

An educator's perspective of the tidyverse

Mine Çetinkaya-Rundel *

Department of Statistical Science, Duke University and RStudio
and

Johanna Hardin †

Department of Mathematics and Statistics, Pomona College
and

Benjamin S. Baumer ‡

Program in Statistical & Data Sciences, Smith College
and

Amelia McNamara §

Department of Computer & Information Sciences, University of St Thomas
and

Nicholas J. Horton ¶

Department of Mathematics and Statistics, Amherst College
and

Colin W. Rundel ||

Department of Statistical Science, Duke University

August 10, 2021

Abstract

Computing makes up a large and growing component of data science and statistics courses. Many of those courses, especially when taught by faculty who are statisticians by training, teach R as the programming language. A number of instructors have opted to build much of their teaching around use of the **tidyverse**. The tidyverse, in the words of its developers, “is a collection of R packages that share a high-level

*mc301@duke.edu

†jo.hardin@pomona.edu

‡bbaumer@smith.edu

§amelia.mcnamara@stthomas.edu

¶nhorton@amherst.edu

||colin.rundel@duke.edu

design philosophy and low-level grammar and data structures, so that learning one package makes it easier to learn the next” ([Wickham et al. 2019](#)). The shared principles have led to the widespread adoption of the tidyverse ecosystem. No small part of this usage is because the tidyverse tools have been intentionally designed to ease the learning process and cognitive load for users as they engage with each new piece of the larger ecosystem. Moreover, the functionality offered by the packages within the tidyverse spans the entire data science cycle, which includes data import, visualisation, wrangling, modeling, and communication. We believe the tidyverse provides an effective and efficient pathway to data science mastery for students at a variety of different levels of experience. In this paper, we introduce the tidyverse from an educator’s perspective, touching on the what (a brief introduction to the tidyverse), the why (pedagogical benefits, opportunities, and challenges), the how (scoping and implementation options), and the where (details on courses, curricula, and student populations).

Keywords: R language, teaching, data science, statistics education, statistical computing

1 Introduction

Computing has a fundamental and growing role in the statistics and data science curriculum (Nolan & Temple Lang 2010, Horton & Hardin 2021). The revised Guidelines for Assessment and Instruction in Statistics Education (GAISE) College report notes the importance of technology and states: “ideally, students should be given numerous opportunities to analyze data with the best available technology (preferably, statistical software)” (Carver et al. 2016). In both statistics and data science courses, we believe it is important to teach authentic tools, i.e., tools that are used by practitioners of these disciplines (academics and industry professionals).

When it comes to an authentic tool, McNamara (2019) argues that a modern statistical computing tool “should be accessible, provide easy entry, privilege data as a first-order object, support exploratory and confirmatory analysis, allow for flexible plot creation, support randomization, be interactive, include inherent documentation, support narrative, publishing, and reproducibility, and be flexible to extensions,” and that such tools ideally allow new users and professionals to “reach across the gap” between tools for teaching and tools for doing to foster continued learning (McNamara 2019, 2015). R exhibits all of these attributes, particularly with careful thought toward pedagogy. It is also flexible, powerful, and open-source. As a result, many instructors, particularly those with a background in statistics, have chosen R (R Core Team 2021) for their teaching.

There are many pedagogical decisions that arise when an instructor chooses to teach with a particular computational platform or tool. Here we describe how the **tidyverse**, a collection of packages intended to provide a consistent interface in R, reduces friction for both the instructor and the student across the entire data analysis cycle. Like many other instructors, we have opted to build much of our teaching around use of the tidyverse. This paper is a synthesis of the reasoning for our choice, along with benefits and challenges associated with teaching the tidyverse.

This paper focuses primarily on undergraduate introductory statistics and data science courses. However, we will also comment on how an introduction to R and the tidyverse in these courses may help prepare a student for success in higher level courses at the undergraduate and graduate levels, as well as in industry.

There are several popular, free, open-source, programming languages that can be used in introductory statistics and data science, including R ([R Core Team 2021](#)), Python ([Van Rossum & Drake Jr 1995](#)), and Julia ([McNicholas & Tait 2019](#)). These languages also display many attributes promulgated by [McNamara \(2019\)](#). We note that the influential [Data 8](#) course at the University of California, Berkeley (as well as the follow-up [Data 100](#) course) are taught in Python, though a significant portion of the instruction centers around a course-specific Python library. We affirm that students benefit from developing literacy in multiple languages and argue that the “tidy data” ([Wickham 2014](#)) approach central to the tidyverse is programming language independent, with notable implementations in the three languages mentioned above as well as domain specific languages like SQL (see [Section 4.6](#)). We have chosen to focus our attention on R, in part because there are good models for teaching statistics and data science with reproducible computing practices—even at the introductory level ([Baumer et al. 2014](#), [Beckman et al. 2021](#)).

Instructors teaching R face a pedagogical decision about how to teach it. Some instructors use the “I’ll just teach it how I learned it” approach, which we assert is not sound pedagogical reasoning. Chances are that many things have changed since the time you first learned R. All the authors have seen major changes to R over their careers, even the most junior among us. This is one of the exciting, albeit challenging, aspects of teaching R (or any computing language or tool): the landscape is continuously evolving. While a changing landscape means instructors need to continue learning, it also means that R has become more user-friendly and student-friendly over time. Recent developments, perhaps most notably the tidyverse, have helped to round off rough edges and make R interfaces and syntax more coherent and consistent. It is important for educators to periodically reevaluate their teaching in light of what is most widely used, what is more user-friendly, what has better documentation, what has better learning resources, what has better community support, etc.

Based on all of these considerations (and more that we will spell out below), and despite the need to incorporate additional learning outcomes into our classes, we recommend teaching with the tidyverse as a way to further integrate computation into our courses and programs. Others before us have focused on how using R to teach statistics can decrease

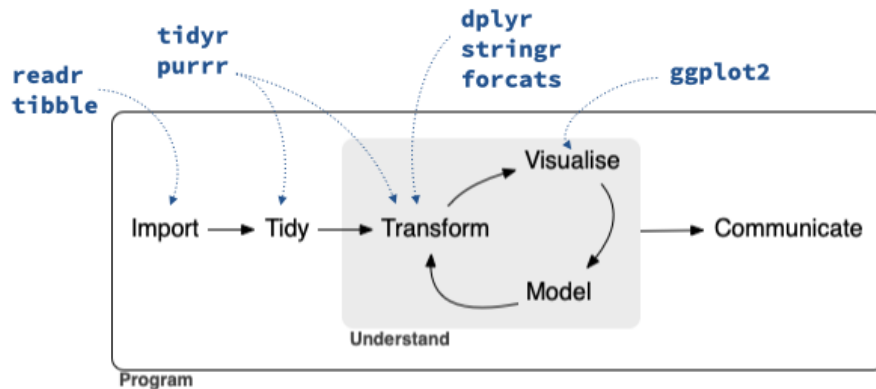


Figure 1: Data science cycle and the core tidyverse packages that address each phase.

the cognitive load of the class (Guzman et al. 2019, Tucker et al. 2021). However, we focus on the learning aspects of the tidyverse in particular and the specific merits it brings to the statistics and data science classroom.

2 What is the tidyverse?

The **tidyverse** package is a meta R package suite that loads eight core packages when invoked, and also bundles numerous other packages upon installation. These packages all share a design philosophy as well as common grammar and data structures. The core packages and the phases of the data science cycle they address are shown in Figure 1.

Wickham (2021d) outlines the design principles of the packages in the tidyverse as the following:

- **human-centered design:** The tidyverse is designed specifically to support the activities of a human data analyst.
- **consistency:** The functions in all tidyverse packages are designed with a consistent interface, which allows the user to apply what they learned in one function to another and minimizes the number of special cases one needs to remember.
- **composability:** The tidyverse functions allow users to solve complex problems by breaking them down into small pieces, many of which can be combined using the “pipe” operator (or a “+” operator for ggplot2). They support iterative exploratory

analysis to find the best solution.

- **inclusivity:** In addition to the packages and functions themselves, the tidyverse also refers to the community of people who use the packages. The packages, their documentation, and the community support all emphasize inclusivity.

Many of the core packages in the tidyverse were originally developed independently to address specific phases of the data science cycle, and subsequently came together under the tidyverse umbrella in 2016 ([Smith 2016](#)). The tidyverse packages are under active development, as the authors iterate to improve functionality.

Many, though not all, of the developers of the tidyverse packages are funded by RStudio, PBC and work full-time on their open-source development. The packages are co-developed with community contributors and released under the open source MIT license. Thus, while development of the tidyverse is funded by RStudio, the packages do not *belong* to RStudio.

2.1 Starting with a (tidy) data frame

In R, data often live in a data frame whose columns represent variables that we want to analyze. While the simple data frame structure is fundamental to understanding data science and statistics, it can be challenging for new learners.

When such a dataset is loaded into R, it is available as an object called a data frame. When using base R the variables in that data frame can be accessed with the `$` operator (e.g., `loans$loan_amount` to access the variable called `loan_amount` in the data frame `loans`). Often, students are tempted to access the `loan_amount` variable in this example by referring to it simply as `loan_amount` and not specifying the name of the data frame in which it lives. This results in a frustrating error: `Error: object 'loan_amount' not found`. Students experience this error when they think about variables as stand-alone objects as opposed to components of the data frame in which they live.

One approach to addressing the data frame versus variable challenge in base R is using the `attach()` function, which makes the variables in a data frame available in the global environment. Copying variables into the global environment is not recommended practice ([Google 2021](#)) as it can cause name collisions when data frames with identically named

variables are attached in the same environment (e.g., if you also had a `car_loans` data frame that also happened to have a variable called `loan_amount` attached). Additionally, using `attach()` to copy the variable into the global environment may muddle understanding of connections between the dataset and the variable within it. Pedagogically, we want students to understand how observations and variables are linked to a structured dataset.

A second approach to working with variables in base R is using the `with()` function, which is used for evaluating an expression within a specified environment (e.g., `with(loans_full_schema, loan_amount)` instead of `loans_full_schema$loan_amount`). Using `with()` avoids the name collision problem introduced by `attach()` but it is more verbose and requires understanding how a new function works just to access a variable in a data frame.

In the tidyverse, the first argument of each function, whether for data wrangling, visualization, or any of the more complex tasks that can be introduced later, is the data frame. Tidyverse functions allow access to variables in the data frame without having to re-specify the name of the data frame, e.g., `arrange(loans, loan_amount)` will arrange the rows of the `loans` data frame based on the value of `loan_amount`. By forcing users to work with tidy data, the tidyverse emphasizes the connection between data frames and variables, helping to underscore fundamental data science and statistics concepts while also simplifying the syntax for common data analysis tasks.

