Lecture 2: Getting started with the tidyverse

Katie Schuler

2023-08-31

1 Tidyverse basics

> The tidyverse is an opinionated collection of R packages designed for data science. All packages share an underlying design philosophy, grammar, and data structures. ~ Tidyverse package docs

The `tidyverse` collection of packages includes:

- `ggplot2` - for data visualization
- `dplyr` - for data wrangling
- `readr` - for reading data
- `tibble` - for modern data frames
- `stringr`: for string manipulation
- `forcats`: for dealing with factors
- `tidyr`: for data tidying
- `purrr`: for functional programming

We load the `tidyverse` like any other package, with `library(tidyverse)`. When we do, we will receive a message with (1) a list packages that were loaded and (2) a warning that there are potential conflicts with base R's `stats` functions

- We can resolve conflicts with the `::` operator, which allows us to specify which package our intended function belongs to as a prefix: `stats::filter()` or `dplyr::filter()`

2 What is tidy data?

The same underlying data can be represented in a table in many different ways; some easier to work with than others. The tidyverse makes use of tidy data principles to make datasets easier to work with in R. **Tidy data** provides a standard way of structuring datasets:

1. each variable forms a **column**; each column forms a variable

2. each observation forms a **row**; each row forms an observation
3. value is a **cell**; each cell is a single value

Why is tidy data easier to work with?

- Because consistency and uniformity are very helpful when programming
- Variables as columns works well for vectorized languages (R!)

3 Functional programming with purrr

> purrr enhances R's functional programming (FP) toolkit by providing a complete and consistent set of tools for working with functions and vectors. If you've never heard of FP before, the best place to start is the family of map() functions which allow you to replace many for loops with code that is both more succinct and easier to read. ~ purrr docs

Let's illustrate the joy of the tidyverse with one of its packages: `purrr`. The docs say that the best place to start is the family of `map()` functions, so we'll do that.

The `map()` functions:

1. take a vector as input
2. apply a function to each element
3. return a new vector

We say "functions" because there are 5, one for each type of vector:

- `map()`
- `map_lgl()`
- `map_int()`
- `map_dbl()`
- `map_chr()`

To illustrate, suppose we have a data frame `df` with 3 columns and we want to compute the mean of each column. We could solve this with copy-and-paste (run `mean()` 3 different times) or try to use a `for` loop, but `map()` can do this with just one line:

```r
map_dbl(df, mean)
```

Now imagine we have 5 more data frames and we want to compute the mean of each of their columns, too. Again, we could copy and paste the `map()` function or use it in a `for` loop. But the `map` family allows us go up a layer of abstraction. We can use `pmap()` when we want to apply a function element-wise to corresponding items in multiple lists.

4 Modern data frames with tibble

> A tibble, or tbl_df, is a modern reimagining of the data.frame, keeping what time has proven to be effective, and throwing out what is not. Tibbles are data.frames that are lazy and surly: they do less and complain more ~ tibble docs

Tibbles do less than data frames, in a good way:

- never changes type of input (never converts strings to factors!)
- never changes the name of variables
- only recycles vectors of length 1
- never creates row names

You can read more in vignette("tibble") if you are interested, but understanding these differences is not necessary to be successful in the course. The take-away is that `data.frame` and `tibble` sometimes behave differently. The behavior of `tibble` makes more sense for modern data science, so we should us it instead!

Create a `tibble` with one of the following:

```
# (1) coerce an existing object with
as_tibble(x)

# (2) pass a column of vectors
tibble(x=1:5, y=1)

# (3) define row-by-row, short for traansposed tibble
tribble(
    ~x, ~y, ~z,
    "a", 2, 3.6,
    "b", 1, 8.5
)
```

We will encounter two main ways tibbles and data frames differ:

- **printing** - by default, tibbles print the first 10 rows and all columns that fit on screen, making it easier to work with large datasets. Tibbles also report the type of each column (e.g. `<dbl>`, `<chr>`)
- **subsetting** - tibbles are more strict than data frames, which fixes two quirks we encountered last lecture when subsetting with `[[` and `$`: (1) tibbles *never* do partial matching, and (2) they *always* generate a warning if the column you are trying to extract does not exist.

To test if something is a `tibble` or a `data.frame`:

- `is_tibble(x)`
- `is.data.frame(x)`

5 Reading data with readr

> The goal of readr is to provide a fast and friendly way to read rectangular data from delimited files, such as comma-separated values (CSV) and tab-separated values (TSV). It is designed to parse many types of data found in the wild, while providing an informative problem report when parsing leads to unexpected results. ~ readr docs

Often we want to read in some data we've generated or collected outside of R. The most basic and common format is **plain-text rectangular files**. We will "read" these into R with readr's `read_*()` functions.

The `read_*()` functions have two important arguments:

- **file path** - the path to the file (that reader will try to parse)
- **column specification** - a description of how each column should be converted from a character vector to a specific data type (`col_types`)

There are 7 supported file types, each with their own `read_*()` function:

- `read_csv()`: comma-separated values (CSV)
- `read_tsv()`: tab-separated values (TSV)
- `read_csv2()`: semicolon-separated values
- `read_delim()`: delimited files (CSV and TSV are important special cases)
- `read_fwf()`: fixed-width files
- `read_table()`: whitespace-separated files
- `read_log()`: web log files

To read `.csv` files, include a path and (optionally) a column specification:

```r
# (1) pass only the path; readr guesses col_types
read_csv(path='path/to/file.csv')

# (2) include a column specification with col_types
read_csv(
    path='path/to/file.csv',
    col_types = list( x = col_string(), y = col_skip() )
)
```

With no colum specification, `readr` uses the the first 1000 rows to guess with a simple heuristic:

- if column contains only T/F, `logical`
- if only numbers, `double`
- if ISO8601 standard, `date` or `date-time`
- otherwise `string`

There are 11 column types that can be specified:

- `col_logical()` - reads as boolean TRUE FALSE values
- `col_integer()` - reads as integer
- `col_double()` - reads as double
- `col_number()` - numeric parser that can ignore non-numbers
- `col_character()` - reads as strings
- `col_factor(levels, ordered = FALSE)` - creates factors
- `col_datetime(format = "")` - creates date-times
- `col_date(format = "")` - creates dates
- `col_time(format = "")` - creates times
- `col_skip()` - skips a column
- `col_guess()` - tries to guess the column

Some useful additional arguments:

- if there is **no header**, include `col_names = FALSE`
- to **provide a header**, include `col_names = c("x","y","z")`
- to **skip some lines**, include `skip = n`, where n is number of lines to skip
- to **select which columns** to import, include `col_select(x, y)`
- to **guess** column types **with all rows**, include `guess_max = Inf`

Sometimes weird things happen. The most common problems are:

- **missing values are not `NA`** - your dataset has missing values, but they are not coded as `NA` as R expects. Solve by adding `na` argument (e.g. `na=c("N/A")`)
- **column names have spaces** - R cannot include spaces in variable names, so it adds backticks (e.g. `` `brain size` ``); we can just refer to them with backticks, but if that gets annoying, see `janitor::clean_names()` to fix them!

Reading more complex file types requires functions outside the tidyverse:

- **excel** with `readxl` - see Spreadsheets in R for Data Science
- **google sheets** with `googlesheets4` - see Spreadsheets in R for Data Science
- **databases** with `DBI` - see Databases in R for Data Science
- **json data** with `jsonlite` - see Hierarchical data in R for Data Science

Further reading and references

Recommended further reading: - Data tidying in R for Data Science - Tibbles in R for Data Science - Data import in R for Data Science: - readr cheatsheet