According to a [Forbes article](https://www.forbes.com/sites/gilpress/2016/03/23/data-preparation-most-time-consuming-least-enjoyable-data-science-task-survey-says), cleaning and organizing data is the most time-consuming and least enjoyable data science task. Of all the resources out there, DataExplorer is one of them, with its sole mission to minimize the 80%, and make it enjoyable. As a result, one fundamental design principle is to be extremely user-friendly. Most of the time, one function call is all you need.

Data manipulation is powered by data.table, so tasks involving big datasets usually complete in a few seconds. In addition, the package is flexible enough with input data classes, so you should be able to throw in any data.frame-like objects. However, certain functions require a data.table class object as input

Now enough said and let's look at some code, shall we?

Take the BostonHousing dataset from the mlbench library:

library(mlbench)

library(data.table)

library(DataExplorer)

data("BostonHousing", package = "mlbench")

**Initial Visualization**

Without knowing anything about the data, my first 3 tasks are almost always:

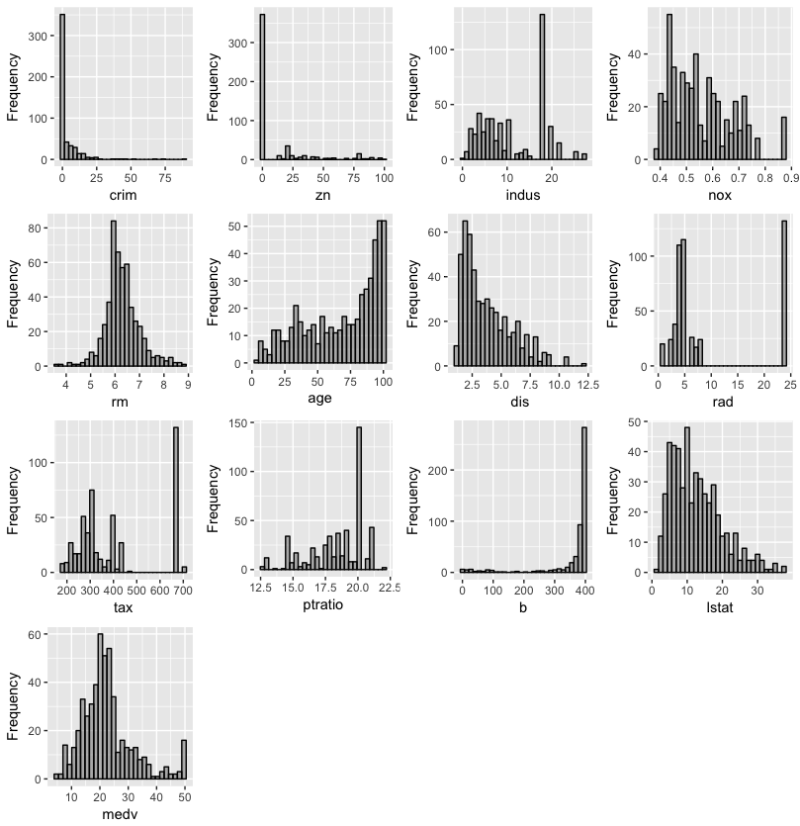
library(DataExplorer)

plot\_missing(BostonHousing) ## Are there missing values, and what is the missing data profile?

plot\_bar(BostonHousing) ## How does the categorical frequency for each discrete variable look like?

plot\_histogram(BostonHousing) ## What is the distribution of each continuous variable?

While there are not many interesting insights from plot\_missing and plot\_bar, below is the output from plot\_histogram.

[](http://revolution-computing.typepad.com/.a/6a010534b1db25970b01b8d2d79d35970c-pi)

Upon scrutiny, the variable **rad** looks like discrete, and I want to group **crim**, **zn**, **indus** and **b** into bins as well. Let's do so:

## Set `rad` to factor

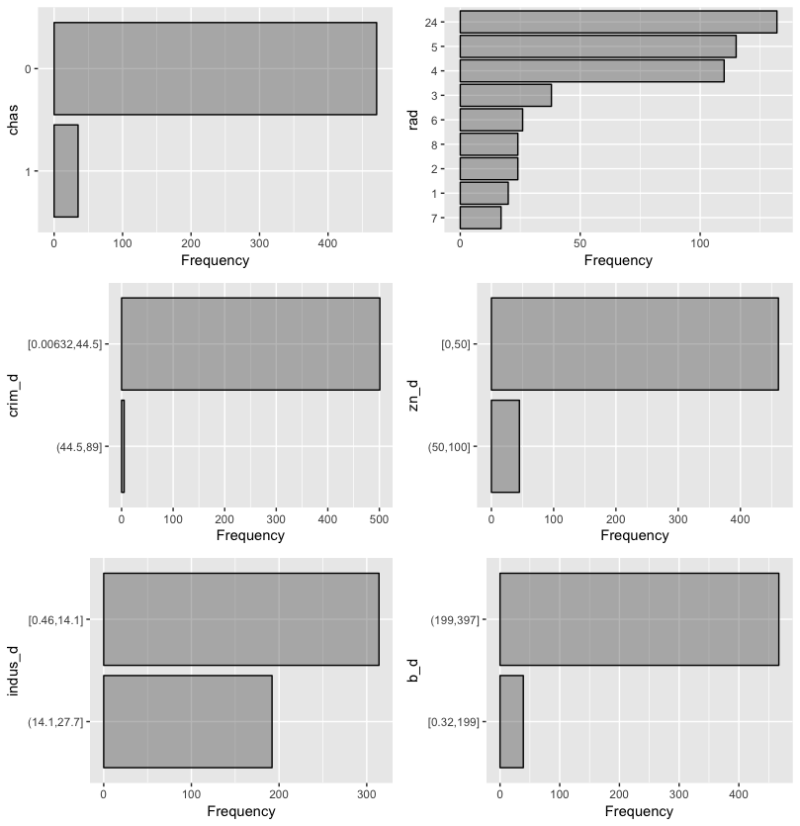
BostonHousing$rad <- as.factor(BostonHousing$rad)

## Create new discrete variables

for (col in c("crim", "zn", "indus", "b"))   
 BostonHousing[[paste0(col, "\_d")]] <- as.factor(ggplot2::cut\_interval(BostonHousing[[col]], 2))

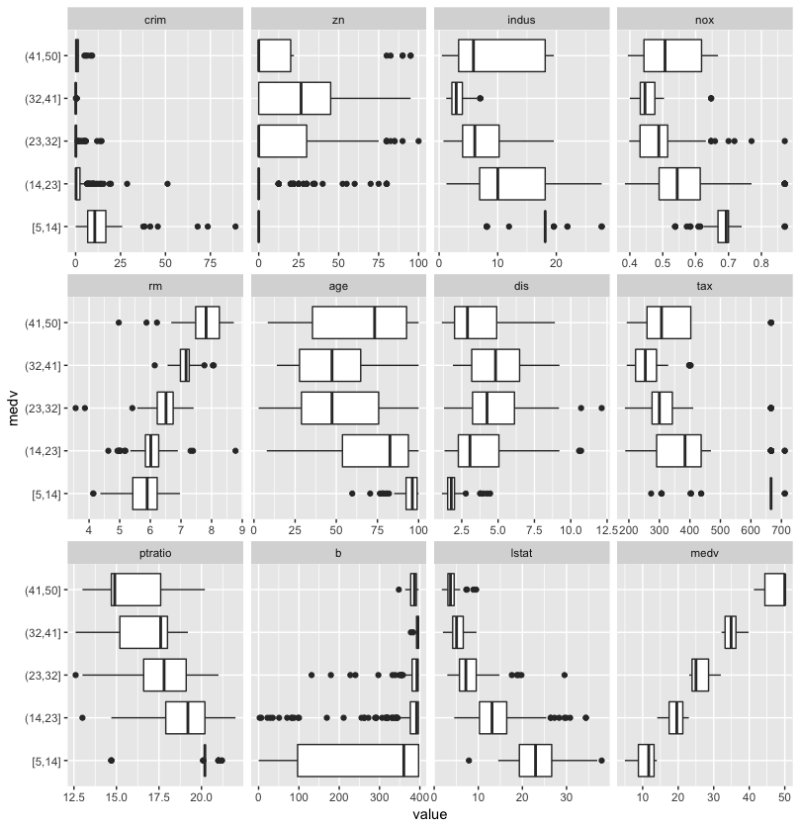
## Plot bar chart for all discrete variables

plot\_bar(BostonHousing)

