Teaching Computational Social Science for All

Jae Yeon Kim, *KDI School of Public Policy and Management, Republic of Korea*
Yee Man Margaret Ng, *University of Illinois Urbana–Champaign, USA*

ABSTRACT Computational methods have become an integral part of political science research. However, helping students to acquire these new skills is challenging because programming proficiency is necessary, and most political science students have little coding experience. This article presents pedagogical strategies to make transitioning from Excel, SPSS, or `Stata` to `R` or Python for data analytics less challenging and more exciting. First, it discusses two approaches for making computational methods accessible: showing the big picture and walking through the workflow. Second, a step-by-step guide for a typical course is provided using three examples: learning programming fundamentals, wrangling messy data, and communicating data analysis.

A precise definition of "computational social science" remains elusive. However, a widely used definition is that it is an interdisciplinary field in which computational tools and techniques are applied to advance social science research (Salganik 2018, xviii). The key premise is that computational tools and techniques empower social scientists—including political scientists—to conduct innovative analyses of unconventional data sources on an unprecedented scale (Alvarez 2016; Grimmer, Roberts, and Stewart 2021). Finding patterns in large-scale unstructured text (Gentzkow, Kelly, and Taddy 2019; Grimmer and Stewart 2013), image (Torres and Cantú 2021; Won, Steinert-Threlkeld, and Joo 2017), or audio (Knox and Lucas 2021) data was once immensely costly and time consuming. Computational tools and techniques have made collecting (e.g., web scraping) and analyzing (e.g., machine learning) these data easier, faster, and reasonably accurate. By complementing existing quantitative and qualitative methodologies, computational methods present many new opportunities to address key questions in political science (Brady 2019; Clark and Golder 2015; Monroe et al. 2015), from polarization and misinformation in the United States (Tucker et al. 2018) to censorship and collective action in China (King, Pan, and Roberts 2013).

Opportunities also come with challenges. For political science students, learning computational tools and techniques is challenging because proficiency in programming, to which they have little prior exposure, is necessary. Most political science students choose the field because they have a deep interest in politics but not necessarily data analysis. In addition, these students first approach data analysis via integrated statistical software packages, such as Microsoft Excel, `Stata`, and SPSS. These tools have both a point-and-click user interface and an easy-to-learn command syntax. For students, learning a programming language such as `R` or Python could be a completely foreign skillset.

However, there is a tradeoff between the simplicity of integrated statistical software and the more complicated yet powerful open-source programming languages. Excel is an excellent tool for spreadsheet creation and management; SPSS and `Stata` provide many handy tools for conducting statistical analyses. Nonetheless, these tools seldom are used to scrape websites, analyze large volumes of text data using machine learning, or create interactive data visualization. Transitioning from Excel, `Stata`, or SPSS to programming languages is necessary to conduct modern data analysis. Teaching programming is essential to help students learn broad and rapidly evolving computational tools and techniques.

This article introduces pedagogical strategies to make this transition less challenging and more exciting. Collectively, the authors have taught computational social science at both graduate and undergraduate levels in semester-long courses and short workshops for several years. Based on this extensive experience, we first present two principles: showing the big picture and walking through the workflow. We then apply these principles to the design of a course that aims to make computational methods accessible.

This discussion of the practical aspects of teaching computational social science comes with two caveats. First, a computational social science project should start with a theoretically driven

Jae Yeon Kim *is assistant professor of data science at the KDI School of Public Policy and Management. He can be reached at jkim@kdischool.ac.kr.*
Yee Man Margaret Ng *is assistant professor of journalism at the University of Illinois Urbana–Champaign. She can be reached at ymn@illinois.edu.*

doi:10.1017/S1049096521001815

research question rather than the latest computational technique. Collecting the largest possible volume of social media data or applying the most powerful machine-learning technique does not directly add value to political science research. These tools become relevant when they provide a new understanding of key theoretical questions in the relevant literature. In addition, linking computational methods to key political science topics has pedagogical value. It enables students to discover how they could apply these methods to explore their research interests (Williams et al. 2021).

Second, emphasizing the value of data-intensive social science research does not undermine the importance of careful human investigation. Big data is not the same as good data. Researchers still must assess whether the data can answer their research questions regarding measurement and internal and external validity (Meng 2018). Therefore, computational methods are better applied when researchers are well trained in research design and methods. A strong foundation in descriptive analysis, hypothesis testing, statistical modeling, and causal inference remains critical to answering empirical research questions.

