Contents

MIDFIELD Workshops

The *Multiple-Institution Database for Investigating Engineering Longitudinal Development* (MIDFIELD) is a partnership of higher education institutions with engineering programs. MIDFIELD contains student record data from 1988–2017 for approximately one million undergraduate, degree-seeking students at the partner institutions.

This site provides access to workshop materials for studying how undergraduate students maneuver through their curricula using MIDFIELD data.

[For more information about MIDFIELD]

Facilitators

Matthew Ohland is the MIDFIELD Director and Principal Investigator. He is Professor and Associate Head of Engineering Education at Purdue University.

Marisa Orr is the MIDFIELD Associate Director and Assistant Professor in Engineering and Science Education with a joint appointment in Mechanical Engineering at Clemson University.

Russell Long is MIDFIELD Managing Director and Data Steward. He developed the stratified data sample for the R packages used in this workshop.

Susan Lord is Director of the MIDFIELD Institute and Professor and Chair of Engineering and Professor of Electrical Engineering at the University of San Diego.

Richard Layton is the MIDFIELD Director of Data Display and Professor of Mechanical Engineering at Rose-Hulman. He is the lead developer of the R packages used in this workshop.

Publications

The MIDFIELD team has been exploring and presenting the stories in the MIDFIELD data for several years. To see a sample of our work, you can follow these links:

- Lord SM, Ohland MW, Layton RA,and Camacho MM (2019) Beyond pipeline and pathways: Ecosystem metrics. *Journal of Engineering Education*, 108, 32–56, https://doi.org/10.1002/jee.20250
- Lord SM, Layton RA, and Ohland MW (2015) Multi-Institution study of student demographics and outcomes in Electrical and Computer Engineering in the USA, *IEEE Transactions on Education*, 58(3), 141–150, http://dx.doi.org/10.1109/TE.2014.2344622
- Brawner CE, Lord SM, Layton RA, Ohland MW, and Long RA (2015) Factors affecting women's persistence in chemical engineering, *International Journal of Engineering Education* 31(6A), 1431–1447, https://tinyurl.com/y6jq58xh

[For more information about MIDFIELD publications]

Licenses

Acknowledgement

Funding provided by the National Science Foundation Grant 1545667 “Expanding Access to and Participation in the Multiple-Institution Database for Investigating Engineering Longitudinal Development.”

Chapter 1

About the MIDFIELD workshops

1.1 What is midfieldr?

Analytical tools for research in student pathways are generally scarce. The R package **midfieldr** provides an entry to this type of intersectional research.

midfieldr is a R package that provides functions for turning student-record data into persistence metrics.

midfielddata is an R package that provides a stratified sample of the MIDFIELD data. This data package contains student records for 98,000 students, suitable for practice with the midfieldr package.

[For more information about midfieldr]
[For more information about midfielddata]

1.2 Why R?

R is an open source language and environment for statistical computing and graphics (**?**), ranked by IEEE in 2018 as the 7th most popular programming language (Python, C++, and Java are the top three) (**?**). If you are new to R, some of its best features, paraphrasing Wickham (**?**), are:

- R is free, open source, and available on every major platform, making it easy for others to replicate your work.
- More than 14,000 open-source R packages are available (April 2019). Many are cutting-edge tools.
- R packages provide deep-seated support for data analysis, e.g., missing values, data frames, and subsetting.
- R packages provide powerful tools for communicating results via html, pdf, docx, or interactive websites.
- It is easy to get help from experts in the R community.

RStudio, an integrated development environment (IDE) for R, includes a console, editor, and tools for plotting, history, debugging, and workspace management as well as access to GitHub for collaboration and version control (**?**).

[For more information about R]
[For more information about RStudio]

1.3 Why R graphics?

Charles Kostelnick (**?**) writes, “The array of design options in software like Microsoft Excel and PowerPoint creates the **illusion** of flexibility. … So marvelously malleable are these graphical effects—but for whom and to what end? Paradoxically, then, even as the technology for visualizing data has become more sophisticated, it does not necessarily engender rhetorically sensitive design.”

The graphics tools in R provide the means to control every pixel in the service of “rhetorically sensitive design.” Designers can craft their visual arguments to balance logos, ethos, pathos, and kairos as appropriate for a given audience in a given rhetorical situation.

