot(res.famd)

```
## Inspect principal components
res.pcamix$eig
```

```
##       Eigenvalue Proportion Cumulative
## dim 1  2.1303801  23.670890   23.67089
## dim 2  1.8046238  20.051375   43.72227
## dim 3  1.2456603  13.840670   57.56294
## dim 4  1.0028761  11.143067   68.70600
## dim 5  0.9971611  11.079568   79.78557
## dim 6  0.9570021  10.633357   90.41893
## dim 7  0.4201255   4.668061   95.08699
## dim 8  0.2652652   2.947391   98.03438
## dim 9  0.1769058   1.965620  100.00000
```

```
# Use Scree Diagram to select the components:
plot(res.pcamix$eig, type="b", main="Scree Diagram", xlab="Number of Component", ylab="Eigenvalues")
abline(h=1, lwd=3, col="red")
```



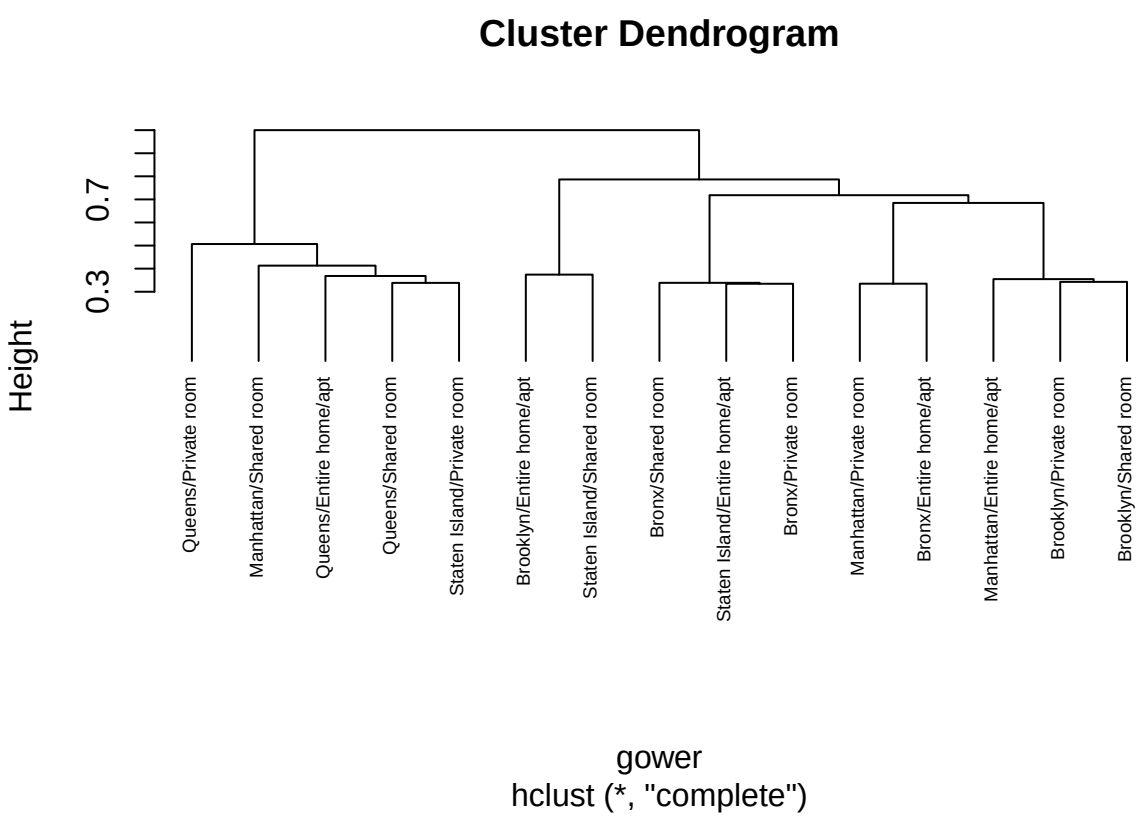
Hierarchical Cluster Analysis

```
agg = aggregate(price ~neighbourhood_group+room_type, clust_data$all , mean)


name_hc = c()
for (n1 in unique(agg$neighbourhood_group))
{
  for(n2 in unique(agg$room_type))
  {
    name_hc = c(name_hc, paste0(n1,"/",n2))
  }
}
rownames(agg) = name_hc

agg
```

