Tidyverse Learnings

Author: Can Aytöre

Last Update: 2022-02-17

Contents

1 Introduction

1.1 Aim

1.2 Motivation

2 Get Started with `tidyverse`

2.1 The R Language

The R language is an extremely popular scripting language used by millions of people around the world. Primarily, it's used for data analysis, modeling, and visualization. Many people think of R as statistical software, but it's fundamentally not and it's a little bit upsetting when people say that. R is programming language that has been adopted and curated by people interested in doing data science as flexibly as possible.

R lives and breathes at the comprehensive R archive network, abbreviated to CRAN. When you download R from CRAN, you've actually installed Base-R. Base-R includes all of the necessary machinery for your computer to be able to run R code. It also installs standard R packages like `stats`, `utils` and `graphics`. These packages allow you to start using R immediately on your machine. In order for data manipulation, using Base R is sufficient most of the time. The Base R way of doing things involves a code that looks very much like this;

```
head(iris[iris$Species == "virginica",])
```

```
##     Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
## 101          6.3         3.3          6.0         2.5 virginica
## 102          5.8         2.7          5.1         1.9 virginica
## 103          7.1         3.0          5.9         2.1 virginica
## 104          6.3         2.9          5.6         1.8 virginica
## 105          6.5         3.0          5.8         2.2 virginica
## 106          7.6         3.0          6.6         2.1 virginica
```

First to access to 'species' column and then we have a double equals to say 'species' is equivalent to 'virginica' and then we have a comma to say we want all of the columns and then the final closing square bracket.

2.2 R Packages

It's possible to do every single thing you could possibly imagine with Base-R because it is a true and complete programming language but you would have to write a lot of code yourself. Most people jump straight into using R packages to make their life easier and more reproducible, so what are R packages? R packages are self-contained collections of functions and/or datasets that provide us with the ability to do any number of things from analyzing data, visualizing data to potentially even generating reports with R which is what R Markdown allows us to do. Now, CRAN has over 10,000 packages and this comprehensive range of packages available from CRAN is part of what makes R such a popular scripting language.

It's a fact of programming and scripting but building everything yourself from scratch is time consuming and more than likely, hugely error prone, this is why R users depend on packages. Using packages makes it easier to start working on a new project in R. Packages can make collaborating with others on R projects easier, as you can be ensured everyone is using the same code base.

3 What is Tidyverse?

There are definitely one or two packages that would make your life with R a little bit easier, i.e. they're designed to do the kind of analysis or data visualization which is important to your domain-specific knowledge. The tidyverse is an ecosystem of R packages designed to work consistently and interdependently together to provide a flexible and easy-to-understand workflow for doing data science with the R language. The fundamental building block of the tidyverse is the concept of tidy data. The tidyverse has been in development since early 2014 and is becoming increasingly mature. But the tidyverse should never be considered a replacement for Base R. It will remain crucial to understand the base R way of doing things.

- In other words, the tidyverse is both a collection of R packages, and an approach to how to do data science effectively, and reproducibly with the R language.

3.1 Why use the Tidyverse?

So what makes the tidyverse different? Well, the core of the tidyverse is developed by developers at RStudio. It's a company with an extremely good reputation including, for R package development. RStudio's own internal tools dependent on components of the tidyverse, helping to reassure us of the long term viability of the tidyverse ecosystem. Tidyverse is developed openly on GitHub, meaning users can track continuing development and if necessary, fork packages in the future if Rstudio themselves, stop updating them.

3.2 Strengths of Tidyverse

What does the tidyverse provide us as end users or data scientists? Well, using tidyverse leads to advantages in the following main areas: - Data importation - Data wrangling - Data visualization

3.2.1 Data import

Let's look at each of those. `readr` completely blows away the base-R tools for importing rectangular data files like csv (comma-separated values) and tsv (tab-separated values) files. It's not only significantly faster than base-R but it's more intelligent. For instance, automatically converting dates to dates, times to times, and converting columns that should be numbers into numbers. And finally, it never ever imports columns of strings as factors. If you're already a base-R user, chances are you've spent hours of frustration because of this issue using base-R. **This section** includes importing data with `readr` as it's for general workhorse of most R users data import toolkit. The `readxl` library makes importing from excel files ridiculously simple allowing worksheets, individual cells, or even cell ranges to be targeted for import easily. The tidyverse also aims to fit into existing workflows. `haven` allows data to be imported from SAS, SPSS, and Stata. Now these three packages significantly decrease the time needed to massage data files into R. And help solve a number of common frustrations with base-R packages. But please note that `readr` is the only package of the three that is part of the core `tidyverse`. You need to separately load `readxl` and `haven` to access those libraries.