More commonly, the `arrange()` code would be written as `loans %>% arrange(loan_amount)` using the pipe operator (`%>%`) to pipe the data frame (or any result from the preceding step of the pipeline) into the subsequent function as its first argument (see Section 3.1 for more details on using the pipe operator in the tidyverse). The tidyverse strongly encourages the use of the pipe operator to construct readable, vertical data pipelines, whether it's for data wrangling or visualization. Pipe operators exist as a common feature across a number of other programming languages (e.g., the UNIX shell, JavaScript, etc.) and were introduced into the R ecosystem by the **magrittr** package (Bache & Wickham 2020). The widespread community adoption of the tidyverse has led to the introduction of a pipe operator into the base language in early 2021 with R version 4.1.0.¹

Like the `lm()` function, or systems such as SAS (which feature a common interface with

¹At present the authors recommend teaching using **magrittr**'s pipe (`%>%`) in favor of base R's pipe (`|>`)

a `DATA = statement`), tidyverse functions take a `data` argument that allows them to localize computations inside the specified data frame. The tidyverse approach is attractive because it doesn't muddy the concept of what is in the current environment (always the data as a data frame, never a variable as a vector) while making it easy for variables in a data frame to be accessed without the use of an additional function (like `with()`) or even quotation marks. Furthermore, unlike `lm()`, functions in the tidyverse always take a data frame as their *first* argument and always *return* a data frame. The consistent structure of data frames and variables makes it easier to get started with data analysis tasks without getting bogged down by language details or using more complicated programming practices.

We note that data frames within the tidyverse are stored as `tibbles` which have class `tbl_df` in addition to `data.frame`. `tibbles` act similarly to all other `data.frames` with less transformation of input (e.g., character vectors are not coerced to factors, column names are not modified). They also feature print functions which are more concise and well-behaved.

2.2 Consistent grammar and vocabulary

[Hermans & Aldewereld \(2017\)](#) develop the metaphor of programming as writing, and we extend their reasoning to assert that R is a programming language with many syntaxes (or flavors/“dialects”). Different R packages can use different syntaxes for the same idiom. Even packages included in the distribution of base R do not all have a consistent grammar and vocabulary. As stated in its design philosophy, the tidyverse strives for consistency across packages ([Wickham 2021d](#)). This makes the tidyverse syntactically different from base R for doing certain tasks, which might lead to learners of the tidyverse being less familiar with base R code, and vice versa.

Tidyverse users tend to use particular vocabulary, e.g., pipes, tibbles, and verbs, compared to base R users who are likely to speak in terms of matrices, dollar signs (`$`), and square brackets (`[]`). A third commonly used approach is the formula syntax, which is characterized by package dependencies being easier to require than an R version dependency (e.g., to use the base R pipe, all students *must* be running the latest version of R).

acterized by the use of tildes (~). The notion of a “grammar” in R code is well-established for both graphics (Wilkinson 2012, Wickham, Chang, Henry, Pedersen, Takahashi, Wilke, Woo, Yutani & Dunnington 2021a) and data wrangling (Wickham, François, Henry & Müller 2021a). In Section 4.4, we develop notions of pronunciation. In spoken language, the word ‘dialect’ refers to a variety of language with distinct vocabulary, grammar, and pronunciation, so we could consider the tidyverse as a dialect among users who read and write R code.

We contend that every educator makes a choice when they select a particular tool. Whether that tool is a TI calculator, an applet designed for introductory statistics, base R, some other statistical computing environment, or the tidyverse, picking a tool is a pedagogical choice.

No matter which approach or tool you use, you should strive to be consistent in the classroom whenever possible. Our choice of the tidyverse offers consistency, something we believe to be of the utmost importance to reduce cognitive load. Switching between tools can lead to confusion for students, and switching between syntaxes is no different. One complication of teaching consistently is that Google and StackOverflow can be less useful for students. Searching online for answers is an important skill to learn, but because of the variety of extant R syntaxes, searches can lead students to patterns that are unfamiliar. Students should be explicitly taught *how* to search online and sift through results, emphasizing the fact they are learning a specific syntax and only responses in that syntax will make sense to them. Tips for better searches include adding the names of specific tidyverse packages to your search query. Being transparent and clear about your use of a syntax will help students situate their knowledge—and misunderstandings—in the broader R ecosystem.

One obvious counter-proposition is that you should teach *all* (or multiple) syntaxes at once. We strongly disagree. Trying to teach two (or more!) syntaxes at once will slow the pace of the course, introduce unnecessary syntactic confusion, and increase the cognitive load for students. In keeping with the “let them eat cake first” approach (Çetinkaya-Rundel & Ellison 2020, Wang et al. 2017), students benefit from seeing the powerful things that they can do with R first. Deeper discussions about syntax and under-the-hood programming

concepts can occur in subsequent courses after students are already invested in using R. The approach we recommend is not tidyverse instead of base R, but tidyverse (mostly) before base R, as one can't do data analysis with the tidyverse without using base R.

At the same time, instructors may choose an appropriate level of flexibility in the code that students submit while being careful about what the instructors teach. We typically espouse a policy of being “disciplined in what we teach, liberal in what we accept” ([Postel 1980](#)). One might adopt a policy that any code that works is acceptable. Another might insist that only tidyverse patterns are acceptable. To continue the analogy with language, a writing instructor might reasonably accept papers written in any vernacular, or they might insist on a particular writing style for a particular assignment. Either way, one thing we have learned is that when we teach the tidyverse consistently, the presence of base R patterns (e.g., using square brackets to select columns instead of `dplyr::select()`) stands out. An assignment completed entirely with base R syntax could signal a student who is less engaged with the learning materials for the course.

3 How does the tidyverse work?

In this section we provide comparative examples of data wrangling and visualization tasks completed with the two most common R syntaxes: base R and the tidyverse.

Our approach is similar to those presented in [Kleinman & Horton \(2009\)](#) for comparison of SAS and R syntax; [McNamara \(2021a\)](#) for comparison of base R, tidyverse, and formula syntaxes in R; and [Dierker \(2021\)](#) for comparisons across multiple statistical software programs.

3.1 Data wrangling

Data wrangling is a major component of data acumen ([National Academies of Science, Engineering, and Medicine 2018](#)) that often takes up a majority of the data analysis cycle. The tidyverse includes a set of common idioms for data wrangling that work in a consistent manner via the `dplyr` and `tidyr` packages.

- `filter()`: select rows
- `select()`: select columns
- `arrange()`: order rows
- `mutate()`: add new or redefine existing columns
- `group_by()`: create partitions of rows
- `summarize()`: aggregate (or “roll up”) across rows
- `*_join()`: merge tables
- `pivot_*()`: reshape tables

While equivalents for each of these exist in base R and have been used for decades, the base routines were developed independently over time and do not share a common interface. In contrast, functions in the tidyverse generally take a data frame as a first argument, return a data frame, and use a consistent naming convention for arguments.

To highlight some of our arguments, we will use the `loans_full_schema` dataset from the [openintro](#) package for code examples ([Çetinkaya-Rundel et al. 2021](#)). The dataset represents thousands of loans made through the Lending Club platform, which allows individuals to lend to other individuals. The dataset contains information on the applicants and their financial history. In [Example 3.1](#), a small amount of data wrangling has been done to set the dataset up for analyses throughout the paper. A fully reproducible version of this paper, including the R code for reproducing all examples, can be found on GitHub at github.com/mine-cetinkaya-rundel/tidypaper.

```
loans <- openintro::loans_full_schema %>%
  mutate(
    homeownership = str_to_title(homeownership),
    bankruptcy = if_else(public_record.bankrupt >= 1, "Yes", "No")
  ) %>%
  filter(annual_income >= 10)
```

Example 3.1: The `loans` dataset from the `openintro` package.

[Example 3.1](#) provides a pipeline for: accessing a data frame within the `openintro` package; creating two new variables using the `mutate()` function, excluding incomes below \$10; and storing the result in a new tibble called `loans`.

Suppose we want to compute the average income of applicants based on their home ownership status. In the tidyverse, we could use the following pipeline shown in [Example 3.2](#).

```
loans %>%
  group_by(homeownership) %>%
  summarize(
    num_applicants = n(),
    avg_loan_amount = mean(loan_amount)
  ) %>%
  arrange(desc(avg_loan_amount))

## # A tibble: 3 x 3
##   homeownership num_applicants avg_loan_amount
##   <chr>          <int>          <dbl>
## 1 Mortgage      4778          18132.
## 2 Own           1350          15665.
## 3 Rent          3848          14396.
```

Example 3.2: Using the tidyverse to count applicants and compute the average loan amount from the loans data, sorted by average loan amount.

The tidyverse syntax expresses the sequential process of the computation. The pipe operator (`%>%`) brings the object from the left of the pipe into the function on the right of the pipe as the first argument; we pronounce the pipe function as “and then.” First, we start with the data frame that contains all the data. And then, we group the data according to the unique values of the `homeownership` variable. And then, for each unique value of `homeownership`, we compute both the number of rows (calling the result `num_applicants`) and the average loan amount in US Dollars (`avg_loan_amount`). And then, we arrange the rows of the resulting data frame in **descending** order according to the value of `avg_loan_amount`.

In base R, we might perform a computation similar to the one in [Example 3.3](#).

```
res1 <- aggregate(loan_amount ~ homeownership, data = loans, FUN = length)
names(res1)[2] <- "num_applicants"
res2 <- aggregate(loan_amount ~ homeownership, data = loans, FUN = mean)
names(res2)[2] <- "avg_loan_amount"
res <- merge(res1, res2)
res[order(res$avg_loan_amount, decreasing = TRUE), ]

##   homeownership num_applicants avg_loan_amount
```

## 1	Mortgage	4778	18132.45
## 2	Own	1350	15665.44
## 3	Rent	3848	14396.44

Example 3.3: Using base R to count applicants and compute the average loan amount from the loans data, sorted by average loan amount.

The base R code is harder to read, and less expressive of the logical process of the computation (i.e., more *cryptic*). It requires storing intermediate objects (`res1`, `res2`, and `res`) that might not otherwise be useful. It uses the `~`, `$`, and `[]` operators. It uses a magic number (2) to hard-code the second variable. It creates two data frames that need to be merged together. It passes the name of a function as an argument to a function. In Section 4.2, we argue that the base R syntax for the task at hand does not scale well for additional summary statistics. And in light of Section 4.4, the base R syntax is quite challenging to read aloud.

A different base R pattern, shown in Example 3.4, is more compact, and avoids some of the pitfalls listed above, but returns a vector. This named vector makes the result easy to read however it makes it cumbersome to include the second variable indicating the number of people.

```
sort(tapply(loans$loan_amount, loans$homeownership, mean), decreasing = TRUE)
```

```
## Mortgage      Own      Rent
## 18132.45 15665.44 14396.44
```

Example 3.4: Using `tapply()` within base R to count applicants and compute the average loan amount from the loans data, sorted by average loan amount.

