Intro to R

Data Cleaning

Data Cleaning

In general, data cleaning is a process of investigating your data for inaccuracies, or recoding it in a way that makes it more manageable.

MOST IMPORTANT RULE - LOOK AT YOUR DATA!

Dealing with Missing Data

Missing data types

One of the most important aspects of data cleaning is missing values.

Types of “missing” data:

- `NA` - general missing data
- `NaN` - stands for “**N**ot **a** **N**umber”, happens when you do 0/0.
- `Inf` and `-Inf` - Infinity, happens when you take a positive number (or negative number) by 0.

Finding Missing data

Each missing data type has a function that returns `TRUE` if the data is missing:

- `NA` - `is.na`
- `NaN` - `is.nan`
- `Inf` and `-Inf` - `is.infinite`
- `is.finite` returns `FALSE` for all missing data and `TRUE` for non-missing

Useful checking functions

- `is.na` - is `TRUE` if the data is `FALSE` otherwise
- `!` - negation (NOT)
  - if `is.na(x)` is `TRUE`, then `!is.na(x)` is `FALSE`
- `all` takes in a `logical` and will be `TRUE` if ALL are `TRUE`
  - `all(!is.na(x))` - are all values of `x` NOT NA
- `any` will be `TRUE` if ANY are true
  - `any(is.na(x))` - do we have any `NA`'s in `x`?

```
A = c(1, 2, 4, NA)
B = c(1, 2, 3, 4)
any(is.na(A)) # are there any NAs - YES/TRUE
```

```
[1] TRUE
```

```
any(is.na(B)) # are there any NAs- NO/FALSE
```

```
[1] FALSE
```

```
A = c(1, 2, 4, NA)
B = c(1, 2, 3, 4)
all(is.na(A)) # are all values NA - NO/FALSE
```

```
[1] FALSE
```

Complete.cases

`complete.cases` - returns `TRUE` if EVERY value of a row is NOT `NA` - very stringent condition - `FALSE` missing one value (even if not important)

```
complete.cases(mtcars)
```

```
 [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRU
[16] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRU
[31] TRUE TRUE
```

naniar

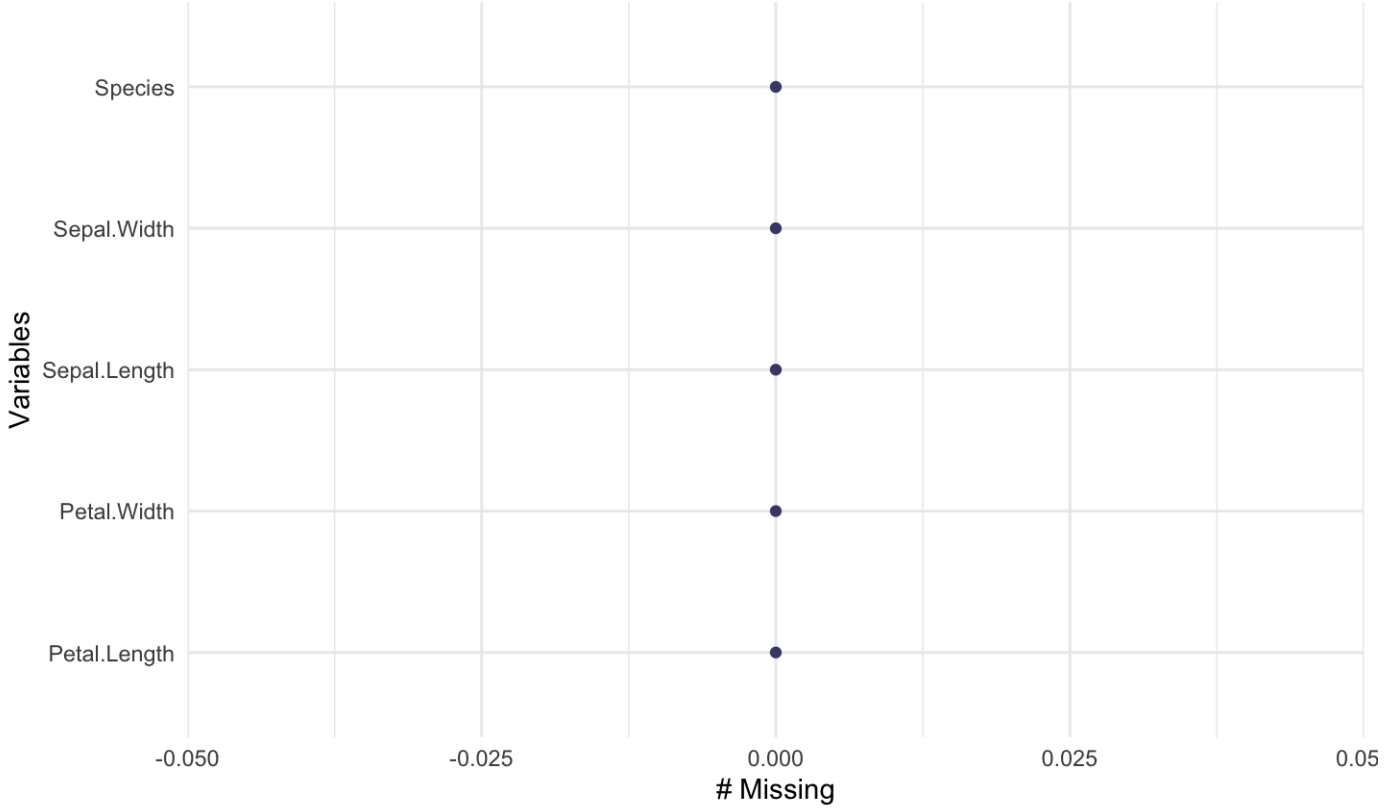
Sometimes you need to look at lots of data though… the `naniar` package is a good option.

```
#install.packageS(naniar)
library(naniar)
x = c(0, NA, 2, 3, 4, -0.5, 0.2)
naniar::pct_complete(x)
```

```
[1] 85.71429
```