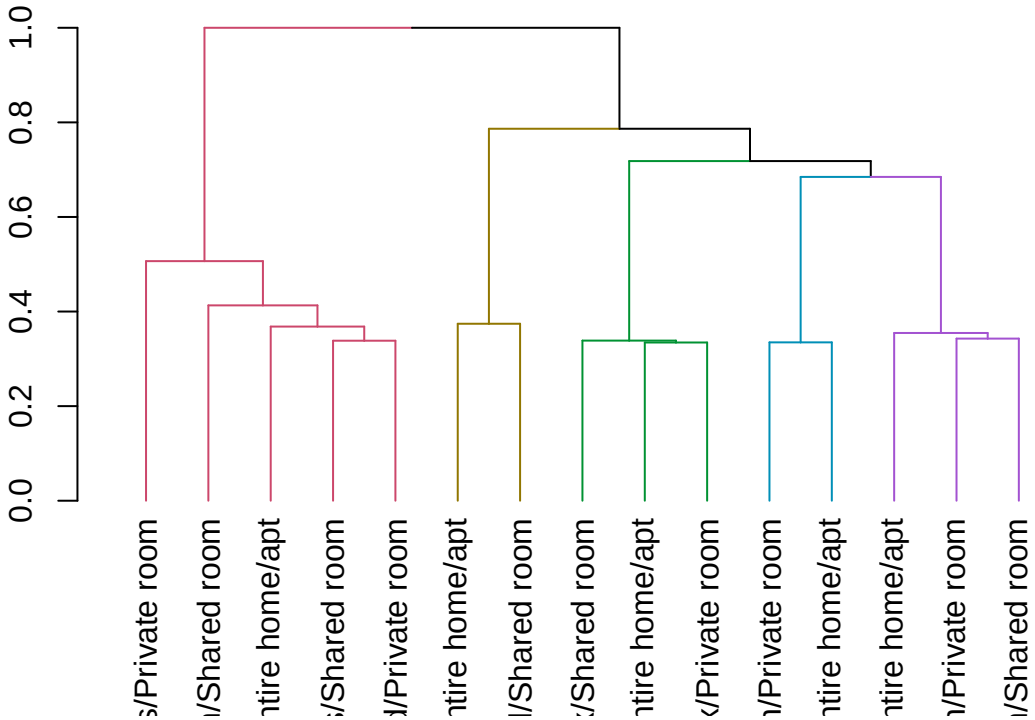
```
##                              neighbourhood_group       room_type       price
## Brooklyn/Private room                   Brooklyn    Private room -0.68331638
## Brooklyn/Entire home/apt                Manhattan    Private room -0.31844499
## Brooklyn/Shared room                      Queens    Private room -0.73368292
## Manhattan/Private room             Staten Island    Private room -0.78758723
## Manhattan/Entire home/apt                 Bronx    Private room -0.79719615
## Manhattan/Shared room                  Brooklyn Entire home/apt  0.32170945
## Queens/Private room                    Manhattan Entire home/apt  0.82148490
```

```
## Queens/Entire home/apt                    Queens Entire home/apt  0.08237678
## Queens/Shared room                  Staten Island Entire home/apt -0.07711867
## Staten Island/Private room                 Bronx Entire home/apt -0.10379804
## Staten Island/Entire home/apt           Brooklyn     Shared room -0.93711857
## Staten Island/Shared room               Manhattan     Shared room -0.53647655
## Bronx/Private room                         Queens     Shared room -0.93058224
## Bronx/Entire home/apt               Staten Island     Shared room -0.77955083
## Bronx/Shared room                          Bronx     Shared room -0.95841842
```

```r
gower <- daisy(agg, metric = "gower")
hc1 <- hclust(gower, method = "complete" )

plot(hc1, cex = 0.6, hang = -1)
```



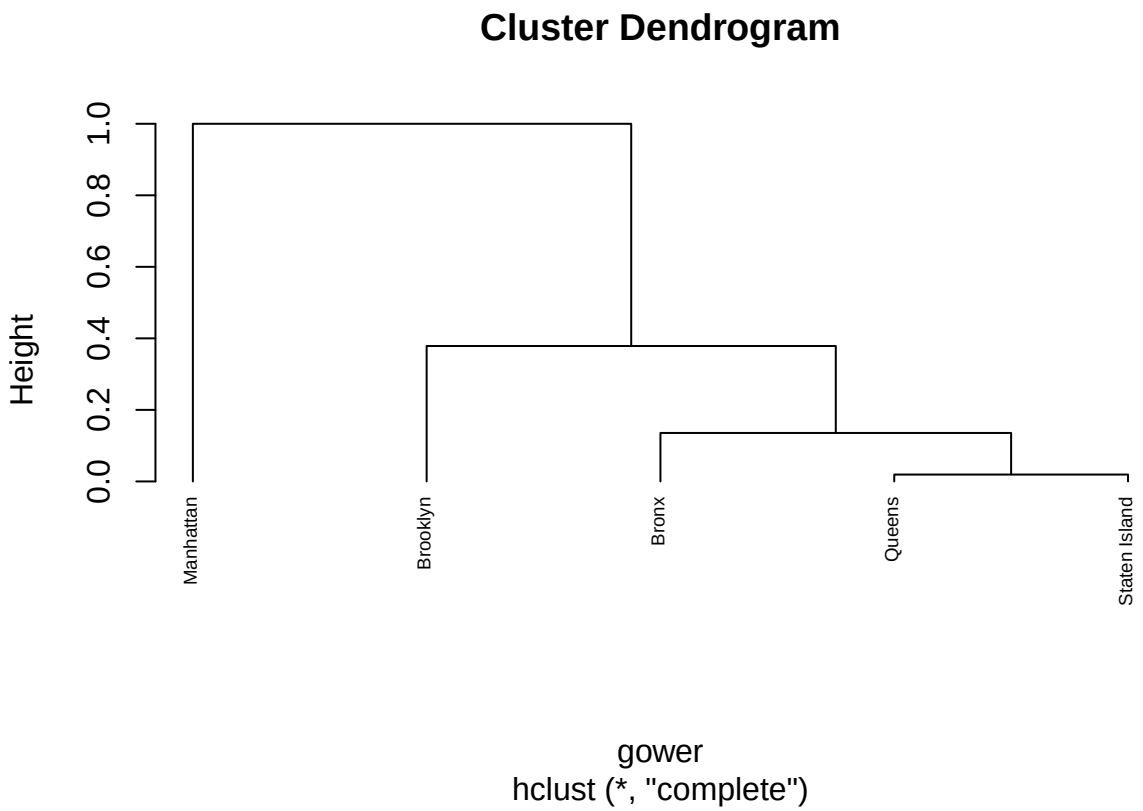

```r
avg_dend_obj <- as.dendrogram(hc1)
avg_col_dend <- color_branches(avg_dend_obj, h = 0.6)
plot(avg_col_dend, cex= 0.6)
```




```
agg = aggregate(price ~neighbourhood_group, clust_data$all , mean)
agg
```

```
##   neighbourhood_group      price
## 1            Brooklyn -0.2147398
## 2           Manhattan  0.3601591
## 3              Queens -0.4396243
## 4       Staten Island -0.4574125
## 5               Bronx -0.5650283
```

```
rownames(agg) = c("Brooklyn","Manhattan",
                  "Queens","Staten Island", "Bronx")
agg$neighbourhood_group = NULL
gower <- daisy(agg, metric = "gower")
hc1 <- hclust(gower, method = "complete" )

plot(hc1, cex = 0.6, hang = -1)
```




```
#clust <- cutree(hc1, k = 5)

#fviz_cluster(list(data = agg, cluster = clust))  ## from 'factoextra' package
```

==========

DA VEDEREE

- Factor Analysis of Mixed Data (FAMD)

http://www.sthda.com/english/articles/31-principal-component-methods-in-r-practical-guide/115-famd-factor-analysis-of-mixed-data-in-r-essentials/ https://nextjournal.com/pc-methods/calculate-pc-mixed-data https://cran.r-project.org/web/packages/FactoMineR/index.html https://stats.stackexchange.com/questions/5774/can-principal-component-analysis-be-applied-to-datasets-containing-a-mix-of-cont

```
library("FactoMineR")
library("factoextra")
```

FAMD (base, ncp = 5, sup.var = NULL, ind.sup = NULL, graph = TRUE) - base : a data frame with n rows (individuals) and p columns (variables). - ncp: the number of dimensions kept in the results (by default 5) - sup.var: a vector indicating the indexes of the supplementary variables. - ind.sup: a vector indicating the indexes of the supplementary individuals. - graph : a logical value. If TRUE a graph is displayed.

