DIME analytics

KRISTOFFER BJÄRKEFUR

LUÍZA CARDOSO DE ANDRADE

BENJAMIN DANIELS

MARIA JONES

DATA FOR DEVELOPMENT IMPACT

DIME ANALYTICS RESOURCE GUIDE

WORLD BANK GROUP

i2i DIME TRANSFORM DEVELOPMENT

UKaid from the British people

Norad

KRISTOFFER BJÄRKEFUR
LUÍZA CARDOSO DE ANDRADE
BENJAMIN DANIELS
MARIA JONES

DATA FOR DEVELOPMENT IMPACT: THE DIME ANALYTICS RESOURCE GUIDE

DIME ANALYTICS

PUBLISHED BY DIME ANALYTICS

http://worldbank.github.com/d4di

First printing, May 2019

Notes on this edition

This is a pre-publication edition of *Data for Development Impact: The DIME Analytics Resource Guide*, mainly intended for initial feedback. The DIME Analytics team is releasing this edition to coincide with the annual *Manage Successful Impact Evaluations* training, so that a wide group of people who make up part of the core audience for this book can use the book to supplement that training and provide feedback and improvements on it.

Whether you are a training participant, a DIME team member, or you work for the World Bank or another organization or university, we ask that you read the contents of this book carefully and critically. It is available as a PDF for your convenience at: `https://worldbank.github.com/d4di`. This website also includes the most updated instructions for providing feedback, as well as a log of errata and updates that have been made to the content.

Feedback

We encourage feedback and corrections so that we can improve the contents of the book in future editions. Please visit `https://worldbank.github.com/d4di/feedback/` to see different options on how to provide feedback. You can also email us at `dimeanalytics@worldbank.org` with input or comments, and we will be very thankful. We hope you enjoy the book!

Abbreviations

2SLS – Two-Stage Least Squares
AEA – American Economic Association
CAPI – Computer-Assisted Personal Interviewing
CI – Confidence Interval
DEC – Development Economics
DD or DiD – Differences-in-Differences
DGP – Data-Generating Process
DIME – Development Impact Evaluations
FC – Field Coordinator
FE – Fixed Effects
HFC – High-Frequency Checks
IRB – Instituional Review Board
IV – Instrumental Variables
MDE – Minimum Detectable Effect
NGO – Non-Governmental Organization
ODK – Open Data Kit
OLS – Ordinary Least Squares
OSF – Open Science Foundation
PI – Principal Investigator
PII – Personally-Identifying Information
QA – Quality Assurance
RA – Research Assistant
RD – Regression Discontinuity
RCT – Randomized Control Trial
SSC – Statistical Software Components
WBG – World Bank Group

Contents

11 Introduction: Data for development impact

15 Handling data ethically

21 Planning data work before going to field

31 Designing research for causal inference

37 Sampling, randomization, and power

53 Collecting primary data

61 Analyzing survey data

67 Publishing collaborative research

71 Appendix: The DIME Analytics Stata style guide

83 Bibliography

Dedicated to all the research assistants
who have wrangled data without being
taught how, hustled to get projects done
on time, wondered if they really should get
their PhD after all, and in doing so made
this knowledge necessary and possible.

Introduction: Data for development impact

Welcome to Data for Development Impact. This book is intended to serve as a resource guide for people who collect or use data for development research. In particular, the book is intended to guide the reader through the process of research using primary survey data, from research design to fieldwork to data management to analysis. This book will not teach you econometrics or epidemiology or agribusiness. This book will not teach you how to design an impact evaluation. This book will not teach you how to do data analysis, or how to code. There are lots of really good resources out there for all of these things, and they are much better than what we would be able to squeeze into this book.

What this book will teach you is how to think about quantitative data, keeping in mind that you are not going to be the only person collecting it, using it, or looking back on it. We hope to provide you two key tools by the time you finish this book. First, we want you to form a mental model of data collection as a "social process", in which many people need to have the same idea about what is to be done, and when and where and by whom, so that they can collaborate effectively on large, long-term projects. Second, we want to provide a map of concrete resources for supporting these processes in practice. As research teams and timespans have grown dramatically over the last decade, it has become inefficient for everyone to have their own personal style dictating how they use different functions, how they store data, and how they write code.

Doing credible research at scale

The team responsible for this book is known as **DIME Analytics**.[1] The DIME Analytics team works within the **Development Impact Evaluation unit (DIME)**[2] at the World Bank's **Development Economics group (DEC)**.[3] After years of supporting hundreds of projects and staff in total, DIME Analytics has built up a set of ideas, practices, and software tools that support all of the research projects operated at DIME.

In the time that we have been working in the development field, the proportion of projects that rely on **primary data** has soared.[4] Today, the scope and scale of those projects continue to expand rapidly, meaning that more and more people are working on the same data over longer and longer timeframes. This is because, while administrative datasets and **big data** have important uses, primary data[5] is critical for answering specific research questions.[6] As the ambition of development researchers grows, so too has the

[1] http://www.worldbank.org/en/research/dime/data-and-analytics

[2] http://www.worldbank.org/en/research/dime

[3] http://www.worldbank.org/en/research/

[4] Angrist, J., Azoulay, P., Ellison, G., Hill, R., and Lu, S. F. (2017). Economic research evolves: Fields and styles. *American Economic Review*, 107(5):293–97

[5] **Primary data:** data collected from first-hand sources.

[6] Levitt, S. D. and List, J. A. (2009). Field experiments in economics: The past, the present, and the future. *European Economic Review*, 53(1):1–18

complexity of the data on which they rely to make policy-relevant research conclusions from **field experiments**.[7] Unfortunately, this seems to have happened (so far) without the creation of guidelines for practitioners to handle data efficiently and transparently, which could provide relevant and objective quality markers for research consumers.

[7] **Field experiment:** experimental intervention in the real world, rather than in a laboratory.

One important lesson we have learned from doing field work over this time is that the most overlooked parts of primary data work are reproducibility and collaboration. You will be working with people who have very different skillsets and mindsets than you, from and in a variety of cultures and contexts, and you will have to adopt workflows that everyone can agree upon, and that save time and hassle on every project. This is not easy. But for some reason, the people who agreed to write this book enjoy doing it. (In part this is because it has saved ourselves a lot of time and effort.) As we have worked with more and more DIME recruits we have realized that we barely have the time to give everyone the attention they deserve. This book itself is therefore intended to be a vehicle to document our experiences and share it with with future DIME team members.

The **DIME Wiki** is one of our flagship resources for project teams, as a free online collection of our resources and best practices.[8] This book therefore complements the detailed-but-unstructured DIME Wiki with a guided tour of the major tasks that make up primary data collection.[9] We will not give a lot of highly specific details in this text, but we will point you to where they can be found, and give you a sense of what you need to find next. Each chapter will focus on one task, and give a primarily narrative account of: what you will be doing; where in the workflow this task falls; when it should be done; who you will be working with; why this task is important; and how to implement the most basic form of the task.

[8] http://dimewiki.worldbank.org/

[9] Like this: https://dimewiki.worldbank.org/wiki/Primary_Data_Collection

We will use broad terminology throughout this book to refer to different team members: **principal investigators (PIs)** who are responsible for the overall success of the project; **field coordinators (FCs)** who are responsible for the operation of the project on the ground; and **research assistants (RAs)** who are responsible for handling technical capacity and analytical tasks.

Writing reproducible code in a collaborative environment

Research reproduciblity and data quality follow naturally from good code and standardized practices. Process standardization means that there is little ambiguity about how something ought to be done, and therefore the tools that are used to do it are set in advance. Good code is easier to read and replicate, making it easier to spot mistakes.

The resulting data contains substantially less noise that is due to sampling, randomization, and cleaning errors. And all the data work can de easily reviewed before it's published and replicated afterwards.

Code is good when it is both correct and is a useful tool to whoever reads it. Most research assistants that join our unit have only been trained in how to code correctly. While correct results are extremely important, we usually tell our new research assistants that *when your code runs on your computer and you get the correct results then you are only half-done writing good code.*

Just as data collection and management processes have become more social and collaborative, code processes have as well.[10] This means other people need to be able to read your code. Not only are things like documentation and commenting important, but code should be readable in the sense that others can: (1) quickly understand what a portion of code is supposed to be doing; (2) evaluate whether or not it does that thing correctly; and (3) modify it efficiently either to test alternative hypotheses or to adapt into their own work.[11]

[10] https://dimewiki.worldbank.org/wiki/Stata_Coding_Practices

[11] https://kbroman.org/Tools4RR/assets/lectures/07_clearcode.pdf

To accomplish that, you should think of code in terms of three major elements: **structure**, **syntax**, and **style**. We always tell people to "code as if a stranger would read it", since tomorrow, that stranger will be you. The **structure** is the environment your code lives in: good structure means that it is easy to find individual pieces of code that correspond to tasks. Good structure also means that functional blocks are sufficiently independent from each other that they can be shuffled around, repurposed, and even deleted without damaging other portions. The **syntax** is the literal language of your code. Good syntax means that your code is readable in terms of how its mechanics implement ideas – it should not require arcane reverse-engineering to figure out what a code chunk is trying to do. **Style**, finally, is the way that the non-functional elements of your code convey its purpose. Elements like spacing, indentation, and naming can make your code much more (or much less) accessible to someone who is reading it for the first time and needs to understand it quickly and correctly.

For some implementation portions where precise code is particularly important, we will provide minimal code examples either in the book or on the DIME Wiki. In the book, they will be presented like the following:

code.do

```
* Load the auto dataset
    sysuse auto.dta , clear

* Run a simple regression
    reg price mpg rep78 headroom , coefl

* Transpose and store the output
    matrix results = `r(table)'

* Load the results into memory
    clear
    svmat results , n(col)
```

We have tried really hard to make sure that all the Stata code runs, and that each block is well-formatted and uses built-in functions. We will also point to user-written functions when they provide important tools. In particular, we have written two suites of Stata commands, `ietoolkit`[12] and `iefieldkit`,[13] that standardize some of our core data collection workflows. Providing some standardization to Stata code style is also a goal of this team, since groups are collaborating on code in Stata more than ever before. We will not explain Stata commands unless the behavior we are exploiting is outside the usual expectation of its functionality; we will comment the code generously (as you should), but you should reference Stata help-files `h [command]` whenever you do not understand the functionality that is being used. We hope that these snippets will provide a foundation for your code style. Alongside the collaborative view of data that we outlined above, good code practices are a core part of the new data science of development research. Code today is no longer a means to an end (such as a paper), but it is part of the output itself: it is a means for communicating how something was done, in a world where the credibility and transparency of data cleaning and analysis is increasingly important.

[12] https://dimewiki.worldbank.org/wiki/ietoolkit

[13] https://dimewiki.worldbank.org/wiki/iefieldkit

While adopting the workflows and mindsets described in this book requires an up-front cost, it should start to save yourself and others a lot of time and hassle very quickly. In part this is because you will learn how to do the essential things directly; in part this is because you will find tools for the more advanced things; and in part this is because you will have the mindset to doing everything else in a high-quality way. We hope you will find this book helpful for accomplishing all of the above, and that you will find that mastery of data helps you make an impact!

– The DIME Analytics Team

Handling data ethically

Development research does not just *involve* real people – it also *affects* real people. Policy decisions are made every day using the results of briefs and studies, and these can have wide-reaching consequences on the lives of millions. As the range and importance of the policy-relevant questions asked by development researchers grow, so too does the (rightful) scrutiny under which methods and results are placed. This scrutiny involves two major components: data handling and analytical quality. Performing at a high standard in both means that research participants are appropriately protected, and that consumers of research can have confidence in its conclusions.

What we call ethical standards in this chapter is a set of practices for data privacy and research transparency that address these two components. Their adoption is an objective measure of to judge a research product's performance in both. Without these transparent measures of credibility, reputation is the primary signal for the quality of evidence, and two failures may occur: low-quality studies from reputable sources may be used as evidence when in fact they don't warrant it, and high-quality studies from sources without an international reputation may be ignored. Both these outcomes reduce the quality of evidence overall. Even more importantly, they usually mean that credibility in development research accumulates at international institutions and top global universities instead of the people and places directly involved in and affected by it. Simple transparency standards mean that it is easier to judge research quality, and making high-quality research identifiable also increases its impact. This section provides some basic guidelines and resources for collecting, handling, and using field data ethically and responsibly to publish research findings.

Protecting confidence in development research

The empirical revolution in development research has led to increased public scrutiny of the reliability of research.[14] Three major components make up this scrutiny: **credibility**,[15] **transparency**,[16] and **replicability**.[17] Replicability is one key component of transparency. Transparency is necessary for consumers of research to be able to tell the quality of the research. Without it, all evidence credibility comes from reputation, and it's unclear what that reputation is based on, since it's not transparent.

Development researchers should take these concerns seriously. Many development research projects are purpose-built to cover specific questions, and may utilize unique data or small samples. This approach opens the door to working with the development community to answer both specific programmatic questions and

[14] Rogers, A. (2017). The dismal science remains dismal, say scientists. *Wired*

[15] Ioannidis, J. P., Stanley, T. D., and Doucouliagos, H. (2017). The power of bias in economics research. *The Economic Journal*

[16] Christensen, G. and Miguel, E. (2018). Transparency, reproducibility, and the credibility of economics research. *Journal of Economic Literature*, 56(3):920–80

[17] Duvendack, M., Palmer-Jones, R., and Reed, W. R. (2017). What is meant by "replication" and why does it encounter resistance in economics? *American Economic Review*, 107(5):46–51

general research inquiries. However, the data researchers utilize have never been reviewed by anyone else, so it is hard for others to verify that it was collected, handled, and analyzed appropriately. Maintaining confidence in research via the components of credibility, transparency, and replicability is the most important way that researchers can avoid serious error, and therefore these are not by-products but core components of research output.

Research replicability

Replicable research, first and foremost, means that the actual analytical processes you used are executable by others.[18] (We use "replicable" and "reproducible" somewhat interchangeably, referring only to the code processes themselves in a specific study; in other contexts they may have more specific meanings.[19]) All your code files involving data construction and analysis should be public – nobody should have to guess what exactly comprises a given index, or what controls are included in your main regression, or whether or not you clustered standard errors correctly. That is, as a purely technical matter, nobody should have to "just trust you", nor should they have to bother you to find out what happens if any or all of these things were to be done slightly differently.[20] Letting people play around with your data and code is a great way to have new questions asked and answered based on the valuable work you have already done. Services like GitHub that expose your code *history* are also valuable resources. They can show things like modifications made in response to referee comments; for another, they can show the research paths and questions you may have tried to answer (but excluded from publication) as a resource to others who have similar questions of their own data.

Secondly, reproducible research[21] enables other researchers to re-utilize your code and processes to do their own work more easily in the future. This may mean applying your techniques to their data or implementing a similar structure in a different context. As a pure public good, this is nearly costless. The useful tools and standards you create will have high value to others. If you are personally or professionally motivated by citations, producing these kinds of resources will almost certainly lead to that as well. Therefore, your code should be written neatly and published openly. It should be easy to read and understand in terms of structure, style, and syntax. Finally, the corresponding dataset should be openly accessible unless for legal or ethical reasons it cannot.[22]

[18] Dafoe, A. (2014). Science deserves better: the imperative to share complete replication files. *PS: Political Science & Politics*, 47(1):60–66

[19] http://datacolada.org/76

[20] Simmons, J. P., Nelson, L. D., and Simonsohn, U. (2011). False-positive psychology: Undisclosed flexibility in data collection and analysis allows presenting anything as significant. *Psychological Science*, 22(11):1359–1366; and Wicherts, J. M., Veldkamp, C. L., Augusteijn, H. E., Bakker, M., Van Aert, R., and Van Assen, M. A. (2016). Degrees of freedom in planning, running, analyzing, and reporting psychological studies: A checklist to avoid p-hacking. *Frontiers in Psychology*, 7:1832

[21] https://dimewiki.worldbank.org/wiki/Reproducible_Research

[22] https://dimewiki.worldbank.org/wiki/Publishing_Data

Research transparency

Transparent research will expose not only the code, but all the processes involved in establishing credibility to the public.[23] This means that readers be able to judge for themselves if the research was done well, and if the decision-making process was sound. If the research is well-structured, and all relevant documentation[24] is shared, this is as easy as possible for the reader to do. This is also an incentive for researchers to make better decisions, be skeptical about their assumptions, and, as we hope to convince you, make the process easier for themselves, because it requires methodical organization that is labor-saving over the complete course of a project.

[23] http://www.princeton.edu/~mjs3/open_and_reproducible_opr_2017.pdf

[24] https://dimewiki.worldbank.org/wiki/Data_Documentation

Registered Reports[25] can help with this process where they are available. By setting up a large portion of the research design in advance,[26] a great deal of work has already been completed, and at least some research questions are pre-committed for publication regardless of the outcome. This is meant to combat the "file-drawer problem",[27] and ensure that researchers are transparent in the additional sense that all the results obtained from registered studies are actually published.

[25] https://blogs.worldbank.org/impactevaluations/registered-reports-piloting-pre-results-review-process-journal-development-economics

[26] https://www.bitss.org/2019/04/18/better-pre-analysis-plans-through-design-declaration-and-diagnosis/

[27] Simonsohn, U., Nelson, L. D., and Simmons, J. P. (2014). P-curve: a key to the file-drawer. *Journal of Experimental Psychology: General*, 143(2):534

Documenting a project in detail greatly increases transparency. This means explicitly noting decisions as they are made, and explaining the process behind them. Documentation on data processing and additional hypothesis tested will be expected in the supplemental materials to any publication. Careful documentation will also save the research team a lot of time during a project, as it prevents you to have the same discussion twice (or more!), since you have a record of why something was done in a particular way. There is a number of available tools that will contribute to producing documentation, but project documentation should always be an active and ongoing process, not a one-time requirement or retrospective task. New decisions are always being made as the plan begins contact with reality, and there is nothing wrong with sensible adaptation so long as it is recorded and disclosed. (Email is `not` a note-taking service.)