Naniar plots

```
naniar::gg_miss_var(iris)
```



Missing Data with Logicals

Recall that mathematical operations with `NA` will result in `NA`s

This is also true for logicals. This is a good thing. The NA data could be `> 2` or not, we don't know, so `R` says there is no `TRUE` or `FALSE`, so that is missing.

```
x = c(0, NA, 2, 3, 4, -0.5, 0.2)
x > 2
```

```
[1] FALSE    NA FALSE  TRUE  TRUE FALSE FALSE
```

filter() and missing data

Be careful with missing data using subsetting:

filter() removes missing values by default. To keep them need to add is.na():

```
x # looks like the 1st and 3rd element should be TRUE
```

```
[1]  0.0   NA  2.0  3.0  4.0 -0.5  0.2
```

```
x %in% c(0, 2) # uh oh - not good!
```

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

```
x %in% c(0, 2) | is.na(x) # do this
```

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

tidyr::drop_na

This function will drop rows with **any** missing data in **any** column.

```
df <-tibble(x = c(0, NA, 2, 0.2),
            y = c(NA, 1, 6, NA))
df
```

```
# A tibble: 4 x 2
      x     y
  <dbl> <dbl>
1   0      NA
2  NA       1
3   2       6
4   0.2    NA
```

```
drop_na(df)
```

```
# A tibble: 1 x 2
      x     y
  <dbl> <dbl>
1     2     6
```

Missing data with Tables and Tabulations

Useful checking functions

- `table(x, useNA = "ifany")` - will have row NA if there are NA values
- `table(x, useNA = "always")` - will have row NA even if there are no NA values

Here we will use `table` to make tabulations of the data. Look at `?table` to see options for missing data.

```
z = c("A", "B", "A", "B")
table(z, useNA = "ifany")
```

```
z
A B
2 2
```

```
table(z, useNA = "always")
```

```
z
   A    B <NA>
   2    2    0
```

Lab Part 1

lab part 1

Website

Recoding Variables

Example of Recoding

Let's say gender was coded as Male, M, m, Female, F, f. Using Excel to find all of these would be a matter of filtering and changing all by hand or using if statements. This can be hectic!

In `dplyr` you can use the `recode` function:

```
data = data %>%
  mutate(gender = recode(gender, M = "Male", m = "Male", Man = "Male"))
```

Or use `ifelse()` or `case_when()`.