Teaching programming is essential to help students learn broad and rapidly evolving computational tools and techniques.

However, having a strong motivation or a deeper understanding of a substantive problem does not automatically turn a research idea into a research project. A successful student-designed research project involving computational methods requires an adequate understanding of programming and its implementation. Nevertheless, programming is challenging for political science students because most have little prior experience. This challenge is especially acute among students from disadvantaged backgrounds because they have had relatively poor access to high-quality quantitative and computational education (for a review, see Xie, Fang, and Shauman 2015). This article addresses this problem by presenting principles and specific steps to lower the barriers to learning these exciting new tools and techniques.

PRINCIPLES OF TEACHING COMPUTATIONAL METHODS

An instructor can help students learn computational methods faster and easier by showing them the big picture and the workflow before discussing specific technical details. The big picture is the broad patterns regarding the beginning (input) and end (output) of specific tasks. The workflow is the logical steps that connect these two components. For example, suppose an instructor aims to teach digital data collection—that is, data collection and parsing from websites, Portable Document Format (PDF) files, and social media posts. The big picture here is the transformation of semi-structured data that do not look like a spreadsheet into structured data that do. Without contextualizing these data-collection techniques in a broad framework, the instructor could prematurely introduce the latest packages and libraries for digital data collection. This approach might work if most students have experience with these techniques and their primary goal is updating their skills. However, if students have little experience with digital data collection, they might not see "the forest for the trees." For them, learning so many techniques simultaneously might be confusing and overwhelming.

Because of these obstacles, it is worth taking a step back and remind students of the big picture and workflow. After that, the instructor can delve into each step. Web scraping is an example: What is the goal of this activity? The term "web scraping" provides an indication but not a clear understanding of the input and output data. If students scrape a website, they will obtain semi-structured data, which is a mix of HyperText Markup Language (HTML), Cascading Style Sheets (CSS), and JavaScript. Although the data look messy and lack a spreadsheet-like structure, they have other types of structures, such as key–value pairs: keys are labels and values are associated information. The instructor then will move on to explaining the workflow. Students can leverage these structures to locate information (Step 1), extract them (Step 2), and bind these elements together (Step 3). Because students typically want to summarize, model, and visualize this information, saving these elements as a data frame akin to a spreadsheet can be effective (Step 4).

An instructor can help students learn computational methods faster and easier by showing them the big picture and the workflow before discussing specific technical details.

This approach not only helps students to feel less intimidated by the subject, it also serves the long-term educational goal of making them effective self-learners. Self-learning skills are valuable because in a fast-evolving field such as computational social science, cutting-edge tools are continually changing. If the hidden context—the big picture and the workflow—is clarified, students find it easier to learn new techniques. Furthermore, learning this big picture and workflow helps them to build a mental framework for digital data collection. For example, if students learn how to scrape websites by understanding the underlying big picture and the workflow, they can quickly learn how to parse PDF files and scrape social media posts because these cases are other variants of semi-structured data.

After students have become accustomed to the workflow, they should learn how to combine these individual components (Perkins 2010). Whereas typing a command is easy, coming up with a logical sequence of computational actions is difficult. Helping students to develop an awareness of how one step is connected to another is critical because the end goal is to use these components together. In addition, considerable differences exist between learning each component and the methods for combining them. For example, a student might want to scrape a newspaper website to collect articles published during a specific

https://doi.org/10.1017/S1049096521001815 Published online by Cambridge University Press

period. However, there typically is a limitation on how many articles can be displayed on a single page. This is a useful feature that saves time for loading a webpage but makes scraping it a challenge. Manually copying, pasting, and tweaking the code on these pages is highly inefficient because, in some cases, the number of pages is in the hundreds or thousands. A better alternative is learning how to repeat the iterative process via loop or functional programming. This approach helps students to write a code with only a few lines—and it works regardless of the number of pages.

This example illustrates how helping students to learn an efficient workflow accomplishes two tasks simultaneously. First, it helps them translate a component of their research design (e.g., collecting data from websites) into computational terms (Brooker 2019). Second, it provides opportunities to learn a computational concept (e.g., automating iterations) in a practical context.

