





Computational skills by stealth in introductory data science teaching

Wesley Burr¹  | Fanny Chevalier^{2,3} | Christopher Collins⁴  |
Alison L Gibbs³  | Raymond Ng⁵ | Chris J Wild⁶ 

¹Department of Mathematics, Trent University, Peterborough, Ontario, Canada

²Department of Computer Science, University of Toronto, Toronto, Ontario, Canada

³Department of Statistical Sciences, University of Toronto, Toronto, Ontario, Canada

⁴Faculty of Science, Ontario Tech University, Oshawa, Ontario, Canada

⁵Department of Computer Science, University of British Columbia, Vancouver, Canada

⁶Department of Statistics, University of Auckland, Auckland, New Zealand

Correspondence

Chris J Wild, Department of Statistics, University of Auckland, Auckland, New Zealand.
Email: c.wild@auckland.ac.nz

Abstract

In 2010, Nolan and Temple Lang proposed “integration of computing concepts into statistics curricula at all levels.” The unprecedented growth in data and emphasis on data science has provided an impetus to finally realizing full implementations of this in new statistics and data science programs and courses. We discuss a proposal for the stealth development of computational skills in students’ exposure to introductory data science through careful, scaffolded exposure to computation and its power. Our intent is to support students, regardless of interest and self-efficacy in coding, in becoming data-driven learners, who are capable of asking complex questions about the world around them, and then answering those questions through the use of data-driven inquiry. Reference is made to the computer science and statistics consensus curriculum frameworks the International Data Science in Schools Project (IDSSP) recently published for secondary school data science or introductory tertiary programs, designed to optimize data-science accessibility.

KEYWORDS

accessibility, computational thinking, data science education, statistical thinking, statistics education, teaching statistics

1 | INTRODUCTION

In 2010, Nolan and Temple Lang [27] called for “computing concepts to be integrated into the statistics curricula at all levels.” At the time of their paper, data science had not yet received the attention and broad consciousness it has currently, yet their call for a “significant cultural shift to embrace computing” reflected many ongoing developments in statistics and data science education, including initiatives in statistics education that fully embrace computation [2], and anticipated full-scale collaborations between computer scientists and statisticians. The International Data Science in Schools Project (<http://idssp.org/>, [22]) is one such collaboration, creating resources for data science education in secondary schools and introductory tertiary levels. In this

paper, we address and demonstrate the approach to computation advocated by the project.

The recent emphasis on data science addresses the needs of a rapidly changing society in which leaders and citizens (community members) should have an understanding of how the use of appropriate methods for learning from data can allow us to face important challenges, from the development of self-driving cars to addressing climate change. IDSSP is a collaboration between statisticians, computer scientists, and educational experts in these fields, aimed for a broad range of students, and also to motivate talented students to pursue further study in data science to address these challenges.

The IDSSP team has developed curriculum frameworks [22] for both teaching data science in the final

2 years of secondary studies or at introductory tertiary level, and for preparing teachers to teach data science. The focus of the frameworks is learning from data, and the development of the necessary statistical and computational skills and corresponding understanding are introduced as needed. A key IDSSP design criterion has been that data science learning should be accessible to a very broad spectrum of students and not just a small minority for whom coding comes easily. It is critical that the community interested in the development of data science confronts the danger that the paralyzing “math anxiety” problems that have bedeviled approaches to statistics education to a greater or lesser extent over many years of statistical development, might simply be replaced by “coding anxiety” in data science.

In section 2, we describe how the desire for a broadly accessible and foundational data science curriculum motivates a stealth approach to computation, with reference to IDSSP. Section 3 details this approach to computation, and section 4 gives two case studies to illustrate how advanced topics in data science could be brought within reach of senior secondary school and introductory tertiary students when appropriate scaffolding has taken place.

2 | CONTEXT AND MOTIVATION

2.1 | Foundational data science and IDSSP

The ubiquity of data and computing power has led to a burgeoning demand for people with professional data science skills and the need for all citizens to understand the role of data in the decisions in their lives. Furthermore, advancing technology is empowering a broader spectrum of citizens to step beyond simply understanding the role of data to gaining significant capabilities in using data. Hence the educational system needs to include foundations in data science, to better enable future learning of important elements of data science for multiple disciplines as well as the more specialized learning leading to the professional practice of data science.

Data science problems require diverse perspectives and approaches to avoid bias and provide better solutions. The practice of data science is often described as being essentially a team or collaborative enterprise, requiring the bringing together of skills relating to data and computing. Statistics has been both a user and driver of computing technology development since the first computer occupied a whole basement. Big data, data availability and massive computer power have brought data science “out from the back room” and have both

driven, and been driven by, statistics and computing. The statistical sciences are inextricably linked to all endeavors involving data, variation and uncertainty across disciplines, business, industry, government and society. Gibson [18] defines statistical leadership as collaborative leadership. The exhortations by many over many years for teaching statistics to reflect the practice of statistics, are now being extended to “greater data science” [9]. However, one describes statistics and data science, they are clearly inextricably linked together as a broad discipline, and it is imperative that this is reflected in education.

The IDSSP aim is for data-science education to be valuable, accessible, enjoyable and enticing - conferring skills that are *valuable* for student's future lives, further study and careers; *accessible* to a broad spectrum of students; *enjoyable* for both students and teachers; and *enticing* in the sense of arousing in students (and teachers) a desire to learn more.

The purpose of the IDSSP is to promote and support the teaching of introductory data science everywhere, not to be fine-tuned for any particular country. Because of the extraordinary variety of educational jurisdictions across (and even within) the various countries involved, and differences in patterns of prior educational exposure, it would be impossible to create a single course/curriculum that would satisfy all jurisdictional requirements. We note that “curriculum” is a dangerous word in an international context because it means rather different things in different countries but there is no other word that serves us better. The curriculum frameworks developed are designed for flexible use by curriculum designers, course developers and teachers for guidance and ideas when preparing curricula and courses to meet their own local needs and challenges. Due to this, the IDSSP frameworks are much more detailed than might be expected, however, their topic-driven presentation only provides a map of a landscape to be traversed and not an ordered set of instructional sequences for traversing it; see [16,14] for further discussion and examples of problem-driven strategies.

In preparation for development processes in which statisticians and computer scientists can work together on a shared enterprise, IDSSP statisticians and computer scientists reached a common understanding of what a modern course in data science would most-desirably contain. This involved coming to understand one another's differing preconceptions and priorities and arriving at a consensus, an endeavor that inevitably involves compromise. As it turned out, we were pleasantly surprised at just how easy it was to arrive at shared perspectives. Apart from use of terminology, attitudes to computer programming were the biggest area of difference. To the

computer scientists it was completely obvious that programming has to be a major component of “data science.” In the Computer Science tradition, programming, one of the major contributions of the area, is taught directly. Without question, programming skills and an appreciation of the power of programming had to be advanced even if actual expertise need not be a universal end-goal. For the statisticians, programming is just a means to statistical ends with a broad range of opinions about how much, if any, programming needs to be involved. Additionally, a fundamental tenet of IDSSP is accessibility for all students. Programming “requirements” could not, therefore be prescriptive. We had to make allowance for different strategies that could be applied to cohorts with different backgrounds.

This then begged the important question of what we wanted *all students* to experience with regard to computer programming in data science. The barest essentials, the team concluded, are:

- for students to gain an appreciation for the usefulness of code and its power for automating data science tasks;
- preventing fear of coding from ever taking hold;
- building confidence through starting to make relatively minor modifications to working code;
- and starting students on a journey toward learning to read code like a story, a sequence of human-understandable instructions, and toward learning to write their own code.

The IDSSP frameworks are largely agnostic about code use. Only one Topic Area in the frameworks, entitled “The data-handling pipeline,” focuses explicitly and systematically on programming ideas. But, for the rest, we envisage that the elements above can be introduced by stealth, so that experience and confidence gradually accrue. In subsequent sections we will discuss and illustrate some stealth strategies. The goals are: “This is not that hard. It is really useful and it can even be fun!”

2.2 | A Stealth Approach to Computation

Motivation is crucial to learning, influencing the “direction, intensity, persistence, and quality of learning behaviours in which students engage” [1, pp. 68–9]. Among the factors that have been shown to positively influence learners’ motivation are problems they value as interesting and important [26]. Moreover, harnessing this interest can make the need to learn technical content attractive to a diverse group of students. In addition to working on problems they find interesting, students must also believe that they can and

will be successful in order to be motivated to put in the effort to learn [1, p. 76]. Thus, we need to avoid barriers to self-efficacy. Mathematics anxiety is a well-studied and widespread problem [24]. Similarly, statistics anxiety [28] and programming anxiety [6] have been identified as distinct but related constructs that may also contribute to negative student experiences.

Fundamentally, however, data science *requires* significant levels of computation. Whether by using dashboards, graphical user interfaces (GUIs), or lines of computer code, one cannot work with data without somehow instructing a computer to perform some heavy lifting. It is simply not feasible to do practical, interesting and engaging problems by hand in this field. Thus, to truly develop student skills, computation (but not necessarily coding) must have a central role.

By stealth approaches, we simply mean that we do not tackle something big and potentially scary (in particular, learning to write computer code) head on – at least not initially. Instead, initial learning happens tangentially in the process of trying to do other things. The ignition of motivation and interest in students can drive their desire for skills to improve their ability to better tackle the interesting problem(s), and then the required skills gained organically via student-driven desire and need. For example, Fergusson and Wild [16] demonstrated stealth approaches to early encounters with harvesting and using data from online databases via APIs (application programming interfaces). Their teaching-and-learning design principles prioritize student engagement and accessibility.

In general, good teaching is often a stealth activity [31]. Students’ curiosity and desire to learn can be encouraged by providing motivation through interesting problems that align with their interests or for which teachers have ignited an interest. To solve such problems, students then need new skills. Learning objectives related to the acquisition of these skills can be disguised through activities such as games or problems from application areas. This stealth motivation has been used in a variety of other settings to teach computational and quantitative reasoning. Yevseyeva and Towhidnejad [39] motivated the development of computational thinking through the analysis of a recent nuclear accident in a secondary school chemistry class; Shreve [32] describes the use of computer games to motivate learning; and Gunn [19] describes the teaching of quantitative methods in political science through the study of election data.