Separating columns based on a separator

- From `tidyr`, you can split a data set into multiple columns:

```
df = tibble(x = c("I really", "like writing", "R code programs"))
```

```
df %>% separate(x, into = c("first", "second", "third"))
```

```
Warning: Expected 3 pieces. Missing pieces filled with `NA` in 2 rows [1, 2].
```

```
# A tibble: 3 x 3
  first second  third
  <chr> <chr>   <chr>
1 I     really  <NA>
2 like  writing <NA>
3 R     code    programs
```

Separating columns based on a separator

You can specify the separator with `sep`. * `extra = "merge"` will not drop data.

```
df %>% separate(x, into = c("first", "second", "third"),
                extra = "merge", sep = " ")
```

```
Warning: Expected 3 pieces. Missing pieces filled with `NA` in 2 rows [1, 2].
```

```
# A tibble: 3 x 3
  first second  third
  <chr> <chr>   <chr>
1 I     really  <NA>
2 like  writing <NA>
3 R     code    programs
```

Uniting columns based on a separator

- From `tidyr`, you can unite:

```
df = tibble(id = rep(1:5, 3), visit = rep(1:3, each = 5))
head(df, 4)
```

```
# A tibble: 4 x 2
     id visit
  <int> <int>
1     1     1
2     2     1
3     3     1
4     4     1
```

```
df_united <- df %>% unite(col = "unique_id", id, visit, sep = "_")
head(df_united, 4)
```

```
# A tibble: 4 x 1
  unique_id
  <chr>
1 1_1
2 2_1
3 3_1
4 4_1
```

Strings functions

Splitting/Find/Replace and Regular Expressions

- R can do much more than find exact matches for a whole string!

The `stringr` package

The `stringr` package:

- Makes string manipulation more intuitive
- We will not cover `grep` or `gsub` - base R functions
    - are used on forums for answers
- Almost all functions start with `str_*`

String Splitting

- `str_split(string, pattern)` - splits strings up - returns list!

```
library(stringr)
x <- c("I really like writing R code")
df = tibble(x = c("I really", "like writing", "R code programs"))
y <- unlist(str_split(x, " "))
y
```

```
[1] "I"       "really"  "like"    "writing" "R"       "code"
```

```
length(y)
```

```
[1] 6
```

A bit on Regular Expressions

- http://www.regular-expressions.info/reference.html
- They can use to match a large number of strings in one statement
- `.` matches any single character
- `*` means repeat as many (even if 0) more times the last character
- `?` makes the last thing optional
- `^` matches start of vector `^a` - starts with “a”
- `$` matches end of vector `b$` - ends with “b”

Let's look at modifiers for `stringr`

```
?modifiers
```

- `fixed` - match everything exactly
- `ignore_case` is an option to not have to use `tolower`

Using a fixed expression

One example case is when you want to split on a period “.”. In regular expressions `.` means **ANY** character, so we need to specify that we want R to interpret “.” as simply a period.

```
str_split("I.like.strings", ".")
```

```
[[1]]
 [1] "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" ""
```

```
str_split("I.like.strings", fixed("."))
```

```
[[1]]
[1] "I"       "like"    "strings"
```

```
str_split("I.like.strings", "\\.")
```

```
[[1]]
[1] "I"       "like"    "strings"
```

'Find' functions: `stringr`

`str_detect`, and `str_replace` search for matches to argument pattern within each element of a character vector (not data frame or tibble!).

- `str_detect` - returns `TRUE` if `pattern` is found
- `str_replace` - replaces `pattern` with `replacement`

Download Salary FY2014 Data

From https://data.baltimorecity.gov/City-Government/Baltimore-City-Employee-Salaries-FY2015/nsfe-bg53, from https://data.baltimorecity.gov/api/views/nsfe-bg53/rows.csv

