# Workshop: Basic R and More, Part 2 of 3

## Graduate Training and Education Center
## Mid-Year Meeting
## University of Mississippi Medical Center (UMMC)
## Jackson, MS

Philip Turk, PhD, MS[1]

Professor of Family and Community Medicine
Director, Clinical and Translational Research Institute
Northeast Ohio Medical University (NEOMED)

12/14/2024 - 12/15/2024


[1] <pturk@neomed.edu>

# Data Frames

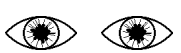
- R stores collections of variables (of different types) in a data structure known as a *data frame*. (Nowadays, you will hear people refer to *tibbles*, which are "optimized" data frames for newer packages.)
- The columns are for different variables, and the rows are for different units (e.g., person, thing, etc.) Hence, looking across a given row, each column represents some information about that unit.
- As a reminder, do not type the line spans + in the following code if you try this at the command line. **From this point on, it will be much easier to type code into an R script and then submit it.**

```
> Scores <- data.frame(
+   Name  = c("Maggie", "Phil", "Jing", "Hieu"),
+   Sex = c("F", "M", "F", "M"),
+   Exam_1 = c(90, 75, 92, 85),
+   Exam_2 = c(87, 71, 95, 81)
+ )
```

# Data Frames

```
> Scores
```

```
    Name Sex Exam_1 Exam_2
1 Maggie   F     90     87
2   Phil   M     75     71
3   Jing   F     92     95
4   Hieu   M     85     81
```

- Here is where RStudio shines. An attractive option is to be able to show the data in an spreadsheet and even interact with it 

# Data Frames

- There are two different ways to access the components of a data frame. The first way allows for the previous 'matrix'-type of indexing using `[]`.

```
> Scores[1, 3] ## Row 1, column 3
```

```
[1] 90
```

```
> Scores[1, ] ## All of row 1
```

```
    Name Sex Exam_1 Exam_2
1 Maggie   F     90     87
```

```
> Scores[, 3] ## All of column 3
```

```
[1] 90 75 92 85
```

# Data Frames

- Because we have created column names, we can access a column by its name as opposed to the column number. The $ sign can be thought of as a pathway: *data frame name$column name within data frame*

```
> Scores$Exam_1
```

```
[1] 90 75 92 85
```

- We can use both the column name followed by `[]`.

```
> Scores$Exam_1[2]
```

```
[1] 75
```

- Notice the automatic completion feature in the Console window

# Importing Data

- For small data sets, we want to read data from an external source where the data are set up like a data frame.
- Every time you start up RStudio, it picks an appropriate working directory. This is the directory where it will first look for .R/.Rmd files or data files. By default, when you double click on a .R/.Rmd file to launch RStudio, it will set the working directory to be the directory that the file was in.
- There are three different possibilities where we tell R the data file resides:
  - a web address
  - an absolute path on your computer
  - a path relative to the location of your .R/.Rmd file

# Importing Data

- The simplest file format for storage is the *comma separated values* text file, or .csv file. Each of the 'cells' of data are separated by a comma. Ideally, there are column names. Any missing data are designated using a period sign. MS Excel can save a workbook as a .csv file.
- Typically when you open up such a file on a computer with MS Excel installed, Excel will open up the file assuming it is a spreadsheet and put each element in its own cell. However, you can also open the file using a more 'primitive' text editor. My favorite is the free version of Sublime because it works very nicely with Python and Git (<https://www.sublimetext.com>).
- Let's take a look at a toy data set called *bears.csv* that is completely ready to be imported into R  

# Importing Data

- To import *bears.csv* from a path relative to the location of your .R/.Rmd file, one approach is the following 

```
> my_data <- read.csv("bears.csv")
```

- If *bears.csv* was instead located in some far-flung location of my computer, then I would extend the path accordingly. Notice the use of forward slashes.

```
> my_data <- read.csv("/Users/pturk/.../bears.csv")
```

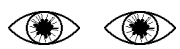
- Caution: Mac OS uses / in paths; Windows uses \\ in paths

# Importing Data

- To read a .csv text file from a web address, we substitute the file path with the web URL. For example, try a.) *carefully typing*, or b.) *copying and pasting* the following URL as the argument for the previous `read.csv()` function and hit Return. Fingers crossed! 

