Data Preprocessing in R: Some unavoidable steps

Ziyue Gao

2018/1/25

load the data from csv file

Usually we could use two package to load large data sets: `data.table` (with function `fread`) and `readr`(with function `read_csv`). The `fread` function can read the data faster that `read_csv` function. However, the `read_csv` function's result is a tibble, which provide great convenience when we want to manipulate the data. I will use the zillow prize dataset from Kaggle: https://www.kaggle.com/c/zillow-prize-1/data as example.

```r
library(data.table)
library(readr)

# zillow <- fread('/Users/ziyue/Documents/Academic/datasets/Zillow/properties_2016.csv',header = T)
zillow <- read_csv('/Users/ziyue/Documents/Academic/datasets/Zillow/properties_2016.csv',col_names = T)
```

```
## Parsed with column specification:
## cols(
##   .default = col_integer(),
##   architecturalstyletypeid = col_character(),
##   bathroomcnt = col_double(),
##   bedroomcnt = col_double(),
##   calculatedbathnbr = col_double(),
##   calculatedfinishedsquarefeet = col_double(),
##   fips = col_character(),
##   hashottuborspa = col_character(),
##   lotsizesquarefeet = col_double(),
##   pooltypeid10 = col_character(),
##   pooltypeid2 = col_character(),
##   propertycountylandusecode = col_character(),
##   propertyzoningdesc = col_character(),
##   rawcensustractandblock = col_character(),
##   roomcnt = col_double(),
##   typeconstructiontypeid = col_character(),
##   yearbuilt = col_double(),
##   fireplaceflag = col_character(),
##   structuretaxvaluedollarcnt = col_double(),
##   taxvaluedollarcnt = col_double(),
##   landtaxvaluedollarcnt = col_double()
##   # ... with 3 more columns
## )

## See spec(...) for full column specifications.
```

```r
logerror = read_csv('/Users/ziyue/Documents/Academic/datasets/Zillow/train_2016_v2.csv',col_names = T)
```

```
## Parsed with column specification:
## cols(
##   parcelid = col_integer(),
##   logerror = col_double(),
##   transactiondate = col_date(format = "")
## )
```

exploratory data analysis

After loading the data, the first thing we need to do is looking at its size, and what kind of variables this data set have. Sometimes the type of the variable is not suitable for it. For example, in the zillow dataset, some ID which represent the type of equipment in the house are loaded as int (which should be loaded as factors)

```
library(tidyverse)
```

```
## Loading tidyverse: ggplot2
## Loading tidyverse: tibble
## Loading tidyverse: tidyr
## Loading tidyverse: purrr
## Loading tidyverse: dplyr

## Conflicts with tidy packages ----------------------------------------------

## between():   dplyr, data.table
## filter():    dplyr, stats
## first():     dplyr, data.table
## lag():       dplyr, stats
## last():      dplyr, data.table
## transpose(): purrr, data.table
```

```
dim(zillow)
```

```
## [1] 2985217      58
```

```
set.seed(1)
z_sample <- zillow[sample(1:nrow(zillow),300000,replace = F),]
table(sapply(z_sample,class))
```

