Overview of simple outlier detection methods with their combination using dplyr and ruler packages.

**Prologue**

During the process of data analysis one of the most crucial steps is to identify and account for [outliers](https://en.wikipedia.org/wiki/Outlier), observations that have essentially different nature than most other observations. Their presence can lead to untrustworthy conclusions. The most complicated part of this task is **to define a notion of “outlier”**. After that, it is straightforward to identify them based on given data.

There are many techniques developed for outlier detection. Majority of them deal with numerical data. This post will describe the most basic ones with their application using [dplyr](http://dplyr.tidyverse.org/) and [ruler](https://echasnovski.github.io/ruler/) packages.

Ruler Package:

This README is structured as follows:

* **Installation** shows ways to install package.
* **Example** shows the basic usage of ruler for exploration of obeying user-defined rules and its automatic validation.
* **Overview** explains basic data and function types with design behind them.
* **Usage** describes ruler’s capabilities in more detail.
* **Other packages for validation and assertions** lists alternatives for described tasks.

**Installation**

You can install current stable version from CRAN with:

[install.packages](https://rdrr.io/r/utils/install.packages.html)("ruler")

Also you can install development version from github with:

# install.packages("devtools")

devtools::[install\_github](https://devtools.r-lib.org/reference/remote-reexports.html)("echasnovski/ruler")

**Example**

# Utilities functions

is\_integerish <- function(x) {

[all](https://rdrr.io/r/base/all.html)(x == [as.integer](https://rdrr.io/r/base/integer.html)(x))

}

z\_score <- function(x) {

[abs](https://rdrr.io/r/base/MathFun.html)(x - [mean](https://rdrr.io/r/base/mean.html)(x)) / [sd](https://rdrr.io/r/stats/sd.html)(x)

}

# Define rule packs

my\_packs <- [list](https://rdrr.io/r/base/list.html)(

[data\_packs](https://echasnovski.github.io/ruler/reference/rule-packs.html)(

dims = . %>% summarise(nrow\_low = [nrow](https://rdrr.io/r/base/nrow.html)(.) >= 10, nrow\_high = [nrow](https://rdrr.io/r/base/nrow.html)(.) <= 15,

ncol\_low = [ncol](https://rdrr.io/r/base/nrow.html)(.) >= 20, ncol\_high = [ncol](https://rdrr.io/r/base/nrow.html)(.) <= 30)

),

[group\_packs](https://echasnovski.github.io/ruler/reference/rule-packs.html)(

vs\_am\_num = . %>% group\_by(vs, am) %>% summarise(vs\_am\_low = n() >= 7),

.group\_vars = [c](https://rdrr.io/r/base/c.html)("vs", "am")

),

[col\_packs](https://echasnovski.github.io/ruler/reference/rule-packs.html)(

enough\_col\_sum = . %>%

summarise\_if(is\_integerish, [rules](https://echasnovski.github.io/ruler/reference/rules.html)(is\_enough = [sum](https://rdrr.io/r/base/sum.html)(.) >= 14))

),

[row\_packs](https://echasnovski.github.io/ruler/reference/rule-packs.html)(

enough\_row\_sum = . %>%

[filter](https://rdrr.io/r/stats/filter.html)(vs == 1) %>%

transmute(is\_enough = [rowSums](https://rdrr.io/r/base/colSums.html)(.) >= 200)

),

[cell\_packs](https://echasnovski.github.io/ruler/reference/rule-packs.html)(

dbl\_not\_outlier = . %>%

transmute\_if(is.numeric, [rules](https://echasnovski.github.io/ruler/reference/rules.html)(is\_not\_out = z\_score(.) < 1)) %>%

slice(-(1:5))

)

)

# Expose data to rules

mtcars\_exposed <- mtcars %>% as\_tibble() %>%

[expose](https://echasnovski.github.io/ruler/reference/expose.html)(my\_packs)

# View exposure

mtcars\_exposed %>% [get\_exposure](https://echasnovski.github.io/ruler/reference/exposure.html)()

#> Exposure

#>

#> Packs info:

#> # A tibble: 5 x 4

#> name type fun remove\_obeyers

#> <chr> <chr> <list> <lgl>

#> 1 dims data\_pack <data\_pck> TRUE

#> 2 vs\_am\_num group\_pack <grop\_pck> TRUE

#> 3 enough\_col\_sum col\_pack <col\_pack> TRUE

#> 4 enough\_row\_sum row\_pack <row\_pack> TRUE

#> 5 dbl\_not\_outlier cell\_pack <cell\_pck> TRUE

#>

#> Tidy data validation report:

#> # A tibble: 117 x 5

#> pack rule var id value

#> <chr> <chr> <chr> <int> <lgl>

#> 1 dims nrow\_high .all 0 FALSE

#> 2 dims ncol\_low .all 0 FALSE

#> 3 vs\_am\_num vs\_am\_low 0.1 0 FALSE

#> 4 enough\_col\_sum is\_enough am 0 FALSE

#> 5 enough\_row\_sum is\_enough .all 19 FALSE

#> 6 dbl\_not\_outlier is\_not\_out mpg 15 FALSE

#> # … with 111 more rows

# Assert any breaker

[invisible](https://rdrr.io/r/base/invisible.html)(mtcars\_exposed %>% [assert\_any\_breaker](https://echasnovski.github.io/ruler/reference/assert_any_breaker.html)())