Piping (using either the new base pipe or magrittr pipe) can simplify the code, as seen in Example 3.5 (equivalent results with the native pipe not shown).

```
tapply(loans$loan_amount, loans$homeownership, mean) %>%
  sort(decreasing = TRUE)
```

Example 3.5: Using the magrittr pipe and `tapply()` to count applicants and compute the average loan amount from the loans data, sorted by average loan amount.

3.2 Data visualization

The process of teaching and learning data visualization is challenging (Nolan & Perrett 2016). Students are asked to engage with multiple interrelated steps: (1) the conceptualization and design of the visualization, (2) the translation of the design into the specific syntax of the plotting tool or library, and (3) the cleaning, conversion, and transformation of the data that will be represented in the visualization. It is for the latter two tasks we believe the tidyverse substantially improves on base R’s functionality, particularly when it comes to new learners.

Base R plotting is very powerful and flexible, but that flexibility leads to confusion and idiosyncratic behavior. Take for example, the creation of a basic scatterplot of the variables `x` and `y` stored in a data frame `d`, all of the following would produce the same plot: `plot(d)`, `plot(y~x, d)`, `plot(dx, dy)`, `with(d, plot(x,y))`, and so on. The flexibility, built on the S3 object system, is useful but can be overwhelming for new learners. The difficulty is particularly apparent when students get stuck and search for terms like “scatterplot in R” and come across solutions which use an approach that differs substantially from their instructor’s preferred method (e.g., using the formula method when they have only been shown `$` column access).

Matters get worse as needs expand beyond the most basic plotting primitives. Students can easily *use* custom plotting methods (e.g., `density(x)`), functions for more specific plotting primitives (e.g., `boxplot(y~x)`), and methods combining both (e.g., `hist(x)`) without building a higher-level understanding of how to create a complex plot. That is, while all the plotting tools are usable, it is hard for students to develop a mental model of their commonalities without first having a deeper understanding of R as a programming language, specifically around generic functions, basic data structures, and classes.

As mentioned previously, most of base R’s plotting functionality is built around S3 specializations of `plot()` and similar high-level plotting functions. In some cases, the plot types are relatively easy to identify by name, e.g., `hist()`, `barplot()`, `boxplot()`, while others are less obvious, e.g., `abline()`, `image()`, or `par()`. The base R plotting functions are part of the **graphics** package, which is loaded automatically by all R sessions.

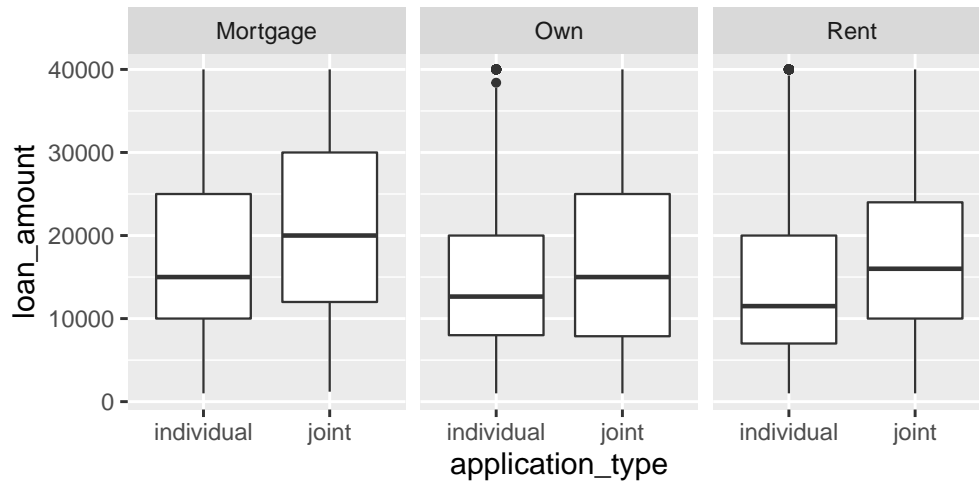
To visualize using tidyverse principles, we use one of the core tidyverse packages: **ggplot2**.

In `ggplot2`, different plot types are implemented using geometry functions (prefixed with `geom_`) which map variables in the data to various aesthetic properties (e.g., horizontal and vertical position, size, color, etc.) of the plots. One of the immediate advantages of using `geom_*()` functions for plotting is the reduction in the search space of possible plotting functions. The prefix works effectively to organize the search space to the possible functions and their associated documentation. The narrowing is true whether within an IDE (using tab-completion), in the documentation (looking just in the `g` section), or searching online. In contrast, standard base graphics begin with a multitude of different characters, so there is no obvious way to simply see all the possibilities.

Additionally, the geometries implemented in `ggplot2` use the same core function arguments and share common aesthetics, which makes it easier to pick up and explore new geometries as well as quickly swap between related visualization methods. For example, changing between boxplots and violin plots (an augmented form of boxplots) only requires changing `geom_boxplot()` to `geom_violin()`—the arguments to the two functions remain the same. As `ggplot2` is built around the principles of the grammar of graphics ([Wilkinson 2012](#)), its syntax is designed to reflect the process of building a visualization through the composition of layers using the `+` operator. In contrast, base R graphics are generally built up using multiple function calls (e.g., `plot(..., add=TRUE)` and `abline()`).

[Example 3.6](#) presents an example of side-by-side boxplots of `loan_amount` as a function of `application_type` and `homeownership`.

```
loans %>%
  ggplot(aes(y = loan_amount, x = application_type)) +
  geom_boxplot() +
  facet_wrap(~ homeownership)
```

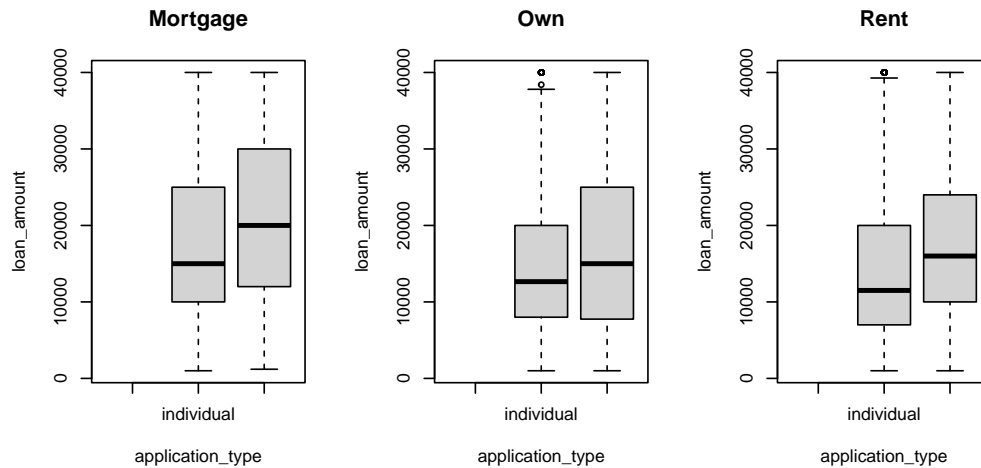


Example 3.6: Using ggplot2 to create boxplots of `loan_amount` broken down by both `application_type` and `homeownership`.

The figure is constructed in three calls: one to map the variables to aesthetics, another to draw the boxplot, and the last to facet by levels of `homeownership`. A similar plot can be constructed using base R graphics (see [Example 3.7](#)), however the process of creating facets is more burdensome as it requires using creation of separate plots with a `for()` loop and subsetting to iterate over the unique levels of `homeownership`.

```
levels = sort(unique(loans$homeownership))
n = length(levels)

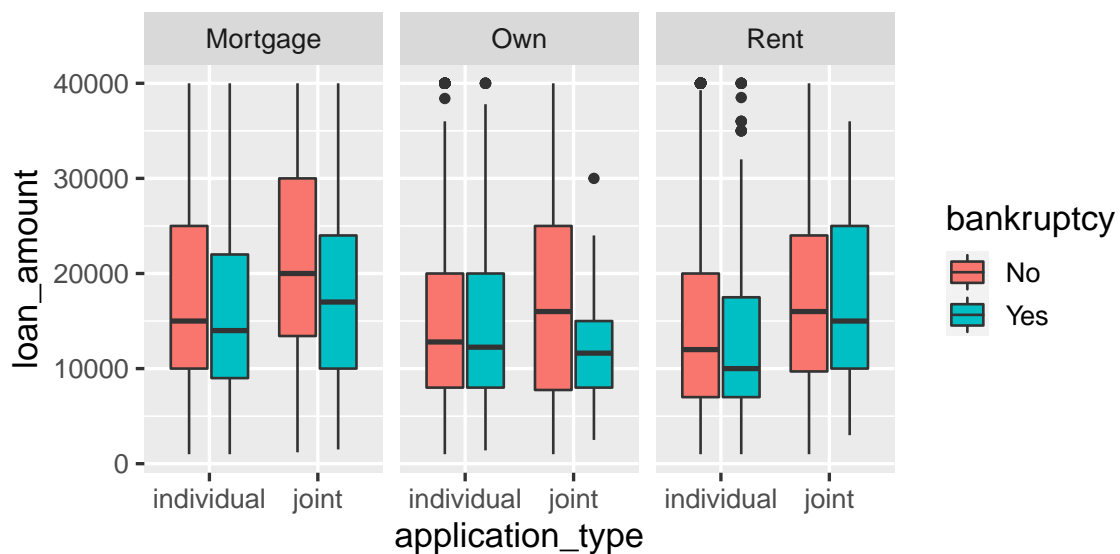
par(mfrow = c(1,n))
for(i in seq_len(n)) {
  boxplot(
    loan_amount ~ application_type,
    data = loans[loans$homeownership == levels[i],],
    main = levels[i]
  )
}
```

Example 3.7: Using base R to create boxplots of `loan_amount` broken down by both `application_type` and `homeownership`.