Data science is a natural environment for stealth learning of computer programming skills because we have bigger purposes in play (understanding data) and we do not necessarily have to write code to be able to pursue the majority of that data understanding. In this

way, learning concepts-and-skills for data analysis and learning to work with computer code can occur in parallel with very little constraint on the relative speeds at which each of these threads has to be advanced. This leaves these settings wide open to teacher or system choices, taking into account the background and inclinations of their cohort of students.

3 | BUILDING COMPUTATIONAL SKILLS

The IDSSP frameworks were designed to provide greater access to many valuable areas that have previously received no or insufficient emphasis in introductory statistics or computer science. There needs to be much more emphasis on: data acquisition, provenance, ethics, privacy, security and sovereignty; data wrangling techniques and the whole data-handling pipeline. For approximately two decades, statisticians and statistical educators have advocated, and been gradually introducing, more coverage of visualization techniques (including interactive and dynamic graphics) and more multivariate data and contexts in introductory treatments, and these must now become the norm rather than the exception. Exposure to many new areas is provided for by including selections from introductions to supervised and unsupervised machine learning, geolocated (map) data, seasonal time series data, text mining and analytics, image data, and recommender systems (which have attracted a lot of bad publicity recently for their putative roles in spreading conspiracy theories, and societal polarization, on social media). It would simply be impossible to experience much of this in a reasonable timeframe if students were required to write detailed code!

As with long-established principles in learning statistics, to learn data science, students must *do* data science, and as discussed above, this requires a variety of levels of computational skill, depending on the desired outcome. In this section, we describe some specific examples to provide clarity as to possible perceived implementation paths. Computer scientists and statisticians alike agree, as in the IDSSP team, that specifying languages or even classes of languages is too restrictive for a curriculum framework intended to assist in developing courses in many different educational contexts.

Through the remainder of this chapter we often make use of the descriptors *high level* and *low level*. We use these in ways that align with their use in the area of thinking skills and also in computer programming. “High level” relates to working with higher levels of generality and “big pictures.” “Low level” relates to lower levels of generality, programming “from scratch,” and doing all

the details. It must be emphasized that this usage does not refer to high levels of technical competence or to higher levels of education.

3.1 | Accessibility and goals for computation

Modern software can contribute enormously to making statistical and data science concepts accessible to a broad range of students and for equipping the students with substantial capabilities to apply these concepts in practice to interesting, real, complex data. Using computers running modern software is fundamental to any serious exposure to data science. This has long been an issue in teaching statistics, and now raises more serious issues of access to technology (a problem COVID-19 lockdowns have highlighted). Such issues are ongoing, and require championing, time and adaptation in different educational contexts.

However, for students to feel that they are capable of success, they need to use modern computers and software to learn from today's complex data from a wide range of sources. All of the software/apps the vast majority of people use, on their phones or computers, use GUIs (graphical user interfaces) to enable users to tell their app what they want it to do. So this is the computing mode that students are already familiar with. To minimize coding in data science, it is possible to leverage the very wide variety of capabilities that can be accessed through GUI systems. When code is used, we advocate heavy use of “high-level functions.” For example, the typical objective for programming in computer science courses is programming mastery, so that the student can write code from scratch given a task description—typically beginning with small, easily understood tasks. In such a framework, the eventual result is long sets of detailed instructions solving small problems. By comparison, most programming done by statistics and data science practitioners makes heavy use of calls to general functions written by others, whereby much shorter commands result in a great deal of sophisticated processing; here we call these high-level functions. Statistics and data science education needs to work with high-level instructions (as practitioners do) because we need our students to become capable of obtaining serious insights and accomplishing powerful results early, so as to support their motivation and willingness to believe in their own learning capabilities. This is much more difficult to accomplish using lower-level (more detailed, “doing everything”) programming approaches. This also illustrates the parallels between too much programming and too much mathematics in statistics and data science

education. Section 3.2.3 provides more detailed demonstration of the distinctions between high and low levels. Statistics and data science students also need to know that “do it yourself” is a very bad practice when well-tested software exists. It is not just an inefficient use of time, it also leads to large numbers of unnecessary errors.

In section 3.2, we expand on our vision for how coding skills can be developed in this context.

3.2 | Exemplar pathways for coding growth

The use of GUI structures (section 3.2.1) and system-generated code (section 3.2.2) in parallel allow for natural evolution in skill on the part of students, starting at the point-and-click level of data analysis, and through careful, judicious use of system-generated code, slowly removing training wheels and working into modification of pre-written and provided code strings. This, if taken to its natural conclusion, should eventually result in students who have gained sufficient mastery of the concepts to be able to write their own snippets of data analysis code (section 3.2.3), fulfilling the core programming goals discussed in section 2.3. Furthermore, this progression has no set time scale attached, so one implementation for a topic could simply never leave a GUI framework, whereas another (perhaps with a prerequisite of a computer science class) might leap immediately into code.

The use of Jupyter notebooks (<https://jupyter.org/>) and R Markdown files (<https://rmarkdown.rstudio.com/>) for reproducible analyzes is also a natural medium for introducing code re-use and mastery, by providing learners with an already functional literate document [23]. These types of documents, which are used by many professionals, combine pieces of textual narrative and chunks of computer code in a single source file. Such files can be compiled, at any time, to generate an output document in which outputs from the computer code are paced at the appropriate positions in the narrative. Reading and understanding what such a document does can then be scaffolded to allow for modification (“change a variable”) or extension (“what if we consider this other factor?”), and can provide a good medium for implementing stealthy introductions to coding.

Later chunks of code can be combined to automate whole sequences of data wrangling and analysis steps with the intent of enabling students “to gain an appreciation for the usefulness of code and its power for automating data science tasks” (an essential goal listed in section 2.1). This is the idea of a data handling pipeline. Sufficiently motivated students may be guided toward the use of more complex data sources that necessitate

the introduction of some sophisticated multi-stage data wrangling techniques to transform them into formats an analysis program can work with.

3.2.1 | GUIs as a vehicle for growth

Into this discussion, we would like to add some cognitive realities about human memory. Short-term memory is a severely limited resource; studies reported by Cowan [7,8] suggest that the average person can only hold two to six pieces of information in their attention at once. Thus, if someone has to invoke and connect too many ideas at once we quickly run into the problem of cognitive overload. Additionally, long-term memories for the details of how to do things fade fast when they are seldom used. This is a particular problem for programming. Both programming and the use of many GUI systems are beset by the problem of knowing and remembering names. You cannot do anything until you know, and remember, the name of the thing you want to do. This is a significant barrier to getting started and also results in significant time-loss getting back up to speed after a period of inactivity.

Some of the biggest advantages of GUI systems flow from the visual cues and reminders they provide of “What’s on offer here?” Interfaces can provide a structured, top-down way of encountering a new area (eg, working with network data) with progressive revelation guiding thinking through a problem via context-aware choice sets. Thus, GUIs can provide serious new capabilities quickly with reduced learning curves, with their memory cues reducing the dependence on human memory [37]. Therefore, they are well placed for presenting options for “How else can I *look at* that?” (display types), and “How else can I *do* that?” (methods) with instant delivery of results from each tentatively-entertained option—leveraging the modern tendency of people, when given buttons and choices, to ask “I wonder what that does?” and just try things out. Not all GUI systems necessarily use these capabilities well. But good GUI systems are helpful for beginners, occasional users, and doing one-off things really fast (provided the system prioritizes them). They can enable users to see a whole range of things we can do with our data and do them very quickly and with very little effort. Furthermore, the clicking, dragging and hovering gestures inherent in a well-designed GUI can also create a level of immersion in data which is quite different from the (often cold) remoteness of coding systems. This is particularly powerful in the hands of relative novices.

The biggest deficiency of GUIs is their inability to do new things the system has not allowed for. The *flexibility*

and *extensibility* that coding provides is probably the biggest reason we need to start students on their journeys toward coding. There are many others. We have already mentioned the *automation* of repetitive tasks. The ability to *reuse* and *share* code, as-is or after modification, confers huge time-efficiency and knowledge-transfer benefits. This is really brought home by the “Oops! Do it again!” experience: an involved analysis has to be repeated because a mistake was discovered in the data. Repeating all the previous work with a GUI is a big, annoying job but it takes almost no time or effort when using code. Sharing code also facilitates *reproducibility*—the ability for someone else to reproduce (and therefore check) an analyst's results. It automatically generates an *audit trail* of what was done and how, and allows *disciplined practices* for working, sharing, version control, and creating *dynamic documents* in which expository text is interspersed with blocks of code that generate the desired data-analysis outputs such as tables and graphs within a report and can easily be recompiled if changes are needed. Students can profitably experience quite a bit of this, so the benefits of working using code are brought home for them, but without any need to become experts.

Fergusson and Pfannkuch [15] discuss some fascinating research exploring some synergies between statistical and computational thinking through the using of both GUI software and code.

3.2.2 | System-generated code

GUIs and coding have complementary strengths, and students may benefit from exposure to both. But there are also GUI systems that can also write code, making available the code that implemented the actions that the point-and-click interface requested. This code can be taken away to be used elsewhere or modified and rerun in the system. Perhaps the best known and most long-standing example of this in the R ecosystem [30] is R Commander [17] which is a graphic user interface for R. Blue Sky Statistics (www.blueskystatistics.com), Jamovi (www.jamovi.org/) and RKWard (<https://rkwad.kde.org/>) also have this ability; see [25] for comparisons.

The systems above appear to be aimed primarily at data-analytic practitioners and their interests are not pedagogical. IntRo [21] is an R Shiny app produced for introductory statistics courses that makes available the R code it uses to generate its results [20]. It is limited to the analyzes that have been most often used in standard introductory statistics, particularly in the United States, but is also much simpler for beginners for that very reason.