  "https://raw.githubusercontent.com/philturk/R_Wshop/master/bears.csv"

# Packages

- One of the greatest strengths about R is that many people have developed add-on *packages* to do some additional functions above and beyond base R.
- RStudio makes it easy to obtain packages. Go the Packages tab in the lower-right window. Click Install. In the Packages field, start typing the name of your package (e.g., `blandr`). Notice the automatic completion again. When you are done, click Install. Let's do this 
- Once a package is installed on your computer, it is available, but not "loaded" into your current R session by default. To improve overall computer performance, only a few packages are loaded by default. You need to explicitly load add-on packages each session using the `library()` function:

```
> library(blandr)
```

# Summarizing Data Using `dplyr`

- I use the `tidyverse` package so much that I load it right away in almost every R session (submit `library(tidyverse)`). Actually, when you load "the `tidyverse`", the suite of core packages (the packages used in most data engineering and analyses) get installed and loaded en masse.
  - <https://www.tidyverse.org> 
- It is very important to be able to take a data set, wrangle it, produce summary statistics, etc. Although there are 'classic' functions in base R that have some capability of doing these, nowadays I rely on the package `dplyr` (a member of the `tidyverse`). This package also allows me to chain together many common actions to accomplish a particular task in a **fast**, convenient, and consistent way.
- Five functions in `dplyr` are so widely used they are called "The Five Verbs".
  - `select()`, `filter()`, `mutate()`, `arrange()`, `summarize()`
  - Specialized variants of The Five Verbs exist for specific purposes.

# Summarizing Data Using `dplyr`

- The `select()` function returns a subset of the *columns* of a data frame. Let's use our `Scores` data frame.

```
> library(tidyverse)
> select(Scores, 1:3)
```

```
    Name Sex Exam_1
1 Maggie   F     90
2   Phil   M     75
3   Jing   F     92
4   Hieu   M     85
```

```
> ## select(Scores, Name, Sex, Exam_1) returns same result
> ## select(Scores, 3:1) reverses column order
```

# Summarizing Data Using `dplyr`

- The `filter()` function returns a subset of the *rows* of a data frame.

```
> filter(Scores, Sex == "F") ## == means 'exactly equal to'
```

```
    Name Sex Exam_1 Exam_2
1 Maggie   F     90     87
2   Jing   F     92     95
```

```
> filter(Scores, Exam_1 < 90)
```

```
  Name Sex Exam_1 Exam_2
1 Phil   M     75     71
2 Hieu   M     85     81
```

# Summarizing Data Using dplyr

- The `mutate()` function allows us to add a new variable or transform a variable.

```
> mutate(Scores, Change = (Exam_2 - Exam_1) / Exam_1)
```

```
    Name Sex Exam_1 Exam_2      Change
1 Maggie   F     90     87 -0.03333333
2   Phil   M     75     71 -0.05333333
3   Jing   F     92     95  0.03260870
4   Hieu   M     85     81 -0.04705882
```

# Summarizing Data Using dplyr

- You can do multiple calculations within the same `mutate()` command, and you can even refer to columns that were created in the same `mutate()` command.

```
> mutate(Scores, Change = (Exam_2 - Exam_1) / Exam_1,
+        Per_Change = round(Change*100, 2))
```

```
    Name Sex Exam_1 Exam_2      Change Per_Change
1 Maggie   F     90     87 -0.03333333      -3.33
2   Phil   M     75     71 -0.05333333      -5.33
3   Jing   F     92     95  0.03260870       3.26
4   Hieu   M     85     81 -0.04705882      -4.71
```

# Summarizing Data Using dplyr

- The `arrange()` function allows us to order the rows by one or more columns in ascending (alphabetical) or descending order.

```
> arrange(Scores, Name) ## Default is ascending
```

```
    Name Sex Exam_1 Exam_2
1   Hieu   M     85     81
2   Jing   F     92     95
3 Maggie   F     90     87
4   Phil   M     75     71
```

# Summarizing Data Using dplyr

- We order first by Sex and then by Exam_1, both in descending *nested* order.

```
> arrange(Scores, desc(Sex), desc(Exam_1))
```

```
    Name Sex Exam_1 Exam_2
1   Hieu   M     85     81
2   Phil   M     75     71
3   Jing   F     92     95
4 Maggie   F     90     87
```

