How to make a pie: Reproductible research for empirical economics and econometrics

Valérie Orozco, Christophe Bontemps, Élise Maigné, Virginie Piguet, Annie Hofstetter, Anne Lacroix, Fabrice Levert, Jean-marc Rousselle

How To Make A Pie: Reproducible Research for Empirical Economics and Econometrics

V. Orozco[*], C. Bontemps[*], E. Maigné[†], V. Piguet[‡], A. Hofstetter[§], A. Lacroix[¶], F. Levert[‖], J.M. Rousselle[§]

July, 2020

Abstract

Empirical economics and econometrics (EEE) research now relies primarily on the application of code to data sets. Handling the workflow that links data sets, programs, results and finally manuscript(s) is essential if one wishes to reproduce results. Herein, we highlight the importance of "reproducible research" in EEE and propose three simple principles to follow: organize your work, code for others and automate as much as you can. The first principle, "organize your work", deals with the overall organization of files and the documentation of a research workflow. "Code for others" emphasizes that we should take care in how we write code that has to be read by others or later by our future self. Finally, "automate as much as you can" is a proposal to avoid any manual treatment and to automate most, if not all, of the steps used in a research process to reduce errors and increase reproducibility. As software is not always the problem and will never be the solution, we illustrate these principles with good habits and tools, with a particular focus on their implementation in most popular software and languages in applied economics.

Keywords. Empirical Economics; Literate Programming; Replication; Reproducibility; Software ; Workflow

[*]Toulouse School of Economics, INRAE, University of Toulouse Capitole, Toulouse, France
[†]Observatoire du Développement Rural, INRAE, Toulouse, France
[‡]CESAER, Agrosup Dijon, INRAE, Université Bourgogne Franche-Comté, Dijon, France
[§]CEE-M, Univ Montpellier, CNRS, INRAE, Montpellier SupAgro, Montpellier, France
[¶]Université Grenoble Alpes, INRAE, CNRS, Grenoble INP, GAEL, Grenoble, France
[‖]SMART-LERECO, INRAE, Rennes, France

1 Introduction

Economists love to use the metaphor of a pie to illustrate some basic economic concepts. However, we must acknowledge that very few among us describe our work — from the beginning to final publication — as clearly as any pie recipe in any cookbook. In the best case, when supplementary material is available for a published paper, we end up with a detailed list of ingredients, such as the source of the data, their contents or the data sets themselves and some code.[1] However, the recipe used for a paper is typically not entirely known to the reader (see Chang & Li, 2017). First, either data or codes (the ingredients) are often unavailable. Second, there is frequently little or no documentation of some stages of the research process, and the documents available (the recipes) are insufficient to understand all stages. Third, even when the code is available, it does not always work, possibly because of errors, incompleteness or inaccuracy of the version provided. One might think that the cook forgot some crucial elements such as when to do what and in what order. Fourth, in some cases, the code works but does not yield the same results (perhaps the code provided is not the final code used by the authors). Even to the cook himself, the cooking process can be fuzzy and poorly recorded. Thus, the pie may taste different, depending on the cook. Finally, in other cases, the code works but is too difficult to understand. To cope with these situations, some readers write to the authors, asking for the code or explanations. However, as with cooks, some authors simply do not share their recipes. It is therefore quite difficult to replicate a paper, even when one possesses the main ingredients and follows the model description or empirical strategy reported in the published paper.[2]

Like anyone, researchers do make mistakes. Dewald *et al.* (1988) and McCullough *et al.* (2006) suggest that the frequency of inadvertent errors in published articles is not low.[3] A few years ago, the so-called "Reinhart & Rogoff case" shed light on many issues closely related to a lack of reproducibility in an important research process. The paper was published in one of the most selective journals, namely, the *American Economic Review*, and the results have been used to provide insights for governments from across the globe tempted to pursue debt reduction, at the cost of an austerity policy (Reinhart & Rogoff, 2010). A controversy emerged when, four years later, Herndon *et al.* (2014) revealed that "*selective exclusion of available data, coding errors and inappropriate weighting of summary statistics lead to serious miscalculations*" in the original paper, which motivated them to refute Reinhart & Rogoff's main results.

[1] In this article, we will use the terms program, code and script interchangeably.

[2] Chang & Li (2017) attempt to replicate 67 papers published in 13 well-regarded economics journals using author-provided files. They obtain data and code replication files for 29 of 35 papers (83%). They successfully replicate the key qualitative results of 22 of 67 papers (33%) without contacting the author and replicate 29 of 59 papers (49%) with assistance from the authors. They conclude that *"economics research is usually not replicable"*.

[3] We are not addressing here ethical problems or falsification, even if such issues have become prominent in recent decades. Martinson *et al.* (2005) show that 25% of scientists admitted to having fabricated, falsified or modified data or results at least once, while *"the number of retraction notices has shot up 10-fold, even as the literature has expanded by only 44%"* (Van Noorden, 2011).

These previously unnoticed weaknesses and errors may have been avoided if the code had been available for checking by referees, other researchers or students. The lessons of this symptomatic case, although clearly stated by the scientific community (see, e.g., Höffler & Kneib, 2013), have not been fully learned, even though some initiatives have recently emerged. First, although few economic journals have an online archive of data and/or code and/or strict rules for ensuring replicability, this situation is slowly changing.[4] Indeed, Appendix A shows that increasingly more journals are leading the way toward improving data and code availability online and impose conditions related to such availability prior to final publication. Some journals have a tighter replication policy and even check the provided material during the editorial process (*e.g.*, *American Journal of Political Science*, *American Economic Association*, see Vilhuber *et al.*, 2020). Other journals have very recently appointed data editors (*e.g.*, *Review of Economic Studies*, *Economic Journal*, *Canadian Journal of Economics*, and so on) in order to promote and improve articles' reproducibility. However, journal's policies may not ensure reproducibility of the research because either the shared numerical materials are provided with a low quality (McCullough, 2018) or because of human or resources constraints. Christensen & Miguel (2018) note that a large share of published papers has a data exemption, mostly for confidential reasons. Even if this might be an obstacle to the enforcement of those policies, it shouldn't refrain good practices.

Second, platforms such as *ExecAndShare* (Hurlin *et al.*, 2014a,b) and *Code Ocean* (Staniland, 2018; Stata News, 2018) have recently been created and allow online replication of papers. There is also a replication wiki in economics (Höffler, 2017).[5] *ScienceDirect* also presents some "executable papers" (Gorp & Mazanek, 2011), while some journals, such as the *Journal of Applied Econometrics*, have a replication section (Pesaran, 2003). New journals such as *ReScience* (Rougier *et al.*, 2017) or the *International Journal for Re-Views in Empirical Economics* (IREE), which was founded in 2017 and is the first journal that intends to publish replication studies based on microeconomic data, are now devoted to the publication of computational replications. However, the path toward an academic world in which all published papers in empirical economics and econometrics (EEE) would be replicable is still not well paved, and there are still many voluntary or involuntary reasons for the nonreproducibility of published papers in EEE (Duvendack *et al.*, 2017).

We argue that the replication of published results, whatever the definition of replication used, should be recognized as an essential part of the scientific method. We will provide more details on these definitions in Section 2. We agree with Huschka (2013): "*Only results that can be replicated are truly scientific results. If there is no chance to replicate research results, they can be regarded as no more than personal views in the opinion or review section of a daily newspaper*".

[4] According to McCullough (2009) and Vlaeminck & Herrmann (2015), 7 journals had such a policy in 2009, while 29 (out of 141) did in 2012.

[5] This wiki is a database of replication articles (published or working papers) and promotes reproducible research. Each replication study is described on a page that reports the availability of raw data, type and degree of replication, a link to the replication article, and so on.

We believe that scientific journals and institutions financing research, such as the National Science Foundation (USA), and European Research Council (Europe), as well as universities and research centers, should at least provide clear incentives or, even better, impose minimal mandatory data and code policies for funded research programs. At the individual level, any researcher, when acting as a referee, should ask for data and code at the early stages of the publication process so that the article can be evaluated in light of its sources. While firms develop quality management programs and engage in ISO certification, universities barely promote reproducible practices; provide examples; and teach those principles, methods and tools in their doctoral programs (Höffler, 2013).

Improving our working habits towards more reproducible research is a long-term endeavor that will require the commitment of researchers, journals, and research institutions and that will naturally take time and continuous efforts. Opponents to reproducible research argue that these practices are time consuming, too constraining, and not worth the small benefit, if any (Donoho *et al.*, 2009). We do not deny the induced cost by a change in working habits, but we briefly enumerate some of the benefits of applying a reproducible research approach when conducting research in EEE. The first benefit goes directly to researchers (Desquilbet *et al.*, 2019). Gentzkow & Shapiro (2014) observe that, in empirical research, we repeat tasks that we have already done or that others may have already done. Writing reusable and shareable code improves our own efficiency, participates in a global improvement of efficiency for communities using the same practices (Stodden *et al.*, 2013) and is thus prone to accelerate research (McCullough, 2009). This process is particularly beneficial for young researchers and students that could access reproducible examples and programs, eventually improving them in an "iterative refinement" (Baiocchi, 2007) and treating new questions. It has also been shown that sharing all the materials used in a paper may increase the impact of the article, the authors' notoriety and emphasizes the authors' numerical and computational skills (Gleditsch & Metelits, 2003). Another invaluable benefit comes from an increase in the credibility of the research and in the public trust in science (Leek & Peng, 2015).

The objectives of this paper are threefold:

First, we propose three main principles to guide the actions of any researcher in EEE toward a higher degree of reproducibility of their work. The research process has, in recent decades, been driven by new data, tools, and methods and by the extensive use of computers and codes (see Butz & Torrey, 2006).[6] Therefore, most, if not every, researcher in EEE is now also a programmer, albeit probably one who lacks the skills, best practices and methods that are now standard in other disciplines such as computer science. These principles are general enough to cover the entire

[6]There has been a significant decrease in the share of theoretical articles in the top-three economics journals (*American Economic Review*, *Quarterly Journal of Economics*, *Journal of Political Economy*): from 51% in 1963 to 19% in 2011 (Hamermesh, 2013). This decrease was in favor of empirical papers using data and theoretical papers with simulation.

research process. Applying them should help to improve the research processes and can be done with different levels of effort. Researchers involved in work packages embedded in funded research projects may identify that some principles are already implicit in their project milestones. These principles are simple and can cover a great diversity of actions. They are also easy to teach and to implement for any empirical work, even for inexperienced students. Thus, many smaller-scale research projects involving only a couple of researchers should also benefit from a more structured approach in their research.

Second, we describe simple and practical tools and methods that can help to achieve a better (and often easier) work organization. Reproducing research results is challenging in our field, where production is often assembled by manual cutting and pasting of "some results" (tables and graphs) produced by "some statistical software" using "some data" and "some treatments" generated by "some code". Following LeVeque (2009), we argue that "*constructing a computer program isn't so different from constructing a formal proof*". Since researchers are unaware of the tools and best practices for efficiently writing code, learning by doing is a common practice (Gentzkow & Shapiro, 2014; Millman & Pérez, 2014). We acknowledge that the problem is not always a software problem and that software will never be the solution, but a greater knowledge of the tools developed and used in other scientific communities could do no harm.

Third, we address the problem of research programs done in collaboration and/or with coauthors. The average number of authors per paper published in the top-five economics journals has increased over time, and research is mostly done within research projects.[7] This evolution induces more interactions between coauthors who need to improve their work organization. More interactions create more complexity and require documentation of each step of the research process. Similarly, we ask students to collaborate in their end-of-course projects — which can look like research projects — and we should be able to show them good examples. We should also take into account the unusual length of the publishing process in EEE compared to other disciplines.[8] The length of the process affects a researcher's memory of how and where programs, data and results are located on a hard drive, and the process is prone to forgetfulness and omissions.

The practices and tools reported here are drawn from our own experiences and readings and thus are not an exhaustive reporting of all methods and tools used in EEE. We will focus on and illustrate practices using simple tools that are easy to implement using off-the-shelf statistical software or languages popular in our community (e.g., Stata, R, SAS, MATLAB, Mathematica, Gams).

[7] In the early 1970s, three-quarters of articles were single authored, and the average number of authors per paper was 1.3. By the early 1990s, the fraction of single-authored papers had fallen to 50%, and the average number of authors reached 1.6. Most recently (2011–2012), more than three-quarters of papers have at least two authors, and the average number of authors is 2.2 (Card & DellaVigna, 2013).

