Submodule 1.4

Data ‘wrangling’ in R

Spring 2024

Load script for submodule #1.4

1. Click here to download the script! Save the script to your working directory (R Project directory).
2. Load your script in your RStudio Project. To do this, open your RStudio Project and click on the folder icon in the toolbar at the top and load your script.

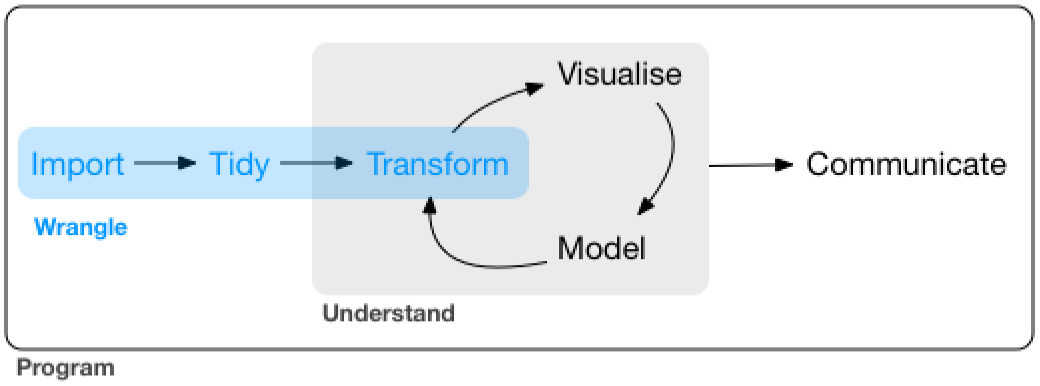
Let’s get started with some data wrangling!

The ‘tidyverse’

Throughout this workshop we have been making use of the ‘tidyverse’ set of packages, which were designed to facilitate everyday data management and visualization tasks in R using a consistent data structure and syntax.

For a much more comprehensive introduction to data wrangling and analysis, you should definitely consider working through the online textbook, R for Data Science by Garrett Grolemund and Hadley Wickham.

Here is a schematic taken from this book, that illustrates the typical data analysis workflow. In general, much of data science consists of **wrangling** data, or getting it in shape for visualization and analysis.



The core tidyverse set of packages includes:

- tidyr, for data tidying
- dplyr, for data transformation
- readr, for data import
- tibble, for tibbles, a re-imagining of data frames
- magrittr, for implementing the ‘pipe’ operator
- ggplot2, for data visualisation (the subject of the next module)

- purrr, for functional programming
- stringr, for strings
- forcats, for factors (categorical variables)
- lubridate, for dates

The following cheat sheets will be helpful in this module:

data 'tidying' data transformation working with dates and times

Learning goals

- Get more comfortable with *tidyverse* grammar (e.g., piping) and data structures to perform basic data manipulation tasks
- Use *tidyr* and *dplyr* data transformation 'verbs' to wrangle data
- Work with and parse dates using *lubridate*

Import sample data

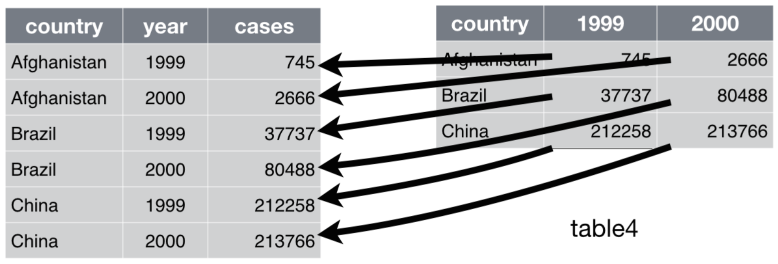
The next few examples use meteorological data from Hungry Horse (HH) and Polson Kerr (PK) dams. You can download this data set by clicking here. Make sure you download this data set into your project directory!!

Let's load this data into R!

```
#  Import and "tidy" your data  ----------------------------

# import meteorological data from Hungry Horse (HH) and Polson Kerr (PK) dams as tibble dataframe using
clim_data <- read_csv("MTMetStations.csv")

# quick look at the contents
clim_data

 # display the last few lines of the data frame (often useful to check!)
tail(clim_data)
```