Read the CSV into R `Sal`:

```
Sal = jhur::read_salaries() # or
```

```
head(Sal)
```

```
# A tibble: 6 x 7
  name       JobTitle       AgencyID Agency       HireDate AnnualSalary GrossP
  <chr>      <chr>          <chr>    <chr>        <chr>    <chr>        <chr>
1 Aaron,Pat… Facilities/Of… A03031   OED-Employm… 10/24/1… $55314.00    $53626
2 Aaron,Pet… ASSISTANT STA… A29045   States Atto… 09/25/2… $74000.00    $73000
3 Abaineh,Y… EPIDEMIOLOGIST A65026   HLTH-Health… 07/23/2… $64500.00    $64403
4 Abbene,An… POLICE OFFICER A99005   Police Depa… 07/24/2… $46309.00    $59620
5 Abbey,Emm… CONTRACT SERV… A40001   M-R Info Te… 05/01/2… $60060.00    $54059
6 Abbott-Co… CONTRACT SERV… A90005   TRANS-Traff… 11/28/2… $42702.00    $20250
```

'Find' functions: finding values: `stringr`

```
Sal %>% filter(str_detect(name, "Rawlings"))
```

```
# A tibble: 3 x 7
  name          JobTitle     AgencyID Agency       HireDate AnnualSalary GrossP
  <chr>         <chr>        <chr>    <chr>        <chr>    <chr>        <chr>
1 Rawlings,Ke… EMERGENCY D… A40302   M-R Info Te… 01/06/2… $48940.00    $73356
2 Rawlings,Pa… COMMUNITY A… A04015   R&P-Recreat… 12/10/2… $19802.00    $10443
3 Rawlings-Bl… MAYOR        A01001   Mayors Offi… 12/07/1… $167449.00   $16524
```

Showing differnce in `str_replace` and `str_replace_all`

`str_replace` replaces only the first instance.

```
head(Sal$Name, 2)
```

```
Warning: Unknown or uninitialised column: `Name`.
```

```
NULL
```

```
head(str_replace(Sal$name, "a", "j"),2)
```

```
[1] "Ajron,Patricia G" "Ajron,Petra L"
```

`str_replace` replaces all instances.

```
head(str_replace_all(Sal$name, "a", "j"), 2)
```

```
[1] "Ajron,Pjtricij G" "Ajron,Petrj L"
```

Pasting strings with `paste` and `paste0`

Paste can be very useful for joining vectors together:

```
paste("Visit", 1:5, sep = "_")
```

```
[1] "Visit_1" "Visit_2" "Visit_3" "Visit_4" "Visit_5"
```

```
paste("Visit", 1:5, sep = "_", collapse = "_")
```

```
[1] "Visit_1_Visit_2_Visit_3_Visit_4_Visit_5"
```

```
# and paste0 can be even simpler see ?paste0
paste0("Visit",1:5) # no space!
```

```
[1] "Visit1" "Visit2" "Visit3" "Visit4" "Visit5"
```

Lab Part 2

lab part 2

Website

!= # Before Cleaning - Subsetting with Brackets =>

=>

=> => =>

Extra Slides

Using Regular Expressions

- Look for any name that starts with:
  - Payne at the beginning,
  - Leonard and then an S
  - Spence then capital C

```
head(str_subset( Sal$name, "^Payne.*"), 3)
```

```
[1] "Payne El,Boaz L"          "Payne El,Jackie"
[3] "Payne Johnson,Nickole A"
```

```
head(str_subset( Sal$name, "Leonard.?S"))
```

```
[1] "Payne,Leonard S"      "Szumlanski,Leonard S"
```

```
head(str_subset( Sal$name, "Spence.*C.*"))
```

```
[1] "Spencer,Charles A"  "Spencer,Clarence W" "Spencer,Michael C"
```

Comparison of `stringr` to base R - not covered

Splitting Strings

Substringing

`stringr`

- `str_split(string, pattern)` - splits strings up - returns list!

Splitting String:

In `stringr`, `str_split` splits a vector on a string into a `list`

```
x <- c("I really", "like writing", "R code programs")
y <- stringr::str_split(x, pattern =  " ") # returns a list
y
```

```
[[1]]
[1] "I"      "really"

[[2]]
[1] "like"    "writing"

[[3]]
[1] "R"        "code"     "programs"
```

str_extract

`str_extract` extracts matched strings - `\\d` searches for DIGITS/numbers

```
head(Sal$AgencyID)
```

```
[1] "A03031" "A29045" "A65026" "A99005" "A40001" "A90005"
```

```
head(str_extract(Sal$AgencyID, "\\d"))
```

```
[1] "0" "2" "6" "9" "4" "9"
```

