Session 3: Tidyverse

Ashwin Karanam

06/29/2020

Contents

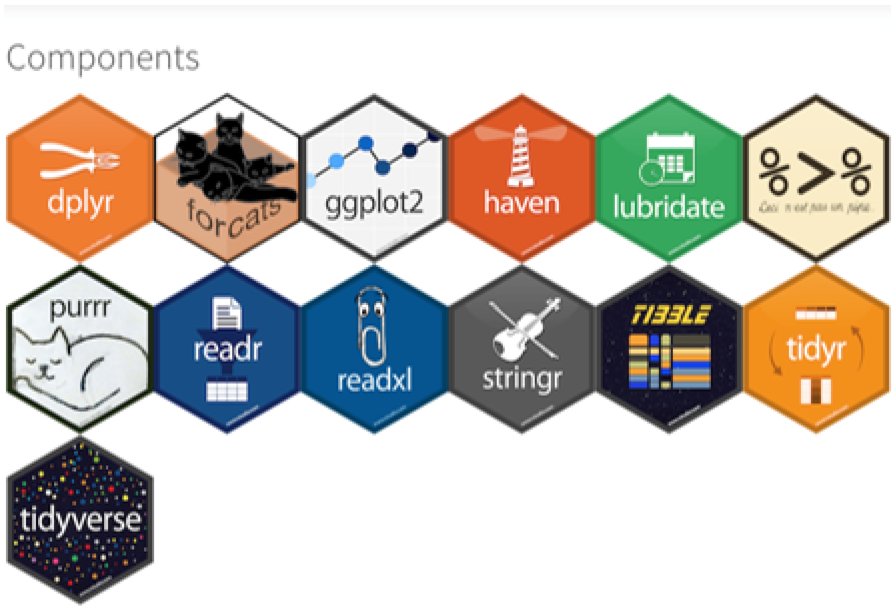
Objectives

- What is `tidyverse` and packages in `tidyverse`?
- `readr`
- `dplyr`

What is tidyverse?

The tidyverse is a collection of open source R packages introduced by Hadley Wickham and his team that "share an underlying design philosophy, grammar, and data structures" of `tidy data`. `Tidy data` is a standard method of displaying a multivariate set of data is in the form of a data matrix in which rows correspond to sample individuals and columns to variables, so that the entry in the i^{th} row and j^{th} column gives the value of the j^{th} variate as measured or observed on the i^{th} individual.

Packages in `tidyverse`



Installing `tidyverse`

```r
install.packages("tidyverse")
```

```r
# Call library
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------------------------------------------- tidy
```

```
## v ggplot2 3.2.1     v purrr   0.3.3
## v tibble  2.1.3     v dplyr   0.8.3
## v tidyr   1.0.0     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0
```

```
## -- Conflicts ---------------------------------------------------------------------------- tidyverse_
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

The `readr` package

Tidyverse packages providing rectangular data importing and saving functions. Cheatsheets can be found `here`. [`readr`] is used for turning flat files into data frames.

Importing data

- `read_csv("file.csv")`: Comma Delimited Files
- `read_csv2("file2.csv")`: Semi-colon Delimited Files
- `read_delim("file.txt", delim = "|")`: Files with Any Delimiter
- `read_fwf("file.fwf", col_positions = c(1, 3, 5))`: Fixed Width Files
- `read_tsv("file.tsv")`: Tab Delimited Files

Let us import the `Puromycin` dataset

```r
Puromycin <- read_csv("Puromycin.csv",
                      #col_names = FALSE, # skip header row
                      #col_names = c("x", "y", "z"), # provide header
                      #skip = n, # skip nth row
                      col_types = cols(conc = col_double(),
                                       rate = col_double(),
                                       state = col_character()))

# Open a tab
View(Puromycin)

# First 5 rows
head(Puromycin)

# Last 5 rows
tail(Puromycin)

# A glimpse of the dataset
glimpse(Puromycin)
```

Saving datafiles

- `write_csv(x, path)`: Comma delimited file
- `write_delim(x, path, delim = " ")`: File with arbitrary delimiter
- `write_excel_csv(x, path)`: CSV for excel
- `write_tsv(x, path)`: write_tsv

Let us save a file of our own. Let's the Theophylline data into a CSV file.

```r
# Take a look at the data
View(Theoph)

write_csv(Theoph,
          "Theophylline.csv",
          # delim = " ", #for delimited files
          # na = "-99", # String used for missing values. Defaults to NA
          # append = T/F #If FALSE, will overwrite existing file.
                         #If TRUE, will append to existing file.
                         #In both cases, if file does not exist a new file is created.
          # col_names = T/F #Write columns names at the top of the file?
          )
```

The dplyr package

dplyr is a grammar of data manipulation, providing a consistent set of verbs that help you solve the most common data manipulation challenges. But before we explore dplyr, we need to understand the pipe/%>% operator from the magrittr

The magrittr package

It offers a set of operators which make your code more readable. Pipes are a powerful tool for clearly expressing a sequence of multiple operations.

Some basic piping rules:

- `x %>% f` is equivalent to `f(x)`
- `x %>% f(y)` is equivalent to `f(x, y)`
- `x %>% f() %>% g() %>% h()` is equivalent to `h(g(f(x)))`
- `x %>% f(y, .)` is equivalent to `f(y, x)`
- `x %>% f(y, z = .)` is equivalent to `f(y, z = x)`