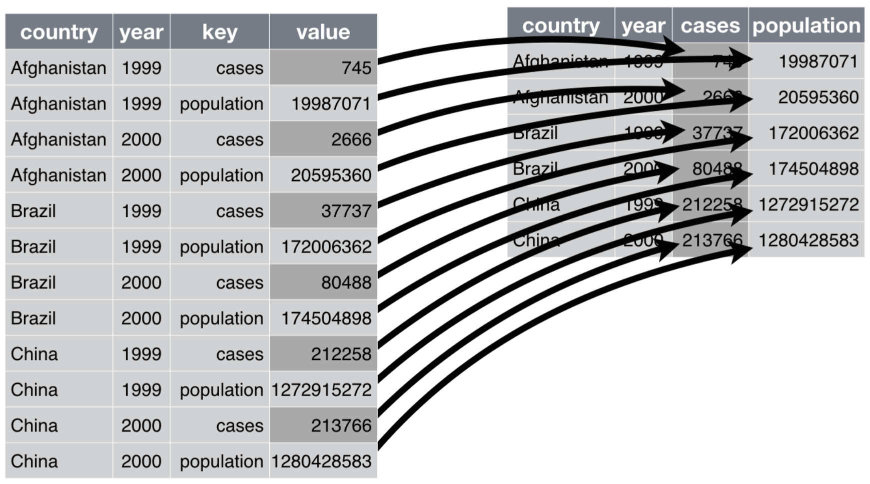
Making data tidy!

The *tidyr* package has several functions to help you get your data into this format. The most important ones are `pivot_longer()` and `pivot_wider()`.



country	year	cases
Afghanistan	1999	745
Afghanistan	2000	2666
Brazil	1999	37737
Brazil	2000	80488
China	1999	212258
China	2000	213766

country	1999	2000
Afghanistan	745	2666
Brazil	37737	80488
China	212258	213766

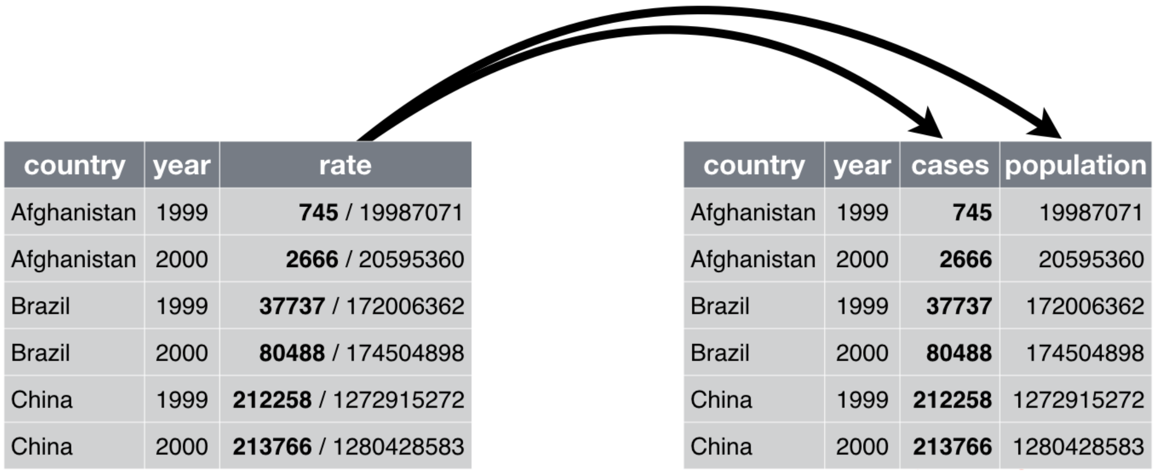


country	year	key	value
Afghanistan	1999	cases	745
Afghanistan	1999	population	19987071
Afghanistan	2000	cases	2666
Afghanistan	2000	population	20595360
Brazil	1999	cases	37737
Brazil	1999	population	172006362
Brazil	2000	cases	80488
Brazil	2000	population	174504898
China	1999	cases	212258
China	1999	population	1272915272
China	2000	cases	213766
China	2000	population	1280428583

country	year	cases	population
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	213766	1280428583

table2

Use `separate()` to separate a single column into two separate columns. `unite()` does the opposite



country	year	rate
Afghanistan	1999	**745** / 19987071
Afghanistan	2000	**2666** / 20595360
Brazil	1999	**37737** / 172006362
Brazil	2000	**80488** / 174504898
China	1999	**212258** / 1272915272
China	2000	**213766** / 1280428583

country	year	cases	population
Afghanistan	1999	**745**	19987071
Afghanistan	2000	**2666**	20595360
Brazil	1999	**37737**	172006362
Brazil	2000	**80488**	174504898
China	1999	**212258**	1272915272
China	2000	**213766**	1280428583

table3

Let's take a close look at the climate dataset and see if there's anything we need to tidy up.

```r
#  Use Tidyr verbs to make data 'tidy'

# look at clim_data -- is it in tidy format? What do we need to do to get it there?
head(clim_data)
```

First of all, recall that tidy data should have have a single observation per row. In this case, each row represents observations taken from two different observatories: PK and HH. Ultimately, we want each row to represent a single date at a single station, one column to represent the station, and three columns to represent three different measurements: TMaxF, TMinF, and PrcpIN.