APPLICATIONS

This discussion focuses on applying these principles to the design of a semester-long computational social science course. Three examples are used: learning programming fundamentals, wrangling messy data, and communicating data analysis.

At the fundamental level, these principles could apply to undergraduate lectures as well as graduate seminars. However, depending on the target audience and their backgrounds and interests, the scope and focus should change. For example, if students are highly motivated and have experience in quantitative analyses, they could quickly absorb a substantial amount of new information. Otherwise, the instructor should decrease the pace and provide more concrete and intuitive examples. In addition, we highly recommend that instructors make all of the slides and code examples used in the lecture available before class so students can follow along more easily with in-class coding examples.

Case 1: Programming Fundamentals

Introducing programming fundamentals early is the key to lowering the initial barrier and helps students to be effective self-learners. Political scientists do not need to be statisticians. However, an adequate understanding of statistics is essential for choosing the correct application and interpretation of these methods. The same applies to computational methods. Students can obtain immediate programming assistance by searching the internet. For this purpose, there are many useful websites, such as Stack Overflow.[1] This practice is a valuable and integral part of self-learning. Copying solutions posted by someone else, however, does not provide the same level of understanding. If students do not fully comprehend the context in which a problem arises, they eventually will encounter the same problem again.

R and Python are both object-oriented and functional programming languages.[2] Objects define the actors and functions define their behavior. Students create, manipulate, and combine objects and functions when performing data analysis in R and Python. Programming is not magic; it is the art of logically combining objects and functions. Moving from Excel, SPSS, or Stata to R or Python means transitioning from pointing and clicking to creating and manipulating objects and functions. Consider importing a comma-separated values (CSV) file in R or Python. First, students must think about which function they should use and whether the function is available without importing an outside package. Second, they must provide the input (i.e., the file path for the CSV file) and save the output to an object they have named. Even this simple exercise involves understanding and using objects and functions.

Therefore, it is beneficial to devote the first two weeks of the course to helping students fully grasp these underlying computational concepts before teaching serious applications. As Buchler (2009, 527) argued, a "move on, and try to figure it out from the context" learning method does not work for subjects that require skill progression. Students can copy, paste, and run example codes for machine learning without comprehending the process. The fact that their codes run without error and produce the desired output does not necessarily mean that they fully understand the process. This emphasis on fundamental computational concepts applies even to those students who have extensive experience conducting quantitative analyses using SPSS or Stata. There are hurdles in moving from SPSS or Stata to R or Python. When a user creates a summary table of descriptive statistics or regression models in SPSS or Stata, knowing the object-oriented programming language is not required. In contrast, in R and Python, everything a user creates is an object, and every function takes and returns particular types of objects. Therefore, careful attention to how input and output data types vary is necessary. Helping students to cultivate this new habit enables them to avoid unnecessary stress.

When teaching these concepts, it is crucial to increase students' understanding by connecting what they know to what they do not know. For example, students may know various data types but not in a computational sense. They may know spreadsheets but not necessarily data frames. Both spreadsheets and data frames have a fixed number of rows and columns. However, using font color and highlighting as data can be helpful in spreadsheets but not in data frames. This additional information would be lost when the spreadsheet is imported as a data frame into R or Python (Broman and Woo 2018).

Introducing programming fundamentals early is the key to lowering the initial barrier and helps students to be effective self-learners.

Case 2: Wrangling Messy Data

After addressing programming fundamentals, an instructor should motivate students by demonstrating how programming can increase research productivity. Unless students are enthusiastic about programming, learning it is not the goal. Thus, they should be shown why learning programming is worth the investment of their time and effort.

https://doi.org/10.1017/S1049096521001815 Published online by Cambridge University Press

Data wrangling illustrates the benefits of learning programming because almost every project requires it. In quantitative methods courses, the example data often are processed. When students start working on real-world data, they soon discover that data are messy; 80% of data analysis involves data wrangling (Dasu and Johnson 2003). Although students can perform data cleaning in Excel, they must do it manually in a point-by-point manner. This is labor intensive and time consuming when managing large files or cleaning multiple files. More troubling is the irreproducibility of this point-and-click approach. Instead, using programming makes data wrangling reproducible and more efficient.

