Summary

Last Updated: 05, October, 2023 at 08:35

Contents

Resource: cheat Sheets

This page features a large number of cheat sheets on different topics: Cheat Sheets

R Basics

The `if` statement: example

```r
my_number <-12

if (my_number < 20){
  x <- sprintf('%i is less than 20', my_number)
  print(x)
}
```

```
## [1] "12 is less than 20"
```

The `for` loop: example

```
my_vector <- runif(5)
for (x in my_vector) {
  y <- x * 3
  print(y)
}
```

```
## [1] 1.847155
## [1] 2.080313
## [1] 1.754968
## [1] 1.766776
## [1] 2.889363
```

One very common use of the `for` loop is to iterate a bit of code exactly n times.

```
number_of_time_i_want_to_repeat_this <-10
for (x in 1:10) {
  print('This is being repeated!')
}
```

```
## [1] "This is being repeated!"
## [1] "This is being repeated!"
## [1] "This is being repeated!"
## [1] "This is being repeated!"
## [1] "This is being repeated!"
## [1] "This is being repeated!"
## [1] "This is being repeated!"
## [1] "This is being repeated!"
## [1] "This is being repeated!"
## [1] "This is being repeated!"
```

The `while` loop: example

```
i <- 1
while (i < 6) {
  print(i)
  i <- i + 1
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
```

Reading data

Reading data from excel using readxl

```
library(tidyverse)
library(readxl)
# Reading the first sheet
data <-read_excel("data/transit-data.xlsx")
# Reading a specific range from a specific sheet
data <-read_excel("data/transit-data.xlsx", sheet = 'info', range = 'B1:C7')
```

Reading data from text files

Reading comma separated files

```
data <- read_csv('data/pakistan_intellectual_capital.csv')
```

```
## New names:
## Rows: 1142 Columns: 13
## -- Column specification
## -------------------------------------------------------- Delimiter: "," chr
## (10): Teacher Name, University Currently Teaching, Department, Province ... dbl
## (3): ...1, S#, Year
## i Use `spec()` to retrieve the full column specification for this data. i
## Specify the column types or set `show_col_types = FALSE` to quiet this message.
## * `` -> `...1`
```

Reading tab separated files

```
data <- read_tsv('data/films.dat')
```

```
## Rows: 100 Columns: 6
## -- Column specification --------------------------------------------------------
## Delimiter: "\t"
## chr (1): Title
## dbl (5): Year, Length, Cast, Rating, Description
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

Reading files seperated by specific charactor

```
# The columns of this file are separated by a space.
data <- read_delim('data/wages1833bis.csv', delim = ' ')
```

```
## New names:
## Rows: 51 Columns: 6
## -- Column specification
## -------------------------------------------------------- Delimiter: " " dbl
## (6): ...1, age, mnum, mwage, fnum, fwage
## i Use `spec()` to retrieve the full column specification for this data. i
## Specify the column types or set `show_col_types = FALSE` to quiet this message.
## * `` -> `...1`
```

Some interesting options when using `read_csv()`, `read_tsv()`, or `read_delim()`

No column names?

If data has no column names, use `col_names = FALSE`

```
read_csv("1,2,3\n4,5,6", col_names = FALSE)
```

```
## Rows: 2 Columns: 3
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl (3): X1, X2, X3
##
```

```
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
## # A tibble: 2 x 3
##      X1    X2    X3
##   <dbl> <dbl> <dbl>
## 1     1     2     3
## 2     4     5     6
```

You can also directly set the column names in this case.

```
read_csv("1,2,3\n4,5,6", col_names = c("x", "y", "z"))
```

```
## Rows: 2 Columns: 3
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl (3): x, y, z
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
## # A tibble: 2 x 3
##       x     y     z
##   <dbl> <dbl> <dbl>
## 1     1     2     3
## 2     4     5     6
```

Specifying missing data

Data cleaning and data operations

Another option that commonly needs tweaking is `na`: this specifies the value (or values) that are used to represent missing values in your file:

```
read_csv("a,b,c\n1,2,.", na = ".")
```

```
## Rows: 1 Columns: 3
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl (2): a, b
## lgl (1): c
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
## # A tibble: 1 x 3
##       a     b c
##   <dbl> <dbl> <lgl>
## 1     1     2 NA
```

Renaming variables to R acceptable format

Human readable names are very handy for coding. **Indeed, in general, only dots and underscores are allowed in variable names.** There is a quick way to solve this.