There are various software solutions for building documentation over time. The **Open Science Framework**[28] provides one such solution, with integrated file storage, version histories, and collaborative wiki pages. **GitHub**[29] provides a transparent task management platform,[30] in addition to version histories and wiki pages. It offers multiple different ways to document a project that can be adapted to different team and project dynamics, but is less effective for file storage. Each project has its specificities, and the exact shape of this process can be molded to the team's needs, but it should be agreed

[28] https://osf.io/

[29] https://github.com

[30] https://dimewiki.worldbank.org/wiki/Getting_started_with_GitHub

on prior to project launch. This way, you can start building a project's documentation as soon as you start making decisions.

Research credibility

The credibility of research is traditionally a function of design choices.[31] Is the research design sufficiently powered through its sampling and randomization? Were the key research outcomes pre-specified or chosen ex-post? How sensitive are the results to changes in specifications or definitions? Tools such as **pre-analysis plans**[32] are important to assuage these concerns for experimental evaluations, but they may feel like "golden handcuffs" for other types of research.[33] Regardless of whether or not a formal pre-analysis plan is utilized, all experimental and observational studies should be **pre-registered**[34] using the **AEA** database,[35] the **3ie** database,[36], or the **OSF** registry[37] as appropriate.[38]

With the rise of empirical research and increased public scrutiny of scientific evidence, however, this is no longer enough to guarantee that findings will hold their credibility. Even if your methods are highly precise, your evidence is just as good as your data, and there are plenty of mistakes that can be made between establishing a design and generating final results that would compromise its conclusions. That is why transparency is key for research credibility. It allows other researchers, and research consumers, to verify the steps to a conclusion by themselves, and decide whether their standards for accepting a finding as evidence are met.

[31] Angrist, J. D. and Pischke, J.-S. (2010). The credibility revolution in empirical economics: How better research design is taking the con out of econometrics. *Journal of Economic Perspectives*, 24(2):3–30; and Ioannidis, J. P. (2005). Why most published research findings are false. *PLoS Medicine*, 2(8):e124

[32] https://dimewiki.worldbank.org/wiki/Pre-Analysis_Plan

[33] Olken, B. A. (2015). Promises and perils of pre-analysis plans. *Journal of Economic Perspectives*, 29(3):61–80

[34] https://dimewiki.worldbank.org/wiki/Pre-Registration

[35] https://www.socialscienceregistry.org/

[36] http://ridie.3ieimpact.org/

[37] https://osf.io/registries

[38] http://datacolada.org/12

Ensuring privacy and security in research

Anytime you are collecting primary data in a development research project, you are almost certainly handling data that include **personally-identifying information (PII)**.[39] PII data contains information that can, without any transformation, be used to identify individual people, households, villages, or firms that were included in **data collection**. This includes names, addresses, and geolocations, and extends to personal information such as email addresses, phone numbers, and financial information. In some contexts this list may be more extensive – for example, if you are working in a small environment, someone's age and gender may be sufficient to identify them even though these would not be considered PII in a larger context. Therefore you will have to use careful judgment in each case to decide which pieces of information fall into this category.[40]

Most of the field research done in development involves human subjects – real people.[41] As a researcher, you are asking people to

[39] **Personally-identifying information:** any piece or set of information that can be used to identify an individual research subject. https://dimewiki.worldbank.org/wiki/De-identification#Personally_Identifiable_Information

[40] https://sdcpractice.readthedocs.io/en/latest/

[41] https://dimewiki.worldbank.org/wiki/Human_Subjects_Approval

trust you with personal information about themselves: where they live, how rich they are, whether they have committed or been victims of crimes, their names, their national identity numbers, and all sorts of other data. PII data carries strict expectations about data storage and handling, and it is the responsibility of the research team to satisfy these expectations.[42] Your donor or employer will most likely require you to hold a certification from a source such as Protecting Human Research Participants[43] or the CITI Program.[44]

Raw data which contains PII *must* be **encrypted**[45] during data collection, storage, and transfer. Most modern data collection software makes the first part straightforward.[46] However, secure storage and transfer are your responsibility.[47] There are plenty of options available to keep your data safe, at different prices, from enterprise-grade solutions to combined free options. You will also want to setup a password manager that allows you to share encryption keys inside your team. These will vary in level of security and ease of use, and sticking to a standard practice will make your life easier, so agreeing on a protocol from the start of a project is ideal.

In general, though, you shouldn't need to handle PII data very often. Once data is securely collected and stored, the first thing you will generally do is **de-identify** it.[48] De-identified data should avoid, for example, you being sent back to every household to alert them that someone dropped all their personal information on a public bus and we don't know who has it. This simply means creating a copy of the data that contains no personally-identifiable information. This data should be an exact copy of the raw data, except it would be okay for it to be publicly released.[49] Ideally, all machines used to store and process PII data are not only password protected, but also encrypted at the hard drive level (most modern operating systems provide such a tool). This means that even if you lose your computer with identifying data in it, anyone who gets hold of it still cannot access the information.

[42] https://dimewiki.worldbank.org/wiki/Research_Ethics

[43] https://humansubjects.nih.gov/sites/hs/phrp/PHRP_Archived_Course_Materials.pdf

[44] https://about.citiprogram.org/en/series/human-subjects-research-hsr/

[45] https://dimewiki.worldbank.org/wiki/encryption

[46] https://dimewiki.worldbank.org/wiki/SurveyCTO_Form_Settings

[47] https://dimewiki.worldbank.org/wiki/Data_Security

[48] https://dimewiki.worldbank.org/wiki/De-identification

[49] Matthews, G. J., Harel, O., et al. (2011). Data confidentiality: A review of methods for statistical disclosure limitation and methods for assessing privacy. *Statistics Surveys*, 5:1–29

Planning data work before going to field

Preparation for data work begins long before you collect any data. In order to be prepared for fieldwork, you need to know what you are getting into. This means knowing which data sets you need, how those data sets will stay organized and linked, and what identifying information you will collect for the different types and levels of data you'll observe. Identifying these details creates a **data map** for your project, giving you and your team a sense of how information resources should be organized. It's okay to update this map once the project is underway – the point is that everyone knows what the plan is.

Then, you must identify and prepare your tools and workflow. All the tools we discuss here are designed to prepare you for collaboration and replication, so that you can confidently manage tools and tasks on your computer. We will try to provide free, open-source, and platform-agnostic tools wherever possible, and provide more detailed instructions for those with which we are familiar. However, most have a learning and adaptation process, meaning you will become most comfortable with each tool only by using it in real-world work. Get to know them well early on, so that you do not spend a lot of time later figuring out basic functions.

Preparing your digital workspace

Being comfortable using your computer and having the tools you need in reach is key. This section provides a brief introduction to key concepts and toolkits that can help you take on the work you will be primarily responsible for. Some of these skills may seem elementary, but thinking about simple things from a workflow perspective can help you make marginal improvements every day you work.

Teams often develop they workflows as they go, solving new challenges when they appear. This will always be necessary, and new challenges will keep coming. However, there are a number of tasks that will always have to be completed on any project. These include organizing folders, collaborating on code, controlling different versions of a file, and reviewing each other's work. Thinking about the best way to do these tasks ahead of time, instead of just doing it as quickly as you can when needed, will save your team a lot of re-working. This chapter will outline the main points to discuss within the team, and point to some possible solutions.

Setting up your computer

First things first: turn on your computer. Make sure you have fully updated the operating system, that it is in good working order, and that you have a **password-protected** login. All machines that will handle personally-identifiable information should be encrypted; this should be available built-in to most modern operating systems (BitLocker on PCs or FileVault on Macs). Then, make sure your computer is backed up. Follow the **3-2-1 rule**: (3) copies of everything; (2) different physical media; (1) offsite storage.[50]

[50] https://www.backblaze.com/blog/the-3-2-1-backup-strategy/

One reasonable setup is having your primary disk, a local hard drive managed with a tool like Time Machine, and a remote copy managed by a tool like Backblaze. Dropbox files count only as local copies and never backups, because others can alter it.

Find your **home folder**. On MacOS, this will be a folder with your username. On Windows, this will be something like "This PC". (It is never your desktop.) Nearly everything we talk about will assume you are starting from here. Ensure you know how to get the **absolute file path** for any given file. On MacOS this will be something like `/users/username/dropbox/project/...`, and on Windows, `C:/users/username/github/project/...`. We will write file paths such as `/Dropbox/project-title/DataWork/EncryptedData/` using forward slashes, and mostly use only A-Z, dash, and underscore. You should *always* use forward slashes (`/`) in file paths, just like an internet address, and no matter how your computer writes them, because the other type will cause your work to break many systems. You can use spaces in names of non-technical files, but not technical ones.[51] Making the structure of your files part of your workflow is really important, as is naming them correctly so you know what is where.

[51] http://www2.stat.duke.edu/~rcs46/lectures_2015/01-markdown-git/slides/naming-slides/naming-slides.pdf

Folder management

The first thing your team will need to create is a shared folder.[52] If every team member is working on their local computers, there will be a lot of crossed wires when collaborating on any single file, and e-mailing one document back and forth is not efficient. Your folder will contain all your project's documents. It will be the living memory of your work. The most important thing about this folder is for everyone in the team to know how to navigate it. Creating folders with self-explanatory names will make this a lot easier. Naming conventions may seem trivial, but often times they only make sense to whoever created them. It will often make sense for the person in the team who uses a folder the most to create it.

[52] Common tools for folder sharing are Dropbox, Box, and OneDrive.

For the purpose of this book, we're mainly interested in the folder that will store the project's data work. Agree with your team on

a specific folder structure, and set it up at the beginning of the research project, to prevent folder re-organization that may slow down your workflow and, more importantly, prevent your code files from running. DIME Analytics created and maintains `iefolder`[53] as a part of our `ietoolkit` suite. This command sets up a standardized folder structure for what we call the `/DataWork/` folder.[54] It includes folders for all the steps of a typical DIME project. However, since each project will always have project-specific needs, we tried to make it as easy as possible to adapt when that is the case. The main advantage of having a universally standardized folder structure is that changing from one project to another requires less time to get acquainted with a new organization scheme.

[53] https://dimewiki.worldbank.org/wiki/iefolder

[54] https://dimewiki.worldbank.org/wiki/DataWork_Folder

`iefolder` also creates master do-files.[55] Master scripts are a key element of code organization and collaboration, and we will discuss some important features soon. With regard to folder structure, it's important to keep in mind that the master script should mimic the structure of the `/DataWork/` folder. This is done through the creation of globals (in Stata) or string scalars (in R). These are "variables" – coding shortcuts that refer to subfolders, so that those folders can be referenced without repeatedly writing out their complete filepaths. Because the `/DataWork/` folder is shared by the whole team, its structure is the same in each team member's computer. What may differ is the path to the project folder (the highest-level shared folder). This is reflected in the master script in such a way that the only change necessary to run the entire code from a new computer is to change the path to the project folder. The code below shows how folder structure is reflected in a master do-file.

[55] https://dimewiki.worldbank.org/wiki/Master_Do-files

Code management

Once you start a project's data work, the number of scripts, datasets and outputs that you have to manage will grow very quickly. This can get out of hand just as quickly, so it's important to organize your data work and follow best practices from the beginning. Adjustments will always be needed along the way, but if the code is well-organized, they will be much easier to make. Below we discuss a few crucial steps to code organization. They all come from the principle that code is an output by itself, not just a means to an end. So code should be written thinking of how easy it will be for someone to read it later.

Code documentation is one of the main factors that contribute to readability, if not the main one. There are two types of comments that should be included in code. The first one describes what is being done. This should be easy to understand from the code itself

if you know the language well enough and the code is clear. But writing plain English (or whichever language you communicate with your team on) will make it easier for everyone to read. The second type of comment is what differentiates commented code from well-commented code: it explains why the code is performing a task in a particular way. As you are writing code, you are making a series of decisions that (hopefully) make perfect sense to you at the time. However, you will probably not remember how they were made in a couple of weeks. So write them down in your code. There are other ways to document decisions (GitHub offers a lot of different documentation options, for example), but information that is relevant to understand the code should always be written in the code itself.

Code organization is the next level. Start by adding a code header. This should include simple things such as stating the purpose of the script and the name of the person who wrote it. If you are using a version control software, the last time a modification was made and the person who made it will be recorded by that software. Otherwise, you should include it in the header. Finally, and more importantly, use it to track the inputs and outputs of the script. When you are trying to track down which code creates a data set, this will be very helpful.

Breaking your code into readable steps is also good practice on code organization. One way to do this is to create sections where a specific task is completed. So, for example, if you want to find the line in your code where a variable was created, you can go straight to `PART 2: Create new variables`, instead of reading line by line of the code. RStudio makes it very easy to create sections, and compiles them into an interactive script index. In Stata, you can use comments to create section headers, though they're just there to make the reading easier. Adding a code index to the header by copying and pasting section titles is the easiest way to create a code map. You can then add and navigate through them using the find command. Since Stata code is harder to navigate, as you will need to scroll through the document, it's particularly important to avoid writing very long scripts. One reasonable rule of thumb is to not write files that have more than 200 lines. This is also true for other statistical software, though not following it will not cause such a hassle.

stata-master-dofile.do

```
/*******************************************************************************
*                           TEMPLATE MASTER DO-FILE                           *
*******************************************************************************
*                                                                             *
*   PURPOSE:     Reproduce all data work, map inputs and outputs,             *
*                facilitate collaboration                                     *
*                                                                             *
*   OUTLINE:     PART 1:  Set standard settings and install packages          *
*                PART 2:  Prepare folder paths and define programs            *
*                PART 3:  Run do files                                        *
*                                                                             *
*******************************************************************************
             PART 1:  Set standard settings and install packages
*******************************************************************************/

    if (0) {
        ssc install ietoolkit,  replace
    }

    ieboilstart, v(15.1)
    `r(version)'

/*******************************************************************************
              PART 2:  Prepare folder paths and define programs
*******************************************************************************/

    * Research Assistant folder paths
    if "`c(username)'" == "ResearchAssistant" {
        global github      "C:/Users/RA/Documents/GitHub/d4di/DataWork"
        global dropbox     "C:/Users/RA/Dropbox/d4di/DataWork"
        global encrypted   "A:/DataWork/EncryptedData"  // Always mount to A disk!
    }


   * Baseline folder globals
    global bl_encrypt       "${encrypted}/Round Baseline Encrypted"
    global bl_dt            "${dropbox}/Baseline/DataSets"
    global bl_doc           "${dropbox}/Baseline/Documentation"
    global bl_do            "${github}/Baseline/Dofiles"
    global bl_out           "${github}/Baseline/Output"

/*******************************************************************************
                            PART 3: Run do files
*******************************************************************************/

/*------------------------------------------------------------------------------
  PART 3.1:  De-identify baseline data
------------------------------------------------------------------------------
 REQUIRES:   ${bl_encrypt}/Raw Identified Data/D4DI_baseline_raw_identified.dta
 CREATES:    ${bl_dt}/Raw Deidentified/D4DI_baseline_raw_deidentified.dta
 IDS VAR:    hhid
------------------------------------------------------------------------------ */
    do "${bl_do}/Cleaning/deidentify.do"

/*------------------------------------------------------------------------------
  PART 3.2:  Clean baseline data
------------------------------------------------------------------------------
  REQUIRES:   ${bl_dt}/Raw Deidentified/D4DI_baseline_raw_deidentified.dta
  CREATES:    ${bl_dt}/Final/D4DI_baseline_clean.dta
              ${bl_doc}/Codebook baseline.xlsx
  IDS VAR:    hhid
------------------------------------------------------------------------------ */
    do "${bl_do}/Cleaning/cleaning.do"

/*------------------------------------------------------------------------------
 PART 3.3:  Construct income indicators
------------------------------------------------------------------------------
  REQUIRES:   ${bl_dt}/Final/D4DI_baseline_clean.dta
  CREATES:    ${bl_out}/Raw/D4DI_baseline_income_distribution.png
              ${bl_dt}/Intermediate/D4DI_baseline_constructed_income.dta
  IDS VAR:    hhid
------------------------------------------------------------------------------ */
    do "${bl_do}/Construct/construct_income.do"
```

To bring all these smaller code files together, maintain a master script. A master script is the map of all your project's data work, a table of contents for the instructions that you code. Anyone should be able to follow and reproduce all your work from raw data to all outputs by simply running this single script. By follow, we mean someone external to the project who has the master script can (i) run all the code and recreate all outputs, (ii) have a general understanding of what is being done at every step, and (iii) see how codes and outputs are related. The master script is also where all the settings are established, such as folder paths, functions and constants used throughout the project.

Agree with your team on a plan to review code as it is written. Reading other people's code is the best way to improve your coding skills. And having another set of eyes on your code will make you more comfortable with the results you find. It's normal (and common) to make mistakes as you write your code quickly. Reading it again to organize and comment it as you prepare it to be reviewed will help you identify them. Try to have code review scheduled frequently, as you finish writing a piece of code, or complete a small task. If you wait for a long time to have your code review, and it gets too long, preparing it for code review and reviewing them will require more time and work, and that is usually the reason why this step is skipped. Making sure that the code is running, and that other people can understand the code is also the easiest way to ensure a smooth project handover.