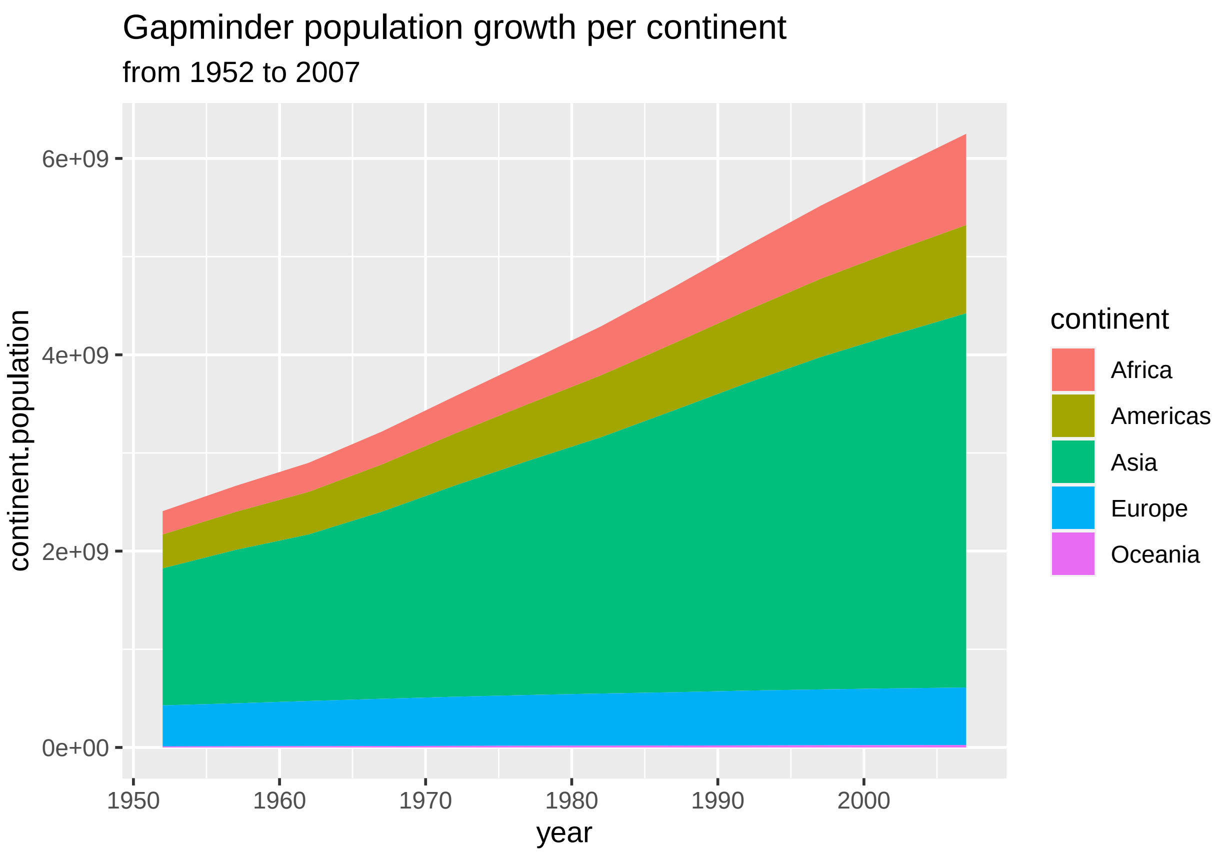
3.2.2 Data wrangling

The tidyverse utilizes the pipe operator (%>%). This percentage greater than percentage thing to provide a logical framework for chaining together common data wrangling tasks. This makes code very faster to write and easier to read. There's **a chapter** dedicated to the pipe operator as it is mysterious magic to many R users. The tidyverse is designed around a concept of *tidydata*. The `tidyr` library is designed for reshaping and transforming your imported data into a structure ready to manipulate, model, and visualize with the tidyverse. `dplyr` is the library for sub-setting, filtering, summarizing, and generally wrangling your data.

It also provides a number of tools for doing database-like operations for working on relational datasets. These packages used together form a core data processing component of the tidyverse. The operations you'll perform with these are both significantly faster and simpler to construct than simply using base-R. `ggplot2` provides a complete consistent and incredibly powerful grammar of graphics, allowing impressive static visualizations to be built with minimal effort.