Our strategy will be the following:

1. Use `pivot_longer()` to make one row for every measurement at every station.

2. Use `separate()` to separate station labels from climate variable labels.
3. Use `pivot_wider()` to put all three measurements from each date/station into its own row.

The first step looks like this:

```
# step 1: pivot this data frame into long format:
#    we will create a new column called 'climvar_station', and all of the numeric precip and temp value
clim_vars_longer <- clim_data %>% pivot_longer(
                           cols = !Date,
                           names_to = "climvar_station",
                           values_to = "value"
                    )

clim_vars_longer
```

In the second step, we separate the station and climate variable labels into two different columns:

```
# step 2: separate the climvar_station column into two separate columns that identify the climate varia
clim_vars_separate <- clim_vars_longer %>% separate(
                           col = climvar_station,
                           into = c("Station","climvar")
                      )

clim_vars_separate
```

Finally, let's put climate measurements from a single station on the same row:

```
# step 3: pivot_wider distributes the clim_var column into separate columns, with the data values from

tidy_clim_data <- clim_vars_separate %>% pivot_wider(
                        names_from = climvar,
                        values_from = value
                  )

tidy_clim_data
```

Note: we can do all of these tidying steps at once using the pipe operator:

```
# repeat above as single pipe series without creation of intermediate datasets

tidy_clim_data <- clim_data %>%
  pivot_longer(cols = !Date,
               names_to = "climvar_station",
               values_to = "value") %>%
  separate(col = climvar_station,
           into = c("Station","climvar")) %>%
  pivot_wider(names_from = climvar,
              values_from = value)

tidy_clim_data
```

Using dplyr – the data wrangling workhorse

dplyr is by far one of the most useful packages in all of the tidyverse as it allows you to quickly and easily summarize and manipulate your data once you have it in tidy form.

We have already worked with dplyr. The basic dplyr verbs include:

```
  * grouping data with *group_by()*
  * filtering rows with *filter()*
  * creating new variables with *mutate()*
  * summarizing with *summarize()*
  * selecting columns with *select()*
  * sorting columns by row with *arrange()*
```

```r
#  Use dplyr verbs to wrangle data   ----------------------------

# example of simple data selection and summary using group_by, summarize, and mutate verbs

# take tidy_clim_data, then
# group data by station, then
# calculate summaries and put in columns with names mean.precip.in, mean.TMax.F, and mean.Tmin.F, then
# transform to metric and put in new columns mean.precip.in, mean.TMax.F, and mean.Tmin.F

station_mean1 <- tidy_clim_data %>%
  group_by(Station) %>%
  summarize(
    mean.precip.in = mean(PrcpIN, na.rm=TRUE),
    mean.TMax.F = mean(TMaxF, na.rm=TRUE),
    mean.TMin.F = mean(TMinF, na.rm=TRUE)) %>%
  mutate(
    mean.precip.mm = mean.precip.in * 25.4,
    mean.TMax.C = (mean.TMax.F - 32) * 5 / 9,
    mean.TMin.C = (mean.TMin.F - 32) * 5 / 9
  )

station_mean1
```

There are several tricks that tidyverse provides that allow you to write as compact code as possible (avoiding re-typing the same code).

For example, the above code block involves doubling the formula for converting to celsius.

```r
# using an even more compact form:

# take tidy_clim_data, then
# group data by station, then
# calculate summary (mean of all non-NA values) for numeric data only, then
# transform temp data (.) from F to C, then
# transform precip data (.) from in to mm


station_mean2 <- tidy_clim_data %>%
  group_by(Station) %>%
  summarize(across(where(is.numeric), mean, na.rm=TRUE)) %>%
  mutate(across(c("TMaxF", "TMinF"), ~(.-32)*(5/9)))   %>%
  rename_with(~gsub("F","C",.),starts_with("T")) %>%
  mutate(across(PrcpIN, ~.*25.4,.names="Prcp_mm")) %>%
  select(!PrcpIN)

station_mean2
```

Working with Dates with *lubridate* - Date from character string

Dates can be tricky to work with and we often want to eventually get to the point that we can summarize our data by years, seasons, months, or even day of the year. The *lubridate* package in R (also part of the tidyverse) is very useful for creating and parsing dates for this purpose. Date/time data often come as strings in a variety of orders/formats. `Lubridate` is useful because it automatically works out the date format once you specify the order of the components. To do this, identify the order in which year, month, and day appear in your dates, then arrange "y", "m", and "d" in the same order. That gives you the name of the lubridate function that will parse your date!