Version control

A **version control system** is the way you manage the changes to any computer file. This is important, for example, for your team to be able to find the version of a presentation that you delivered to your donor, but also to understand why the significance of your estimates has changed. Everyone who has ever encountered a file named something like `final_report_v5_LJK_KLE_jun15.docx` can appreciate how useful such a system can be. Most file sharing solutions offer some level of version control. These are usually enough to manage changes to binary files (such as Word and PowerPoint documents) without needing to appeal to these dreaded file names. For code files, however, a more complex version control system is usually desirable. We recommend using Git[56] for all plain text files. Git tracks all the changes you make to your code, and allows you to go back to previous versions without losing the information on changes made. It also makes it possible to work on two parallel versions of the code, so you don't risk breaking the code for other team members

[56] **Git:** a multi-user version control system for collaborating on and tracking changes to code as it is written.

as you try something new,

Increasingly, we recommend the entire data work folder to be created and stored separately in GitHub. Nearly all code and outputs (except datasets) are better managed this way. Code is written in its native language, and increasingly, written outputs such as reports, presentations and documentations can be written using different **literate programming** tools such as LaTeX and dynamic documents. You should therefore feel comfortable having both a project folder and a code folder. Their structures can be managed in parallel by using `iefolder` twice. The project folder can be maintained in a synced location like Dropbox, and the code folder can be maintained in a version-controlled location like GitHub. While both are used for sharing and collaborating, there is a sharp difference between the functionality of sync and version control. Namely, sync forces everyone to have the same version of every file at all times and does not support simultaneous editing well; version control does the opposite. Keeping code in a version-controlled folder will allow you to maintain better control of its history and functionality, and because of the specificity with which code depends on file structure, you will be able to enforce better practices there than in the project folder.

Output management

One more thing to be discussed with your team is the best way to manage outputs. A great number of them will be created during the course of a project, from raw outputs such as tables and graphs to final products such as presentations, papers and reports. When the first outputs are being created, agree on where to store them, what software to use, and how to keep track of them.

Decisions about storage of final outputs are made easier by technical constraints. As discussed above, Git is a great way to control for different versions of plain text files, and sync software such as Dropbox are better for binary files. So storing raw outputs in formats like `.tex` and `.eps` in Git and final outputs in PDF, PowerPoint or Word, makes sense. Storing plain text outputs on Git makes it easier to identify changes that affect results. If you are re-running all of your code from the master when significant changes to the code are made, the outputs will be overwritten, and changes in coefficients and number of observations, for example, will be highlighted.

Though formatted text software such as Word and PowerPoint are still prevalent, more and more researchers are choosing to write final outputs using LaTeX.[57] LaTeX is a document preparation system that can create both text documents and presentations. The

[57] https://www.latex-project.org

main difference between them is that LaTeX uses plain text, and it's necessary to learn its markup convention to use it. The main advantage of using LaTeX is that you can write dynamic documents, that import inputs every time they are compiled. This means you can skip the copying and pasting whenever an output is updated. Because it's written in plain text, it's also easier to control and document changes using Git. Creating documents in LaTeX using an integrated writing environment such as TeXstudio is great for outputs that focus mainly on text, but include small chunks of code and static code outputs. This book, for example, was written in LaTeX.

Another option is to use the statistical software's dynamic document engines. This means you can write both text (in Markdown) and code in the script, and the result will usually be a PDF or html file including code, text and code outputs. Dynamic document tools are better for including large chunks of code and dynamically created graphs and tables, but formatting can be trickier. So it's great for creating appendices, or quick document with results as you work on them, but not for final papers and reports. RMarkdown[58] is the most widely adopted solution in R. There are also different options for Markdown in Stata, such as German Rodriguez' `markstat`,[59] Stata 15 dynamic documents,[60] and Ben Jann's `webdoc`[61] and `texdoc`.[62]

Whichever options you decide to use, agree with your team on what tools will be used for what outputs, and where they will be stored before you start creating them. Take into account ease of use for different team members, but keep in mind that learning how to use a new tool may require some time investment upfront that will be paid off as your project advances.

Finally, no matter what choices you make regarding software and folder organization, you will need to make changes to your outputs quite frequently. And anyone who has tried to recreate a graph after a few months probably knows that it can be hard to remember where you saved the code that created it. Here, naming conventions and code organization play a key role in not re-writing script again and again. Use intuitive and descriptive names when you save your code. It's often desirable to have the names of your outputs and scripts linked, so, for example, `merge.do` creates `merged.dta`. Document output creation in the Master script, meaning before the line that runs a script there are a few lines of comments listing data sets and functions that are necessary for it to run, as well as all outputs created by that script. When performing data analysis, it's ideal to write one script for each output, as well as linking them through name. This means you may have a long script with "exploratory analysis", just to document everything you have tried. But as you start to export tables and graphs, you'll want to save separate scripts,

[58] https://rmarkdown.rstudio.com/

[59] https://data.princeton.edu/stata/markdown

[60] https://www.stata.com/new-in-stata/markdown/

[61] http://repec.sowi.unibe.ch/stata/webdoc/index.html

[62] http://repec.sowi.unibe.ch/stata/texdoc/index.html

where `descriptive_statistics.do` creates `descriptive_statistics.tex`.

Preparing for collaboration and replication

Choosing the right personal and team working environment can also make your work easier. Let's start looking at where you write code. If you are working in R, **RStudio** is great.[63] For Stata, the built-in do-file editor is the most widely adopted code editor, but **Atom**[64] and **Sublime**[65] can also be configured to run Stata code. Opening an entire directory and loading the whole tree view in the sidebar, which gives you access to directory management actions, is a really useful feature. This can be done using RStudio projects in RStudio, Stata projects in Stata, and directory managers in Atom and Sublime.

When it comes to collaboration software,[66] the two most common softwares in use are Dropbox and GitHub.[67] GitHub issues are a great tool for task management, and Dropbox Paper also provides a good interface with notifications. Neither of these tools require much technical knowledge; they merely require an agreement and workflow design so that the people assigning the tasks are sure to set them up in the system. GitHub is useful because tasks can clearly be tied to file versions; therefore it is useful for managing code-related tasks. It also creates incentives for writing down why changes were made as they are saved, creating naturally documented code. Dropbox Paper is useful because tasks can be easily linked to other documents saved in Dropbox; therefore it is useful for managing non-code-related tasks. Our team uses both.

[63] https://www.rstudio.com

[64] https://atom.io

[65] https://www.sublimetext.com/

[66] https://dimewiki.worldbank.org/wiki/Collaboration_Tools

[67] https://michaelstepner.com/blog/git-vs-dropbox/

Designing research for causal inference

Research design is the process of structuring field work – both experimental design and data collection – that will answer a specific research question. You don't need to be an expert in this, and there are lots of good resources out there that focus on designing interventions and evaluations. This section will present a very brief overview of the most common methods that are used, so that you can have an understanding of how to construct appropriate counterfactuals, data structures, and the corresponding code tools as you are setting up your data structure before going to the field to collect data.

You can categorize most research questions into one of two main types. There are **cross-sectional**, descriptive, and observational analyses, which seek only to describe something for the first time, such as the structure or variation of a population. We will not describe these here, because there are endless possibilities and they tend to be sector-specific. For all sectors, however, there are also causal research questions, both experimental and quasi-experimental, which rely on establishing **exogenous variation** in some input to draw a conclusion about the impact of its effect on various outcomes of interest. We'll focus on these **causal designs**, since the literature offers a standardized set of approaches, with publications and code tools available to support your work.

Counterfactuals and treatment effects

In causal analysis, a researcher is attempting to obtain estimates of a specific **treatment effect**, or the change in outcomes caused by a change in exposure to some intervention or circumstance.[68] In the potential outcomes framework, we can never observe this directly: we never see the same person in both their treated and untreated state.[69] Instead, we make inferences from samples: we try to devise a comparison group that evidence suggests would be identical to the treated group had they not been treated.

This **control group** serves as a counterfactual to the treatment group, and we compare the distributions of outcomes within each to make a computation of how different the groups are from each other. *Causal Inference* and *Causal Inference: The Mixtape* provides a detailed practical introduction to and history of each of these methods, so we will only introduce you to them very abstractly in this chapter.[70] Each of the methods described in this chapter relies on some variant of this basic strategy. In counterfactual causal analysis, the econometric models and estimating equations do not attempt to

[68] Abadie, A. and Cattaneo, M. D. (2018). Econometric methods for program evaluation. *Annual Review of Economics*, 10:465–503

[69] http://www.stat.columbia.edu/~cook/qr33.pdf

[70] https://www.hsph.harvard.edu/miguel-hernan/causal-inference-book/ http://scunning.com/cunningham_mixtape.pdf

create a predictive or comprehensive model of how the outcome of interest is generated – typically we do not care about measures of fit or predictive accuracy like R-squareds or root mean square errors. Instead, the econometric models desribed here aim to correctly describe the experimental design being used, so that the correct estimate of the difference between the treatment and control groups is obtained and can be interpreted as the effect of the treatment on outcomes.

Correctly describing the experiment means accounting for design factors such as stratification and clustering, and ensuring that time trends are handled sensibly. We aren't even going to get into regression models here. Almost all experimental designs can be accurately described as a series of between-group comparisons.[71] It means thinking carefully about how to transform and scale your data, using fixed effects to extract "within-group" comparisons as needed, and choosing estimators appropriate to your design. As the saying goes, all models are wrong, but some are useful. The models you will construct and estimate are intended to do two things: to express the intention of your research design, and to help you group the potentially endless concepts of field reality into intellectually tractable categories. In other words, these models tell the story of your research design.

[71] http://nickchk.com/econ305.html

Experimental research designs

Experimental research designs explicitly allow the research team to change the condition of the populations being studied,[72] in the form of NGO programs, government regulations, information campaigns, and many more types of interventions.[73] The classic method is the **randomized control trial (RCT)**.[74] (Not everyone agrees this is the best way to do research.[75]) There treatment and control groups are drawn from the same underlying population so that the strong condition of statistical equality in the absence of the experiment can be assumed. Three RCT-based methods are discussed here: **cross-sectional randomization** ("endline-only" studies), **difference-in-difference** ("panel-data" studies), and **regression discontinuity** ("cutoff" studies).

[72] https://dimewiki.worldbank.org/wiki/Experimental_Methods

[73] Banerjee, A. V. and Duflo, E. (2009). The experimental approach to development economics. *Annual Review of Economics*, 1(1):151–178

[74] https://dimewiki.worldbank.org/wiki/Randomized_Control_Trials

[75] https://www.nber.org/papers/w14690.pdf

Cross-sectional RCTs

Cross-sectional RCTs are the simplest possible study design: a program is implemented, surveys are conducted, and data is analyzed. The randomization process, as in all RCTs, draws the treatment and control groups from the same underlying population. This

implies the groups' outcome means would be identical in expectation before intervention, and would have been identical at measurement – therefore, differences are due to the effect of the intervention. Cross-sectional data is simple because for research teams do not need track individuals over time, or analyze attrition and follow-up other than non-response. Cross-sectional designs can have a time dimension; they are then called "repeated cross-sections", but do not imply a panel structure for individual observations.

Typically, the cross-sectional model is developed only with controls for the research design. **Balance checks**[76] can be utilized, but an effective experiment can use **stratification** (sometimes called blocking) aggressively[77] to ensure balance before data is collected.[78] Stratification disaggregates a single experiment to a collection of smaller experiments by conducting randomization within "sufficiently similar" strata groups. Adjustments for balance variables are never necessary in RCTs, because it is certain that the true data-generating process has no correlation between the treatment and the balance factors.[79] However, controls for imbalance that are not part of the design may reduce the variance of estimates, but there is debate on the importance of these tests and corrections.

[76] https://dimewiki.worldbank.org/wiki/iebaltab

[77] https://blogs.worldbank.org/impactevaluations/impactevaluations/how-randomize-using-many-baseline-variables-guest-post-thomas-barrios

[78] Athey, S. and Imbens, G. W. (2017). The econometrics of randomized experiments. *Handbook of Economic Field Experiments*, 1:73–140

[79] https://blogs.worldbank.org/impactevaluations/should-we-require-balance-t-tests-baseline-observables-randomized-experiments

Differences-in-differences

Differences-in-differences[80] (abbreviated as DD, DiD, diff-in-diff, and other variants) deals with the construction of controls differently: it uses a panel data structure to additionally use each unit in the pre-treatment phase as an additional control for itself post-treatment (the first difference), then comparing that mean change with the control group (the second difference).[81] Therefore, rather than relying entirely on treatment-control balance for identification, this class of designs intends to test whether *changes* in outcomes over time were different in the treatment group than the control group.[82] The primary identifying assumption for diff-in-diff is **parallel trends**, the idea that the change in all groups over time would have been identical in the absence of the treatment.

[80] https://dimewiki.worldbank.org/wiki/Difference-in-Differences

[81] McKenzie, D. (2012). Beyond baseline and follow-up: The case for more T in experiments. *Journal of Development Economics*, 99(2):210–221

[82] https://blogs.worldbank.org/impactevaluations/often-unspoken-assumptions-behind-difference-difference-estimator-practice

Diff-in-diff experiments therefore require substantially more effort in the field work portion, so that the **panel** of observations is well-constructed.[83] Since baseline and endline data collection may be far apart, it is important to create careful records during the first round so that follow-ups can be conducted with the same subjects, and **attrition** across rounds can be properly taken into account.[84] Depending on the distribution of results, estimates may become completely uninformative with relatively little loss to follow-up.

[83] https://www.princeton.edu/~otorres/Panel101.pdf

[84] http://blogs.worldbank.org/impactevaluations/dealing-attrition-field-experiments

The diff-in-diff model is a four-way comparison.[85] The experimental

[85] https://dimewiki.worldbank.org/wiki/ieddtab

design intends to compare treatment to control, after taking out the pre-levels for both.[86] Therefore the model includes a time period indicator, a treatment group indicator (the pre-treatment control is the base level), and it is the *interaction* of treatment and time indicators that we interpret as the differential effect of the treatment assignment.

[86] https://www.princeton.edu/~otorres/DID101.pdf

Regression discontinuity

Regression discontinuity (RD) designs differ from other RCTs in that the treatment group is not directly randomly assigned, even though it is often applied in the context of a specific experiment.[87] (In practice, many RDs are quasi-experimental, but this section will treat them as though they are designed by the researcher.) In an RD design, there is a **running variable** which gives eligible people access to some program, and a strict cutoff determines who is included.[88] This is ussally justified by budget limitations. The running variable should not be the outcome of interest, and while it can be time, that may require additional modeling assumptions. Those who qualify are given the intervention and those who don't are not; this process substitutes for explicit randomization.[89]

[87] https://dimewiki.worldbank.org/wiki/Regression_Discontinuity

[88] Lee, D. S. and Lemieux, T. (2010). Regression discontinuity designs in economics. *Journal of Economic Literature*, 48(2):281–355

[89] http://blogs.worldbank.org/impactevaluations/regression-discontinuity-porn

For example, imagine that there is a strict income cutoff created for a program that subsidizes some educational resources. Here, income is the running variable. The intuition is that the people who are "barely eligible" should not in reality be very different from those who are "barely ineligible", and that resulting differences between them at measurement are therefore due to the intervention or program.[90] For the modeling component, the **bandwidth**, or the size of the window around the cutoff to use, has to be decided and tested against various options for robustness. The rest of the model depends largely on the design and execution of the experiment.

[90] Imbens, G. W. and Lemieux, T. (2008). Regression discontinuity designs: A guide to practice. *Journal of Econometrics*, 142(2):615–635

Quasi-experimental designs

Quasi-experimental research designs,[91] are inference methods based on methods other than explicit experimentation. Instead, they rely on "experiments of nature", in which natural variation can be argued to approximate the type of exogenous variations in circumstances that a researcher would attempt to create with an experiment.[92]

[91] https://dimewiki.worldbank.org/wiki/Quasi-Experimental_Methods

[92] DiNardo, J. (2016). Natural experiments and quasi-natural experiments. *The New Palgrave Dictionary of Economics*, pages 1–12

Unlike with planned experimental designs, quasi-experimental designs typically require the extra luck of having data collected at the right times and places to exploit events that occurred in the past. Therefore, these methods often use either secondary data, or use primary data in a cross-sectional retrospective method,

applying additional corrections as needed to make the treatment and comparison groups plausibly identical.

Instrumental variables

Instrumental variables designs utilize variation in an otherwise-unrelated predictor of exposure to a treatment condition as an "instrument" for the treatment condition itself.[93] The simplest example is actually experimental – in a randomization design, we can use instrumental variables based on an *offer* to join some program, rather than on the actual inclusion in the program.[94] The reason for doing this is that the **second stage** of actual program takeup may be severely self-selected, making the group of program participants in fact wildly different from the group of non-participants.[95] The corresponding **two-stage-least-squares (2SLS)** estimator[96] solves this by conditioning on only the random portion of takeup – in this case, the randomized offer of enrollment in the program.

Unfortunately, instrumental variables designs are known to have very high variances relative to **ordinary least squares**.[97] IV designs furthermore rely on strong but untestable assumptions about the relationship between the instrument and the outcome.[98] Therefore IV designs face special scrutiny, and only the most believable designs, usually those backed by extensive qualitative analysis, are acceptable as high-quality evidence.

Matching estimators

Matching estimators rely on the assumption that, conditional on some observable characteristics, untreated units can be compared to treated units, as if the treatment had been fully randomized.[99] In other words, they assert that differential takeup is sufficiently predictable by observed characteristics. These assertions are somewhat testable,[100] and there are a large number of "treatment effect" packages devoted to standardizing reporting of various tests.[101]

However, since most matching models rely on a specific linear model, such as the typical **propensity score matching** estimator, they are open to the criticism of "specification searching", meaning that researchers can try different models of matching until one, by chance, leads to the final result that was desired. Newer methods, such as **coarsened exact matching**,[102] are designed to remove some of the modelling, such that simple differences between matched observations are sufficient to estimate treatment effects given somewhat weaker assumptions on the structure of that effect. One solution, as with the experimental variant of 2SLS proposed above, is to incorporate matching models into explicitly experimental designs.

