Teaching and Educational Methods

Data Visualization in Applied Economics Instruction and Outreach

Jared Hutchins[a] and Andrew J. Van Leuven[b]

[a]University of Illinois, Urbana-Champaign, [b]University of Vermont

Abstract

This article highlights the critical role of data visualization in applied economics education and outreach. We first outline some general principles for teaching graph literacy and data visualization principles in and out of the classroom. We then discuss the mechanics of visualizing data—collection, preparation, and visualization—with an emphasis on how instructors can teach each step using the R and/or Python statistical environments. We ultimately contend that the requisite skills for successful data visualization are indispensable for students trained in today's agricultural and applied economics programs to communicate their research effectively.

1 Introduction

As the agriculture sector itself has become increasingly reliant on data collection and analysis (Elliott and Elliott 2020b), so have agricultural and applied economics researchers, over time, had to enhance their ability to work with and convey the broad implications of data. This need for data science skills arises not only from the technical demands of cutting-edge big data analysis but also from the necessity of clearly and responsibly communicating research findings to diverse audiences. With several decades of advancements in statistical computing and the proliferation of open-source software, there are now very few reasons why budding applied economists should not leave their studies with a solid understanding of the basic principles of data visualization and how to execute them.

In his introductory text on data visualization, Healy (2019) implores researchers to simply "look at your data" before attempting to communicate any corresponding ideas or findings. This paper focuses on two complementary aspects of how instructors in agricultural and applied economics can better teach audiences to look at their data. First and foremost, every student, researcher, or individual is, at one point or another, the audience of a data visualization. Thus, educators also have a responsibility to teach students how to properly *consume* visualized data. While applied researchers are often well aware of how they can—intentionally or inadvertently—"lie with statistics" (Huff 1954), students and stakeholders without a statistical background are often unaware of how data visualizations might mislead. As such, the principles of graph literacy are just as important as those of visualization itself. Moreover, given the role many applied economists play as state specialists through their universities' Cooperative Extension programs, it is crucial that they use their expertise to teach graph literacy to stakeholders in agriculture and rural development.

Second is the process of visualization itself. Researchers have virtually endless options for conveying findings via their data, and there are numerous pitfalls that can render a visualization ineffective at best and deceptive at worst. In addition to their role in the classroom, applied economists often engage in outreach activities that help translate research into actionable insights. This includes working with Cooperative Extension programs to communicate complex information in ways that

support decision-making in agricultural and rural communities. Ensuring that stakeholders can both interpret and create effective data visualizations is essential for bridging the gap between research and real-world applications.

This paper outlines principles and techniques for teaching data visualization and graph literacy to advanced students in agricultural and applied economics as well as outreach audiences. After discussing visualization principles for students and stakeholders, we propose a framework for teaching data visualization in R or Python and then discuss guidelines and best practices for applied economics instructors to help students transform their raw data into effective storytelling for their research. Our framework for teaching data visualization is most suited for courses designed for students with experience in a coding language and a beginner to intermediate grasp of math and statistics. For learning the basics of coding, there are a number of free, online resources on data visualization and coding that can complement this framework or be taught as a prerequisite.[1] As data visualization has become an indispensable part of a researcher's toolkit, we believe that these skills are now indispensable for any applied economics program.

Our discussion of data visualization principles joins a few other articles discussing the role of data analytics in agricultural economics and agribusiness education. Jin et al. (2024) and Elliot and Elliot (2020b) discuss data analysis and visualization exercises and lessons learned from their implementation. Minegishi and Mieno (2020) and Elliot and Elliot (2020a) discuss resources in R for analysis in applied economics and Extension education. Our paper builds on this work by focusing on the principles of teaching data visualization and by providing practical resources for teaching them in the classroom.

The remainder of this paper is structured as follows. Section 2 discusses teaching students and learners of all backgrounds how to interpret and critically consume visualized data, particularly from an engaged outreach context (i.e., Cooperative Extension). Section 3 outlines a set of principles for instructing students on both the mechanical processes of visualizing data as well as the aesthetic, practical, and ethical considerations that contribute to quality visualizations. Section 4 introduces the *data visualization pipeline*, which includes the three key stages of data collection, processing, and visualization. We conclude with a reproducible code example illustrating the stages of the pipeline in R and Python.

2 Teaching Graph Literacy to Student and Outreach Audiences

While not all students or stakeholders will regularly produce data visualizations, they are all very likely to be regular consumers of data visualizations. Thus, teaching graph literacy to both university and nonuniversity audiences is important to the mission of many agricultural and applied economics departments. University faculty, particularly in agricultural and applied economics departments, are often called upon to extend their expertise beyond research and classroom teaching to address pressing local issues. In the United States, these departments are uniquely positioned within academia, frequently leveraging Cooperative Extension as a key outreach platform. In outreach settings, graph literacy is crucial for empowering stakeholders to interpret data visualizations accurately. By teaching stakeholders how to identify misleading visualizations and understand the context of data, we enhance their ability to make informed decisions based on research findings. In this section, we discuss some principles of graph literacy that can be used for outreach education, focusing primarily on correctly interpreting data visualizations and critiquing poor and ineffective ones.

---

[1] For Python, Jake VanderPlas's book *Python Data Science Handbook*, Jared Hutchins's course Data Science for Applied Economics, and Matthew Brett's course Coding for Data are good resources that are publicly available. For R, Julian Ludwig's course Data Analysis for Economic Research and Nick Hagerty's course Advanced Data Analytics in Economics also have lecture material that is made publicly available.

Applied economists doing outreach frequently engage in settings where they must communicate data-driven findings or conclusions—on topics ranging from crop yields to farm finance and community economic development—to nonacademic audiences. While straightforward tables and raw figures can convey basic information, data visualizations offer a more intuitive and impactful way to communicate complex findings. Visual tools such as bar charts, line graphs, and scatter plots are central to Extension work, allowing experts to readily convey patterns, comparisons, and relationships in ways that resonate with broader audiences.

These experts, however, have an important responsibility not just to present data but also to equip stakeholders with the skills to interpret future visualizations—whether USDA charts, US Census maps, or other graphical representations of key trends. This empowers community members to independently analyze and apply data long after the expert has left the room.

We strongly encourage all applied economists to incorporate an educational component on graph literacy into their outreach efforts. While instructional approaches will differ based on context and learners' knowledge levels, the following four questions can serve as a starting point for developing graph literacy among outreach audiences:

- *What do you see?* Ask learners to observe the figure closely and summarize the main takeaway in one sentence. This exercise encourages them to focus on the core message of the visualization and extract a clear, understandable insight from complex data.

- *Who created the visualization?* Emphasize the importance of understanding the source and context of the figure, especially if it might convey a political or advocacy-based message. Recognizing the creator often reveals the purpose behind the figure's design.

- *Are you being tricked?* Introduce learners to common techniques that can make visualizations deceptive. Even a basic awareness of these tactics can empower them to critically evaluate and interpret what they see.

- *What is missing?* Encourage learners to consider what data or context might have been left out of the visualization. Doing so helps them think critically about possible gaps, assumptions, or alternative perspectives that could change the interpretation.

Though not comprehensive, these principles provide a solid foundation to educate nonacademic stakeholders—farmers, small business owners, local government officials, etc.—on how to more effectively discern insights communicated through data visualizations. By fostering graph literacy, applied economists can enhance the long-term impact of their outreach, enabling stakeholders to make more informed decisions based on data in their everyday operations and planning.

3 Teaching Data Visualization Principles in the Classroom

Beyond simply knowing how to write code to produce visualizations, students need to understand the principles of good data visualization and when to use the tools they have. In this section, we focus on data visualization principles that are important for effective communication. We first discuss how to teach students to choose the appropriate type of graph and then transition to discussing data visualization principles and pitfalls to discuss with students. Rather than discuss these principles in detail, we give a broad outline of some principles and point the reader to more detailed discussions such as Wilke (2019) and Tufte (2001) for general principles and Healy (2019) and Kabacoff (2024) for R specifically.) Kabacoff (2024) and Wilke (2019) may be particularly useful to instructors and students because they are available as free e-books online under Creative Commons licenses.
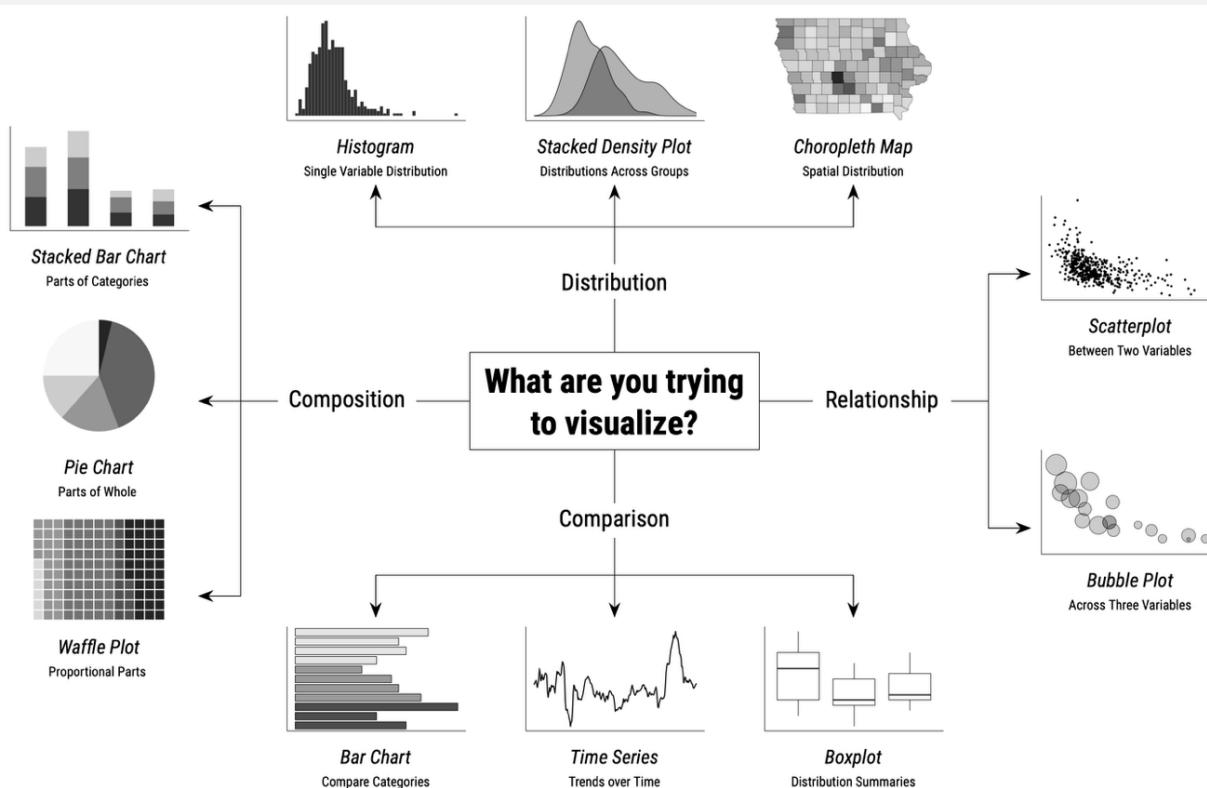
Figure 1. Data Visualization Decision Tree

Source: Adapted from Abela (2009).