The "tidy-data" principles proposed by Wickham (2014) are an extension of Codd's (1990) relational algebra. They illustrate the big picture of data wrangling in statistical and computational analysis. First, the input data must be a data frame (i.e., a fixed number of rows and columns) to facilitate data summarization, modeling, and visualization. Second, columns are variables and rows are observations. Third, each observational unit comprises a data frame (Wickham 2014, 4). The recognition of this endgame enables students to see the specific problems with their messy data and consider using R or Python to fix them. This is easy because the "tidyr" (Wickham 2021) and "dplyr" (Wickham et al. 2021) packages in R and the "pandas" library in Python (McKinney 2011) provide comprehensive tools for implementing this foundational framework.

Wrangling can be categorized based on the issues with the data to increase students' awareness of the problems. For example, one category of problems is the shape of the dataset. Many spreadsheets are stored in a wide format to facilitate inspection of the data by human eyes. If these are administrative data collected yearly from 2000 to 2010, the dataset might have columns titled with those years. Unfortunately, summarization, modeling, and visualization require a long format. To reshape the data, instructors should guide students to create the variable representing all of those years.

When they are learning the typology of data-wrangling problems and solutions, students should not lose sight of the broad context. Because of the many data-wrangling techniques, it is easy for them to become lost in the details and fail to see the big picture. The tidy-data principles are valuable because data have an entire production cycle. Raw data have little value and are not self-explanatory. Humans need to summarize, model, and visualize raw data to add value for other humans.

Case 3: Communicating Data Analysis

Teaching data wrangling and visualization poses different challenges. Data wrangling has a clear goal, as summarized in the tidy-data principles; data visualization does not. There are abstract principles, such as the creation of accurate, easily interpretable, and aesthetically pleasing plots (Tufte 1985); however, their implementation is contextual. The decision to use a line or bar plot or logarithmic rescaling of the y-axis depends on the information to be communicated and the target audience.

When teaching data visualization, the focus should be the effective delivery of key messages. Teaching how to create point, bar, and line plots with popular packages and libraries in R and Python is insufficient. Students must know the types of visualization that are appropriate for delivering various forms of data. For example, representing raw data points with a scatter plot is valuable because descriptive statistics summaries (e.g., the mean) could overlook the unique distribution of the dataset (Anscombe 1973; Healy and Moody 2014).

After understanding the purpose of each visualization technique, students can learn to build the visualization of their choice from nothing. This will teach them the relationship between data and graphics. It also will enhance their understanding of the variables that are used for the aesthetics (e.g., x-axis and y-axis) of a plot and their ability to add and subtract information as needed to facilitate the audience's interpretation. This data-visualization approach, which is referred to as the "grammar of graphics" (Wilkinson 1999), has been used extensively in academia and industry because of the wide acceptance of the ggplot2 library in R (Wickham 2006, 2010).

OPTIONAL SUBJECTS

Teaching digital data collection, computational text analysis, and machine learning is optional. These are powerful tools for collecting and analyzing complex data (e.g., text, images, and audio) at scale. However, teaching programming fundamentals and these advanced topics in the same course can be overwhelming. If this is the case, students should be aware that these topics are optional. For example, suppose students are interested in how public opinion shifted during social movements (e.g., Black Lives Matter) using social media data. They can focus on a specific part of the research project (e.g., scraping social media posts) and apply the skills they learned from the course.

There are specific tips to help students learn these advanced subjects more easily. First, provide glossaries of computer science terms. Many canonical works in natural-language processing, machine learning, and big data were developed in computer science. Terms used in these works often are confusing to political science majors. For instance, the term "features" in machine learning is, in fact, "variables" in the social sciences. Weights and biases in deep learning are learnable parameters, and they should not be confused with their homonym counterparts in statistics.

Second, provide mathematical or statistical intuitions. For instance, typical survey data have a larger number of observations (i.e., participants) and a smaller number of variables (i.e., questions). In contrast, typical text data have a larger number of variables (i.e., words) and a smaller number of observations (i.e., documents). Analyzing this type of high-dimensional data poses two challenges. First, researchers must find the best ways to represent complex data numerically because computers understand numbers, not text. Second, they also should know how to reduce the number of these variables (also called features and dimensions) to ensure that the analysis is driven by signals rather than noise. Making this hidden context explicit helps students to better appreciate the purposes, assumptions, and mechanics of these techniques.