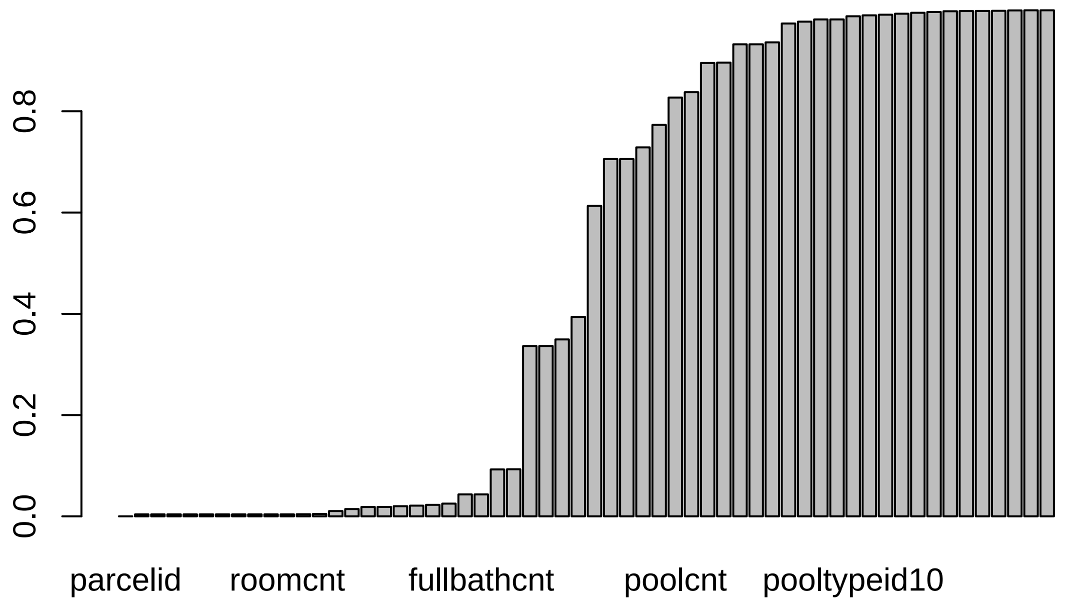
```
##
## character   integer   numeric
##        11        35        12
```

```
id_idx = grep('id',names(z_sample))
z_sample[,id_idx[-1]] = lapply(z_sample[,id_idx[-1]], factor)
```

missing values

Next, we need to visualize the missing data and clear off variables with too many missings (the threshold here is 70%)

```
z_missing <- sapply(z_sample, function(x) sum(is.na(x)))/nrow(z_sample)
barplot(sort(z_missing))
```



```r
missing_names <- names(which(z_missing > 0.5))
missing_names
```

```
##  [1] "airconditioningtypeid"    "architecturalstyletypeid"
##  [3] "basementsqft"             "buildingclasstypeid"
##  [5] "decktypeid"               "finishedfloor1squarefeet"
##  [7] "finishedsquarefeet13"     "finishedsquarefeet15"
##  [9] "finishedsquarefeet50"     "finishedsquarefeet6"
## [11] "fireplacecnt"             "garagecarcnt"
## [13] "garagetotalsqft"          "hashottuborspa"
## [15] "poolcnt"                  "poolsizesum"
## [17] "pooltypeid10"             "pooltypeid2"
## [19] "pooltypeid7"              "regionidneighborhood"
## [21] "storytypeid"              "threequarterbathnbr"
## [23] "typeconstructiontypeid"   "yardbuildingsqft17"
## [25] "yardbuildingsqft26"       "numberofstories"
## [27] "fireplaceflag"            "taxdelinquencyflag"
## [29] "taxdelinquencyyear"
```

```r
z_sample <- z_sample[,z_missing < 0.5]
dim(z_sample)
```

```
## [1] 300000     29
```

variable selection (1)

Next, we will look into the variables. The steps are:

1. Look at the numerical and categorical variables separately. Methods usually contains: correlation plot, contingency table.
2. Find out relationship between different classes of variables. Methods usually contains: boxplot, ggplot colored/faceted by category.

During the whole process, we could pick out the variables that are useful.

numerical variable: correlation matrix

First we look into the numerical ones. The `melt` function in `reshape2` package is used to melt the correlation matrix and plot the correlation heat matrix. Here we could only find that some variables are highly correlated but do not know there name.