3.1 The Data Visualization Decision Tree

The effectiveness of any given data visualization hinges on a single question: "What are you trying to show?" Figure 1 shows a decision tree adapted from Abela (2009) with four main branches—distribution, composition, relationship, and comparison—each pertaining to a different tactic for visualizing data. We briefly discuss each branch below and a fifth approach, geography, which includes maps and visualizations that are uniquely spatial.

3.1.1 Distribution

Visualizing the distribution of data entails illustrating how data points are dispersed across different values. Common approaches include histograms, box plots, and kernel densities, all of which help observers grasp the frequency of data points within certain ranges. For instance, consider a simple dataset with the number of active crop insurance payments by county. Descriptive statistics, such as the mean and standard deviation, can provide a general sense of how the data is distributed. However, a visualization of the distribution of policy payments can help uncover patterns like skewness or outliers, which basic descriptive statistics may not indicate.

3.1.2 Composition

Pie charts are often regarded as the go-to method for showing how a whole is divided into parts, but many alternatives for visualizing data composition are preferable to pie charts. Waffle plots, for instance, serve the same purpose but are generally considered more readable, as the human brain struggles to compare angles and slices of slightly different sizes (Van den Eeckhout, 2021). While both plots in Figure

2 use data from the latest Census of Agriculture to illustrate how cropland is allocated among the top seven field crop commodities in Michigan, the waffle plot at right communicates the relative proportions much more clearly.

Figure 2. Side-by-side comparison of pie chart and waffle plot

Source: USDA NASS (2024).
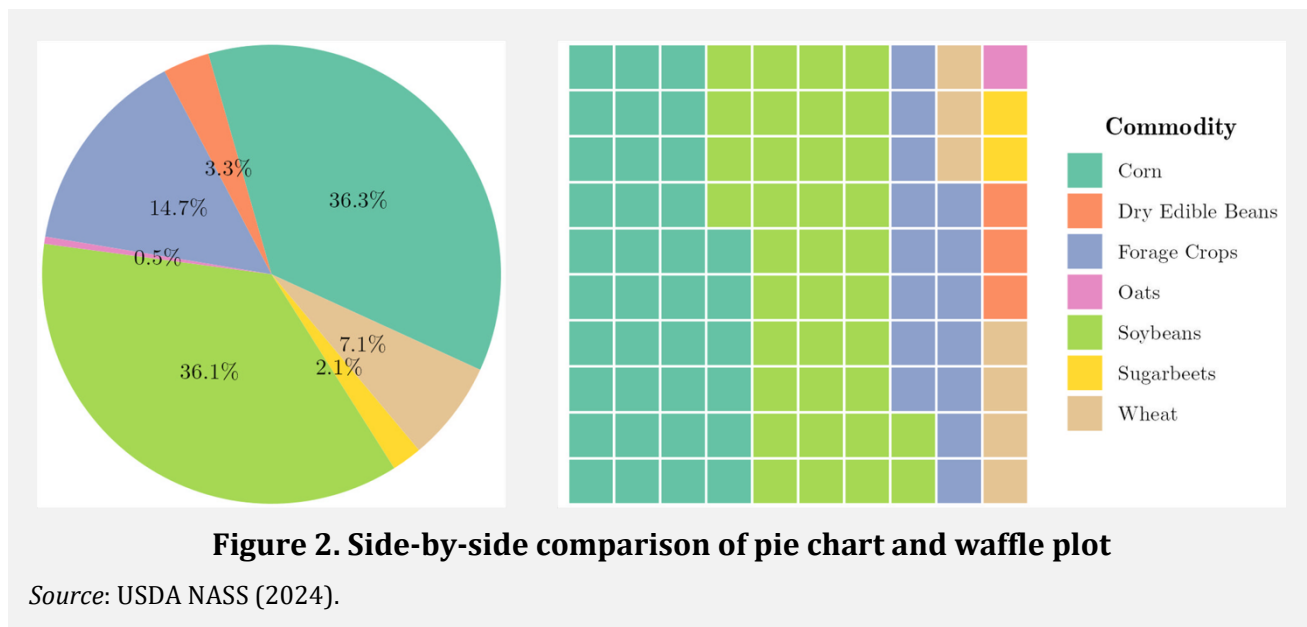
3.1.3 Relationship
Visualizations like the scatter plot, with two variables plotted along vertical and horizontal axes, are best for exploring how multiple variables interact. Additional variables can be represented in a scatter plot by allowing the size, shape, color, and transparency of each point to vary. However, too many variables might obscure rather than clarify, as a simple bivariate scatterplot is already a very efficient way to illustrate correlations or associations between two or more variables.
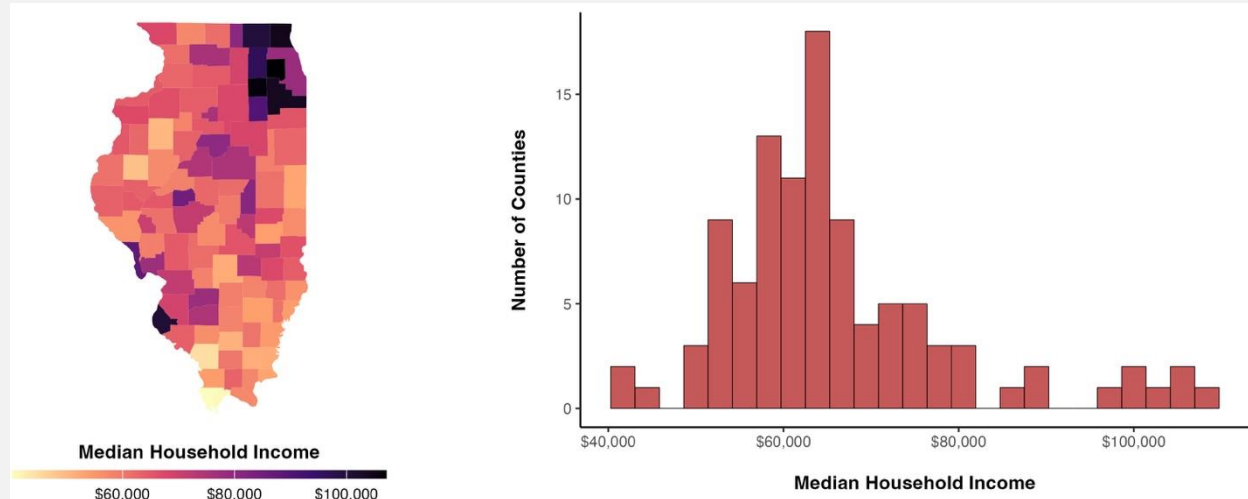
3.1.4 Comparison
Comparison visualizations are useful for comparing different datasets or categories, offering a clear way to visually assess values across various groups or monitor changes over time. Comparison visualizations—like bar charts and line charts—make spotting trends, patterns, and differences within the data easier.

3.1.5 Geography
It could be argued that spatial data visualization is not a separate category from the approaches listed above. However, while it is true that maps may be created to show spatial distributions, compositions, relationships, and comparisons, maps are a wholly distinct type of data visualization in that they pertain to real, physical space. As demonstrated in Figure 3, a basic histogram (at right)—using Census data to visualize, for example, the distribution of median household incomes across all counties in Illinois— cannot speak to the question of *where* this distribution takes place. In comparison, the choropleth map (at left) is not as helpful at visualizing the numeric distribution of income, but it provides an immediate indication of which counties are home to the highest incomes. Spatial data visualization is part of a larger field of knowledge, geographic information systems (GIS). Maps can be a powerful tool for visualizing data with a crucial geographic component.

Figure 3. Side-by-side comparison of choropleth map and histogram

Source: US Census Bureau (2023).

## 3.2 Teaching Visualization Principles

It has often been said that where there are many treatments there is no cure. The aphorism holds true for data visualization: there is no one, universally agreed upon set of principles that all students should learn. Instead, many resources present their own specific philosophy of data visualization, including a list of principles and usually a list of "sins" that should be avoided. This subsection outlines some popular principles from Tufte (2001) and Healy (2019) that can engage students and help them think more critically about data visualization.

One seminal text in data visualization is *The Visual Display of Quantitative Information* by Tufte (2001), a statistician at Princeton University. The volume contains many examples of good and bad data visualization, both historical and current, which he uses to illustrate his principles of graphical design. In the 2001 edition, these five principles are

- above all else, show the data;
- maximize the data-ink ratio;
- erase nondata-ink;
- erase redundant data-ink;
- revise and edit.