[8] According to Björk & Solomon (2013), economics journals have the longest publishing delay: 17.70 months compared to physics at 1.93, biomedicine at 9.47, mathematics at 13.3, and arts and letters at 14.21.

We will also mention less statistically oriented software (Python, Julia). Some papers and books have described and illustrated reproducible practices using specific software such as R (Meredith & Racine, 2009; Gandrud, 2015; Xie, 2015; Xie *et al.*, 2018), Stata (Gentzkow & Shapiro, 2014; Jann, 2016, 2017; Rodriguez, 2017), SAS (Lenth & Højsgaard, 2007; Arnold & Kuhfeld, 2012), Mathematica (Wolfram Research, Inc., 2008) and Python (Bilina & Lawford, 2012), but to the best of our knowledge, such a broad presentation of principles leading to better practices using different software, or no software at all, has not previously been undertaken. We emphasize here also that many of the practices and methods proposed in this paper can be implemented independently of researchers' usual practices, preferred software or data confidentiality level. It should be straightforward for EEE students and young researchers, who are not bound to old practices, to adopt these principles.

The paper is organized as follows. In Section 2, we introduce the various notions of reproducibility and propose three main principles that lead to reproducible research. Section 3 is dedicated to the organization of the work, Section 4 addresses coding, Section 5 discusses automation. In each of Sections 3 to 5, a gradient of solutions is proposed, from simple to more technical. Section 6 concludes the paper.

2 Reproducible Research

The idea of reproducible research was defined by geologist John Claerbout as the possibility of the "*replication [of a paper] by other scientists*" (Claerbout, 1990; Fomel & Claerbout, 2009). This notion has since circulated and evolved, primarily in physics and computational sciences, in addition to global reflections on science and the goals of scientific publication. Building on the definitions proposed by Hunter (2001), Hamermesh (2007) proposes to distinguish the following two notions: "pure replication" and "scientific replication". The idea of "pure replication" refers to the ability to replicate almost exactly the research at hand, mostly for validation. This notion is essential in EEE, where the publication process is quite lengthy and therefore the need for researchers to replicate former results is crucial. The idea of "scientific replication", however, corresponds to the ability to reuse the research materials on another data set and can be seen as a robustness test or as an attempt to extend the initial work (see also Clemens, 2017).

The definition and concept of reproducible research has thus evolved across disciplines, with refinements, subtle distinctions or even conflicting terminologies (Stodden *et al.*, 2013; Barba, 2018). For many, a research project would be classified as reproducible if the authors of the project provided all the materials for any other researcher to replicate the results without any additional information from the author. In a strict sense, this means that a replication data set exists and is available and that managing the entire process, beginning with data preprocessing and ending with

the paper, including all steps in the descriptive analysis, modelization and handling of results, can be repeated. Thus, this definition applies to a research project that is composed of many elements and not solely to a paper. This notion acknowledges that "*a scientific publication is not the scholarship itself, (...) the actual scholarship is the complete software development environment and the complete set of instructions which generated the figures*" (Buckheit & Donoho, 1995), referring to "Claerbout's Principle" (de Leeuw, 2001).

How to achieve reproducibility is also a matter of intense debate. For many, including Claerbout (1990), replication is mainly a technical problem that "*can be largely overcome by standardized software generally available that is not hard to use*". For Schwab *et al.* (2000), the process of reproducing documents includes some technical components and a set of naming conventions. Others, such as Long (2009), invoke workflow management as a cornerstone for replication. These views have primarily been expressed in computer science, where code is the central element, but they can be transposed into EEE at various levels of granularity. Reproducible research can be roughly achieved without any specific additional software using only some common-sense rules and habits or, on the contrary, at a very fine level with a detailed description of each piece of each element involved in the process, including not only the code, of course, but also the software and OS version. It all depends on the project and on the expected or intended level of precision or reusability by other researchers.

At this stage, it is important to distinguish the quality of the management of the research process from the quality of the research itself. A stream of the economic literature focuses on the related problem of transparency and selection bias in methods and results in academic journals (Christensen & Miguel, 2018). These papers focus on the replicability of certain econometric methods, leading not only to practices such as cherry-picking and publication biases but also to the failure of replication due to opacity in the research process (Ioannidis, 2005). We will focus here on the practices used during the production process of a research project leading to publication and not on the methods used within the research process. Returning to the pie analogy, the goal here is not the final quality or taste of the pie but the reproducibility of the process leading to the production of the same pie, whatever its taste. Thus, reproducible research should be used even for papers with modest publication objectives and not only for top-ranked papers. Note also that some top-ranked papers are not reproducible (see, e.g., Herndon *et al.* (2014) concerning Reinhart & Rogoff (2010)'s paper, McCrary (2002) for Levitt (1997)'s paper and Levitt (2002)'s response, Foote & Goetz (2008) for Donohue & Levitt (2001)'s paper and Donohue & Levitt (2008)'s reply, and Rothstein (2007) for Hoxby (2000)'s paper and Hoxby (2007)'s answer).

We are not focusing here on a particular definition; on the contrary, we examine all practical issues linked to any of these notions. Our goal is to promote reproducibility in all its dimensions

by providing advice, methods and tools. Hence, we use the words reproducible, reproducibility, replication and replicability, in a very broad sense, throughout this paper. All definitions also include some degree of sharing (either privately or publicly) of the materials used in the complete process preceding publication.

Previous papers attempted to delimit the notion of reproducible research to a set of precise rules or principles to apply in specific contexts and software (Sandve *et al.*, 2013; Gentzkow & Shapiro, 2014; Hinsen, 2015). We propose only three main and simple principles to enhance the reproducibility of research in a broader sense. These principles are as follows:

- Organize your work
- Code for others
- Automate as much as you can

These three principles should not been seen as separate elements to apply sequentially on the path toward increasing the reproducibility of research but as interacting within a researcher's everyday practices. Note that these principles are already (at least partly) implicitly embedded in our own usual practices. Most of these principles can be applied gradually such that each practitioner may improve his own practices without investing and at low cost. The checklist provided in Appendix B may be used to read the paper in a nonlinear way, with each checklist question topic leading to a section in the paper.

Whether we use "good" or "bad" practices is a personal question and is not central to the approach. What matters here is the ability to reproduce, explain and share the key elements used in the process, leading to a result published in a journal. Following the pie analogy, "consumers" may be interested not only in the result but also in the ingredients, the recipe and all the cooks' little secrets that made the result enjoyable and meaningful.

3 Organize Your Work

A research project can be a complex process modeled by Long (2009) as a cycle that typically exhibits the following sequence: plan, organize, compute, document. The cycle breaks down into multiple iterative phases. At the beginning of the project, plans are rather general and become more precise as the project progresses. Some phases can be sequentially executed, while others may overlap. This explains why the organization of a project has consequences throughout its life. Therefore, it is better to consider and plan the organization at the beginning of the project. This is particularly true for "big" projects involving a great number of researchers but remains valid for the most common situation of research done by a single author and leading to an output

comprising one research paper.

One mandatory principle for achieving reproducible research is thus to organize the whole process and, specifically, to organize all the tasks needed and involved in the process leading to publication. These ingredients need to be properly organized if the pie is to be cooked again. It should be precisely known at which step in the recipe (phase and task of the project) which ingredients (e.g., data, methods) and which recipes (e.g., codes, documentation) are used and what are the interactions and relationships between each element to the resulting pie (e.g., project results). This process involves addressing related topics: task and documentation writing, file organization, workflow management and file manipulation. Many organizational forms can be considered: some are relevant for individual research projects, while others are better suited for projects involving many researchers or a team.

3.1 Organizing Tasks and Documentation

A good way to manage a project consists of knowing all the project's tasks and their contents, outcomes, organization and goals. It is useful to know the different people involved and their roles in the project and task deadlines. Of course, in a research project, tasks will evolve, as hypotheses, results and choices may change and reshape the project. It is thus necessary to organize the work and to document the tasks completed, directions abandoned and new directions chosen. For that matter, documentation is the key element. In the absence of any documentation, no research would be reproducible.

3.1.1 From Post-Its to Task Management Systems

For many researchers in social sciences, the usual practice is to write some sort of post-its or to-do lists of things to remember or to write notes in a notebook (always better than loose sheets of paper). This notebook is in fact a precious record of the research process and contains very useful information. However, from a long-term perspective — that is, from a reproducible research perspective — electronic documentation should be preferred to paper support for many reasons. First, digital documentation can easily be transformed into a printed archive of the work as long as no information is deleted.[9] Second, digital notes can easily be structured (and restructured). Organizing notes in chronological order (as in a paper notebook) is not the only possibility. Notes can be structured according to tasks or states. Third, it is easier to search in the document, rather than losing precious time turning a notebook's pages.[10] Finally, unlike a physical notebook, the

[9] It is useful to include the date and name of the current file at the beginning of the document. This will facilitate its retrieval when consulting a printed version of the notebook.

[10] However, to facilitate the search among dozens of files, indexation of all documents is recommended using one or several keywords enclosed by specific delimiters (e.g., ":RR:" is a possible index for this paper).

document, if stored on a network drive, may be accessible even when out of the office.

There exist a wide range of technologies, from basic to more complex systems, that can be implemented to handle these electronic documents. Text files (such as Readme files) can be a simple and efficient way of documenting any element used in a task and even the task itself (Baiocchi, 2007; Dupas & Robinson, 2013). Other simple tools involve lists and spreadsheets to keep track of ideas and record notes and information about tasks. However, in team projects, interactions between collaborators are common, and these simple tools cannot be used to record interactions and each individual's notes.

Electronic laboratory notebooks (ELNs) are a more convenient way to organize and manage notes. These applications make it possible to synchronize individual notes across platforms (computer, tablet, phone). When working in a team, to facilitate communication between collaborators, ELNs are now often accompanied by a tool that allows users to share files and write in real time. Specialized ELNs initially developed for the transcription of experiments now have extended features for usability, security (password protection, access rights), compatibility and backup. eLabFT is one example of a specialized open-source ELN (LabsExplorer, 2019), while Evernote, OneNote and Etherpad are the most common general-purpose ELNs used today.

Finally, task management systems (TMSs) offer features for a more general collaborative and centralized task organization. The usual way to share information with coauthors is to exchange emails where hypotheses, programs and data are shared and discussed through long sequences of carefully stored messages. This is far from a clean documentation strategy, as it generates numerous messages that mix different topics and may lead to ambiguous decisions (see the explicit example in Gentzkow & Shapiro, 2014). Such written elements are difficult to maintain in the long run, and searching for some piece of information (such as a sequence of decisions), in numerous emails with possibly the same title can be very complicated. On the contrary, TMSs are designed to help people "*collaborate and share knowledge for the accomplishment of collective goals*" (Wikipedia). In the design of TMSs, each task is represented as a card that can contain a description and attached exchanges with collaborators. Each task can be assigned to a collaborator and can be moved into a "To do", "In progress" or "Done" category on a dashboard as soon as the state of the task changes. A due date can be added. Each dashboard is filled with vertical "lists" that constitute the task prioritization system, a functionality that is not included in ELNs.[11] Several TMSs exist and are available on the web, e.g., Trello, Asana, MS Project and Wrike.

[11] TMSs have many more functionalities, such as the possibility to keep track of all tasks, even completed ones, either on the web service itself or by exporting it in a numerical format (such as JSON, csv); the possibility to measure the time spent per task; the ability to draw Gantt diagrams (allowing users to visualize the project schedule with bar charts that illustrate the start and completion dates of different tasks); calendar sharing; file versioning (including the ability to link to GitHub); and the configuration of email notifications depending on due dates. Another interesting feature of TMSs such as Trello is that they can be linked with ELNs (OneNote, Evernote).

3.1.2 From Comments to Task Documentation

Documenting a research project is often regarded as a painful and time-consuming activity. However, many dimensions of documentation are easy, helpful and time efficient. The documentation of tasks and their purpose is often implicit (names, habits) and should be explicitly stated to ensure some reproducibility and traceability. This is not merely a matter of writing additional comments or documents "on" or "in" the code or "in" the working paper. Documenting tasks is an important aspect of documentation and should be considered one of the first steps. During a project, regular updates should be planned and implemented to avoid any loss of implicit and unwritten information.