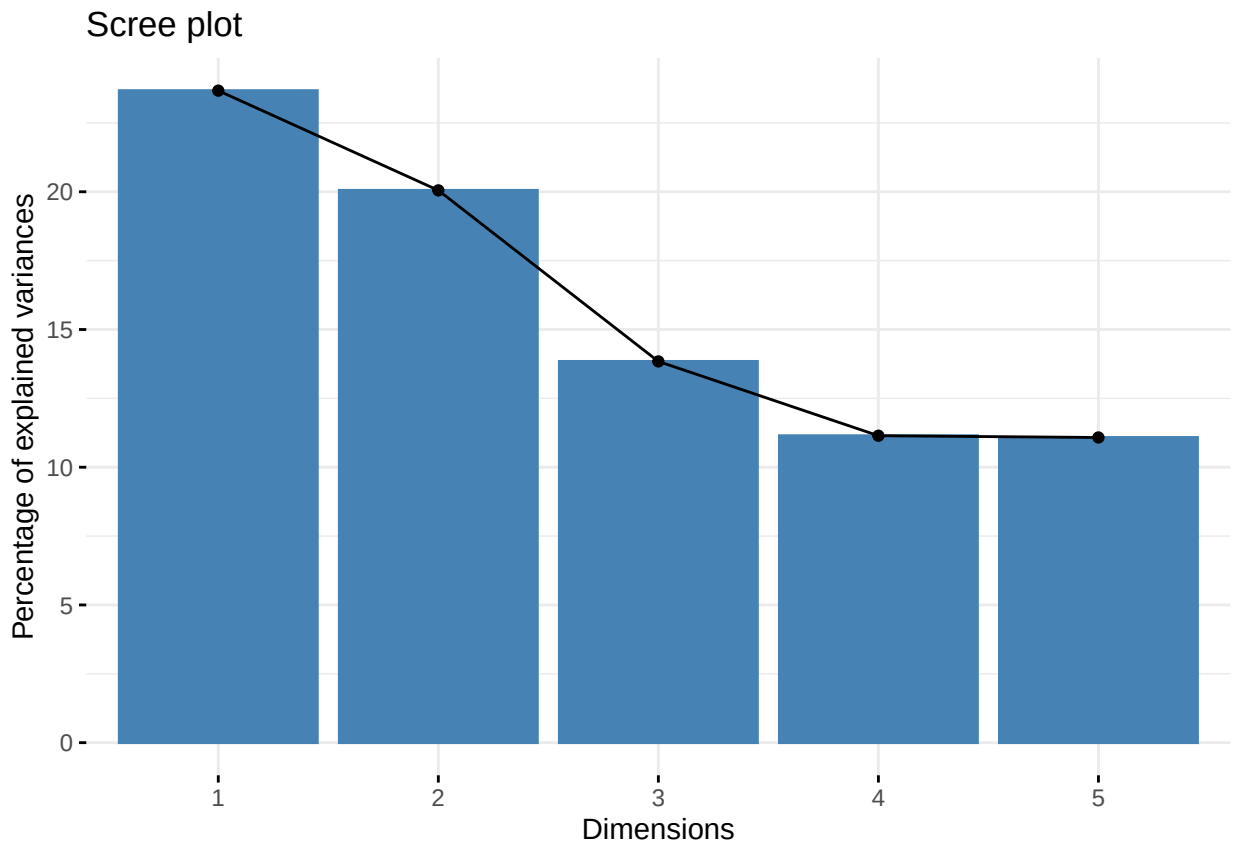
```
res.famd <- FAMD(clust_data$all, graph = F, ncp = 5)
print(res.famd)
```

```
## *The results are available in the following objects:
##
##   name          description
## 1 "$eig"        "eigenvalues and inertia"
## 2 "$var"        "Results for the variables"
## 3 "$ind"        "results for the individuals"
## 4 "$quali.var"  "Results for the qualitative variables"
## 5 "$quanti.var" "Results for the quantitative variables"
```

```
eig.val <- get_eigenvalue(res.famd)
head(eig.val)
```

```
##       eigenvalue variance.percent cumulative.variance.percent
## Dim.1  2.1303801         23.67089                     23.67089
## Dim.2  1.8046238         20.05138                     43.72227
## Dim.3  1.2456603         13.84067                     57.56294
## Dim.4  1.0028761         11.14307                     68.70600
## Dim.5  0.9971611         11.07957                     79.78557
```

```
fviz_screeplot(res.famd)
```