# Summarizing Data Using `dplyr`

- The `summarize()` function allows us to generate or calculate summary statistics for a given column in the data frame.

```
> summarize(Scores, Exam_1_mean = mean(Exam_1),
+           Exam_1_sd = sd(Exam_1))

  Exam_1_mean Exam_1_sd
1        85.5  7.593857
```

# Summarizing Data Using dplyr

- The group_by() function is not one of The Five Verbs, but it is important in its own right. It tells dplyr functions to perform their actions separately for each 'group'.

```
> summarize(group_by(Scores, Sex),
+           Exam_1_mean = mean(Exam_1),
+           Low_Score = min(Exam_1))
```

```
# A tibble: 2 x 3
  Sex   Exam_1_mean Low_Score
  <chr>       <dbl>     <dbl>
1 F              91        90
2 M              80        75
```

- Minor caution: be careful using group_by(). Some R gurus tend to use the ungroup() function after every group_by() for a few technical reasons.

# Summarizing Data Using `dplyr`

- In `dplyr`, the so-called *pipe operator* %>% is a very useful command.
- For example, if we wanted to start with a data frame `x`, and first apply the function `f()`, then `g()`, and then `h()`:
  - Hard way: the usual R command would be `h(g(f(x)))`, a nested sequence of functions starting at the innermost set of parentheses
  - Easy way: using the pipe command %>%, this sequence of operations becomes `x %>% f() %>% g() %>% h()`
- Pro tip: in RStudio, the keyboard shortcut for the pipe operator %>% is Ctrl + Shift + M (Windows) or Cmd + Shift + M (Mac).

# Summarizing Data Using dplyr

- The hard way (yikes!):

```
> summarize(group_by(filter(Scores, Exam_1 > 75),
+           Sex), Best_Score = max(Exam_2))
```

- The easy way:

```
> Scores %>% filter(Exam_1 > 75) %>% group_by(Sex) %>%
+ summarize(Best_Score = max(Exam_2))
```

```
# A tibble: 2 x 2
  Sex   Best_Score
  <chr>      <dbl>
1 F             95
2 M             81
```

# Data Reshaping Using `tidyr`

- Many procedures in statistical software packages expect the data to be in the *long view*, where each row is a unit and each column is a distinct variable measured on the unit. Unfortunately, I often get data sets with repeated measurements on the same variable (e.g., Time) strung out horizontally on a single row; that is, the data are in the *wide view*.
- When this happens for a data set, we often need to switch, or pivot, from the wide view to the long view in order to do our data analysis.
- An example will make this clear.

# Data Reshaping Using `tidyr`

- Consider our `Scores` data set below. It is in the wide view. Specifically, values for `Exam_1` and `Exam_2` could be thought of as repeated measurements on the same variable `Midterm`, say, which has two levels `Exam_1` and `Exam_2`. Therefore, we will *reshape* the `Exam_1` and `Exam_2` columns into two columns: a new categorical variable `Midterm` and a new response variable `Score`.

```
> Scores
```

```
    Name Sex Exam_1 Exam_2
1 Maggie   F     90     87
2   Phil   M     75     71
3   Jing   F     92     95
4   Hieu   M     85     81
```

- The `tidyr` package (in the `tidyverse`) allows us to "tidy" our data frame; specifically, we use the `pivot_longer()` function to go from wide to long view for the `Scores` data frame.

# Data Reshaping Using `tidyr`

- In the `pivot_longer()` function, the `cols` argument tells R which columns to 'pivot_longer' and apply this to, `names_to` is the new categorical variable, and `values_to` is the new response variable.

```
> tidy_Scores <- Scores %>%
+   pivot_longer(cols = Exam_1:Exam_2, names_to = "Midterm",
+                values_to = "Score")
> tidy_Scores
```

```
# A tibble: 8 x 4
  Name   Sex   Midterm Score
  <chr>  <chr> <chr>   <dbl>
1 Maggie F     Exam_1     90
2 Maggie F     Exam_2     87
3 Phil   M     Exam_1     75
4 Phil   M     Exam_2     71
5 Jing   F     Exam_1     92
6 Jing   F     Exam_2     95
7 Hieu   M     Exam_1     85
8 Hieu   M     Exam_2     81
```