#> Breakers report

#> Tidy data validation report:

#> # A tibble: 117 x 5

#> pack rule var id value

#> <chr> <chr> <chr> <int> <lgl>

#> 1 dims nrow\_high .all 0 FALSE

#> 2 dims ncol\_low .all 0 FALSE

#> 3 vs\_am\_num vs\_am\_low 0.1 0 FALSE

#> 4 enough\_col\_sum is\_enough am 0 FALSE

#> 5 enough\_row\_sum is\_enough .all 19 FALSE

#> 6 dbl\_not\_outlier is\_not\_out mpg 15 FALSE

#> # … with 111 more rows

#> Error: assert\_any\_breaker: Some breakers found in exposure.

**Overview**

**Rule** is a function which converts data unit of interest (data, group, column, row, cell) to logical value indicating whether this object satisfies certain condition.

**Rule pack** is a function which combines several rules into one functional block. The recommended way of creating rules is by creating packs right away with the use of dplyr and [magrittr](https://magrittr.tidyverse.org/)’s pipe operator.

**Exposing** data to rules means applying rules to data, collecting results in common format and attaching them to the data as an exposure attribute. In this way actual exposure can be done in multiple steps and also be a part of a general data preparation pipeline.

**Exposure** is a format designed to contain uniform information about validation of different data units. For reproducibility it also saves information about applied packs. Basically exposure is a list with two elements:

1. **Packs info**: a [tibble](https://tibble.tidyverse.org/) with the following structure:
   * *name* <chr> : Name of the pack. If not set manually it will be imputed during exposure.
   * *type* <chr> : Name of pack type. Indicates which data unit pack checks.
   * *fun* <list> : List of rule pack functions.
   * *remove\_obeyers* <lgl> : Whether rows about obeyers (data units that obey certain rule) were removed from report after applying pack.
2. **Tidy data validation report**: a tibble with the following structure:
   * *pack* <chr> : Name of rule pack from column ‘name’ in packs info.
   * *rule* <chr> : Name of the rule defined in rule pack.
   * *var* <chr> : Name of the variable which validation result is reported. Value ‘.all’ is reserved and interpreted as ‘all columns as a whole’. **Note** that *var* doesn’t always represent the actual column in data frame: for group packs it represents the created group name.
   * *id* <int> : Index of the row in tested data frame which validation result is reported. Value 0 is reserved and interpreted as ‘all rows as a whole’.
   * *value* <lgl> : Whether the described data unit obeys the rule.

There are four basic combinations of var and id values which define five basic data units:

* var == '.all' and id == 0: Data as a whole.
* var != '.all' and id == 0: Group (var shouldn’t be an actual column name) or column (var should be an actual column name) as a whole.
* var == '.all' and id != 0: Row as a whole.
* var != '.all' and id != 0: Described cell.

With exposure attached to data one can perform different kinds of actions: exploration, assertion, imputation and so on.

**Usage**

**Creating packs**

**Data packs**

# List of two rule packs for checking data properties

my\_data\_packs <- [data\_packs](https://echasnovski.github.io/ruler/reference/rule-packs.html)(

# data\_dims is a pack name

data\_dims = . %>% summarise(

# ncol and nrow are rule names

ncol = [ncol](https://rdrr.io/r/base/nrow.html)(.) == 12,

nrow = [nrow](https://rdrr.io/r/base/nrow.html)(.) == 32

),

# Data after subsetting should have number of rows in between 10 and 30

# Rules are applied separately

vs\_1 = . %>% [filter](https://rdrr.io/r/stats/filter.html)(vs == 1) %>%

summarise(

nrow\_low = [nrow](https://rdrr.io/r/base/nrow.html)(.) > 10,

nrow\_high = [nrow](https://rdrr.io/r/base/nrow.html)(.) < 30

)

)

**Group packs**

# List of one nameless rule pack for checking group property

my\_group\_packs <- [group\_packs](https://echasnovski.github.io/ruler/reference/rule-packs.html)(

# Name will be imputed during exposure

. %>% group\_by(vs, am) %>%

summarise(any\_cyl\_6 = [any](https://rdrr.io/r/base/any.html)(cyl == 6)),

# One should supply grouping variables for correct interpretation of output

.group\_vars = [c](https://rdrr.io/r/base/c.html)("vs", "am")

)

**Column packs**

# rules() is a dplyr::funs() with necessary name imputations

# In column packs it should always be used instead of dplyr::funs()

# List of two rule pack for checking certain columns' properties

my\_col\_packs <- [col\_packs](https://echasnovski.github.io/ruler/reference/rule-packs.html)(

sum\_bounds = . %>% summarise\_at(

# Check only columns with names starting with 'c'

vars(starts\_with("c")),

[rules](https://echasnovski.github.io/ruler/reference/rules.html)(sum\_low = [sum](https://rdrr.io/r/base/sum.html)(.) > 300, sum\_high = [sum](https://rdrr.io/r/base/sum.html)(.) < 400)