One thing needs to be noticed is that we when deleting the highly correlated variables, we need to choose the one with more missing values to delete. We could use the `findCorrelation` function in `caret` package to pick out the highly linear correlated variables and delete them.

```
library(caret)
```

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

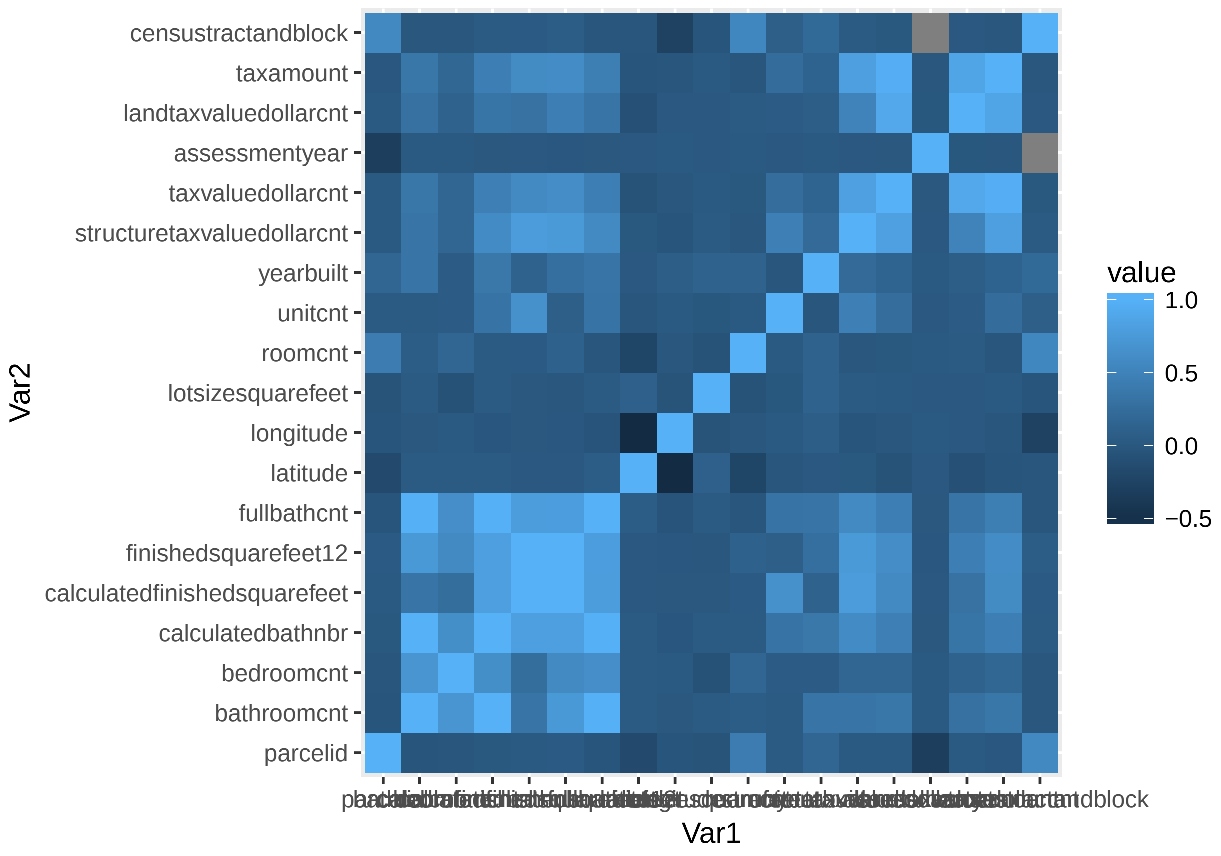
```
##
## Attaching package: 'caret'
```