As seen in [Example 3.8](#), in `ggplot2` it is straightforward to extend the original boxplot to more data dimensions by mapping another variable to an additional aesthetic. For example, adding `fill = bankruptcy` to the `aes()` call will create a plot that now displays four dimensions of the original data.

```
loans %>%
  ggplot(aes(y = loan_amount, x = application_type, fill = bankruptcy)) +
  geom_boxplot() +
  facet_wrap(~ homeownership)
```

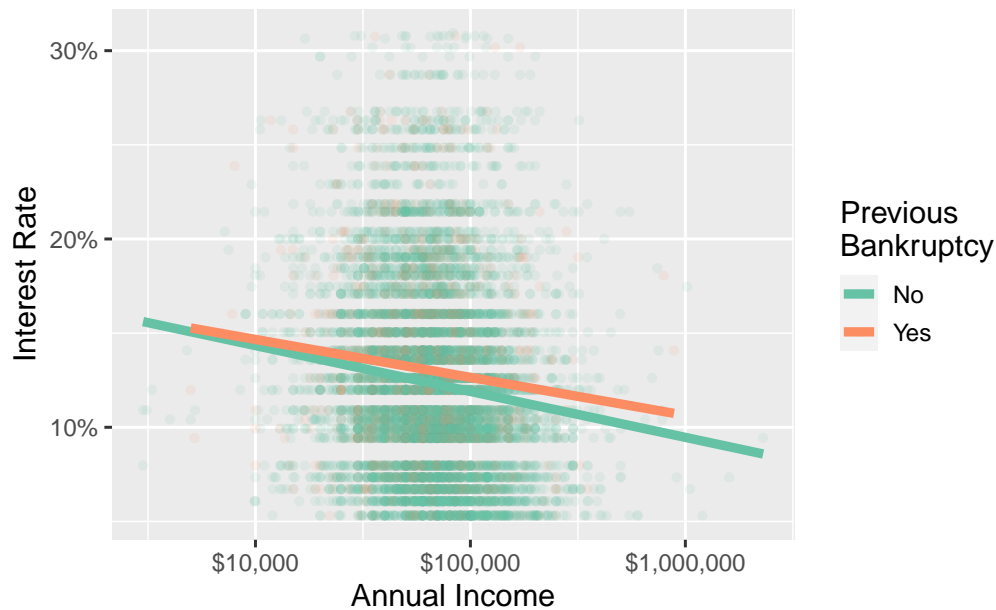


Example 3.8: Using ggplot2 to create boxplots of `loan_amount` broken down by both `application_type` and `homeownership`, filled by `bankruptcy`.

Splitting the data to create multiple boxplots based on different variables is possible with base R, however the implementation is more complex and verbose, and it is left as an exercise to the reader.

Next we generate scatterplots of interest rate as a function of income, where the points and linear fits are colored by the bankruptcy status of the loan recipient.

```
loans %>%
  ggplot(aes(y = interest_rate, x = annual_income, color = bankruptcy)) +
  geom_point(alpha = 0.1) +
  geom_smooth(method = "lm", size = 2, se = FALSE) +
  scale_x_log10(labels = scales::label_dollar()) +
  scale_y_continuous(labels = scales::label_percent(scale = 1)) +
  scale_color_brewer(type = "qual", palette = "Set2") +
  labs(x = "Annual Income", y = "Interest Rate",
       color = "Previous\nBankruptcy")
```



Example 3.9: Using ggplot2 to create a scatterplot of `interest_rate` versus annual income, colored by `bankruptcy`.

We note that [Example 3.9](#) requires seven function calls, but each call builds on a common framework and interface.

Base R code is used in [Example 3.10](#) to create a similar scatterplot, but the syntax requires careful study of the documentation for `par()` and multiple separate calls of `lm()`.

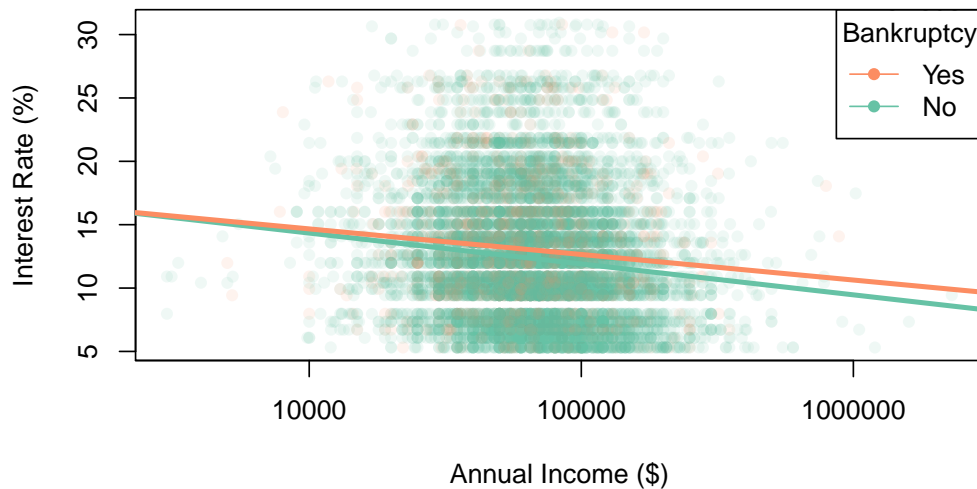
```
# From the ColorBrewer Set2 palette
cols = c(No = "#66c2a5", Yes = "#fc8d62")

plot(
  loans$annual_income,
  loans$interest_rate,
  pch = 16,
  col = adjustcolor(cols[loans$bankruptcy], alpha.f = 0.1),
  log = "x",
  xlab = "Annual Income ($)",
  ylab = "Interest Rate (%)",
  xaxp = c(1000, 10000000, 1)
)

lm_b_no = lm(
  interest_rate ~ log10(annual_income),
  data = loans[loans$bankruptcy == "No",]
)
lm_b_yes = lm(
  interest_rate ~ log10(annual_income),
  data = loans[loans$bankruptcy == "Yes",]
)

abline(lm_b_no, col = cols["No"], lwd = 3)
abline(lm_b_yes, col = cols["Yes"], lwd = 3)

legend(
  "topright",
  legend = c("Yes", "No"),
  title = "Previous\nBankruptcy",
  col = cols[c("Yes", "No")],
  pch = 16, lwd = 1
)
```



Example 3.10: Using base R to create a scatterplot of `interest_rate` versus annual income, colored by `bankruptcy`.

We note several differences between the two scatterplot implementations. The `ggplot2` approach uses function arguments vs. base graphics `par` parameters. Some of the parameters (e.g., `xaxp`, `log`) are idiosyncratic and difficult to find in the documentation. Specifying colors and alpha transparency levels are quite different in the two implementations. In base graphics, legends require a separate function with additional manual bookkeeping, which increases the potential for human error. Explicit iteration is required to create multivariate plots and the user is required to specify the display structure. Care is needed in constructing axis values.

We note that the `+` operator in the `ggplot2` syntax differs from the pipe (`%>%`) operator for historical reasons (Wickham 2015). We have found, however, that the `+` operator is straightforward for students to learn. The `+` operator is consistent with the layering aspect of creating a plot. Additionally, the error message when `%>%` is used in place of `+` explicitly asks: “Did you use `%\>%` instead of `+`?”.

3.3 Beyond wrangling, statistical summaries, and visualizations

While data wrangling and data visualization are high on the list of important tasks for working with data, there are many additional tasks and tools needed for a complete data analysis. We argue that setting your students up to understand the tidyverse approach

will not only help them work fluently with data at the exploratory data analysis stage, but will also flow seamlessly into inference, modeling, working with databases (e.g., SQL), and other data-focused operations.

Statistical inference can be taught using the **infer** package (Bray et al. 2021), which uses a consistent syntax for one- and two-sample inferential techniques (both computational methods like randomization tests and bootstrapping as well as mathematical models like t- and z-tests). With the **infer** syntax, students learn a single set of functions that walk through the inferential process and focus attention on each step of the process. Indeed, the functions themselves are named to reinforce the conceptual understanding of the process. For example, a student will **specify()** the variables *and then* **hypothesize()** about the conditions *and then* **generate()** a sampling distribution *and then* **calculate()** the statistic of interest. The output of these functions can be wrangled (e.g., to determine a p-value) or visualized (e.g., to inspect the null sampling distribution).

When doing **modeling and machine learning**, many data analysts rely on functions like **lm()** and **glm()** as important tools in the analysis process. To avoid reinventing the wheel, packages like **broom** (Robinson et al. 2021) allow **tidy modeling with base R functions** to extend the convention of data frame as input and data frame as output. Using **lm()** as an example, note that there are model output pieces that: 1) describe each of the variables (e.g., the coefficients); 2) describe each of the observational units (e.g., the residuals); and 3) describe the entire model (e.g., R^2). Since the **lm()** output is a list with a specialized print method, it is not immediately clear how to access certain components of it (e.g., the value of the slope or the intercept) programatically. The broom paradigm—which can be applied to **lm()** and **glm()** as well as many other model objects—uses **tidy()** for output which describes variables, **augment()** to describe observational units, and **glance()** to describe the entire model.

In addition to modeling done using using base R functions, a user interested in machine learning methods or approaches (e.g., cross validation) is likely to benefit from **tidy modeling functionality** available in the **tidymodels** package (Kuhn & Wickham 2021). The core idea of tidymodels is to simplify the practice of modeling by pre-processing *and then* training *and then* validating models. The pre-processing step can scale variables or filter

for highly correlated covariates. The training step can use anything from a linear model to a support vector machine to a neural network. The validating step incorporates cross validation or prediction on a test dataset to evaluate the user's metric of choice.

Database technologies are supported using the **dplyr** (Wickham, François, Henry & Müller 2021a) and **dbplyr** (Wickham, Girlich & Ruiz 2021) packages, which facilitate access to SQL databases using the same general syntax and idioms learned with the tidyverse. As an example, we can access and summarize data from a publicly accessible repository of audiological measurements (Voss 2019) using the same form as our earlier wrangling (see Section 3.1).

```
db <- DBI::dbConnect(
  RMySQL::MySQL(),
  dbname = "wai",
  host = "scidb.smith.edu",
  username = "waiuser",
  password = "smith_waiDB" # publicly accessible database
)

db %>%
  tbl("Subjects") %>%
  group_by(AgeCategoryFirstMeasurement) %>%
  summarize(num_people = n())
```



```
## # Source:   lazy query [?? x 2]
## # Database: mysql 5.5.58-0ubuntu0.14.04.1-log [@scidb.smith.edu:/wai]
##   AgeCategoryFirstMeasurement num_people
##   <chr>                        <dbl>
## 1 Adult                        713
## 2 Child                       116
## 3 Infant                      810
## 4 NICU                       41
```

Example 3.11: Applying tidyverse wrangling to data which has been queried from a SQL database.

Note that the output of the wrangled SQL data is a **tibble** similar to that of Section 3.1, but with additional information about the MySQL database server.

4 Why teach with the tidyverse?

In this section, we expand on the core benefits of the tidyverse outlined in Section 3 by highlighting a few supplementary points in its favor.

4.1 Consistency does not mean exclusivity

Consistent syntax and interface is a hallmark of the tidyverse’s design principles (as outlined in Section 2). As instructors, we strive for consistency in how we use the tidyverse in our teaching. To achieve this, we avoid mixing-and-matching tidyverse patterns with base R patterns *within pipelines*. However, because the input and output of tidyverse functions are “normal” R objects (typically, a `data.frame`), an instructor can be consistent without coding exclusively in the tidyverse. An example of inconsistent behavior that we avoid would be using base graphics for boxplots and `ggplot2` for scatterplots.