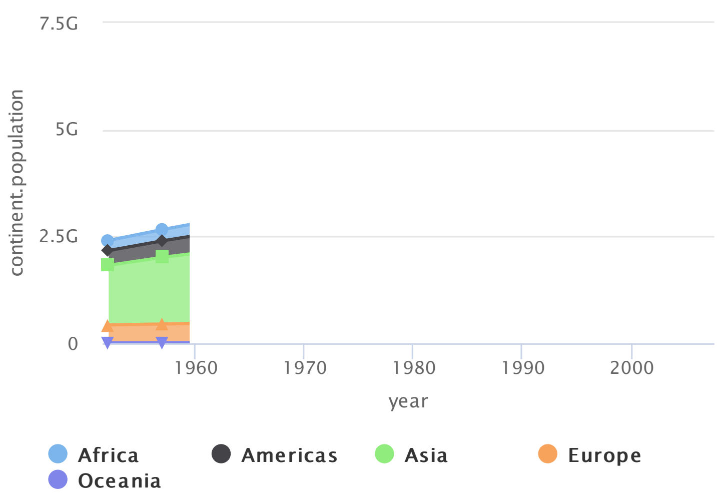
I'm going to quickly show what's possible using all the components for tidyverse together. After loading libraries `gapminder` and `tidyverse`, we have a little bit of code which uses `dplyr` (to group my data by continent and year, and then summaries to calculate the continent population). And then I use `ggplot2` to generate my chart. I have quite a beautiful looking **static** visualization generated with `ggplot2`. It's important to know that `ggplot2` provides a powerful consistent grammar for creating static shots.

```r
library("gapminder")
library("tidyverse")

gapminder %>%
  group_by(continent, year) %>%
  summarise(continent.population = sum(as.numeric(pop))) %>%
  ggplot(aes(x = year, y = continent.population)) +
  geom_area(aes(fill = continent), position = "stack") +
  ggtitle("Gapminder population growth per continent",
          subtitle = paste("from", min(gapminder$year), "to", max(gapminder$year)))
```



If you're interested in building interactive charts for the web with R, you'll need to learn about `htmlwidgets`. But in most commonly used `htmlwidgets` libraries, lean heavily on the tidyverse for preparing data for visualization. Plus all the good `htmlwidgets` are designed to use the pipe operator. Let's have a look at what we could build using `htmlwidgets` in R. We use the `dplyr` library to wrangler our data. And then we build our chart in the `highcharter` library. So instead of a **static** chart, what we have now is an **interactive** chart that I can move my cursor through and get information for specific points in the dataset. I can also actually remove series if I'm interested in doing that. So, `ggplot2` is part of the `tidyverse` and allows us to build static charts in a consistent manner. `htmlwidgets` allows us to create interactive charts and often depends on the tidyverse for wrangling and constructing our data for visualization.

```
library("gapminder")
library("tidyverse")
library("highcharter")

gapminder %>%
  group_by(continent, year) %>%
  summarise(continent.population = sum(as.numeric(pop))) %>%
  hchart("area",
         hcaes(x = year,
               y = continent.population,
               group = continent)
         ) %>%
  hc_plotOptions(area = list(stacking = "normal")) %>%
  hc_tooltip(shared = TRUE)
```



And finally, there's the `shiny` library which allows us to build web applications using only the R language without knowing any html or javascript. And `shiny` allows us to use `htmlwidgets` to embed interactive charts inside of our web applications.

3.3 Maintain the Tidyverse

So how do we go about maintaining the tidyverse? Well, remember that in order to use the tidyverse on our system we've installed three discrete things: R itself, RStudio, and the tidyverse collection of packages. To keep fully up to date with the tidyverse, unfortunately we really need to keep on top of all three of these things individually. So let's look at how we do that.

The process you need to go through to keep up to date with R is the most frustrating of all the three tools since there's not a consistent way to do this from within RStudio. Base-R is updated approximately 4 times a year but there's not a regular release cycle. I'd advise that you check every few months for a new version of R. If R console tells you there's a newer version then type a y + Enter and this will launch you to your web browser to the downloads page for R for your operating system. When you're there it's really important to check the release date of that version of R. I thoroughly advise that you do not update R until the release is at least one month old. This is to give the package maintainers that you rely on enough time to update their packages to depend on this new version of R.

So how about updating RStudio? Well, RStudio will automatically check whether there's a new version of RStudio available when you open it up. Updates are typically released a few times a year and generally they include awesome new features that are of general interest. For instance support for new features in R markdown documents or simply making it easier to publish Shiny apps. I thoroughly recommend that as soon as RStudio tells you there's a new version click update, it will take you to the RStudio website and allow you to download the new version.