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

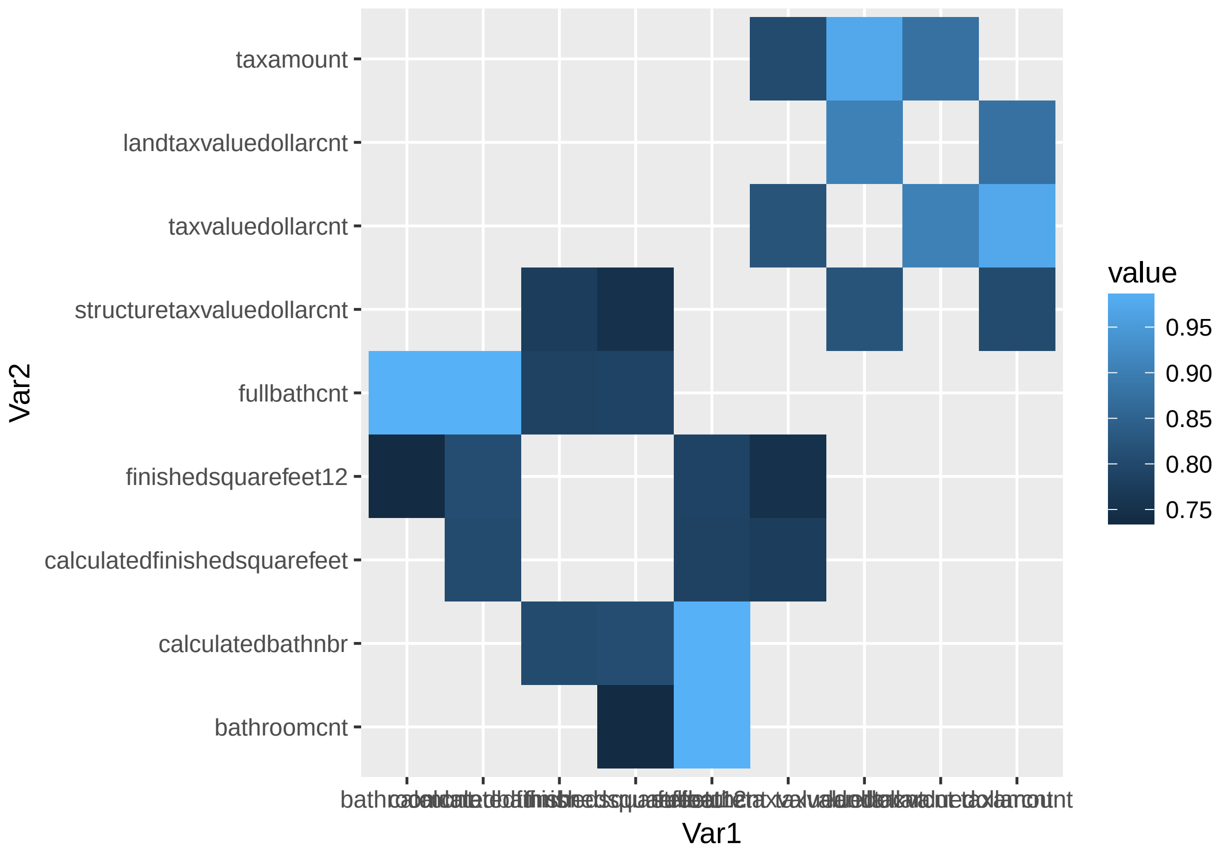
```
varname_byclass <- function(x,class_name){
  varclass <- sapply(x, class)
  pick <- varclass %in% class_name
  return(pick)
}

z_numeric <- z_sample[,varname_byclass(z_sample,c('numeric','integer'))]
cor_matrix <- cor(z_numeric,use = 'pairwise.complete.obs')
```

```
## Warning in cor(z_numeric, use = "pairwise.complete.obs"): the standard
## deviation is zero
```

```
cor_list <- melt(cor_matrix)

cor_list %>%
  ggplot(aes(x = Var1, y = Var2, fill = value)) +
  geom_tile()
```



```r
cor_list %>%
  filter(abs(value) > 0.7 & abs(value) < 1) %>%
  ggplot(aes(x = Var1, y = Var2, fill = value)) +
  geom_tile()
```



```r
high_corr_var <- findCorrelation(cor_matrix,cutoff = 0.7)
z_numeric <- z_numeric[,-high_corr_var]
```

categorical variables

Then we look into the character/factor variables. And we will pick out the factors which have only a few levels because too many different levels is not helpful when modeling.

```r
library(YaleToolkit)
```