[93] https://dimewiki.worldbank.org/wiki/instrumental_variables

[94] Angrist, J. D. and Krueger, A. B. (2001). Instrumental variables and the search for identification: From supply and demand to natural experiments. *Journal of Economic Perspectives*, 15(4):69–85

[95] http://www.rebeccabarter.com/blog/2018-05-23-instrumental_variables/

[96] http://www.nuff.ox.ac.uk/teaching/economics/bond/IV%20Estimation%20Using%20Stata.pdf

[97] Young, A. (2017). Consistency without inference: Instrumental variables in practical application. *Unpublished manuscript, London: London School of Economics and Political Science. Retrieved from: http://personal.lse.ac.uk/YoungA*

[98] Bound, J., Jaeger, D. A., and Baker, R. M. (1995). Problems with instrumental variables estimation when the correlation between the instruments and the endogenous explanatory variable is weak. *Journal of the American Statistical Association*, 90(430):443–450

[99] https://dimewiki.worldbank.org/wiki/Matching

[100] https://dimewiki.worldbank.org/wiki/iematch

[101] http://fmwww.bc.edu/repec/usug2016/drukker_uksug16.pdf

[102] Iacus, S. M., King, G., and Porro, G. (2012). Causal inference without balance checking: Coarsened exact matching. *Political Analysis*, 20(1):1–24

Synthetic controls

Synthetic controls methods[103] are designed for a particularly interesting situation: one where useful controls for an intervention simply do not exist. Canonical examples are policy changes at state or national levels, since at that scope there are no other units quite like the one that was affected by the policy change (much less sufficient N for a regression estimation).[104] In this method, **time series data** is almost always required, and the control comparison is contructed by creating a linear combination of other units such that pre-treatment outcomes for the treated unit are best approximated by that specific combination.

[103] Abadie, A., Diamond, A., and Hainmueller, J. (2015). Comparative politics and the synthetic control method. *American Journal of Political Science*, 59(2):495–510

[104] Gobillon, L. and Magnac, T. (2016). Regional policy evaluation: Interactive fixed effects and synthetic controls. *Review of Economics and Statistics*, 98(3):535–551

Sampling, randomization, and power

Sampling, randomization, and power calculations are the core elements of experimental design. **Sampling** and **randomization** determine which units are observed and in which states. Each of these processes introduces statistical noise or uncertainty into the final estimates of effect sizes. Sampling noise produces some probability of selection of units to measure that will produce significantly wrong estimates, and randomization noise produces some probability of placement of units into treatment arms that does the same. Power calculation is the method by which these probabilities of error are meaningfully assessed. Good experimental design has high **power** – a low likelihood that these noise parameters will meaningfully affect estimates of treatment effects.

Not all studies are capable of achieving traditionally high power: the possible sampling or treatment assignments may simply be fundamentally too noisy. This may be especially true for novel or small-scale studies – things that have never been tried before may be hard to fund or execute at scale. What is important is that every study includes reasonable estimates of its power, so that the evidentiary value of its results can be honestly assessed. Demonstrating that sampling and randomization were taken seriously into consideration before going to field lends credibility to any research study. Using these tools to design the most highly-powered experiments possible is a responsible and ethical use of donor and client resources, and maximizes the likelihood that reported effect sizes are accurate.

Reproducibility in sampling, randomization, and power calculation

Reproducibility in statistical programming is absolutely essential.[105] This is especially true when simulating or analyzing random processes, and sampling, randomization, and power calculation are the prime examples of these sorts of tasks. This section is a short introduction to ensuring that code which generates randomized outputs is reproducible. There are three key inputs to assuring reproducibility in these processes: **versioning**, **sorting**, and **seeding**. Without these, other people running your code may get very different results in the future.

[105] Orozco, V., Bontemps, C., Maigne, E., Piguet, V., Hofstetter, A., Lacroix, A., Levert, F., Rousselle, J.-M., et al. (2018). How to make a pie? reproducible research for empirical economics & econometrics. *Toulouse School of Economics Working Paper*, 933

Versioning means using the same version of the software. (All software versions of Stata above version 13 currently operate identically on all platforms.) If anything is different, the underlying randomization algorithms may have changed, and it will be impossible to recover the original result. In Stata, the `version` command ensures that the software algorithm is fixed. The `ieboilstart` command in `ietoolkit`

provides functionality to support this requirement.[106] In general, you will use `ieboilstart` at the beginning of your master do-file[107] to set the version once; in this guide, we will use `version 13.1` in examples where we expect this to already be done.

[106] https://dimewiki.worldbank.org/wiki/ieboilstart

[107] https://dimewiki.worldbank.org/wiki/Master_Do-files

Sorting means that the actual data that the random process is run on is fixed. Most random outcomes have as their basis an algorithmic sequence of pseudorandom numbers. This means that if the start point is set, the full sequence of numbers will not change. A corollary of this is that the underlying data must be unchanged between runs: to ensure that the dataset is fixed, you must make a `LOCKED` copy of it at runtime. However, if you re-run the process with the dataset in a different order, the same numbers will get assigned to different units, and the randomization will turn out different. In Stata, `isid [id_variable], sort` will ensure that order is fixed over repeat runs.

Seeding means manually setting the start-point of the underlying randomization algorithm. You can draw a standard seed randomly by visiting `http://bit.ly/stata-random`. You will see in the code below that we include the timestamp for verification. Note that there are two distinct concepts referred to here by "randomization": the conceptual process of assigning units to treatment arms, and the technical process of assigning random numbers in statistical software, which is a part of all tasks that include a random component.[108] If the randomization seed for the statistical software is not set, then its pseudorandom algorithm will pick up where it left off. By setting the seed, you force it to restart from the same point. In Stata, `set seed [seed]` will accomplish this.

[108] https://blog.stata.com/2016/03/10/how-to-generate-random-numbers-in-stata/

The code below code loads and sets up the `auto.dta` dataset for any random process. Note the three components: versioning, sorting, and seeding. Why are `check1` and `check3` the same? Why is `check2` different?

replicability.do

```
* Set the version
    ieboilstart , v(13.1)
    `r(version)'

* Load the auto dataset and sort uniquely
    sysuse auto.dta , clear
    isid make, sort

* Set the seed using random.org (range: 100000 - 999999)
    set seed 287608 // Timestamp: 2019-02-17 23:06:36 UTC

* Demonstrate stability under the three rules
    gen check1 = rnormal()
    gen check2 = rnormal()

    set seed 287608
    gen check3 = rnormal()

//Visualize randomization results
    graph matrix check1 check2 check3 , half
```

Commands like `bys:` and `merge` will re-sort your data as part of their execution, To reiterate: any process that includes a random component is a random process, including sampling, randomization, power calculation, and many algorithms like bootstrapping. and other commands may alter the seed without you realizing it.[109] Any of these things will cause the output to fail to replicate. Therefore, each random process should be independently executed to ensure that these three rules are followed. Before shipping the results of any random process, save the outputs of the process in a temporary location, re-run the file, and use `cf _all using [dataset]` targeting the saved file. If there are any differences, the process has not reproduced, and `cf` will return an error, as shown here.

[109] https://dimewiki.worldbank.org/wiki/Randomization_in_Stata

randomization-cf.do

```
* Make one randomization
    sysuse bpwide.dta , clear
    isid patient, sort
    version 13.1
    set seed 796683 // Timestamp: 2019-02-26 22:14:17 UTC

    sample 100

* Save for comparison
    tempfile sample
    save    `sample' , replace

* Identical randomization
    sysuse bpwide.dta , clear
    isid patient, sort
    version 13.1
    set seed 796683 // Timestamp: 2019-02-26 22:14:17 UTC

    sample 100
    cf _all using `sample'

* Do something wrong
    sysuse bpwide.dta , clear
    sort bp*
    version 13.1
    set seed 796683 // Timestamp: 2019-02-26 22:14:17 UTC

    sample 100
    cf _all using `sample'
```

Sampling

Sampling is the process of randomly selecting units of observation from a master list for survey data collection.[110] This list may be called a **sampling universe**, a **listing frame**, or something similar. We refer to it as a **master data set**[111] because it is the authoritative source for the existence and fixed characteristics of each of the units that may be surveyed.[112] If data collected in the field contradicts the master data, the master data always dominates (unless the field data is so inconsistent that a master update is necessary). Most importantly, the master data set indicates how many individuals are eligible for sampling and surveying, and therefore contains statistical information about the likelihood that each will be chosen.

The code below demonstrates how to take a uniform-probability random sample from a population using the `sample` command. More advanced sampling techniques, such as clustering and stratification, are in practice identical in implementation to the randomization section that follows – instead of creating a `treatment` variable, create a `sample` variable.

[110] https://dimewiki.worldbank.org/wiki/Sampling_%26_Power_Calculations

[111] https://dimewiki.worldbank.org/wiki/Master_Data_Set

[112] https://dimewiki.worldbank.org/wiki/Unit_of_Observation

simple-sample.do

```
/*
    Simple reproducible sampling
*/

* Set up reproducbilitiy
    ieboilstart , v(12)     // Version
    `r(version)'            // Version
    sysuse auto.dta, clear  // Load data
    isid make, sort         // Sort
    set seed 215597         // Timestamp: 2019-04-26 17:51:02 UTC

* Take a sample of 20%
    preserve
        sample 20
        tempfile sample
        save    `sample' , replace
    restore

* Merge and complete
    merge 1:1 make using `sample'
    recode _merge (3 = 1 "Sampled") (* = 0 "Not Sampled") , gen(sample)
    label var sample "Sampled"
    drop _merge

* Check
    tab sample
```

The fundamental contribution of sampling to the power of a research design is this: if you randomly sample a set number of observations from a set frame, there are a large – but fixed – number of sample sets which you may draw.[113] From any large group, you can find some possible sample sets that have higher-than-average values of some measure; similarly, you can find some sets that have lower-than-average values. The variation of these values across the range of all possible sample sets is what creates **sampling noise**, the uncertainty in statistical estimates caused by sampling.

[113] https://davegiles.blogspot.com/2019/04/what-is-permutation-test.html

Portions of this noise can be reduced through design choices such as clustering and stratification. In general, all sampling requires **inverse probability weights**. These are conceptually simple in that the weights for each individual must be precisely the inverse of the probability with which that individual is included in the sample, but may be practically difficult to calculate for complex methods. When the sampling probability is uniform, all the weights are equal to one. Sampling can be structured such that subgroups are guaranteed to appear in a sample: that is, you can pick "half the level one facilities and half the level two facilities" instead of "half of all facilities". The key here is that, *for each facility*, the probability of being chosen remains the same – 0.5. By contrast, a sampling design that chooses unbalanced proportions of subgroups has changed the probability that a given individual is included in the sample, and needs to be

reweighted in case you want to calculate overall average statistics.[114]

The sampling noise in the process that we choose determines the size of the confidence intervals for any estimates generated from that sample.[115] In general, for any underlying distribution, the Central Limit Theorem implies that the distribution of variation across the possible samples is exactly normal. Therefore, we can use what are called **asymptotic standard errors** to express how far away from the true population parameters our estimates are likely to be. These standard errors can be calculated with only two datapoints: the sample size and the standard deviation of the value in the chosen sample. The code below illustrates the fact that sampling noise has a distribution in the sense that some actual executions of the sample give estimation results far from the true value, and others give results close to it.

[114] http://blogs.worldbank.org/impactevaluations/tools-of-the-trade-when-to-use-those-sample-weights

[115] https://economistjourney.blogspot.com/2018/06/what-is-sampling-noise.html

sample-noise.do

```
* Reproducible setup: data, isid, version, seed
    sysuse auto.dta , clear
    isid make, sort
    version 13.1
    set seed 556292 // Timestamp: 2019-02-25 23:30:39 UTC

* Get true population parameter for price mean
    sum price
    local theMean = `r(mean)'

* Sample 20 units 1000 times and store the mean of [price]
    cap mat drop results              // Make matrix free
    qui forvalues i = 1/1000 {
        preserve
            sample 20 , count               // Remove count for 20%
            sum price                       // Calculate sample mean
            * Allow first run and append each estimate
            mat results = nullmat(results) \ [`r(mean)']
        restore
    }

* Load the results into memory and graph the distribution
    clear
    mat colnames results = "price_mean"
    svmat results , n(col)
    kdensity price_mean , norm xline(`theMean')
```

The output of the code is a distribution of means in sub-populations of the overall data. This distribution is centered around the true population mean, but its dispersion depends on the exact structure of the population. We use an estimate of the population variation taken from the sample to assess how far away from that true mean any given sample draw is: essentially, we estimate the properties of the distribution you see now. With that estimate, we can quantify the uncertainty in estimates due to sampling noise, calculate precisely how far away from the true mean our sample-based estimate is likely

to be, and report that as the standard error of our point estimates. The interpretation of, say, a 95% **confidence interval** in this context is that, conditional on our sampling strategy, we would anticipate that 95% of future samples from the same distribution would lead to parameter estimates in the indicated range. This approach says nothing about the truth or falsehood of any hypothesis.

Randomization

Randomization is the process of assigning units to some kind of treatment program. Many of the Stata techniques shown here can also be used for sampling, by understanding "being included in the sample" as a treatment in itself. Randomization is used to assign treatment programs in development research because it guarantees that, *on average*, the treatment will not be correlated with anything it did not cause.[116] However, as you have just seen, any random process induces noise: so does randomization. You may get unlucky and create important correlations by chance – in fact, you can almost always identify some treatment assignment that creates the appearance of statistical relationships that are not really there. This section will show you how to assess and control this **randomization noise**.

[116] Duflo, E., Glennerster, R., and Kremer, M. (2007). Using randomization in development economics research: A toolkit. *Handbook of Development Economics*, 4:3895–3962

To do that, we create a randomization **program**, which allows us to re-run the randomization method many times and assess the amount of randomization noise correctly.[117] Storing the randomization code as a program allows us to access the whole code block with a single line of code, so we can tinker with the randomization process separately from its application to the data. Programming takes a few lines of code that may be new to you, but getting used to this structure is very useful. A simple randomization program is shown below. This code randomizes observations into two groups by combining `xtile` and `recode`, which can be extended to any proportions for any number of arms.

[117] `https://data.princeton.edu/stata/programming`

randomization-program-1.do

```
* Define a randomization program
cap prog drop my_randomization
    prog def  my_randomization

    * Syntax with open options for [ritest]
    syntax , [*]
    cap drop treatment

    * Group 2/5 in treatment and 3/5 in control
    xtile group = runiform() , n(5)
    recode group (1/2=0 "Control") (3/5=1 "Treatment") , gen(treatment)
    drop group

    * Cleanup
    lab var treatment "Treatment Arm"

end
```

With this program created and executed, the next part of the code, shown below, can set up for reproducibility. Then it will call the randomization program by name, which executes the exact randomization process we programmed to the data currently loaded in memory. Having pre-programmed the exact randomization does two things: it lets us write this next code chunk much more simply, and it allows us to reuse that precise randomization as needed. Specifically, the user-written `ritest` command[118] allows us to execute a given randomization program repeatedly, visualize how the randomization might have gone differently, and calculate alternative p-values against null hypotheses. These **randomization inference**[119] significance levels may be very different than those given by asymptotic confidence intervals, particularly in small samples (up to several hundred clusters).

After generating the "true" treatment assignment, `ritest` illustrates the distribution of correlations that randomization can spuriously produce between `price` and `treatment`.

[118] http://hesss.org/ritest.pdf

[119] https://dimewiki.worldbank.org/wiki/Randomization_Inference

randomization-program-2.do

```
* Reproducible setup: data, isid, version, seed
    sysuse auto.dta , clear
    isid make, sort
    version 13.1
    set seed 107738 // Timestamp: 2019-02-25 23:34:33 UTC

* Call the program
    my_randomization
    tab treatment

* Show randomization variation with [ritest]
    ritest treatment _b[treatment] ///
      , samplingprogram(my_randomization) kdensityplot ///
      : reg price treatment
```

Clustering and stratification

To control randomization noise, we often use techniques that reduce the likelihood of a "bad draw".[120] These techniques can be used in any random process, including sampling; their implementation is nearly identical in code. We mean this in a specific way: we want to exclude randomizations with certain correlations, or we want to improve the **balance** of the average randomization draw.[121] The most common is **stratification**,[122] which splits the sampling frame into "similar" groups – **strata** – and randomizes *within* each of these groups. It is important to preserve the overall likelihood for each unit to be included, otherwise statistical corrections become necessary. For a simple stratified randomization design, it is necessary to include strata **fixed effects**, or an indicator variable for each strata, in any final regression. This accounts for the fact that randomizations were conducted within the strata, by comparing each unit to the others within its own strata.

However, manually implementing stratified randomization is prone to error: in particular, it is difficult to precisely account for the interaction of group sizes and multiple treatment arms, particularly when a given strata can contain a small number of clusters, and when there are a large number of treatment arms.[123] The user-written `randtreat` command properly implements stratification, with navigable options for handling common pitfalls. We demonstrate the use of this command here.