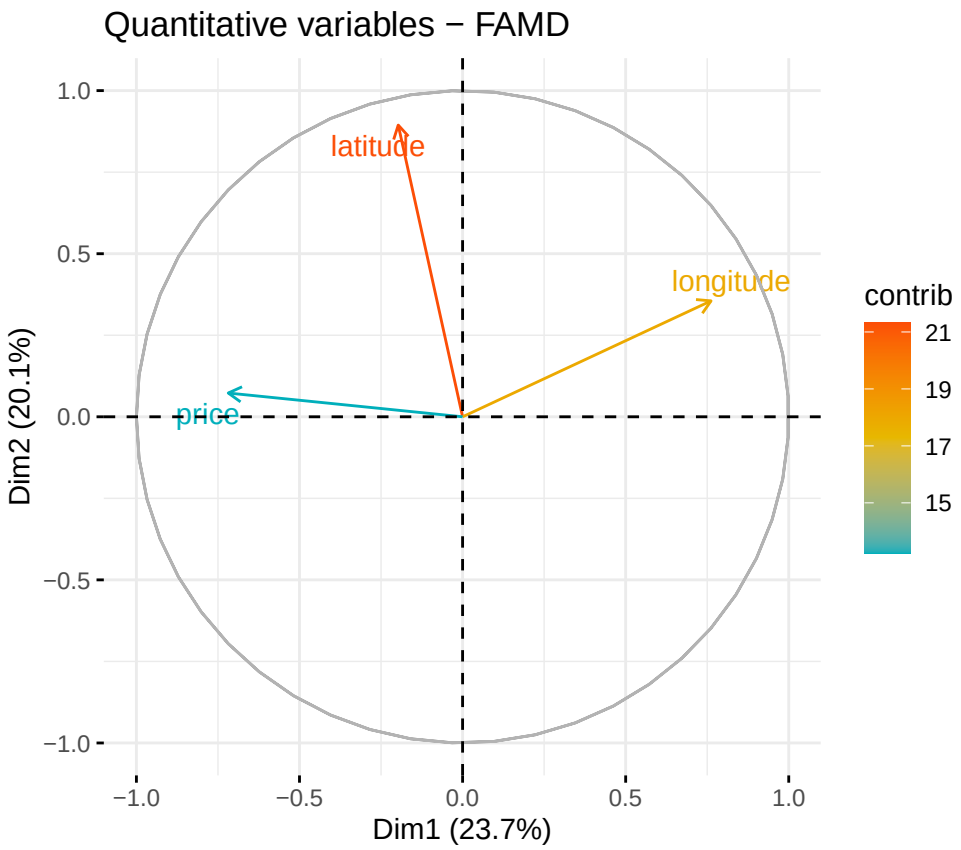



```
quanti.var <- get_famd_var(res.famd, "quanti.var")
quanti.var
```

```
## FAMD results for quantitative variables
##  ===================================================
##   Name       Description
## 1 "$coord"   "Coordinates"
```

```
## 2 "$cos2"    "Cos2, quality of representation"
## 3 "$contrib" "Contributions"
```

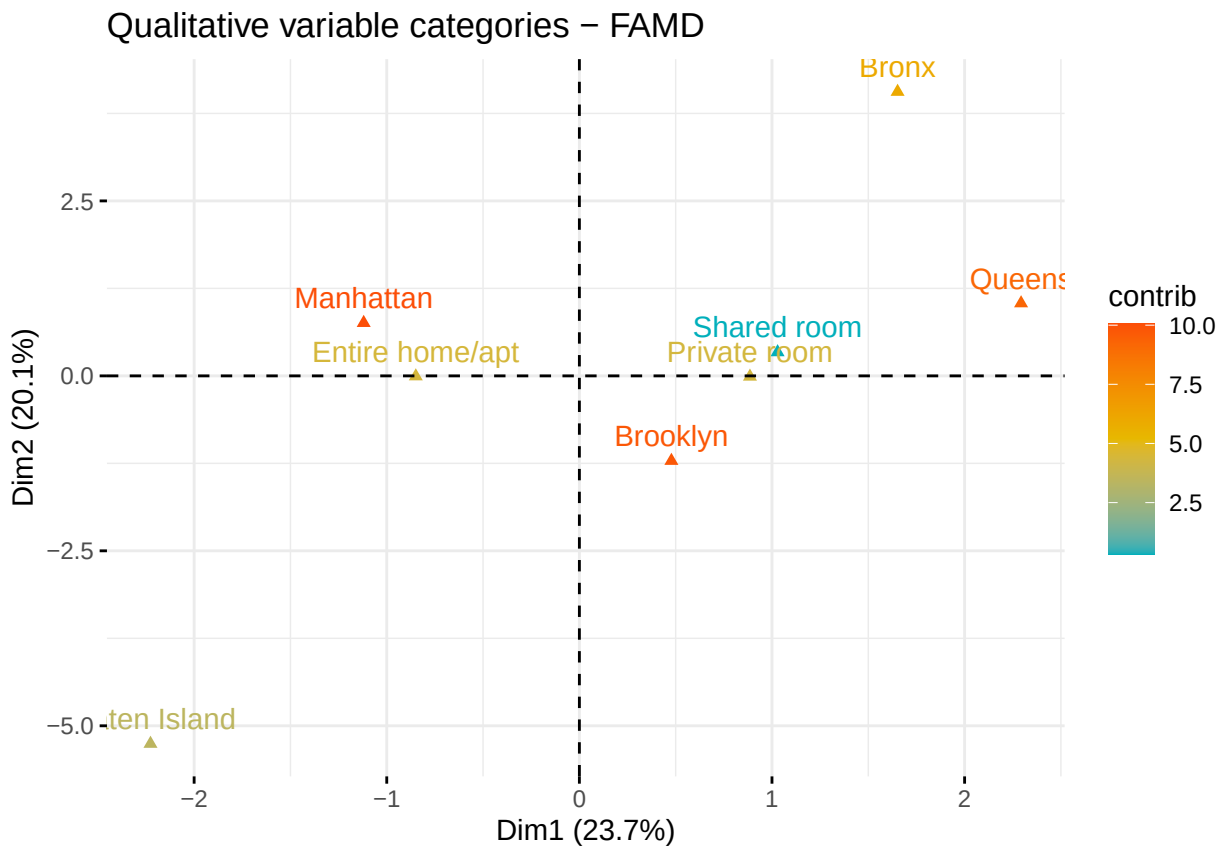
```
fviz_famd_var(res.famd, "quanti.var", col.var = "contrib",
              gradient.cols = c("#00AFBB", "#E7B800", "#FC4E07"),
              repel = TRUE)
```




```
quali.var <- get_famd_var(res.famd, "quali.var")
quali.var
```

```
## FAMD results for qualitative variable categories
##  ==================================================
##   Name       Description
## 1 "$coord"   "Coordinates"
## 2 "$cos2"    "Cos2, quality of representation"
## 3 "$contrib" "Contributions"
```

```
fviz_famd_var(res.famd, "quali.var", col.var = "contrib",
              gradient.cols = c("#00AFBB", "#E7B800", "#FC4E07")
              )
```