Large parts of iNZight (<https://inzight.nz/>, [11,37,38]), and also its online version iNZight Lite ([\[stat.auckland.ac.nz/\]\(http://stat.auckland.ac.nz/\)\), construct and make available R code. The system takes user instructions from the GUI, constructs R code to implement them, and then runs that code and also stores it \(automatically for anything that changes the data, and otherwise by user request\). Recently, in the interests of further facilitating strategies being discussed here, iNZight's lead developer Tom Elliott has designed and implemented the model exemplified in Figure 1.](https://lite.docker.</p>
</div>
<div data-bbox=)

In all of the windows most often used by beginners:

- The function call that produces the display just asked for is shown - with provision for storing, or changing and rerunning the code
- Settings in the GUI determine the function call and the output
- Changing and rerunning a valid function call changes the settings in the GUI to match the user's code (**Reset** returns things to the last set of GUI instructions)

The strategies in play are:

- “the code that does it” is always in view to foster learning by osmosis
- The mappings between GUI settings, argument values of the function call and output are direct, to foster seeing the relationships between them
- Because the system is responsive, each request in the GUI for a minor change or customization triggers an instant change both in the output and in the code, also helping highlight what particular code elements do
- Opportunity is provided to experiment with the code within a familiar environment with expected behaviors and expected outputs
- Restricting what is always being shown to just the current function call keeps things simple and makes the mappings between code and GUI settings more obvious. (Other strategies are required for learning to “string together” code.)

Currently code is stored automatically for all data-wrangling operations performed, and when asked, for graphics and statistical output produced by the base module (including ggplot code) and the generalized-linear-modeling module.

System-(or GUI-)generated code provides an alternative to instructor-provided code as a source of building blocks for a program. Take data-wrangling operations as an example: we would not expect students to become *experts* at data-wrangling. Even if it was possible, the time it would take would be completely out of proportion to its value in the time available for an introduction to data science. But we would at least like students to often experience using data wrangling operations necessitated by

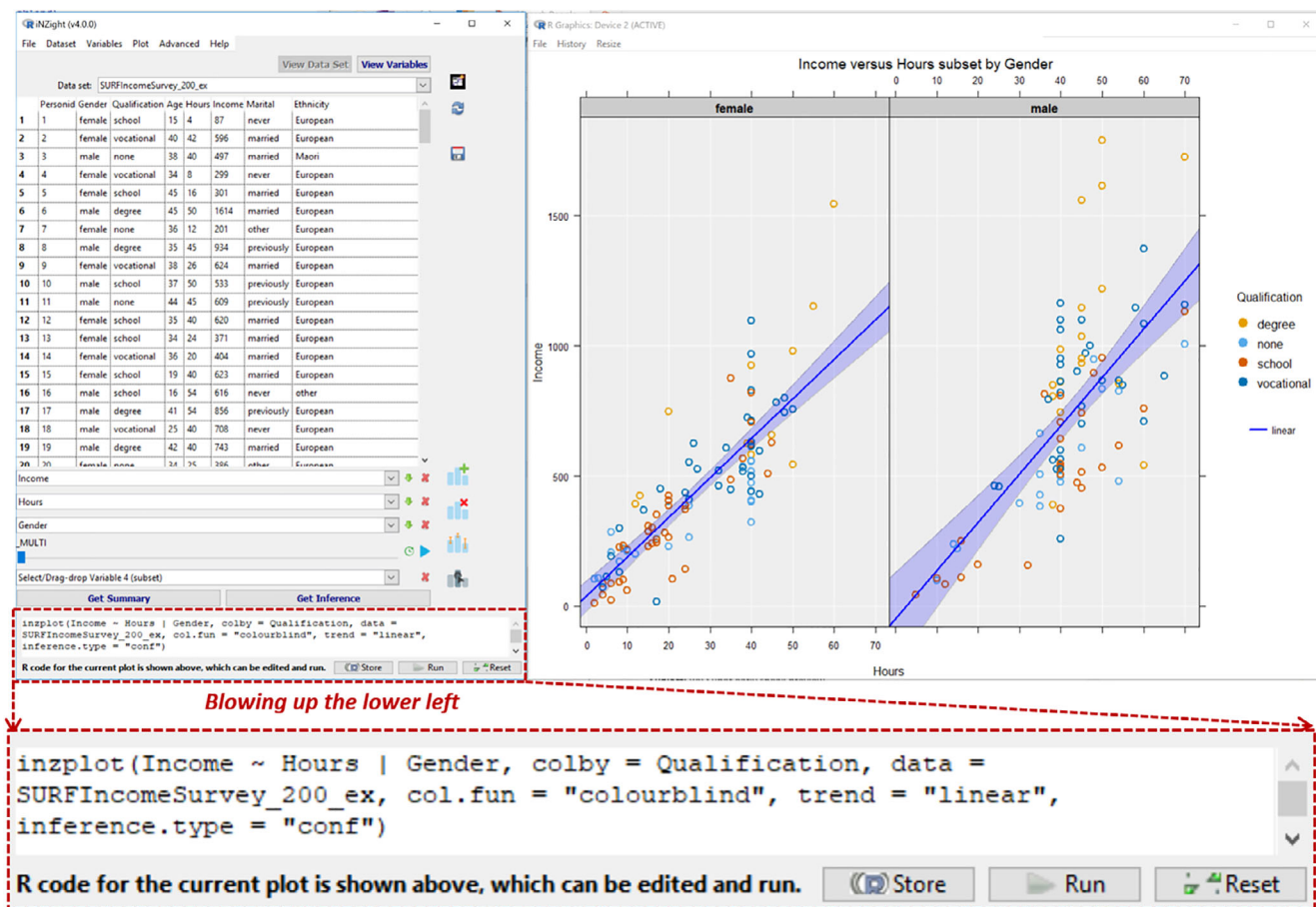


FIGURE 1 Relationships between graphical user interface settings, code and output in iNZight [Colour figure can be viewed at wileyonlinelibrary.com]

particular data sets they were working with as part of repeatedly experiencing the whole data-science cycle. The importance of working through the whole data investigative process has been so long advocated and established by statisticians and statistics educators that it is dispiriting to think we still have to emphasize this. However, now we have an extra dimension to this advocacy, namely, that data wrangling and use of powerful technology tools must be integral to authentic learning of data investigations [27].

iNZight has the capacity to use dialogs to lead users through almost all of the data-wrangling operations in the book *R for Data Science* [35] and write the R code needed. This means that when students need to employ particular operations to wrangle their data in preparation for analysis, the system and its documentation can lead them through those operations, even with side-by-side pre- and post-views of the data for the more complicated ones, and then provide the code they need for their own program to automate a larger process. Working this way contributes to understanding operations and using them in a program. And it can be done at the individual (rather than class) level. Students can also process and analyze data using a GUI, obtain the

code for a program that duplicates everything they have just done, and then modify that program to adapt it for a somewhat different purpose. Obviously, this can also be done with pre-written dynamic documents (eg, R Markdown notebooks; Jupyter notebooks), which students can take and run in order to see functionality, and then later modify for new data sets or new explorations.

We have seen no substantive pedagogical research on using system-written, or teacher-provided code to help with learning to code, just data analysts talking online about how the code-writing provisions of R Commander (for example) helped them to learn R. There is a need for research not only into how code generated by existing systems can be used to enhance learning but also on the ways code delivery and interaction in statistical and data-science GUI systems can be structured to better help learners.

3.2.3 | Levels of coding

We will now elaborate on our distinction between higher-level vs lower-level coding and why this distinction is important. We will take a simple problem and approach it using different levels of coding. Our focus is on the nature

of the code itself and what that code does, not the technological environment in which it is used.

Coding using *tidyverse* commands (www.tidyverse.org, [34]) tends to be higher-level than programming using *base R* commands which is lower-level in the following sense. The *tidyverse* contains collections of powerful functions, built on top of *base R* commands. The *tidyverse* commands are intended to make some very important and commonly used operations much easier for an analyst to perform, with less code and less attention to detail (bigger picture, more generality) than is needed with *base R* commands where more detailed instructions are required (lower-level coding).

This higher-vs-lower-levels-of-coding distinction is important for strategizing about students' knowledge and memory demands, and the avoidance of cognitive overload (section 2.3.1). Take the problem of knowing and remembering names. With code you cannot do anything unless you know and remember the names of the things you are trying to do, and the syntaxes used in putting them together. Lower levels of coding require more programming complexity and higher demands on human knowledge and memory, thus reducing broad accessibility to students.

Our brief illustrations start with the same small, clean rectangular data set from a workforce survey read into a dataframe called **incomeData**. The first six rows are:

```

Personid Gender Qualification Age Hours Income
Marital Ethnicity.
1 1 female school 15 4 87 never European
2 2 female vocational 40 42 596 married European
3 3 male none 38 40 497 married Maori
4 4 female vocational 34 8 299 never European
5 5 female school 45 16 301 married European
6 6 male degree 45 50 1614 married European

```

Very high-level code (in the sense that almost everything about how things are done is decided by defaults) This example uses functions from the *iNZightPlots* package [10] in R, cf. code window in Figure 1.

```
inzplot(Income, data=incomeData)
```

“Show me the data on the variable *Income* (using a plot).”

```
inzplot(Income~Gender, data=incomeData)
```

“Show me the relationship between *Income* and *Gender*.”

```
inzsummary(Income~Gender, data=incomeData)
inzinference(Income~Gender, data=incomeData)
```

“Tell me more about this relationship using (i) the summary statistics and (ii) the inferential information that people usually want to see in a situation like this.”

How these commands are “obeyed” is automated using defaults in the software (taking variable type into account). The simplicity of code that does not expose details makes it very easy to see how important things can be changed; in this case it is easy to see how other variables can be explored and what you would need to do to use a different data set. This also provides a good starting point for learning to read code as a narrative. R is well suited to this as function arguments with defaults need not appear in function calls.