Schematically, task documentation is composed of a brief description of things to do, the people involved, the scheduled tasks and their states (to do, in progress or done). Information that cannot be explained within the task (documents relating to a specific ingredient such as the code) should also be documented at the task level: general choices about the project (hypotheses and decisions such as the type of modelization, the population under study, and abandoned tested directions) and technical specifications that can have an impact on the results. For example, inclusion/exclusion criteria for observations, the randomization method and random seeds, initial values and parameters chosen for optimization, robustness checks, or the algorithm used to display the results (e.g., interpolating, smoothing) have to be clearly documented.

At the task level, all tasks involve different ingredients, and each of these ingredients should be accompanied by a piece of information with relevant data on the ingredient itself. This practice is well known by practitioners that use, e.g., a data dictionary or comments in code. However, this is far from sufficient.

Data dictionaries attached to a data set describe the variables and their definitions and are familiar objects. However, a more global description of the data such as data sources, name, producer, number of files, format, date of reception, covered period of time, file keys, sample method, and the weighting method is needed to precisely characterize the data set, its origin and its evolution over time. For confidential data, the whole process for data accessibility and the data terms of use have to be mentioned. The information can be recorded using standard metadata (Dublin Core, or Data Documentation Initiative — DDI — for social sciences), which provide more precise information on the data set itself. Journals and funding agencies are giving increasingly more care to data integrity and data dissemination by providing metadata guidance (e.g., *American Economic Association*, *Social Science Data Editors*) or by requiring the implementation of data management plans. Citing data with DOI identification from open trusted repositories can also facilitate the reusability of data sets (Stodden *et al.*, 2016; Lagoze & Vilhuber, 2017). As DOI identification does not impose the availability of data, it could be provided by the data producer

even for confidential data. Furthermore, security and access conditions can be described by the datatags system proposed by Sweeney *et al.* (2015).

Programs also need to be documented, and it is true that the documentation of most programs can be embedded in the code (see Section 4.1). However, the technology is evolving rapidly, and code execution may change if the computer environment changes. Furthermore, documenting a research program is not limited to documentation of programs but must ensure that the overall work sequence (the workflow) is explained in great detail. It may be seen as a picture of the environment, as well as all the states of elements used, in an easy-to-read way. We will present all these elements in this paper. We also suggest, as a good practice, the inclusion of a specific section in the final publication describing the computing environment (Koenker & Zeileis, 2009).

3.2 Organizing Workspace

Most, if not all, of the documents produced by researchers essentially consist of files organized in very different and personal ways. This makes it very difficult for researchers to work together when they are all using different practices to organize information. Undoubtedly, there is no perfect organization, but there are some elements to consider when organizing a research project, just as there are some tricks for organizing a library or kitchen. We focus here on two major aspects of file organization: directory structure and naming convention.

The directory structure of a project is intended to facilitate finding the elements (code, data, output) one needs. This is particularly important in projects with long time horizons and inactive periods that can last from a few days to a few months. To avoid confusion and to facilitate memorization, it can be helpful to maintain a consistent structure across projects and to always define the same directory organization for each project.

When constructing a project directory structure, two guiding ideas can be used:

- a folder should contain homogeneous elements of the same type (data, programs, text, documentation);
- a clear distinction should be made between inputs to the project and outputs from the project.

The aim of the latter point is to prevent unintentional deletion of pieces of the project, as it seems obvious that files located in input directories must never be updated or deleted. We propose in Figure 1 a simple directory structure very similar to that of Gentzkow & Shapiro (2014). This architecture is an illustration following the guiding ideas defined above and can be modified or completed to fit personal preferences, habits and project type. Depending on the complexity of the sources, the "Inputs" folder may also contain subfolders to distinguish "raw" or "original" data

sets from "treated" or "refined" ones as well as other inputs in a broad sense. The same applies to the "Outputs" folder depending on the nature of the outputs (figures, tables, estimations, ...). Some practitioners also add a temporary folder (sandbox) for saving temporary versions of code or documents for a short period of time. However, there is a trade-off between the complexity and efficiency of the architecture, as complexifying the tree structure increases browsing time when searching for files (Santaguida, 2010).

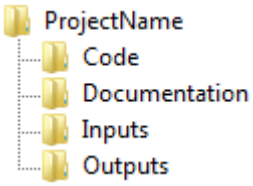

Figure 1: Example of a Well-Organized Directory Structure.

In the computer science community, file organization also uses simple ingredients such as file-naming conventions. We suggest using these ideas; that is, every file created for a project should follow a naming convention. Typically, a naming convention is implicit and personal to each researcher. A good practice is to explicitly select a naming convention, maintain it and share it with coauthors, if any.

Before naming programs, one should have a clear idea of the goal of the programs. Whether it is a stand-alone program or a piece of a more general programming framework should be explicit in the name of the program. Furthermore, it is recommended to use separate programs for constructing a "clean" data set and for the statistical analyses based on that data set (Nagler, 1995). This practice allows new analyses to be run without regenerating the data set from scratch. The name of the program should therefore reflect its purpose. For example, we know immediately, without opening it, that the file named `stats_desc.R` is an R program whose purpose is a descriptive analysis.

The same naming convention can be used not only for programs but also for every file.[12] An output can be named according to the name of the program that generated it and followed by a suffix. This simple rule allows locating the file in the workflow. For example, the program `stats_desc.R` generates the output `stats_desc_out.tex` containing the results, probably in LaTeX, of the descriptive analysis.

However, an explicit — and possibly long — name is insufficient, and names should be kept simple, as short as possible, and portable across systems and software. Following Long (2009),

[12]Electronic versions of papers (bibliography) should also follow naming conventions to facilitate finding them. For example, for a single author A that wrote in year Y, we could name their paper A_Y.pdf.

we recommend limiting the characters used in file or folder names to `a-z`, `A-Z`, `0-9`, and the underscore. Additional information about naming conventions will be provided later from a coding perspective (see Section 4.1.2).

Using a naming convention for files can also aid manual management of file versions. It is standard practice to use date suffixes (declared as `yyyy_mm_dd` to keep the automatic ordering consistent with the order of the files) for the names of programs or to use version suffixes (v1, v2, v3) specifying the current version of the file.[13] This organization is still quite basic as it does not really help in following the research process, generates many similar files, and may still lead to confusion. We will see in Section 3.4.2 that more powerful tools exist that can automatically manage file versions.

When working with others, explicit conventions for directory structure and naming files have proven to be very helpful and allow coauthors to understand and find what has been done by whom and to contribute as soon as files are shared.

3.3 Keeping Track of the Workflow

EEE projects, even simple ones limited to the writing of a single paper, are usually quite long, longer than projects in many disciplines (Björk & Solomon, 2013), and keeping track of the workflow is another substantial issue. There are many ways to represent the workflow (such as different scales and different conventions), just as there are many ways to conduct research, but there is a common global structure to any project in EEE, and it follows the sequential creation of files displayed on the left hand-side of Figure 2. One cannot avoid following a workflow that goes from raw data to a working (clean) data file (or files) leading to the creation of some intermediate files that will in turn be used in a working paper and/or in the final publication. This is a common basic workflow, not only in EEE. How one should move from one file to another is probably less standard and depends upon each project or researcher even if, at a broad level, there should be programs that link all the files and follow a precise path and order.

This broadly universal representation of a project can help to outline any research project's workflow by simply illustrating the links (programs, actions) to and from elementary blocks. To make an entire research project reproducible, all files, all links, and the whole workflow should be as clear as possible and explicit. This process is key to understanding how programs (the circles in Figure 2) and data (rectangles) are related, how programs are related to one another, and the running order of programs.

[13] However, note that even if this habit can be improved using other tools, it is less dangerous than having the same named file with different contents in two different locations!

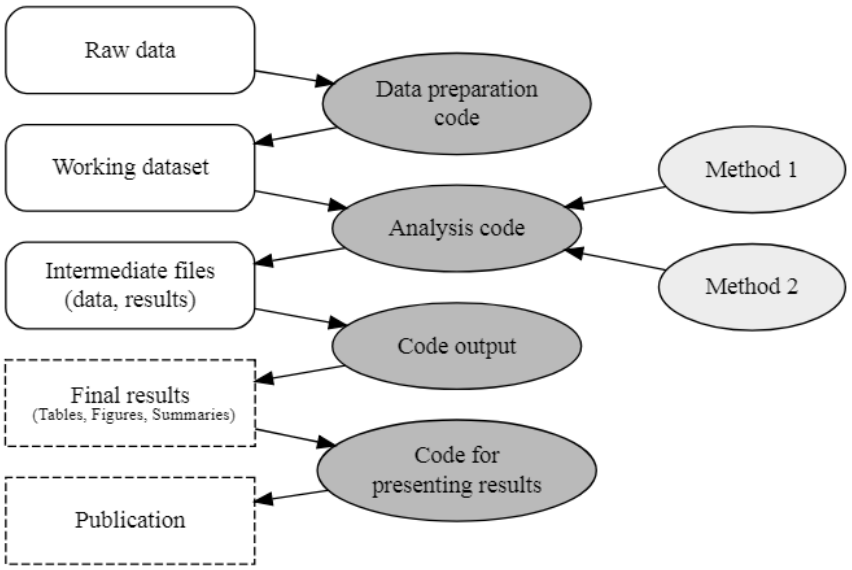

Figure 2: A Simple Example of a Workflow Generated with GraphViz (see Code in Appendix C)

There are various ways to manage a workflow. One solution is to document the entire workflow and generate a graph, as in Figure 2. Different tools exist to create, manage and graphically visualize a workflow. Some illustration tools can be used to draw workflows; these tools require that one either manually draw the workflow (e.g., Dia) or code the elements of the workflow (boxes and links) to automatically generate the figure (e.g., GraphViz, RGraphViz). Finally, some tools such as SAS Enterprise Miner allow interactively generating an image of the workflow, while others (e.g., Snakemake) generate a graph directly from the analysis of the links between code and data in the workflow.

Another complementary idea is to use a naming convention. For example, Chuang *et al.* (2015) propose placing a number indicating the order of execution before the name of the program, as in Figure 3. This is what Long (2009) calls *"the run order rule"*. Running the programs in that specific order will allow a researcher to reproduce the entire workflow.

01_preparing_data.sas 02_stat_desc.sas 03_model1.sas 03_model2.sas	This naming convention indicates that the program that prepares the data has to be run first, followed by the descriptive statistics program, but for the model part, there is no order. *03_model1.sas* and *03_model2.sas* can be executed independently as soon as *02_stat_desc.sas* has been executed. This can be useful for someone that does not use any integrated approach.

Figure 3: Naming Tricks for Workflow Management.

Note that this solution, although straightforward to implement, is not flexible in the case of modification of the order or when using scripts in other modules. A more flexible solution will be

presented in Section 5.3 and consists of managing the workflow directly and automatically using a dedicated software.

3.4 Handling Files

Undoubtedly, good file organization is always good for reproducibility and can improve productivity. However, the need for very clear organization becomes crucial when working alone with several computers or when working with others. Sharing a structure among members of a project (directory structure and naming convention) and committing to the same relational scheme (workflow) can greatly enhance the efficiency of a project. However, as a project increases in size as well as in coauthors and time, meaningful growth in the file-handling strategy is needed to overcome three main problems: How should files be shared? How should files be compared and versions managed? How should researchers collaborate when writing a joint paper?

3.4.1 Sharing Files

A common way to share a project and its structure is to compress all the files and their tree structure in a .zip or .tar file to ensure the consistency of programs and that the files are correctly linked. Then, the compressed file is sent via email with or without some accompanying text. This common practice can rapidly become dangerous, as decompressing an archive may erase recent work done locally. Moreover, if updates are frequent or if there are many people working on the project, the process can become intractable.

For projects involving many files and directories, files can be copied onto a USB key or a portable hard drive. In Table 1, we present a nonexhaustive list of tools and practices from the simplest one, which is not very tractable, to more efficient ones. One approach is to place the files on a secure external server devoted to sending heavy files, such as FileSender available from RENATER[14] or WeTransfer. More common practices now include using tools such as Dropbox, Google Drive, and OneDrive that provide a free working directory with a capacity of approximately 15 GB where one can share files with anyone by controlling access and writing authorizations. Unfortunately, these most popular solutions come with some security concerns. The problem here is that the files are uploaded onto private servers, and there is no guarantee that they will not be used for other purposes. Obviously the above mentioned practices should respect the data terms of use. In particular, the code may be shared even with confidential data.