ind <- get_famd_ind(res.famd) ind

fviz_famd_ind(res.famd, col.ind = "cos2", gradient.cols = c("#00AFBB", "#E7B800", "#FC4E07"), repel = TRUE)

```
var <- get_famd_var(res.famd)
var
```

```
## FAMD results for variables
##  ===================================================
##   Name       Description
## 1 "$coord"   "Coordinates"
## 2 "$cos2"    "Cos2, quality of representation"
## 3 "$contrib" "Contributions"
```

```
# Coordinates of variables
head(var$coord)
```

```
##                          Dim.1       Dim.2      Dim.3        Dim.4        Dim.5
## latitude            0.03892236 0.799743946 0.04903898 0.0016811055 0.0018258497
## longitude           0.58030735 0.126293692 0.16517205 0.0001203636 0.0001685452
## price               0.51607212 0.005313173 0.23572672 0.0005079180 0.0001842764
## neighbourhood_group 0.63900709 0.871714919 0.43427934 0.5273304673 0.4806939587
## room_type           0.35607121 0.001558062 0.36144319 0.4732362202 0.5142884950
```

```
# Cos2: quality of representation on the factore map
head(var$cos2)
```

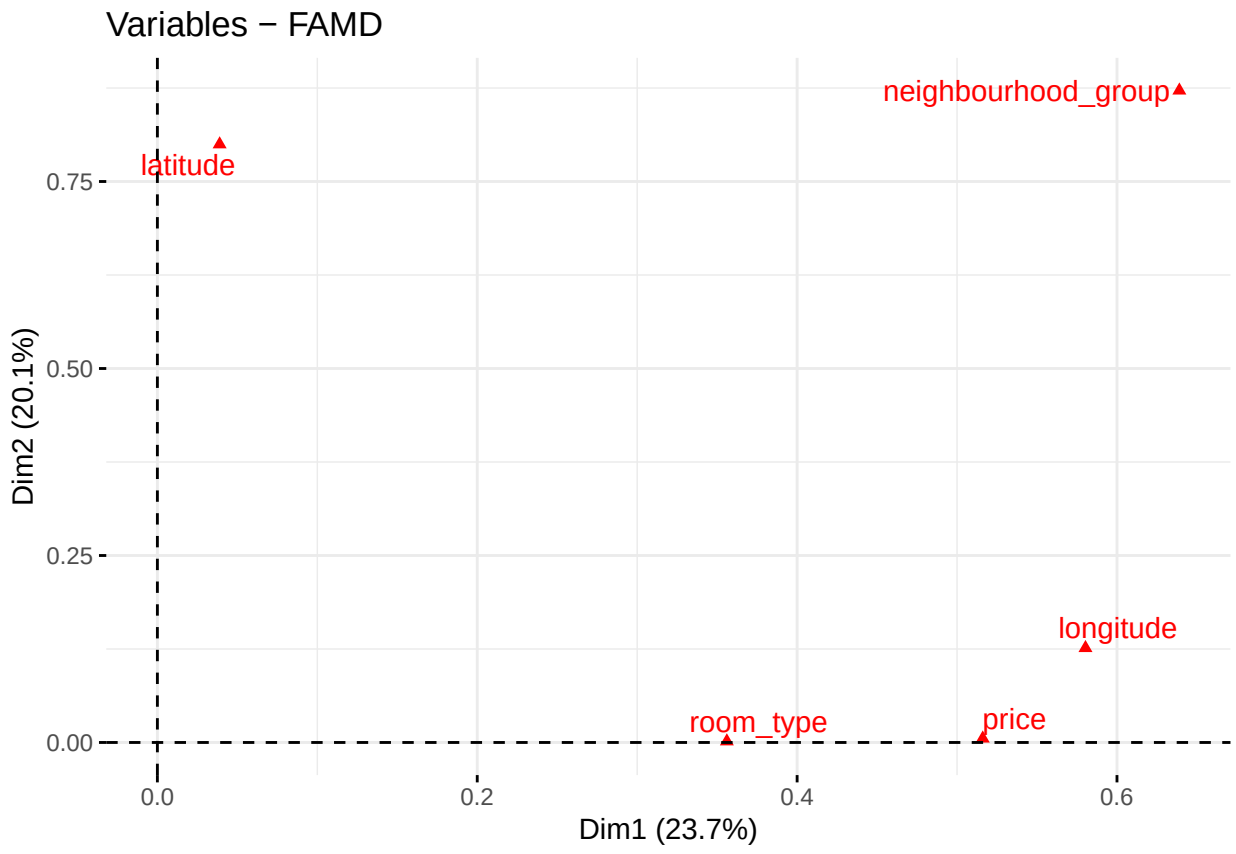
```
##                          Dim.1        Dim.2       Dim.3        Dim.4
```

```
## latitude            0.00151495 6.395904e-01 0.002404821 2.826116e-06
## longitude           0.33675663 1.595010e-02 0.027281805 1.448739e-08
## price               0.26633043 2.822981e-05 0.055567086 2.579807e-07
## neighbourhood_group 0.10208251 1.899717e-01 0.047149637 6.951936e-02
## room_type           0.06339335 1.213779e-06 0.065320588 1.119763e-01
##                            Dim.5
## latitude            3.333727e-06
## longitude           2.840749e-08
## price               3.395780e-08
## neighbourhood_group 5.776667e-02
## room_type           1.322463e-01
```