In the above, there is no user control applied to what happens and how. We can get more control by overriding defaults and/or by moving to lower levels of coding. We pay a price, of course, in complexity. By adding arguments that override defaults, the calls above (particularly the plotting calls) are also enormously customizable, thus opening up a huge range of opportunities for exploring the data using features such as changing plot types, faceting, sizing and coloring elements by other variables and labeling; all with little programming complexity, for example,

```
iNZPlot(Income~Qualification | Gender,
plottype="gg_violin", data=incomeData)
```

With extremely high-level code like this, where small pieces of code deliver rich information, students' first experiences of direct interactions with code can associate it with power, not drudgery (they get rich graphics and information with very little effort). This is very similar to the philosophy espoused by statisticians and statistics educators over the past two decades [5], of the importance of the first-day hook: get students up and running with powerful, interactive data analysis as early as functionally possible, so that the power overwhelms the fear.

Intermediate code (in the sense that some decisions about how things are done are decided by defaults, but the user is expected to provide a little more guidance)

We lower the coding level slightly with this example by using the *mosaic* package from Project MOSAIC [29] to obtain similar output.

```
gf_boxplot(Income ~ Gender, data = incomeData)
favstats(Income ~ Gender, data = incomeData)
confint(t.test(Income ~ Gender, data = incomeData))
```

We now need to know a little bit more to issue our instructions. Here, we now need to know that the type of plot we want to see is called a boxplot, the name of the

function that gives you that particular plot, and that for our inferential information we want confidence intervals from the `t.test` function (this last instruction is not actually mosaic).

Mosaic is an important package for learning journeys in R. From its inception, it has prioritized enabling consistent syntax, and easily readable code that conveys a narrative, as a means of stripping away unhelpful complications from students' coding journeys.

Lower-level code (in the sense that fewer decisions about how things are done are decided by defaults, and the user is expected to almost fully guide the analysis)

Our final example lowers the code level further with a call to `ggplot2` [36] and some base R code to produce summaries and inferential information that, for this particular combination of variable types, are almost the same as produced by our highest level of code above.

```
ggplot(data = incomeData, aes(x = Gender, y =
Income)) +
  geom_boxplot() +
  geom_dotplot(binaxis = "y", dotsize = .5) +
coord_flip()
male_incomes <- incomeData$Income[incomeData$Gender
== "male"]
female_incomes <- incomeData$Income[incomeData
$Gender == "Female"]
c(summary(male_incomes), sd(male_incomes, na.rm =
TRUE))
c(summary(female_incomes), sd(female_incomes, na.rm
= TRUE))
t.test(male_incomes)$conf.int
t.test(female_incomes)$conf.int
t.test(Income ~ Gender, data = incomeData)
```

The core point we are trying to make here is that by using sufficiently high-level functions we can reduce the complexity of the code students have to work with immensely, and consequently the cognitive and memory demands put on them. This gives the opportunity to build coding experience gradually without sacrificing speed of progress through learning-from-data experiences even in an entirely code-driven course. Topics encountered early in the course sequence can be tackled with simple calls to powerful high-level commands, while topics encountered later may be able to make more detailed and complex coding demands as students accumulate experience and confidence in working with code. Both ends of the spectrum are illustrated in the case studies in section 4.

In computer programming terms, however, everything we have shown above is actually still very high-level programming. Hadley Wickham's `ggplot` is a very high-level function with powerful abstractions which

facilitate building complex multi-part graphs in a principled way. Even R is itself a very high-level programming language. What we have really been illustrating here are just shadings of degree.

4 | CASE STUDIES ON LEARNING COMPUTATION BY STEALTH

In section 3, we discussed how GUI-driven code can be scaffolded to provide novice students with a starting point which will be accessible to almost any computer user. We then examined and defined different "levels" of coding expertise, categorized primarily based on the amount of support and lifting that the computer system provides. Now, to illustrate this approach to computation, we provide two case studies in how teachers and students might engage with two topics: working with time series data, and interactive data visualization. Each case starts with consideration of a rich, multivariate data set, albeit of different formats, to motivate the need for acquiring new skills and to inspire students' desire to creatively explore. For these topics, students will need sophisticated computational ideas that are not traditionally taught at the secondary school or introductory tertiary level. The case studies illustrate how our approach to teaching computational skills by stealth can make these topics accessible to all students. In addition, following section 3.2.3, the two case studies use quite different coding levels, as a demonstration of the fact that the depth of the material and exploration does not need to be closely tied to the coding level *if* the computational tools available allow for high- or intermediate-level coding.

Our first case study works with seasonal time series data. It takes a high-level approach easily accessible early in a coding learning-journey. It uses very simple, high-level function calls to the `iNZightTS` package [12] in R. Because these are also the function calls that `iNZight`'s GUI-system itself issues and there are obvious mappings between the code and settings in the GUI, the GUI and code could even be used in parallel if so desired. This case study was actually written to be a dynamic document using R Markdown (<https://rmarkdown.rstudio.com/>), and has been brought into this paper in its present form for demonstration purposes—we envision that a classroom setting might start with a templated markdown framework showing a worked example, and then successively adapt that framework into something like what we have here.

The second case study on interactive data visualization, coming from a more computer science perspective, takes a more intermediate- or low-level approach in the sense that the coding experience is much more central to,

and integrated with, the statistical experience. Similar to the first case study, the data visualization work was originally created as a Jupyter notebook (<https://jupyter.org/>), and a similar classroom vision applies. Please note we are not arguing here that time series should be approached one way and interactive visualization the other. Both can be approached from anywhere on the spectrum between GUI and low-level code, and we are simply illustrating how two very different choices can play out.

4.1 | Case study: Climate over time in Toronto

The educational context for this case study is learning to investigate and learn from seasonal time-series data. The real-world context is global warming (or “climate change”), a topic of great interest to many young people. Wherever you live on our planet, the **climate** of our environment appears to be changing. NASA's Earth Observatory program says “The world is getting warmer. Whether the cause is human activity or natural variability – and the preponderance of evidence says it's humans – thermometer readings all around the world have risen steadily since the beginning of the Industrial Revolution ... the average global temperature on Earth has increased by about 0.8° Celsius (1.4° Fahrenheit) since 1880. Two-thirds of the warming has occurred since 1975, at a rate of roughly 0.15–0.20°C per decade.”

Case study: Given that we consider “local” to mean Ontario, Canada (as four of the authors of this paper live there!), (1) Can we observe a warming **trend** in the data from Toronto, Ontario, Canada?; and (2) What **prediction** can we make about average temperatures in the near future?

We will explore some temperature data from Environment and Climate Change Canada which are **time series**: repeated observations of the same unit or measurement over **time**. ECCC maintains hourly

observations of temperature at many locations across Canada which are freely available on a data portal at <http://climate.weather.gc.ca/>. Similar databases exist for other countries such as Australia, New Zealand, the United States of America and the United Kingdom as well as most other countries on Earth.

A primary motivator for developing a computer program for this situation (and almost all time series data) is the fact that new data are always accumulating as time marches on, so we are very likely to want to update any analysis we do sometime in the future to include all the new data. Our process is to first obtain the data from the database, then wrangle it to get it into a form that the R package we want to use likes as input, and then start analyzing it.

The database allows us to extract hourly, daily or monthly data. We will take monthly data. We get a rectangular data set with the variables/fields shown in Figure 2 if we request data for Toronto's Pearson International Airport.

There are fields that are redundant for this analysis because they are characteristics of Pearson Airport and never change, for example, latitude and longitude. There are others that we do not need such as “Mean Temp Flag” which takes the value “M” if the value of “Mean Temperature” is missing, and is blank otherwise; a little investigation reveals there are no missing data from 1970 (actually after 1937). And we may also narrow our focus to a smaller number of variables of particular interest. So we want a wrangling step to reduce the number of variables being considered to a smaller number of variables of particular interest.

When the data are imported, R does not like special characters in variable names; for example, “Extr Min Temp (°C)” is automatically converted to “Extr Min Temp.C.” So another desirable wrangling step is to change the variable names after import to simpler, more attractively-readable names. Additionally, the R package `iNZightTS` [11] we will use for our time series analysis in this case study does not automatically recognize year-month data in the form “1971-02,” but it does automatically recognize

Longitude (x), Latitude (y), Station Name, Climate ID, Date/Time, Year, Month, Mean Max Temp (°C), Mean Max Temp Flag, Mean Min Temp (°C), Mean Min Temp Flag, Mean Temp (°C), Mean Temp Flag, Extr Max Temp (°C), Extr Max Temp Flag, Extr Min Temp (°C), Extr Min Temp Flag, Total Rain (mm), Total Rain Flag, Total Snow (cm), Total Snow Flag, Total Precip (mm), Total Precip Flag, Snow Grnd Last Day (cm), Snow Grnd Last Day Flag, Dir of Max Gust (10's deg), Dir of Max Gust Flag, Spd of Max Gust (km/h), Spd of Max Gust Flag

FIGURE 2 All fields available from ECCC for climate data from Pearson International Airport, in the format presented when downloaded from the portal mentioned

“1971M02” so we want to change the format used by the Date/Time variable (which turns to “Date.Time” on import). Note that the units (degrees Celsius, kilometers per hour) are compressed by this change, and labels will need to be manually updated in the following (eg, for figures).