```
data <-read_excel("data/transit-data.xlsx", sheet='transport data', skip = 1)
colnames(data)
```

```
## [1] "sender location"    "sender latitude"    "sender longitude"
## [4] "receiver location"  "receiver latitude"  "receiver longitude"
## [7] "date"               "number of items"
```

```
colnames(data) <- make.names(colnames(data))
colnames(data)
```

```
## [1] "sender.location"    "sender.latitude"    "sender.longitude"
## [4] "receiver.location"  "receiver.latitude"  "receiver.longitude"
## [7] "date"               "number.of.items"
```

Creating new variables using `mutate()`

```
data <- read_csv('data/pizzasize.csv')
```

```
## Rows: 250 Columns: 5
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (3): Store, CrustDescription, Topping
## dbl (2): ID, Diameter
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
data <- mutate(data, surface = 3.14 * (Diameter/2)^2)
```

Selecting columns using `select()`

```
data <- read_excel("data/transit-data.xlsx", sheet = 'transport data', skip=1)
colnames(data) <- make.names(colnames(data))
subset <- select(data, date, sender.latitude)
head(subset)
```

```
## # A tibble: 6 x 2
##   date  sender.latitude
##   <chr>           <dbl>
## 1 5729             51.0
## 2 5741             51.0
## 3 5743             51.0
## 4 5752             51.0
## 5 5757             51.0
## 6 5765             51.0
```

Filtering using `filter()`

```
subset <- filter(data, sender.latitude < 50)
subset <- filter(subset, sender.latitude > 32)
subset <- filter(subset, sender.longitude > 0)
subset <- select(subset,  sender.latitude, sender.longitude)
summary(subset)
```

```
##  sender.latitude sender.longitude
##  Min.   :47.54   Min.   : 2.130
##  1st Qu.:47.54   1st Qu.: 9.212
##  Median :48.12   Median : 9.300
##  Mean   :48.24   Mean   : 9.020
##  3rd Qu.:48.80   3rd Qu.:10.739
##  Max.   :49.57   Max.   :16.320
```

Splitting and uniting variable values using separate() and unite()

```
patient_data <-read_tsv('data/inpatient.tsv')
```

```
## Rows: 163065 Columns: 12
## -- Column specification --------------------------------------------------------
## Delimiter: "\t"
## chr (10): DRG Definition, Provider Name, Provider Street Address, Provider C...
## dbl  (2): Provider Id, Total Discharges
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
colnames(patient_data) <- make.names(colnames(patient_data))
# Split a column into two columns, based on a character
test1 <- separate(patient_data, Hospital.Referral.Region.Description, into=c('state', 'city'), sep='-')
```

```
## Warning: Expected 2 pieces. Additional pieces discarded in 791 rows [710, 711,
## 718, 830, 853, 1918, 1921, 1927, 1930, 1940, 2055, 2086, 3398, 3401, 3411, 3415,
## 3594, 3630, 5424, 5425, ...].
```

```
# Join two columns
test2 <- unite(patient_data, 'combined', c(Provider.Id, Provider.Name), sep='_+_', remove = FALSE)
```

String manipulation using the stringr library

It does happen that you need to clean textual data. The stringr package has a bunch of functions to make your life easier (but not easy). I will run through some examples but do have a look at the cheatsheet as well.

Grouping and summarizing data using group_by() and summarize()

The group_by() function takes a tibble and returns the same tibble, but with some extra information so that any subsequent function can act on each unique combination defined in the group_by().

```
car_data <- read_delim('data/cars.txt', delim = ' ')
```

```
## Rows: 93 Columns: 26
## -- Column specification --------------------------------------------------------
## Delimiter: " "
## chr  (6): make, model, type, cylinders, rearseat, luggage
## dbl (20): min_price, mid_price, max_price, mpg_city, mpg_hgw, airbag, drive,...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
grouped <- group_by(car_data, type, make)
summaries <- summarise(grouped, mean.length = mean(length))
```

```
## `summarise()` has grouped output by 'type'. You can override using the
## `.groups` argument.
```

```
summaries
```

```
## # A tibble: 81 x 3
## # Groups:   type [6]
##    type    make          mean.length
##    <chr>   <chr>               <dbl>
##  1 Compact Audi                  180
##  2 Compact Chevrolet             183
##  3 Compact Chrysler              183
##  4 Compact Dodge                 181
##  5 Compact Ford                  177
##  6 Compact Honda                 185
##  7 Compact Mazda                 184
##  8 Compact Mercedes-Benz         175
##  9 Compact Nissan                181
## 10 Compact Oldsmobile            188
## # ... with 71 more rows
```

You can ask for more than one summary statistic.