),

# In the edge case of checking one column with one rule there is a need

# for forcing inclusion of names in the output of summarise\_at().

# This is done with naming argument in vars()

vs\_mean = . %>% summarise\_at(vars(vs = vs), [rules](https://echasnovski.github.io/ruler/reference/rules.html)([mean](https://rdrr.io/r/base/mean.html)(.) > 0.5))

)

**Row packs**

z\_score <- function(x) {

(x - [mean](https://rdrr.io/r/base/mean.html)(x)) / [sd](https://rdrr.io/r/stats/sd.html)(x)

}

# List of one rule pack checking certain rows' property

my\_row\_packs <- [row\_packs](https://echasnovski.github.io/ruler/reference/rule-packs.html)(

row\_mean = . %>% mutate(rowMean = [rowMeans](https://rdrr.io/r/base/colSums.html)(.)) %>%

transmute(is\_common\_row\_mean = [abs](https://rdrr.io/r/base/MathFun.html)(z\_score(rowMean)) < 1) %>%

# Check only rows 10-15

# Values in 'id' column of report will be based on input data (i.e. 10-15)

# and not on output data (1-6)

slice(10:15)

)

**Cell packs**

is\_integerish <- function(x) {

[all](https://rdrr.io/r/base/all.html)(x == [as.integer](https://rdrr.io/r/base/integer.html)(x))

}

# List of two cell pack checking certain cells' property

my\_cell\_packs <- [cell\_packs](https://echasnovski.github.io/ruler/reference/rule-packs.html)(

my\_cell\_pack\_1 = . %>% transmute\_if(

# Check only integer-like columns

is\_integerish,

[rules](https://echasnovski.github.io/ruler/reference/rules.html)(is\_common = [abs](https://rdrr.io/r/base/MathFun.html)(z\_score(.)) < 1)

) %>%

# Check only rows 20-30

slice(20:30),

# The same edge case as in column rule pack

vs\_side = . %>% transmute\_at(vars(vs = "vs"), [rules](https://echasnovski.github.io/ruler/reference/rules.html)(. > [mean](https://rdrr.io/r/base/mean.html)(.)))

)

**Exposing**

By default exposing removes obeyers.

mtcars %>%

[expose](https://echasnovski.github.io/ruler/reference/expose.html)(my\_data\_packs, my\_group\_packs) %>%

[get\_exposure](https://echasnovski.github.io/ruler/reference/exposure.html)()

#> Exposure

#>

#> Packs info:

#> # A tibble: 3 x 4

#> name type fun remove\_obeyers

#> <chr> <chr> <list> <lgl>

#> 1 data\_dims data\_pack <data\_pck> TRUE

#> 2 vs\_1 data\_pack <data\_pck> TRUE

#> 3 group\_pack\_\_1 group\_pack <grop\_pck> TRUE

#>

#> Tidy data validation report:

#> # A tibble: 3 x 5

#> pack rule var id value

#> <chr> <chr> <chr> <int> <lgl>

#> 1 data\_dims ncol .all 0 FALSE

#> 2 group\_pack\_\_1 any\_cyl\_6 0.0 0 FALSE

#> 3 group\_pack\_\_1 any\_cyl\_6 1.1 0 FALSE

One can leave obeyers by setting .remove\_obeyers to FALSE.

mtcars %>%

[expose](https://echasnovski.github.io/ruler/reference/expose.html)(my\_data\_packs, my\_group\_packs, .remove\_obeyers = FALSE) %>%

[get\_exposure](https://echasnovski.github.io/ruler/reference/exposure.html)()

#> Exposure

#>

#> Packs info:

#> # A tibble: 3 x 4

#> name type fun remove\_obeyers

#> <chr> <chr> <list> <lgl>

#> 1 data\_dims data\_pack <data\_pck> FALSE

#> 2 vs\_1 data\_pack <data\_pck> FALSE

#> 3 group\_pack\_\_1 group\_pack <grop\_pck> FALSE

#>

#> Tidy data validation report:

#> # A tibble: 8 x 5

#> pack rule var id value

#> <chr> <chr> <chr> <int> <lgl>

#> 1 data\_dims ncol .all 0 FALSE

#> 2 data\_dims nrow .all 0 TRUE

#> 3 vs\_1 nrow\_low .all 0 TRUE

#> 4 vs\_1 nrow\_high .all 0 TRUE

#> 5 group\_pack\_\_1 any\_cyl\_6 0.0 0 FALSE

#> 6 group\_pack\_\_1 any\_cyl\_6 0.1 0 TRUE

#> # … with 2 more rows

By default [expose()](https://echasnovski.github.io/ruler/reference/expose.html) guesses the pack type if ‘not-pack’ function is supplied. This behaviour has some edge cases but is useful for interactive use.

mtcars %>%

[expose](https://echasnovski.github.io/ruler/reference/expose.html)(

some\_data\_pack = . %>% summarise(nrow = [nrow](https://rdrr.io/r/base/nrow.html)(.) == 10),

some\_col\_pack = . %>% summarise\_at(vars(vs = "vs"), [rules](https://echasnovski.github.io/ruler/reference/rules.html)([is.character](https://rdrr.io/r/base/character.html)(.)))