[For a gallery of R graphics]

Chapter 2

MIDFIELD Institute 2019

MIDFIELD Institute
June 3–4, 2019
Purdue University, West Lafayette, Indiana, USA
Neil Armstrong Hall of Engineering
Room B-098 (basement)
Contact: Russell Long, ralong@purdue.edu with questions
Registration: http://www.conf.purdue.edu/MIDFIELD.

2.1 Description

We are offering the first Multiple Institution Database for Investigating Engineering Longitudinal Development (MIDFIELD) Institute on Monday June 3 and Tuesday June 4, 2019 in West Lafayette, Indiana.

We welcome faculty, staff, and graduate students.

Our learning objectives can be categorized in two broad classes: qualitative and computational. Qualitatively, by the end of the workshop participants should be able to:

1. Describe the data available in MIDFIELD
2. Describe how the MIDFIELD data are organized
3. Describe key principles of effective data visualization
4. Identify deficiencies of common graph types
5. List potential resources beyond MIDFIELD that can contribute to answering research questions

Computationally, participants should be able use **midfieldr**, an R package specifically designed for use with MIDFIELD, to:

1. Calculate and evaluate educational metrics
2. Produce a table of data that addresses a research question
3. Explore and tell a story from MIDFIELD data

2.2 Before you arrive

- MIDFIELD Workshops for an introduction to MIDFIELD and the workshop facilitators.
- Getting started for pre-workshop software installation instructions—assuming you plan on using your own laptop. If not, we will have computers available onsite.
- Optional: For several years now, we have been using R to explore and present the stories in the MIDFIELD data. To see a sample of our data graphics, you can follow the links in the publications section.

top of page

2.3 Sunday agenda

2019-06-02

This is an **optional session** designed for R novices.

Time	Activities
4:30–5:30 pm	*Optional.* Time to help anyone needing assistance with the software installation.
5:30–6:00 pm	*Introductions.* Introduce presenters, participants, and learning objectives. Verify that software is installe
6:00	Pizza should arrive
6:00–6:50	[*R basics.*](#r-basics) The RStudio environment and R objects, functions, and scripts
7:00–7:50	[*Graph basics.*](#graph-basics) Meet ggplot, geom layers, aesthetic mappings, and facets. Writing graph
8:00–8:50	[*Data basics.*](#data-basics) Data import, data structure, and data transformation. Writing data to file.
9:00	*Adjourn*

top of page

2.4 Monday agenda

2019-06-03

Time	Activities
8:00-9:00	*Breakfast.* Provided in the workshop room.
9:00–10:00	*Introductions.* Introduce presenters, participants, objectives, MIDFIELD, and persistence metrics we have
	Break
10:15–12:15	*Guided practice.* Self-paced tutorials using midfieldr and midfielddata.
12:15–1:15	*Lunch.* Provided in the workshop room.
1:15–2:15	*Defining your question.* Examples of things to look out for and consider when exploring research questions
	Break
2:30–4:30	*Self-directed practice.* Review the variables in midfielddata. Define a problem involving the data that inte
4:30–4:45	*Conclusion.* Reflection and discussion
	Break
5:00–7:00	*Reception.* Purdue Union, Anniversary Drawing Room
7:00	*Adjourn.* Make your own dinner plans.

top of page

2.5 Tuesday agenda

2019-06-04

Time	Activities
8:00–9:00	*Breakfast.* Provided in the workshop room.
9:00–10:00	*Data visualization.* Deficiencies of common graphs. Creating more effective graphs.
	Break
10:15–10:45	*Explore data.* Finding and presenting stories in the data.
10:45–12:15	*Self-directed work.* Continue the collaborative work from Day 1. Produce data displays that address your
12:15–3:15	*Lunch and poster preparation.* Lunch provided in the workshop room. Participants make posters to displa
3:15–4:15	*Poster session.*
4:15–5:00	*Conclusion.* Summarize objectives. Plans for proposed 2020 FIE Special Session to showcase participants'
5:00	*Adjourn*

top of page

Chapter 3

Getting started with R

If you already have R and RStudio installed, please update to the most recent releases and update your R packages as well.

If you are joining us for the first time, it is vital that you attempt to set up your computer with the necessary software in advance or it will be difficult to keep up.

Unless noted otherwise, we assume the reader is an R novice. Thus the first steps are to install R and RStudio.

3.1 Install R and RStudio

Windows users may have to login as an Administrator (localmgr) before installing the software.