[120] Athey, S. and Imbens, G. W. (2017). The econometrics of randomized experiments. *Handbook of Economic Field Experiments*, 1:73–140

[121] Bruhn, M. and McKenzie, D. (2009). In pursuit of balance: Randomization in practice in development field experiments. *American Economic Journal: Applied Economics*, 1(4):200–232

[122] https://dimewiki.worldbank.org/wiki/Stratified_Random_Sample

[123] Carril, A. (2017). Dealing with misfits in random treatment assignment. *The Stata Journal*, 17(3):652–667

randtreat-strata.do

```
* Use [randtreat] in randomization program ----------------
cap prog drop my_randomization
    prog def  my_randomization

    * Syntax with open options for [ritest]
    syntax , [*]
    cap drop treatment
    cap drop strata

    * Create strata indicator
    egen strata = group(sex agegrp) , label
        label var strata "Strata Group"

    * Group 1/5 in control and each treatment
    randtreat,              ///
        generate(treatment) /// New variable name
        multiple(6)         /// 6 arms
        strata(strata)      /// 6 strata
        misfits(global)     /// Randomized altogether

    * Cleanup
    lab var treatment "Treatment Arm"
    lab def treatment 0 "Control" 1 "Treatment 1" 2 "Treatment 2" ///
        3 "Treatment 3" 4 "Treatment 4" 5 "Treatment 5" , replace
    lab val treatment treatment
end // ---------------------------------------------------

* Reproducible setup: data, isid, version, seed
    sysuse bpwide.dta , clear
    isid patient, sort
    version 13.1
    set seed 796683 // Timestamp: 2019-02-26 22:14:17 UTC

* Randomize
   my_randomization
   tab treatment strata
```

Many studies collect data at a different level of observation than the randomization unit.[124] For example, a policy may only be able to affect an entire village, but you are interested in household behavior. This type of structure is called **clustering**,[125] because the units are assigned to treatment in clusters. Because the treatments are assigned in clusters, however, there are in reality fewer randomized groups than the number of units in the data. Therefore, standard errors for clustered designs must also be clustered, at the level at which the treatment was assigned.[126]

Clustered randomization must typically be implemented manually; it typically relies on subsetting the data intelligently to the desired assignment levels. We demonstrate here.

[124] https://dimewiki.worldbank.org/wiki/Unit_of_Observation

[125] https://dimewiki.worldbank.org/wiki/Multi-stage_(Cluster)_Sampling

[126] https://blogs.worldbank.org/impactevaluations/when-should-you-cluster-standard-errors-new-wisdom-econometrics-oracle

randtreat-clusters.do

```
* Use [randtreat] in randomization program ----------------
cap prog drop my_randomization
    prog def  my_randomization

    * Syntax with open options for [ritest]
    syntax , [*]
    cap drop treatment
    cap drop cluster

    * Create cluster indicator
    egen cluster = group(sex agegrp) , label
    label var cluster "Cluster Group"

    * Save data set with all observations
    tempfile ctreat
    save  `ctreat' , replace

    * Keep only one from each cluster for randomization
    bysort cluster : keep if _n == 1

    * Group 1/2 in control and treatment in new variable treatment
    randtreat, generate(treatment) multiple(2)

    * Keep only treatment assignment and merge back to all observations
    keep cluster treatment
        merge 1:m cluster using `ctreat' , nogen

    * Cleanup
    lab var treatment "Treatment Arm"
    lab def treatment 0 "Control" 1 "Treatment" , replace
    lab val treatment treatment
end // ------------------------------------------------

* Reproducible setup: data, isid, version, seed
    sysuse bpwide.dta , clear
    isid patient, sort
    version 13.1
    set seed 796683 // Timestamp: 2019-02-26 22:14:17 UTC

* Randomize
    my_randomization
    tab cluster treatment
```

Power calculations

When we have decided on a practical sampling and randomization design, we next assess its **power**.[127] Statistical power can be described in a few ways, each of which has different uses.[128] The purpose of power calculations is not to demonstrate that a study is "strong", but rather to identify where the strengths and weaknesses of your design are located, so that readers can correctly assess the evidentiary value of any results (or null results) in the analysis. This should be done before going to the field, across the entire range of research questions your study might try to answer, so you know the relative tradeoffs you will face by changing your sampling and randomization schemes and can select your final strategy appropriately.

[127] https://dimewiki.worldbank.org/wiki/Power_Calculations_in_Stata

[128] http://www.stat.columbia.edu/~gelman/stuff_for_blog/chap20.pdf

The classic definition of power is "the likelihood that your design detects a significant treatment effect, given that there is a non-zero true effect in reality". Here we will look at two useful practical applications of that definition and show what quantitative results can be obtained. We suggest doing all power calculations by simulation; you are very unlikely to be able to determine analytically the power of your study unless you have a very simple design. Stata has some commands that can calculate power for very simple designs – `power` and `clustersampsi` – but they will not answer most of the practical questions that complex experimental designs require.

Minimum detectable effect

To determine the **minimum detectable effect**[129] – the smallest true effect that your design can detect – conduct a simulation for your actual design. The structure below uses fake data, but you should use real data whenever it is available, or you will have to make assumptions about the distribution of outcomes. If you are willing to make even more assumptions, you can use one of the built-in functions.[130]

[129] https://dimewiki.worldbank.org/wiki/Minimum_Detectable_Effect

[130] https://dimewiki.worldbank.org/wiki/Power_Calculations_in_Stata

Here, we use an outer loop to vary the size of the assumed treatment effect, which is later used to simulate outcomes in a "true" data-generating process (DGP). The data generating process is written similarly to a regression model, but it is a separate step. A data generating process is the "truth" that the regression model is trying to estimate. If our regression results are close to the DGP, then the regression is "good" in the sense we care about. For each of 100 runs indexed by `i`, we ask the question: If this DGP were true, would our design have detected it in this draw? We run our planned regression model including all controls and store the result, along with an indicator of the effect size we assumed.

When we have done this 100 times for each effect size we are interested in, we have built a large matrix of regression results. That can be loaded into data and manipulated directly, where each observation represents one possible randomization result. We flag all the runs where the p-value is significant, then visualize the proportion of significant results from each assumed treatment effect size. Knowing the design's sensitivity to a variety of effect sizes lets us calibrate whether the experiment we propose is realistic given the constraints of the amount of data we can collect.

minimum-detectable-effect.do

```
* Simulate power for different treatment effect sizes
   clear
   set matsize 5000
   cap mat drop results
   set seed 852526 // Timestamp: 2019-02-26 23:18:42 UTC

* Loop over treatment effect sizes (te)
* of 0 to 0.5 standard deviations
* in increments of 0.05 SDs
    qui forval te = 0(0.05)0.5 {
        forval i = 1/100 {      // Loop 100 times
            clear               // New simulation
            set obs 1000        // Set sample size to 1000

            * Randomly assign treatment
            * Here you could call a randomization program instead:
            gen t = (rnormal() > 0)

            * Simulate assumed effect sizes
            gen e = rnormal()       // Include a normal error term
            gen y = 1 + `te'*t + e  // Set functional form for DGP

            * Does regression detect an effect in this assignment?
            reg y t

            * Store the result
            mat a = r(table)                // Reg results
            mat a = a[....,1]               // t parameters
            mat results = nullmat(results) \ a' , [`te'] // First run and accumulate
        } // End iteration loop
    } // End incrementing effect size

* Load stored results into data
    clear
    svmat results , n(col)

* Analyze all the regressions we ran against power 80%
    gen sig = (pvalue <= 0.05) // Flag significant runs

* Proportion of significant results in each effect size group (80% power)
    graph bar sig , over(c10) yline(0.8)
```

Minimum sample size

Another way to think about the power of a design is to figure out how many observations you need to include to test various hypotheses – the **minimum sample size**. This is an important practical consideration when you are negotiating funding or submitting proposals, as it may also determine the number of treatment arms and types of hypotheses you can test. The basic structure of the simulation is the same. Here, we use the outer loop to vary the sample size, and report significance across those groups instead of across variation in the size of the effect.

minimum-sample-size.do

```
* Power for varying sample size & a fixed treatment effect
    clear
    set matsize 5000
    cap mat drop results
    set seed 510402 // Timestamp: 2019-02-26 23:19:00 UTC

* Loop over sample sizes (ss) 100 to 1000, increments of 100
    qui forval ss = 100(100)1000 {
        forval i = 1/100 {    // 100 iterations per each
            clear
            set obs `ss'      // Simulation with new sample size

            * Randomly assign treatment
            * Here you could call a randomization program instead
            gen t = (rnormal() > 0)

            * Simulate assumed effect size: here 0.2SD
            gen e = rnormal()     // Normal error term
            gen y = 1 + 0.2*t + e // Functional form for DGP

            * Does regression detect an effect in this assignment?
            reg y t

            * Store the result
            mat a = r(table)               // Reg results
            mat a = a[....,1]              // t parameters
            mat results = nullmat(results) \ a' , [`ss'] // First run and accumulate
        } // End iteration loop
    } // End incrementing sample size

* Load stored results into data
    clear
    svmat results , n(col)

* Analyze all the regressions we ran against power 80%
    gen sig = (pvalue <= 0.05) // Flag significant runs

* Proportion of significant results in each effect size group (80% power)
    graph bar sig , over(c10) yline(0.8)
```

Using the concepts of minimum detectable effect and minimum sample size in tandem can help answer a key question that typical approaches to power often do not. Namely, they can help you determine what tradeoffs to make in the design of your experiment. Can you support another treatment arm? Is it better to add another cluster, or to sample more units per cluster? Simultaneously, planning out the regression structure in advance saves a lot of work once the data is in hand, and helps you think critically about what you are really testing. It also helps you to untangle design issues before they occur. Therefore, simulation-based power analysis is often more of a design aid than an output for reporting requirements. At the end of the day, you will probably have reduced the complexity of your experiment significantly. For reporting purposes, such as grantwriting and registered reports, simulation ensures you will

have understood the key questions well enough to report standard measures of power once your design is decided.

Collecting primary data

Most data collection is now done using digital data entry using tools that are specially designed for surveys. These tools, called **computer-assisted personal interviewing (CAPI)** software, provide a wide range of features designed to make implementing even highly complex surveys easy, scalable, and secure. However, these are not fully automatic: you still need to actively design and manage the survey. Each software has specific practices that you need to follow to enable features such as Stata-compatibility and data encryption.

You can work in any software you like, and this guide will present tools and workflows that are primarily conceptual: this chapter should provide a motivation for planning data structure during survey design, developing surveys that are easy to control for quality and security, and having proper file storage ready for sensitive PII data.

Designing CAPI questionnaires

CAPI surveys[131] are primarily created in Excel, Google Sheets, or software-specific form builders making them one of the few research outputs for which little coding is required.[132] The `ietestform` command,[133] part of `iefieldkit`, implements form-checking routines for **SurveyCTO**, a proprietary implementation of **Open Data Kit (ODK)**. However, since they make extensive use of logical structure and relate directly to the data that will be used later, both the field team and the data team should collaborate to make sure that the survey suits all needs.[134]

Generally, this collaboration means building the experimental design fundamentally into the structure of the survey. In addition to having prepared a unique anonymous ID variable using the master data, that ID should be built into confirmation checks in the survey form. When ID matching and tracking across rounds is essential, the survey should be prepared to verify new data against **preloaded data** from master records or from other rounds. **Extensive tracking** sections – in which reasons for **attrition**, treatment **contamination**, and **loss to follow-up** can be documented – are essential data components for completing CONSORT[135] records.[136]

Questionnaire design[137] is the first task where the data team and the field team must collaborate on data structure.[138] Questionnaire

[131] https://dimewiki.worldbank.org/wiki/Computer-Assisted_Personal_Interviews_(CAPI)

[132] https://dimewiki.worldbank.org/wiki/SurveyCTO_Coding_Practices

[133] https://dimewiki.worldbank.org/wiki/ietestform

[134] Krosnick, J. A. (2018). Questionnaire design. In *The Palgrave Handbook of Survey Research*, pages 439–455. Springer

[135] **CONSORT:** a standardized system for reporting enrollment, intervention allocation, follow-up, and data analysis through the phases of a randomized trial of two groups.

[136] Begg, C., Cho, M., Eastwood, S., Horton, R., Moher, D., Olkin, I., Pitkin, R., Rennie, D., Schulz, K. F., Simel, D., et al. (1996). Improving the quality of reporting of randomized controlled trials: The CONSORT statement. *JAMA*, 276(8):637–639

[137] https://dimewiki.worldbank.org/wiki/Questionnaire_Design

[138] https://dimewiki.worldbank.org/wiki/Preparing_for_Field_Data_Collection

design should always be a separated task from questionnaire programming, meaning that the team should have designed and agreed on a questionnaire before that questionnaire is programmed in the CAPI software used. Otherwise teams tend to spend more time discussing technical programming details, rather than the content in the questionnaire. The field-oriented staff and the PIs will likely prefer to capture a large amount of detailed *information* in the field, some of which will serve very poorly as *data*.[139] In particular, **open-ended responses** and questions which will have many null or missing responses by design will not be very useful in statistical analyses unless pre-planned. You must work with the field team to determine the appropriate amount of abstraction inherent in linking concepts to responses.[140] For example, it is always possible to ask for open-ended responses to questions, but it is far more useful to ask for things like **Likert scale**[141] responses instead of asking, for instance, "How do you feel about the proposed policy change?"

[139] https://iriss.stanford.edu/sites/g/files/sbiybj6196/f/questionnaire_design_1.pdf

[140] https://www.povertyactionlab.org/sites/default/files/documents/Instrument%20Design_Diva_final.pdf

[141] **Likert scale:** an ordered selection of choices indicating the respondent's level of agreement or disagreement with a proposed statement.

Coded responses are always more useful than open-ended responses, because they reduce the time necessary for post-processing by expensive specialized staff. For example, if collecting data on medication use or supplies, you could collect: the brand name of the product; the generic name of the product; the coded compound of the product; or the broad category to which each product belongs (antibiotic, etc.). All four may be useful for different reasons, but the latter two are likely to be the most useful for the analyst. The coded compound requires providing a translation dictionary to field staff, but enables automated rapid recoding for analysis with no loss of information. The generic class requires agreement on the broad categories of interest, but allows for much more comprehensible top-line statistics and data quality checks.

Broadly, the questionnaire should be designed as follows. The workflow will feel much like writing an essay: begin from broad concepts and slowly flesh out the specifics. The **theory of change**, **experimental design**, and any **pre-analysis plans** should be discussed and the structure of required data for those conceptualized first.[142] Next, the conceptual outcomes of interest, as well as the main covariates, classifications, and other variables needed for the experimental design should be listed out. The questionnaire *modules* should be outlined based on this list. At this stage, modules should not be numbered – they should use a short prefix so they can be easily reordered. Each module should then be expanded into specific indicators to observe in the field. There is not yet a full consensus over how individual questions should be identified: formats like `hq_1` are hard to remember and unpleasant to reorder, but formats like `hq_asked_about_loans` quickly become cumbersome. Finally,

[142] https://dimewiki.worldbank.org/wiki/Literature_Review_for_Questionnaire

the questionnaire can be **piloted**[143] in a non-experimental sample. Revisions are made, and the survey is then translated into the appropriate language and programmed electronically.[144]

[143] https://dimewiki.worldbank.org/wiki/Survey_Pilot

[144] https://dimewiki.worldbank.org/wiki/Questionnaire_Programming

Collecting data securely

At all points in the data collection and handling process, access to personally-identifying information (PII) must always be restricted to the members of the team who have been granted that permission by the appropriate agreement (typically an IRB approving data collection or partner agency providing it). Any established data collection platform will always **encrypt**[145] all data submitted from the field automatically while in transit (i.e., upload or download), so if you use servers hosted by SurveyCTO or SurveySolutions this is nothing you need to worry about. Your data will be encrypted from the time it leaves the device (in tablet-assisted data collation) or your browser (in web data collection) until it reaches the server.

[145] **Encryption:** the process of making information unreadable to anyone without access to a specific deciphering key. https://dimewiki.worldbank.org/wiki/Encryption

Encryption at rest is the only way to ensure that PII data remains private when it is stored on someone else's server on the open internet. Encryption makes data files completely unusable without access to a security key specific to that data – a higher level of security than password-protection. The World Bank's and many of our donors' security requirements for data storage can only be fulfilled by this method. We recommend keeping your data encrypted whenever PII data is collected – therefore, we recommend it for all field data collection.

Encryption in cloud storage, by contrast, is not enabled by default. This is because the service will not encrypt user data unless you confirm you know how to operate the encryption system and understand the consequences if basic protocols are not followed. Encryption at rest is different from password-protection: encryption at rest makes the underlying data itself unreadable, even if accessed, except to users who have a specific private **keyfile**. Encryption at rest requires active participation from you, the user, and you should be fully aware that if your private key is lost, there is absolutely no way to recover your data.

To enable data encryption in SurveyCTO, for example, simply select the encryption option when you create a new form on a SurveyCTO server. Other data collection services should work similarly, but the exact implementation of encryption at rest will vary slightly from service to service. At that time, the service will allow you to download – once – the keyfile pair needed to decrypt the data. You must download and store this in a secure location. Make sure you store keyfiles with descriptive names to match the

survey to which it corresponds. Any time you access the data - either when viewing it in browser or syncing it to your computer - the user will be asked to provide this keyfile. It could differ between the software, but typically you would copy that keyfile from where it is stored (for example LastPass) to your desktop, point to it, and the rest is automatic. After each time you use the keyfile, delete it from your desktop, but not from your password manager.

Finally, you should ensure that all teams take basic precautions to ensure the security of data, as most problems are due to human error. Most importantly, all computers, tablets, and accounts used *must* have a logon password associated with them. Ideally, the machine hard drives themselves should also be encrypted. This policy should also be applied to physical data storage such as flash drives and hard drives; similarly, files sent to the field containing PII data such as the sampling list should at least be password-protected. This can be done using a zip-file creator. LastPass can also be used to share passwords securely, and you cannot share passwords across email. This step significantly mitigates the risk in case there is a security breach such as loss, theft, hacking, or a virus, and adds very little hassle to utilization.