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

```r
z_str <- z_sample[,varname_byclass(z_sample,c('character'))]
z_str <- z_str[,sapply(z_str, function(x) length(unique(x))) < 20]
z_str <- lapply(z_str,factor)
```

```r
z_fac = z_sample[,varname_byclass(z_sample,c('factor'))]
str(z_fac)
```

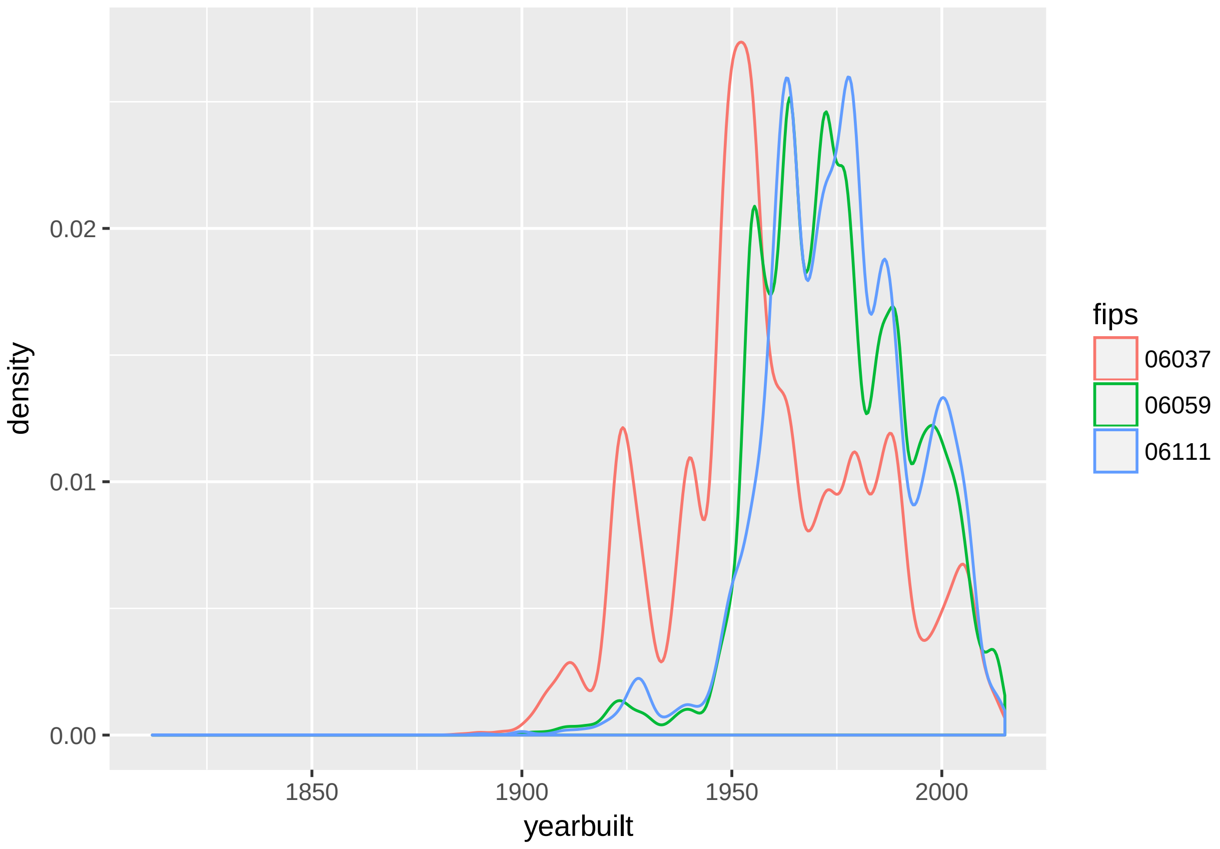
```
## Classes 'tbl_df', 'tbl' and 'data.frame':    300000 obs. of  6 variables:
##  $ buildingqualitytypeid: Factor w/ 11 levels "1","3","4","5",..: 1 NA 6 6 3 6 6 6 1 6 ...
##  $ heatingorsystemtypeid: Factor w/ 12 levels "1","2","6","7",..: 2 NA 2 2 2 NA 4 2 2 4 ...
##  $ propertylandusetypeid: Factor w/ 15 levels "31","47","246",..: 7 7 13 11 7 3 7 7 11 7 ...
##  $ regionidcity         : Factor w/ 183 levels "3491","3980",..: 20 44 20 67 69 20 146 54 20 134 ...
##  $ regionidcounty       : Factor w/ 3 levels "1286","2061",..: 3 1 3 3 3 3 3 3 3 3 ...
##  $ regionidzip          : Factor w/ 394 levels "95982","95983",..: 47 123 25 125 127 40 253 249 96 1
```