We believe that the best way is to directly share a workspace and, if possible, have precise control of who is able to access what (such as reading, writing, and erasing). Free tools exist to

[14]RENATER (National telecommunications network for Technology, Education and Research) is the French public broadband provider.

Tool	Advantages	Disadvantages
Temporary exchange		
WeTransfer	Easy	Registration mandatory, 2 GB max, security concern (U.S.- based servers)
JustBeamIt	Easy (drag and drop), peer to peer	2 GB max
FileSender (RENATER)	Easy, up to 20 GB, secured and free for registered institutions, cloud based in France	Upload reserved to French institutions
Shared working spaces		
Dropbox	Synchronization of files on the computer, integration with MS Office, backup, offline availability	Security concern (U.S.-based servers), synchronization after offline changes risky
Google Drive	Online editor (docs, sheets, slides), browser access, easy sharing, comments in document, cloud-based storage	Security and privacy concern, Google account mandatory, no editing offline
Joomla	Web-based access, total control of user rights (private, public), unconstrained space	Content Management System (CMS) to be installed on a server, no offline editing
Agora	Web-based access, easy control of user rights, unconstrained space	CMS that needs to be installed on a server and administrated, no offline editing
SharePoint	Easy integration with MS Office software	Needs to be installed on a server, transparent use on an intranet, offline editing risky
PARTAGE (RENATER)	Webmail, shared task manager and calendar, messenger	Restricted to French institutions

Table 1: Tools for Sharing Files.

transform any computer into a server using FTP protocol (for example, FileZilla). This functionality makes it possible to share a directory on a computer and to give access to collaborators for downloading files. Other tools such as Joomla, Agora, and SharePoint are designed to build and update websites, thereby allowing authorized people to share and work on the same files. These are professional solutions that are costly to construct but might be available at some researchers' institutions.

3.4.2 Version Control

In any research process, many file changes are made over time: corrections, improvements, additions, comments, and new ideas are part of a researcher's day-to-day life. This leads to a general problem of having different versions of the same file, possibly differentiated by adding a suffix (number, date) to the file name. This manual control of the different versions of a file requires the application of very strict rules for file naming and file management to work properly. Visualizing the differences between two versions of a file and understanding or undoing changes are not easy and require specific tools. Software such as Notepad, Baloo, Total Commander, WinMerge or muCommander is quite effective for text or Ascii files and allows the researcher to identify the differences within each block of the document or code. However, this requires some effort and, once again, some file manipulation by the researcher.

To avoid the manual control of files, version management tools offer attractive alternatives. These programs facilitate file comparison, record the history of changes, allow for the restoration of an old version of the document, allow for comments on the differences across versions, and provide a way to work with coauthors on the same files, eliminating any risk of overwriting changes (Koenker & Zeileis, 2009). Appendix D details the main concepts of version control systems, which can be centralized or distributed. Distributed version control software (Git, Mercurial, Bazaar) allow working locally and offline, and they are thus more convenient than centralized version control tools (Subversion, CVS).

Every version management tool is associated with a server that hosts all versions of the project. This server can either be on one's personal computer, on a local server, or on an external server. At present, the most popular hosting services are GitHub, GitLab, SourceForge, BitBucket, and CloudForge (Wilson *et al.*, 2014). As for file sharing, the choice of the versioning tool should satisfy the data terms of use.

3.4.3 Collaborative Writing

Versioning is well suited to the sharing of code but is more difficult to implement when writing a research paper. Technical difficulties are usually encountered and are time-consuming: a few examples are differences in work habits (such as using Microsoft Word, OpenOffice, or LaTeX), software versions, and bibliography management.

Two common "schools" still coexist: those working with MS Word, OpenOffice or Google docs on a WYSIWYG (What You See Is What You Get) basis and those working with LaTeX. People who use Word like its simplicity and its embedded revision feature. The revision mode makes it possible to change and comment on the document, and it is easy to see different versions of the document with or without changes made by others. However, even with that feature, it is still difficult to cope with multiple changes made by multiple coauthors over time.

Proponents of LaTeX appreciate its nice and customizable publishing style, its interoperability with different software (R, Sweave, BibTeX) and the substantial number of possible extensions. Some people are deterred because it is disturbing not to see the written result of what is typed directly on screen, for which a compilation step is usually necessary. Some tools are worth mentioning in this regard, such as Overleaf, that allow users, even those unfamiliar with LaTeX, to work (simultaneously) with LaTeX files on a single online document.[15] The file is accessible through a

[15]This paper was written using LaTeX. The document was shared between the authors and edited online using Overleaf, where all the versions and the bibliography files (Bibtex) were stored. We shared the literature using the Agora platform. Codes used in this paper can be found at `https://github.com/HowToMakeAPieTheCodes/`

browser on the web and allows one to see, on two separate panes, the typed text and, almost simultaneously, the published results. Several users can collaborate on the same document, and new users can learn from experienced ones.[16]

Basic recommendations can be provided: the use of a stable (over time), portable and unique file format (text, LaTeX) allows easy file sharing between coauthors and avoids software version problems encountered in WYSIWYG environments such as MS Word (see Koenker & Zeileis, 2009). Another recommendation is to keep track of changes in the shared document and let others add new elements (comments, references). Reference manager software, such as Zotero or Endnote, provides easy-to-use collaborative features, thereby allowing researchers to share references and bibliographical files.

4 Code for Others (Including Your Future Self)

For many, an important feature of any piece of code is being understood by the computer, which has to execute the code and thus compute unambiguous and correct instructions that reflect the intentions of the author (Gentzkow & Shapiro, 2014). However, code need not only run correctly but also be written in a clear way such that it is understandable by humans, including the future self, coauthors, programmers, collaborators, or students. This idea is not new; Knuth (1984) wrote "*let us concentrate rather on explaining to humans what we want the computer to do*". Wilson *et al.* (2014) recommend that one should "*write programs for people, not computers*". Donoho *et al.* (2009) and Koenker & Zeileis (2009) recommend working as if a "*stranger*" (anyone not in possession of our current short-term memory and experiences) has to use the code and thus has to understand it. These ideas are echoed in many dimensions of the replication process and in the code-writing activity that is generally called "style".

4.1 Programming with Style

Each person programming has a style based on experience, influences and readings. However, some general rules, practices and tricks exist to facilitate the reading of any program. We provide here some advice that should be used in conjunction with parsimony and common sense. For Kernighan & Pike (1999), good programming relies on three basic rules: simplicity, "*which keeps programs short and manageable*", clarity, "*which makes sure they are easy to understand, for people as well as machines*", and generality, "*which means they work well in a broad range of situations and adapt well as new situations arise*". These rules apply whatever the level of programming involved (single

`HowToMakeAPie`.

[16] A paid version of Overleaf has a history function that makes it possible to go back to previous saved versions and compare the actual version with older ones.

program, complete application) and regardless of the language used.[17] Note that a well-written program is also easier to debug and to maintain.

To have a consistent programming style, one may follow a style guide based on these three rules, leading to conventions on layout and naming and on the writing of generic code. We provide here some programming style conventions along these three dimensions. It is important to note that there is a trade-off between writing perfectly legible code and the time spent writing it. Pair programming, where two researchers sit together while writing code, is recognized as a good way to improve writing style (Wilson *et al.*, 2014). This method can be particularly helpful in the early stages of coding for defining conventions among several programmers. It is also a valuable option when debugging code. When researchers work in different places, using a screen-sharing platform (e.g., Skype, TeamViewer, Zoom) can also improve the writing style.

4.1.1 Conventions on Layout

The number of characters per line should be limited: van Rossum *et al.* (2001) suggests using a maximum of 80 characters; Miara *et al.* (1983) suggest that the indentation level and the number of indentations should be limited. Some guides also recommend placing spaces around all mathematical operators (=, +, -, *, /).

Then, the code should be structured and follow a standard sequence to be analyzed and understood faster by others (Levin, 2006). All definitions should be placed at the top of the code, followed by function definitions and finally by executed statements. For the sake of readability, and to take into account the limited human working memory (Wilson *et al.*, 2014), Boswell & Foucher (2011) also recommend defragmenting programs such that each piece of code does only one task at a time.

Some programs such as R and Python have developed tools to control and/or correct the code following predefined conventions: "*pylint*" in Python, the "*check*" function for writing R packages and "*formatR*" to adapt the code to the standard R layout.

4.1.2 Conventions on Naming

As for files, variable names must be chosen to facilitate the reader's understanding without contradicting their content. The rule "*Make names consistent, distinctive and meaningful*" from Wilson *et al.* (2014) identifies several points to address when naming objects, variables and functions in a program. Boswell & Foucher (2011) provide the example of a variable named "size" that is too general and can be replaced with more informative names such as "height", "NumNode", or

[17] Each programming community often provides its own principles (e.g., the "zen of Python", Peters, 2004).

"MemoryByte". Nagler (1995) uses the example of the gender variable often encountered in social sciences. In the files provided by the French National Institute of Statistics and Economic Studies (INSEE), this variable is coded 1 for men and 2 for women. A more purposeful and less ambiguous variable would be a dummy variable called women that takes a value of 0 for men and 1 for women.

There is no general rule on the length of variable names. The important point is to build meaningful names by combining uppercase and lowercase letters, underscores and digits using what Wilson *et al.* (2014) call CamelCaseNaming or pothole_case_naming.

Conventions can also help to label variables (Nagler, 1995): uppercase names for constants (`MY_CONSTANT`) and lowercase names for variables or functions and methods (`my_function`). Using uppercase can also help to distinguish created variables from the original ones provided in the source. Others suggest identifying dummy variables with an "I_" or "i_" prefix (e.g., i_strawberry).[18] Moreover, the use of descriptive names for global variables and short names for local variables (Kernighan & Pike, 1999) will aid comprehension. Variables named i or j are usually used for loop counters. van Rossum *et al.* (2001) further recommend not using the lowercase letter l or the uppercase letter O because they can be easily confused with the digits one and zero.

4.1.3 Writing Generic Code

Wilson *et al.* (2014) explain that the DRY (Don't Repeat Yourself) principle increases the readability and maintainability of code, leading to greater reusability of the code, which is necessary for any potential replication.

A first easy-to-implement habit is to use relative paths in programs when calling a function, another program, or a data set. In Figure 4, we provide four examples (in Stata, R, GAMS and SAS) that clearly demonstrate the application of the DRY principle. Specific (to a researcher's computer) directory paths are stored in either local or global variables that are used as the reference directory. In the rest of the program, all references to any path are relative (hence the use of dos-like commands with `../` and `cd`). Once coauthors begin to follow the same directory structure, this practice will enhance the compatibility and portability of the whole code on another machine.

The importance of keeping code as generic as possible is illustrated by the counterexample in Figure 5. In this figure, the code does not follow the advice of "*Never type anything that you can obtain from a saved result*" (*Drukker's dictum*), which means that it is important to use information (mean, estimated coefficient, matrix) that the software can provide as a part of the code

[18] Some authors consider that the letter capital "I" should never be used to avoid confusion with the numeral "1" (van Rossum *et al.*, 2001). However, conventions must be selected depending on use. Stata, for example, automatically creates variables with the prefix "_I" when specifying dummies for regressions.

```
/**** Stata EXAMPLE ****/
  /**** Definition of the useful path ****/
  local CodeFolder "c:/ApplePie/Progs"

  /**** Positioning ****/
  cd "`CodeFolder'"

  /**** Using data that is in another folder ****/
  use ../Raw_Data/Sugar.dta, replace
  append using ../Raw_Data/Apple.dta

  save ../Final_Data/ApplePie.dta, replace
  qui log close
```

```
####  R EXAMPLE ####
  # Definition of the useful path
  CodeFolder <- "c:/ApplePie/Progs"
  GraphFolder <- "../Graphs/"

  # Positioning
  setwd(CodeFolder)

  # Saving the graph to another folder
  file <- paste(GraphFolder, "MySuperPie.png", sep="")
  png(filename = file)
  pie(rep(1,8), col=1:8)
  dev.off()
```

```
*### GAMS EXAMPLE ####
  * Select "Apple" or "Banana"
  $setglobal Fruit Apple

  * Using data-loading programs to another folder
  $ifi %Fruit% == Apple $include Raw_Data\AppleData.gms
  $ifi %Fruit% == Banana $include Raw_Data\BananaData.gms

  * Solving the model in the current folder
  $include Recipe.gms

  * Exporting results to another folder
  execute_unload 'Final_Data\%Fruit%Pie.gdx'
```

```
* SAS example ;

  * Directory root of the project ;
  %let rep=c:\ApplePie;

  * Definition of input and output directories ;
  libname ini "&rep.\Raw_Data";
  libname fin "&rep.\Final_Data";

  * Output data set derived from input data sets ;
  data fin.ApplePie;
       set ini.sugar ini.apple;
  run;
```

Figure 4: Implementation and Use of Relative Paths in Different Software.