Once we have wrangled our data we have a data set in a tidy form [33] ready for initial analysis. But when we come to update the analysis with new data we would not want to do all this again. If we had a program that does all the steps, and we wanted to update the analysis we would only have to re-run the program. Coming up with the code to do these things will be either beyond most

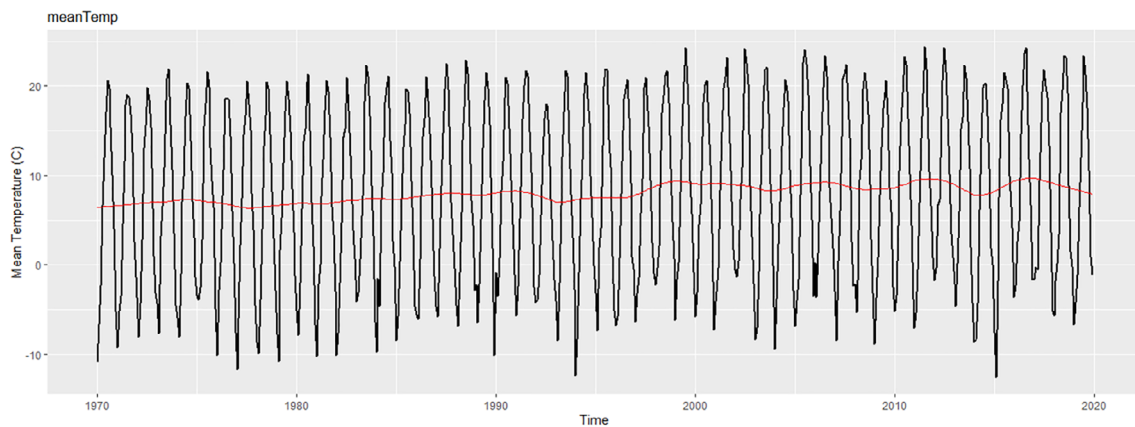
```
head(pearsonClim)
```

| date | year | meanTemp | meanMaxTemp | meanMinTemp | totalPrecip | maxWind |
|---------|------|----------|-------------|-------------|-------------|---------|
| 1970M01 | 1970 | -10.9 | -5.9 | -15.8 | 25.7 | 61 |
| 1970M02 | 1970 | -6.6 | -1.6 | -11.6 | 29.7 | 72 |
| 1970M03 | 1970 | -2.3 | 1.9 | -6.4 | 43.7 | 85 |
| 1970M04 | 1970 | 6.6 | 12.3 | 0.9 | 81.3 | 90 |
| 1970M05 | 1970 | 12 | 18.1 | 5.9 | 56.6 | 63 |
| 1970M06 | 1970 | 17.1 | 23.9 | 10.3 | 38.1 | 60 |

```
library(iNZightTS)
```

```
meanTemps = iNZightTS(pearsonClim, var = "meanTemp")
```

```
plot(meanTemps, t=100, xlab = "Time", ylab = "Mean Temperature (C)")
```



```
seasonplot(meanTemps)
```

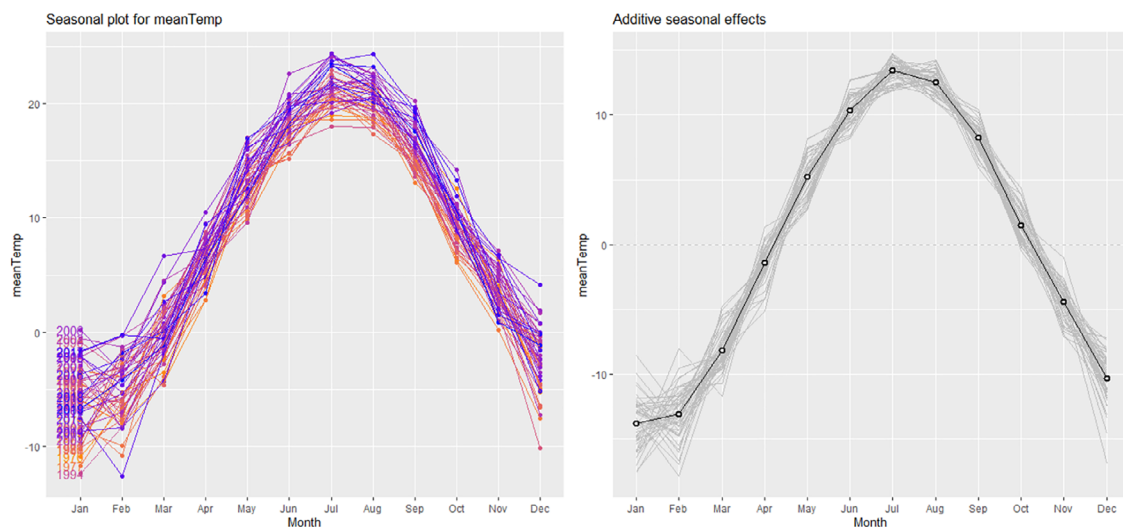


FIGURE 3 Display of the available variables after cleaning and tidying the data, and an introductory graphical analysis of the mean temperature series, from 1970 to 2019 [Colour figure can be viewed at wileyonlinelibrary.com]

students or take a long time to arrive at, and is beyond the scope of an introductory course. Additionally, different data sets require different sets of wrangling steps (from minimal to excessive). Adapting the opening sentence of Tolstoy's *Anna Karenina* (about families), all tidy data is alike but every untidy data set is untidy in its own way. So in this context, teacher input to wrangling-code formation will normally be required. Alternatively system-written code could be helpful.

Once we are in the tidy world, with variables organized, everything is much better systematized. In what follows we emphasize the connection between commands and what they produce, regardless of the programming environment. In Figures 3 and 4 the bolded lines are lines of computer code (commands), while the plain text lines and plots that follow are the computer output produced in response to these commands.

Figure 4 shows that the seasonal movements (month of the year effects) in the mean temperatures are much larger than any movement in the trend (as we would have expected). Showing the seasonal differences restricts

the depiction of movement in the overall trend to a very small interval of vertical space on the graph, thus making it much harder to see what is happening to the trend. So next we aggregate our data to yearly averages and plot that (see Figure 5).

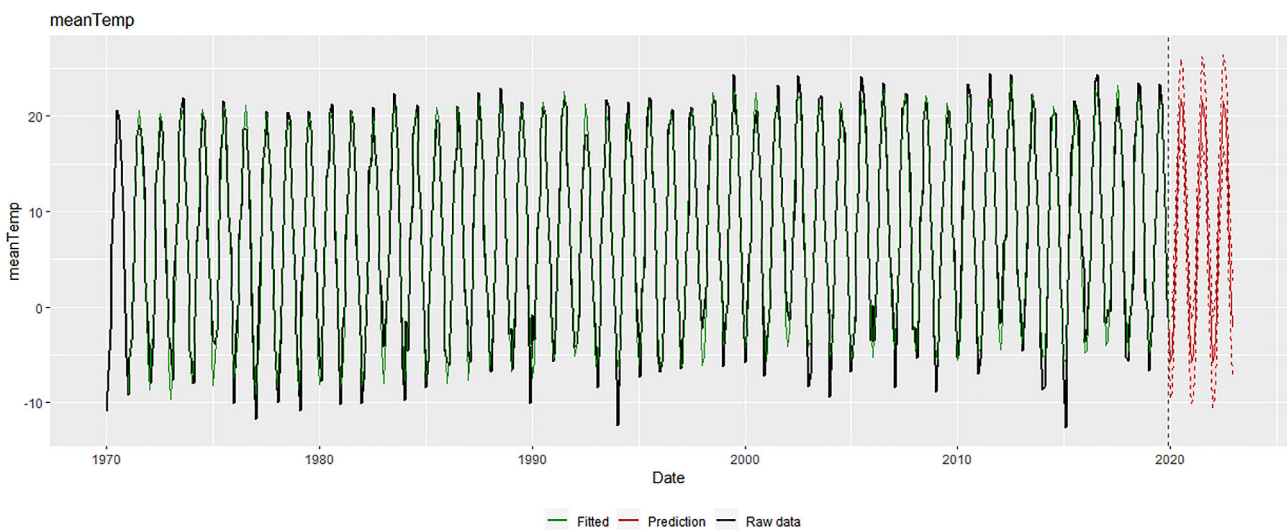
In Figure 5 a warming effect is now much more obvious. Mean annual temperatures at Toronto Airport appear to have climbed fairly steadily from 1970 to around the year 2000 but seem to have leveled off since.

Other variables in the data show contrasting behavior, for example, total precipitation and maximum windspeed have very weak seasonal effects and do not show any consistent pattern of increase or decrease over the time period.

Note that we are using this example to illustrate a high-level computation approach by stealth, but do not comment here on required prior learning or on what curricula topics are needed.

Although we have just shown a minimalist computational-approach to time series here, time series data is a natural fit as a vector for computation by stealth since so much

```
plot(meanTemps, forecast = 36)
```



Forecasts with upper and lower prediction limits

| | fit | upr | lwr |
|----------|------------|------------|-----------|
| Jan 2020 | -5.6143922 | -1.6224643 | -9.60632 |
| Feb 2020 | -4.8569001 | -0.8380746 | -8.875726 |
| Mar 2020 | -0.3247944 | 3.7207499 | -4.370339 |
| Apr 2020 | 6.3629026 | 10.4349903 | 2.290815 |
| ... | ... | ... | ... |
| Oct 2021 | 9.7061322 | 14.277852 | 5.134412 |
| Nov 2021 | 3.0950695 | 7.6902944 | -1.500155 |
| Dec 2021 | -2.0812654 | 2.537345 | -6.699876 |

FIGURE 4 Forecasting mean monthly temperatures for the next 3 years (36 months) [Colour figure can be viewed at wileyonlinelibrary.com]