For example:

```
#  Using lubridate to format and create date data types -----------------

library(lubridate)

date_string <- ("2017-01-31")

# convert date string into date format by identifing the order in which
#   year, month, and day appear in your dates, then arrange "y", "m", and #   "d" in the same order. Th
#    function that will parse your date

ymd(date_string)
```

```
## [1] "2017-01-31"
```

```
# note the different formats of the date_string and date_dtformat objects in the environment window.

# a variety of other formats/orders can also be accommodated. Note how each of these are reformatted to

mdy("January 31st, 2017")
```

```
## [1] "2017-01-31"
```

```
dmy("31-Jan-2017")
```

```
## [1] "2017-01-31"
```

```
ymd(20170131)
```

```
## [1] "2017-01-31"
```

```r
ymd(20170131, tz = "UTC")
```

```
## [1] "2017-01-31 UTC"
```

```r
# can also make a date from components.
#   this is useful if you have columns for year, month,
#   day in a dataframe

year<-2017
month<-1
day<-31
make_date(year, month, day)
```

```
## [1] "2017-01-31"
```

```r
# times can be included as well. Note that unless otherwise specified, R assumes UTC time

ymd_hms("2017-01-31 20:11:59")
```

```
## [1] "2017-01-31 20:11:59 UTC"
```

```r
mdy_hm("01/31/2017 08:01")
```

```
## [1] "2017-01-31 08:01:00 UTC"
```

```r
# we can also have R tell us the current time or date

now()
```

```
## [1] "2024-02-01 20:46:54 PST"
```

```r
today()
```

```
## [1] "2024-02-01"
```

Working with Dates with lubridate - Parsing Dates

Once dates are in date format, we can easily pull out their individual components like this:

```r
#  Parsing dates with lubridate

datetime <- ymd_hms("2016-07-08 12:34:56")

# year
year(datetime)
```

```
## [1] 2016
```

```r
# month as numeric
month(datetime)
```

```
## [1] 7
```

```r
# month as name
month(datetime, label = TRUE)
```

```
## [1] Jul
## Levels: Jan < Feb < Mar < Apr < May < Jun < Jul < Aug < Sep < Oct < Nov < Dec
```

```r
# day of month
mday(datetime)
```

```
## [1] 8
```

```r
# day of year (often incorrectly referred to as julian day)
yday(datetime)
```

```
## [1] 190
```

```r
# day of week
wday(datetime)
```

```
## [1] 6
```

```r
wday(datetime, label = TRUE, abbr = FALSE)
```

```
## [1] Friday
## Levels: Sunday < Monday < Tuesday < Wednesday < Thursday < Friday < Saturday
```

Working with Dates - Parsing Dates Example

We can use lubridate and dplyr to make new columns from a date for year, month, day of month, and day of year

first we convert the character string date into date format. Here we are naming the new column "Date", so R will replace character string with date formatted dates

```r
###  Using lubridate with dataframes and dplyr verbs

# going back to our tidy_clim_data dataset we see that
#   the date column is formatted as character, not date

head(tidy_clim_data)

# change format of date column
tidy_clim_data <- tidy_clim_data %>%
  mutate(Date = mdy(Date))
tidy_clim_data
```

Now we can use mutate to create individual columns for the date components

```
# parse date into year, month, day, and day of year columns
tidy_clim_data <- tidy_clim_data %>%
  mutate(
    Year = year(Date),
    Month = month(Date),
    Day = mday(Date),
    Yday = yday(Date)
  )

tidy_clim_data

# calculate total annual precipitation by station and year
annual_sum_precip_by_station <- tidy_clim_data %>%
  group_by(Station, Year) %>%
  summarise(PrecipSum = sum(PrcpIN))

annual_sum_precip_by_station
```

Challenge Exercises

1. Try to put tidyr's built-in `relig_income` dataset into a tidier format. This dataset stores counts based on a survey which (among other things) asked people about their religion and annual income:

Your final result should look like this:

2. Use the dplyr verbs and the tidy_clim_data dataset you created above to calculate monthly average Tmin and Tmax for each station
3. Using lubridate, make a date object out of a character string of your birthday (you can use a fake birthday if you want!) and find the day of week it occurred on. Note, you can use the 'wday' function in lubridate to extract the day of the week (1 is sunday).

```
# CHALLENGE EXERCISES   -------------------------------------

# 1. Try to put tidyr's built-in `relig_income` dataset into a tidier format.
#      This dataset stores counts based on a survey which (among other things)
#      asked people about their religion and annual income:

# 2. Use the dplyr verbs and the tidy_clim_data dataset you created above
#       to calculate monthly average Tmin and Tmax for each station (in Fahrenheit is okay)

# 3. Using lubridate, make a date object out of a character string of your
#        birthday and find the day of week it occurred on. Note, you
#        can use the 'wday' function in lubridate to extract the day of the
#         week (1 is sunday).
```

–go to next submodule–