(Long, 2009). The R code in Figure 6 provides a better solution.

```
coeff_variation_Sugar_Qty <- 2.1201803  # sd / mean = 4234 / 1997
coeff_variation_Chocolate_Qty <- 4  # sd / mean = 4/1
```

Figure 5: Example of an R Program without Genericity (Some Values are Fixed by the User).

```
standard_deviation_Sugar_Qty <- sd(Sugar_Qty)
mean_Sugar_Qty <- mean(Sugar_Qty)
coeff_variation_Sugar_Qty <- standard_deviation_Sugar_Qty/mean_Sugar_Qty
```

Figure 6: Example of an R Program with Genericity (the Values are Computed from the Data Set).

Following the DRY principle should also induce greater modularity. A modular program, composed of reusable blocks of code (functions, packages), is easier to read and to understand. Anticipating or determining what to put in a function is not always easy. When it appears that duplicate lines are necessary, it is helpful to write a function and to refactor the program. Refactoring methods include all modifications that are made without changing any of the functionalities while improving the internal structure of the code (Fowler *et al.*, 1999). The initial code will be replaced by the call to the function, and additional calls to the function can easily be introduced

while limiting the risk of errors. It is also possible to use the function in other projects.[19] For example, the R code in Figure 7 is a modular version of that in Figure 6.

```
fct_coef_variation <- function(numvector)
{
    if( is.numeric(numvector) == F | is.vector(numvector) == F )
    {
        stop( "The data should be a numeric vector" )
    }
    standard_deviation_data <- sd(numvector)
    mean_data <- mean(numvector)
    coef_variation_data <- standard_deviation_data / mean_data
    return(coef_variation_data)
}
# Call the function for Sugar
fct_coef_variation(Sugar_Qty)
# Call for Chocolate
fct_coef_variation(Chocolate_Qty)
```

Figure 7: Example of an R Modular Program Based on the Generic Elements of Figure 6.

4.2 Documenting the Code

Figure 5 also illustrates the poor use of comments in code. Comments should be sparse and well considered, not "post-its" used to justify a lazy coding structure. Excessive comments can hamper the readability of a program. In many cases, unnecessary comments can be avoided (for example, by cleverly naming variables, parameters, and functions). Following Nagler (1995), we recommend including comments before each block of code that explain the purpose of the block or provide a helpful reference to consider. End-of-line comments should be infrequent. Using a good naming convention for variables and a logical code structure that "self-documents" the code should greatly reduce the need for comments and enhance the code's readability (see Gentzkow & Shapiro, 2014; Millman & Pérez, 2014). To quote Koenker & Zeileis (2009), "*Source code is itself the ultimate form of documentation for computational science*".

In addition to well-written code and effective use of comments, it is useful to indicate, at the top of any program, the information needed to understand it, for example, the date, description, goal of the code, version and changes from previous versions, input (and output) data files, input parameters, the version of the software at the date of writing, packages used (and their version), and the name of the creator. An example is provided in Figure 8.

It is also important to ensure the reproducibility of computation. Many components are involved in such computations, and each should be checked. Statistical software is updated regularly,

[19] Wilson *et al.* (2014) extend this principle to others (Don't Repeat Others) and call for the use of prior code from reliable sources instead of creating completely new code.

```
Program for pie cooking technology

Goal: Generate the Chocolate Foam estimations
Date: 2017/01/05
Author: Top Chief
Running under R version 3.2.2 (2015-08-14)
Platform: x86_64-w64-mingw32/x64 (64-bit)

Input files: chocolate.csv, eggs.txt
Output: ChocolateFoam.R, ChocolateFoam.tex

Version 4 of the program: + function fct_coef_variation
```

Figure 8: Example Documentation of the Computing Environment and Code.

meaning that the inner code of some commands is revised. These improvements may cause failures in efforts to reproduce the computation. However, keeping track of the software (and its version), the packages used (and their version), and the computer used (CPU, operating system) helps to avoid such disagreements.[20] This should be documented within programs (see Figure 8).

A nice way of sharing code documentation is to use a documentation generator that parses and extracts all comments from the code and automatically creates well-formatted documents (Millman & Pérez, 2014). Several programs include this feature (e.g., MATLAB, GAMS, Python, R). A Python example is given in Figure 9.[21] The Python *docstring* comments are composed of two triple quotes and can be extracted using a specific tool such as *pydoc*. The generator is also able to extract the set of variables defined in the code (see "Data" in the right panel of Figure 9). Embedded documentation is intended to limit inconsistency within the code by facilitating documentation updates when modifying code (Wilson *et al.*, 2014).

5 Automate As much As You Can

If any research workflow, such as the standard one presented in Figure 2 (Section 3.3), is conceived such that all elements (programs, data, files) are clearly linked within programs, then it will be easy to automate the entire process. In terms of reproducibility, it will greatly help any end user to use and reproduce, even partially, the research output. Unfortunately, this ideal vision of research organization is far from reality. Hopefully, as with any practice, automation can be achieved at different levels, with simple or sophisticated tools, demanding various levels of effort and time. The

[20] Some software programs have commands or packages that address this issue by allowing the user to automatically save and load the computing environment. See, in this respect, the "checkpoint" R package (Racine, 2019) and "packrat" (Ushey *et al.*, 2016). For Stata, the "version" command indicates which Stata version is needed to run the code.

[21] For MATLAB, *Publish* (from the editor) does this. For GAMS, the *model2tex* tool is intended to document the modeling parts of GAMS programs as LaTeX documents. In R, the *Roxygen2* package allows users to generate automatic documentation of a package.

Figure 9: Example of Automatic Documentation Generation (using Docstring in Python).

required conditions are straightforward: code must exist at every stage, and all the code for all the stages should provide access to all the results. Under those simple conditions, automation, either by using a script file or within software or notebooks, is merely a matter of personal organization and preferences.

5.1 Coding Everything

Point-and-click software, such as MS Excel, is widely used in EEE (Barreto & Howland, 2005), even for complex computations that can still be done without typing a single line of code. Tedious searches for syntax errors and command names are then avoided by using drag-and-drop menus and copy-pasting elements from one cell to another. In MS Excel, the code is fully embedded in the spreadsheet for the data set. On the one hand, as the code and data are in the same file, it is easy to manage the workflow since everything is in that data set. On the other hand, the code cannot be easily examined, printed or shared outside the self-contained MS Excel file. Automating tasks is thus difficult, albeit feasible using the Visual Basic for Applications (VBA) language, but its use is quite limited and complex. Therefore, copy-pasting cells and direct programming within a cell are common practices. Moreover, the output (tables and graphs) is also attached to the MS Excel file. For the final article, the tables and graphs are copied and pasted, often without any reference or tangible link to the code and the MS Excel file.[22] The Reinhart & Rogoff (2010) case

[22]It is possible to link a graph or a table between MS Excel and Word, but links are broken when files are renamed or moved. Moreover, broken links are not always indicated to the user. This process is also highly vulnerable to

showed how results produced with this technology are fragile and not suited to a proper review prior to publishing. MS Excel itself is not the cause of the lack of reproducibility and readability, but its use facilitates unrecommended practices such as drag-and-drop and copy-paste.

When analyzing time series, Microfit is a popular point-and-click tool (Pesaran & Pesaran, 2010). Unfortunately, this software does not provide any possibility to read, save, or recover any line of the underlying and invisible code assembled after a series of menu-driven mouse manipulations. This software makes it difficult to save, reproduce and share work. Thus, despite its great econometric features, a reproducible research approach is not feasible using Microfit.

Other programs, such as Stata or Eviews, also offer a point-and-click approach to facilitate the discovery of commands and to shorten the coding time needed for some lengthy commands (such as for graphics). However, each drag-and-drop action is displayed in the console and recorded so that it can be learned, saved and reused. Automating actions, recording code, and saving logs, tables and results are then easy tasks, and these features greatly enhance the likelihood of producing reproducible research. Although it is difficult to imagine that a fully reproducible approach could be applied to research done with MS Excel, it is not necessarily true that using Stata, MATLAB or R provides simple solutions without best practices. The point is that the software is not always the problem, and it will never be the solution. Practices have to be adapted to software use and possibilities. Nevertheless, some software makes it easier for researchers to automate, record, recover and share their work.

Since working on a research program consists primarily of writing code at each stage of the process, code represents the most valuable component of the research. Therefore, at each stage, code should explicitly mention the input (data used) and output (results) following the advice cited in Section 4, and it should be possible to properly save any piece of code in a way that is readable, understandable and reusable. Note that coding is not limited to statistically based work on a given piece of software, as an important part of any research is done either before (data preparation, sampling) or after (results handling, tests, refinements) such efforts.

5.2 Exporting the Results

In a research paper, tables and graphics are the visible and final aspects of the research project. The common practice for generating tables of results (see, for example, Table 2) consists of reporting the results cell by cell or by manually rearranging raw output copied from a software output console and pasted somewhere else. The automatic generation of all of the estimations, numbers, graphics and tables produced for an article is a minimal requirement for ensuring the traceability of any results from the raw data set to the final paper. Even when carefully done, copy-paste

potential compatibility issues across different versions of MS Excel.

practices do not guarantee that the results printed in a paper could be obtained again.

	OLS		2SLS	
Price	-0.001^{***}	(0.000)	-0.001	(0.001)
Cooker level	0.161^{***}	(0.006)	0.161^{***}	(0.006)
Number of different ingredients	0.030^{***}	(0.007)	0.040	(0.036)
Number of servers	-0.042	(0.038)	-0.044	(0.039)
French recipe dummy	0.016^{*}	(0.009)	0.016^{*}	(0.009)
Michelin rating rank	0.050^{***}	(0.008)	0.049^{***}	(0.009)
Constant	-0.051	(0.113)	-0.098	(0.201)
Observations	428		428	
R^2	0.736		0.734	
Sargan statistic			0.923	
Sargan p			0.630	

Standard errors are in parentheses.
IV are input prices: sugar, flour and egg prices.
The Sargan test is an overidentification test of all instruments.
This is a fictive example (no real interpretation).
* p $<$ 0.10, ** p $<$ 0.05, *** p $<$ 0.01.

Table 2: Regression Table Created Using the Stata *esttab* Command.

Many programs have included features (commands, packages) to export different types of output in a portable format (txt, rtf, LaTeX, and html, or PNG, JPEG and WMF for graphics; see Table 3). Most programs also provide log files that report the executed code and the output, albeit without incorporating them. Table 2 was created automatically in the software using a dedicated line of code and exported (saved) to an external file. Here, we used a Stata function (the *esttab* command; see the corresponding code in Appendix E) to create *RegressionTable.tex*.

Once properly labelled and named according to their source, these external files can simply be incorporated into the research article with an explicit mention of their origin. In our example (Table 2), the following line would be introduced into our current LaTeX document:

```
% file created by RegressionPie.do
\include{RegressionTable.tex}
```

5.3 Linking Everything

Writing programs and using software that allows the easy export of output is a good start on the path toward more reproducible research, but, as mentioned in Section 3.3, the workflow needs special attention since several programs can export different outputs used later by other programs, among other concerns. To manage the workflow, a good practice is to use a "master" or "global program" that embeds all aspects of programs in a clear and logical way (Gentzkow & Shapiro,

	Analysis output (descriptive statistics, estimation results)	Graph
R	xtable, texreg *(.tex, .html, .doc)*, stargazer, tables	png(), jpeg(), pdf(), tiff() *(.png, .jpg, .pdf, .tiff)*
Stata	esttab *(.tex, .rtf)*, sutex *(.tex)*, latabstat *(.tex)*, putexcel *(.xlsx)*, outtable *(.tex)*	graph export *(.eps, .pdf, .wmf, .png)*
SAS	ods rtf *(.doc)*, ods html *(.html, .xls)*, ods pdf *(.pdf)*, ods tagsets.latex *(.tex)*	ods graphics *(.png, .tiff, .jpg, .ps)*
MATLAB	writetable *(.xls, .csv)*, xlswrite *(.xls)*	saveas *(.png, .eps, .pdf)*
Gams	gams2tbl *(.txt, .tex, .prn, .html)*, gdxxrw, xlexport, xldump *(.csv, .xls)*	gnuplot, gnuplotxyz *(.png)*
Mathematica	Export[ ] (*.xls*), CloudExport[, "pdf"] *(.pdf)*	Export[ ] *(.gif, .jpg)*

Table 3: Useful Tools for Reproducible Output (Output Formats in Italics).