```
summaries <- summarise(grouped, mean.length = mean(length), max.length = max(length), std_rpm = sd(rpm))
```

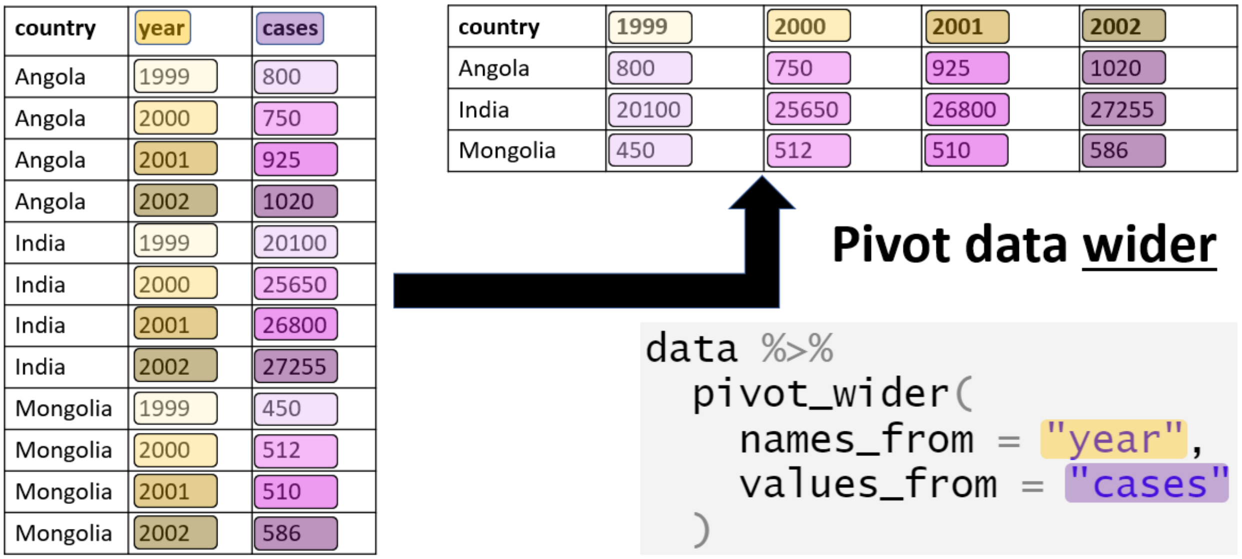
```
## `summarise()` has grouped output by 'type'. You can override using the
## `.groups` argument.
```

```
summaries
```

```
## # A tibble: 81 x 5
## # Groups:   type [6]
##    type    make          mean.length max.length std_rpm
##    <chr>   <chr>               <dbl>      <dbl>   <dbl>
##  1 Compact Audi                  180        180      NA
##  2 Compact Chevrolet             183        184       0
##  3 Compact Chrysler              183        183      NA
##  4 Compact Dodge                 181        181      NA
##  5 Compact Ford                  177        177      NA
##  6 Compact Honda                 185        185      NA
##  7 Compact Mazda                 184        184      NA
##  8 Compact Mercedes-Benz         175        175      NA
##  9 Compact Nissan                181        181      NA
## 10 Compact Oldsmobile            188        188      NA
## # ... with 71 more rows
```

Converting to wide format using `pivot_wider()`

The result can be reshaped into a wide format. While this format is often not suited for plotting or analysis, it might make it easier to look at the data. Here is a quick visual:



country	year	cases
Angola	1999	800
Angola	2000	750
Angola	2001	925
Angola	2002	1020
India	1999	20100
India	2000	25650
India	2001	26800
India	2002	27255
Mongolia	1999	450
Mongolia	2000	512
Mongolia	2001	510
Mongolia	2002	586

country	1999	2000	2001	2002
Angola	800	750	925	1020
India	20100	25650	26800	27255
Mongolia	450	512	510	586

Pivot data **wider**

```
data %>%
  pivot_wider(
    names_from = "year",
    values_from = "cases"
  )
```