) %>%

[get\_exposure](https://echasnovski.github.io/ruler/reference/exposure.html)()

#> Exposure

#>

#> Packs info:

#> # A tibble: 2 x 4

#> name type fun remove\_obeyers

#> <chr> <chr> <list> <lgl>

#> 1 some\_data\_pack data\_pack <data\_pck> TRUE

#> 2 some\_col\_pack col\_pack <col\_pack> TRUE

#>

#> Tidy data validation report:

#> # A tibble: 2 x 5

#> pack rule var id value

#> <chr> <chr> <chr> <int> <lgl>

#> 1 some\_data\_pack nrow .all 0 FALSE

#> 2 some\_col\_pack rule\_\_1 vs 0 FALSE

To write strict and robust code one can set .guess to FALSE.

mtcars %>%

[expose](https://echasnovski.github.io/ruler/reference/expose.html)(

some\_data\_pack = . %>% summarise(nrow = [nrow](https://rdrr.io/r/base/nrow.html)(.) == 10),

some\_col\_pack = . %>% summarise\_at(vars(vs = "vs"), [rules](https://echasnovski.github.io/ruler/reference/rules.html)([is.character](https://rdrr.io/r/base/character.html)(.))),

.guess = FALSE

) %>%

[get\_exposure](https://echasnovski.github.io/ruler/reference/exposure.html)()

#> Error in expose\_single.default(X[[i]], ...): There is unsupported class of rule pack.

**Acting after exposure**

General actions are recommended to be done with [act\_after\_exposure()](https://echasnovski.github.io/ruler/reference/act_after_exposure.html). It takes two arguments:

* .trigger - a function which takes the data with attached exposure and returns TRUE if some action should be made.
* .actor - a function which takes the same argument as .trigger and performs some action.

If trigger didn’t notify then the input data is returned untouched. Otherwise the output of .actor() is returned. **Note** that [act\_after\_exposure()](https://echasnovski.github.io/ruler/reference/act_after_exposure.html) is often used for creating side effects (printing, throwing error etc.) and in that case should invisibly return its input (to be able to use it with pipe).

trigger\_one\_pack <- function(.tbl) {

packs\_number <- .tbl %>%

[get\_packs\_info](https://echasnovski.github.io/ruler/reference/packs_info.html)() %>%

[nrow](https://rdrr.io/r/base/nrow.html)()

packs\_number > 1

}

actor\_one\_pack <- function(.tbl) {

[cat](https://rdrr.io/r/base/cat.html)("More than one pack was applied.\n")

[invisible](https://rdrr.io/r/base/invisible.html)(.tbl)

}

mtcars %>%

[expose](https://echasnovski.github.io/ruler/reference/expose.html)(my\_col\_packs, my\_row\_packs) %>%

[act\_after\_exposure](https://echasnovski.github.io/ruler/reference/act_after_exposure.html)(

.trigger = trigger\_one\_pack,

.actor = actor\_one\_pack

) %>%

[invisible](https://rdrr.io/r/base/invisible.html)()

#> More than one pack was applied.

ruler has function [assert\_any\_breaker()](https://echasnovski.github.io/ruler/reference/assert_any_breaker.html) which can notify about presence of any breaker in exposure.

mtcars %>%

[expose](https://echasnovski.github.io/ruler/reference/expose.html)(my\_col\_packs, my\_row\_packs) %>%

[assert\_any\_breaker](https://echasnovski.github.io/ruler/reference/assert_any_breaker.html)()

#> Breakers report

#> Tidy data validation report:

#> # A tibble: 4 x 5

#> pack rule var id value

#> <chr> <chr> <chr> <int> <lgl>

#> 1 sum\_bounds sum\_low cyl 0 FALSE

#> 2 sum\_bounds sum\_low carb 0 FALSE

#> 3 vs\_mean rule\_\_1 vs 0 FALSE

#> 4 row\_mean is\_common\_row\_mean .all 15 FALSE

#> Error: assert\_any\_breaker: Some breakers found in exposure.

After reading this post you will know:

* Most basic outlier detection techniques.
* A way to implement them using dplyr and ruler.
* A way to combine their results in order to obtain a new outlier detection method.
* A way to discover notion of “diamond quality” without prior knowledge of this topic (as a happy consequence of previous point).

**Overview**

We will perform an analysis with the goal to find not typical diamonds listed in [diamonds](http://ggplot2.tidyverse.org/reference/diamonds.html) dataset from [ggplot2](http://ggplot2.tidyverse.org/) package. Here one observation represents one diamond and is stored as a row in data frame.

The way we will do that is **by combining different outlier detection techniques to identify rows which are “strong outliers”**, i.e. which might by considered outliers based on several methods.

Packages required for this analysis:

library(dplyr)

library(tidyr)

library(ggplot2)

library(ruler)

**Outlier detection methods**

To do convenient outlier detection with ruler it is better to define notion of **non-outlier** in form of the rule “Observation **is not** an outlier if …”. This way actual outliers are considered as **rule breakers**, objects of interest of ruler package. **Note** that definition of non-outlier is essentially a definition of outlier because of total two possibilities.