- Install R for your operating system
- Install RStudio, a user interface for R

If you need additional assistance for Mac OS or Linux, these links might be useful

- Install R and RStudio on Mac OS by Michael Galarnyk (or you can Google more recent instructions)
- How to Install R Ubuntu 16.04 Xenial by Kris Eberwein (or you can Google more recent instructions)

Once the installation is complete, you can take a 2-minute tour of the RStudio interface. Please use headphones or ear-buds if you watch the video during the workshop.

- Let's start (00:57–02:32) by R Ladies Sydney (**?**)

The same video includes a longer (7 minute) tour of the four quadrants (panes) in RStudio if you are interested.

- The RStudio quadrants (07:21–14:40) by R Ladies Sydney (**?**)

3.2 Install an R package

The fundamental unit of shareable code in R is the *package*. For the R novice, an R package is like an "app" for R—a collection of functions, data, and documentation for doing work in R that is easily shared with others (**?**).

Most packages are obtained from the CRAN website (**?**). To install a package using RStudio:

- Launch RStudio

The RStudio interface has several panes. We want the Files/Plots/Packages pane.

- Select the *Packages* tab

Next,

- Click *Install* on the ribbon
- In the dialog box, type **tidyverse**

- Check the *Install dependencies* box
- Click the *Install* button
- Repeat to install the package **devtools**

Alternatively (for future reference), if you prefer using the command-line, you can install a CRAN package (or a vector of packages) by typing `install.packages()` in the Console, for example,

```r
install.packages(pkgs = c("tidyverse", "devtools"))
```

Some packages are archived in a repository other than CRAN, GitHub being a current favorite. For such packages, we use `install_github()` from the devtools package in this form,

```r
devtools::install_github(repo = "user_name/repo_name")
```

3.3 Install midfielddata and midfieldr

In this workshop, we work with the **midfieldr** package and **midfielddata** data-package. The **midfielddata** package is too large to be stored in CRAN, so we use a special "drat-repository" to make the package source files available. We install these packages by typing lines of code in the Console at the prompt.

The Console in the default RStudio pane layout is on the left. The R command prompt in the Console is `>`.

At the prompt, type a line of code and press *Enter* from your keyboard. Alternatively, you can copy a line of code from this page, paste it in the console, and press *Enter*. We only run these lines of code once, so you do not have to type the lines into a script.

Install **midfielddata** from the our drat repo. The data package is large so this step takes time. Be patient and wait for the Console prompt `>` to reappear.

```r
install.packages(pkgs = "midfielddata",
        repos = "https://MIDFIELDR.github.io/drat/",
        type  = "source")
```

In the Console, load the package by typing,

```r
library("midfielddata")
```

If that installation was successful, only then can you install **midfieldr** from its GitHub repo,

```r
devtools::install_github(repo = "MIDFIELDR/midfieldr")
```

In the Console, load the package by typing,

```r
library("midfield")
```

If the installation was successful, at the prompt you can type

```
? midfieldr
```

and see the **midfieldr** help page in the RStudio Help pane.

3.4 Create an R project

To begin any project, we create an RStudio *Project* file and directory. You can recognize an R project file by its *.Rproj* suffix.

We will create a project named after the workshop, for example, `midfield_institute.Rproj`, `fie_workshop.Rproj`, etc.

If you prefer your instructions with commentary (please use headphones or ear-buds if you watch the video during the workshop),

- Start with a Project (02:34–04:50) by R Ladies Sydney (**?**)

If you prefer basic written instructions,

- RStudio, *File > New Project... > New Directory > New Project*
- Or, click the *New Project* button in the Console ribbon,

In the dialog box that appears,

- Type the workshop name as the directory name, for example, `midfield_institute`, `fie_workshop`, etc.

- Use the browse button to select a location on your computer to create the project folder

- Click the *Create Project* button

3.5 Create directories

While file organization is a matter of personal preference, we ask that you use the directory structure shown here for your work in the workshop.

Create three folders in the project main directory, where `your_project` is the name you gave the project, foe example, `midfield_institute` or `fie_workshop`.

If you prefer your instructions with commentary (please use headphones or ear-buds if you watch the video during the workshop),

- Make some folders (04:50–06:08) by R Ladies Sydney (**?**)

If you prefer basic written instructions,

- use your usual method of creating new folders on your machine
- or you can use the *New Folder* button in the Files pane

We use the folders as follows:

- `data` for data files
- `figures` for finished data displays

- `scripts` for R scripts that operate on data to produce results

And that concludes the setup.