```
car_data <- read_delim('data/cars.txt', delim = ' ')
```

```
## Rows: 93 Columns: 26
## -- Column specification --------------------------------------------------------
## Delimiter: " "
## chr  (6): make, model, type, cylinders, rearseat, luggage
## dbl (20): min_price, mid_price, max_price, mpg_city, mpg_hgw, airbag, drive,...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
grouped <- group_by(car_data, type, make)
summaries <- summarise(grouped, mean.length = mean(length))
```

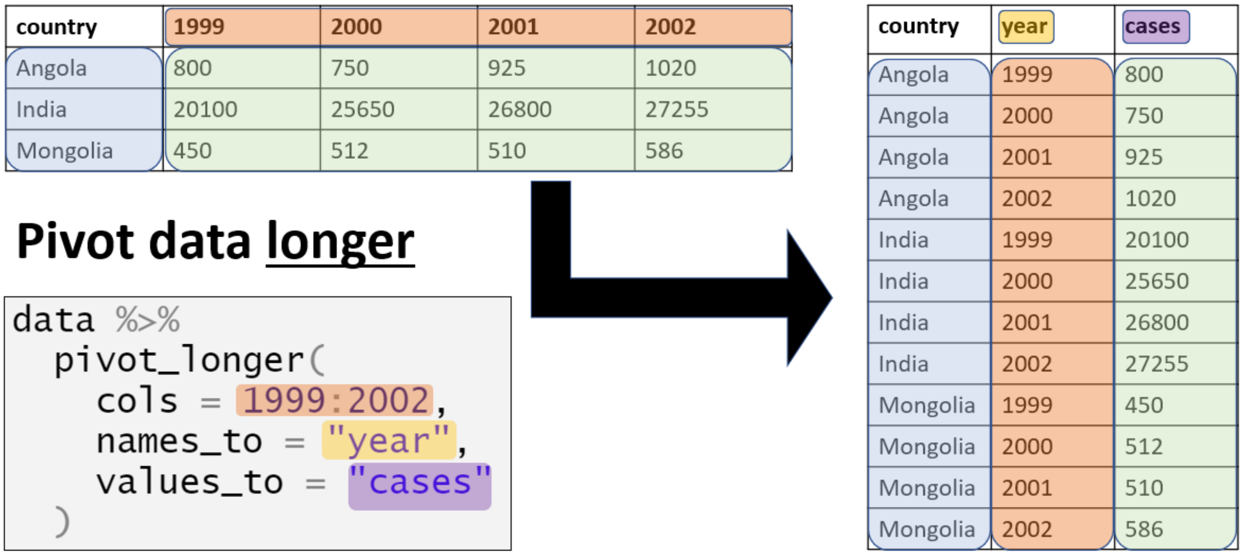
```
## `summarise()` has grouped output by 'type'. You can override using the
## `.groups` argument.
```

```
wide <- pivot_wider(summaries, id_cols = make, names_from  = type, values_from = mean.length)
head(wide)
```

```
## # A tibble: 6 x 7
##   make      Compact Large Midsize Small Sporty   Van
##   <chr>       <dbl> <dbl>   <dbl> <dbl>  <dbl> <dbl>
## 1 Audi          180    NA     193    NA     NA    NA
## 2 Chevrolet     183   214     198    NA    186   186
## 3 Chrysler      183   203      NA    NA     NA    NA
## 4 Dodge         181    NA     192   173    180   175
## 5 Ford          177   212     192   156    180.   176
## 6 Honda         185    NA      NA   173    175    NA
```

Making data longer (melting data) using `pivot_longer()`

Here is a quick graphic:



country	1999	2000	2001	2002
Angola	800	750	925	1020
India	20100	25650	26800	27255
Mongolia	450	512	510	586

Pivot data longer

```
data %>%
  pivot_longer(
    cols = 1999:2002,
    names_to = "year",
    values_to = "cases"
  )
```

country	year	cases
Angola	1999	800
Angola	2000	750
Angola	2001	925
Angola	2002	1020
India	1999	20100
India	2000	25650
India	2001	26800
India	2002	27255
Mongolia	1999	450
Mongolia	2000	512
Mongolia	2001	510
Mongolia	2002	586

Let’s look at some data:

```
head(relig_income, 5)
```

```
## # A tibble: 5 x 11
##   religion        `<$10k` $10-2~1 $20-3~2 $30-4~3 $40-5~4 $50-7~5 $75-1~6 $100-~7
##   <chr>             <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
## 1 Agnostic             27      34      60      81      76     137     122     109
## 2 Atheist              12      27      37      52      35      70      73      59
## 3 Buddhist             27      21      30      34      33      58      62      39
## 4 Catholic            418     617     732     670     638    1116     949     792
## 5 Don’t know/re~       15      14      15      11      10      35      21      17
## # ... with 2 more variables: `>150k` <dbl>, `Don't know/refused` <dbl>, and
## #   abbreviated variable names 1: `$10-20k`, 2: `$20-30k`, 3: `$30-40k`,
## #   4: `$40-50k`, 5: `$50-75k`, 6: `$75-100k`, 7: `$100-150k`
```

This data is in a wider format. But we can easily melt it to a long format.