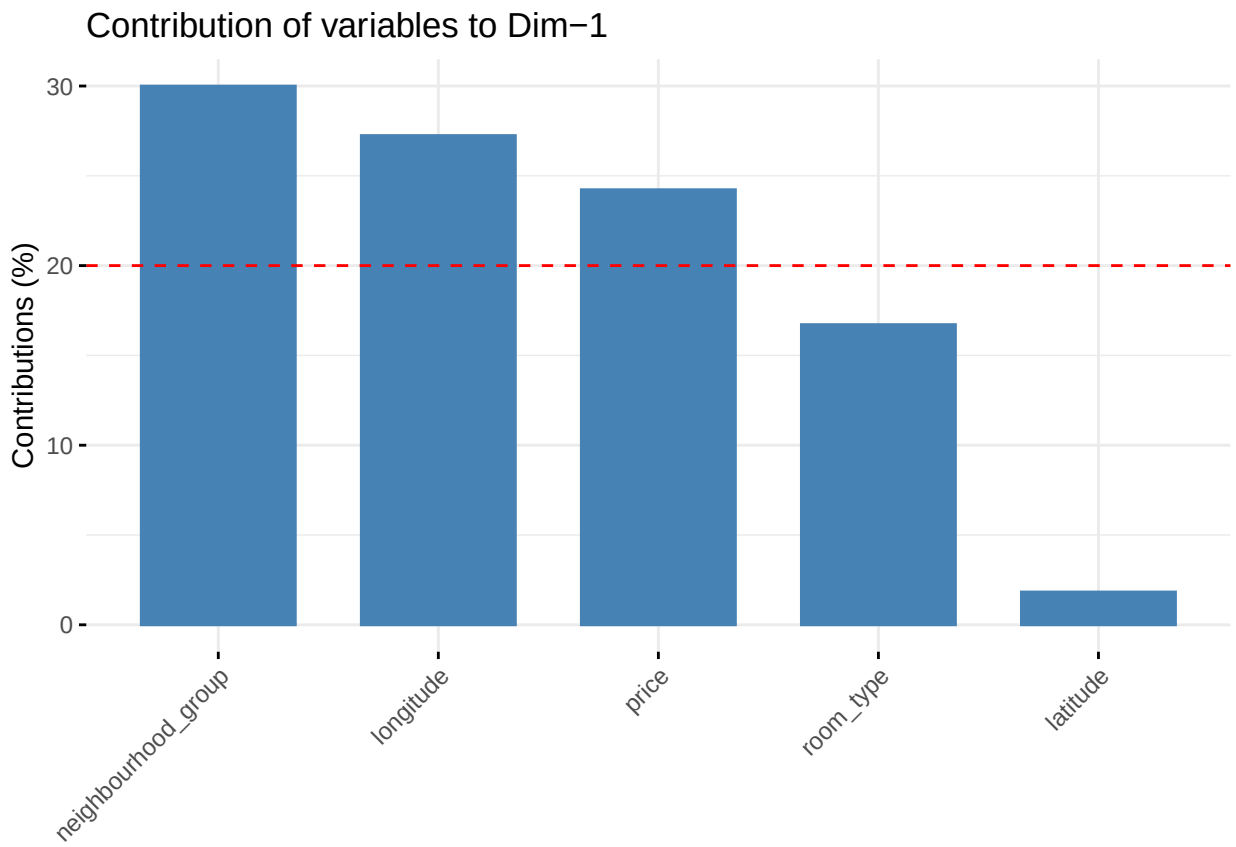
```
# Contributions to the  dimensions
head(var$contrib)
```

```
##                         Dim.1       Dim.2     Dim.3       Dim.4       Dim.5
## latitude             1.827015 44.31638047  3.936786  0.16762844  0.18310478
## longitude           27.239615  6.99833910 13.259799  0.01200184  0.01690250
## price               24.224415  0.29441998 18.923837  0.05064614  0.01848011
## neighbourhood_group 29.994980 48.30452324 34.863386 52.58181750 48.20624738
## room_type           16.713975  0.08633722 29.016193 47.18790608 51.57526523
```

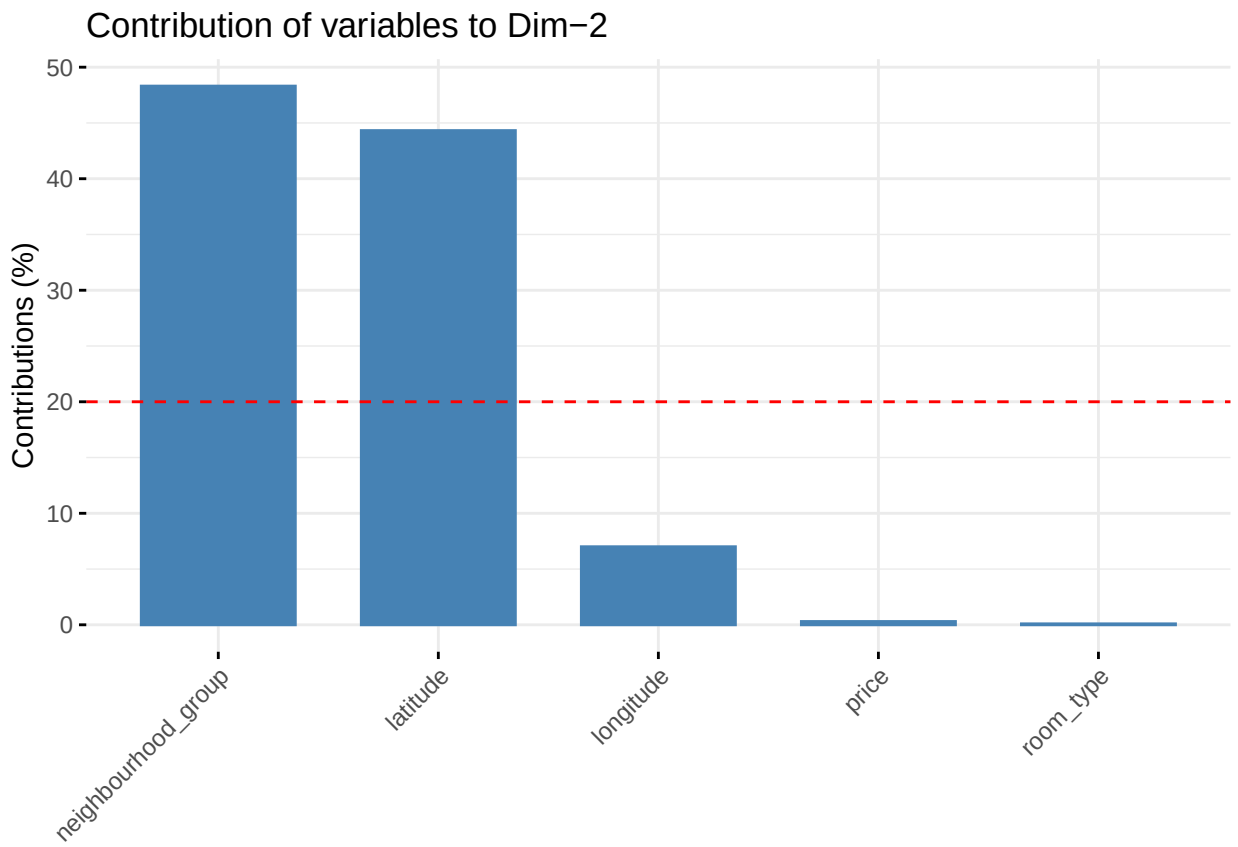
```
# Plot of variables
fviz_famd_var(res.famd, repel = TRUE)
```