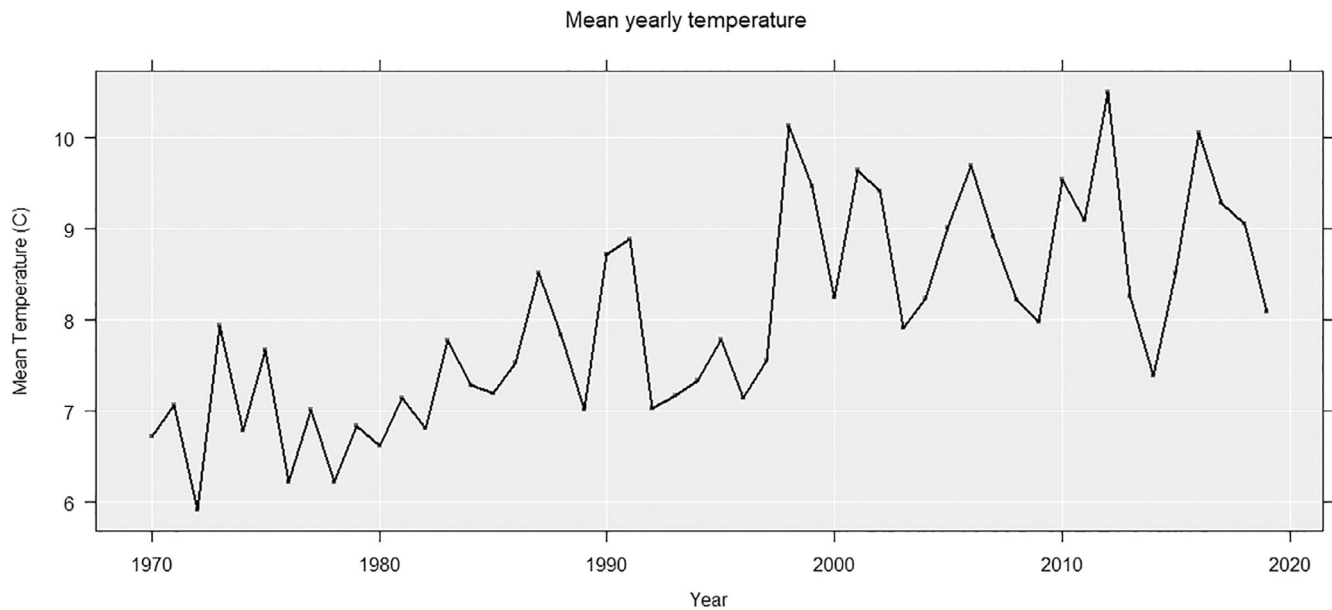


FIGURE 5 Mean yearly temperatures (in degrees Celsius) at Toronto Pearson International Airport, 1970 to 2019

of available real-world data is actually recorded as time series, and time series come with fascinating structural behavior that can be visible directly from plots. Thus, going back to our earlier point about motivation: if teachers wish to encourage student-driven learning via motivated investigations, they will almost certainly encounter time series data as the natural fit for their question-driven inquiry, allowing students to feel powerful as they “detective” their way through an investigation. Further, time series naturally come with a date-time structure, which almost *requires* code to be easily wrangled: encouraging the cleaning of dates and times is a great hook to get students interested in repeatable, reproducible analysis-focused code.

The types of plot shown above, together with decomposition plots and plots of multiple series, have been the mainstay of teaching, data exploration and assessment for time series at the last-year of high school in New Zealand since 2013. (In New Zealand, the majority of students entering their last year of school study statistics.) The educational experience is centered on using graphs of data to make sense of the world and the students typically use GUI systems [4]. High-stakes assessments call for written reports on what the student has learned from a particular set of data.

4.2 | Case study: Interactive data visualization

Many modern data sets are highly multivariate, meaning items in the data set have many attributes, and these

attributes can be quantitative or categorical. Many such data sets are represented in a tabular format in a standard relational database. Examples include the features of a computer (speed, storage, price, screen resolution), the characteristics of a movie (length, box office sales, date of release, genre) or information about athletes' performance (name, sex, age, height, weight, year, sport, medal). While tables are a helpful way to organize and sort data, discovering relationships between attributes using a table or a collection of tables can be challenging. Appropriate visualizations of such data can help identify patterns or form hypotheses [3,13]. However, because of the sheer number of attributes, it is not trivial to determine which particular visual representations of high-dimensional data sets best support decision making or variable selection for model building.

This case study gives an approach for interactive data visualization by stealth through a learning sequence. It assumes the data set is complete and does not have significant errors, as dealing with data cleaning and missing data is a different problem. There is a plethora of open multivariate data sets that are readily available online, and teachers can also engage students in producing a data set of their own through simple data collection methods involving sensing technology, surveys, or even directly scraping data from webpages where allowed, or through APIs. This allows for endless possibilities in terms of the themes and problems that students can perform data analysis on, to foster motivation. Most importantly, in demonstrating to students, teachers should focus on a data problem that is relatable to the students, for which the answer is not trivial,

and for which the data set is rich and complex enough to pose interesting data analysis and visualization challenges.

Case study: Your parents are considering purchasing a new computer for the family. You have done some research and found a spreadsheet of about 6000 models to consider, including the price, processor speed, hard drive size, RAM, graphics capabilities, brand reputation, and warranty. Can you trim down this list to retain 5 top options based on the computer properties? Which properties will you choose to focus on? How can you then effectively communicate to your parents why you have chosen these options, as compared to any other choices from the initial list?

The data set used in this case is large both in terms of the number of items (n) and dimensions (p) to consider, making it virtually impossible to process the data manually. Note that teachers could also easily adapt this case to identifying a subset of points of interest in any multivariate data set and situating these items with regards to the larger collection (eg, What are the best cities to travel to? What book should I read next? What are the most endangered species in North America? What are the highest ranked colleges? What makes an exceptional athlete exceptional?).

Note that this case study is not an example of a data investigation but is focusing on producing some visualizations useful in exploring and presenting aspects of a large dataset. Criteria have not been set for choices, there is no randomness in the data, no inferences to be made, and no modeling to be done. It is an example of some

useful visualizations for a large dataset with a large number of attributes (variables). By choosing to explore a large fixed non-random data set, we can focus on learning visualization choices by stealth in a context familiar to students without any considerations needed to be given to all the issues and complications of a statistical data investigation.

The first useful activity with the computer data set in this case study could be to create single charts using commercial GUI-driven visualization technology, such as Tableau, PowerBI, or even Excel to explore different facets of the dataset. For example, the students could examine the relationship between *price* and *processor speed* using a 2D scatterplot. Given the complex relations between the attributes of the data set, and the number of items, it quickly becomes apparent that examining the data from one point of view at a time does not give a clear picture of the options.

The next activity could be to create *coordinated views* of the data, sometimes called a **dashboard**. These views can be interactively linked to reveal the same data item across plots, or to highlight items matching a selection on any given plot. For example, a scatterplot of *price* and *processor speed* can be interactively linked to a histogram of the number of computers in each bin of RAM (see the left sub-plot in Figure 6 for the histogram) to allow for interactive highlighting of points in each RAM group. This activity builds the ability of students to explore data from multiple perspectives, seeing that a data item (a computer in this case) can be viewed in different ways based on a question of interest. Software such as Tableau and PowerBI offer the capability to create these dashboards easily. Because there are virtually an infinite number of ways to design a single dashboard, learners will be faced with the question of what

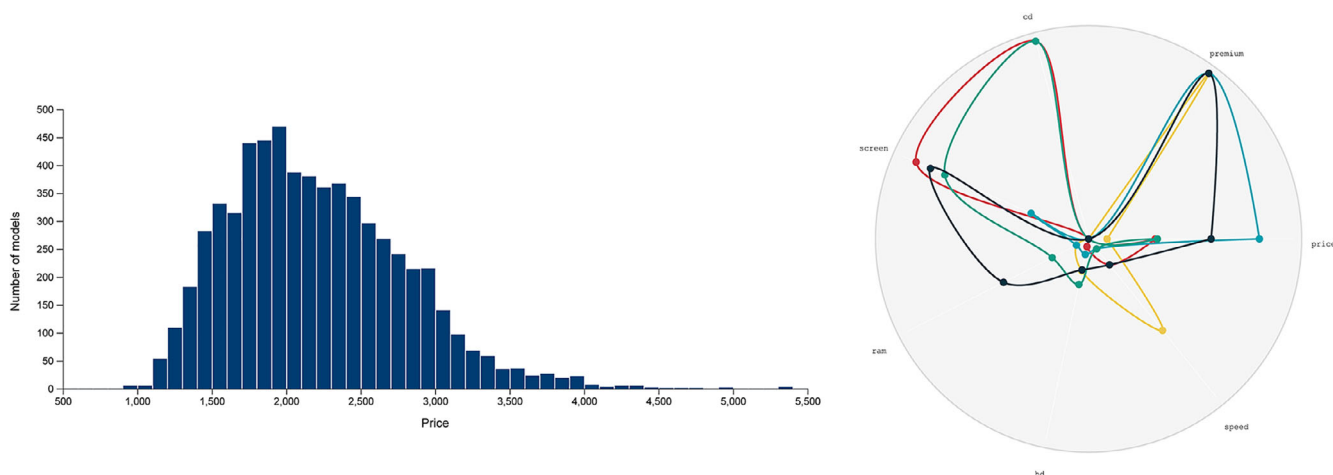


FIGURE 6 Histogram displaying the number of computer models binned, and a radar chart of the features of the five selected computers, developed using Observable notebooks [Colour figure can be viewed at wileyonlinelibrary.com]

particular set of views they wish to integrate to their own dashboard. This is a good spot for initiating a discussion about *goals* and *tasks*, and how visualization supports these, allowing students to be confronted with the need to document their reasoning processes that led to particular choices. At this point, the instructor should discuss the tradeoffs of given dashboards, and the fact that a dashboard that excels at supporting certain tasks will inevitably be less good at supporting some other tasks. An important lesson learned through this activity is that visualization design involves making compromises, and that different solutions can be equally valid, depending on where priorities are set.

Dashboards and coordinate views are powerful members of the world of exploratory data analysis methods, with dynamic query, selection, filtering and observation. However, communicating results which are dependent on dynamic choices can be difficult. The *communication of data insights* requires the user to curate one or a collection of visual representations of the data that best illustrate the information in the data, often also accompanied with explanations in writing that make it clear to the intended audience what the important information is. Thus, having explored the possible presentations via Tableau, we could then create a short dynamic document (for the students' hypothetical parents) comparing the properties of their selection options for a new computer, so that the pros and cons of each can be communicated, with the dynamic nature of the visualizations emphasized. We lead students to one useful chart which is

somewhat interpretable: a radar chart, comparing multiple attributes in a helpful way (see the right sub-plot in Figure 6). This figure can be interpreted to compare and contrast their selected options, as support for their short report.