# Data Reshaping Using `tidyr`

- On the other hand, there are certain instances in data science where we prefer to have the data in a wide view (e.g., cluster analysis).
- In the simpler `pivot_wider()` function, `names_from` specifies which column to get the names of the output columns from, while `values_from` specifies which column to get the cell values from.

```
> tidy_Scores %>%
+   pivot_wider(names_from = "Midterm",
+               values_from = "Score")
```

```
# A tibble: 4 x 4
  Name   Sex   Exam_1 Exam_2
  <chr>  <chr>  <dbl>  <dbl>
1 Maggie F         90     87
2 Phil   M         75     71
3 Jing   F         92     95
4 Hieu   M         85     81
```

# Table Joins Using `dplyr`

- As a sidebar, let's get the most from RStudio. Go to Tools > Global Options and do:
  - Appearance, Editor theme, and pick your favorite
  - Accessibility, click Highlight focused panel
  - Code, Display, click Rainbow parentheses
  - Pane Layout, Add Column
- Handy reference:

<https://stat545.com/join-cheatsheet>

# Table Joins Using dplyr

- We often need to join together two data frames that do not have the same number of rows. Suppose we also have the following data frame and want to join it with our new data frame `tidy_Scores`.

```
> Scores02 <- data.frame(
+   Name  = c("Maggie", "Bill", "Jing", "Hieu"),
+   Area = c("CO", "NC", "CN", "NC"),
+   Hobby = c("Horse", "Poker", "Horse", "Fish")
+ )
> Scores02
```

```
    Name Area Hobby
1 Maggie   CO Horse
2   Bill   NC Poker
3   Jing   CN Horse
4   Hieu   NC  Fish
```

# Table Joins Using `dplyr`

- We will use the function `full_join()` from the `dplyr` package to join these two data frames. It return all rows and all columns from both data frames. Where there are not matching values, it returns `NA` for the value missing.
- You will note that each of these data frames `tidy_Scores` and `Scores02` has a `Name` column. This is good practice to do.
- When we join these two data frames, the row that describes `Maggie`, say, from `Scores02` should be duplicated for each row in `tidy_Scores` that corresponds with `Maggie`.
- Notice `Bill` does not appear in `tidy_Scores` and `Phil` does not appear in `Scores02`. Therefore, after the join, "missing values" will be denoted as `<NA>` (for categories) and `NA` (for numbers) in the appropriate columns.

# Table Joins Using dplyr

```
> tidy_Scores %>%
+   full_join(Scores02) ## Could specify by = "Name"
```

```
Joining with `by = join_by(Name)`
```

```
# A tibble: 9 x 6
  Name   Sex   Midterm Score Area  Hobby
  <chr>  <chr> <chr>   <dbl> <chr> <chr>
1 Maggie F     Exam_1     90 CO    Horse
2 Maggie F     Exam_2     87 CO    Horse
3 Phil   M     Exam_1     75 <NA>  <NA>
4 Phil   M     Exam_2     71 <NA>  <NA>
5 Jing   F     Exam_1     92 CN    Horse
6 Jing   F     Exam_2     95 CN    Horse
7 Hieu   M     Exam_1     85 NC    Fish
8 Hieu   M     Exam_2     81 NC    Fish
9 Bill   <NA>  <NA>       NA NC    Poker
```

# Table Joins Using dplyr

- There are other types of joins we often use in practice. For example, `A %>% left_join(B)` return all rows from `A`, and all columns from `A` and `B`. Rows in `A` with no match in `B` will have NA values in the new columns. If there are multiple matches between `A` and `B`, all combinations of the matches are returned 

```
> Scores02 %>% left_join(tidy_Scores) ## I chose order
```

```
Joining with `by = join_by(Name)`
```

| | Name | Area | Hobby | Sex | Midterm | Score |
|---|---|---|---|---|---|---|
| 1 | Maggie | CO | Horse | F | Exam_1 | 90 |
| 2 | Maggie | CO | Horse | F | Exam_2 | 87 |
| 3 | Bill | NC | Poker | <NA> | <NA> | NA |
| 4 | Jing | CN | Horse | F | Exam_1 | 92 |
| 5 | Jing | CN | Horse | F | Exam_2 | 95 |
| 6 | Hieu | NC | Fish | M | Exam_1 | 85 |
| 7 | Hieu | NC | Fish | M | Exam_2 | 81 |