```
new <- pivot_longer(relig_income, cols = !religion)
head(new, 5)
```

```
## # A tibble: 5 x 3
##   religion name    value
##   <chr>    <chr>   <dbl>
## 1 Agnostic <$10k      27
## 2 Agnostic $10-20k    34
## 3 Agnostic $20-30k    60
## 4 Agnostic $30-40k    81
## 5 Agnostic $40-50k    76
```

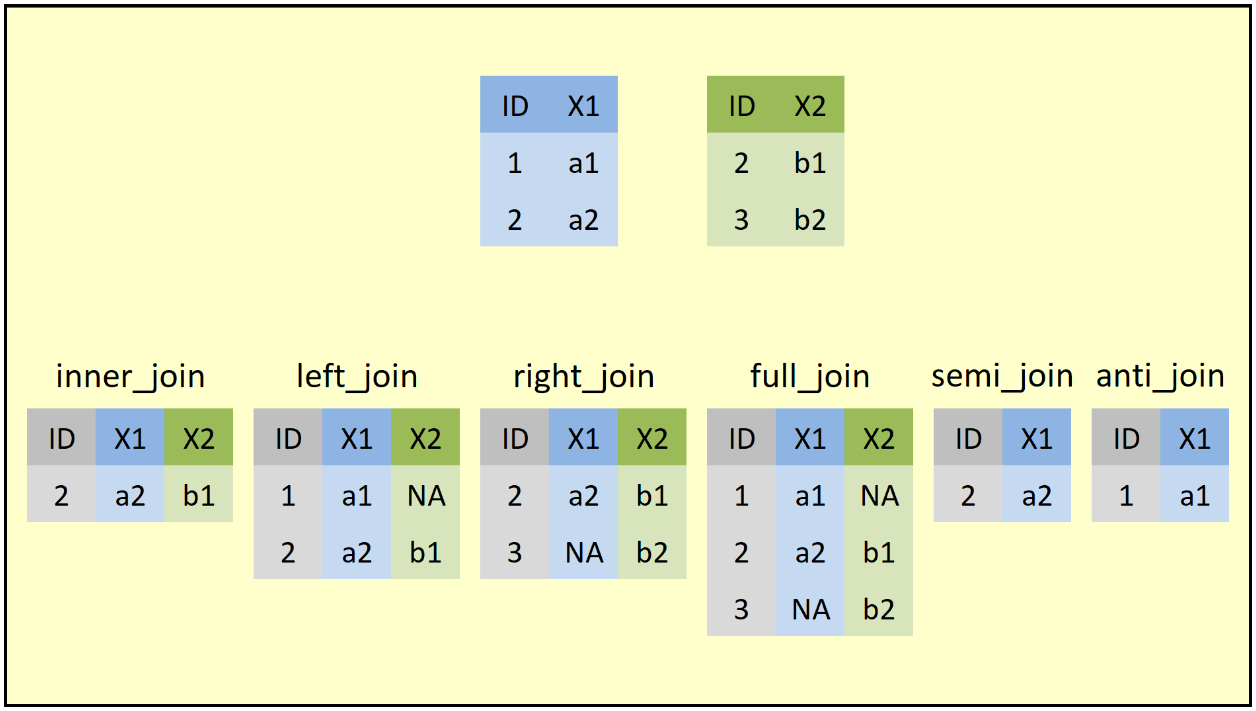
You can specify names for the new columns while melting.

```
new <- pivot_longer(relig_income, !religion, names_to = "income", values_to = "count")
head(new, 5)
```

```
## # A tibble: 5 x 3
##   religion income  count
##   <chr>    <chr>   <dbl>
## 1 Agnostic <$10k      27
## 2 Agnostic $10-20k    34
## 3 Agnostic $20-30k    60
## 4 Agnostic $30-40k    81
## 5 Agnostic $40-50k    76
```

Merging data using the `_join()` functions

The different merge operations are illustrated in the image below. The various operations differ in the way they handle rows missing in the left or right tibble. In the image below, the merge is done by the variable `ID`.



ID	X1
1	a1
2	a2

ID	X2
2	b1
3	b2

inner_join

ID	X1	X2
2	a2	b1

left_join

ID	X1	X2
1	a1	NA
2	a2	b1

right_join

ID	X1	X2
2	a2	b1
3	NA	b2

full_join

ID	X1	X2
1	a1	NA
2	a2	b1
3	NA	b2

semi_join

ID	X1
2	a2

anti_join

ID	X1
1	a1