To understand these principles, students must first be introduced to some of his concepts, including data-ink, which refers to parts of the graph depicting the data; nondata-ink, which refers to all other parts of the graph; and the data-ink ratio, which refers to what percentage of the graph's "ink" is actually depicting the data. Tufte's approach is considerably minimalist and particularly unforgiving to parts of graphics that do not depict actual data points. For example, Tufte (2001) devotes Chapter 5 to what he calls "chartjunk," for example hatching of bar graphs, grids, and needless ornamentation. Some may find Tufte's dogged removal of nondata chart elements to be extreme, but it serves the pedagogical purpose of encouraging students to be intentional about what they put in their graphs. One class activity can be displaying a plot from the book and working through which elements Tufte removes (see Figure 4 for an example with a box plot inspired by the book). The exercise can then be repeated with a new chart in groups or collaboratively as a class.
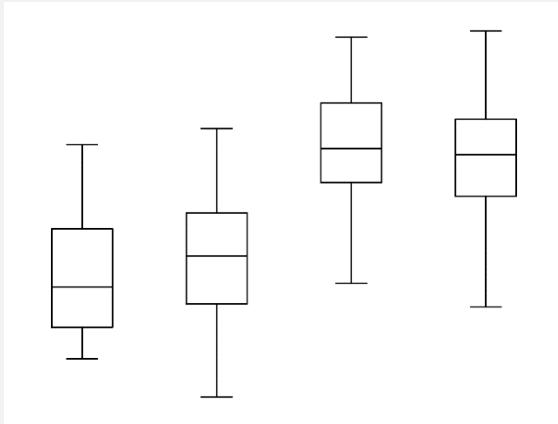
4(a) Example box plot   **4(b) Same box plot with no nondata-ink**
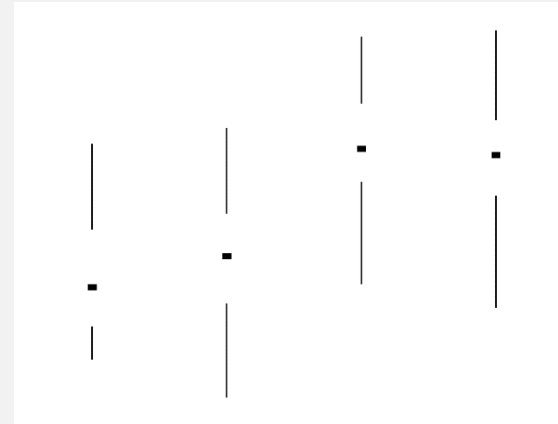
Figure 4. Box plot with Tufte-style edits

Notes: See Tufte (2001).

Arguably, one of Tufte's most enduring principles is that "the representation of numbers, as physically measured on the surface of the graphic itself, should be directly proportional to the numerical quantities represented" (Tufte 2001, p. 56). A version of this principle is often referred to as the *Principle of Proportional Ink*, a name attributed to Bergstrom and West (2016). Simply put, if a data point takes up a lot of space on the graph, it should be because the data point has a large value. Chapter 2 of Tufte (2001), "Graphical Integrity," contains multiple violations of this principle. Along with the data-ink ratio, Tufte calculates the "lie factor" of a graph as the extent to which the space devoted to a data point in the graph over- or understates the true value. From these examples, Tufte derives more design principles:

(1) Show data variation, not design variation (i.e., parts of the graph not depicting data).

(2) The number of information-carrying dimensions depicted should not exceed the number of dimensions in the data.

- Corollary: never use more than one dimension to depict one-dimensional data.

(3) Graphics must not quote data out of context.

A different but related set of visualization principles is articulated in Wilke (2019). Along with the Principle of Proportional Ink, Wilke gives guidance on color selection and aesthetics that can complement Tufte. Wilke distinguishes between three uses of color in a visualization in Chapter 4, "Color Scales":

- color as a tool to distinguish
- color to represent data values
- color as a tool to highlight

In the first case, the color scale is discrete and helps distinguish between different categories. In the second case, the color scale is continuous and helps distinguish a continuous variable. In the third case, a color is used sparingly to draw attention to a specific data point or points.

Misapplication of these principles leads to some pitfalls that Wilke (2019) discusses in Chapter 19, "Common Pitfalls of Color Use." For using color to distinguish, Wilke suggests only distinguishing three to

5(a) Nonmonotonic colormap (name: "Jet")
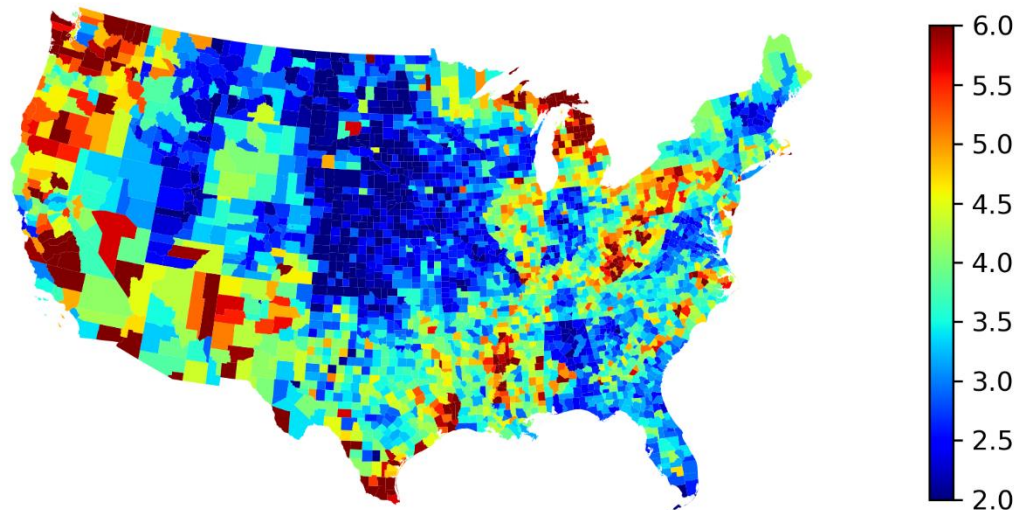
5(b) Monotonic colormap (name: "Blues")
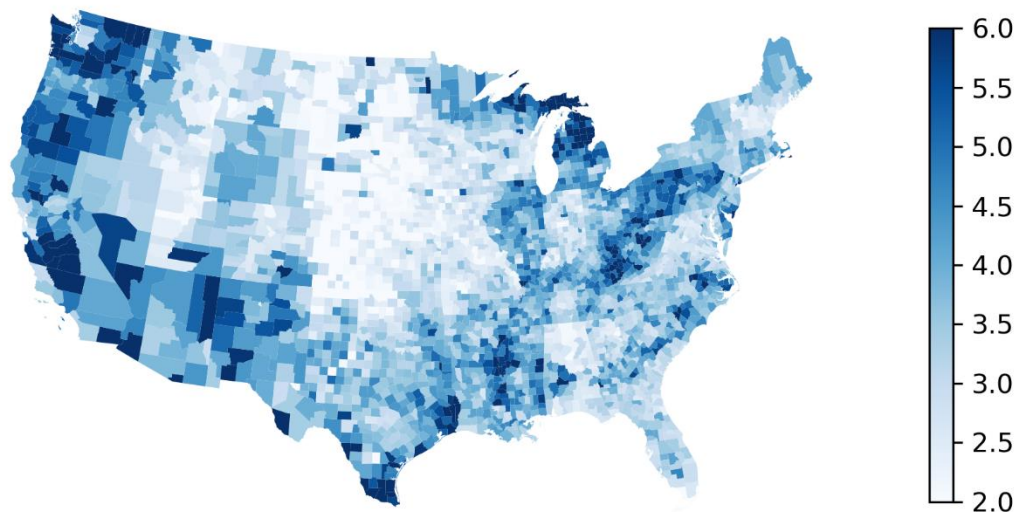
Figure 5. County unemployment rates in 2022

Source: Bureau of Labor Statistics.

five categories at most. Otherwise, it is difficult for the reader to keep track of differences. Another pitfall is "color for the sake of color," where data points are colored but the colors hold no meaning. Wilke also discusses the use of nonmonotonic color maps for monotonic values. Figure 5 shows county-level US unemployment rates for 2022 from the Bureau of Labor Statistics using two color map options: rainbow (called "Jet") and blues (called "Blues"). The unemployment rate is a monotonic continuous variable, but the rainbow colormap is a nonmonotonic colormap. The effect of using this colormap is that it invites

grouping of counties by the similarities of their colors: the eye wants to group red and orange counties together, as it does for green counties and blue counties. This leads to grouping counties in the 2–3.5 unemployment rate range as well as those in the 3.5–4.5 and 4.5–6 ranges. However, there is no intuitive reason for red to be "higher" than blue, and so the comparisons are more difficult if the objective is to compare one county to any other. When using a "blues" colormap, which is monotonic, the comparison between high and low unemployment rate counties is much easier and more intuitive. This is an example where a nonmonotonic colormap such as a rainbow is inappropriate for a continuous variable.[2]

Another pitfall that Wilke points out is the inappropriate use of a diverging color map. A red-blue diverging colormap, for example, can be useful for drawing attention to values that deviate from a midpoint or to show growth rates by making all data points with a negative growth rate red and those with a positive growth rate blue. Figure 6 charts change in county-level US unemployment rates between 2021 and 2022. Between these two years, the unemployment rate fell in 96 percent of counties.

In Figure 6(a), the top and bottom of the color bar are 5 and –5 percentage points, making the diverging point 0. In Figure 6(b), the top of the color bar is shifted to 3 to make –1 the diverging point instead. Simply changing the midpoint drastically changes the interpretation. The first plot appears to accurately reflect the fact that unemployment fell nearly everywhere, while the second makes it appear that it only fell in certain places. In fact, only 48 percent of counties experienced unemployment rate drops greater than 1, which makes roughly half of the counties red and half blue in the second figure. As in Figure 5, readers tend to group counties by color, and a simple change in the diverging point leads to significant changes in which values a reader will group as similar.

A final pitfall that Wilke points out is not using colormaps that are robust to color-vision deficiency. When publishing figures, it is important to understand the range of audience that will be reading it and use colors that can be distinguished by as many people as possible. Depending on the publication outlet, how the colors look in grayscale might also need to be considered. Correcting for color-vision deficiency and grayscale printing were two considerations that led to the creation of the colormap *viridis*, now the default in `matplotlib`, which can be interpreted regardless of color-vision deficiency and distinguished in grayscale (Smith and van der Walt 2015).