So how about keeping the tidyverse up to date? Well thankfully that's quite simple. The constituent components of the tidyverse don't have a regular update frequency but that doesn't matter as we have the wonderful tidyverse update function which handles everything for us.

```
tidyverse_update()
```

In RStudio simply run the function tidyverse update, it will go away and check are there new versions or packages available, and if so it will grab those versions and install them ready for you to use. Now you might sometimes hear on the grapevine that there are new features available in the development versions of the tidyverse packages. So where do these development versions live? Well they all live on GitHub and you can install the development version of a tidyverse library using the `devtools` library, if necessary. But where is this grapevine of news about tidyverse? Well, one of the best places to keep up to date with what's going on in the tidyverse, and in R in general, is on Twitter. The core RStudio development team are actually very active here and the two hashtags that you want to look out for are `#rstats` and `#tidyverse`.

So if you do want one of these development builds of a tidyverse package, you could use the `devtools` library and the function *install_github*.

```
library(devtools)
install_github("tidyverse/package_name")
```

But how about if you find that the development version isn't just working out for you, you want to return to the stable version? Well, at any time simply use the function *install.packages* and this will recover you to the crown version of all the libraries within the tidyverse.

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

4 Being Tidy with RStudio Projects

4.1 Why should we use projects in RStudio?

4.1.1 Easier import

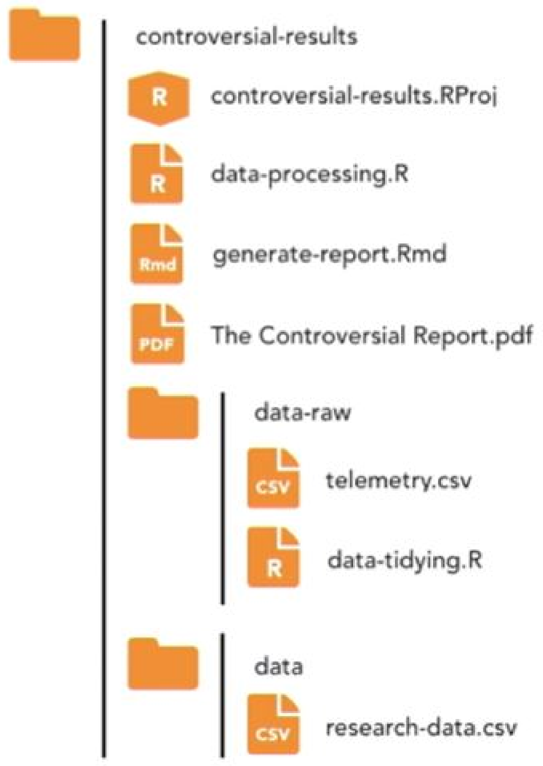
Projects are a powerful tool in RStudio for developing reproducible code, whether for individual data analyses, data-driven reports, or even developing your own packages. Projects also get you ready from the beginning of a collaborate with you, for instance, on GitHub. But the main advantage of projects is that they make your life easier when *importing data*. If you've used R before, you'll be familiar with the concept of a **working directory**. If not, then a working directory is simply the place where R is currently looking for files. If you're not using projects, then you'll likely have seen this at the top of your script files.

```
setwd("computer_name/.../data_folder")  # Path to data folder
```

Your data files live in that folder. But then if you send your code to others, they'll need to change their path to include their own computer's name and whatever long path they have, as well. Also, you'll probably actually forget to send them the data file in the first place with a script file. Projects completely negate the need to do this, because projects make the whole concept of working with files and their paths easier. You'll be actively discouraged from the terrible practice of using absolute file paths.

4.1.2 Improved Reproducibility

Reproducibility is a hot topic in research. How can we reassure others that results of our analyses or conclusions about research data are accurate, without providing the necessary code and explanations of our methodologies? R is an excellent toolkit for ensuring reproducible research, thanks to its open source underbelly. Anybody can go and look at the source code for base R, as well as the packages that you rely on. Organizing your own research into projects minimizes the work others need to do to reproduce your results.



Above, we see an exceedingly clean R project. The folder is called *'controversial-results'*, and we can see it contains both the raw telemetry behind our research in a folder called *'data-raw'*, and a tidied form of the data in a folder called *'data'*. Then the process for data wrangling is kept within the *'data-processing.R'* file, and finally, there's an R markdown file *'(.Rmd)'* which is used to generate *'The Controversial Report.pdf'* which we communicate to others. Anyone can reproduce this analysis by simply obtaining the *'controversial-results'* folder, opening up ***'controversial-results.RProj'*** file in RStudio, and then running the *'data-processing.R'* file. Finally, projects make collaboration much easier, because projects keep all your files together.