Overseeing fieldwork and quality assurance

While the team is in the field, the research assistant and field coordinator will be jointly responsible for making sure that the survey is progressing correctly,[146] that the collected data matches the survey sample, and that errors and duplicate observations are resolved quickly so that the field team can make corrections.[147] Modern survey software makes it relatively easy to control for issues in individual surveys, using a combination of in-built features such as hard constraints on answer ranges and soft confirmations or validation questions.

These features allow you to spend more time looking for issues that the software cannot check automatically.[148] Namely, these will be suspicious patterns across multiple responses or across a group of surveys rather than errors in any single response field (those can often be flagged by the questionnaire software); enumerators who are taking either too long or not long enough to complete their work, "difficult" groups of respondents who are systematically incomplete; and systematic response errors. These are typically done in two main forms: high-frequency checks (HFCs) and back-checks.

High-frequency checks are carried out on the data side.[149] First, observations need to be checked for duplicate entries: `ieduplicates`[150] provides a workflow for collaborating on the resolution of duplicate entries between you and the field team. Next, observations must be

[146] https://dimewiki.worldbank.org/wiki/Data_Quality_Assurance_Plan

[147] https://dimewiki.worldbank.org/wiki/Duplicates_and_Survey_Logs

[148] https://dimewiki.worldbank.org/wiki/Monitoring_Data_Quality

[149] https://github.com/PovertyAction/high-frequency-checks/wiki

[150] https://dimewiki.worldbank.org/wiki/ieduplicates

validated against the sample list: this is as straightforward as `merging` the sample list with the survey data and checking for mismatches. High-frequency checks should carefully inspect key treatment and outcome variables so that the data quality of core experimental variables is uniformly high, and that additional field effort is centered where it is most important. Finally, data quality checks should be run on the data every time it is downloaded to flag irregularities in observations, sample groups, or enumerators. `ipacheck`[151] is a very useful command that automates some of these tasks.

[151] https://github.com/PovertyAction/high-frequency-checks

Unfortunately, it is very hard to specify in general what kinds of quality checks should be utilized, since the content of surveys varies so widely. Fortunately, you will know your survey very well by the time it is programmed, and should have a good sense of the types of things that would raise concerns that you were unable to program directly into the survey. Thankfully it is also easy to prepare high-frequency checking code in advance. Once you have built and piloted the survey form, you will have some bits of test data to play with. Using this data, you should prepare code that outputs a list of flags from any given dataset. This HFC code is then ready to run every time you download the data, and should allow you to rapidly identify issues that are occurring in fieldwork. You should also have a plan to address issues found with the field team. Discuss in advance which inconsistencies will require a revisit, and how you will communicate to the field teams what were the problems found. Some issues will need immediate follow-up, and it will be harder to solve them once the enumeration team leaves the area.

Back-checks[152] involve more extensive collaboration with the field team, and are best thought of as direct data audits. In back-checks, a random subset of the field sample is chosen and basic information from the full survey is verified using a small survey re-done with the same respondent. Back-checks should be done with a substantial portion of the full sample early in the survey so that the enumerators and field team get used to the idea of **quality assurance**. Checks should continue throughout fieldwork, and their content and targeting can be refined if particular questionnaire items are flagged as error-prone or specific enumerators or observations appear unusual.

[152] https://dimewiki.worldbank.org/wiki/Back_Checks

Back-checks cover three main types of questions. First, they validate basic information that should not change. This ensures the basic quality control that the right respondent was interviewed or observed in a given survey, and flags cases of outright quality failure that need action. Second, they check the quality of enumeration, particularly in cases that involve measurement or calculation on the part of the enumerator. This can be anything such as correctly

calculating plot sizes, family rosters, or income measurements. These questions should be carefully validated to determine whether they are reliable measures and how much they may vary as a result of difficulty in measurement. Finally, back-checks confirm that questions are being asked and answered in a consistent fashion. Some questions, if poorly phrased, can be hard for the enumerator to express or for all respondents to understand in an identical fashion. Changes in responses between original and back-check surveys of the same respondent can alert you and the team that changes need to be made to portions of the survey. `bcstats` is a Stata command for back checks that takes the different question types into account when comparing surveys.

When all data collection is complete, the survey team should have a final field report ready for validation against the sample frame and the dataset. This should contain all the observations that were completed; it should merge perfectly with the received dataset; and it should report reasons for any observations not collected. Reporting of **missingness** and **attrition** is critical to the interpretation of any survey dataset, so it is important to structure this reporting such that broad rationales can be grouped into categories but also that the field team can provide detailed, open-ended responses for any observations they were unable to complete. This reporting should be validated and saved alongside the final raw data, and treated the same way. This information should be stored as a dataset in its own right – a **tracking dataset** – that records all events in which survey substitutions and loss to follow-up occurred in the field and how they were implemented and resolved.

Receiving primary data

In this section, you finally get your hands on some data! What do we do with it? Data handling is one of the biggest "black boxes" in primary research – it always gets done, but teams have wildly different approaches for actually doing it. This section breaks the process into key conceptual steps and provides at least one practical solution for each. Initial receipt of data will proceed as follows: the data will be downloaded, and a "gold master" copy of the raw data should be permanently stored in a secure location. Then, a "master" copy of the data is placed into an encrypted location that will remain accessible on disk and backed up. This handling satisfies the rule of three: there are two on-site copies of the data and one off-site copy, so the data can never be lost in case of hardware failure. For this step, the remote location can be a variety of forms: the cheapest is a long-term cloud storage service such as Amazon Web Services or

Microsoft Azure. Equally sufficient is a physical hard drive stored somewhere other than the primary work location (and encrypted with a service like BitLocker To Go). If you remain lucky, you will never have to access this copy – you just want to know it is out there, safe, if you need it.

The copy of the raw data you are going to use should be handled with care. Since you will probably need to share it among the team, it should be placed in an encrypted storage location, although the data file itself may or may not need to be encrypted. Enterprise cloud solutions like Microsoft OneDrive can work as well. If the service satisfies your security needs, the raw data can be stored unencrypted here. Placing encrypted data (such as with VeraCrypt) into an unencrypted cloud storage location (such as Dropbox) may also satisfy this requirement for some teams, since this will never make the data visible to someone who gets access to the Dropbox, without the key to the file that is generated on encryption. *The raw data file must never be placed in Dropbox unencrypted, however.* The way VeraCrypt works is that it creates a virtual copy of the unencrypted file outside of Dropbox, and lets you access that copy. Since you should never edit the raw data, this will not be very cumbersome, but the decryption and usage of the raw data is a manual process. With the raw data securely stored and backed up, you are ready to move to de-identification, data cleaning, and analysis.

Analyzing survey data

Data analysis is hard. Making sense of a dataset in such a way that makes a substantial contribution to scientific knowledge requires a mix of subject expertise, programming skills, and statistical and econometric knowledge. The process of data analysis is therefore typically a back-and-forth discussion between the various people who have differing experiences, perspectives, and research interests. The research assistant usually ends up being the fulcrum for this discussion, and has to transfer and translate results among people with a wide range of technical capabilities while making sure that code and outputs do not become tangled and lost over time (typically months or years).

Organization is the key to this task. The structure of files needs to be well-organized, so that any material can be found when it is needed. Data structures need to be organized, so that the various steps in creating datasets can always be traced and revised without massive effort. The structure of version histories and backups need to be organized, so that different workstreams can exist simultaneously and experimental analyses can be tried without a complex workflow. Finally, the outputs need to be organized, so that it is clear what results go with what analyses, and that each individual output is a readable element in its own right. This chapter outlines how to stay organized so that you and the team can focus on getting the work right rather than trying to understand what you did in the past.

Organizing primary survey data

There are many schemes to organize primary survey data. We provide the `iefolder`[153] package (part of `ietoolkit`[154]) so that a large number of teams will have identical folder structures. This means that PIs and RAs face very small costs when switching between projects, because all the materials will be organized in exactly the same basic way.[155] Namely, within each survey round folder,[156] there will be dedicated folders for raw (encrypted) data; for de-identified data; for cleaned data; and for final (constructed) data. There will be a folder for raw results, as well as for final outputs. The folders that hold code will be organized in parallel to these, so that the progression through the whole project can be followed by anyone new to the team. Additionally, `iefolder` provides **master do-files**[157] so that the entire order of the project is maintained in a top-level dofile, ensuring that no complex instructions are needed to exactly replicate the project.

Master do-files are equally important as a tool to allow all the

[153] https://dimewiki.worldbank.org/wiki/iefolder

[154] https://dimewiki.worldbank.org/wiki/ietoolkit

[155] https://dimewiki.worldbank.org/wiki/DataWork_Folder

[156] https://dimewiki.worldbank.org/wiki/DataWork_Survey_Round

[157] https://dimewiki.worldbank.org/wiki/Master_Do-files

analysis to be executed from one do-file that runs all other files in the correct order, as a human readable map to the file and folder structure used for all the code. By reading the master do-file anyone not familiar to the project should understand which are the main tasks, what are the do-files that execute those tasks and where in the project folder they can be found.

Raw data should contain only materials that are received directly from the field. These datasets will invariably come in a host of file formats and nearly always contain personally-identifying information. These should be retained in the raw data folder *exactly as they were received*, including the precise filename that was submitted, along with detailed documentation about the source and contents of each of the files. This data must be encrypted if it is shared in an insecure fashion, and it must be backed up in a secure offsite location. Everything else can be replaced, but raw data cannot. Therefore, raw data should never be interacted with directly.

Instead, the first step upon receipt of data is **de-identification**.[158] There will be a code folder and a corresponding data folder so that it is clear how de-identification is done and where it lives. Typically, this process only involves stripping fields from raw data, naming, formatting, and optimizing the file for storage, and placing it as a `.dta` or other data-format file into the de-identified data folder. The underlying data structure is unchanged: it should contain only fields that were collected in the field, without any modifications to the responses collected there. This creates a survey-based version of the file that is able to be shared among the team without fear of data corruption or exposure. Only a core set of team members will have access to the underlying raw data necessary to re-generate these datasets, since the encrypted raw data will be password-protected. The de-identified data will therefore be the underlying source for all cleaned and constructed data. It will also become the template dataset for final public release.

[158] https://dimewiki.worldbank.org/wiki/De-identification

Cleaning and constructing derivative datasets

Once the de-identified dataset is created, data must be **cleaned**.[159] This process does not involve the creation of any new data fields. Cleaning can be a long process: this is the phase at which you obtain an extensive understanding of the contents and structure of the data you collected in the field. You should use this time to understand the types of responses collected, both within each survey question and across respondents. Understanding this structure will make it possible to do analysis.

[159] https://dimewiki.worldbank.org/wiki/Data_Cleaning

There are a number of tasks involved in cleaning data. The

`iecodebook` command suite, part of `iefieldkit`, is designed to make some of the most tedious components of this process, such as renaming, relabeling, and value labeling, much easier (including in data appending).[160] A data cleaning do-file will load the de-identified data, make sure all the fields are named and labelled concisely and descriptively, apply corrections to all values in the dataset, make sure value labels on responses are accurate and concise, and attach any experimental-design data (sampling and randomization) back to the clean dataset before saving. It will ensure that ID variables[161] are correctly structured and matching, ensure that data storage types are correctly set up (including tasks like dropping open-ended responses and encoding strings), and make sure that data missingness is appropriately documented using other primary inputs from the field.[162]

[160] https://dimewiki.worldbank.org/wiki/iecodebook

[161] https://dimewiki.worldbank.org/wiki/ID_Variable_Properties

[162] https://dimewiki.worldbank.org/wiki/Data_Documentation

Clean data then becomes the basis for constructed (final) datasets. While data cleaning typically takes one survey dataset and mixes it with other data inputs to create a corresponding cleaned version of that survey data (a one-to-one process), **data construction** is a much more open-ended process. Data construction files mix-and-match clean datasets to create a large number of potential analysis datasets. Data construction is the time to create derived variables (binaries, indices, and interactions, to name a few), to reshape data as necessary; to create functionally-named variables, and to sensibly subset data for analysis.

Data construction is never a finished process. It comes "before" data analysis only in a limited sense: the construction code must be run before the analysis code. Typically, however it is the case that the construction and analysis code are written concurrently – both do-files will be open at the same time. It is only in the process of writing the analysis that you will come to know which constructed variables are necessary, and which subsets and alterations of the data are desired.

You should never attempt to create a "canonical" analysis dataset. Each analysis dataset should be purpose-built to answer an analysis question. It is typical to have many purpose-built analysis datasets: there may be a `data-wide.dta`, `data-wide-children-only.dta`, `data-long.dta`, `data-long-counterfactual.dta`, and many more as needed. Data construction should never be done in analysis files, and since much data construction is specific to the planned analysis, organizing analysis code into many small files allows that level of specificity to be introduced at the correct part of the code process. Then, when it comes time to finalize analysis files, it is substantially easier to organize and prune that code since there is no risk of losing construction code it depends on.

Writing data analysis code

Data analysis is the stage of the process to create research outputs. Many introductions to common code skills and analytical frameworks exist: *R for Data Science*;[163] *A Practical Introduction to Stata*;[164] *Mostly Harmless Econometrics*;[165] and *Causal Inference: The Mixtape*.[166] This section will not include instructions on how to conduct specific analyses, as these are highly specialized and require field and statistical expertise. Instead, we will outline the structure of writing analysis code, assuming you have completed the process of data cleaning and construction. As mentioned above, typically you will continue to keep the data-construction files open, and add to them as needed, while writing analysis code.

[163] https://r4ds.had.co.nz/

[164] https://scholar.harvard.edu/files/mcgovern/files/practical_introduction_to_stata.pdf

[165] https://www.researchgate.net/publication/51992844_Mostly_Harmless_Econometrics_An_Empiricist's_Companion

[166] http://scunning.com/mixtape.html

It is important to ignore the temptation to write lots of analysis into one big, impressive, start-to-finish analysis file. This is counterproductive because it subtly encourages poor practices such as not clearing the workspace or loading fresh data. Every output table or figure should have a completely fresh workspace and load its data directly prior to running that specific analysis. This encourages data manipulation to be done upstream (in construction), and prevents you from accidentally writing pieces of code that depend on each other, leading to the too-familiar "run this part, then that part, then this part" process. Each output should be able to run completely independently of all other code sections. You can go as far as coding every output in a separate file. There is nothing wrong with code files being short and simple – as long as they directly correspond to specific published tables and figures.

To accomplish this, you will need to make sure that you have an effective system of naming, organization, and version control. Version control allows you to effectively manage the history of your code, including tracking the addition and deletion of files. This lets you get rid of code you no longer need simply by deleting it in the working directory, and allows you to recover those chunks easily if you ever need to get back previous work. Therefore, version control supports a tidy workspace. (It also allows effective collaboration in this space, but that is a more advanced tool not covered here.)

Organizing your workspace effectively is now essential. `iefolder` will have created a `/Dofiles/Analysis/` folder; you should feel free to add additional subfolders as needed. It is completely acceptable to have folders for each task, and compartmentalize each analysis as much as needed. It is always better to have more code files open than to have to scroll around repeatedly inside a given file. Just like you named each of the analysis datasets, each of the individual analysis files should be descriptively named. Code files such as `spatial-`

`diff-in-diff.do`, `matching-villages.do`, and `summary-statistics.do` are clear indicators of what each file is doing and allow code to be found very quickly. Eventually, you will need to compile a "release package" that links the outputs more structurally to exhibits, but at this stage function is more important than structure.

Outputs should, whenever possible, be created in lightweight formats. For graphics,[167] `.eps` or `.tif` files are acceptable; for tables,[168] `.tex` is preferred (don't worry about making it really nicely formatted until you are ready to publish). Excel `.xlsx` files are also acceptable, although if you are working on a large report they will become cumbersome to update after revisions. If you are writing your code in Git/GitHub, you should output results into that directory, not the directory where data is stored (i.e. Dropbox). Results will differ across different branches and will constantly be re-writing each other: you should have each result stored with the code that created it.

[167] https://dimewiki.worldbank.org/wiki/Checklist:_Reviewing_Graphs

[168] https://dimewiki.worldbank.org/wiki/Checklist:_Submit_Table

Visualizing data

Data visualization[169] is increasingly popular, but a great deal of it is very low quality.[170] The default graphics settings in Stata, for example, are pretty bad.[171] Thankfully, graphics tools like Stata are highly customizable. There is a fair amount of learning curve associated with extremely-fine-grained adjustment, but it is well worth reviewing the graphics manual to understand how to apply basic colors, alignments, and labelling tools[172] The manual is worth skimming in full, as it provides many visual examples and corresponding code examples that detail how to produce them.