Chapter 4

MIDFIELD Pre-Institute Workshop

Agenda [link]

This is an **optional session** designed for R novices. If you cannot attend this session on Sunday, you are welcome to work these tutorials on your own before the Monday/Tuesday sessions.

The tutorials give the R novice a quick introduction to three essential elements of data science using R:

- R basics
- Graph basics
- Data basics

The tutorials are designed to be completed by an R novice in less than 50 minutes each. The timing has been student-tested, but of course your mileage may vary.

top of page

4.1 R basics

An introduction to R adapted from (?) with extra material from (?). If you already have R experience, you might still want to browse this section in case you find something new.

If the prerequisites have been met, the tutorial should take no longer than 50 minutes.

4.1.1 Prerequisites

- We assume you have completed all of the Getting started instructions.
- Run `midfield_institute.Rproj` to start every work session

Packages

- Run the following line of code in the RStudio Console to install the **socviz** package.

```
devtools::install_github("kjhealy/socviz")
```

Use *File > New File > R Script* to create a new R script

- Name the script `01-R-basics.R` (R filenames can begin with numbers)
- Save it in the `scripts` directory

Add a minimal header at the top of the script (if you wish) then use `library()` to load the packages we will use. Loading all the libraries at the top of a script is conventional practice.

```r
# workshop R basics
# name
# date

library("tidyverse")
library("socviz")
```

Guidelines

- In this script type the lines of code in the tutorial below one line at a time.
- After every line, *File > Save*, and hit the *Source* button to run the code.
- Confirm that your result matches the result in the tutorial.
- **Your turn** exercises give you chance to devise your own examples and check them out. You learn by doing (but you knew that already)!

4.1.2 Everything in R has a name

In R, every object has a name.

- named entities, like `x` or `y`

- data you have loaded, like `my_data`
- functions you use, like `sin()`

Some names are forbidden

- reserved words, like `TRUE` or `FALSE`
- programming words, like `Inf`, `for`, `else`, and `function`
- special entities, like `NA` and `NaN`

Some names should not be used because they name commonly used functions

- `q()` quit
- `c()` combine or concatenate
- `mean()`
- `range()`
- `var()` variance

Names in R are case-sensitive

- `my_data` and `My_Data` are different objects
- I follow the style guide used in the tidyverse by naming things in lower case, with words separated by underscores, and no spaces

If you want to know if a name has already been used in a package you have loaded, go to the RStudio console, type a question mark followed by the name, e.g.,

- `? c()`
- `? mean()`

If the name is in use, a help page appears in the RStudio Help pane.

4.1.3 Everything in R is an object

Origins of R objects

- Some objects are built in to R
- Some objects are loaded with packages
- Some objects are created by you

Type this line of code in your script, Save, Source. `c()` is the function to combine or concatenate its elements to create a vector.

```
c(1, 2, 3, 1, 3, 25)
```

In these notes, everything that comes back to us in the Console as the result of running a script is shown prefaced by `#>`. For example, after running your script, the Console should show,

```
#> [1]  1  2  3  1  3 25
```

But what is that `[1]` here? It's just a row label. We'll go into that later, not needed yet.

We can assign the vector to a name.

```
x <- c(1, 2, 3, 1, 3, 25)
y <- c(5, 31, 71, 1, 3, 21, 6)
```

To see the result in the Console, type the object name in the script, Save, and Source. (Remember, type the line of code but not the line prefaced by `#>`—that's the output line so you can check your results.)

```
x
#> [1]  1  2  3  1  3 25

y
#> [1]  5 31 71  1  3 21  6
```

You create objects my assigning them names

- `<-` is the assignment operator (keyboard shortcut: ALT –)
- objects exist in your R project workspace, listed in the RStudio Environment pane

Datasets are also named objects, and a large number of datasets are included in the base R installation. For example,`LakeHuron` contains annual measurements of the lake level, in feet, from 1875–1972.

```
LakeHuron
#> Time Series:
#> Start = 1875
#> End = 1972
#> Frequency = 1
#>  [1] 580.38 581.86 580.97 580.80 579.79 580.39 580.42 580.82 581.40 581.32
#> [11] 581.44 581.68 581.17 580.53 580.01 579.91 579.14 579.16 579.55 579.67
#> [21] 578.44 578.24 579.10 579.09 579.35 578.82 579.32 579.01 579.00 579.80
#> [31] 579.83 579.72 579.89 580.01 579.37 578.69 578.19 578.67 579.55 578.92
#> [41] 578.09 579.37 580.13 580.14 579.51 579.24 578.66 578.86 578.05 577.79
#> [51] 576.75 576.75 577.82 578.64 580.58 579.48 577.38 576.90 576.94 576.24
#> [61] 576.84 576.85 576.90 577.79 578.18 577.51 577.23 578.42 579.61 579.05
#> [71] 579.26 579.22 579.38 579.10 577.95 578.12 579.75 580.85 580.41 579.96
#> [81] 579.61 578.76 578.18 577.21 577.13 579.10 578.25 577.91 576.89 575.96
#> [91] 576.80 577.68 578.38 578.52 579.74 579.31 579.89 579.96
```