Furthermore, instructors who teach culturally diverse groups may also need to discuss what colors could connotate to different audiences. For example, the map in Figure 6 shows unemployment rates across the United States using red and blue colors. However, since red and blue are also colors associated with political parties in the United States, that colormap may inadvertently cause the map to have a political interpretation. This also extends to connotations of emotions or physical sensations that may be attached to certain colors in multiple contexts (e.g., red is hot and blue is cold) (see Adams and Osgood 1973; Madden et al. 2000).
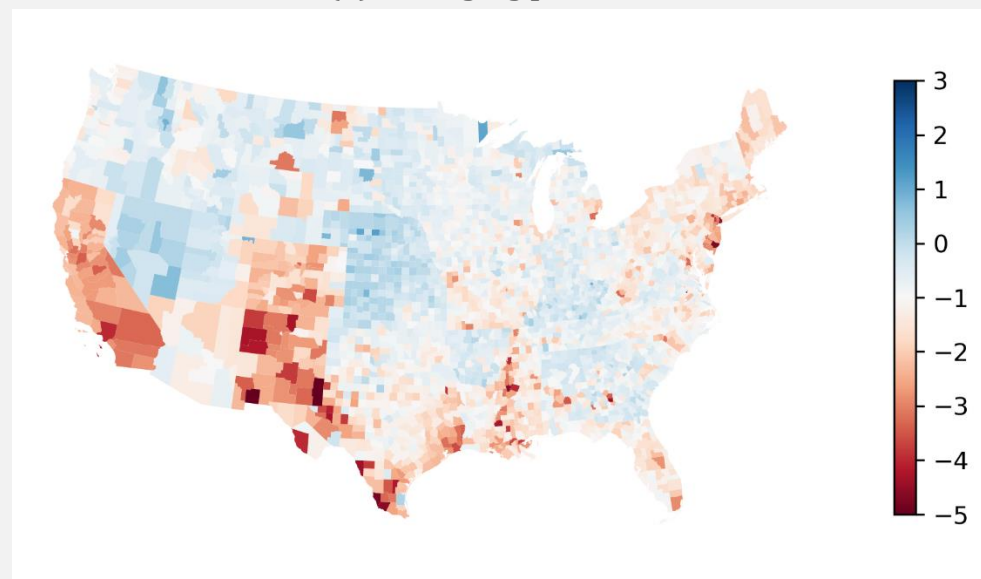
3.3 Teaching Data Visualization "Sins" in the Classroom

To reinforce both graph literacy and design principles for students and stakeholders, instructors may find it useful to teach using negative examples. Further applications of the above principles to real-life examples can help students spot some common "sins" committed by deceptive graphs, including

- truncating the y-axis to make small changes appear large,
- using changes in area to depict changes in a single variable (univariate data),
- using inconsistent axes and ticks to distort trends.

---

[2] The rainbow colormap is, in fact, a subject of constant debate. There continues to be a lively discussion about its use in data visualization, where earlier papers view it unfavorably (Rogowitz and Treinish 1998; Borland and Taylor 2007) and other, recent papers defend its use (Reda 2022; Ware et al. 2023). Engaging a class in this debate is another fun way to engage students to think critically about color in visualization.

6(a) Diverging point at 0
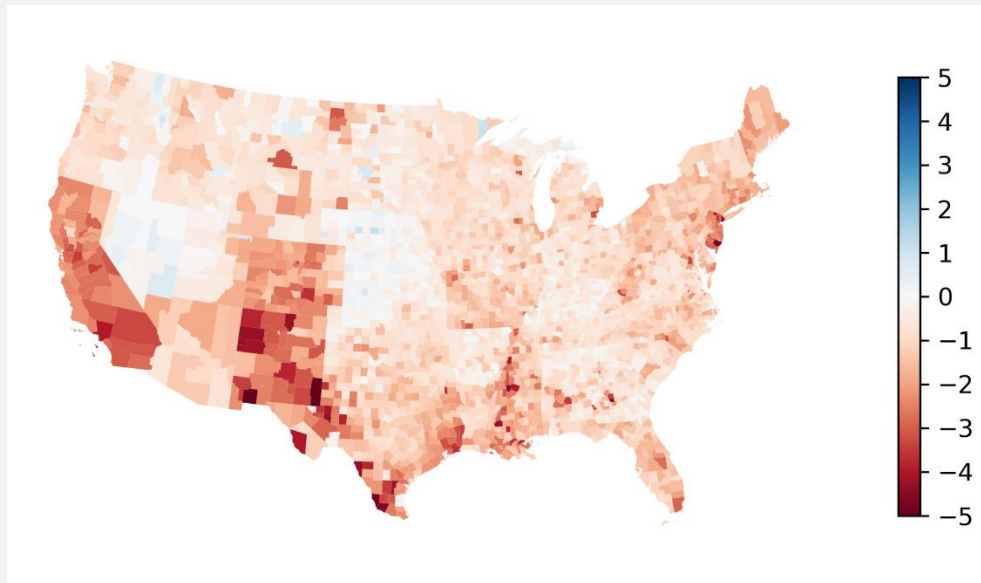
6(b) Diverging point at −1

Figure 6. Change in county-level US unemployment rates, 2021–2022

Source: Bureau of Labor Statistics

Rather than teaching what to do, it can often be helpful to help students understand what not to do by critiquing data visualizations that are of poor quality. Wilke (2019) offers one framework for critiquing visualizations using the terms

- *ugly*: having to do with aesthetic problems (e.g., distracting fonts or colors that are too bright);
- *bad*: having to with perception problems (e.g., a figure is hard to read);
- *wrong*: having problems of accuracy (e.g., an inconsistent x-axis scale).

Throughout his text, Wilke (2019) uses this framework to critique several visualizations of the same data to help readers compare visualization approaches. Similarly, Tufte (2001) critiques visualizations with his own, more minimalist principles and is especially critical of chartjunk. Tufte is especially critical of when excessive ink makes data points appear too big, a violation of the proportional ink principle.

Using the above applications and framework, we suggest two class activities that can be implemented to help students articulate and apply these principles. The first exercise is to ask students to submit poorly implemented data visualizations to be analyzed by the class. One recurring classroom activity used by an author of this paper is encouraging students to find data visualizations that have these issues. That author implements this assignment in their data science class and calls it "the hunt for the worst data visualization," encouraging students to submit visualizations throughout the semester. At the beginning of the semester, two student submissions of data visualizations are assessed and the worst one is considered the first winner. Each time there is a new submission, students vote on which one is worse, the new submission or the current winner, and the winner goes on to be judged against the next submission. At the end of the course, the final winning submission is judged against the previous year's winner.

The idea of this exercise is not simply to point out what is wrong with a visualization. Instead, for each submission, students should be prompted to articulate what exactly about the visualization is unsightly or misleading. After understanding its pitfalls, students should then be asked how they would improve it using the principles taught from the texts.

Another exercise is to compare different visualizations of the same data. In the above exercise, it may be the case that the data being visualized is of poor quality and distracts students from critiquing the design. Using the same data source but different visualizations focuses attention on the advantages and disadvantages of each approach. Wilke (2019) provides some pairs of graphs that can be used for this purpose; however, an instructor can also create their own examples using the decision tree described in Figure 1. For example, students can be shown a distribution of data visualized with first a histogram and then a kernel density. Unlike the "hunt for the worst data visualization," the objective here is less to point out pitfalls and more to emphasize what each visualization technique emphasizes in the data.

4 The Data Visualization Pipeline

In this section, we outline a concept useful for teaching data visualization and data science skills: the data visualization pipeline. The pipeline can be thought of as the process by which data is read in, processed, and analyzed to visualize it effectively (see Table 1). Each stage of the pipeline requires specific skills to be taught. The first stage of the data visualization pipeline is *data collection* via APIs, FTP, web scraping, or other means. After the data is obtained, the next stage is *data processing*, meaning cleaning and other necessary preparation for data analysis and/or visualization. Once the data is prepared, the last stage is *data visualization*, crafting the visualization or analysis to be used in research, outreach, or science communication. Teaching the pipeline can help students understand the steps needed in data visualization and motivates them to learn the coding skills, packages, and software needed in each step. Thinking of the process in these stages can also help students conceptualize the workflow and document their process with more clarity. For each stage of the pipeline, we describe what skills typically need to be taught at each stage.

While proprietary software (e.g., Stata, MATLAB) can also be used for data visualization, we emphasize teaching students to code using open-source software and packages for three reasons. First, open-source software is free to use without a license. This is especially important for students who may move on to jobs where they will not have access to licenses or to a company/institution that is not willing to pay for a license. Second, new tools are often available in the open-source community earlier than in

Table 1. The data visualization pipeline

Stage of Pipeline	Python	R
Data collection		
APIs and URLs	requests	httr2
	json	Curl
		Rjson
		tidycensus
Scraping	BeautifulSoup	Rvest
	scrapy	Selenium
	selenium	
Data processing		
Text data	nltk	stringr
	re	tidytext
		quanteda
Image/spatial data	numpy	magick
	skimage	terra
	rasterio	sf
	xarray	tigris
	geopandas	tidycensus
Numeric data	pandas	dplyr
	dask	data.table
	multiprocessing	parallel
Data visualization	matplotlib	ggplot2
Advanced	seaborn	ggvis
	plotly	plotly
	bokeh	Leaflet

proprietary software updates. Third, using open-source software provides an opportunity for students to begin engaging with the open-source software community. One way to do this is to teach students how to collaborate on projects using the version control software Git and its online repository GitHub. Most cutting-edge open-source software is developed and available on GitHub; encouraging engagement with the platform can help students become producers of open-source software rather than just consumers.

There are some downsides to using open-source software, however, that students and instructors should keep in mind. First, packages in R or Python are often developed independently of one another and can occasionally conflict. The more top-down approach taken by closed-source or proprietary software is more effective at alleviating this issue since the company retains more control over who collaborates on the software and how. A second downside is that, given the decentralized nature of open-source software, technical assistance may be harder to find. Students must often learn to troubleshoot their own problems when using open-source software. Proprietary software often has a specific channel through which to obtain technical support. Regardless of these downsides, we feel that the benefits of open-source software outweigh the costs, so we demonstrate the data visualization pipeline using only R and Python packages.

In discussing each stage, we mention software packages and best practices within both the R and Python statistical computing environments. While not an exhaustive guide, our treatment of these steps should provide agricultural and applied economics instructors with guidance on how they can help students learn each stage of the pipeline.

4.1 Data Collection

Thanks to the prevalence of application programming interfaces (APIs), File Transfer Protocol (FTP), and web scraping tools, data is even easier to collect from the internet than in years past. While most students are familiar with downloading a CSV file onto their hard disk and reading it into software, some may be less familiar with downloading data directly into their software environment using an API or a URL. Learning to write scripts for downloading data that document the source URL and API is increasingly essential for students, as it enhances the transparency and reproducibility of research.

A few important skills need to be taught to students to collect these data effectively. For data that is already cleaned and curated, students should be taught to use both APIs and FTP. An application programming interface (API) is a platform that allows users to request data using a set of protocols and definitions. A typical example of an API *call* is when someone uses a smartphone app to request real-time data on weather conditions in their local area. The API is the interface that connects a user's request for data with the weather database.