4.1.3 Improved Collaboration

They're perfectly designed for version control systems, like *Git*. All of the tidyverse R libraries are developed in *GitHub* and available as projects. So here's a repository for `haven` with a great README telling me how the package should be used and what it's for, but if I want to grab the whole package, all I need to do is go to the green button and select 'Download ZIP', navigate to my downloads folder, and then inside of the *haven-master* folder, you'll find a *.Rproj* file. And if I open that up, it opens up Rstudio, and I'm ready to begin my own modifications of the `haven` package, if I wanted to.

Projects negate the need for setting working directories, as everything becomes a relative file path to the .Rproj-containing folder. Projects improve both reproducibility and collaboration. If you want others to work with you, the best thing you can do is set up an RStudio project and host it on a version control system like GitHub.

4.2 Create a new Project

tbu..

5 Introducing the %>% Operator

The pipe operator is an incredibly important component of the modern R workflow.

5.1 What is the %>% Operator

Whenever you see percentage, greater than, percentage (%>%) in R code, you should pronounce it "pipe". The pipe operator is simply *Syntactic Sugar*, but it's incredibly popular syntactic sugar, to the extent that it's pretty much the workhorse of the tidyverse and HTML widgets, which is why it's so important to master for our uses. But what is syntactic sugar? Well, syntactic sugar is designed by developers to make code easier to read or to write for human beings. Typical use cases for syntactic sugar are: reducing the number of keystrokes needed to write code, or improving the "flow" of writing code, where "flow" simply means the stream of consciousness of the programmer. We want to minimize the amount we have to think about writing code and instead think about the task we're trying to achieve. In R, our programming tasks are typically about data manipulation. The pipe is excellent syntactic sugar for reducing the number of keypresses and emphasizing the flow of data in R. So the pipe operator is "syntactic sugar" for chaining operations together. For instance, this is piped into that, which is piped into one more thing, which is then piped into one last thing, and then piped into finished. Now let's look at a real example in R Studio. So I have R Studio open, and I'm going to create a new project in which we learn a little bit about pipes as sugar. So we're going to go to Projects in top right hand corner, New Project, we're going to select New Directory, we're going to create an empty project, we're going to make it on our desktop, and we're going to call it pipes as sugar. Now we're ready to create a script file, which we'll do with command shift N. We'll save this script file inside of our project, and we'll save that as pipes as sugar dot R. In line one, let's create a simple vector we'll call data, which stores the prime numbers up to 17. So, we'll call it data, we'll use the assignment operator, we'll open the vector, and we'll go one, three, five, seven, and 11, 13, and then 17. We will store this vector by running the code with command enter. We can see in our environment we have the value data. If we wanted to calculate the rolling differences between these numbers, we could write this using standard notation as follows: so I'll write diff data and hit command enter. And now we get the rolling differences between the numbers in our vector, but we could just as easily write this with a pipe. We could write data percentage, greater than, percentage, pipe, hit enter, a line is automatically indented, and then we could type diff, open close parentheses, and hit command enter. Now we get a warning from the R console that R Studio doesn't currently know about the pipe. It can't find the function percentage greater than percentage, and that's because it's syntactic sugar introduced by a package called magrittr. What we really should have done at the top of our script file is to load the tidyverse, as this also loads the pipe from magrittr. So let's do that. We'll go to line one, we'll enter a new line, and we'll type library, open parentheses, open quote, tidyverse, we'll run that code with command enter, we can see that the packages of tidyverse have been loaded, and now if we highlight line seven through six, and run the code, we can see we get the exact same as we would get if we were to run line four. Now, if we wanted to calculate the mean difference between the primes, it's really simple to add another operation to our pipe chain. We would simply write at the end of line seven, percentage, greater than, percentage, pipe, enter, mean. And now, if we hit command enter, our pipe chain is sent to the console, and we get the mean differences between the primes between one and 17. If we were to write this in standard notation, we would have to rewrite the code on line four, we would have to go to the beginning of the line, we would have to write mean, open parentheses, go to the end of the line, and then close parentheses. While this is technically fewer keypresses, we have had to rewrite our code. We don't have the steps in the operation we want to perform obvious within our code. So, in traditional R notation, expressions need to be rewritten for new operations to be added. If you need to move to the beginning of the line, and to the end of the line to add a closing parentheses. Whereas with the pipe operator, one can simply continue to chain together your thought process, and data processing tasks, without having to interrupt yourself to reorganize the code. That's because the pipe simply chains together operations.

5.2 Identify where to use %>%

5.3 Significance of %>%

5.4 Alternate options to %>%

6 Importing, Modifying, and Filtering Data

References

Appendices

About Author