What is Tidyverse?

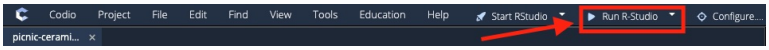
Tidyverse

Tidyverse is a collection of R packages that help users perform useful tasks such as creating graphics, reading data, manipulating data, using functions to sort and organize data, and more. Some of these packages include: `ggplot2`, `dplyr`, `tidyr`, `readr`, and `tibble`.

To access Tidyverse to start using these packages, we include `library(tidyverse)` within our code in the RStudio text editor.

Note that Tidyverse is not required to perform the tasks mentioned above. You can still use base R to perform the same tasks. However, for the purposes of this course and for ease of use, we will be using some of the packages in Tidyverse.

IMPORTANT:
As you work through the modules, if you do not see RStudio being opened automatically, you can always click on the `Run R-Studio` button at the top menu to start it.



.guides/img/num/activate-rstudio

Importing Data

Before we begin, let's open up the `wrangle.r` file within RStudio. See instructions below:

info

Open the `wrangle.r` file

Within RStudio, open the `wrangle.r` file by selecting: `File` —> `Open File...` —> `code` —> `import` —> `wrangle.r`

Reading Data

To read data and store that data within a variable (or object), you can use variations of the `read()` command. Here are some of them and the types of files the command can read:

- `read_csv()` - reads comma-separated (CSV) files
- `read_tsv()` - reads tab-separated files
- `read_delim()` - reads general delimited files
- `read_fwf()` - reads fixed-width files
- `read_table()` - reads tabular files where columns are separated by white-space
- `read_log()` - reads web log files

For this course, we will be working mostly with CSV files, which is the most common file type that data is typically stored in. We have to specify the file name and path where the data sits in order for us to read the file. Use the following command to import data into RStudio and to store that data into the variable `d`.

```
d <- read_csv("data/biostats.csv")
```

To take a look at what the data looks like, you can print `d`.

```
print(d)
```

After clicking the `Source` button, you should see something that looks like the following in the console:

```
> d <- read_csv("data/biostats.csv")
Parsed with column specification:
cols(
  Name = col_character(),
  `"Sex"` = col_character(),
  `"Age"` = col_integer(),
  `"Height (in)"` = col_integer(),
  `"Weight (lbs)"` = col_integer()
)

> print(d)
# A tibble: 18 x 5
   Name  `"Sex"` `"Age"` `"Height (in)"` `"Weight (lbs)"`
   <chr> <chr>     <int>           <int>            <int>
 1 Alex  "\"M\""      41              74              170
 2 Bert  "\"M\""      42              68              166
 3 Carl  "\"M\""      32              70              155
 4 Dave  "\"M\""      39              72              167
 5 Elly  "\"F\""      30              66              124
 6 Fran  "\"F\""      33              66              115
 7 Gwen  "\"F\""      26              64              121
 8 Hank  "\"M\""      30              71              158
 9 Ivan  "\"M\""      53              72              175
10 Jake  "\"M\""      32              69              143
11 Kate  "\"F\""      47              69              139
12 Luke  "\"M\""      34              72              163
13 Myra  "\"F\""      23              62               98
14 Neil  "\"M\""      36              75              160
15 Omar  "\"M\""      38              70              145
16 Page  "\"F\""      31              67              135
17 Quin  "\"M\""      29              71              176
18 Ruth  "\"F\""      28              65              131
```

.guides/img/import/read-csv

The above output contains more than just the data inside the file. The output gives you a lot of details such the dimensions or "tibble" of the data as well as the data type that each column holds (e.g. `<int>` for integer and `<chr>` for character). Although these details are helpful, they can be distracting. Thus, moving forward, we will make use of `read.csv()` (with a period) instead of `read_csv()` (with an underscore). `read.csv()` shows all of the data without any of the additional details that `read_csv()` shows.

Try the following commands:

```
d <- read.csv("data/biostats.csv")
print(d)
```

And you will see something like:

```
> d <- read.csv("data/biostats.csv")

> print(d)
   Name      Sex Age Height..in. Weight..lbs.
1  Alex        M  41          74          170
2  Bert        M  42          68          166
3  Carl        M  32          70          155
4  Dave        M  39          72          167
5  Elly        F  30          66          124
6  Fran        F  33          66          115
7  Gwen        F  26          64          121
8  Hank        M  30          71          158
9  Ivan        M  53          72          175
10 Jake        M  32          69          143
11 Kate        F  47          69          139
12 Luke        M  34          72          163
13 Myra        F  23          62           98
14 Neil        M  36          75          160
15 Omar        M  38          70          145
16 Page        F  31          67          135
17 Quin        M  29          71          176
18 Ruth        F  28          65          131
```

.guides/img/import/biostats-csv

The output above is a little bit easier to read because it only contains the data itself. The numbers all the way to the left represent the line number of each row. **Note** that there is no line number for the categories row which include: `Sex`, `Age`, `Height..in.`, and `Weight..lbs.`. `Height` is measured in inches `in.` while `Weight` is measured in pounds `lbs.`.

Parsing Data

Parsing Data

To parse data, or extract certain parts of a data collection, we can use a special symbol to help us do that. That symbol is the dollar sign symbol `$`.

For example, if we want to extract the data from the column/category `Height..in.`, I would first specify the data that I have, which is `d`, then the symbol `$`, then the category name `Height..in.`. For example,

```
h <- d$Height..in.
```

will extract all of the data from `Height..in.` from the data `d` and store that data as a vector called `h`.

Let's do the same for `Weight..lbs.`:

```
w <- d$Weight..lbs.
```

To view the extracted data, use the print command on the variables we created, `h` and `w`.

```
print(h)
print(w)
```

Console Result:

```
 [1] 74 68 70 72 66 66 64 71 72 69 69 72 62 75 70 67 71 65
 [1] 170 166 155 167 124 115 121 158 175 143 139 163  98 160 145
       135 176 131
```

The first line represents the height data and the second represents the weight data.

Storing Data in Data Frames

Using Data Frames

Unlike vectors where all of the data has to be of the same type, a data frame does not. You can create data frames to store only the relevant data that you want. Previously, we extracted `Height..in.` and `Weight..lbs.` from our data set `d` using the commands:

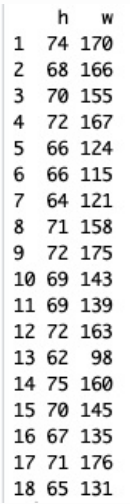
```
h <- d$Height..in.
w <- d$Weight..lbs.
```

However, that created two separate vectors, which may be difficult to read. To make things more collective, we can use the function `data.frame()` to store both variables into a data frame called `df`.

```
df <- data.frame(h, w)
```

Then we can print the data frame `df` to see what it looks like.

```
print(df)
```



	h	w
1	74	170
2	68	166
3	70	155
4	72	167
5	66	124
6	66	115
7	64	121
8	71	158
9	72	175
10	69	143
11	69	139
12	72	163
13	62	98
14	75	160
15	70	145
16	67	135
17	71	176
18	65	131

.guides/img/import/height-weight

Notice how the data frame `df` looks very much like a table. You can see how we started with the original data set `d` which included a lot of data but ended up with just two pieces of data `h` and `w` for height and weight which

we can then use to analyze later.