2014). Such a master program can be written within a specific software (R, Stata, Gams, ...). In example 1 of Figure 10, we show an example of a Stata program that makes successive calls to different Stata scripts. An alternative is to write a shell script or a batch file in the Windows operating system, as in example 2 of Figure 10.[23] One great advantage is that batch files can successively call various programs as in example 2 of Figure 10, where R and Stata scripts are called by the global.bat file.

```
EXAMPLE 1 (Stata code): global.do

// Definition of the project's directory
local CodeFolder "c:/ApplePie/Progs"
cd "`CodeFolder'"

// Preparation of the data
do DataPreparation.do

// Some analysis code
do AnalysisCode.do

// Production of figures
do OutputCode.do

// Production of the paper
do MakingPaper.do
```

```
EXAMPLE 2 (Batch file): global.bat

REM Definition of the project's directory
set CodeFolder="C:\ApplePie\Progs"
cd %CodeFolder%

REM Preparation of the data
R CMD BATCH DataPreparation.R

REM Some analysis code
stata-se /e do AnalysisCode.do

REM Production of figures
stata-se /e do OutputCode.do

REM Production of the paper
R CMD BATCH MakingPaper.R
```

Figure 10: Examples of "master" Programs in Stata (Left Panel) or in Batch (Right Panel).

Makefile is a more powerful alternative to a batch file. Makefile is very popular in many disciplines and, as mentioned by Wilson *et al.* (2014) and Millman & Pérez (2014), provides researchers

[23]Batch files can call any software. Note further that in some statistical software, special commands exist to call external programs. For example, R has the `RPython` package to call Python programs. Python has the `RPy2` extension to call R code. SAS has the `%sysexec` command for running other software, and Stata has the `rsource` to call R code. IPython can also be used as a system shell (Pérez & Granger, 2007).

with two major benefits. First, the dependencies between inputs, outputs and programs are explicit. Second, for each execution, the Makefile system records which files have been modified and checks their dependencies. Thus, when re-executed, only the parts that need to be modified are called and rerun.

In the Makefile example in Figure 11, the first line of each step defines the dependencies, whereas the second line indicates the command to execute. In part 1, the data set `WorkingDataset.dta` is generated from the text file `RawData.csv` and the Stata program `DataPreparation.do`. Then, only if `OutputCode.R` has been modified since the last compilation will the Makefile re-execute parts 3, 4 and 5 and use the existing elements computed in parts 1 and 2. This parsimonious feature will be greatly appreciated when a program's runtime becomes long.[24] In the same spirit as Makefile, new tools have recently emerged with additional features; for example, Snakemake (in Python) can automatically create a graphical image of the workflow (Köster & Rahmann, 2012).

```
# 1. Preparation of the data:
WorkingDataset.dta: RawData.csv DataPreparation.do
        stata-se -b do "DataPreparation.do"

# 2. Some analysis code:
StatisticalTable.tex: WorkingDataset.dta AnalysisCode.do
        stata-se -b do "AnalysisCode.do"

# 3. Production of two figures. The '%' character can be used as a shortcut:
Figure%.pdf: WorkingDataset.dta OutputCode.R
        Rscript "OutputCode.R"

# 4. Production of the paper (from figures, table and bibliography):
Paper.pdf: Paper.tex  biblio.bib  Figure1.pdf Figure2.pdf StatisticalTable.tex
        pdflatex "Paper.tex"

# 5. Production of a zip file
zip MyZipFile.zip  Paper.pdf  Paper.tex RawData.csv /
DataPreparation.do AnalysisCode.do  OutputCode.R
```

Figure 11: Implementation of the Workflow in Makefile.

5.4 Creating Reproducible Documents

Automating the entire production process, including the generation of external files (data sets, results), even if mandatory for achieving a reproducible document, can be a tedious exercise, even if simplified by the use of global programs such as those described previously.

[24] A Makefile has no extension. The Unix program `Make` or `Make for Windows (GnuWin)`, or `Cygwin` is needed to compile this file (see also "cake", an early attempt to improve "make" for reproducible research in Claerbout & Nichols, 1989).

Another option is to directly write *reproducible research documents* following the idea of *literate programming* introduced by Knuth (1984, 1992). *Reproducible research documents* were primarily conceived for improving the readability of programs and making code and text that is glued and linked together. This notion is extended and revisited by Gentleman & Temple Lang (2007), who propose the concept of a *compendium*, i.e., a dynamic document (or package) embedding a mixture of code and text, that combines the power of a programming language with the readability of a documentation language. This idea has been recently implemented in the R package `rrtools`.

The basic structure of a reproducible research document (see the left panel of Figure 12) follows a logic of sequences of commands in some programming language ("*code chunks*") embedded in the text (or "*text chunks*" embedded in the code). Note that the text parts will be written in a specific narrative language ("*markup language*") that is not the programming language of the statistical software. In the same way that a piece of code has to be executed to obtain results, the document itself is compiled to obtain both results and formatted text as output. Thus, the output document will be identically structured, with code replaced by results (see the right panel of Figure 12).

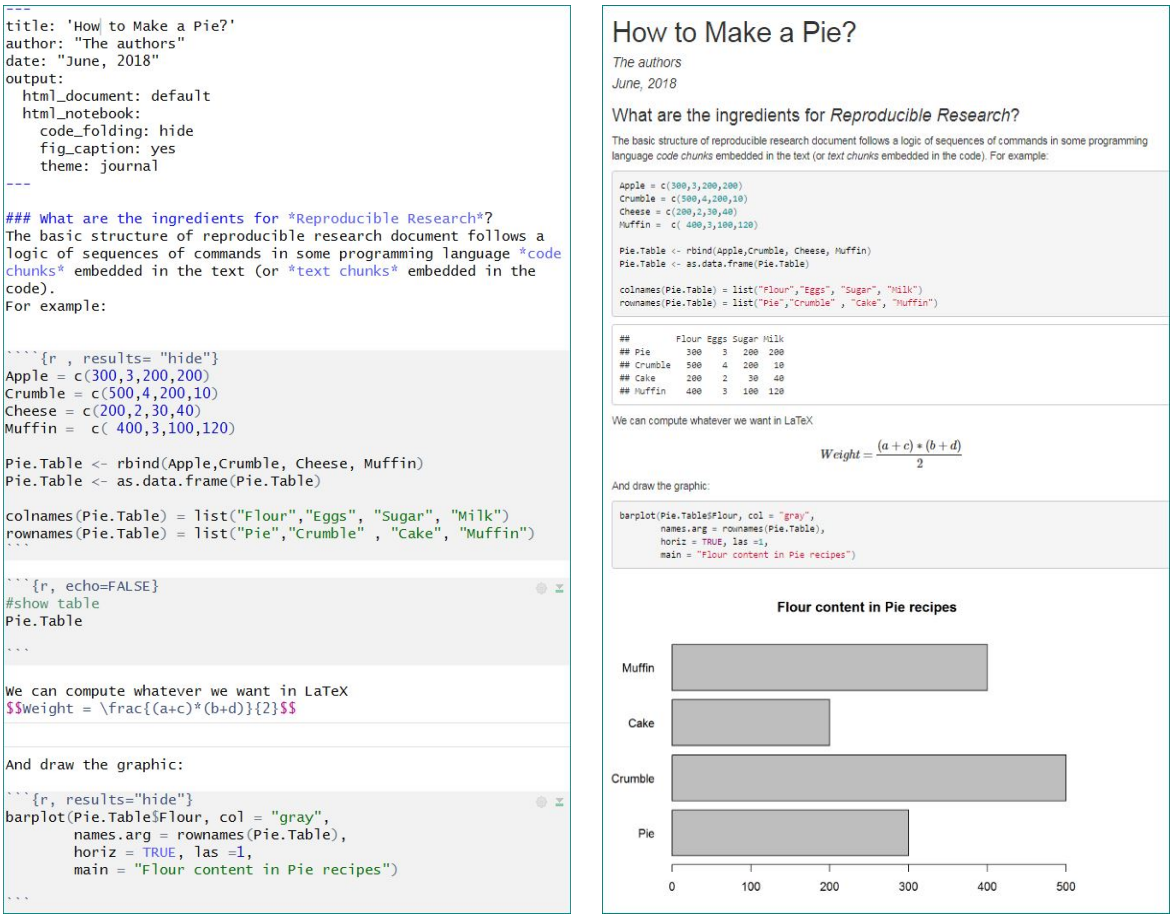

Figure 12: Example of a Reproducible Research Document (Left), and Its Resulting Report (Right).

For those using LaTeX and R, it is straightforward to perform literate programming using Sweave (Meredith & Racine, 2009), a package embedded as a native package in R. For those less familiar with LaTeX or willing to produce documents in various output formats (e.g., tex, html, doc, pdf), there is a more recent tool that uses a simplified markup language called R Markdown. It uses Markdown as a narrative language, knitr for compilation (Xie, 2015) and pandoc for output format conversion (MacFarlane, 2016).[25] By using Sweave or R Markdown, one can create a document written in LaTeX or Markdown that includes the statistical analysis within R chunks.

Other popular statistical software have included, more or less recently, the same type of literate programming tools. At last, Stata 15 (launched in June 2017) offers new native commands (*dyndoc*, *putdocx*, *putpdf*) that allow one to create html, Word or pdf documents (respectively) with text, code and embedded results from usual do files.[26] Recently Rodriguez (2017) released *markstat*, a new Stata command written in the R Markdown spirit, which seems to be very handy since a single input file can produce html, tex and pdf formats.[27] Writing the narrative parts in a very simple and light style using the Markdown language (rather than in LaTeX or html) is the new step in literate programming.

A selection of the main literate programming tools is presented in Table 4, and additional information can be found in Appendix F. Most of these tools are embedded in statistical software (as is R Markdown). Others can be both external and specific to one software environment (such as StatRep for SAS) and require several manipulations to obtain the final report. However, other external software options such as StatWeave allow the compilation of the entire document using different programming languages (e.g., R, SAS, Stata). Covering the largest set of output formats and markup languages is the current challenge in developing literate programming tools.[28]

Due to the flexibility of the concept, not only will the chunk of output adjust to any change in the code but the text itself can also be made dynamic using special commands. For example, if one wants to write a description of a data set with quantitative information ("*inline code*" as part of a narrative text), one can automatically use the average of a variable or a count of something using these commands. The logic is the same with different syntaxes depending on the software: `\Sexpr{}` for Sweave (R) ; `‘r expression‘` for R Markdown (R); `\Stataexpr{}` for StatWeave (Stata, R, SAS, ...) ; and `‘s expression‘` for *markstat* (Stata). Leisch (2006) and Gentleman

[25] Markdown has a plain text appearance with simple visual markup (Millman & Pérez, 2014). Pandoc converts files from one markup format to another (e.g., Markdown, LaTeX, html, Microsoft Word docx, LibreOffice odt).

[26] To fill the gap, several user-developed initiatives were developed with tools made available on the web, but their syntax was not straightforward, and some functionalities were limited. We can cite the *webdoc* and *texdoc* commands that allow the compilation of html, Markdown and LaTeX documents (Jann, 2016, 2017) and the *MarkDoc* command (Haghish, 2016a,b).

[27] On his website, Rodriguez provides examples written both with *dyndoc* (*putpdf*) and *markstat* to compare their syntax, demonstrating the simplicity of his command.

[28] For example, R Markdown is more complete than the initial Sweave tool, allowing users to use more markup languages and allowing for the generation of tex, html, Beamer and Microsoft Word outputs from a single code.