**Z-score**

[Z-score](https://en.wikipedia.org/wiki/Standard_score), also called a standard score, of an observation is [broadly speaking] a distance from the population center measured in number of normalization units. The default choice for center is sample mean and for normalization unit is standard deviation.

⬛ *Observation is not an outlier based on z-score* if its absolute value of default z-score is lower then some threshold (popular choice is 3).

Here is the function for identifying non-outliers based on z-score:

isnt\_out\_z <- function(x, thres = 3, na.rm = TRUE) {

abs(x - mean(x, na.rm = na.rm)) <= thres \* sd(x, na.rm = na.rm)

}

It takes a numeric vector as input and returns logical vector of the same length indicating whether input value is a non-outlier.

**Z-score with MAD**

[Median Absolute Deviation](https://en.wikipedia.org/wiki/Median_absolute_deviation) is a robust normalization unit based on median as a population center. In order to use MAD “as a consistent estimator for the estimation of the standard deviation” one takes its value multiplied by a factor. This way base R function [mad](https://www.rdocumentation.org/packages/stats/versions/3.4.3/topics/mad?tap_a=5644-dce66f&tap_s=10907-287229) is implemented.

⬛ *Observation is not an outlier based on MAD* if its absolute value of z-score with median as center and MAD as normalization unit is lower then some threshold (popular choice is 3).

isnt\_out\_mad <- function(x, thres = 3, na.rm = TRUE) {

abs(x - median(x, na.rm = na.rm)) <= thres \* mad(x, na.rm = na.rm)

}

**Tukey’s fences**

[Tukey’s fences](https://en.wikipedia.org/wiki/Outlier#Tukey&%2339;s_fences) is a technique used in box plots. The non-outlier range is defined with \([Q\_1 – k(Q\_3 – Q\_1),~ Q\_3 + k(Q\_3 – Q\_1)]\), where \(Q\_1\) and \(Q\_3\) are the lower and upper quartiles respectively, \(k\) – some nonnegative constant (popular choice is 1.5).

⬛ *Observation is not an outlier based on Tukey’s fences* if its value lies in non-outlier range.

isnt\_out\_tukey <- function(x, k = 1.5, na.rm = TRUE) {

quar <- quantile(x, probs = c(0.25, 0.75), na.rm = na.rm)

iqr <- diff(quar)

(quar[1] - k \* iqr <= x) & (x <= quar[2] + k \* iqr)

}

**Mahalanobis distance**

All previous approaches were created for univariate numerical data. To detect outliers in multivariate case one can use [Mahalanobis distance](https://en.wikipedia.org/wiki/Mahalanobis_distance) to reduce to univariate case and then apply known techniques.

⬛ *Observation is not an outlier based on Mahalanobis distance* if its distance is not an outlier.

maha\_dist <- . %>% select\_if(is.numeric) %>%

mahalanobis(center = colMeans(.), cov = cov(.))

isnt\_out\_maha <- function(tbl, isnt\_out\_f, ...) {

tbl %>% maha\_dist() %>% isnt\_out\_f(...)

}

This function takes as input a data frame of interest (with possible non-numeric columns which are ignored) and function performing univariate outlier detection. It returns a logical vector of the same length as number of rows in input data frame.

To read more about practical usefulness of Mahalanobis distance in detecting outliers go to [Steffen’s very helpful post](https://www.steffenruefer.com/2016/12/outlier-detection-with-mahalanobis-distance/).

**Using dplyr and ruler**

**Definition of non-outlier row**

Package ruler, based on dplyr grammar of data manipulation, offers tools for validating the following data units: data as a whole, group [of rows] as a whole, column as a whole, row as a whole, cell. Our primary interest is row as a whole. However, using this framework, we can construct several approaches for definition of the non-outlier row:

1. *Row is not an outlier based on some column* if it doesn’t contain outlier (computed based on the target column) on the intersection with that column. In other words, first a univariate outlier detection is performed based solely on data from target column and then all rows containing non-outliers are named non-outlier rows.
2. *Row is not an outlier based on Mahalanobis distance* if its distance (computed based on the selected numeric columns) is not an outlier.
3. *Row is not an outlier based on grouping* if it is a part of a non-outlier group [of rows]. A group [of rows] is not an outlier if its summary value is not an outlier among summary values of other groups.

**Note** that all listed approached depend on the choice of the univariate outlier detection method. We will use all three previously listed univariate techniques.

isnt\_out\_funs <- funs(

z = isnt\_out\_z,

mad = isnt\_out\_mad,

tukey = isnt\_out\_tukey

)

**Implementation**

In ruler framework rules are defined in [packs](https://echasnovski.github.io/ruler/reference/rule-packs.html) (to learn more go to ruler [vignettes](https://echasnovski.github.io/ruler/articles/)).