For example, in Section 3.3 we illustrate how the `infer` package can be used in an introductory statistics course to extend the tidyverse framework to include statistical inference. However, one can certainly teach introductory statistics by combining the core tidyverse functionality (i.e., `dplyr` and `ggplot2`) with base R implementations of inferential functions (e.g., `t.test()`, `chisq.test()`, etc.). This will necessitate some inconsistency, since some base R inferential functions accept vectors as inputs rather than data frames. However, as with explaining the difference between the `+` and `%>%`, instructors can explain that data wrangling and visualization always use data frames, but that inferential functions sometimes use vectors. We believe that inference is smoother with `infer`, but a decision not to adopt `infer` should not preclude an instructor from adopting the tidyverse for other tasks. A similar argument holds for statistical modeling, where `lm()` can be used alongside tidyverse code. We contend that `broom` and/or `tidymodels` reduce friction when analyzing data, but several of the authors still teach `lm()` in their own courses.

4.2 Tidyverse code scales

In Section 3.1, we illustrated how the `group_by()` and `summarize()` verbs make it easy to “roll up” a data frame by groups. Next, we consider extending that analysis in two different ways: by adding one or more summary computations; and by adding one or more additional grouping variables. In both cases, we argue that tidyverse code adapts more easily than base R while reducing additional cognitive overhead.

First, in Example 4.1, we note that adding an aggregate computation involves only a comma and the expression involving the relevant summary function in the call to `summarize()`. Here, we compute two quantities: a count (`num_applicants`) and a mean (`avg_loan_amount`).

```
loans %>%
  group_by(homeownership) %>%
  summarize(
    num_applicants = n(),
    avg_loan_amount = mean(loan_amount)
  ) %>%
  arrange(desc(avg_loan_amount))
```



```
## # A tibble: 3 x 3
##   homeownership num_applicants avg_loan_amount
##   <chr>          <int>          <dbl>
## 1 Mortgage      4778            18132.
## 2 Own           1350            15665.
## 3 Rent          3848            14396.
```

Example 4.1: Using the tidyverse to count applicants and compute the average loan amount from the loans data, sorted by average loan amount.

If we wanted to compute p quantities, for any integer $p > 1$, it is straightforward to add p arguments to `summarize()`. Thus, this type of operation is *scalable*, because the number of lines of code is proportional to the number of quantities computed.

Additionally, if the variables to be summarized can be selected based on their types or names, we can use `dplyr::across()` to summarize many variables with one line of code. In example 4.2, we show how to calculate the mean for across variables in the `loans` dataset that contain the character string "paid" for each `homeownership` group.


```
loans %>%
  group_by(homeownership) %>%
  summarise(across(contains("paid"), mean))

## # A tibble: 3 x 5
##   homeownership paid_total paid_principal paid_interest paid_late_fees
##   <chr>          <dbl>         <dbl>         <dbl>         <dbl>
## 1 Mortgage      2734.         2089.         646.         0.0810
## 2 Own           2512.         1950.         563.         0.111
## 3 Rent          2191.         1635.         555.         0.171
```

Example 4.2: Using the tidyverse to calculate mean amounts across variables that contain the character string 'paid' for each homeownership group.

Achieving scalability in base R is possible, but the approaches involve additional programming concepts. A conceptually straightforward base R approach to scaling up is to call `tapply()` p times and combine the resulting vectors using `cbind()` (see [Example 4.3](#)). While the approach also involves $O(p)$ lines of code, you would need to type the names of the two variables (`loan_amount` and `homeownership`) p times.

```
with(
  loans,
  cbind(
    num_applicants = tapply(loan_amount, homeownership, length),
    avg_loan_amount = tapply(loan_amount, homeownership, mean)
  )
)
```

```
##           num_applicants avg_loan_amount
## Mortgage           4778       18132.45
## Own                1350       15665.44
## Rent               3848       14396.44
```

Example 4.3: Using `tapply()` in base R to calculate multiple summary statistics.

A base R construction that is more scalable involves writing a custom summary function and iteratively combining the results using `do.call()` and `rbind()` (see [Example 4.4](#)). The approach requires two programming concepts (writing a user-defined function and iterating a function) that are likely beyond the scope of introductory statistics or data science.

```
my_summary <- function(x) {
  data.frame(
    num_applicants = length(x),
    avg_loan_amount = mean(x)
  )
}
do.call(rbind, with(loans, tapply(loan_amount, homeownership, my_summary)))
```

```
##           num_applicants avg_loan_amount
## Mortgage           4778           18132.45
## Own                1350           15665.44
## Rent               3848           14396.44
```

Example 4.4: Using a custom function in base R to calculate multiple summary statistics.

Second, we note that adding a second grouping variable in the tidyverse involves only adding another argument to `group_by()`. Moreover, adding k grouping variables involves adding k items to `group_by()` (see [Example 4.5](#)). No additional programming knowledge is necessary.

```
loans %>%
  group_by(homeownership, verified_income) %>%
  summarize(
    num_applicants = n(),
    avg_loan_amount = mean(loan_amount)
  )

## # A tibble: 9 x 4
## # Groups:   homeownership [3]
##   homeownership verified_income num_applicants avg_loan_amount
##   <chr>         <fct>             <int>         <dbl>
## 1 Mortgage     Not Verified         1580         14739.
## 2 Mortgage     Source Verified      1963         19085.
## 3 Mortgage     Verified             1235         20960.
## 4 Own          Not Verified          495         13161.
## 5 Own          Source Verified       547         16537.
## 6 Own          Verified              308         18142.
## 7 Rent         Not Verified         1498         12287.
## 8 Rent         Source Verified      1605         14964.
## 9 Rent         Verified              745         17413.
```

Example 4.5: Using the tidyverse to add a layer of grouping before calculating summary statistics.

In base R, adding additional grouping variables can be achieved by wrapping the set of grouping variables names in `list()`. Here again, while the approach is scalable, it introduces a data structure (`list`) that is not typically necessary for introductory classes (see [Example 4.6](#)). The output also omits the factor levels that correspond to each row, which introduces potential confusion.

```
do.call(  
  rbind,  
  with(  
    loans,  
    tapply(loan_amount, list(homeownership, verified_income), my_summary)  
  )  
)
```

```
##   num_applicants avg_loan_amount  
## 1             1580      14739.15  
## 2              495      13161.21  
## 3             1498      12287.43  
## 4             1963      19084.74  
## 5              547      16536.97  
## 6             1605      14964.45  
## 7             1235      20960.04  
## 8              308      18142.29  
## 9              745      17413.39
```

Example 4.6: Using base R to add a layer of grouping before calculating summary statistics.

Thus, in both cases, while scalable programming in base R is possible, it brings with it additional extraneous programming concepts that may distract—rather than support—students from learning of statistics and data science. Some of the confusion is because base R functions operate on vectors, matrices, and data frames idiosyncratically. In contrast, because tidyverse functions focus on data frames, users of the tidyverse may be able to fall into the “pit of success,” wherein scaling one’s analysis is natural and requires no extraneous cognition or bookkeeping.

4.3 User-centered design eases learning

The tidyverse has been developed with a user-centered design process ([Kling 1977](#), [Norman & Draper 1986](#)), that can also be considered learner-centered ([Soloway et al. 1994](#)). While many R packages are designed once and then updated incrementally with bug fixes, a number of the packages within the tidyverse have undergone large scale API changes to improve usability.

A prime example of the user-centered approach to development in the tidyverse is the evolution of the functions for reshaping data. Reshaping data is a key data wrangling skill, as data does not always come in a format conducive to analysis. For example, consider data containing counts of fruits and vegetables sold at a produce stand where rows are years and months (and totals!) and columns are fruits and vegetables (and totals!), as in Figure 2. Note that in the sample dataset, variables (year and month) are in the rows, which keeps the data from being tidy. Additionally, the row and column totals make the task of visualizing and summarizing difficult.

	A	B	C	D	E	F	G	H	I	J	K
1	Year	Month	Apples	Oranges	Bananas	Fruits	Carrots	Cucumbers	Onions	Veggies	Total
2	2018	January	4	2	5	11	1	1	5	7	18
3		February	5	3	3	11	3	3	8	14	25
4		March	2	4	2	8	2	1	2	5	13
5		...									
6		December	8	9	1	18	6	1	2	9	27
7		Total	65	58	48	171	41	34	46	121	292
8	2019	January	7	5	6	18	7	3	2	12	30
9		February	3	7	3	13	1	1	8	10	23
10		March	2	3	9	14	5	5	5	15	29
11		...									
12		December	9	5	2	16	4	3	4	11	27
13		Total	62	58	67	187	60	50	65	175	362
14	2020	January	1	6	3	10	4	2	7	13	23
15		February	2	7	1	10	1	3	7	11	21
16		March	3	8	4	15	6	7	4	17	32
17		...									
18		December	6	4	3	13	5	8	3	16	29
19		Total	46	72	36	154	41	60	62	163	317

Figure 2: Non-tidy data which can be wrangled using `pivot_longer()`.

The **reshape** package ([Wickham 2018](#)) was introduced in 2005, and offered the functions `melt()` and `cast()` to perform data transformations. The package author later realized that the functions were not consistent with other parts of the tidyverse (such as **plyr** ([Wickham 2020a](#)), a predecessor to `dplyr`), and that the `cast()` function needed to be

split into two: `dcast()` for data frames and `acast()` for arrays and matrices. These functions were introduced in the **reshape2** (Wickham 2020b) package in 2010. However, that was not the end of the improvements to the functions. Many users reported difficulty remembering when to use `melt()` and when to use `cast()`. While they were somewhat ‘cute’ names, they didn’t hold inherent meaning in the context of data analysis. The next iteration in 2014 introduced the functions `gather()` and `spread()` in the **tidyr** (Wickham 2021c) package. The verbs `gather()` and `spread()` were somewhat easier for users to remember, and included the arguments `key` and `value`, which were familiar to database programmers. However, the improvement left even experienced programmers consulting the documentation too frequently.

The most recent iteration of these functions is `pivot_wider()` and `pivot_longer()`, introduced in version 1.0.0 of **tidyr**. The names for the functions were developed as part of a design process that included user surveys administered by Hadley Wickham. The resulting functions are much more expressive than the original `melt()` and `cast()` (Wickham 2019). Instead of `key` and `value`, the arguments use `names_to` and `values_to` in `pivot_longer()` and `names_from` and `values_from` in `pivot_wider()`. Anecdotal evidence (including but limited to the experience of the authors) suggests that the newest set of functions empower users to write code more fluently without looking at the documentation.