Now you can see how the row labels work. There are 10 numbers per row, here, so the second row starts with the 11th, indicated by `[11]`. The last row starts with the 91st value `[91]` and ends with the 98th value.

- In the Console, type `? LakeHuron` to see the help page for the data set

Individual elements of a vector are obtained using `[]` notation.

For example, the first five lake level readings are extracted with

```
LakeHuron[1:5]
#> [1] 580.38 581.86 580.97 580.80 579.79
```

The 4th element alone,

```
LakeHuron[4]
#> [1] 580.8
```

4.1.4 Do things in R using functions

Functions do something useful

- functions are objects the perform actions for you
- functions produce output based on the input it receives
- functions are recognized by the parentheses at the end of their names

The parentheses are where we include the inputs (arguments) to the function

- `c()` concatenates the comma-separated numbers in the parentheses to create a vector
- `mean()` computes the mean of a vector of numbers
- `sd()` computes the standard deviation of a vector of numbers
- `summary()` returns a summary of the object

If we try `mean()` with no inputs, we get an error statement

```
mean()
#> Error in mean.default() : argument "x" is missing, with no default
```

If we use the Lake Huron dataset as the argument, the function is computed and displayed. Add these lines to your script, Save, and Source.

```
mean(LakeHuron)
#> [1] 579.0041

sd(LakeHuron)
#> [1] 1.318299

summary(LakeHuron)
#>    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#>   576.0   578.1   579.1   579.0   579.9   581.9
```

We can extract subsets of data using functions. For example, If we wanted only the first five even-numbered elements, we use `c()` to create a vector of indices to the desired elements,

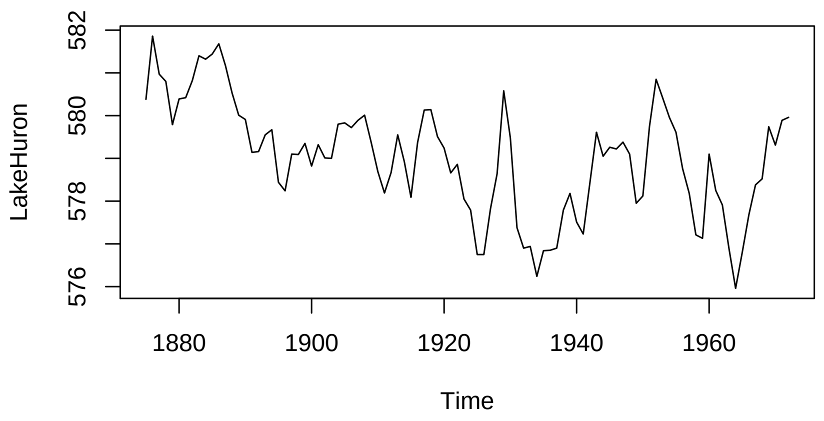
```
LakeHuron[c(2, 4, 6, 8, 10)]
#> [1] 581.86 580.80 580.39 580.82 581.32
```

If we wanted every 5th entry over the full data set, we use `length()` to determine how many entries there are, and the sequence function `seq()` to create the vector of indices,

```
n <- length(LakeHuron)
LakeHuron[seq(from = 5, to = n, by = 5)]
#>  [1] 579.79 581.32 580.01 579.67 579.35 579.80 579.37 578.92 579.51 577.79
#> [11] 580.58 576.24 578.18 579.05 577.95 579.96 577.13 575.96 579.74
```

Because we will be using the ggplot2 package for graphics, we will not be using the base R `plot()` function very often, but it is useful for a quick look at data. Add these lines to your script, Save, and Source.

```
plot(LakeHuron)
```



The help pages for functions are quickly accessed via the Console. In the Console type one line at a time and *Enter* to see the function help page.