**Column based non-outlier rows**

For diamonds dataset rules for column based non-outlier rows can be defined based on 7 numeric columns and 3 presented univariate detection methods. There is a convenient way of computing all them at once using [scoped variant](http://dplyr.tidyverse.org/reference/summarise_all.html) of dplyr::transmute():

diamonds %>% transmute\_if(is.numeric, isnt\_out\_funs)

## # A tibble: 53,940 x 21

## carat\_z depth\_z table\_z price\_z x\_z y\_z z\_z carat\_mad depth\_mad

##

## 1 TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE

## 2 TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE

## 3 TRUE FALSE FALSE TRUE TRUE TRUE TRUE TRUE FALSE

## 4 TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE

## 5 TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE

## # ... with 5.394e+04 more rows, and 12 more variables: table\_mad ,

## # price\_mad , x\_mad , y\_mad , z\_mad ,

## # carat\_tukey , depth\_tukey , table\_tukey ,

## # price\_tukey , x\_tukey , y\_tukey , z\_tukey

The result has outputs for 21 methods. Their names are of the form \_. So the name ‘carat\_z’ is interpreted as result of univariate method with name ‘z’ for column with name ‘carat’.

**Mahalanobis based non-outlier rows**

To define non-outlier rows based on Mahalanobis distance one should apply univariate method for distances computed for some subset of numeric columns. To simplify a little bit, we will one “subset” with all numeric columns and all listed methods:

diamonds %>%

transmute(maha = maha\_dist(.)) %>%

transmute\_at(vars(maha = maha), isnt\_out\_funs)

## # A tibble: 53,940 x 3

## maha\_z maha\_mad maha\_tukey

##

## 1 TRUE TRUE TRUE

## 2 TRUE FALSE FALSE

## 3 TRUE FALSE FALSE

## 4 TRUE TRUE TRUE

## 5 TRUE TRUE TRUE

## # ... with 5.394e+04 more rows

The result has outputs for 3 methods. Their names are considered as method names. **Note** that with this approach outlier rows are not only the ones far from multivariate center, but also the ones that are unnaturally close to it.

**Group based non-outlier rows**

Definition of non-outlier rows based on grouping depends on group summary function and univariate outlier detection method. As grouping column we will choose all non-numeric columns (cut, color and clarity) [united](http://tidyr.tidyverse.org/reference/unite.html) into one called group (for later easier imputation of non-outlier rows). As reasonable summary functions we will choose mean value of some numeric column (total of 7 functions):

data\_tbl <- diamonds %>%

unite(col = "group", cut, color, clarity)

compute\_group\_non\_outliers <- . %>%

# Compute per group mean values of columns

group\_by(group) %>%

summarise\_if(is.numeric, mean) %>%

ungroup() %>%

# Detect outliers among groups

mutate\_if(is.numeric, isnt\_out\_funs) %>%

# Remove unnecessary columns

select\_if(Negate(is.numeric))

data\_tbl %>% compute\_group\_non\_outliers()

## # A tibble: 276 x 22

## group carat\_z depth\_z table\_z price\_z x\_z y\_z z\_z carat\_mad

##

## 1 Fair\_D\_I1 FALSE TRUE TRUE TRUE TRUE TRUE TRUE FALSE

## 2 Fair\_D\_IF TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE

## 3 Fair\_D\_SI1 TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE

## 4 Fair\_D\_SI2 TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE

## 5 Fair\_D\_VS1 TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE

## # ... with 271 more rows, and 13 more variables: depth\_mad ,

## # table\_mad , price\_mad , x\_mad , y\_mad ,

## # z\_mad , carat\_tukey , depth\_tukey , table\_tukey ,

## # price\_tukey , x\_tukey , y\_tukey , z\_tukey

The result has outputs for 21 methods applied to the 276 groups. Their names are of the form \_. So the name ‘carat\_z’ is interpreted as result of method ‘z’ for summary function equal to mean value of ‘carat’ column. Column group defines names of the groupings.

**Exposure**

Column and Mahalanobis based definition of non-outlier rows can be expressed with [row packs](https://echasnovski.github.io/ruler/reference/row-pack.html) and group based – as [group packs](https://echasnovski.github.io/ruler/reference/group-pack.html).

row\_packs\_isnt\_out <- row\_packs(

# Non-outliers based on some column

column = . %>% transmute\_if(is.numeric, isnt\_out\_funs),

# Non-outliers based on Mahalanobis distance

maha = . %>% transmute(maha = maha\_dist(.)) %>%

transmute\_at(vars(maha = maha), isnt\_out\_funs)

)

group\_packs\_isnt\_out <- group\_packs(

# Non-outliers based on grouping

group = compute\_group\_non\_outliers,

.group\_vars = "group"

)

Application of all those packs is called [exposing process](https://echasnovski.github.io/ruler/reference/expose.html). The result is an [exposure](https://echasnovski.github.io/ruler/reference/exposure.html) from which we can extract [tidy data validation report](https://echasnovski.github.io/ruler/reference/ruler-report.html) using get\_report.

# Don't remove obeyers to compute total number of applied rules

full\_report <- data\_tbl %>%

expose(row\_packs\_isnt\_out, group\_packs\_isnt\_out,

.remove\_obeyers = FALSE) %>%

get\_report()

used\_rules <- full\_report %>%

distinct(pack, rule)

breaker\_report <- full\_report %>%

filter(!(value %in% TRUE))

used\_rules contains data about all definitions of non-outlier rows applied to data. They are encoded with combination of columns pack and rule.

breaker\_report contains data about data units that break certain rules. Packs column and maha has actual row numbers of data\_tbl listed in id column of report (for rows which should be considered as outliers).