Graphics are hard to get right because you, as the analyst, will already have a pretty good idea of what you are trying to convey. Someone seeing the illustration for the first time, by contrast, will have no idea what they are looking at. Therefore, a visual image has to be compelling in the sense that it shows that there is something interesting going on and compels the reader to figure out exactly what it is. A variety of resources can help you figure out how best to structure a given visual. The **Stata Visual Library**[173] has many examples of figures that communicate effectively. The Tapestry conference focuses on "storytelling with data".[174] *Fundamentals of Data Visualization* provides extensive details on practical application;[175] as does *Data Visualization: A Practical Introduction*.[176]

[169] https://dimewiki.worldbank.org/wiki/Data_visualization

[170] Healy, K. (2018). *Data visualization: A practical introduction*. Princeton University Press; and Wilke, C. O. (2019). *Fundamentals of Data Visualization: A Primer on Making Informative and Compelling Figures*. O'Reilly Media

[171] Gray Kimbrough's *Uncluttered Stata Graphs* code is an excellent default replacement that is easy to install. https://graykimbrough.github.io/uncluttered-stata-graphs/

[172] https://www.stata.com/manuals/g.pdf

[173] https://worldbank.github.io/Stata-IE-Visual-Library/

[174] https://www.youtube.com/playlist?list=PLb0GkPPcZCVE9EAm9qhlg5eXMgLrrfMRq

[175] https://serialmentor.com/dataviz

[176] http://socvis.co

Publishing collaborative research

Increasingly, research assistants are relied on to manage some or all of the publication process. This can include managing the choice of software, coordinating referencing and bibliography, tracking changes across various authors and versions, and preparing final reports or papers for release or submission. Modern software tools can make a lot of these processes easier. Unfortunately there is some learning curve, particularly for lead authors who have been publishing for a long time. This chapter suggests some tools and processes that can make writing and publishing in a team significantly easier. It will provide resources to judge how best to adapt your team to the tools you agree upon, since all teams vary in composition and technical experience.

Ideally, your team will spend as little time as possible fussing with the technical requirements of publication. It is in nobody's interest for a skilled and busy research assistant to spend days re-numbering references (and it can take days) if a small amount of up-front effort could automate the task. However, experienced academics will likely have a workflow with which they are already comfortable, and since they have worked with many others in the past, that workflow is likely to be the least-common-denominator: Microsoft Word with tracked changes. This chapter will show you how you can avoid at least some of the pain of Microsoft Word, while still providing materials in the format that co-authors prefer and journals request.

Collaborating on academic writing

The gold standard for academic writing is LaTeX.[177] LaTeX allows automatically-organized sections like titles and bibliographies, imports tables and figures in a dynamic fashion, and can be version controlled using Git. Unfortunately, LaTeX can be a challenge to set up and use at first, particularly for people who are unfamiliar with plaintext, code, or file management. LaTeX requires that all formatting be done in its special code language, and it is not particularly informative when you do something wrong. This can be off-putting very quickly for people who simply want to get to writing, like lead authors. Therefore, if we want to take advantage of the features of LaTeX, without getting stuck in the weeds of it, we will need to adopt a few tools and tricks to make it effective.

[177] https://www.maths.tcd.ie/~dwilkins/LaTeXPrimer/GSWLaTeX.pdf

The first is choice of software. The easiest way for someone new to LaTeX to be able to "just write" is often the web-based Overleaf suite.[178] Overleaf offers a **rich text editor** that behaves pretty similarly

[178] https://www.overleaf.com

to familiar tools like Word. TeXstudio[179] and atom-latex[180] are two popular desktop-based tools for writing LaTeX; they allow more advanced integration with Git, among other advantages, but the entire team needs to be comfortable with LaTeX before adopting one of these tools. With minimal workflow adjustments, you can to show coauthors how to write and edit in Overleaf, so long as you make sure you are always available to troubleshoot LaTeX crashes and errors. The most common issue will be special characters, namely `&`, `%`, and `_`, which need to be **escaped** (instructed to interpret literally) by writing a backslash (`\`) before them, such as `40\%` for the percent sign to function. Overleaf offers a convenient selection of templates so it is easy to start up a project and replicate a lot of the underlying setup code. The main issue with using Overleaf is that you need to upload input files (such as figures and tables) manually. This can create conflicts when these inputs are still being updated – namely, the main document not having the latest results. One solution is to move to Overleaf only once there will not be substantive changes to results.

[179] https://www.texstudio.org

[180] https://atom.io/packages/atom-latex

One of the important tools available in LaTeX is the BibTeX bibliography manager.[181] This tool stores unformatted references in an accompanying `.bib` file, and LaTeX then inserts them in text using the `\cite{}` command. With this structure, LaTeX automatically pulls all the citations into text. LaTeX allows you to specify how they should be displayed in text (ie, as superscripts, inline references, etc.) and how the bibliography should be styled and in what order. A reference in the `.bib` file can be copied directly from Google Scholar by clicking "BibTeX" at the bottom of the Cite window. When pasted into the `.bib` file they look like the following:

[181] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.365.3194&rep=rep1&type=pdf

sample.bib

```
@article{flom2005latex,
  title={LATEX for academics and researchers who (think they) don't need it},
  author={Flom, Peter},
  journal={The PracTEX Journal},
  volume={4},
  year={2005},
  publisher={Citeseer}
}
```

BibTeX citations are then used as follows:

citation.tex

```
With these tools, you can ensure that co-authors are writing
in a format you can manage and control.\cite{flom2005latex}
```

With these tools, you can ensure that co-authors are writing in a format you can manage and control.[182] The purpose of this setup, just like with other synced folders, is to avoid there ever being more than one master copy of the document. This means that people can edit simultaneously without fear of conflicts, and it is never necessary to manually resolve differences in the document. Finally, LaTeX has one more useful trick: if you download a journal styler from the **Citation Styles Library**[183] and use **`pandoc`**,[184] you can translate the raw document into Word by running the following code from the command line:

[182] Flom, P. (2005). LaTeX for academics and researchers who (think they) don't need it. *The PracTEX Journal*, 4

[183] https://github.com/citation-style-language/styles

[184] http://pandoc.org/

pandoc.sh

```
pandoc -s -o main.docx main.tex --bibliography sample.bib --csl=[style].csl
```

Therefore, even in the case where you are requested to provide `.docx` versions of materials to others, or tracked-changes versions, you can create them effortlessly, and use Word's compare feature to generate a single integrated tracked version.

Publishing data and code for replication

Data and code should always be released with any publication.[185] Many journals and funders have strict open data policies, and providing these materials to other researchers allows them to evaluate the credibility of your work as well as to re-use your materials for further research.[186] If you have followed the steps in this book carefully, you will find that this is a very easy requirement to fulfill.[187] You will already have a publishable (either de-identified or constructed) version of your research dataset. You will already have your analysis code well-ordered, and you won't have any junk lying around for cleanup. You will have written your code so that others can read it, and you will have documentation for all your work, as well as a well-structured directory containing the code and data.

[185] https://dimewiki.worldbank.org/wiki/Publishing_Data

[186] https://dimewiki.worldbank.org/wiki/Exporting_Analysis

[187] https://www.bitss.org/2016/05/23/out-of-the-file-drawer-tips-on-prepping-data-for-publication/

If you are at this stage, all you need to do is find a place to publish your work! GitHub provides one of the easiest solutions here, since it is completely free for static, public projects and it is straightforward to simply upload a fixed directory and obtain a permanent URL for it. The Open Science Framework also provides a good resource, as does ResearchGate (which can also assign a permanent digital object identifier link for your work). Any of these locations is acceptable – the main requirement is that the system can handle the structured directory that you are submitting, and that it can provide a stable,

structured URL for your project.

You should release a structured directory that allows a user to immediately run your code after changing the project directory. The folders should include: all necessary de-identified data for the analysis (including only the data needed for analysis); the data documentation for the analysis code (describing the source and construction of variables); the ready-to-run code necessary for the analysis; and the raw outputs you have used for the paper. Using `iefolder` from our `ietoolkit` can help standardize this in Stata. In either the `/dofiles/` folder or in the root directory, include a master script (`.do` or `.r` for example). The master script should allow the reviewer to change one line of code setting the directory path. Then, running the master script should run the entire project and re-create all the raw outputs exactly as supplied. Check that all your code will run completely on a new computer. This means installing any required user-written commands in the master script (for example, in Stata using `ssc install` or `net install` and in R include code for installing packages, including installing the appropriate version of the package if necessary). Make sure settings like `version`, `matsize`, and `varabbrev` are set.[188] All outputs should clearly correspond by name to an exhibit in the paper, and vice versa.

[188] In Stata, `ietoolkit`'s `ieboilstart` command will do this for you.

Finally, you may also want to release an author's copy or preprint. Check with your publisher before doing so; not all journals will accept material that has been released. Therefore you may need to wait until acceptance is confirmed. This can be done on a number of pre-print websites, many of which are topically-specific. You can also use GitHub and link the file directly on your personal website or whatever medium you are sharing the preprint through. Do not use Dropbox or Google Drive for this purpose: many organizations do not allow access to these tools, and that includes blocking staff from accessing your material.

Appendix: The DIME Analytics Stata style guide

Most academic programs that prepare students for a career in the type of work discussed in this book spend a disproportionately small amount of time teaching their students coding skills, in relation to the share of their professional time they will spend writing code their first years after graduating. Recent Masters' program graduates that have joined our team tended to have very good knowledge in the theory of our trade, but tended to require a lot of training in its practical skills. To us, it is like hiring architects that can sketch, describe, and discuss the concepts and requirements of a new building very well, but do not have the technical skill set to actually contribute to a blueprint using professional standards that can be used and understood by other professionals during construction. The reasons for this are probably a topic for another book, but in today's data-driven world, people working in quantitative economics research must be proficient programmers, and that includes more than being able to compute the correct numbers.

This appendix first has a short section with instructions on how to access and use the code shared in this book. The second section contains a the current DIME Analytics style guide for Stata code. Widely accepted and used style guides are common in most programming languages, and we think that using such a style guide greatly improves the quality of research projects coded in Stata. We hope that this guide can help to increase the emphasis in the Stata community on using, improving, sharing and standardizing code style. Style guides are the most important tool in how you, like an architect, draw a blueprint that can be understood and used by everyone in your trade.

Using the code examples in this book

You can access the raw code used in examples in this book in many ways. We use GitHub to version control everything in this book, the code included. To see the code on GitHub, go to: `https://github.com/worldbank/d4di/tree/master/code`. If you are familiar with GitHub you can fork the repository and clone your fork. We only use Stata's built-in datasets in our code examples, so you do not need to download any data from anywhere. If you have Stata installed on your computer, then you will have the data files used in the code.

A less technical way to access the code is to click the individual file in the URL above, then click the button that says **Raw**. You will then get to a page that looks like the one at: `https://raw.githubusercontent.com/worldbank/d4di/master/code/code.do`. There, you can copy the code from your browser window to your do-file editor with the formatting intact. This method is only practical

for a single file at the time. If you want to download all code used in this book, you can do that at: https://github.com/worldbank/d4di/archive/master.zip. That link offers a .zip file download with all the content used in writing this book, including the LaTeX code used for the book itself. After extracting the .zip-file you will find all the code in a folder called /code/.

Understanding Stata code

Regardless if you are new to Stata or have used it for decades, you will always run into commands that you have not seen before or do not remember what they do. Every time that happens, you should always look that command up in the helpfile. For some reason, we often encounter the conception that the helpfiles are only for beginners. We could not disagree with that conception more, as the only way to get better at Stata is to constantly read helpfiles. So if there is a command that you do not understand in any of our code examples, for example `isid`, then write `help isid`, and the helpfile for the command `isid` will open.

We cannot emphasize too much how important we think it is that you get into the habit of reading helpfiles.

Sometimes, you will encounter code employing user-written commands, and you will not be able to read those helpfiles until you have installed the commands. Two examples of these in our code are `reandtreat` or `ieboilstart`. The most common place to distribute user-written commands for Stata is the Boston College Statistical Software Components (SSC) archive. In our code examples, we only use either Stata's built-in commands or commands available from the SSC archive. So, if your installation of Stata does not recognize a command in our code, for example `randtreat`, then type `ssc install randtreat` in Stata.

Some commands on SSC are distributed in packages, for example `ieboilstart`, meaning that you will not be able to install it using `ssc install ieboilstart`. If you do, Stata will suggest that you instead use `findit ieboilstart` which will search SSC (among other places) and see if there is a package that has a command called `ieboilstart`. Stata will find `ieboilstart` in the package `ietoolkit`, so then you will type `ssc install ietoolkit` instead in Stata.

We understand that this can be confusing the first time you work with this, but this is the best way to set up your Stata installation to benefit from other people's work that they have made publicly available, and once used to installing commands like this it will not be confusing at all. All code with user-written commands, furthermore, is best written when it installs such commands at

the beginning of the master do-file, so that the user does not have to search for packages manually.

Why we use a Stata style guide

Programming languages used in computer science always have style guides associated with them. Sometimes they are official guides that are universally agreed upon, such as PEP8 for Python.[189] More commonly, there are well-recognized but non-official style guides like the JavaScript Standard Style[190] for JavaScript or Hadley Wickham's[191] style guide for R.

Aesthetics is an important part of style guides, but not the main point. The existence of style guides improves the quality of the code in that language produced by all programmers in the community. It is through a style guide that unexperienced programmers can learn from more experienced programmers how certain coding practices are more or less error-prone. Broadly-accepted style guides make it easier to borrow solutions from each other and from examples online without causing bugs that might only be found too late. Similarly, globally standardized style guides make it easier to solve each others' problems and to collaborate or move from project to project, and from team to team.

There is room for personal preference in style guides, but style guides are first and foremost about quality and standardization – especially when collaborating on code. We believe that a commonly used Stata style guide would improve the quality of all code written in Stata, which is why we have begun the one included here. You do not necessarily need to follow our style guide precisely. We encourage you to write your own style guide if you disagree with us. The best style guide woud be the one adopted the most widely. What is most important is that you adopt a style guide and follow it consistently across your projects.

[189] https://www.python.org/dev/peps/pep-0008/

[190] https://standardjs.com/#the-rules

[191] http://adv-r.had.co.nz/Style.html

The DIME Analytics Stata style guide

While this section is called a *Stata* Style Guide, many of these practices are agnostic to which programming language you are using: best practices often relate to concepts that are common across many languages. If you are coding in a different language, then you might still use many of the guidelines listed in this section, but you should use your judgment when doing so. All style rules introduced in this section are the way we suggest to code, but the most important thing is that the way you style your code is *consistent*. This guide allows our team to have a consistent code style.

Commenting code

Comments do not change the output of code, but without them, your code will not be accessible to your colleagues. It will also take you a much longer time to edit code you wrote in the past if you did not comment it well. So, comment a lot: do not only write *what* your code is doing but also *why* you wrote it like that.

There are three types of comments in Stata and they have different purposes:

1. `/* */` indicates narrative, multi-line comments at the beginning of files or sections.
2. `*` indicates a change in task or a code sub-section and should be multi-line only if necessary.
3. `//` is used for inline clarification a single line of code.

stata-comments.do

```
/*
    This is a do-file with examples of comments in Stata. This
    type of comment is used to document all of the do-file or a large
    section of it
*/

* Standardize settings (This comment is used to document a task
* covering at maximum a few lines of code)
    ieboilstart, version(13.1)
    `r(version)'

* Open the dataset
    sysuse auto.dta // Built in dataset (This comment is used to document a single line)
```

Abbreviating commands

Stata commands can often be abbreviated in the code. In the helpfiles you can tell if a command can be abbreviated, indicated by the

part of the name that is underlined in the syntax section at the top. Only built-in commands can be abbreviated; user-written commands can not. Although Stata allows some commands to be abbreviated to one or two characters, this can be confusing – two-letter abbreviations can rarely be "pronounced" in an obvious way that connects them to the functionality of the full command. Therefore, command abbreviations in code should not be shorter than three characters, with the exception of `tw` for `twoway` and `di` for `display`, and abbreviations should only be used when widely accepted abbreviation exists. We do not abbreviate `local`, `global`, `save`, `merge`, `append`, or `sort`. Here is our non-exhaustive list of widely accepted abbreviations of common Stata commands.

Abbreviation	Command
`tw`	`twoway`
`di`	`display`
`gen`	`generate`
`mat`	`matrix`
`reg`	`regress`
`lab`	`label`
`sum`	`summarize`
`tab`	`tabulate`
`bys`	`bysort`
`qui`	`quietly`
`cap`	`capture`
`forv`	`forvalues`
`prog`	`program`

Writing loops

In Stata examples and other code languages, it is common that the name of the local generated by `foreach` or `forvalues` is named something as simple as `i` or `j`. In Stata, however, loops generally index a real object, and looping commands should name that index descriptively. One-letter indices are acceptable only for general examples; for looping through **iterations** with `i`; and for looping across matrices with `i`, `j`. Other typical index names are `obs` or `var` when looping over observations or variables, respectively. But since Stata does not have arrays such abstract syntax should not be used in Stata code otherwise. Instead, index names should describe what the code is looping over, for example household members, crops, or medicines. This makes code much more readable, particularly in nested loops.

stata-loops.do

```
* This is BAD
    foreach i in potato cassava maize {
    }

* These are GOOD
    foreach crop in potato cassava maize {
    }

    * or

    local crops potato cassava maize
    * Loop over crops
    foreach crop of local crops {
        * Loop over plot number
        forvalues plot_num = 1/10 {
        }
    }
```

Using whitespace

In Stata, one space or many spaces does not make a difference, and this can be used to make the code much more readable. In the example below the exact same code is written twice, but in the good example whitespace is used to signal to the reader that the central object of this segment of code is the variable `employed`. Organizing the code like this makes the code much quicker to read, and small typos stand out much more, making them easier to spot.