Language used for:		Tool name	Output format	References
Code	**Text**			
Sweave-like tools				
R	LaTeX	Sweave	TeX, Beamer, PDF	Leisch (2002), Meredith & Racine (2009)
R, Python, SAS, SQL, ...	Markdown	R Markdown	HTML, PDF, MS Word, Beamer, ...	Xie (2015), Gandrud (2015), Xie *et al.* (2018)
R, SAS	LaTeX, noweb	SASWeave	TeX, PDF	Lenth & Højsgaard (2007), Morrisson & Karafa (2012)
R, SAS, MATLAB, Stata, ...	LaTeX, OpenOffice	StatWeave*	TeX, ODT	Lenth & Højsgaard (2011), Lenth (2012)
Stata	Markdown	Markstat	TeX, PDF, HTML	Rodriguez (2017)
SAS	LaTeX	StatRep	TeX, PDF	Arnold & Kuhfeld (2012, 2015), Morrisson & Karafa (2012)
MATLAB	plain text markup	Publish	TeX, MS Word, HTML, PDF	Hunt *et al.* (2014) McCarthy (2018)
R, Stata, MATLAB, Python, ...	plain text markup	Org-mode	TeX, PDF, HTML, ODT, ...	Schulte & Davison (2011), Schulte *et al.* (2012), Dominik (2019)
Notebooks				
Python, R, SAS, Stata**, MATLAB, Julia, ...	Markdown	Jupyter Notebook	HTML, rST, PDF	LeVeque (2009), Kluyver *et al.* (2016), de Kok (2016)
Mathematica	Wolfram language	Mathematica Notebook	HTML, PDF, TeX, ...	Wolfram Research, Inc. (2008)
R, Python, SAS, SQL, ...	Markdown	R Notebook	HTML, PDF, MS Word, Beamer, ...	Gandrud (2015)
MATLAB	Formatted text	Live Scripts	HTML, PDF	Hunt *et al.* (2014) McCarthy (2018)

*: Statweave, a software independent from Stata is no longer maintained.

**: IPyStata enables the use of Stata together with Python via the Jupyter notebook (de Kok, 2016).

Table 4: Literate Programming Tools.

& Temple Lang (2007) provide detailed introductory examples.

Interesting options exist to hide code if we do not want it in the final document, if we want to display the code without evaluating it, or if we want to hide the output if it is not needed in the document. Most of the commands presented in Table 4 allow this. Their syntax is software specific. Another interesting option (available in R Markdown) is the "cache=TRUE" option, which makes it possible to not run a code chunk that is time consuming and has already been run once.

Thanks to these tools, one can do literate programming and produce documents that are fully reproducible and contain all the required materials (narrative, code, outputs). Literate programming is now extended to "*notebooks*" (see Table 4), inspired by the Mathematica notebooks initiative,

which are growing in popularity. This concept follows the same logic and the same goals, namely, to produce easily reproducible and exportable documents with a single file embedding text and code in "cells". With notebooks, documents are now interactive, meaning that the code is automatically executed in each cell. While Sweave and R Markdown documents have to be compiled to obtain the output document, notebooks allow the user to execute chunks (or not) and see the results interactively on the fly in the source document.

The Jupyter Notebook, previously known as the IPython notebook system (Pérez & Granger, 2007), a project initially designed for Julia, Python and R (Ju-Pyt-e-R) users, offers an interactive data science framework for scientific computing across all programming languages (Toomey, 2016). In Jupyter, many other languages are supported and may even coexist in different cells in the same notebook document. Each cell returns the output of the desired language to the notebook interface.

Our experience shows that notebooks are great tools for interactively playing with hypotheses, subsamples or estimators or for testing small chunks of code on the fly. Notebooks are also a good way to share programs and results with coauthors, even if Sweave-like tools are better suited for final printing or for writing a companion paper. However, organizing all the steps of the workflow within a single literate programming document can be difficult, and it may be more convenient to use several literate documents, each devoted to a specific question or task.

6 Conclusion

In this paper, we propose three main principles and illustrate their implementation to improve reproducibility in EEE research projects and papers. The first principle, "organize your work", deals with the overall organization of files and the documentation of a research workflow. We provide elements and tools that may help researchers move from post-its to a more structured organization using notebooks or task documentation. We illustrate easy-to-follow guiding ideas to organize files and keep track of the workflow and demystify the use of tools used by other disciplines, such as version control and sharing platforms. Then, "code for others" recalls that, since code is everywhere, we should take care in how we write code that has to be read by others or later by our future self. We emphasize through simple examples the benefits of adopting layouts and naming conventions and show that modularizing the code to make it clear, simple, readable, and reusable is crucial. Finally, "automate as much as you can" is a proposal to avoid any manual treatment and to automate most, if not all, of the steps used in a research process to reduce errors and increase reproducibility.

Despite all the tools available and illustrated here, reproducible research remains a challenge for the scientific community. Many papers have emphasized the lack of reproducibility in EEE and have sent alerts to the community (Dewald *et al.*, 1988; McCullough & Vinod, 2003; Koenker &

Zeileis, 2009). Nevertheless, nonreproducible papers are still published (Hamermesh, 2013; Höffler, 2017), and some are cited as seminal references. So, how do we move on from the current state and convince our colleagues to change their habits? More importantly, who should we convince in the first place? There may be different paths, involving different actors, not to mention our own willingness to act.

Researchers continue to regard controlling, mastering or sometimes automating the overall process leading to a publication as a time-consuming constraint. It is true that even the most convinced of our readers may face some obstacles on the path towards more reproducible practices. A key element is that adopting, even partially, more reproducible practices is always better than carrying on with nonreproducible ones. Reproducible research should be seen as a process of progressive improvements. Simple day-to-day practices and solutions, mostly based on common sense, can easily be implemented in any research project, small or large. Moreover, many statistical software packages are improving their coding interfaces, and some are implementing notebooks (R, Python) or are compatible with Jupyter (Julia, Python, R, Stata, SAS). Thus, reproducible research is no longer a technical issue, and on-the-shelf tools are available for greatly improving EEE's usual cooking practices.

Teachers also have a role to play in promoting reproducible research and establishing new norms for any publication. We increasingly ask groups of students to work together on empirical toy models or to replicate some empirical papers. Then, our evaluation is based on end-of-course projects that resemble research projects. We should be able not only to show examples of our organizational skills and tools but also to teach students how to organize themselves and evaluate the overall reproducibility of their work. Some universities have started to promote reproducible practices, provide examples and teach principles, methods and tools in their doctoral programs (Höffler, 2013), and many initiatives have emerged to improve common practices (see, e.g., Stodden, 2014; Duvendack *et al.*, 2017). There are several Massive Open Online Courses (MOOCs) devoted to reproducible research on platforms such as Coursera or on the French platform FUN. Efforts towards a better numerical literacy of our students and colleagues should also be fostered by any means, including online teaching initiatives such as the Data and Software Carpentry, DataCamp or the TIER project.

Journals can play a major role in the quest for more reproducible research in EEE. Following Galiani *et al.* (2017) and Chang & Li (2017), we believe that more journals should provide clear incentives at the early stages of the publication process, should ask that original data and code be evaluated together with the article during or even before the traditional review process. Journals should also establish a procedure for data and code sharing, either privately or publicly, and provide resources (online repositories) in accordance. Such mandatory prepublication policies would

clearly signal the minimum replication standards for publication. The Data and Code Availability Policy, edited by the *American Economic Association* (AEA) and implemented by several EEE journals (see Appendix A), is a first move towards imposing strict necessary conditions on a paper's data and code before final acceptance. It follows some of the Transparency and Openness Promotion guidelines (Nosek *et al.*, 2015) implemented by more than 1000 journals in other disciplines. One further step would be to impose these conditions even before starting the traditional review process. If implemented, such a policy would not only emphasize the importance of replication in the paper evaluation but would also disseminate a clear message on what should be the minimal elements embedded within a publication. Some journals have already taken this step (*e.g.* the AEA) and have invested in partnerships and in infrastructures to perform prepublication verification.

Journals editors could be the frontline of an improved peer review process involving a strict evaluation of submitted material. Today, editors select referees for their ability to detect mathematical errors and mistakes in submitted documents and share with the authors the responsibility of the validity of the results. Similarly, editors in charge of empirical papers should choose at least one referee with some abilities to read, check and reproduce codes (Heroux, 2015) or rely on an associated technical editor or a trusted third-party to achieve this crucial validation task. One example is the *American Journal of Political Science* that collaborates with the *Odum Institute Data Archive* to verify replicability before acceptance for final publication (Christian *et al.*, 2018). In the case of confidential data, often mentioned as an obstacle to reproducibility, the editor could either set up a short-term contract between the journal referees and the confidential-data producers or use third-party platforms to ensure the integrity and the nondissemination of confidential data during the evaluation of the reproducibility of the results.

Research institutions' help could be twofold. First, we may expect these institutions to finance projects proposing technical solutions for enhancing data and code sharing or proposing tools improving, automating or even certifying the reproducibility of papers. We can already mention a few examples, such as the NSF funded Privacy Tools Project for sharing sensitive research data initiated by Harvard University. This project aims at developing secure repositories, together with legal instruments, enabling broader access to privacy-sensitive data sets. Other ideas include curators models allowing online data analysis with strict, human or automatic control of outputs (Crosas *et al.*, 2015).[29] It is worth mentioning also the new French Certification Agency for Scientific Code And Data (Pérignon *et al.*, 2019), which develops partnerships with other restricted data access centers and journals (*e.g.* AER) and proposes a certification attesting that the results in a given

[29] See the Inter-university Consortium for Political and Social Research (ICPSR) or the Harvard University Privacy tools project (Private Data Sharing Interface).

scientific article can be reproduced using the code and data provided by the author, even when the data are confidential. These initiatives provide technical help for both researchers and journals to enrich and facilitate the peer review process. Second, research institutions could condition their research grants on mandatory policies, requiring the reproducibility of results obtained in financed research projects. In the same spirit, the evaluation of researchers and researchers' publications could include some evaluation of the reproducibility or the reusability of their work for others.

Users and firms are also active in developing new technical alternatives for many of the tasks encountered in a research project. Recently created online platforms that provide secure solutions for archiving code and data on permanent repository platforms such as *Dataverse*, *figshare*, *Zenodo* or *RunMyCode* should be supported and promoted. Other platforms, such as *Code Ocean* (Staniland, 2018; Stata News, 2018), *ExecAndShare* (Hurlin *et al.*, 2014a,b) or *Binder*, that allow the direct execution of code are also promising solutions. Finally, other quite recent tools provide a whole ecosystem of technical solutions (e.g., *Docker*, *mlflow*).

Today, our community is facing difficulties in addressing methodological problems such as "p-hacking" (Benjamin *et al.*, 2018) or "HARking" (*Hypothesizing After the Results are Known*, see Kerr, 1998), and some doubt has been cast on science, either due to errors (Levitt, 1997; Hoxby, 2000; Donohue & Levitt, 2001; Reinhart & Rogoff, 2010), fraud (Duvendack *et al.*, 2017) or retraction. As a response, we need more rigorous, more transparent and more reproducible scientific processes to assess our results. We do not focus here on the related Open Science debate, which focuses on achieving the FAIR (*Findable Accessible Interoperable Reusable*) principle even if reproducible research principles, as exposed here, fit into this general trend. More reproducible research may even be the key element to tackle all these challenges and improve the way we create, comment and share our work. As the cherry on the cake, this process may also improve our productivity.

Acknowledgments

The authors wish to thank Christophe Bisière, Christine Boizot-Szantai, Philippe Bontems, Sylvain Chabé-Ferret, Fabrice Etilé, Pascal Lavergne, Steve Lawford, Chantal Le Mouël, Olivier de Mouzon, and Tim Richards for their helpful comments and suggestions, as well as all participants of the JRSS INRA-SFER-CIRAD conferences in Rennes (2010) and Lyon (2017), the food economics seminar at the Toulouse School of Economics (2016) and the Reproducible Research Workshop in Porto (2019). We also are grateful to Lise Frappier and Alexandra Coppolino for their help collecting journal publication data, and to Sandrine Guillaume for producing some figures. We wish to sincerely thank the two anonymous referees for their suggestions and useful comments.

References

Arnold, Tim, & Kuhfeld, Warren. 2012. *Using SAS and LaTeX to Create Documents with Reproducible Results.* `https://support.sas.com/resources/papers/proceedings12/324-2012.pdf`, 16 p.

Arnold, Tim, & Kuhfeld, Warren. 2015. *The StatRep System for Reproducible Research.* `http://support.sas.com/rnd/app/papers/statrep/statrepmanual.pdf`, 69 p.