interaction

Now we could look at the interelation between numerical and factor variables. Here are two examples.

```
z_samplenew = cbind(z_numeric,z_fac,z_str)
z_samplenew$censustractandblock = NULL
z_samplenew %>%
  ggplot() +
  geom_density(aes(x = yearbuilt, col = fips))
```

```
## Warning: Removed 6014 rows containing non-finite values (stat_density).
```



zero-variance variable

Some variables will have over 95% same value, so we want to find them out and do not include them in our model.

```
nzv = nearZeroVar(z_samplenew, saveMetrics = T)
drop_names <- rownames(nzv[nzv$nzv,])
z_samplenew[,drop_names] = NULL
```

Feature preprocessing

Feature preprocessing is a very important step before we building models. And different kinds of features have different preprocessing methods. Here we will use other small datasets to show the process.

numerical

Usually we can use the `preprocess` (with method = scale or center and so on) and `predict` function from package `caret`. The function will only choose numerical variables. For tree-based model, we do not need to do the scaling.

```r
library(caret)
data(iris)
pre_pro <- preProcess(iris, method = c('center','scale'))
summary(predict(pre_pro,iris))
```

```
##   Sepal.Length       Sepal.Width       Petal.Length      Petal.Width
##  Min.   :-1.86378   Min.   :-2.4258   Min.   :-1.5623   Min.   :-1.4422
##  1st Qu.:-0.89767   1st Qu.:-0.5904   1st Qu.:-1.2225   1st Qu.:-1.1799
##  Median :-0.05233   Median :-0.1315   Median : 0.3354   Median : 0.1321
##  Mean   : 0.00000   Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.0000
##  3rd Qu.: 0.67225   3rd Qu.: 0.5567   3rd Qu.: 0.7602   3rd Qu.: 0.7880
##  Max.   : 2.48370   Max.   : 3.0805   Max.   : 1.7799   Max.   : 1.7064
##        Species
##  setosa    :50
##  versicolor:50
##  virginica :50
##
##
##
```

categorical and ordinal features

If we want to use a non-tree (linear model, kNN, neural nets) based model, the way is to use one-hot coding. If two categorical features interacts, then we could combine these two first and do the one-hot coding. We could use the function `dummyVars` from the package 'caret'

If it is a ordinal and we want to use tree based model, we could just map the categoies into numbers.

```r
dv <- dummyVars(~Species,iris)
head(predict(dv,iris))
```

```
##   Species.setosa Species.versicolor Species.virginica
## 1              1                  0                 0
## 2              1                  0                 0
## 3              1                  0                 0
## 4              1                  0                 0
## 5              1                  0                 0
## 6              1                  0                 0
```

date and time

Basically there are three ways: Use Package 'lubridate', use function 'as.date', and when there is only year and month, use function as.yearmon from library 'zoo'(the class of the result will be 'yearmon').