To use APIs, students at a minimum need to be taught how to find the API they need and the general method for making an API request in their software. Most APIs use the Representational State Transfer (REST) format and will require users to make requests with the command `get` and analyze the response code to determine whether their `get` request was successful. If the request succeeds, the data is usually available as a JSON object in the programming environment or can be downloaded onto the disk.

A researcher can call an API to read data from a database directly into their programming environment. For example, most US federal statistical agencies have an API that allows users to query data directly from government databases after signing up for an API key (usually for free). Instead of searching a government website for a link to a CSV file, the API allows the researcher to make a more specific data query of a known, public data source. Collecting data this way makes research easier to reproduce and allows data to be collected in real time and at a larger scale. For example, a script can be written to automatically download new data as it becomes available (e.g., the Department of Labor jobs report).

Table 2 provides a nonexhaustive list of APIs that are likely useful to students in applied economics. It also lists some APIs for private data—such as Zillow, X, and Google Maps—that can only be used after paying for access to the API. These companies typically charge by the number of requests, which means it can be quite cost-effective for researchers if they only need to query the data a few times.[3]

Another method for obtaining data is the File Transfer Protocol (FTP), a communication protocol for transferring files across a network and one of the oldest ways to transfer data between computers. Instead of having an API, some websites will store their files on an FTP server that users can log on to. Some FTP servers require an authentication step, while others are accessible from the web browser, allowing data to be accessed using the URL directly without authentication. FTP servers accessible through a URL typically have a web address that starts with "ftp:\\" or include "ftp" somewhere in the name. Two examples of FTP servers for data are the US Census LEHD Origin-Destination Employment Statistics (LODES) data and the PRISM climate group data from Oregon State University.

---

[3] The main customers of these APIs are usually app developers who want to allow their users to query the information in their databases repeatedly. For example, the Google Maps API could be used by an app developer to allow the app users to see their real-time location. Since many researchers do not anticipate having to query the database indefinitely, sending one-time data requests to these APIs may be affordable even for student research projects.

Table 2. Popular APIs for data in applied economics

Name of API	URL
Agriculture	
QuickStats - USDA	https://quickstats.nass.usda.gov/api/
Cropland Data Layer	https://croplandcros.scinet.usda.gov/
Labor	
Bureau of Labor Statistics	https://www.bls.gov/bls/api_features.htm
Department of Labor	https://developer.dol.gov/beginner/
Census	
US Census/American Community Survey	https://www.census.gov/data/developers/data-sets.html
IPUMS	https://developer.ipums.org/docs/v2/apiprogram
Public Finance	
Bureau of Economic Analysis	https://apps.bea.gov/api/_pdf/bea_web_service_api_user_guide.pdf
Federal Reserve (FRED)	https://fred.stlouisfed.org/docs/api/fred/
US Treasury	https://fiscaldata.treasury.gov/api-documentation/
Trade	
World Bank (WITS)	http://wits.worldbank.org/data/public/WITSAPI_UserGuide.pdf
World Trade Organization	https://apiportal.wto.org/
Health	
CDC	https://open.cdc.gov/apis.html
Healthcare.gov	https://data.healthcare.gov/api
Climate and Weather	
NOAA	https://www.weather.gov/documentation/services-web-api
Geography	
OpenStreetMap (Overpass API)	https://wiki.openstreetmap.org/wiki/API
US Census Boundaries (TIGER)	https://www.census.gov/data/developers/data-sets/TIGERweb-map-service.html
Open Source Routing Machine	https://project-osrm.org/docs/v5.5.1/api/#general-options
Private Companies	
X/Twitter	https://developer.x.com/en/products/x-api
Zillow	https://www.zillowgroup.com/developers/
Google Maps	https://developers.google.com/maps

Accessing data on FTP servers is usually as simple as making a request to the FTP server, similar to making a request to an API server. In Python, using an API or an FTP server can be done using the `requests` package. For an API call, `requests` needs the URL and a list of parameters for querying the database with a get request. Downloading a file from an FTP server can also be done with `requests`. It requires students to give the URL and then navigate either through an authentication step or directly to the file and then use Python or R to download it either onto the disk or directly into the programming environment. If JSON data is downloaded, students can work with JSON files in Python the same way they would work with the Python object `dictionaries`. In R, the packages `httr` and `curl` are common for making requests to APIs or downloading files from FTP servers. For working with the resulting JSON data, the R packages `rjson` and `jsonlite` are also helpful.[4]

A third method of obtaining data is through web scraping. Web scraping involves collecting data from web pages directly, either structured or unstructured, and is an indispensable skill for obtaining data that is not in a curated database but still available on web pages. An example of structured data from a web page is a table embedded in an HTML file, whereas unstructured data could be statistics found in different places in the text of an HTML file. Collecting the data manually can be time-consuming when several web pages need to be searched. Writing a script that searches each HTML file and extracts the data can be more efficient. This allows students to automate the data collection procedure and even scrape multiple sites in parallel using their computer's multiple cores.

An essential aspect of teaching students web scraping is addressing its ethical considerations. While most websites can be legally scraped, students should be aware that some sites include areas the owner prefers not to be accessed by automated tools. Website owners usually specify these sites in a file called `robots.txt` to keep search engines from directing traffic to parts of their website that can crash from too much traffic. Scraping web pages that owners request not to be scraped is not only an irresponsible research practice but can also be detrimental to the operation of the website that they are obtaining data from (e.g., flooding the website with so many requests that it crashes). Students should be encouraged to be current on these norms and protocols before scraping websites for data.

The `BeautifulSoup` package—which downloads static HTML websites and allows them to be searched by HTML tags—is arguably the workhorse web scraping tool for Python. For R programming, the package `rvest` provides similar tools. For parallelizing web scraping, the Python package `scrapy` has safeguards for not overloading websites with requests. Another more advanced topic students can learn in web scraping is the use of packages like `selenium`, available in both R and Python, which allows the script to interact with the web page. This is an important tool for websites that have JavaScript elements, such as buttons that need to be clicked to get data, which cannot be scraped from a static HTML page with a package like `BeautifulSoup`.

4.2 Data Processing

Once data is downloaded into the R or Python environment, the next stage is to process it. Most students are familiar with the basic steps of data cleaning: tidying columns, fixing errors, identifying outliers, merging data sources, etc. For numeric data, this involves standard data manipulation packages like `pandas` in Python or `dplyr` in R.

In this subsection, we focus on the processing of two types of unstructured data: text and image. Here, we are distinguishing unstructured data as data that is not in the row-column form already familiar to most students. Processing and cleaning numeric data already in CSV format is arguably taught already in most econometrics courses or is learned during the process of doing research. Thus, we find it more

---

[4] A few coding examples using these packages can be found in Appendix A1.

useful to cover processing text and image data in this article while also mentioning more advanced data processing topics, such as parallelization and distributed programming.

For teaching text analysis in economics, concepts like n-grams, tokenization, and sentiment analysis are important to help students engage with the current economics literature using these tools (Gentzkow and Shapiro 2010; Currie et al. 2020; Elliott and Elliott 2020b; Indaco 2020; Spangler and Smith 2022; Shapiro et al. 2022). For more details on processing text data, readers can refer to Gentzkow et al. (2019), which reviews some common skills and concepts needed for text analysis and examples of economics papers using text data. In terms of packages, an important text processing package in Python is `nltk` (Natural Language Toolkit), which includes tools for breaking down text into various levels for analysis (a process called *tokenization*) and sentiment analyzers. The sentiment analysis models in `nltk` include off-the-shelf, rule-based models such as Valence Aware Dictionary and Sentiment Reasoner (VADER) and trainable models such as the Naive Bayes classifier. In R, similar resources are available in the `tidytext` and `quanteda` packages. For either R or Python, students can also benefit from learning how to use regular expressions to search text and extract particular patterns of text.

Though the use of text data is growing in economics, the use of spatial data has become even more ubiquitous. In particular, the economics field now heavily uses satellite (Donaldson and Storeygard 2016) and weather (Auffhammer et al. 2013) data to analyze topics in urban, environmental, agricultural, and practically every other field of applied economics. Since spatial data is often not presented in CSV format, teaching students a baseline level of geospatial data processing skills is essential to teaching students how to visualize spatial data. Students must know at least two file formats for basic geospatial data processing: raster (e.g., .png, .tiff) and vector (e.g., .shp, .json).

Understanding coordinate reference systems is essential for students to learn how to work with these files. Students can start by learning about geographic reference systems, which approximate the earth as an ellipsoid (a *geodetic datum*) and use a geographic coordinate system to describe locations, often in latitude and longitude. Converting these ellipsoid-based coordinates to a flat surface results in a projected coordinate system. Teaching this process helps students recognize the differences between spatial data in geographic coordinates (like latitude and longitude) and projected coordinates and understand how to compare them correctly. It also introduces different map projections, which can prioritize accuracy in either distance or area but not both, illustrating the trade-offs of each approach.

To process raster files, an image file with a geographic reference system attached, students also need to understand the basics of image processing. An image file is simply a numerical array that can be manipulated with any standard matrix operation. Image processing techniques, thresholding, and masking can then be effectively taught as simple applications of array manipulation to images.

With foundational GIS knowledge, students can start processing raster files as two-dimensional arrays, where each pixel's value corresponds to a row and column position. Raster files add geographic metadata to these positions, translating them into real-world coordinates like latitude and longitude. Thus, any operation performed on arrays can also be applied to rasters; after each transformation, the geographic metadata needs to be re-embedded to make it a raster again. In Python, packages like `skimage` support image processing, while `rasterio` and `xarray` handle rasters; in R, `magick` and `terra` serve similar functions.

For vector files, sequences of points that represent a boundary map on a geographic grid, students can be similarly taught to manipulate the vector of *x* and *y* coordinates the way they would a dataset and then re-embed the geographic metadata. Useful packages include Python's `geopandas` and `shapely` packages, with `geopandas` providing an interface very similar to `pandas`. R's `sf` package is a powerful tool for vector editing, while the `tigris` package allows users to download US political boundary data directly from the Census Bureau's TIGER/Line database.