Beyond changes to function names, the default values of arguments present in the tidyverse have been thoughtfully designed with the intention of making life easier for analysts and preventing mistakes. These defaults can be more flexibly updated than those in base R, which moves extremely slowly.

While all graphics libraries in R provide for customization, the initial plots generated by **ggplot2** look much more “finished” than graphics from base R or **lattice** (Sarkar 2021) graphics (Myint et al. 2020), which helps students feel pride in their work from the beginning. The graphics look more polished because the defaults have been chosen based on research. For example, the default **ggplot2** color scheme has been updated to use **viridis** (Garnier 2021), a set of color scales based on perceptual research. The default grey background is used so that the plot is of similar visual weight as surrounding text (Wickham, Navarro & Pedersen 2021). When colors or facets are applied, **ggplot2** automatically pro-

vides a legend.

The tidyverse provides warnings to analysts to help them avoid making mistakes. For example, when you create a plot that involves a variable with missing values, the package will warn you.

```
ggplot(loans) +  
  geom_boxplot(aes(y = emp_length, x = application_type))
```

```
## Warning: Removed 794 rows containing non-finite values (stat_boxplot).
```

Example 4.7: ggplot2 displays a warning when plotting a variable with missing values.

Defaults in other areas of the tidyverse are also designed for success.

The `drop_na()` function forces users to think more intentionally about the way they wish to deal with missing values. In base R, addressing NA values is either done with a destructive `na.omit()` call or as an argument to each function, as shown in [Example 4.8](#).

```
mean(loans$annual_income_joint)
```

```
## [1] NA
```

```
mean(loans$annual_income_joint, na.rm = TRUE)
```

```
## [1] 128085.2
```

Example 4.8: When working with missing data in base R, each function needs an additional argument.

In the tidyverse, dropping missing values becomes part of the pipeline, as shown in [Example 4.9](#).

```
loans %>%  
  drop_na(annual_income_joint) %>%  
  summarize(mean(annual_income_joint))
```

```
## # A tibble: 1 x 1  
##   'mean(annual_income_joint)'  
##                               <dbl>  
## 1                               128085.
```

Example 4.9: Working with missing data in the tidyverse becomes part of the pipeline.

In base R, it is easy to break the relationship between factor levels and their labels, but the **forcats** package (Wickham 2021a) provides many custom functions (prefixed **fct_**) for wrangling categorical data while maintaining sound and reproducible analysis (McNamara & Horton 2018). Defaults from the tidyverse—such as not reading in strings as factor variables—have gained so much popularity they have been integrated into base R.

We argue that using functionality created to be deliberately user-centered is vital to bringing new software tools to the classroom. The more intuitive and memorable the functions, the lighter the cognitive load for the novice learners we hope to retain. The more functions that have been designed with thoughtful defaults, the easier it is for students to find success.

4.4 Readability is important

Programming instruction is improved by reading code out loud (Swidan & Hermans 2019), so it stands to reason that statistics programming instruction could be similarly improved. People learning to read a human language (e.g., English) learn by reading aloud and then moving to “subvocalizing”: saying words in one’s head. The reading process allows learners to connect the sound of the word to the concept it signifies.

Learning programming by reading aloud provides the same cognitive benefits. Students will likely try to subvocalize code as they read silently, but without examples of phonology (the specific way parts of the language should be vocalized), they have to make up their own pronunciations, which may be inconsistent throughout their reading, adding additional cognitive load (Hermans et al. 2018). ? connects ideas of vocalization to programming using R code, demonstrating specific phonology for reading R aloud. As with human language, there can be regional variations in how particular symbols are voiced, but an instructor should strive to be as consistent as possible with their choices of vocalizations. Additionally, consistency with vocalization can help while pair programming or debugging from afar (e.g., over Zoom), because one person can dictate code to another using shared

language.

The focus on function names as “verbs” in the tidyverse lends itself well to vocalization. Unfortunately, vocalization does not transfer well to the page, so this is difficult to convey. However, please consider the sample code in [Example 4.10](#) and [Example 4.11](#).

```
loans %>%  
  mutate(bankruptcy = if_else(public_record_bankrupt >= 1, "Yes", "No")) %>%  
  group_by(bankruptcy) %>%  
  summarize(avg_loan_amount = mean(loan_amount))
```

Example 4.10: A tidyverse wrangling of the bankruptcy variable in the loans data.

```
loans$bankruptcy <- ifelse(loans$public_record_bankrupt >= 1, "Yes", "No")  
tapply(loans$loan_amount, loans$bankruptcy, mean)
```

Example 4.11: A base R wrangling of the bankruptcy variable in the loans data.

Try reading these code snippets out loud to yourself. Which elements do you vocalize? Which do you skip? When we read the tidyverse code, we pronounce the `%>%` operator as “and then.” “Group by bankruptcy and then summarize.” When we read the base code, it is more repetitive because we need to repeatedly say things like “loans dollar sign bankruptcy.” Typically, we do not vocalize every character on the screen. Most commonly, we do not read out line breaks, underscores, or many parentheses (particularly closing parentheses). However, when doing a more verbose vocalization (perhaps used when dictating to a newer student) would likely include the additional punctuation. If you would like to hear one of us read the code snippets aloud, please see [McNamara \(2021b\)](#). Hopefully, the exercise of reading code aloud helps illustrate that the tidyverse is more designed for speakability. The tidyverse verbs sound more like English sentences (e.g., compare `tapply()` to `group_by()`).

We should note that readable doesn’t necessarily mean discoverable—one wouldn’t necessarily think of the word “summarize” to calculate the mean of a column, but once you learn the framework of “summarize” it’s likely to stick with you because the verb does what it says.

Additionally, even when the function names may seem self-explanatory, it is still important not to assume learners can tell what the function does without explaining the meaning of the word. Tidyverse verbs are based on English words, so they privilege people whose first language is English. And, even for native English speakers, some of the words are not immediately transferable into the data analytic context. For example, if you’ve never come across `tidyr::hoist()`, would you be able to guess what “hoist” means in this context? The advantage of well-chosen function names is that once you explain what a function does, you likely don’t need to explain it again.

4.5 Connecting to resources

It is worth pointing out the tidyverse’s popularity across a wide variety of disciplines and application areas, including in industry. Popularity not only indicates that professional users find it to be a worthwhile tool, but also can actually increase students’ ability to engage with R. [DataScienceMeta.com](https://www.data-science-meta.com/) tracks CRAN downloads of R packages. Six of the top 10 packages (as of July 2, 2021) downloaded from CRAN are in the tidyverse, and the tidyverse package itself is the 20th most downloaded from CRAN. The popularity is an indication that when students are searching for help (e.g., from Google or StackOverflow), they are likely to come upon a tidyverse solution.

Additionally, although older textbooks predominately use base R to introduce statistical computing, more and more texts are using tidyverse syntax. A popular example is *R for Data Science* ([Wickham & Grolemund 2016](#)), a textbook specifically focused on using the tidyverse to do data science. [Ismay & Kim \(2019\)](#), [Baumer et al. \(2021\)](#), [Roback & Legler \(2021\)](#), [Hyndman & Athanasopoulos \(2021\)](#), and the R materials associated with [Çetinkaya-Rundel & Hardin \(2021\)](#) all use primarily tidyverse code. Providing our students contemporary tools like the tidyverse will prepare them to engage fully with the larger community of statisticians and data scientists who have adopted the tidyverse into their work.

4.6 Get SQL for free

The majority of Section 4 has argued that beyond the form and function of the tidyverse, the reduction in cognitive learning load will make the tidyverse easier than base R for a new learner. Here, we provide one additional reason for bringing the tidyverse into an undergraduate classroom full of students who will be heading toward a workforce in a data-centered world.

The careful construction of the tidyverse, and in particular the dplyr package, can have additional benefits to learners in the context of database querying. Since the development of relational database modeling begun by Codd (1970), Structured Query Language (SQL) has been the dominant paradigm for interacting with relational databases. SQL databases are widely deployed through technologies like SQLite, MySQL, PostgreSQL, Microsoft SQL Server, and Oracle. Moreover, many newer technologies that seek to supersede SQL are predicated on their users' knowledge of SQL. This includes cloud-based services (e.g., Google Big Query), as well as non-tabular database systems (e.g., “noSQL”). Thus, for undergraduates, learning how to write SQL queries is a useful step towards a career in data science (Horton et al. 2015).

The dplyr package was written with SQL in mind. As described in Section 3.1, the main verbs, along with the various `join_*()` functions, comprise a set of functions that can serve as the building blocks for SQL queries. The **dbplyr** package provides functionality that will translate dplyr pipelines into SQL queries. This enables R users to query SQL databases without having to write SQL code. However, since knowing how to write an SQL query is a useful skill for students to develop, learning data wrangling through dplyr has the beneficial side effect of giving students a conceptual understanding of SQL with minimal additional cognitive overhead. That is, the work that students have put in to learn data wrangling in dplyr can be easily ported to SQL fluency. Instructors can pair six weeks of dplyr instruction with two weeks of SQL instruction and have reasonable confidence that students will be proficient in *both* technologies.

Students can engage in a comparative literature exercise in which they map each function in the dplyr pipeline to a different clause in the SQL statement. The comparison of SQL and dplyr syntax can reinforce the message that the underlying concepts are the same here—

it is only the programming syntax that differs between R and SQL. To fully drive the 1-to-1 equivalence home, **dbplyr** contains the function `show_query()` that will explicitly translate a dplyr pipeline to a SQL query. [Example 4.12](#) shows the SQL translation of the audiological measurement query shown in [Example 3.11](#).

```
db %>%
  tbl("Subjects") %>%
  group_by(AgeCategoryFirstMeasurement) %>%
  summarize(num_people = n()) %>%
  show_query()

## <SQL>
## SELECT 'AgeCategoryFirstMeasurement', COUNT(*) AS 'num_people'
## FROM 'Subjects'
## GROUP BY 'AgeCategoryFirstMeasurement'
```

Example 4.12: A SQL translation of the dplyr pipeline shown in [Example 3.11](#).

Thus, by learning dplyr students get SQL (almost) for free.

5 Discussion

In this section, we reflect on how the tidyverse fits into a larger curriculum, discuss the importance and challenge of staying current, rebut some common criticisms of our approach, and conclude with final thoughts.

5.1 Building a curriculum

We have made the argument that students' first introduction to R can (and should) be with the tidyverse. However this doesn't mean learning materials should be structured around tidyverse packages, but rather around data science concepts. A good starting point is data visualization (and hence the `ggplot2` package) followed by single-table verbs from the dplyr package ([Çetinkaya-Rundel & Ellison 2020](#)). Then, the learning goals for a given course should embrace bits and pieces of tidyverse packages that are designed to help with

relevant data analysis tasks related to those goals. For example, multivariate thinking can be introduced alongside `ggplot2` functionality that maps variables to additional aesthetics like color, size, shape, and facets. Relational data can be introduced with two-table verbs from `dplyr` (i.e., `*_join()` functions).