We have now completed one iteration of exploration, developing several visualizations and sub-setting the dimensions down to useful attributes. The next step is to develop alternative visualizations, to further illustrate that the same data can be presented in different ways, revealing (and hiding) details equally. An example of what might be useful is to *encode* the data in a different way, for example, using a parallel coordinates plot. The radar chart (Figure 6) creates a "shape" for each computer in the data set which can be compared, while parallel coordinates plots (Figure 7) more clearly illustrate which item is at the top or bottom for each characteristic (or attribute, or feature).

Having built visualizations and created a rough report, we can then manipulate these frameworks, applying changes to color models, rendering schemes, and scaling functions. Lines, axes, and fills can be changed in the radar chart, inspiring students' creativity and encouraging them in the pursuit of understood visualizations, while still keeping the bulk of the implementation details reserved. In addition, labels, and legends should be added as appropriate.

This activity progresses students' ability to interpret existing code as a sequence of instructions to process and create a visualization from the data given as input to

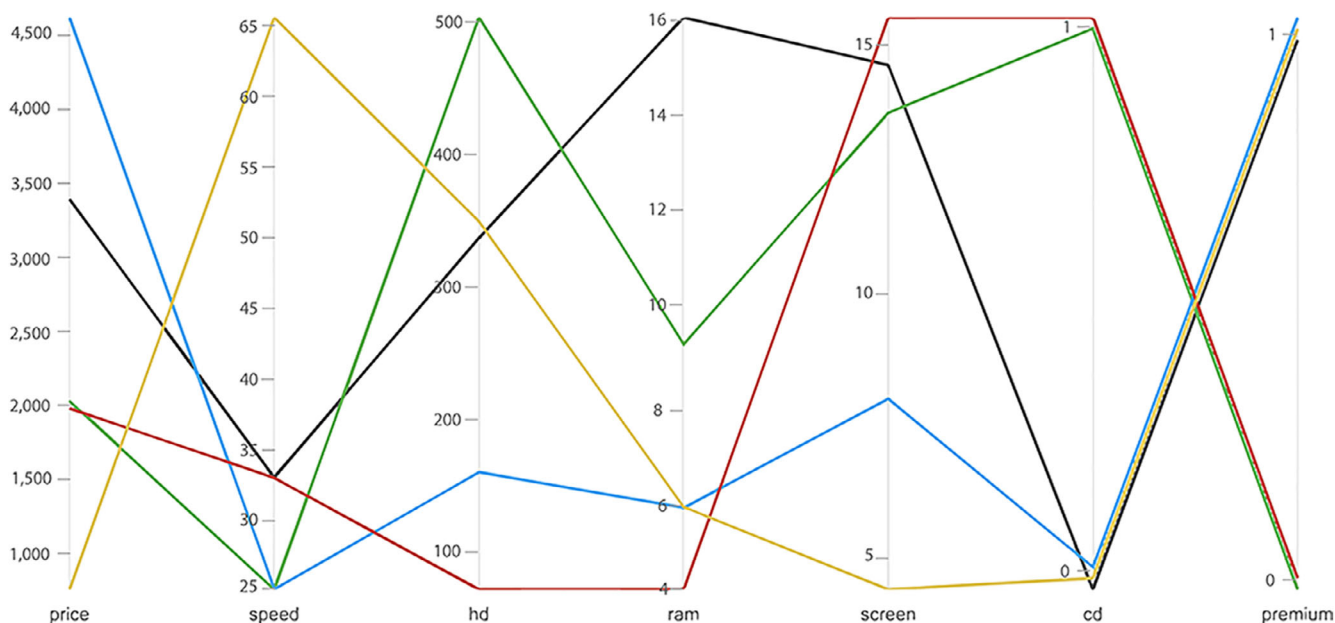


FIGURE 7 Parallel coordinates plot showing the characteristics of all five selected computers [Colour figure can be viewed at wileyonlinelibrary.com]

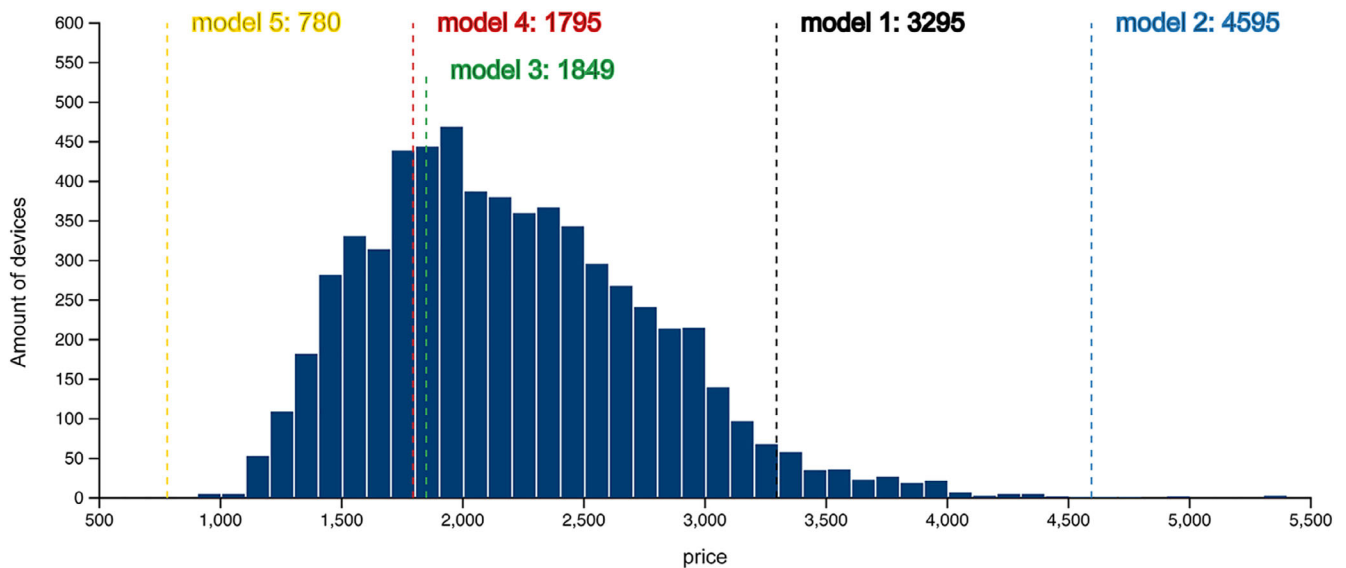


FIGURE 8 An improved histogram displaying the number of computer models binned, customized to contextualize the price of the five selected models compared to the whole collection of computer models [Colour figure can be viewed at wileyonlinelibrary.com]

these functions and develop their ability to refer to the documentation of functions to identify what each function does, and what are each of its parameters. Our goal is to emphasize that, besides the actual data set that is given as an input, a function often encompasses other input parameters that allow them to customize a more general concept, as for instance, “plot.”

We now suppose that our “client,” our parents, respond (regardless of completeness of exploration from each individual student) that they are concerned that we have selected expensive computers based on brand rather than making a balanced choice across capabilities and price. In other words, the client wants more work. To counter this, we could situate the selected options within the larger picture of the broad data set.

We could add a distribution of attribute values across the complete data set using a set of histograms, one for each variable, *repurposing code* by splicing together code snippets from other types of charts, such as the histograms already present in the same workbook. The aim is to show that the price is reasonable while achieving high values on the selected target attributes. For this we *calculate derived measures* from the values. The new view annotates the percentiles (including the median) of each attribute, and is demonstrated through *adding standard interactions*: items selected on the radar chart will be highlighted in the summary histograms; bars in the distribution will select filter ranges on the radar chart. An example of this annotation is shown in Figure 8, locating five computer models on the overall histogram of price.

Returning to the report aspect, we would finish our report by completing explanatory text blocks to help the

rationale of our explorations and selected visualizations, aimed at explaining to the clients (parents) how the exploration works, and how it can be manipulated dynamically (as it is not static!).

5 | CONCLUSION

In 2010 as one of the dimensions to ensuring continued and broader impact of statistical sciences, Nolan and Temple Lang [27] called for statistics educators to expand the use of computation in statistics curricula at all levels. They identified three key components to the integration of computing more broadly in statistics: (a) broaden statistical computing; (b) deepen computational reasoning and literacy; and (c) compute with data in the practice of statistics. Ten years on, the popularization of data science, and the corresponding need for more integration of computing and statistics in the pursuit of learning from data has brought focus and urgency to this call, and gave impetus for data science, including these components, to be brought to more students and to start earlier than post-secondary school. Statistical computing is necessary and can no longer be ignored for any reasonable approach to learning from data, and statistical thinking and computational literacy need to be partnered in all educational efforts to develop reasoning from data. The IDSSP curriculum frameworks have all been designed with (a) and (b) as implicit goals in an approach that allows development of curricula in, and foundations for, learning data science to be accessible at senior school and introductory tertiary levels to a wide range of students. All aspects of the IDSSP frameworks are designed with the

motivation of gaining skill in learning from data, in the structure of the cycle of data investigations, including formulating a problem, acquiring, exploring and analyzing the data, and communicating the results. For most areas existing software should suffice; for others new software may need to be written to ensure sufficient accessibility. But we believe this is all perfectly doable using the software tools we already have (eg, in the R and Python ecosystems) and adapting software models already in existence (including notebooks and IDEs).

The type and extent of use of technology in learning statistics at both school and tertiary levels has been a challenge for at least 30 years, not least because of the speed of technological development in the *practice* of statistics together with the rapid expansion of the world of learning from data to encompass more disciplines and increasingly complex data sets and contexts. Curriculum frameworks must now consider the explicit inclusion of computation from almost the very start. The IDSSP frameworks contribute to this by bringing together statisticians and computer scientists in outlining paths for how we might integrate these components in a soundly foundational way in secondary school or introductory tertiary for a broad range of students. As long advocated in developing statistical thinking, interesting data sets in a variety of areas can engage the students and motivate them to learn technical material when needed, with need-driven learning leading to students obtaining technical and computational skills “just in time,” as their motivation and interest leads them to understand and communicate the information that can be learned from the data.