In addition to the rise of unstructured data, numeric data has also grown larger and more challenging to work with. For some data visualization tasks, students may benefit from learning how to parallelize repetitive data processing tasks like scraping websites, estimating bootstrap statistics, or processing multiple files independently but simultaneously. In Python, popular parallelization packages are `dask` for the simplest operations to parallelize (often called *embarrassingly parallel*) and `multiprocessing` for more complex operations. In R, similar functionality is available with the package `parallel` and `boot` if students specifically need to bootstrap statistics. For working with large datasets, the Python packages `dask` and `PySpark` are useful for distributed programming, meaning processing data across multiple nodes of a computing cluster. In R, the package `data.table` is useful for working with large datasets while still using the `data.frame` front end that most R users are familiar with.

For the most memory- or CPU-intensive operations, instructors may consider devoting part of their instruction to teaching students to use a high-performance computing (HPC) cluster. Working on these systems requires a baseline knowledge of programming in the terminal language Bash since most systems run on a Linux operating system and can only be accessed through a terminal using a Secure Shell (SSH) protocol. The usefulness of teaching students to use an HPC cluster will depend on the scale of the data they usually use and each class's access to these resources. Even if they do not have access to an HPC, instructors can buy computing time on a larger server using the cloud computing services Google Cloud and Amazon Web Services (AWS).

4.3 Data Visualization

The final stage of the pipeline involves using R or Python to create data visualizations. Each language has a core plotting package essential for students to learn, as most advanced visualization packages build upon it. Developing familiarity with this foundational package is key, as it provides the basis for understanding and using other visualization tools within the language.[5]

In Python, this package is `matplotlib`. Almost every plotting package in Python—such as `seaborn`, `plotly`, and `bokeh`—call this package as their main dependency and are built off its architecture. One way to approach teaching `matplotlib` to students is to focus on teaching students the two main ways to write `matplotlib` code: MATLAB style and object-oriented style. Originally, `matplotlib` was written to mimic the graphics commands of the software MATLAB (hence the "mat" in the name), and so the first way to use `matplotlib` is through editing an active plot (Hunter 2007). In this way of writing code, the user writes commands that update the data and parameters of only one plot. To make and edit another plot, the user has to clear that plot as the active plot. In Python programming, this is done by calling functions from the sublibrary `matplotlib.pyplot` (almost always imported into code as the shorthand `plt`).

The object-oriented method of using `matplotlib` is arguably a more Pythonic way to write code, as it does not use an active plot but instead creates objects that the user edits directly. In this style of writing code, a user usually calls the function `subplots` from the library `matplotlib.pyplot`, which creates a `figure` object and an `axes` object. To manipulate the plot, the user can either edit the `figure` object, which is the top-level container for the whole plot, or the `axes` object, which contains the actual plotted data as well as the axis labels. The advantage of this approach is that students can create and edit multiple plots as objects instead of clearing the active plot when they want to create a new one. For more details on the differences between these approaches, readers can refer to the summary in Sanap (2020) and the `matplotlib` user guide.[6]

---

[5] Examples of code producing basic plots in both R and Python are available in Appendix A2.
[6] User guide can be found at this link.

While there are plotting functions in base R (default functions included without importing packages), the primary plotting package has more or less become `ggplot2`. While `matplotlib` was written to originally replicate MATLAB plotting commands, `ggplot2` was designed as a "layered grammar of graphics" by Wickham (2010), taking inspiration from the original book *The Grammar of Graphics* (Wilkinson, 2005). The purpose of a "grammar of graphics" is to be "a tool that enables us to concisely describe the components of a graphic" (Wickham 2010, p. 3). The components of a graphic in this framework include the data, aesthetic mappings, statistical transformations, scaling, and the coordinate system. In `ggplot2`, a graph layer specifies the data, which variables are mapped to each part of the graph (e.g., *x*-axis or *y*-axis), what geometric object should represent their relationship, and what transformations should be done to the data.

In practice, code in `ggplot2` looks like a series of function calls added together with the + sign, with each function call representing a component of the graph. For example, the function call `aes(x=height,y=weight)` specifies that `height` should be mapped to the *x*-axis and `weight` to the *y*-axis, `geom_point()` specifies the data will be represented as a scatter plot, and `coord_cartesian()` defines the coordinate system as Cartesian. Put together, a graph in `ggplot2` could be coded as `ggplot(data,aes(x=height,y=weight)) + geom_point() + coord_cartesian()`. Students may find this structure unintuitive, which makes it important to include an explanation of the layered grammar of graphics philosophy explained in Wickham (2010).

Since most of these packages are based on `matplotlib` and `ggplot2`, it is helpful for students to have experience with the design and syntax of Python and R's most important plotting packages before learning more advanced ones. Having learned the syntax of the two packages, students can better understand and modify the parameters of the more advanced packages. More advanced plotting packages include `seaborn`, `plotly`, and `bokeh` for Python and `lattice`, `ggiraph`, and `Leaflet` for R. The purpose of many of these packages is to expand the sorts of plots that can be done in `matplotlib` and `ggplot2`, including maps, interactive graphs, and more complicated plot types. The packages `plotly`, `bokeh`, and `ggvis` can specifically be used to deploy interactive graphs as JavaScript applications, which can be deployed to websites. For both Python and R, the package `shiny` can also be used to deploy interactive data visualizations to web pages.

5 An Example of the Pipeline

To assist those unfamiliar with the process, we provide a brief illustration of the data visualization pipeline at work. In a recent study of crop insurance use among minority and veteran farmers, researchers visualized the overall volume of crop insurance policies in each US county (Hagerman et al. 2025, refer to Figure 1). Creating a similar figure for the year 2017 uses all three stages of the data visualization pipeline. We present excerpts from reproducible code examples that collect, process, and visualize data from the USDA Risk Management Agency (RMA), USDA National Agricultural Statistics Service (NASS), and the US Census Bureau. In the appendix material, we include the full scripts that can reproduce the figures after inserting an API key for the NASS API.[7]

First, data were collected from three sources:

(1) USDA Risk Management Agency: the annual number of policies sold—pooled across all levels of coverage—for a given commodity in each US county in 2017
(2) USDA National Agricultural Statistics Service: the number of farms in each US county in 2017
(3) US Census Bureau: the TIGER/Line files for all counties in the lower 48 United States

---

[7] The scripts for this section are also publicly available at the GitHub repository link for this paper.

Python Code

```python
import requests

api_key = "API Key"
base_url = "https://quickstats.nass.usda.gov/api/api_GET/"

params = {'key': api_key,
          'source_desc': 'CENSUS',
          'source_desc': 'CENSUS',
          'domain_desc': 'TOTAL',
          'short_desc':'FARM OPERATIONS - NUMBER OF OPERATIONS',
          'year': '2017',
          'agg_level_desc': 'COUNTY'
          }

response = requests.get(base_url, params=params)

nass17 = response.json()

nass17_df = pd.DataFrame(nass17['data'])
```

R Code

```r
library(httr)
library(utils)

api_key <- "API Key"
base_url <- "https://quickstats.nass.usda.gov/api/api_GET/"

params <- list(key = api_key,
               source_desc = "CENSUS",
               domain_desc = "TOTAL",
               short_desc = "FARM OPERATIONS - NUMBER OF OPERATIONS",
               year = "2017",
               agg_level_desc = "COUNTY",
               format = "csv")

response <- GET(base_url, query = params)
nass17_df <- content(response, "parsed")
```

Figure 7. Using the NASS API to download data

Python Code

```python
import pandas as pd

## RMA

# Create county FIPS
rma17['GEOID'] = rma17['state_fips'].astype(str).str.zfill(2)
             + rma17['FIPS'].astype(str).str.zfill(3)

# Total policies by county
rma17 = rma17.groupby(['GEOID'])['policies_sold_total'].sum()
rma17 = rma17.reset_index()

## NASS

# Create county FIPS
nass17_df['GEOID'] = nass17_df['state_fips_code'] + \
                nass17_df['county_code']

# Convert to integer
nass17_df['farms'] = nass17_df['Value'].str.replace(",","")\
                            .astype(int)

# Select just two columns
nass17_df = nass17_df[['GEOID','farms']]

## Census

# Continental US only
counties = counties[(counties['GEOID'].astype(int) < 60000) &
                    (counties['STUSPS'] != 'AK') &
                    (counties['STUSPS'] != 'HI')]

counties = counties.to_crs({'init': 'epsg:5070'})

# Only keep states in the county boundary
US_states =  states['STUSPS'].isin(counties['STUSPS'].unique())
states= states[US_states]

states = states.to_crs({'init': 'epsg:5070'})

# Merge in data on GEOID (FIPS)
df = counties.merge(rma17,how="left",on="GEOID")
df = df.merge(nass17_df,how="left",on="GEOID")

# Make variable
df['policies_per_farm'] = df['policies_sold_total']/df['farms']

# Fill NAs with zero
df['policies_per_farm'] = df['policies_per_farm'].fillna(0)
```

R Code

```r
library(dplyr)
library(sf)
library(tidyverse)

## RMA
rma17 <- rma17 %>%
         # Create county FIPS
         mutate(GEOID = paste0(state_fips,FIPS)) %>%
         # Total policies by county
         summarize(policies = sum(policies_sold_total), .by = GEOID)

## NASS
nass17_df <- nass17_df %>%
         # Create county FIPS
         mutate(GEOID = paste0(state_fips_code,county_code)) %>%
         # Convert to integer
         mutate(farms = as.numeric(gsub(",","",Value))) %>%
         # Select just two columns
         select(GEOID, farms)

## Census
county_boundaries <- counties %>%
         shift_geometry() %>%
         select(GEOID, STUSPS, NAME) %>%
         # Continental US only
         filter(GEOID < 60,
                STUSPS != 'AK',
                STUSPS != 'HI')

# Only keep states in the county boundary
state_boundaries  <- states %>%
         filter(STUSPS %in% pull(counties,STUSPS))

# Merge in data
df <- county_boundaries %>%
  left_join(rma17, by = "GEOID") %>%
  left_join(nass17_df, by = "GEOID")

df <- df %>%
         # Make variable
         mutate(policies_per_farm = policies/farms) %>%
         # Fill NAs with zero
         mutate(policies_per_farm = ifelse(is.nan(policies_per_farm),
                                            0,
                                            policies_per_farm)) %>%
         mutate(policies_per_farm = ifelse(is.na(policies_per_farm),
                                            0,
                                            policies_per_farm))
```