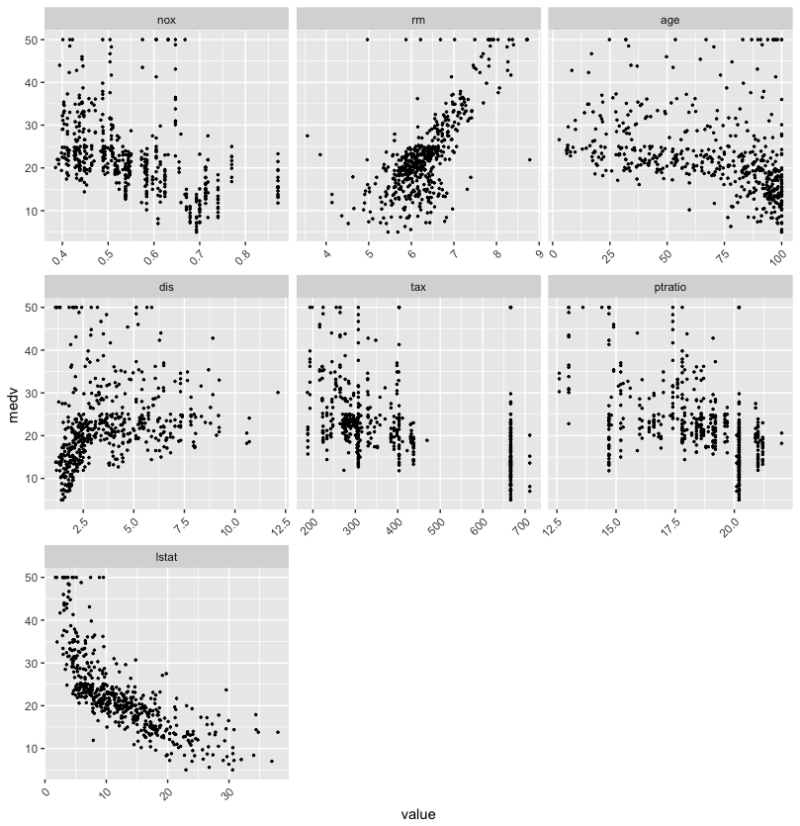
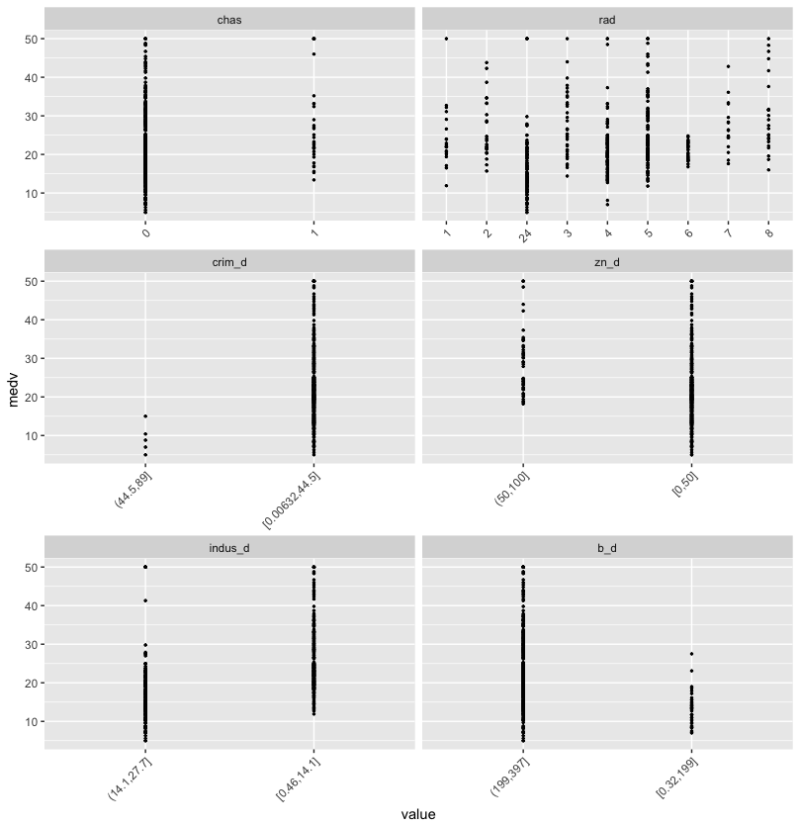
[](http://revolution-computing.typepad.com/.a/6a010534b1db25970b01b7c94d2fa0970b-pi)

At this point, we have much better understanding of the data distribution. Now assume we are interested in **medv** (median value of owner-occupied homes in USD 1000's), and would like to build a model to predict it. Let's plot it against all other variables:

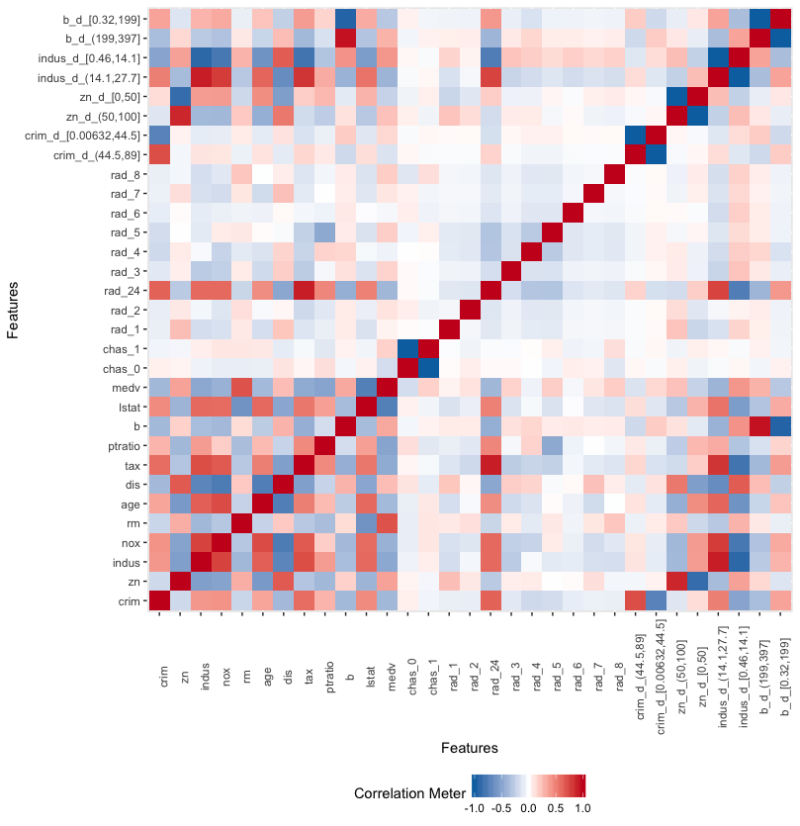
plot\_boxplot(BostonHousing, by = "medv")

[](http://revolution-computing.typepad.com/.a/6a010534b1db25970b01b7c94d2fbc970b-pi)

plot\_scatterplot(  
 subset(BostonHousing, select = -c(crim, zn, indus, b)),   
 by = "medv", size = 0.5)

[](http://revolution-computing.typepad.com/.a/6a010534b1db25970b01b7c94d2fc6970b-pi)  
[](http://revolution-computing.typepad.com/.a/6a010534b1db25970b01b7c94d2fd4970b-pi)

plot\_correlation(BostonHousing)

[](http://revolution-computing.typepad.com/.a/6a010534b1db25970b01bb09f098c7970d-pi)

And this is how you slice & dice your data, and analyze correlation with merely 3 lines of code.

**Feature Engineering**

Feature engineering is a crucial step in building better models. DataExplorer provides a couple of functions to ease the process. All of them require a data.table as the input object, because it is lightning fast. However, if you don't feel like coding in data.table syntax, you may adopt the following process:

## Set your data to `data.table` first

your\_data <- data.table(your\_data)

## Apply DataExplorer functions

group\_category(your\_data, ...)

drop\_columns(your\_data, ...)

set\_missing(your\_data, ...)

## Set data back to the original object

class(your\_data) <- "original\_object\_name"

Let's return to the BostonHousing dataset. For the rest of this section, we'll assume the data has been converted to a data.table already.

library(data.table)

BostonHousingDT <- data.table(BostonHousing)

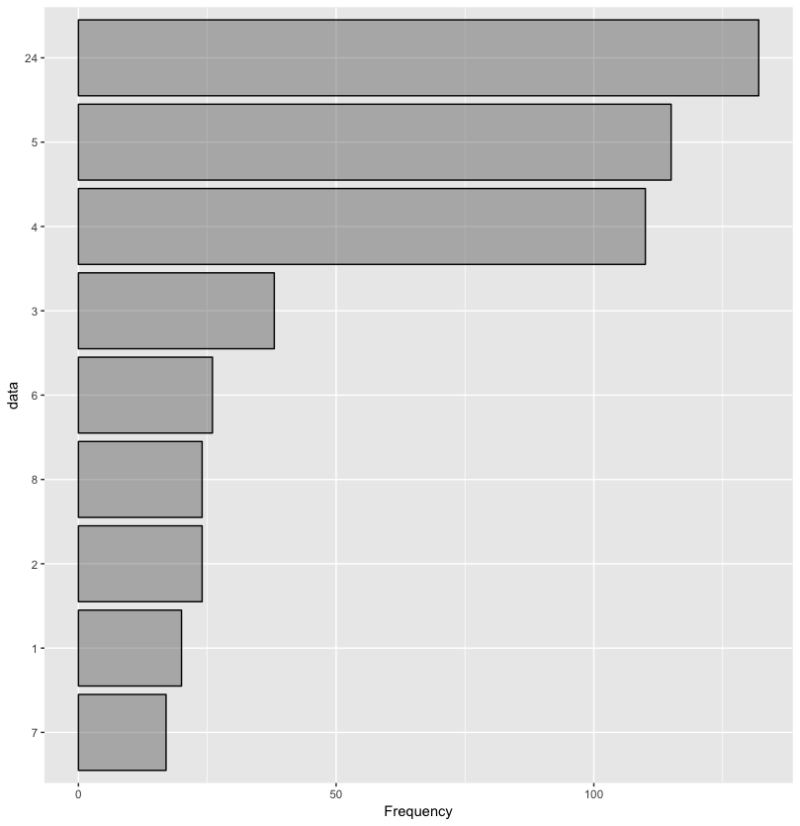
Remember those transformed continuous variables? Let's drop them:

drop\_columns(BostonHousingDT, c("crim", "zn", "indus", "b"))

Note: Because data.table updates by reference, the original object is updated without the need to re-assign a returned object.

Let's take a look at the discrete variable **rad**:

plot\_bar(BostonHousingDT$rad)

[](http://revolution-computing.typepad.com/.a/6a010534b1db25970b01b7c94d2fee970b-pi)

I think categories other than 4, 5 and 24 are too sparse, and might skew my model fit. How could I group all the sparse categories together?

group\_category(BostonHousingDT, "rad", 0.25, update = FALSE)

# rad cnt pct cum\_pct

# 1: 24 132 0.2608696 0.2608696

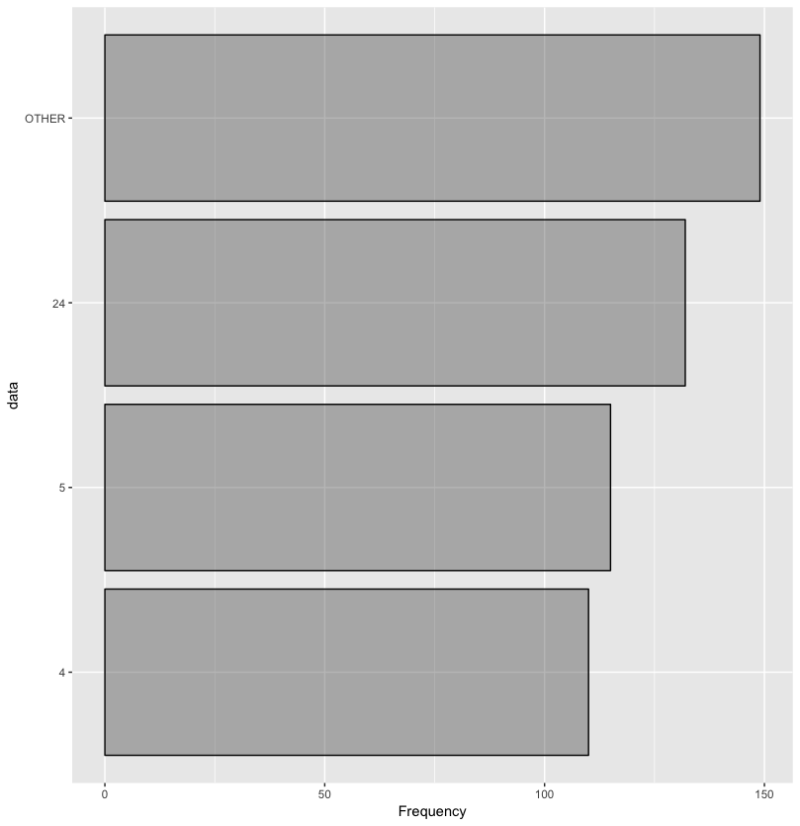
# 2: 5 115 0.2272727 0.4881423

# 3: 4 110 0.2173913 0.7055336

Looks like grouping by bottom 25% of **rad** would give me what I need. Let's do so:

group\_category(BostonHousingDT, "rad", 0.25, update = TRUE)

plot\_bar(BostonHousingDT$rad)

[](http://revolution-computing.typepad.com/.a/6a010534b1db25970b01b7c94d2ff8970b-pi)

In addition to categorical frequency, you may also play with the measure argument to group by the sum of a different variable. See ?group\_category for more example use cases.

**Data Report**

To generate a report of your data:

create\_report(BostonHousing)