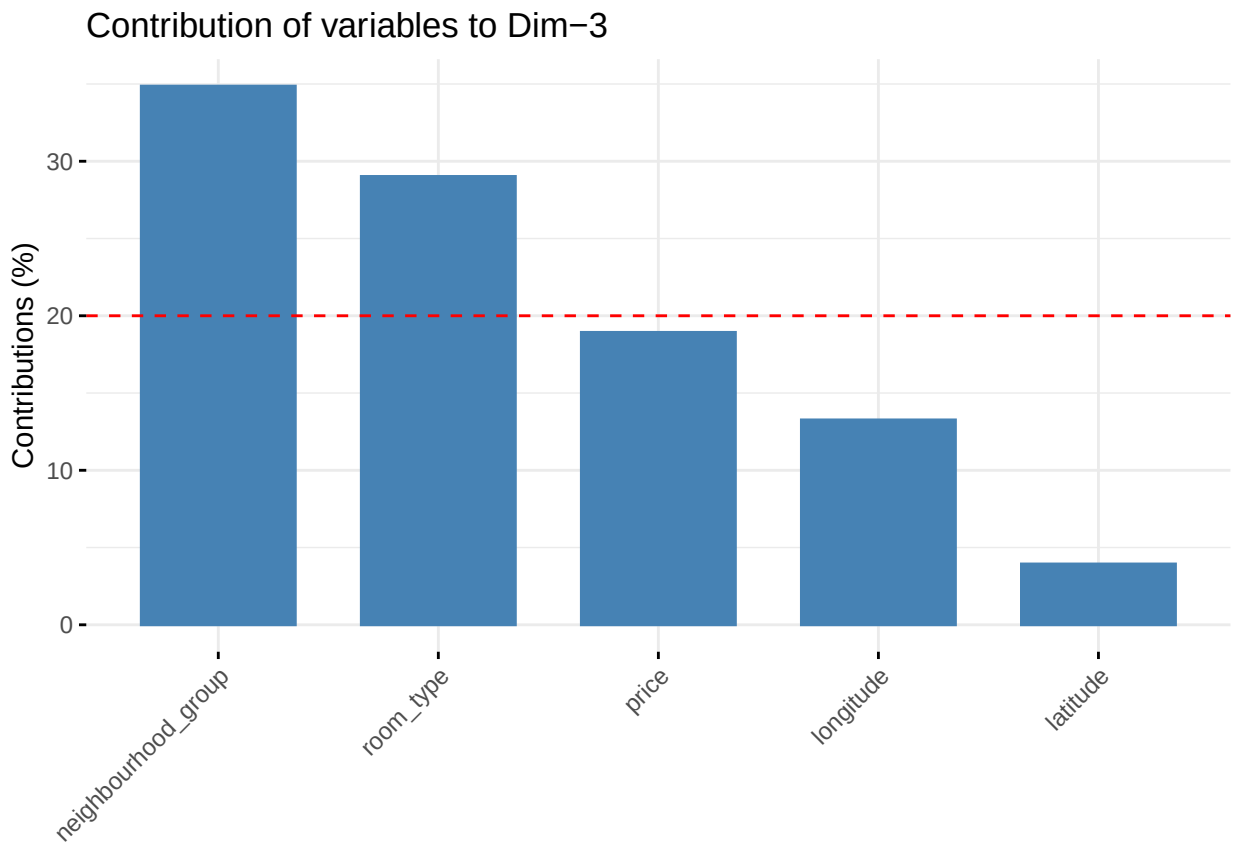

```
# Contribution to the first dimension
fviz_contrib(res.famd, "var", axes = 1)
```