The advent of interactive visualization tools and literate programming frameworks has the potential to change the restriction of “writing code” being a gatekeeper, and allow students with varying experiences and diverse interests to engage in and be able to *do* basic data science. This is fundamentally a question of accessibility, and our hope is that teaching computational skills through stealth will allow greater access to students from a broad range of backgrounds and interests. Encouraging agency in learners has tremendously compounded benefits in engagement and retention, and ensuring breadth in the academic skill, background and interest of data science students will encourage the growth of the data-driven questioning framework in disparate fields, and eventually, if perpetuated, across all of society. Long-term, this inclusivity may also help our professions tackle fundamental questions like algorithmic bias which are presenting themselves in our modern, data-driven economy.

Integration of statistical and computational skills requires scaffolding for both. Much work has been done over many decades in teaching statistical thinking and

data investigations, with accompanying educational research. Here, we advocate that computation be treated as a scaffolded, introduction-via-stealth activity, in order to achieve the integration of statistical and computational skills which are at the heart of statistics and data science. We intentionally refrain from stating “a right way to do computation.” Quite different approaches can be used to achieve very similar ends, and suit different teachers and different types of students. We have touched on some possible computational approaches in this paper. But whatever computational approaches are chosen, the desired ends cannot be met unless the students being taught find their computational experiences unthreatening, interesting, empowering and enjoyable.

ORCID

Wesley Burr  <https://orcid.org/0000-0002-2058-1899>

Christopher Collins  <https://orcid.org/0000-0002-4520-7000>

Alison L Gibbs  <https://orcid.org/0000-0002-0408-3435>

Chris J Wild  <https://orcid.org/0000-0003-0115-6448>

REFERENCES

1. S. A. Ambrose, M. W. Bridges, M. DiPietro, M. C. Lovett, and M. K. Norman, *How Learning Works: Seven Research-Based Principles for Smart Teaching*, Wiley, San Francisco, CA, 2010.
2. American Statistical Association Undergraduate Guidelines Workgroup (2014), *Curriculum Guidelines for Undergraduate Programs in Statistical Science*, <http://www.amstat.org/asa/files/pdfs/EDU-guidelines2014-11-15.pdf>.
3. F. J. Anscombe, *Graphs in statistical analysis*, *Am Stat* **27** (1973), 17–21. <https://doi.org/10.1080/00031305.1973.10478966>.
4. Census at School NZ. (2020), Investigate time series data, <https://new.censusatschool.org.nz/resources/3-8/>.
5. M. Çetinkaya-Rundel and V. Ellison, *A fresh look at introductory data science*, *J Stat Data Sci Educ* **29** (2021), S16–S26. <https://doi.org/10.1080/10691898.2020.1804497>.
6. C. Connolly, E. Murphy, and S. Moore, *Programming anxiety amongst computing students—a key in the retention debate?* *IEEE Trans Educ* **52** (2009), 52–56. <https://doi.org/10.1109/TE.2008.917193>.
7. N. Cowan, *Processing limits of selective attention and working memory: potential implications for interpreting*, *Interpreting* **5** (2000), 117–146. <https://doi.org/10.1075/intp.5.2.05cow>.
8. N. Cowan, *The magical number 4 in short-term memory: a reconsideration of mental storage capacity*, *Behav Brain Sci* **24** (2001), 87–114. <https://doi.org/10.1017/S0140525X01003922>.
9. D. Donoho, *50 years of data science*, *J Comput Graph Stat* **26** (4) (2017), 745–766. <https://doi.org/10.1080/10618600.2017.1384734>.
10. Elliott T, Soh YH, Barnett D. iNZightPlots: graphical tools for exploring data with iNZight. R package version 2.13; 2021, <https://CRAN.R-project.org/package=iNZightPlots>.
11. Elliott T, Wild C, Barnett D, Sporle A. iNZight: A Graphical User Interface for Data Visualisation and Analysis through R; 2021 https://inzight.nz/papers/2021_jss.pdf.

12. Elliott T, Zeng J, Potter S, Banks D, Kuper M, Zhang D. iNZightTS: Time series for 'iNZight'. R package version 1.5.7; 2020. <https://CRAN.R-project.org/package=iNZightTS>.
13. J. D. Fekete, J. J. Van Wijk, J. T. Stasko, and C. North, *The value of information visualization*, in *Information Visualization*, Vol **4950**, A. Kerren, J. T. Stasko, J. D. Fekete, and C. North, Eds., Springer, Berlin, Heidelberg, 2008, 1–18.
14. Fergusson A. Popularity contest [Blog post]; 2020. <https://askgoodquestions.blog/2020/08/17/59-popularity-contest/>.
15. A. Fergusson and M. Pfannkuch, *Introducing teachers who use GUI-driven tools for the randomization test to code-driven tools*, Math Think Learn (2021). <https://doi.org/10.1080/10986065.2021.1922856>.
16. A. Fergusson and C. J. Wild, *On traversing the data landscape: Introducing APIs to data-science students*, Teach Stat **43** (2021), S71–S83. SII. DOI: 10.1111/test.12266
17. J. Fox, *Using the R Commander: A Point-and-Click Interface for R*, Chapman and Hall, New York, 2017.
18. E. W. Gibson, *Leadership in Statistics: increasing our value and visibility*, Am Stat **73**(2) (2019), 109–116. <https://doi.org/10.1080/00031305.2017.1336484>.
19. A. Gunn, *Embedding quantitative methods by stealth in political science: developing a pedagogy for psephology*, Teach Public Admin **35** (2017), 301–320. <https://doi.org/10.1177/0144739417708838>.
20. E. Hare and A. Kaplan, *Designing modular software: a case study in introductory statistics*, J Comput Graph Stat **26** (2017), 493–500. <https://doi.org/10.1080/10618600.2016.1276839>.
21. Hare E, Kaplan A. intRo: Shiny-based statistics learning application; 2019, <https://github.com/gammarama/intRo>.
22. IDSSP Curriculum Team. Curriculum Frameworks for Introductory Data Science; 2019, http://idssp.org/files/IDSSP_Frameworks_1.0.pdf
23. D. E. Knuth, *Literate programming*, Comput J **27** (1984), 97–111. <https://doi.org/10.1093/comjnl/27.2.97>.
24. S. Luttenberger, S. Wimmer, and M. Paechter, *Spotlight on math anxiety*, Psychol Res Behav Manag **11** (2018), 311–322. <https://doi.org/10.2147/PRBM.S141421>.
25. Muenchen RA. R Graphical User Interface Comparison; 2020, <http://r4stats.com/articles/software-reviews/r-gui-comparison/>, Accessed June 25, 2020.
26. National Academies of Sciences, Engineering, and Medicine, *How People Learn II: Learners, Contexts, and Cultures*, The National Academies Press, Washington, DC, 2018. <https://doi.org/10.17226/24783>.
27. D. Nolan and D. Temple Lang, *Computing in the statistics curricula*, Am Stat **64** (2010), 97–107. <https://doi.org/10.1198/tast.2010.09132>.
28. C. Primi and F. Chiesi, *The role of mathematics anxiety and statistics anxiety in learning statistics*, in *Looking Back, Looking Forward. Proceedings of the Tenth International Conference on Teaching Statistics (ICOTS10, July, 2018)*, M. A. Sorto, A. White, and L. Guyot, Eds., International Statistical Institute, Kyoto, Japan, 2018.
29. R. Pruijm, D. Kaplan, and N. Horton, *The mosaic package: Helping students to think with data using R*, R J **9** (2017), 77–102. <https://doi.org/10.32614/RJ-2017-024>.
30. R Core Team, *R: A language and environment for statistical computing*, R Foundation for Statistical Computing, Vienna, Austria, 2020 <https://www.R-project.org/>.
31. L. A. Sharp, *Stealth learning: unexpected learning opportunities through games*, J Instruct Res **1** (2012), 42–48.
32. Shreve, J. (2005), Let the Games Begin. Video Games, Once Confiscated in Class, Are Now a Key Teaching Tool If They're Done Right, George Lucas Educational Foundation. <https://www.edutopia.org/video-games-classroom>.
33. H. Wickham, *Tidy data*, J Stat Softw **59**(10) (2014), 1–23. <https://doi.org/10.18637/jss.v059.i10>.
34. Wickham H. Tidyverse: easily install and load the 'Tidyverse', R Package version; 2019 <https://CRAN.R-project.org/package=tidyverse>.
35. H. Wickham and G. Grolemund, *R for Data Science: import, tidy, transform, visualize, and model data*, O'Reilly Media, Sebastopol, California, 2016. <https://r4ds.had.co.nz/>.
36. H. Wickham, *ggplot2: Elegant Graphics for Data Analysis*, Springer-Verlag, New York, 2016.
37. C. J. Wild, T. Elliott, and A. Sporle, *On democratizing data science: some iNZights into empowering the many*, Harvard Data Sci Rev **3**(2) (2021). <https://hdsr.mitpress.mit.edu/pub/8fxt1zop/release/1>.
38. C. J. Wild and J. Ridgway, *Civic Statistics and iNZight; illustrations of some design principles for educational software*, in *Statistics for Empowerment and Social Engagement – Teaching Civic Statistics to Develop Informed Citizens*, J. Ridgway, Ed., Springer, Berlin, 2021.
39. K. Yevseyeva and M. Towhidnejad, *Work in Progress: Teaching Computational Thinking in Middle and High School*, Proc Front Educ Conf (2012). 1–2. <https://doi.org/10.1109/FIE.2012.6462487>.

How to cite this article: W. Burr, F. Chevalier, C. Collins, A. L. Gibbs, R. Ng, and C. J. Wild, *Computational skills by stealth in introductory data science teaching*, Teaching Statistics **43** (2021), S34–S51. <https://doi.org/10.1111/test.12277>