The difference between dates are also very important, and we may also want to find some holidays. (use the function `isHoliday` from package `tis`)

```r
library(lubridate)
```

```
##
## Attaching package: 'lubridate'

## The following objects are masked from 'package:data.table':
##
##     hour, isoweek, mday, minute, month, quarter, second, wday,
##     week, yday, year

## The following object is masked from 'package:base':
##
##     date
```

```r
dates <- c('20150101','20140205','2006-02-02')
ymd(dates)
```

```
## [1] "2015-01-01" "2014-02-05" "2006-02-02"
```

```r
as.Date(dates, format = '%Y%m%d') # cannot handle the last
```

```
## [1] "2015-01-01" "2014-02-05" NA
```

```r
dates <- ymd(dates)
as.numeric(dates[1] - dates[2])
```

```
## [1] 330
```

```r
library(tis)
```

```
##
## Attaching package: 'tis'

## The following objects are masked from 'package:lubridate':
##
##     day, hms, month, period, quarter, today, year, ymd

## The following object is masked from 'package:dplyr':
##
##     between

## The following objects are masked from 'package:data.table':
##
##     between, month, quarter, year
```

```r
isHoliday(dates)
```

```
## [1]  TRUE FALSE FALSE
```

missing value imputation

Random forest could be used to do missing value imputation. Here we use the `missForest` package. However this function cannot handle categorical predictors with more than 53 categories, so we need to pick out the categories first.

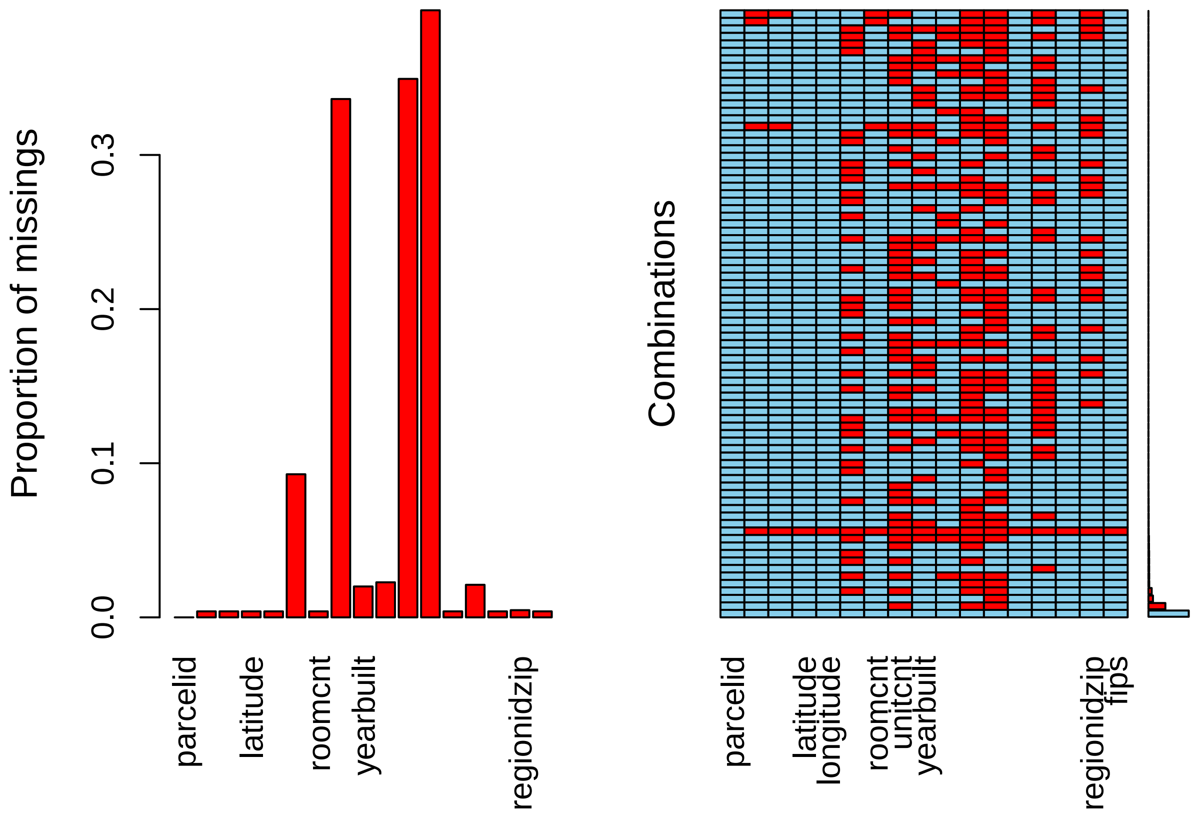
```r
library(VIM)
```

```
## Loading required package: colorspace

## VIM is ready to use.
##  Since version 4.0.0 the GUI is in its own package VIMGUI.
##
##          Please use the package to use the new (and old) GUI.
## Suggestions and bug-reports can be submitted at: https://github.com/alexkowa/VIM/issues

##
## Attaching package: 'VIM'

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