- `? mean()`
- `? sd()`
- `? summary()`

4.1.5 R functions come in packages

Functions are bundled in packages

- Families of useful functions are bundled into packages that you can install, load, and use
- Packages allow you to build on the work of others
- You can write your own functions and packages too
- A lot of the work in data science consists of choosing the right functions and giving them the right arguments to get our data into the form we need for analysis or visualization

Functions operate on the input you provide and give you back a result. Type the following in your script, Save, and Source.

```
table(x) # table of counts
#> x
#>  1  2  3 25
#>  2  1  2  1

sd(y)    # standard deviation
#> [1] 25.14435

x * 5    # multiply every element by a scalar
#> [1]   5  10  15   5  15 125

y + 1    # add a scalar to every element
#> [1]  6 32 72  2  4 22  7
```

```
x + x     # add elements
#> [1]  2  4  6  2  6 50
```

Comments are annotations to make the source code easier for humans to understand but are ignored by R. Comments in R are denoted by a hashtag `#`.

4.1.6 R objects have class

Everything is an object and every object has a class.

```
class(x)
#> [1] "numeric"

class(summary)
#> [1] "function"
```

Certain actions will change the class of an object. Suppose we try create a vector from the `x` object and a text string,

```
new_vector <- c(x, "Apple")

new_vector
#> [1] "1"     "2"     "3"     "1"     "3"     "25"    "Apple"

class(new_vector)
#> [1] "character"
```

By adding the word "Apple" to the vector, R changed the class from "numeric" to "character". All the numbers are enclosed in quotes: they are now character strings and cannot be used in calculations.

The most common class of data object we will use is the data frame.

```
titanic # data in the socviz package
#>       fate    sex    n percent
#> 1 perished   male 1364    62.0
#> 2 perished female  126     5.7
#> 3 survived   male  367    16.7
#> 4 survived female  344    15.6

class(titanic)
#> [1] "data.frame"
```

You can see there are four variables: fate, sex, n, percent. Two variables (columns) are numeric, two are categorical.

You can pick variable out of a data frame using the `$` operator,

```
titanic$percent
#> [1] 62.0  5.7 16.7 15.6
```

From the tidyverse, we will regularly use a augmented data frame called a *tibble.* We can convert the titanic data frame to a tibble using `as_tibble()`.

```
titanic_tb <- as_tibble(titanic)

class(titanic_tb)
#> [1] "tbl_df"     "tbl"        "data.frame"
```

```
titanic_tb
#> # A tibble: 4 x 4
#>   fate     sex        n percent
#>   <fct>    <fct>  <dbl>   <dbl>
#> 1 perished male    1364    62  
#> 2 perished female   126     5.7
#> 3 survived male     367    16.7
#> 4 survived female   344    15.6
```

The tibble includes additional information about the variables

4.1.7 R objects have structure

To see inside an object ask for its structure using the `str()` function.

```
str(x)
#>  num [1:6] 1 2 3 1 3 25

str(titanic)
#> 'data.frame':    4 obs. of  4 variables:
#>  $ fate   : Factor w/ 2 levels "perished","survived": 1 1 2 2
#>  $ sex    : Factor w/ 2 levels "female","male": 2 1 2 1
#>  $ n      : num  1364 126 367 344
#>  $ percent: num  62 5.7 16.7 15.6

str(titanic_tb)
#> Classes 'tbl_df', 'tbl' and 'data.frame':    4 obs. of  4 variables:
#>  $ fate   : Factor w/ 2 levels "perished","survived": 1 1 2 2
#>  $ sex    : Factor w/ 2 levels "female","male": 2 1 2 1
#>  $ n      : num  1364 126 367 344
#>  $ percent: num  62 5.7 16.7 15.6
```

I also like to use the `glimpse()` function from the tidyverse.

```
glimpse(x)
#>  num [1:6] 1 2 3 1 3 25

glimpse(titanic)
#> Observations: 4
#> Variables: 4
#> $ fate    <fct> perished, perished, survived, survived
#> $ sex     <fct> male, female, male, female
#> $ n       <dbl> 1364, 126, 367, 344
#> $ percent <dbl> 62.0, 5.7, 16.7, 15.6

glimpse(titanic_tb)
#> Observations: 4
#> Variables: 4
#> $ fate    <fct> perished, perished, survived, survived
#> $ sex     <fct> male, female, male, female
#> $ n       <dbl> 1364, 126, 367, 344
#> $ percent <dbl> 62.0, 5.7, 16.7, 15.6
```