On the other hand, pack group defines group pack and is represented in breaker\_report with id 0. To obtain row outliers based on grouping we need to expand those rows with information about rows in the data that belong to those groups. This can be done using dplyr::left\_join():

group\_breakers <- breaker\_report %>%

# Filter group packs

filter(pack == "group") %>%

# Expand rows by matching group with its rows

select(-id) %>%

left\_join(

y = data\_tbl %>% transmute(var = group, id = 1:n()),

by = "var"

) %>%

select(pack, rule, var, id, value)

outliers <- bind\_rows(

breaker\_report %>% filter(pack != "group"),

group\_breakers

) %>%

select(pack, rule, id)

# Not all group based definitions resulted with outliers

outliers %>%

count(pack, rule) %>%

filter(pack == "group") %>%

print(n = Inf)

## # A tibble: 13 x 3

## pack rule n

##

## 1 group carat\_mad 37

## 2 group carat\_tukey 37

## 3 group carat\_z 29

## 4 group depth\_mad 1093

## 5 group depth\_tukey 1016

## 6 group depth\_z 156

## 7 group price\_mad 209

## 8 group price\_tukey 1146

## 9 group price\_z 44

## 10 group table\_mad 920

## 11 group table\_tukey 8

## 12 group table\_z 7

## 13 group z\_z 23

Tibble outliers contains data about outlier rows. Combination of columns pack and rule defines non-outlier/outlier definition approach and column id defines row number of input data frame that should be considered an outlier based on the definition.

Definitions with most outliers are as follows:

outliers %>%

count(pack, rule, sort = TRUE)

## # A tibble: 37 x 3

## pack rule n

##

## 1 maha maha\_mad 6329

## 2 maha maha\_tukey 5511

## 3 column price\_mad 5386

## 4 column price\_tukey 3540

## 5 column table\_mad 2560

## # ... with 32 more rows

Two out of three Mahalanobis based definition yielded the most row outliers.

**Combination**

Given outliers data frame, one can do whatever he/she wants to identify outliers. Here we will use the basic combination approach based on average score.

*Combined outlier detection score* for certain row can be defined as **share of applied methods that tagged it as outlier**. Alternatively one can define it just as number of those methods as it will only change absolute value of the result and not the order.

outlier\_score <- outliers %>%

group\_by(id) %>%

# nrow(used\_rules) equals total number of applied methods

summarise(score = n() / nrow(used\_rules))

# Top 10 outliers

outlier\_score %>% arrange(desc(score)) %>% slice(1:10)

## # A tibble: 10 x 2

## id score

##

## 1 26432 0.5777778

## 2 27416 0.5777778

## 3 27631 0.5777778

## 4 27131 0.4666667

## 5 23645 0.4222222

## 6 26445 0.4222222

## 7 26745 0.4000000

## 8 27430 0.4000000

## 9 15952 0.3777778

## 10 17197 0.3777778

Finally we will tag those rows as **strong outliers** which has score more than 0.2 (subjective threshold which should be researched more).

diam\_tbl <- diamonds %>%

mutate(id = 1:n()) %>%

left\_join(y = outlier\_score, by = "id") %>%

mutate(

score = coalesce(score, 0),

is\_out = if\_else(score > 0.2, "Outlier", "Not outlier")

)

# Total number of outliers

sum(diam\_tbl$score > 0.2)

## [1] 161

Tibble diam\_tbl is basically the diamonds but with three more columns: id for row number, score for combined outlier score and is\_out for non-outlier/outlier tag.

Plots illustrating strong outliers:

theme\_set(theme\_bw())

plot\_outliers <- function(tbl, x, y, facet\_var) {

tbl %>%

arrange(is\_out) %>%

ggplot(aes\_string(x, y, colour = "is\_out")) +

geom\_point() +

facet\_wrap(facets = facet\_var) +

scale\_colour\_manual(values = c("#AAAAAA", "#004080")) +

guides(colour = guide\_legend(title = NULL,

override.aes = list(size = 4))) +

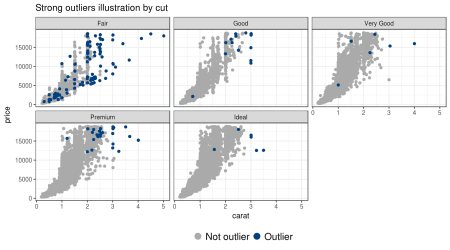
labs(title = paste0("Strong outliers illustration by ", facet\_var)) +

theme(legend.position = "bottom",

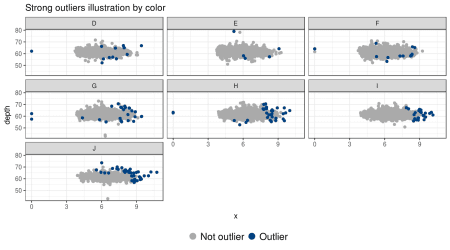
legend.text = element\_text(size = 14))