'Find' functions: stringr compared to base R

Base R does not use these functions. Here is a "translator" of the `stringr` function to base R functions

- `str_detect` - similar to `grepl` (return logical)
- `grep(value = FALSE)` is similar to `which(str_detect())`
- `str_subset` - similar to `grep(value = TRUE)` - return value of matched
- `str_replace` - similar to `sub` - replace one time
- `str_replace_all` - similar to `gsub` - replace many times

Important Comparisons

Base R:

- Argument order is `(pattern, x)`
- Uses option `(fixed = TRUE)`

`stringr`

- Argument order is `(string, pattern)` aka `(x, pattern)`
- Uses function `fixed(pattern)`

'Find' functions: Finding Indices

These are the indices where the pattern match occurs:

```
grep("Rawlings",Sal$Name)
```

```
Warning: Unknown or uninitialised column: `Name`.
```

```
integer(0)
```

```
which(grepl("Rawlings", Sal$Name))
```

```
Warning: Unknown or uninitialised column: `Name`.
```

```
integer(0)
```

```
which(str_detect(Sal$Name, "Rawlings"))
```

```
Warning: Unknown or uninitialised column: `Name`.
```

```
integer(0)
```

'Find' functions: Finding Logicals

These are the indices where the pattern match occurs:

```
head(grepl("Rawlings",Sal$Name))
```

```
Warning: Unknown or uninitialised column: `Name`.
```

```
logical(0)
```

```
head(str_detect(Sal$Name, "Rawlings"))
```

```
Warning: Unknown or uninitialised column: `Name`.
```

```
logical(0)
```

'Find' functions: finding values, base R

```
grep("Rawlings",Sal$Name,value=TRUE)
```

```
Warning: Unknown or uninitialised column: `Name`.

character(0)
```

```
Sal[grep("Rawlings",Sal$Name),]
```

```
Warning: Unknown or uninitialised column: `Name`.

# A tibble: 0 x 7
# … with 7 variables: name <chr>, JobTitle <chr>, AgencyID <chr>, Agency <chr>
#   HireDate <chr>, AnnualSalary <chr>, GrossPay <chr>
```

Showing differnce in `str_extract`

`str_extract` extracts just the matched string

```
ss = str_extract(Sal$Name, "Rawling")
```

```
Warning: Unknown or uninitialised column: `Name`.
```

```
head(ss)
```

```
character(0)
```

```
ss[ !is.na(ss)]
```

```
character(0)
```

Showing differnce in `str_extract` and `str_extract_all`

`str_extract_all` extracts all the matched strings

```
head(str_extract(Sal$AgencyID, "\\d"))
```

```
[1] "0" "2" "6" "9" "4" "9"
```

```
head(str_extract_all(Sal$AgencyID, "\\d"), 2)
```

```
[[1]]
[1] "0" "3" "0" "3" "1"

[[2]]
[1] "2" "9" "0" "4" "5"
```

Using Regular Expressions

- Look for any name that starts with:
  - Payne at the beginning,
  - Leonard and then an S
  - Spence then capital C

```
head(grep("^Payne.*", x = Sal$name, value = TRUE), 3)
```

```
[1] "Payne El,Boaz L"          "Payne El,Jackie"
[3] "Payne Johnson,Nickole A"
```

```
head(grep("Leonard.?S", x = Sal$name, value = TRUE))
```

```
[1] "Payne,Leonard S"      "Szumlanski,Leonard S"
```

```
head(grep("Spence.*C.*", x = Sal$name, value = TRUE))
```

```
[1] "Spencer,Charles A"  "Spencer,Clarence W" "Spencer,Michael C"
```

Using Regular Expressions: `stringr`

```
head(str_subset( Sal$name, "^Payne.*"), 3)
```

```
[1] "Payne El,Boaz L"          "Payne El,Jackie"
[3] "Payne Johnson,Nickole A"
```

```
head(str_subset( Sal$name, "Leonard.?S"))
```

```
[1] "Payne,Leonard S"      "Szumlanski,Leonard S"
```

```
head(str_subset( Sal$name, "Spence.*C.*"))
```

```
[1] "Spencer,Charles A"  "Spencer,Clarence W" "Spencer,Michael C"
```