We also recommend using `library(tidyverse)` to load all eight core tidyverse packages and not allocating much time or energy to distinguishing which function lives in which of these eight packages, at least in the introductory course. It's important to let students know where they can find information in package documentation, but beyond that, making distinctions within the tidyverse core packages can add to unnecessary cognitive load.

Even just the core eight packages in the tidyverse offer a vast array of functions for doing data analysis tasks. The breadth of that functionality goes well beyond the topics that can reasonably be covered in an introductory statistics or data science course within the span of an academic term. For example, in introductory courses where the audience is new to working with data, statistics, and programming, we recommend delaying introduction of the **purrr** package ([Henry & Wickham 2020](#)) and the functional programming paradigm. One of the strengths of the **purrr** package is working with list-columns, which are relevant when applying functions to many columns or when working with hierarchical data. If working with advanced data structures is a topic included in the learning goals of an introductory course, we recommend solving the problems using functionality recently added to packages like `dplyr` (e.g., the `across()` function) and `tidyr` (e.g., `unnest_*()` functions) in order to avoid introducing functional programming as an additional topic in the curriculum.

This is not to say incorporating the tidyverse into a curriculum can be done without any adjustments to the learning goals. For example, if teaching R without the tidyverse, one might avoid the discussion of the pipe operator or the notion of a **tibble** (tidyverse's implementation of a data frame) entirely. On the other hand, adding some new learning goals to the course to support the teaching of the tidyverse can provide a principled framework that allows for tackling modern data problems while using a consistent syntax.

5.2 Keeping up with the tidyverse

Like the majority of (particularly open-source) software, the tidyverse evolves over time. Many of the changes are responses to feature requests or reported user difficulties with functions. The tidyverse team explicitly solicits feedback from the community on (particularly major) proposed changes via surveys and blog posts. Changes are generally announced with each CRAN release of a package in blog posts as well as in NEWS files of the packages. While the majority of changes are backwards compatible, a carefully evaluated small subset of them can be breaking changes ([Wickham 2021b](#)).

The tidyverse uses the **lifecycle** package to communicate information on the lifecycle of functions and packages ([Henry & Wickham 2021](#)). Clear messaging via lifecycle badges can help instructors evaluate whether to teach newly introduced functionality. For example, one might choose to teach a new function that is in a *stable* stage but might hold off on an *experimental* one. Similarly, *deprecated* and *superseded* functions are good candidates for removal from course materials when updating them.

Teaching the tidyverse will therefore always take a little preparation before class, even if you have materials from a previous term. Because of occasional breaking changes, you will want to re-run any code you are providing to students before class. However, quickly reviewing materials before class is an important practice for any instructor.

5.3 Rebuttals

One common argument against teaching with the tidyverse is a general objection to teaching with packages ([Matloff 2020](#)). There are two main rationales for this point of view. First, detractors fear students will learn a set of functions too specialized or idiosyncratic to be able to translate into more general R syntax they are likely to see after the course. Second, these detractors may focus on the durability and robustness of the code written. Since packages change much more frequently than R itself, code that relies on packages is more likely to break in the future. While these concerns are valid, we offer rebuttals in the case of the tidyverse.

To the first objection, the tidyverse has become so popular (see Section [4.5](#)) that the fear

that tidyverse code will be unrecognizable is unfounded. While this objection might hold for other teaching-focused R packages, the tidyverse ecosystem is too big for this to be a legitimate concern. In fact, the opposite may be closer to the truth. In recent years we have seen ideas that were popularized in the tidyverse implemented in base R. Namely, the change in the default behavior of the `stringsAsFactors` argument to `data.frame()` that occurred in R 4.0, and the introduction of the native pipe operator (`|>`) in R 4.1.

To the second objection, while breaking changes do occur in the tidyverse, the packages themselves are very well-maintained. We ascribe these breaking changes as the price one has to pay for software that is continually progressing. While robustness is important, there is a complementary danger of missing out on innovations that will put students in better positions to succeed.

Other criticisms of teaching R with the tidyverse to introductory students center around the tidyverse's extensive use of non-standard evaluation (NSE). Much of the user-centered design of the tidyverse relies on the use of NSE within R (e.g., not having to quote column names within a `dplyr` function). The complexity of NSE is hidden from students because it is not something they will meaningfully encounter until they try to write certain kinds of functions. Specifically, a generic function that uses functions from the tidyverse, which in turn make use of NSE. This need is unlikely to arise in a first or even second course in statistics or data science. In advanced courses that might teach R as a programming language (e.g., package development), programming with NSE as well as other evaluation patterns used in R can be covered. Many tidyverse packages provide specific documentation to help users learn how to use tidyverse tools in functions and packages they write (Wickham, François, Henry & Müller 2021b, Wickham, Chang, Henry, Pedersen, Takahashi, Wilke, Woo, Yutani & Dunnington (2021b)).

5.4 Coda

We are all converts to the tidyverse and have made a conscious choice to use it in our research and our teaching. We each learned R without the tidyverse and have all spent quite a few years teaching without it at a variety of levels from undergraduate introductory statistics courses to graduate statistical computing courses. This paper is a synthesis of

the reasons supporting our tidyverse choice, along with benefits and challenges associated with teaching statistics with the tidyverse.

Acknowledgements

The authors thank Mara Averick, Ken Kleinman, Roger Peng, Randall Pruim, Tracy Teal, and Hadley Wickham for their comments on an earlier draft of this manuscript.

References

Bache, S. M. & Wickham, H. (2020), *magrittr: A Forward-Pipe Operator for R*. R package version 2.0.1.

URL: <https://CRAN.R-project.org/package=magrittr>

Baumer, B., Cetinkaya-Rundel, M., Bray, A., Loi, L. & Horton, N. J. (2014), ‘R mark-down: Integrating a reproducible analysis tool into introductory statistics’, *Technology Innovations in Statistics Education* **8**(1).

URL: <https://escholarship.org/uc/item/90b2f5xh>

Baumer, B. S., Kaplan, D. T. & Horton, N. J. (2021), *Modern Data Science with R*, 2nd edn, Chapman and Hall/CRC Press: Boca Raton.

URL: <https://www.routledge.com/Modern-Data-Science-with-R/Baumer-Kaplan-Horton/p/book/9780367191498>

Beckman, M. D., Çetinkaya-Rundel, M., Horton, N. J., Rundel, C. W., Sullivan, A. J. & Tackett, M. (2021), ‘Implementing version control with git and github as a learning objective in statistics and data science courses’, *Journal of Statistics and Data Science Education* **29**(sup1), S132–S144.

Bray, A., Ismay, C., Chasnovski, E., Couch, S., Baumer, B. & Cetinkaya-Rundel, M. (2021), *infer: Tidy Statistical Inference*. <https://github.com/tidymodels/infer>, <https://infer.tidymodels.org/>.

Carver, R., Everson, M., Gabrosek, J., Horton, N. J., Lock, R. H., Mocko, M., Rossman, A., Rowell, G. H., Velleman, P., Witmer, J. A. & Wood, B. (2016), *Guidelines for Assessment and Instruction in Statistics Education: College Report 2016*, American Statistical Association: Alexandria, VA.

URL: <https://commons.erau.edu/publication/1083>

Çetinkaya-Rundel, M., Diez, D., Bray, A., Kim, A. Y., Baumer, B., Ismay, C., Paterno, N. & Barr, C. (2021), *openintro: Data Sets and Supplemental Functions from OpenIntro Textbooks and Labs*. R package version 2.2.0.

URL: <https://CRAN.R-project.org/package=openintro>

- Çetinkaya-Rundel, M. & Ellison, V. (2020), ‘A fresh look at introductory data science’, *Journal of Statistics Education* pp. 1–11.
URL: <https://doi.org/10.1080/10691898.2020.1804497>
- Çetinkaya-Rundel, M. & Hardin, J. (2021), *Introduction to Modern Statistics*, OpenIntro: Los Angeles.
URL: <https://www.openintro.org/book/ims/>
- Codd, E. F. (1970), ‘A relational model of data for large shared data banks’, *Communications of the ACM* **13**(6), 377–387.
URL: <https://doi.org/10.1145/362384.362685>
- Dierker, L. (2021), *Passion-Driven Statistics: A Supportive, Multidisciplinary, Project-Based, Introductory Course*, Wesleyan University: Middletown, CT.
URL: <https://passiondrivenstatistics.wescreates.wesleyan.edu/e-book>
- Garnier, S. (2021), *viridis: Colorblind-Friendly Color Maps for R*. R package version 0.6.1.
URL: <https://CRAN.R-project.org/package=viridis>
- Google (2021), *Google’s R Style Guide*, Google: Mountain View, CA.
URL: <https://google.github.io/styleguide/Rguide.html>
- Guzman, L. M., Pennell, M. W., Nikelski, E. & Srivastava, D. S. (2019), ‘Successful integration of data science in undergraduate biostatistics courses using cognitive load theory’, *CBE—Life Sciences Education* **18**(4), 1–10.
URL: <https://doi.org/10.1187/cbe.19-02-0041>
- Henry, L. & Wickham, H. (2020), *purrr: Functional Programming Tools*. R package version 0.3.4.
URL: <https://CRAN.R-project.org/package=purrr>
- Henry, L. & Wickham, H. (2021), *lifecycle: Manage the Life Cycle of your Package Functions*. R package version 1.0.0.
URL: <https://CRAN.R-project.org/package=lifecycle>

Hermans, F. & Aldewereld, M. (2017), Programming is writing is programming, *in* ‘Companion to the First International Conference on the Art, Science and Engineering of Programming’, Programming ’17, Association for Computing Machinery, New York, NY, USA.

URL: <https://doi.org/10.1145/3079368.3079413>

Hermans, F., Swidan, A. & Aivaloglou, E. (2018), Code phonology: an exploration into the vocalization of code, *in* ‘2018 ACM/IEEE 26th International Conference on Program Comprehension’.

URL: <https://doi.org/10.1145/3196321.3196355>

Horton, N. J., Baumer, B. S. & Wickham, H. (2015), ‘Taking a chance in the classroom: Setting the stage for data science: Integration of data management skills in introductory and second courses in statistics’, *Chance* **28**(2), 40–50.

URL: <https://doi.org/10.1080/09332480.2015.1042739>

Horton, N. J. & Hardin, J. S. (2021), ‘Integrating computing in the statistics and data science curriculum: Creative structures, novel skills and habits, and ways to teach computational thinking’, *Journal of Statistics and Data Science Education* **29**(sup1), S1–S3.