```r
x <- rnorm(n=10,mean=20,sd=5)

# Let's say I want to round the vector to nearest integer and then take mean

# non-pipe formulation
mean(round(x,digits = 0))

#pipe formulation
x %>%
  round(digits = 0) %>% # same as round(.,digits=0)
  mean() # same as mean(.)
```

Back to dplyr

This package is massive and beautiful. We will be covering only the basics of this package, so please look at the cheatsheets

Let us use the Theophylline data to explore this package. First, let's start with summarizing the dataset

```r
Theo_summ <-
  Theoph %>%
  #Summarise the data
  #Calculate number of subjects, weight and dose ditributions
  summarise(nID = n_distinct(Subject), # count the number of individuals
            `Weight (Mean,SD)` = paste(Wt %>% mean() %>% round(digits=1),
                                       "(",
                                       Wt %>% sd() %>% round(digits=1),
                                       ")",
                                       sep=""),
            `Dose (Median,Range)` = paste(Dose %>% median() %>% round(digits=1),
                                          "(",
                                          range(Dose)[1],
                                          "-",
                                          range(Dose)[2],
                                              ")",
                                              sep=""))


# Individual Cmax/Cmed

Cmax.med <-
  Theoph %>%
  group_by(Subject) %>%
  summarise(Cmax = max(conc),
            Cmed = median(conc))
```

Other useful summarize functions:

- `summarise_all()` - Apply funs to every column.
- `summarise_at()` - Apply funs to specific columns.
- `summarise_if()` - Apply funs to all cols of one type.

Now let's try to manipulate cases/vairables:

```
# Filter subjects with dose less than 4.5
# Add a column of weight descriptor
# Select only the Subject, Time and conc columns and weight descriptor
# Renmae Subject to ID and conc to DV

Theop2 <-
  Theoph %>%
  filter(Dose < 4.5) %>%
  mutate(WTLT60 = if_else(Wt <= 60,1,0)) %>%
  select(Subject, Time, conc, WTLT60) %>%
  rename(ID=Subject,DV=conc)

## Let's create a NONMEM datatset

# Let's start with the dosing rows

dosing <-
  Theoph %>%
  group_by(Subject) %>%
  filter(Time == 0) %>%
  mutate(conc = NA)

# Concentration rows

conc <-
  Theoph %>%
  group_by(Subject) %>%
  filter(! (Time == 0 & conc == 0)) %>%
  mutate(Time = if_else(Time == 0, 0.001, Time),
         Dose = NA,
         LNDV = log(conc) %>% round(2))
         )
```

```r
# join the two

Theop_NM <-
  dosing %>%
  bind_rows(conc) %>%
  arrange(Subject,.by_group=TRUE) %>%
  mutate(MDV = if_else(is.na(Dose),0,1),
         EVID = if_else(is.na(Dose),0,1),
         WTLT60 = if_else(Wt <= 60,1,0)) %>%
  rename(ID=Subject,DV=conc,AMT=Dose)
```

Other useful manipulation functions:

- `distinct(.data, ..., .keep_all = FALSE)` - Remove rows with duplicate values
- `sample_frac(tbl, size = 1, replace = FALSE)` - Randomly select fraction of rows
- `slice(.data, .)` - Select rows by position
- `transmute(.data, .)` - Compute new column(s), drop others
- `mutate_all(.tbl, .funs, .)` - Apply funs to every column
- `mutate_at(.tbl, .cols, .funs, .)` - Apply funs to specific columns
- `add_column(.data, ..., .before = NULL, .after = NULL)` - Add new column(s)

Now, let's look at combining tables

```r
# Let's say you have a dataset with additional demographic data

demog <- data.frame(
  ID = unique(Theop_NM$ID),
  SEX = sample(c("M","F"),12,replace = TRUE),
  BSA = rnorm(12,1.73,0.2) %>% round(2),
  GENO = sample(c("PM","WT","FM"),12,replace = TRUE)
)

# Merge this with the NM dataset

Theop_NM2 <-
  Theop_NM %>%
  left_join(demog,by="ID")

# But we cannot have strings in NONMEM, so let us convert them to numeric

Theop_NM3 <-
  Theop_NM2 %>%
  mutate(SEX2 = if_else(SEX == "M",1,0),
         GENO2 = case_when(
           GENO == "WT" ~ 0,
           GENO == "PM" ~ 1,
           GENO == "FM" ~ 2)
         )
```

Other important join functions:

- `right_join(x, y, by = NULL,.)` - Join matching values from x to y

- `inner_join(x, y, by = NULL,.)` - Retain only rows with matches
- `full_join(x, y, by = NULL,.)` - Retain all values, all rows
- `semi_join(x, y, by = NULL, .)` - Return rows of x that have a match in y
- `anti_join(x, y, by = NULL, .)` - Return rows of x that do not have a match in y

Let's write out this file using the `write_csv()` function

```
write_csv(Theop_NM3, "Theop06292020.csv", na = ".")
```

Takehome exercises:

- The best exercise is to start working with your own dataset. The complexities and intracacies that exist in real world data cannot be replicated with dummy datasets
- Online exercises:
  - `Basics`
  - `Intermediate`
  - `Advanced`