```r
library(missForest)
```

```
## Loading required package: randomForest

## randomForest 4.6-12

## 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

## Loading required package: foreach

##
## Attaching package: 'foreach'

## The following objects are masked from 'package:purrr':
##
##     accumulate, when

## Loading required package: itertools

## Loading required package: iterators
```

```r
aggr(z_samplenew)
```



```r
z_missing <- sapply(z_samplenew, function(x) sum(is.na(x)))/nrow(z_samplenew)
str(z_samplenew)
```

```
## 'data.frame':    300000 obs. of  17 variables:
##  $ parcelid             : int  11647058 13851750 11944257 11856750 12396316 11979057 12994077 129144
##  $ bathroomcnt          : num  3 2 1 2 2 2 1 2 2 1 ...
##  $ bedroomcnt           : num  3 3 1 3 3 3 2 3 2 3 ...
##  $ latitude             : int  34078379 33928874 34084882 34041583 33973282 34114159 34064887 341031
##  $ longitude            : int  -118460583 -117964714 -118278500 -118092976 -118085470 -118181685 -11
##  $ lotsizesquarefeet    : num  35999 7920 NA 255664 5038 ...
##  $ roomcnt              : num  0 6 0 0 0 0 0 0 0 0 ...
##  $ unitcnt              : int  1 NA 1 1 1 2 1 1 1 1 ...
##  $ yearbuilt            : num  1951 1958 1926 1977 1957 ...
##  $ landtaxvaluedollarcnt: num  1664991 173426 88759 40775 124810 ...
##  $ buildingqualitytypeid: Factor w/ 11 levels "1","3","4","5",..: 1 NA 6 6 3 6 6 6 1 6 ...
##  $ heatingorsystemtypeid: Factor w/ 12 levels "1","2","6","7",..: 2 NA 2 2 2 NA 4 2 2 4 ...
##  $ propertylandusetypeid: Factor w/ 15 levels "31","47","246",..: 7 7 13 11 7 3 7 7 11 7 ...
##  $ regionidcity         : Factor w/ 183 levels "3491","3980",..: 20 44 20 67 69 20 146 54 20 134 ...
##  $ regionidcounty       : Factor w/ 3 levels "1286","2061",..: 3 1 3 3 3 3 3 3 3 3 ...
##  $ regionidzip          : Factor w/ 394 levels "95982","95983",..: 47 123 25 125 127 40 253 249 96 1
##  $ fips                 : Factor w/ 3 levels "06037","06059",..: 1 2 1 1 1 1 1 1 1 1 ...
```

```r
# Take a lot of time!
# z_imputed <- missForest(z_samplenew[,-which(names(z_samplenew) %in% c("parcelid", "regionidcity", "re
```