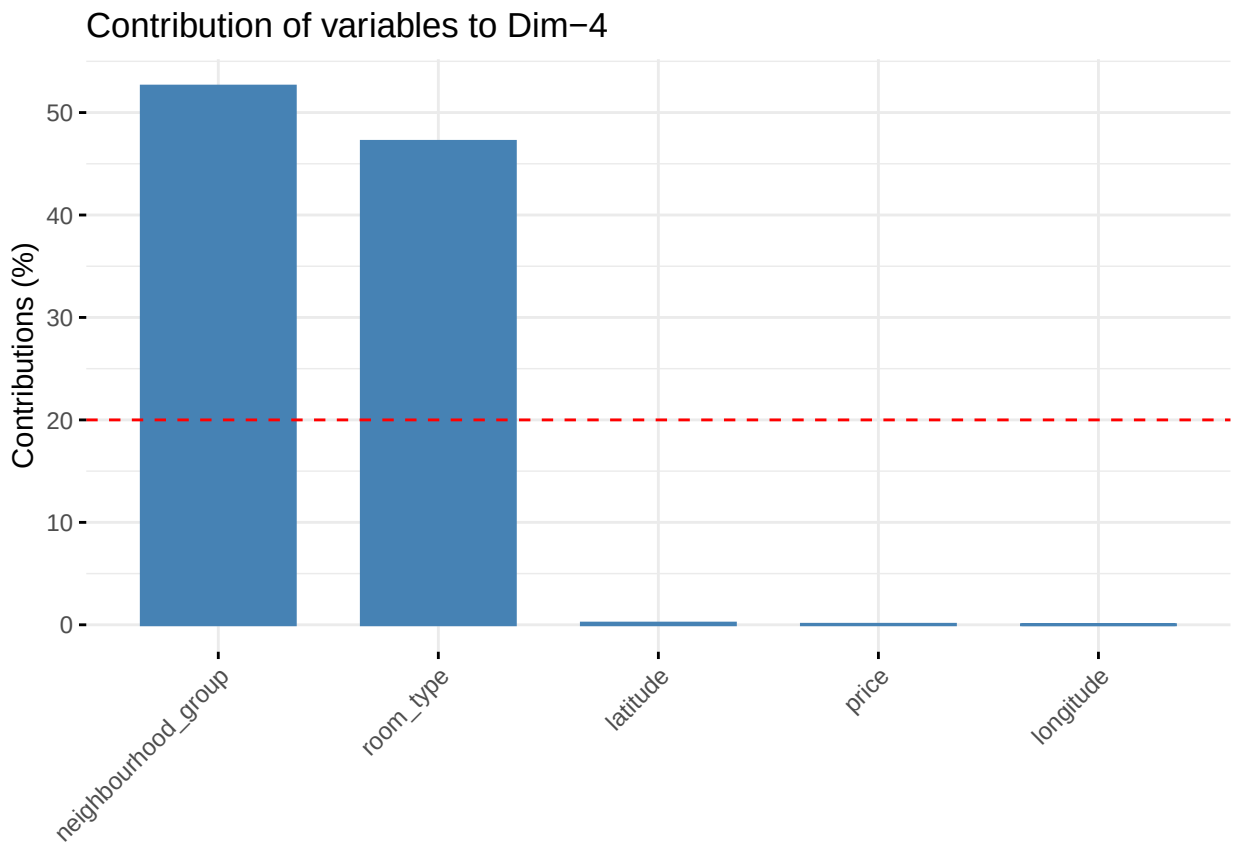

```
# Contribution to the second dimension
fviz_contrib(res.famd, "var", axes = 2)
```



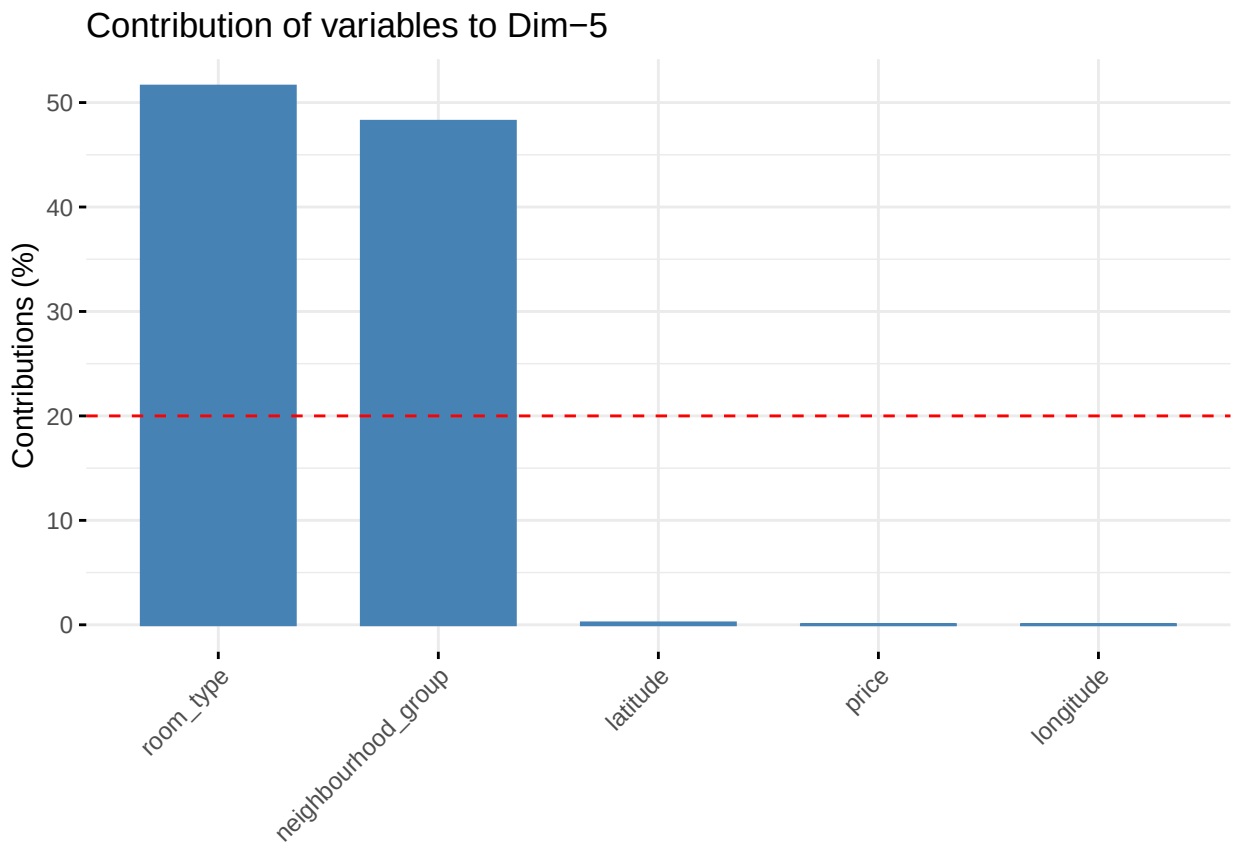

```
# Contribution to the third dimension
fviz_contrib(res.famd, "var", axes = 3)
```




```
# Contribution to the forth dimension
fviz_contrib(res.famd, "var", axes = 4)
```




```
# Contribution to the fifth dimension
fviz_contrib(res.famd, "var", axes = 5)
```




```
# Contribution to the sixth dimension
#fviz_contrib(res.famd, "var", axes = 6)
# Contribution to the seventh dimension
#fviz_contrib(res.famd, "var", axes = 7)
# Contribution to the eighth dimension
#fviz_contrib(res.famd, "var", axes = 8)
```