Figure 8. Cleaning and merging data

Python Code

```python
1   import matplotlib.pyplot as plt
2   from matplotlib.colors import Normalize
3   from matplotlib.colorbar import ColorbarBase
4
5   # Create plot
6   fig, ax = plt.subplots(figsize=(10, 6))
7
8   # Create normalization for colormap
9   norm = Normalize(vmin=0, vmax=20)
10
11  # Plot counties with the coloring for policies per operation
12  df.plot(ax=ax,
13          column='policies_per_operation',
14          cmap='magma_r',
15          linewidth=0.1,
16          edgecolor='black',
17          norm=norm)
18
19  # Plot state boundaries
20  state_boundaries.boundary.plot(ax=ax,
21                                 color='black',
22                                 linewidth=0.25)
23
24  # Remove axes
25  ax.set_axis_off()
26
27  # Create colorbar
28  cbar_ax = fig.add_axes([0.25, 0.05, 0.5, 0.03])
29  cbar = ColorbarBase(cbar_ax,
30                      norm=norm,
31                      cmap="magma_r",
32                      orientation='horizontal',
33                      ticks=[0,5,10,15,20])
34  cbar.set_label("Policies Sold (Per Farm Operation), 2017",
35                 fontweight='bold', labelpad=10)
36
37  # Adjust layout
38  plt.tight_layout()
39  plt.subplots_adjust(bottom=0.15)
40
41  # Display the map
42  plt.show()
```
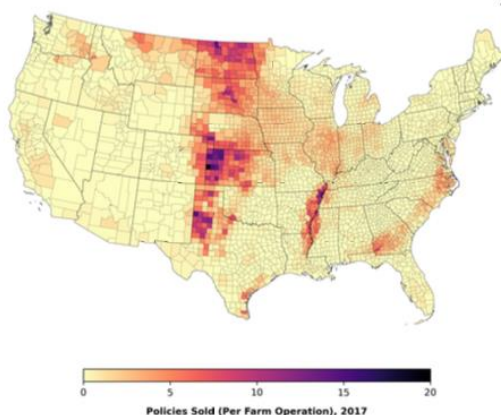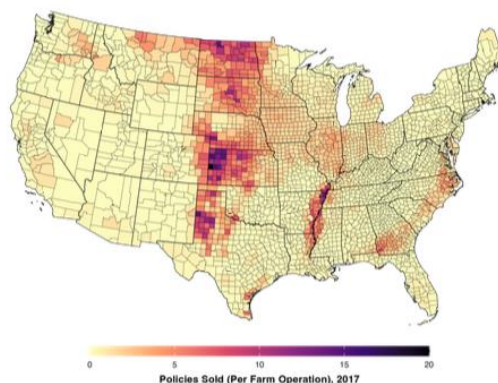
R Code

```r
1   library(ggplot2)
2   library(scales)
3
4   # Create plot
5   p <- ggplot() +
6
7   # Plot counties with the coloring for policies per operation
8     geom_sf(data = df,
9             aes(fill = policies_per_operation),
10            color = "black",  linewidth = .1) +
11
12  # Plot state boundaries
13    geom_sf(data = state_boundaries,
14            fill = NA,
15            color = "black", linewidth = .25) +
16
17  # Set colormap and create colorbar
18    scale_fill_viridis_c(option = 'magma',
19                         direction = -1,
20                         # Create normalization for colormap
21                         limits = c(0, 20), oob = squish,
22                         name = "Policies Sold (Per Farm Operation), 2017",
23                         guide = guide_colourbar(title.position = "top",
24                                                 title.hjust = 0.5,
25                                                 barheight = 0.35,
26                                                 barwidth = 25)) +
27    theme_void() +
28
29  # Set legend options
30    theme(legend.title = element_text(hjust = 0.5,
31                                      vjust = 0.5,
32                                      face = 'bold'),
33          legend.position = 'bottom')
34
35  # Display the map
36  print(p)
```


Figure 9. Creating a map of crop insurance policies

These data can either be downloaded as CSV files (to be processed as local data files on the researcher's machine) or, in the case of NASS and census data, the data can be accessed via an API.[8] In Figure 7, we show how to use `requests` in Python and `httr` to download census data from the NASS API. In both cases, the code sends a `get` request to the URL and sends back a JSON or CSV file. In the case of

---

[8] Cartographic boundaries for US counties were easily collected using the `tigris` package (Walker 2016), which imports shapefiles with a few short lines of code.

Python, the response comes back as a JSON file and there is an extra step to convert it to a `pandas` dataframe.

Second, we merge the boundary files, NASS data, and RMA data into a single dataset (see Figure 8). The data files gathered from USDA include a common variable—the county FIPS (Federal Information Processing Standards) code—which is used as a unique identifier to join together policies sold (USDA RMA) and farm operations (USDA NASS). The FIPS codes are created by concatenating state and county FIPS codes. Before joining them, both dataframes are manipulated to sum policies by county in the RMA data and remove commas from numbers to make the columns numeric in the NASS data. The county and state boundaries are reprojected with only the lower 48 states included in the file. The boundary files are then merged to the RMA and NASS data to create a dataframe called `df`.

At this stage, a spatial data visualization (i.e., a county map) is possible, but it would be misleading and largely unhelpful. This is because the crop insurance data is stored as a *count* of policies sold; to be truly beneficial to the intended audience, the data should be presented as a normalized measure, such as a rate or ratio. This is accomplished by dividing policies sold by the number of farm operations, transforming the variable into a per capita measure that can be readily compared across counties. In the dataframe, we divide the total policies per county by the number of farms, creating the variable `policies_per_farm`.

Finally, once the data were collected and processed, we can use `matplotlib` in Python or `ggplot2` in R to create a choropleth map of policies sold per farm at the county level. Figure 9 shows code in both languages that create the map. In addition to creating the map itself, the code creates a labeled color bar at the bottom of the figure.

These two pathways—using either R or Python—both arrive at the same goal: a clear and concise map that illustrates how the use of crop insurance varies across space. The data visualization pipeline effectively transforms raw data into actionable insights by systematically collecting, processing, and presenting information in a clear and interpretable manner. This process not only enhances the clarity of complex data but also empowers stakeholders to make more informed decisions by providing them with intuitive visualizations that reveal patterns and trends that might otherwise remain hidden.

6 Conclusion

The ability to effectively visualize data is an essential skill for today's applied economists and researchers. As the volume and complexity of data available continue to grow, the capacity to transform raw numbers into clear, compelling graphics will continue to be indispensable for both instructional and outreach purposes.

This paper has outlined a comprehensive framework for teaching data visualization, from the foundational stages of data collection and processing to the principles of effective chart design and communication. By equipping students and stakeholders with the technical know-how to manipulate data in open-source software programs, as well as the critical eye to identify good versus bad visualizations, instructors can better prepare the next generation of applied economists to harness the power of data visualization. Moreover, by strengthening both data visualization and graph literacy skills, we enhance the ability of students and stakeholders to communicate and apply research insights effectively in real-world contexts.

The framework for teaching data visualization outlined in this paper supports not only academic research but also the critical outreach and Extension activities that are central to applied and agricultural economics. The emphasis on graph literacy skills for outreach audiences ensures that the insights derived from agricultural research can be shared and applied by a wide range of stakeholders beyond the classroom. As data-driven decision-making becomes ever more prevalent, these skills will only grow in importance for the field.

Appendix A: Supplementary Code Examples

A1 Data Collection

In Python, the `requests` package is the most popular package for sending `get` requests to websites and databases. In order to download the HTML data from a webpage, the user need only put the URL into the `get` function:

```python
import requests
r = requests.get("www.example.com/document.html")
```

The status code can be checked by calling the `status` attribute of the request like so:

```python
r.status_code
```

A status code of 200 indicates success and any number starting with 4 usually means a failure (e.g., 404, 401). When the `get` request is being sent to an API, the request must also contain data specifying what data is being requested. For example, this call to the NASS QuickStats API requests the total number of milk cows at the state level from the 2017 Census of Agriculture:

```python
URL = "http://quickstats.nass.usda.gov/api/api_GET/"

params = {
        "key": api_key, # Put the API KEY
        "year":"2017", # The year we want.
        "domain_desc":"TOTAL", # Total across all domains
        "source_desc":"CENSUS",# Census, not survey
        "agg_level_desc":"STATE", # Level of data.
        "short_desc":"CATTLE, COWS, MILK - INVENTORY" # variable name
        }

r = requests.get(url = URL, params = params)
```

The python dictionary `params` contains the information the API needs to pull the data and is an argument of the function `get`. If the call is successful, the data will be stored in JSON format within the `response` object, here called *r*.

Here is an example of an API call just using a URL and the website Open Source Routing Machine (OSRM), a free API that can calculate road distance using the road network in Open Street Maps.[9] Below, we can calculate the distance and duration of a route between Boston and New York by supplying the

---

[9] These code samples were drafted with Claude 3.5 but then edited and tested by the authors. The prompt was, "Can you create two new code examples, one in R and one in Python, that make an API call to Open Source Routing Machine and extract the distance between two points?" which was followed up with "Don't make any functions, just run the New York and Boston example" to make it only one API call with no functions defined. Both scripts were tested in Python 3.8 and R 4.4.1.

latitude and longitude points to the URL. In both R and Python, the coordinates have to be pasted into the URL http://router.project-osrm.org/route/v1/driving/{coords}?overview=false, where the pairs of coordinates go in {coords}, separated by a semicolon.