ASSESSMENT

To provide quick, frequent feedback, instructors should give students many small assignments instead of one large one. Designing small exercises can be problematic. If the exercises are not challenging, the students will not learn new information. If they are too complicated, students can lose confidence. To hit the "sweet spot," some principles must be observed. First, the exercises should be based on the content covered in the lecture.

https://doi.org/10.1017/S1049096521001815 Published online by Cambridge University Press

Second, instructors should encourage students to apply their skills beyond the contexts addressed in class. Third, the exercises should have an accompanying step-by-step guide and a template (e.g., R Markdown or Jupyter notebook). In essence, students must be given a framework that they can use, opportunities that they can explore, and constraints within which they can engage in exploration without excessive risk (Wickham 2015).

If instructors choose to require final projects instead of examinations, these small assignments can enable students to gradually develop necessary skills. These assignments can enhance their motivation to extend their best effort on each assignment and, in the final project, demonstrate their ability to integrate the skills that they acquired during the semester.

When providing feedback, it is crucial to focus on how students can improve. Students should be taught to write code so that they can fail quickly, often, and systematically. Computational methods are valuable not because they prevent human error but rather because they make mistakes transparent. In addition, identifying mistakes in a structured way is easy in programming. Students can create tests that check whether their functions produce the expected outcomes. When grading and commenting on each assignment, it is beneficial to remind students of the programming standards they need to meet, point to areas in which they need to improve, and suggest concrete steps that they can take to advance their skills.

CONCLUDING REMARKS

The path to mastering computational methods is a journey. Learning a wide range of complex technical skills and practicing them requires substantial time and effort. When students feel left behind, it is natural to identify themselves with their current failures (Dweck 2006). This perception discourages their efforts and widens the achievement gap between them and their peers. This article introduced pedagogical strategies that could restore their confidence and help them discover the joy of learning programming languages and their practical value. Understanding the big picture and workflow makes this journey easier and faster by connecting their intuition with new information and shaping their efforts to be more structured and focused. ■

NOTES

1. See https://stackoverflow.com.
2. There also are differences between the two languages. If a student's goal is to learn to program and use it for general purposes, learning Python would be helpful because it is widely used across academia and industry. In contrast, R was developed for doing data science (Chambers 2020) and is popular among social scientists. Thus, if a student's goal is to use the power of programming for social science research, R might be a better choice. Whether Python or R is more useful depends on a student's desired career path, learning goal, and primary usage.

REFERENCES

Alvarez, R. Michael (ed.). 2016. *Computational Social Science: Discovery and Prediction*. New York: Cambridge University Press.

Anscombe, Francis J. 1973. "Graphs in Statistical Analysis." *The American Statistician* 27 (1): 17–21.

Brady, Henry E. 2019. "The Challenge of Big Data and Data Science." *Annual Review of Political Science* 22:297–323.

Brooker, Phillip D. 2019. *Programming with Python for Social Scientists*. London and Thousand Oaks, CA: SAGE Publications.

Broman, Karl W., and Kara H. Woo. 2018. "Data Organization in Spreadsheets." *The American Statistician* 72 (1): 2–10.

Buchler, Justin. 2009. "Teaching Quantitative Methodology to the Math Averse." *PS: Political Science & Politics* 42 (3): 527–30.

Chambers, John M. 2020. "S, R, and Data Science." *Proceedings of the ACM on Programming Languages* 4 (HOPL): 1–17.

Clark, William Roberts, and Matt Golder. 2015. "Big Data, Causal Inference, and Formal Theory: Contradictory Trends in Political Science? Introduction." *PS: Political Science & Politics* 48 (1): 65–70.

Codd, Edgar F. 1990. *The Relational Model for Database Management: Version 2*. Reading, MA: Addison-Wesley.

Dasu, Tamraparni, and Theodore Johnson. 2003. *Exploratory Data Mining and Data Cleaning*. New York: Wiley-InterScience.

Dweck, Carol S. 2006. *Mindset: The New Psychology of Success*. New York: Random House.

Gentzkow, Matthew, Bryan Kelly, and Matt Taddy. 2019. "Text as Data." *Journal of Economic Literature* 57 (3): 535–74.

Grimmer, Justin, Margaret E. Roberts, and Brandon M. Stewart. 2021. "Machine Learning for Social Science: An Agnostic Approach." *Annual Review of Political Science* 24:395–419.