Replace

Let’s say we wanted to sort the data set by Annual Salary:

```
class(Sal$AnnualSalary)
```

```
[1] "character"
```

```
sort(c("1", "2", "10")) #  not sort correctly (order simply ranks the data)
```

```
[1] "1"  "10" "2"
```

```
order(c("1", "2", "10"))
```

```
[1] 1 3 2
```

Replace

So we must change the annual pay into a numeric:

```
head(Sal$AnnualSalary, 4)
```

```
[1] "$55314.00" "$74000.00" "$64500.00" "$46309.00"
```

```
head(as.numeric(Sal$AnnualSalary), 4)
```

```
Warning in head(as.numeric(Sal$AnnualSalary), 4): NAs introduced by coercion
```

```
[1] NA NA NA NA
```

R didn't like the `$` so it thought turned them all to `NA`.

`sub()` and `gsub()` can do the replacing part in base R.

Replacing and subbing

Now we can replace the `$` with nothing (used `fixed=TRUE` because `$` means ending):

```
Sal$AnnualSalary <- as.numeric(gsub(pattern = "$", replacement="",
                                    Sal$AnnualSalary, fixed=TRUE))
Sal <- Sal[order(Sal$AnnualSalary, decreasing=TRUE), ]
Sal[1:5, c("name", "AnnualSalary", "JobTitle")]
```

```
# A tibble: 5 x 3
  name              AnnualSalary JobTitle
  <chr>                    <dbl> <chr>
1 Mosby,Marilyn J         238772 STATE'S ATTORNEY
2 Batts,Anthony W         211785 Police Commissioner
3 Wen,Leana               200000 Executive Director III
4 Raymond,Henry J         192500 Executive Director III
5 Swift,Michael           187200 CONTRACT SERV SPEC II
```

Replacing and subbing: `stringr`

We can do the same thing (with 2 piping operations!) in dplyr

```
dplyr_sal = Sal
dplyr_sal = dplyr_sal %>% mutate(
  AnnualSalary = AnnualSalary %>%
    str_replace(
      fixed("$"),
      "") %>%
    as.numeric) %>%
  arrange(desc(AnnualSalary))
check_Sal = Sal
rownames(check_Sal) = NULL
all.equal(check_Sal, dplyr_sal)
```

```
[1] TRUE
```

Website

Website

Extra slides

Creating Two-way Tables

A two-way table. If you pass in 2 vectors, `table` creates a 2-dimensional table.

```
tab <- table(c(0, 1, 2, 3, 2, 3, 3, 2,2, 3),
             c(0, 1, 2, 3, 2, 3, 3, 4, 4, 3),
              useNA = "always")
tab
```

```
      0 1 2 3 4 <NA>
  0   1 0 0 0 0    0
  1   0 1 0 0 0    0
  2   0 0 2 0 2    0
  3   0 0 0 4 0    0
  <NA> 0 0 0 0 0    0
```

Creating Two-way Tables

```
tab_df = tibble(x = c(0, 1, 2, 3, 2, 3, 3, 2,2, 3),
                y = c(0, 1, 2, 3, 2, 3, 3, 4, 4, 3))
tab_df %>% count(x, y)
```

```
# A tibble: 5 x 3
      x     y     n
  <dbl> <dbl> <int>
1     0     0     1
2     1     1     1
3     2     2     2
4     2     4     2
5     3     3     4
```

Creating Two-way Tables

```
tab_df %>%
  count(x, y) %>%
  group_by(x) %>% mutate(pct_x = n / sum(n))
```

```
# A tibble: 5 x 4
# Groups:   x [4]
      x     y     n pct_x
  <dbl> <dbl> <int> <dbl>
1     0     0     1   1
2     1     1     1   1
3     2     2     2   0.5
4     2     4     2   0.5
5     3     3     4   1
```

Creating Two-way Tables

```
library(scales)
tab_df %>%
  count(x, y) %>%
  group_by(x) %>% mutate(pct_x = percent(n / sum(n)))
```

```
# A tibble: 5 x 4
# Groups:   x [4]
      x     y     n pct_x
  <dbl> <dbl> <int> <chr>
1     0     0     1 100%
2     1     1     1 100%
3     2     2     2 50%
4     2     4     2 50%
5     3     3     4 100%
```