Python Code

```python
1   # Use requests package
2   import requests
3
4   # New York and Boston coordinates
5   ny_lon, ny_lat = -74.006, 40.7128
6   boston_lon, boston_lat = -71.0589, 42.3601
7
8   # Create coordinates string for API
9   coords = f"{ny_lon},{ny_lat};{boston_lon},{boston_lat}"
10
11  # Construct URL with coordinates string
12  url = f"http://router.project-osrm.org/route/v1/driving/\
13        {coords}\
14        ?overview=false"
15
16  # GET request
17  response = requests.get(url)
18
19  # Get JSON
20  data = response.json()
21
22  # Get distance and duration
23  distance_km = data["routes"][0]["distance"] / 1000   # Meters to kilometers
24  duration_min = data["routes"][0]["duration"] / 3600   # Seconds to hours
25
26  print(f"Driving distance: {distance_km:.1f} km")
27  print(f"Estimated duration: {duration_min:.1f} hours")
```

R Code

```r
1   # R version
2   library(httr)
3   library(jsonlite)
4
5   # New York and Boston coordinates
6   ny_coords <- c(-74.006, 40.7128)
7   boston_coords <- c(-71.0589, 42.3601)
8
9   # Create coordinates string for API
10  coords <- paste(
11    paste(ny_coords[1], ny_coords[2], sep = ","),
12    paste(boston_coords[1], boston_coords[2], sep = ","),
13    sep = ";"
14  )
15
16  # Construct URL with the coordinates string
17  url <- paste0("http://router.project-osrm.org/route/v1/driving/",
18                coords,
19                "?overview=false")
20
21  # GET request
22  response <- GET(URL)
23
24  # Get JSON
25  data <- fromJSON(rawToChar(response$content))
26
27  # Get distance and duration
28  distance_km <- data$routes$distance / 1000   # Meters to kilometers
29  duration_min <- data$routes$duration / 3600   # Seconds to hours
30
31  cat(sprintf("Driving distance: %.1f km\n", distance_km))
32  cat(sprintf("Estimated duration: %.1f hours\n", duration_min))
```

A2 Data Visualization

To demonstrate how the base visualization packages are designed in Python and R, we include code in R and Python that demonstrate different ways to make plots. In the first example, a random dataset is produced and plotted in matplotlib in the "Matlab" style and the "Object-Oriented" style.

"Matlab" Style

```python
1   # Import packages
2   import numpy as np
3   import matplotlib.pyplot as plt
4
5   # Data generation
6   np.random.seed(42)
7   x = np.random.normal(0, 1, 100)
8   y = 0.5 * x + np.random.normal(0, 0.5, 100)
9
10  # Create the figure
11  plt.figure(figsize=(8, 6))
12
13  # Scatter plot
14  plt.scatter(x, y, alpha=0.5)
15
16  # Set labels and title
17  plt.xlabel('X values')
18  plt.ylabel('Y values')
19  plt.title('Basic Scatter Plot with Matplotlib, "Matlab style"')
20
21  # Display the plot
22  plt.show()
```

Object-Oriented Style

```python
1   # Import packages
2   import numpy as np
3   import matplotlib.pyplot as plt
4
5   # Data generation
6   np.random.seed(42)
7   x = np.random.normal(0, 1, 100)
8   y = 0.5 * x + np.random.normal(0, 0.5, 100)
9
10  # Create figure and axis objects
11  fig, ax = plt.subplots(figsize=(8, 6))
12
13  # Create the scatter plot using the axis object
14  scatter = ax.scatter(x, y, alpha=0.5)
15
16  # Set labels and title
17  ax.set_xlabel('X values')
18  ax.set_ylabel('Y values')
19  ax.set_title('Basic Scatter Plot with Matplotlib (Object-Oriented)')
20
21  # Display the plot
22  plt.show()
```

In the second example, we plot the random data in "base" R and `ggplot2`.

Base R

```
1        # Set random seed for reproducibility
2    set.seed(42)
3
4    # Generate example data
5    data <- data.frame(
6      x = rnorm(100),
7      y = rnorm(100)
8    )
9
10   # Create the plot
11   # First, create an empty plot with the basic structure
12   plot(data$x, data$y,
13        type = "n",  # "n" means no plotting initially
14        main = "Basic Scatter Plot with Base R",
15        xlab = "X values",
16        ylab = "Y values")
17
18   # Add points on top of grid
19   points(data$x, data$y,
20        pch = 19,         # Solid circle point type
21        col = adjustcolor("black", alpha.f = 0.5))  # Add transparency
```

ggplot2

```
1    # R version using ggplot2
2    library(ggplot2)
3
4    # Generate example data
5    set.seed(42)
6    data <- data.frame(
7      x = rnorm(100),
8      y = rnorm(100)
9    )
10
11   # Create the scatter plot
12   ggplot(data, aes(x = x, y = y)) +
13   geom_point(alpha = 0.5) +
14   labs(
15       title = "Basic Scatter Plot with ggplot2",
16       x = "X values",
17       y = "Y values"
18     ) +
19   theme_minimal()
```

About the Authors: Jared Hutchins (jhtchns2@illinois.edu) is an Assistant Professor of Agricultural & Consumer Economics with the University of Illinois Urbana-Champaign. Andrew J. Van Leuven (andrew.vanleuven@uvm.edu) is an Assistant Professor of Community Development & Applied Economics at the University of Vermont.

References

Abela, A. 2009. "Chart Suggestions: A Thought Starter." https://extremepresentation.typepad.com/files/choosing-a-good-chart-09.pdf.

Adams, F.M., and C.E. Osgood. 1973. "A Cross-Cultural Study of the Affective Meanings of Color." *Journal of Cross-Cultural Psychology* 4(2):135–156.

Auffhammer, M., S.M. Hsiang, W. Schlenker, and A. Sobel. 2013. "Using Weather Data and Climate Model Output in Economic Analyses of Climate Change." *Review of Environmental Economics and Policy* 7(2):181–198.

Bergstrom, C.T., and J.D. West. 2016. "The Principle of Proportional Ink." *CallingBullshit.org* https://www.callingbullshit.org/tools/tools_proportional_ink.html

Borland, D., and Taylor, R.M. II 2007. "Rainbow Color Map (Still) Considered Harmful." *IEEE Computer Graphics and Applications* 27(2):14–17.

Currie, J., H. Kleven, and E. Zwiers. 2020. "Technology and Big Data Are Changing Economics: Mining Text to Track Methods." *AEA Papers and Proceedings* 110:42–48.

Donaldson, D., and A. Storeygard. 2016. "The View from Above: Applications of Satellite Data in Economics." *Journal of Economic Perspectives* 30(4):171–198.

Elliott, M.S., and L.M. Elliott. 2020a. "Developing R Shiny Web Applications for Extension Education." *Applied Economics Teaching Resources* 2(4):9–19.

Elliott, M.S. and L.M. Elliott. 2020b. "Using Data Analytics and Decision-Making Tools for Agribusiness Education." *Applied Economics Teaching Resources* 2(2):38–50.

Gentzkow, M., B. Kelly, and M. Taddy. 2019. "Text as Data." *Journal of Economic Literature* 57(3):535–574.

Gentzkow, M., and J.M. Shapiro. 2010. "What Drives Media Slant? Evidence from US Daily Newspapers." *Econometrica* 78(1):35–71.

Hagerman, A.D., K.A. Schaefer, A.J. Van Leuven, F. Tsiboe, A.M. Young, and Y.A. Zereyesus. 2025. "Mitigating Structural Inequities in US Agricultural Risk Management." *Journal of Agricultural and Applied Economics* 57(1):86–113.

Healy, K. 2019. *Data Visualization: A Practical Introduction*. Princeton University Press.

Huff, D. 1954. *How to Lie with Statistics*. WW Norton.

Hunter, J.D. 2007. "matplotlib: A 2D Graphics Environment." *Computing in Science & Engineering* 9(3):90– 95.

Indaco, A. 2020. "From Twitter to GDP: Estimating Economic Activity from Social Media." *Regional Science and Urban Economics* 85:103591.

Jin, Y., M. Arrindell, S. Austin, L. Benny, J. Campbell, Q. Chen, L. Fithian, L. Vasquez, and J. Yi. 2024. "Analyzing and Visualization of Data: A Team Project in an Undergraduate Course Evaluating Food Insecurity in US Households." *Applied Economics Teaching Resources* 6(3):78–96.

Kabacoff, R. 2024. *Modern Data Visualization with R*. CRC Press. https://rkabacoff.github.io/datavis/

Madden, T.J., K. Hewett, and M.S. Roth. 2000. "Managing Images in Different Cultures: A Cross-National Study of Color Meanings and Preferences." *Journal of International Marketing* 8(4):90–107.

Minegishi, K., and T. Mieno. 2020. "Gold in Them Tha-R Hills: A Review of R Packages for Exploratory Data Analysis." *Applied Economics Teaching Resources* 2(3):29–68.

Reda, K. 2022. "Rainbow Colormaps: What Are They Good and Bad For?" *IEEE Transactions on Visualization and Computer Graphics* 29(12):5496–5510.

Rogowitz, B.E., and L.A. Treinish. 1998. "Data Visualization: The End of the Rainbow." *IEEE Spectrum* 35(12):52–59.

Sanap, T. 2020. "Pyplot vs Object-Oriented Interface." *Matplotblog*. https://matplotlib.org/matplotblog/posts/pyplot-vs-object-oriented-interface/

Shapiro, A.H., M. Sudhof, and J.D. Wilson. 2022. "Measuring News Sentiment." *Journal of Econometrics* 228(2):221–243.

Smith, N., and S. van der Walt. 2015. "A Better Default Colormap for matplotlib." https://www.youtube.com/watch?v=xAoljeRJ3lU

Spangler, E., and B. Smith. 2022. "Let Them Tweet Cake: Estimating Public Dissent Using Twitter." *Defence and Peace Economics* 33(3):327–346.

Tufte, E.R. 2001. *The Visual Display of Quantitative Information*, 2nd ed. Graphics Press.

US Census Bureau. 2023. "2018-2022 American Community Survey 5-Year Estimates, Table B19013_001: Median Household Income in the Past 12 Months." https://data.census.gov/

US Department of Agriculture National Agricultural Statistics Service (USDA NASS). 2024. "Acres Harvested by Commodity." *Quickstats* [database]. https://quickstats.nass.usda.gov/ [Accessed October 2024]

Van den Eeckhout, K. 2021, September 15. "Uncommon Chart Types: Waffle Chart." *Medium*. https://koenvandeneeckhout.medium.com/uncommon-chart-types-waffle-chart-51387eb52f5e

Walker, K. 2016. "tigris: An R Package to Access and Work with Geographic Data from the US Census Bureau." *R Journal* 8(2):231.

Ware, C., M. Stone, and D.A. Szafir. 2023. "Rainbow Colormaps Are Not All Bad." *IEEE Computer Graphics and Applications* 43(3):88–93.

Wickham, H. 2010. "A Layered Grammar of Graphics." *Journal of Computational and Graphical Statistics* 19(1):3–28.

Wilke, C.O. 2019. *Fundamentals of Data Visualization: A Primer on Making Informative and Compelling Figures*. O'Reilly Media. https://clauswilke.com/dataviz/

Wilkinson, L. 2005. *The Grammar of Graphics*, 2nd ed. Springer.