Grimmer, Justin, and Brandon M. Stewart. 2013. "Text as Data: The Promise and Pitfalls of Automatic Content Analysis Methods for Political Texts." *Political Analysis* 21 (3): 267–97.

Healy, Kieran, and James Moody. 2014. "Data Visualization in Sociology." *Annual Review of Sociology* 40:105–28.

King, Gary, Jennifer Pan, and Margaret E. Roberts. 2013. "How Censorship in China Allows Government Criticism but Silences Collective Expression." *American Political Science Review* 107 (2): 326–43.

Knox, Dean, and Christopher Lucas. 2021. "A Dynamic Model of Speech for the Social Sciences." *American Political Science Review* 115 (2): 649–66.

McKinney, Wes. 2011. "Pandas: A Foundational Python Library for Data Analysis and Statistics." *Python for High Performance and Scientific Computing* 14 (9): 1–9.

Meng, Xiao-Li. 2018. "Statistical Paradises and Paradoxes in Big Data (I): Law of Large Populations, Big Data Paradox, and the 2016 US Presidential Election." *The Annals of Applied Statistics* 12 (2): 685–726.

Monroe, Burt L., Jennifer Pan, Margaret E. Roberts, Maya Sen, and Betsy Sinclair. 2015. "No! Formal Theory, Causal Inference, and Big Data Are Not Contradictory Trends in Political Science." *PS: Political Science & Politics* 48 (1): 71–74.

Perkins, David. 2010. *Making Learning Whole: How Seven Principles of Teaching Can Transform Education*. San Francisco: Jossey-Bass.

Salganik, Matthew J. 2018. *Bit By Bit: Social Research in the Digital Age*. Princeton, NJ: Princeton University Press.

Torres, Michelle, and Francisco Cantú. 2021. "Learning to See: Convolutional Neural Networks for the Analysis of Social Science Data." *Political Analysis* 30 (1): 1–19.

Tucker, Joshua, Andrew Guess, Pablo Barberá, Cristian Vaccari, Alexandra Siegel, Sergey Sanovich, Denis Stukal, and Brendan Nyhan. 2018. *Social Media, Political Polarization, and Political Disinformation: A Review of the Scientific Literature*. Technical report. Menlo Park, CA: Hewlett Foundation.

Tufte, Edward R. 1985. "The Visual Display of Quantitative Information." *Journal for Healthcare Quality* 7 (3): 15.

Wickham, Hadley. 2006. "An Introduction to ggplot: An Implementation of the Grammar of Graphics in R." *Statistics*. DOI:10.1.1.469.6875.

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

Wickham, Hadley. 2014. "Tidy Data." *Journal of Statistical Software* 59 (10): 1–23.

Wickham, Hadley. 2015. "Teaching Safe-Stats, Not Statistical Abstinence." *Online Supplement Discussion of "Mere Renovation Is Too Little Too Late: We Need to Rethink Our Undergraduate Curriculum from the Ground Up" by G. Cobb, the American Statistician,* 69. https://nhorton.people.amherst.edu/mererenovation/17_Wickham.PDF.

Wickham, Hadley. 2021. *tidyr: Tidy Messy Data*. R Package Version 1.1.3.

Wickham, Hadley, Romain François, Lionel Henry, and Kirill Müller. 2021. *dplyr: A Grammar of Data Manipulation*. R Package Version 1.0.5.

Wilkinson, Leland. 1999. *The Grammar of Graphics*. New York: Springer.

Williams, Unislawa, Robert Brown, Marilyn Davis, Tinaz Pavri, and Fatemeh Shafiei. 2021. "Teaching Data Science in Political Science: Integrating Methods with Substantive Curriculum." *PS: Political Science & Politics* 54 (2): 336–39.

Won, Donghyeon, Zachary C. Steinert-Threlkeld, and Jungseock Joo. 2017. "Protest Activity Detection and Perceived Violence Estimation from Social Media Images." In *Proceedings of the 25th ACM International Conference on Multimedia*, 786–94. Mountain View, CA, October 23–27.

Xie, Yu, Michael Fang, and Kimberlee Shauman. 2015. "STEM Education." *Annual Review of Sociology* 41:331–57.

https://doi.org/10.1017/S1049096521001815 Published online by Cambridge University Press