}

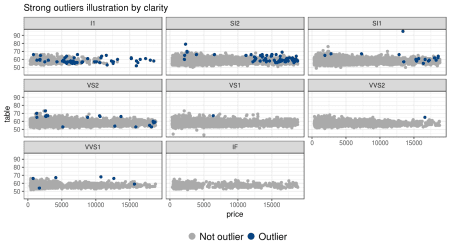
diam\_tbl %>% plot\_outliers("carat", "price", facet\_var = "cut")



diam\_tbl %>% plot\_outliers("x", "depth", facet\_var = "color")



diam\_tbl %>% plot\_outliers("price", "table", facet\_var = "clarity")



Based on those plots we see the complicated nature of “strong outliers”. They are not necessary located “on the edge” of two-dimensional scatter plots, but most extreme cases are tagged as outliers.

Also one interesting observation: most outliers are concentrated in the combination of “Fair” cut, “J” colour and “I1” clarity which are worst options among their features. The reason of this effect is group-based definitions of non-outliers which tagged certain groups more than others:

breaker\_report %>%

filter(pack == "group") %>%

count(var, sort = TRUE) %>%

print(n = 10)

## # A tibble: 47 x 2

## var n

##

## 1 Fair\_D\_I1 7

## 2 Fair\_J\_I1 7

## 3 Fair\_H\_VVS1 6

## 4 Ideal\_J\_I1 6

## 5 Fair\_J\_VVS1 5

## 6 Fair\_G\_VVS1 4

## 7 Fair\_D\_VVS1 3

## 8 Fair\_E\_I1 3

## 9 Fair\_F\_I1 3

## 10 Fair\_H\_I1 3

## # ... with 37 more rows

Here we see that “Fair” cut is among majority of top breaker groups. There are also some interesting combinations: Fair\_D\_I1 (“worst”-“best”-“worst”), Fair\_J\_I1 (“worst”-“worst”-“worst”), Ideal\_J\_I1 (“best”-“worst”-“worst”).

This fact might be interpreted as **suggested combined outlier detection approach discovered notion of diamond quality without prior knowledge about it**.

**Conclusions**

* Using only basic outlier detection methods one can achieve insightful results by combining them. Observations which are tagged as outlier by more than some threshold number of methods might be named as “strong outliers”. Those should be considered as outliers based on the whole data rather then on separate features.
* With ruler combining results of several outlier detection methods is straightforward due to the format of tidy data validation report.
* Suggested “strong outlier” observations in diamonds dataset are not only those with extreme numerical values but also ones based on quality of diamonds. This is achieved without prior knowledge of “diamond quality” notion.

[sessionInfo()](http://www.questionflow.org/2017/12/26/combined-outlier-detection-with-dplyr-and-ruler/#sessionInfo)

sessionInfo()

## R version 3.4.3 (2017-11-30)

## Platform: x86\_64-pc-linux-gnu (64-bit)

## Running under: Ubuntu 16.04.3 LTS

##

## Matrix products: default

## BLAS: /usr/lib/openblas-base/libblas.so.3

## LAPACK: /usr/lib/libopenblasp-r0.2.18.so

##

## locale:

## [1] LC\_CTYPE=ru\_UA.UTF-8 LC\_NUMERIC=C

## [3] LC\_TIME=ru\_UA.UTF-8 LC\_COLLATE=ru\_UA.UTF-8

## [5] LC\_MONETARY=ru\_UA.UTF-8 LC\_MESSAGES=ru\_UA.UTF-8

## [7] LC\_PAPER=ru\_UA.UTF-8 LC\_NAME=C

## [9] LC\_ADDRESS=C LC\_TELEPHONE=C

## [11] LC\_MEASUREMENT=ru\_UA.UTF-8 LC\_IDENTIFICATION=C

##

## attached base packages:

## [1] methods stats graphics grDevices utils datasets base

##

## other attached packages:

## [1] bindrcpp\_0.2 ruler\_0.1.0 ggplot2\_2.2.1 tidyr\_0.7.2 dplyr\_0.7.4

##

## loaded via a namespace (and not attached):

## [1] Rcpp\_0.12.14 knitr\_1.17 bindr\_0.1 magrittr\_1.5

## [5] tidyselect\_0.2.3 munsell\_0.4.3 colorspace\_1.3-2 R6\_2.2.2

## [9] rlang\_0.1.4 plyr\_1.8.4 stringr\_1.2.0 tools\_3.4.3

## [13] grid\_3.4.3 gtable\_0.2.0 htmltools\_0.3.6 lazyeval\_0.2.1

## [17] yaml\_2.1.16 rprojroot\_1.2 digest\_0.6.13 assertthat\_0.2.0

## [21] tibble\_1.3.4 bookdown\_0.5 purrr\_0.2.4 glue\_1.2.0

## [25] evaluate\_0.10.1 rmarkdown\_1.8 blogdown\_0.4 labeling\_0.3

## [29] stringi\_1.1.6 keyholder\_0.1.1 compiler\_3.4.3 scales\_0.5.0

## [33] backports\_1.1.2 pkgconfig\_2.0.1