URL: <https://doi.org/10.1080/10691898.2020.1870416>

Hyndman, R. J. & Athanasopoulos, G. (2021), *Forecasting: Principles and Practice*, 3rd edn, OTexts: Melbourne, Australia.

URL: <https://otexts.com/fpp3/>

Ismay, C. & Kim, A. Y. (2019), *Statistical Inference via Data Science: A ModernDive into R and the Tidyverse*, Chapman and Hall/CRC Press: Boca Raton.

URL: <https://moderndive.com/>

Kleinman, K. & Horton, N. J. (2009), *SAS and R: Data management, statistical analysis, and graphics*, Chapman and Hall/CRC: New York.

URL: <https://doi.org/10.1201/9781420070590>

Kling, R. (1977), ‘The organizational context of user-centered software designs’, *MIS quar-*

terly **1**(4), 41–52.

URL: <https://doi.org/10.2307/249021>

Kuhn, M. & Wickham, H. (2021), *tidymodels: Easily Install and Load the Tidymodels Packages*. R package version 0.1.3.

URL: <https://CRAN.R-project.org/package=tidymodels>

Matloff, N. (2020), ‘Tidyverse skeptic: An alternate view of the tidyverse ”dialect” of the R language, and its promotion by RStudio’, GitHub.

URL: <https://github.com/matloff/TidyverseSkeptic/blob/master/READMEFull.md>

McNamara, A. (2015), Bridging the Gap Between Tools for Learning and for Doing Statistics, Phd thesis, University of California, Los Angeles.

URL: <https://www.proquest.com/docview/1694580439>

McNamara, A. (2019), ‘Key attributes of a modern statistical computing tool’, *The American Statistician* **73**(4), 375–384.

URL: <https://doi.org/10.1080/00031305.2018.1482784>

McNamara, A. (2021a), ‘R syntax cheatsheet’.

URL: <https://osf.io/2k8fw/>

McNamara, A. (2021b), ‘Reading R code for ”An educator’s perspective of the tidyverse”’.

URL: <https://osf.io/r8mez/>

McNamara, A. & Horton, N. J. (2018), ‘Wrangling categorical data in R’, *The American Statistician* **72**(1), 97–104.

URL: <https://doi.org/10.1080/00031305.2017.1356375>

McNicholas, P. D. & Tait, P. (2019), *Data Science with Julia*, CRC Press: Boca Raton.

URL: <https://www.routledge.com/Data-Science-with-Julia/McNicholas-Tait/p/book/9781138499980>

Myint, L., Hadavand, A., Jager, L. & Leek, J. (2020), ‘Comparison of beginning r students’ perceptions of peer-made plots created in two plotting systems: A randomized experi-

ment’, *Journal of Statistics Education* **28**(1), 98–108.

URL: <https://doi.org/10.1080/10691898.2019.1695554>

National Academies of Science, Engineering, and Medicine (2018), *Data Science for Undergraduates: Opportunities and Options*, National Academies Press: Washington, DC. Accessed: 2020-06-07.

URL: <https://nas.edu/envisioningds>

Nolan, D. & Perrett, J. (2016), ‘Teaching and learning data visualization: Ideas and assignments’, *The American Statistician* **70**(3), 260–269.

URL: <https://doi.org/10.1080/00031305.2015.1123651>

Nolan, D. & Temple Lang, D. (2010), ‘Computing in the statistics curriculum’, *The American Statistician* **64**(2), 97–107.

URL: <https://doi.org/10.1198/tast.2010.09132>

Norman, D. A. & Draper, S. W. (1986), *User Centered System Design; New Perspectives on Human-Computer Interaction*, 1st edn, L. Erlbaum Associates Inc.: Hillsdale, NJ, USA.

URL: <https://dl.acm.org/doi/10.5555/576915>

Postel, J. (1980), ‘Dod standard internet protocol’, *ACM SIGCOMM Computer Communication Review* **10**(4), 12–51.

URL: <https://datatracker.ietf.org/doc/html/rfc760>

R Core Team (2021), *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria.

URL: <https://www.R-project.org/>

Roback, P. & Legler, J. (2021), *Beyond Multiple Linear Regression: Applied Generalized Linear Models and Multilevel Models In R*, Chapman and Hall Texts in Statistical Science, 1st edn, CRC Press: Boca Raton.

URL: <https://bookdown.org/roback/bookdown-BeyondMLR/>

- Robinson, D., Hayes, A. & Couch, S. (2021), *broom: Convert Statistical Objects into Tidy Tibbles*. R package version 0.7.8.
URL: <https://CRAN.R-project.org/package=broom>
- Sarkar, D. (2021), *lattice: Trellis Graphics for R*. R package version 0.20-44.
URL: <http://lattice.r-forge.r-project.org/>
- Smith, D. (2016), ‘Welcome to the Tidyverse’.
URL: <https://blog.revolutionanalytics.com/2016/09/tidyverse.html>
- Soloway, E., Guzdial, M. & Hay, K. E. (1994), ‘Learner-Centered Design: The Challenge for HCI in the 21st Century’, *Interactions* **1**(2), 36–48.
URL: <https://doi.org/10.1145/174809.174813>
- Swidan, A. & Hermans, F. (2019), The effect of reading code aloud on comprehension: An empirical study with school students, *in* ‘Proceedings of the ACM Conference on Global Computing Education’, CompEd ’19, Association for Computing Machinery, New York, NY, USA, pp. 178–184.
URL: <https://doi.org/10.1145/3300115.3309504>
- Tucker, M., Shaw, S., Son, J. & Stigler, J. (2021), Integrating R in a college statistics course improves student attitudes toward programming, *in* ‘Annual Meeting of the American Educational Research Association’, Orlando, Florida. accepted.
- Van Rossum, G. & Drake Jr, F. L. (1995), *Python tutorial*, Vol. 620, Centrum voor Wiskunde en Informatica: Amsterdam.
URL: <https://fossies.org/linux/misc/python-3.9.5-docs-pdf-a4.tar.bz2/docs-pdf/tutorial.pdf>
- Voss, S. E. (2019), ‘Resource review’, *Ear and Hearing* **40**(6), 1481.
URL: <https://doi.org/10.1097/aud.0000000000000790>
- Wang, X., Rush, C. & Horton, N. J. (2017), ‘Data visualization on day one: Bringing big ideas into intro stats early and often’, *Technology Innovations in Statistics Education*

10(1).

URL: <https://escholarship.org/uc/item/84v3774z>

Wickham, H. (2014), ‘Tidy data’, *Journal of Statistical Software* **59**(10), 1–23.

URL: <http://dx.doi.org/10.18637/jss.v059.i10>

Wickham, H. (2015), ‘I’m Hadley Wickham, Chief Scientist at RStudio and creator of lots of R packages (incl. ggplot2, dplyr, and devtools). I love R, data analysis/science, visualisation: ask me anything!’, reddit.com.

URL: https://www.reddit.com/r/dataisbeautiful/comments/3mp9r7/im_hadley_wickham_chief_scientist/

Wickham, H. (2018), *reshape: Flexibly Reshape Data*. R package version 0.8.8.

URL: <http://had.co.nz/reshape>

Wickham, H. (2019), “‘please help me figure out good names for the new pivot verbs in tidyr by filling out this (very short!) survey: <https://forms.gle/vvygbw1ewhk69ga17> #rstats”, Twitter.

URL: <https://twitter.com/hadleywickham/status/1109132826631421952>

Wickham, H. (2020a), *plyr: Tools for Splitting, Applying and Combining Data*. R package version 1.8.6.

URL: <https://CRAN.R-project.org/package=plyr>

Wickham, H. (2020b), *reshape2: Flexibly Reshape Data: A Reboot of the Reshape Package*. R package version 1.4.4.

URL: <https://github.com/hadley/reshape>

Wickham, H. (2021a), *forcats: Tools for Working with Categorical Variables (Factors)*. R package version 0.5.1.

URL: <https://CRAN.R-project.org/package=forcats>

Wickham, H. (2021b), ‘Maintaining the house the tidyverse built’.

URL: <https://www.rstudio.com/resources/rstudioglobal-2021/maintaining-the-house-the-tidyverse-built/>

Wickham, H. (2021c), *tidyr: Tidy Messy Data*. R package version 1.1.3.

URL: <https://CRAN.R-project.org/package=tidyr>

Wickham, H. (2021d), *The tidyverse style guide*, bookdown.

URL: <https://style.tidyverse.org>

Wickham, H., Averick, M., Bryan, J., Chang, W., McGowan, L. D., François, R., Grolemond, G., Hayes, A., Henry, L., Hester, J., Kuhn, M., Pedersen, T. L., Miller, E., Bache, S. M., Müller, K., Ooms, J., Robinson, D., Seidel, D. P., Spinu, V., Takahashi, K., Vaughan, D., Wilke, C., Woo, K. & Yutani, H. (2019), ‘Welcome to the tidyverse’, *Journal of Open Source Software* 4(43), 1686.

URL: <https://doi.org/10.21105/joss.01686>

Wickham, H., Chang, W., Henry, L., Pedersen, T. L., Takahashi, K., Wilke, C., Woo, K., Yutani, H. & Dunnington, D. (2021a), *ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics*. R package version 3.3.5.

URL: <https://CRAN.R-project.org/package=ggplot2>

Wickham, H., Chang, W., Henry, L., Pedersen, T. L., Takahashi, K., Wilke, C., Woo, K., Yutani, H. & Dunnington, D. (2021b), *Using ggplot2 in packages*.

URL: <https://ggplot2.tidyverse.org/articles/ggplot2-in-packages.html>

Wickham, H., François, R., Henry, L. & Müller, K. (2021a), *dplyr: A Grammar of Data Manipulation*. R package version 1.0.7.

URL: <https://CRAN.R-project.org/package=dplyr>

Wickham, H., François, R., Henry, L. & Müller, K. (2021b), *Programming with dplyr*.

URL: <https://dplyr.tidyverse.org/articles/programming.html>

Wickham, H., Girlich, M. & Ruiz, E. (2021), *dbplyr: A dplyr Back End for Databases*. R package version 2.1.1.

URL: <https://CRAN.R-project.org/package=dbplyr>

Wickham, H. & Grolemond, G. (2016), *R for data science: import, tidy, transform, visu-*

alize, and model data, O'Reilly Media, Inc.: Sebastopol, CA.

URL: <https://r4ds.had.co.nz/>

Wickham, H., Navarro, D. & Pedersen, T. L. (2021), *ggplot2: Elegant graphics for data analysis*, 3rd edn, Springer: New York.

URL: <https://ggplot2-book.org/>

Wilkinson, L. (2012), The grammar of graphics, in ‘Handbook of Computational Statistics’, Springer, pp. 375–414.

URL: https://doi.org/10.1007/978-3-642-21551-3_13