We are all very well trained in using whitespace in software like PowerPoint and Excel: we would never present a PowerPoint presentation where the text does not align or submit an Excel table with unstructured rows and columns, and the same principles apply to coding.

stata-whitespace-columns.do

```
* This is BAD
    * Create dummy for being employed
    generate employed = 1
    replace employed = 0 if (_merge == 2)
    label variable employed "Person exists in employment data"
    label define yesno 1 "Yes" 0 "No"
    label value employed yesno

* This is GOOD
    * Create dummy for being employed
    generate       employed = 1
    replace        employed = 0 if (_merge == 2)
    label variable employed "Person exists in employment data"
    label define            yesno 1 "Yes" 0 "No"
    label value    employed yesno
```

Indentation is another type of whitespace that makes code

more readable. Any segment of code that is repeated in a loop or conditional on an if-statement should have indentation of 4 spaces relative to both the loop or conditional statement as well as the closing curly brace. Similarly, continuing lines of code should be indented under the initial command. If a segment is in a loop inside a loop, then it should be indented another 4 spaces, making it 8 spaces more indented than the main code. In some code editors this indentation can be achieved by using the tab button on your keyboard. However, the type of tab used in the Stata do-file editor does not always display the same across platforms, such as when publishing the code on GitHub. Therefore we recommend that indentation be 4 manual spaces instead of a tab.

stata-whitespace-indentation.do

```
* This is GOOD
    * Loop over crops
    foreach crop in potato cassava maize {
        * Loop over plot number
        forvalues plot_num = 1/10 {
            gen crop_`crop'_`plot_num' = "`crop'"
        }
    }

    * or
    local sampleSize = `c(N)'
    if (`sampleSize' <= 100) {
        gen use_sample = 0
    }
    else {
        gen use_sample = 1
    }

* This is BAD
    * Loop over crops
    foreach crop in potato cassava maize {
    * Loop over plot number
    forvalues plot_num = 1/10 {
    gen crop_`crop'_`plot_num' = "`crop'"
    }
    }

    * or
    local sampleSize = `c(N)'
    if (`sampleSize' <= 100) {
    gen use_sample = 0
    }
    else {
    gen use_sample = 1
    }
```

Writing conditional expressions

All conditional (true/false) expressions should be within at least one set of parentheses. The negation of logical expressions should use

bang (`!`) and not tilde (`~`).

stata-conditional-expressions1.do

```
* These examples are GOOD
    replace gender_string = "Female" if (gender == 1)
    replace gender_string = "Male"   if ((gender != 1) & !missing(gender))

* These examples are BAD
    replace gender_string = "Female" if gender == 1
    replace gender_string = "Male"   if (gender ~= 1)
```

You should also always use `if-else` statements when applicable even if you can express the same thing with two separate `if` statements. When using `if-else` statements you are communicating to anyone reading your code that the two cases are mutually exclusive in an `if-else` statement which makes your code more readable. It is also less error-prone and easier to update.

stata-conditional-expressions2.do

```
   local sampleSize = _N // Get the number of observations in dataset

* This example is GOOD
    if (`sampleSize' <= 100) {
    }
    else {
    }

* This example is BAD
    if (`sampleSize' <= 100) {
    }
    if (`sampleSize' > 100) {
    }
```

Using macros

Stata has several types of **macros** where numbers or text can be stored temporarily, but the two most common macros are **local** and **global**. Locals should always be the default type and globals should only be used when the information stored is used in a different do-file. Globals are error-prone since they are active as long as Stata is open, which creates a risk that a global from one project is incorrectly used in another, so only use globals where they are necessary. Our recommendation is that globals should only be defined in the **master do-file**. All globals should be referenced using both the the dollar sign and the curly brackets around their name; otherwise, they can cause readability issues when the endpoint of the macro name is unclear.

There are several naming conventions you can use for macros with long or multi-word names. Which one you use is not as important as whether you and your team are consistent in how you name then. You can use all lower case (`mymacro`), underscores (`my_macro`), or "camel case" (`myMacro`), as long as you are consistent.

stata-macros.do

```
* Define a local and a global using the same name convention
    local  myLocal  "A string local"
    global myGlobal "A string global"

* Reference the local and the global macros
    display "`myLocal'"
    display "${myGlobal}"

* Escape character. If backslashes are used just before a local
* or a global then two backslashes must be used
    local myFolderLocal  "Documents"
    local myFolderGlobal "Documents"

* These are BAD
    display "C:\Users\username\`myFolderLocal'"
    display "C:\Users\username\${myFolderGlobal}"

* These are GOOD
    display "C:\Users\username\\`myFolderLocal'"
    display "C:\Users\username\\${myFolderGlobal}"
```

Writing file paths

All file paths should be absolute and dynamic, should always be enclosed in double quotes, and should always use forward slashes for folder hierarchies (`/`), since Mac and Linux computers cannot read file paths with backslashes. File paths should also always include the file extension (`.dta`, `.do`, `.csv`, etc.), since to omit the extension causes ambiguity if another file with the same name is created (even if there is a default).

Absolute file paths means that all file paths must start at the root folder of the computer, for example `C:/` on a PC or `/Users/` on a Mac. This makes sure that you always get the correct file in the correct folder. **We never use `cd`.** We have seen many cases when using `cd` where a file has been overwritten in another project folder where `cd` was currently pointing to. Relative file paths are common in many other programming languages, but there they are relative to the location of the file running the code, and then there is no risk that a file is saved in a completely different folder. Stata does not provide this functionality.

Dynamic file paths use globals that are set in a central master do-file to dynamically build your file paths. This has the same function

in practice as setting `cd`, as all new users should only have to change these file path globals in one location. But dynamic absolute file paths are a better practice since if the global names are set uniquely there is no risk that files are saved in the incorrect project folder, and you can create multiple folder globals instead of just one location as with `cd`.

stata-filepaths.do

```
* Dynamic, absolute file paths

    * Dynamic (and absolute) - GOOD
    global myDocs    "C:/Users/username/Documents"
    global myProject "${myDocs}/MyProject"
    use "${myProject}/MyDataset.dta"

* Relative and absolute file paths

    * Relative - BAD
    cd "C:/Users/username/Documents/MyProject"
    use MyDataset.dta

    * Absolute but not dynamic - BAD
    use "C:/Users/username/Documents/MyProject/MyDataset.dta"
```

Placing line breaks

Long lines of code are difficult to read if you have to scroll left and right to see the full line of code. When your line of code is wider than text on a regular paper you should introduce a line break. A common line breaking length is around 80 characters, and Stata and other code editors provide a visible "guide line" to tell you when you should start a new line using `///`. This breaks the line in the code editor while telling Stata that the same line of code continues on the next row in the code editor. Using the `#delimit` command is only intended for advanced programming and is discouraged for analytical code in an article in Stata's official journal.[192] These do not need to be horizontally aligned in code unless they have comments, since indentations should reflect that the command continues to a new line. Line breaks and indentations can similarly be used to highlight the placement of the **option comma** in Stata commands.

[192] Cox, N. J. (2005). Suggestions on stata programming style. *The Stata Journal*, 5(4):560–566

stata-linebreak.do

```
* This is GOOD
    graph hbar ///
        invil if (priv == 1) ///
      , over(statename, sort(1) descending) blabel(bar, format(%9.0f)) ///
        ylab(0 "0%" 25 "25%" 50 "50%" 75 "75%" 100 "100%") ///
        ytit("Share of private primary care visits made in own village")

* This is BAD
    #delimit ;
    graph hbar
        invil if (priv == 1)
      , over(statename, sort(1) descending) blabel(bar, format(%9.0f))
        ylab(0 "0%" 25 "25%" 50 "50%" 75 "75%" 100 "100%")
        ytit("Share of private primary care visits made in own village");
    #delimit cr
```

Using boilerplate code

Boilerplate code is a type of code that always comes at the top of the code file, and its purpose is to harmonize the settings across users running the same code to the greatest degree possible. There is no way in Stata to guarantee that any two installations of Stata will always run code in exactly the same way. In the vast majority of cases it does, but not always, and boilerplate code can mitigate that risk (although not eliminate it). We have developed a command that runs many commonly used boilerplate settings that are optimized given your installation of Stata. It requires two lines of code to execute the `version` setting that avoids difference in results due to different versions of Stata.

stata-boilerplate.do

```
* This is GOOD
    ieboilstart, version(13.1)
    `r(version)'

* This is GOOD but less GOOD
    set more off
    set maxvar 10000
    version 13.1
```

Saving data

Similarly to boilerplate code, there are good practices that should be followed before saving the data set. These are sorting and ordering the data set, dropping intermediate variables that are not needed, and compressing the data set to save disk space and network bandwidth.

If there is an ID variable or a set of ID variables, then the code

should also test that they are uniqueally and fully identifying the data set.[193] ID variables are also perfect variables to sort on, and to order leftmost in the data set.

[193] https://dimewiki.worldbank.org/wiki/ID_Variable_Properties

The command `compress` makes the data set smaller in terms of memory usage without ever losing any information. It optimizes the storage types for all variables and therefore makes it smaller on your computer and faster to send over a network or the internet.

stata-before-saving.do

```
* If the data set has ID variables, create a local and test
* if they are fully and uniquely identifying the observations.
local idvars household_ID household_member year
isid `idvars'

* Sort and order on the idvars (or any other variables if there are no ID variables)
sort  `idvars'
order * , seq // Place all variables in alphanumeric order (optional but useful)
order `idvars' , first // Make sure the idvars are the leftmost vars when browsing

* Drop intermediate variables no longer needed

* Optimize disk space
compress

* Save data settings
save "${myProject}/myDataFile.dta" , replace // The folder global is set in master do-file
 use "${myProject}/myDataFile.dta" , clear // It is useful to be able to recall the data quickly
```

Bibliography

Abadie, A. and Cattaneo, M. D. (2018). Econometric methods for program evaluation. *Annual Review of Economics*, 10:465–503.

Abadie, A., Diamond, A., and Hainmueller, J. (2015). Comparative politics and the synthetic control method. *American Journal of Political Science*, 59(2):495–510.

Angrist, J., Azoulay, P., Ellison, G., Hill, R., and Lu, S. F. (2017). Economic research evolves: Fields and styles. *American Economic Review*, 107(5):293–97.

Angrist, J. D. and Krueger, A. B. (2001). Instrumental variables and the search for identification: From supply and demand to natural experiments. *Journal of Economic Perspectives*, 15(4):69–85.

Angrist, J. D. and Pischke, J.-S. (2010). The credibility revolution in empirical economics: How better research design is taking the con out of econometrics. *Journal of Economic Perspectives*, 24(2):3–30.

Athey, S. and Imbens, G. W. (2017). The econometrics of randomized experiments. *Handbook of Economic Field Experiments*, 1:73–140.

Banerjee, A. V. and Duflo, E. (2009). The experimental approach to development economics. *Annual Review of Economics*, 1(1):151–178.

Bound, J., Jaeger, D. A., and Baker, R. M. (1995). Problems with instrumental variables estimation when the correlation between the instruments and the endogenous explanatory variable is weak. *Journal of the American Statistical Association*, 90(430):443–450.

Bruhn, M. and McKenzie, D. (2009). In pursuit of balance: Randomization in practice in development field experiments. *American Economic Journal: Applied Economics*, 1(4):200–232.

Carril, A. (2017). Dealing with misfits in random treatment assignment. *The Stata Journal*, 17(3):652–667.

Christensen, G. and Miguel, E. (2018). Transparency, reproducibility, and the credibility of economics research. *Journal of Economic Literature*, 56(3):920–80.

Cox, N. J. (2005). Suggestions on stata programming style. *The Stata Journal*, 5(4):560–566.

Dafoe, A. (2014). Science deserves better: the imperative to share complete replication files. *PS: Political Science & Politics*, 47(1):60–66.

DiNardo, J. (2016). Natural experiments and quasi-natural experiments. *The New Palgrave Dictionary of Economics*, pages 1–12.

Duflo, E., Glennerster, R., and Kremer, M. (2007). Using randomization in development economics research: A toolkit. *Handbook of Development Economics*, 4:3895–3962.

Duvendack, M., Palmer-Jones, R., and Reed, W. R. (2017). What is meant by "replication" and why does it encounter resistance in economics? *American Economic Review*, 107(5):46–51.

Flom, P. (2005). LaTeX for academics and researchers who (think they) don't need it. *The PracTEX Journal*, 4.

Gobillon, L. and Magnac, T. (2016). Regional policy evaluation: Interactive fixed effects and synthetic controls. *Review of Economics and Statistics*, 98(3):535–551.

Healy, K. (2018). *Data visualization: A practical introduction*. Princeton University Press.

Iacus, S. M., King, G., and Porro, G. (2012). Causal inference without balance checking: Coarsened exact matching. *Political Analysis*, 20(1):1–24.

Imbens, G. W. and Lemieux, T. (2008). Regression discontinuity designs: A guide to practice. *Journal of Econometrics*, 142(2):615–635.

Ioannidis, J. P. (2005). Why most published research findings are false. *PLoS Medicine*, 2(8):e124.

Ioannidis, J. P., Stanley, T. D., and Doucouliagos, H. (2017). The power of bias in economics research. *The Economic Journal*.

Krosnick, J. A. (2018). Questionnaire design. In *The Palgrave Handbook of Survey Research*, pages 439–455. Springer.

Lee, D. S. and Lemieux, T. (2010). Regression discontinuity designs in economics. *Journal of Economic Literature*, 48(2):281–355.

Levitt, S. D. and List, J. A. (2009). Field experiments in economics: The past, the present, and the future. *European Economic Review*, 53(1):1–18.

Matthews, G. J., Harel, O., et al. (2011). Data confidentiality: A review of methods for statistical disclosure limitation and methods for assessing privacy. *Statistics Surveys*, 5:1–29.

McKenzie, D. (2012). Beyond baseline and follow-up: The case for more T in experiments. *Journal of Development Economics*, 99(2):210–221.

Olken, B. A. (2015). Promises and perils of pre-analysis plans. *Journal of Economic Perspectives*, 29(3):61–80.

Orozco, V., Bontemps, C., Maigne, E., Piguet, V., Hofstetter, A., Lacroix, A., Levert, F., Rousselle, J.-M., et al. (2018). How to make a pie? reproducible research for empirical economics & econometrics. *Toulouse School of Economics Working Paper*, 933.

Rogers, A. (2017). The dismal science remains dismal, say scientists. *Wired*.

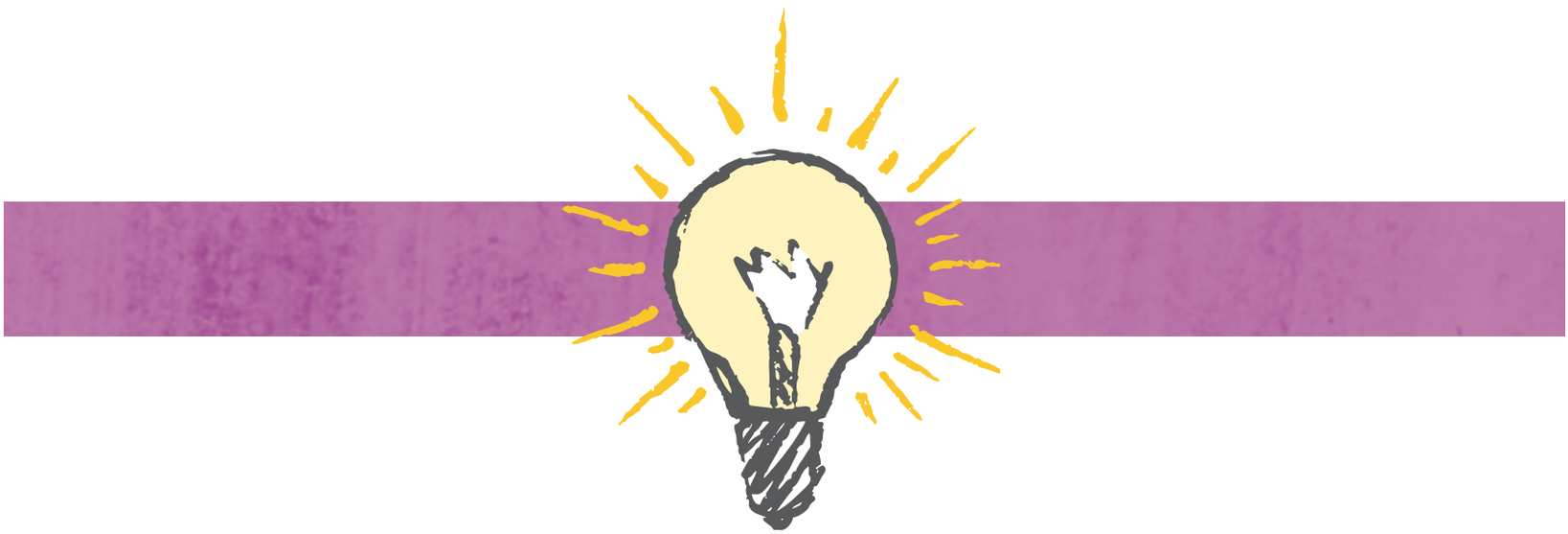
Simmons, J. P., Nelson, L. D., and Simonsohn, U. (2011). False-positive psychology: Undisclosed flexibility in data collection and analysis allows presenting anything as significant. *Psychological Science*, 22(11):1359–1366.

Simonsohn, U., Nelson, L. D., and Simmons, J. P. (2014). P-curve: a key to the file-drawer. *Journal of Experimental Psychology: General*, 143(2):534.

Wicherts, J. M., Veldkamp, C. L., Augusteijn, H. E., Bakker, M., Van Aert, R., and Van Assen, M. A. (2016). Degrees of freedom in planning, running, analyzing, and reporting psychological studies: A checklist to avoid p-hacking. *Frontiers in Psychology*, 7:1832.

Wilke, C. O. (2019). *Fundamentals of Data Visualization: A Primer on Making Informative and Compelling Figures*. O'Reilly Media.

Young, A. (2017). Consistency without inference: Instrumental variables in practical application. *Unpublished manuscript, London: London School of Economics and Political Science. Retrieved from: http://personal.lse.ac.uk/YoungA*.



WORLD BANK GROUP

i2i DIME
TRANSFORM DEVELOPMENT

DIME analytics