Baiocchi, Giovanni. 2007. Reproducible research in computational economics: guidelines, integrated approaches, and open source software. *Computational Economics*, **30**(1), 19–40.

Barba, L. A. 2018. *Terminologies for Reproducible Research.* `https://arxiv.org/abs/1802.03311`.

Barreto, Humberto, & Howland, Frank. 2005. *Introductory Econometrics: using Monte Carlo simulation with Microsoft Excel.* Cambridge University Press.

Benjamin, Daniel J, Berger, James O, Johannesson, Magnus, Nosek, Brian A, Wagenmakers, E-J, Berk, Richard, Bollen, Kenneth A, Brembs, Björn, Brown, Lawrence, Camerer, Colin, Cesarini, David, Chambers, Christopher D., Clyde, Merlise, Cook, Thomas D., De Boeck, Paul, Dienes, Zoltan, Dreber, Anna, Easwaran, Kenny, Efferson, Charles, Fehr, Ernst, Fidler, Fiona, Field, Andy P., Forster, Malcolm, George, Edward I., Gonzalez, Richard, Goodman, Steven, Green, Edwin, Green, Donald P., Greenwald, Anthony G., Hadfield, Jarrod D., Hedges, Larry V., Held, Leonhard, Hua Ho, Teck, Hoijtink, Herbert, Hruschka, Daniel J., Imai, Kosuke, Imbens, Guido, Ioannidis, John P. A., Jeon, Minjeong, Jones, James Holland, Kirchler, Michael, Laibson, David, List, John, Little, Roderick, Lupia, Arthur, Machery, Edouard, Maxwell, Scott E., McCarthy, Michael, Moore, Don A., Morgan, Stephen L., Munafó, Marcus, Nakagawa, Shinichi, Nyhan, Brendan, Parker, Timothy H., Pericchi, Luis, Perugini, Marco, Rouder, Jeff, Rousseau, Judith, Savalei, Victoria, Schönbrodt, Felix D., Sellke, Thomas, Sinclair, Betsy, Tingley, Dustin, Van Zandt, Trisha, Vazire, Simine, Watts, Duncan J., Winship, Christopher, Wolpert, Robert L., Xie, Yu, Young, Cristobal, Zinman, Jonathan, & Johnson, Valen E. 2018. Redefine statistical significance. *Nature Human Behaviour*, **2**, 6–10.

Bilina, Roseline, & Lawford, Steve. 2012. Python for Unified Research in Econometrics and Statistics. *Econometric Reviews*, **31**(5), 558–591.

Björk, Bo-Christer, & Solomon, David. 2013. The publishing delay in scholarly peer-reviewed journals. *Journal of Informetrics*, **7**(4), 914–923.

Boswell, Dustin, & Foucher, Trevor. 2011. *The Art of Readable Code : Simple and Practical Techniques for Writing Better Code.* O'Reilly.

Buckheit, Jonathan B., & Donoho, David L. 1995. *WaveLab and Reproducible Research.* Lecture Notes in Statistics, vol. 103. Springer New York.

Butz, William P, & Torrey, Barbara Boyle. 2006. Some frontiers in social science. *Science*, **312**(5782), 1898–1900.

Card, David, & DellaVigna, Stefano. 2013. Nine Facts about Top Journals in Economics. *Journal of Economic Literature*, **51**(1), 144–161.

Chang, Andrew C, & Li, Phillip. 2017. A Preanalysis Plan to Replicate Sixty Economics Research Papers That Worked Half of the Time. *American Economic Review*, **107**(5), 60–64.

Christensen, Garret, & Miguel, Edward. 2018. Transparency, Reproducibility, and the Credibility of Economics Research. *Journal of Economic Literature*, **56**(3), 920–80.

Christian, Thu-Mai, Lafferty-Hess, Sophia, Jacoby, William, & Carsey, Thomas. 2018. Operationalizing the Replication Standard: A Case Study of the Data Curation and Verification Workflow for Scholarly Journals. *International Journal of Digital Curation*, **13**(1), 114–124.

Chuang, Erica, Pollock, Harrison Diamond, & Wykstra, Stephanie. 2015. Reproducible Research: Best Practices for Data and Code Management. *IPA : Innovations for Poverty Action*.

Claerbout, Jon. 1990. Active documents and reproducible results. *SEP*, **67**, 139–144.

Claerbout, Jon, & Nichols, Dave. 1989. Why active documents need cake. *SEP*, **61**, 341–344.

Clemens, Michael A. 2017. The meaning of failed replications: A review and proposal. *Journal of Economic Surveys*, **31**(1), 326–342.

Crosas, Merce, King, Gary, Honaker, James, & Sweeney, Latanya. 2015. Automating Open Science for Big Data. *ANNALS of the American Academy of Political and Social Science*, **659**(1), 260–273.

de Kok, Ties. 2016. Combine Stata with Python using the Jupyter Notebook. Stata Conference, Chicago 2016.

de Leeuw, Jan. 2001. *Reproducible Research. The Bottom Line*. `http://www.escholarship.org/uc/item/9050x4r4`.

Desquilbet, Loic, Granger, Sabrina, Hejblum, Boris, Legrand, Arnaud, Pernot, Pascal, Rougier, Nicolas P., de Castro Guerra, Elisa, Courbin-Coulaud, Martine, Duvaux, Ludovic, Gravier, Pierre, Le Campion, Grégoire, Roux, Solenne, & Santos, Frédéric. 2019. *Towards reproducible research*. Unité régionale de formation à l'information scientifique et technique de Bordeaux.

Dewald, William G, Thursby, Jerry G, & Anderson, Richard G. 1988. Replication in Empirical Economics: The Journal of Money, Credit and Banking Project: Reply. *American Economic Review*, **78**(5), 1162–1163.

Dominik, Carsten. 2019. *The Org Mode 9.2 Reference Manual*. 12th Media Services.

Donoho, David, Maleki, Arian, Rahman, Inam, Shahram, Morteza, & Stodden, Victoria. 2009. Reproducible Research in Computational Harmonic Analysis. *Computing in Science and Engineering*, **11**(1), 8–18.

Donohue, III, John J., & Levitt, Steven D. 2001. The Impact of Legalized Abortion on Crime. *The Quarterly Journal of Economics*, **116**(2), 379–420.

Donohue, III, John J., & Levitt, Steven D. 2008. Measurement Error, Legalized Abortion, and the Decline in Crime: A Response to Foote and Goetz. *The Quarterly Journal of Economics*, **123**(1), 425–440.

Dupas, Pascaline, & Robinson, Jonathan. 2013. Savings Constraints and Microenterprise Development: Evidence from a Field Experiment in Kenya. *American Economic Journal: Applied Economics*, **5**(1), 163–192.

Duvendack, Maren, Palmer-Jones, Richard, & Reed, W Robert. 2017. What Is Meant by "Replication" and Why Does It Encounter Resistance in Economics? *American Economic Review*, **107**(5), 46–51.

Ernst, Michael. 2012. *Version control concepts and best practices*. `https://homes.cs.washington.edu/~mernst/advice/version-control.html`.

Fomel, Sergey, & Claerbout, Jon F. 2009. Guest Editors' Introduction: Reproducible Research. *Computing in Science and Engineering*, **11**(1), 5–7.

Foote, Christopher L., & Goetz, Christopher F. 2008. The Impact of Legalized Abortion on Crime: Comment. *The Quarterly Journal of Economics*, **123**(1), 407–423.

Fowler, Martin, Beck, Kent, Brant, John, Opdyke, William, & Roberts, Don. 1999. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Professional.

Galiani, Sebastian, Gertler, Paul, & Romero, Mauricio. 2017. *Incentives for replication in economics*. Working Paper 23576. National Bureau of Economic Research.

Gandrud, Christopher. 2015. *Reproducible Research with R and RStudio Second Edition.* Chapman & Hall/CRC The R Series.

Gentleman, Robert, & Temple Lang, Duncan. 2007. Statistical Analyses and Reproducible Research. *Journal of Computational and Graphical Statistics*, **16**(1), 1–23.

Gentzkow, Matthew, & Shapiro, Jesse M. 2014. *Code and Data for the Social Sciences : a practitioner's guide.* University of Chicago mimeo.

Gleditsch, Nils Petter, & Metelits, Claire. 2003. The replication debate. *International Studies Perspectives*, **4**(1), 72–79.

Gorp, Pieter Van, & Mazanek, Steffen. 2011. SHARE: a web portal for creating and sharing executable research papers. *Procedia Computer Science*, **4**, 589–597.

Haghish, E. F. 2016a. markdoc: Literate programming in Stata. *Stata Journal*, **16**(4), 964–988.

Haghish, E. F. 2016b. Rethinking literate programming in statistics. *Stata Journal*, **16**(4), 938–963.

Hamermesh, Daniel S. 2007. Viewpoint: Replication in Economics (Réplication en science économique). *The Canadian Journal of Economics / Revue Canadienne d'Economique*, **40**(3), 715–733.

Hamermesh, Daniel S. 2013. Six decades of top economics publishing: Who and how? *Journal of Economic Literature*, **51**(1), 162–172.

Herndon, Thomas, Ash, Michael, & Pollin, Robert. 2014. Does high public debt consistently stifle economic growth? A critique of Reinhart and Rogoff. *Cambridge Journal of Economics*, **38**(2), 275–279.

Heroux, Michael A. 2015. ACM TOMS replicated computational results initiative. *ACM Transactions on Mathematical Software (TOMS)*, **41**(3), 13.

Hinsen, Konrad. 2015. ActivePapers: a platform for publishing and archiving computer-aided research [version 3]. *F1000Research*, **3:289**.

Höffler, Jan H. 2013. Teaching Replication in Quantitative Empirical Economics. *In: World Economics Association Conferences "the economics curriculum: towards a radical reformation".*

Höffler, Jan H. 2017. Replication and economics journal policies. *American Economic Review*, **107**(5), 52–55.

Höffler, Jan H. 2017. ReplicationWiki - Improving Transparency in the Social Sciences. *D-Lib Magazine*, **23**(3/4).

Höffler, Jan H, & Kneib, Thomas. 2013 (4). *Economics Needs Replication.* `http://ineteconomics.org/ideas-papers/blog/economics-needs-replication`.

Hoxby, Caroline M. 2000. Does Competition among Public Schools Benefit Students and Taxpayers? *American Economic Review*, **90**(5), 1209–1238.

Hoxby, Caroline M. 2007. Does Competition among Public Schools Benefit Students and Taxpayers? Reply. *The American Economic Review*, **97**(5), 2038–2055.

Hunt, Brian R., Lipsman, Ronald L., & Rosenberg, Jonathan M. 2014. *A Guide to MATLAB: For Beginners and Experienced Users Third Edition.* Cambridge University Press.

Hunter, John E. 2001. The Desperate Need for Replications. *Journal of Consumer Research*, **28**(1), 149–158.

Hurlin, C., Pérignon, C., & Stodden, V. 2014a. RunMyCode.org: a novel dissemination and collaboration platform for executing published computational results. *Open Science Framework.*

Hurlin, C., Pérignon, C., & Stodden, V. 2014b. RunMyCode.org: A Research-Reproducibility Tool for Computational Sciences, in Implementing Reproducible Research. *In:* V., Stodden, F., Leisch, & R., Peng (eds), *Implementing Reproducible Research.* Chapman & Hall/CRC The R Series.

Huschka, Denis. 2013. *Why should we share our data, how can it be organized, and what are the challenges ahead?* RatSWD German Data Forum.

Ioannidis, John PA. 2005. Why most published research findings are false. *PLoS Med*, **2**(8), e124.

Jann, Ben. 2016. Creating LaTeX documents from within Stata using texdoc. *Stata Journal*, **16**(2), 245–263.

Jann, Ben. 2017. Creating HTML or Markdown documents from within Stata using webdoc. *Stata Journal*, **17**(1), 3–38.

Kernighan, Brian W, & Pike, Rob. 1999. *The Practice of Programming.* Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Kerr, Norbert L. 1998. HARKing: Hypothesizing after the results are known. *Personality and Social Psychology Review*, **2**(3), 196–217.

Kluyver, Thomas, Ragan-Kelley, Benjamin, Pérez, Fernando, Granger, Brian E, Bussonnier, Matthias, Frederic, Jonathan, Kelley, Kyle, Hamrick, Jessica B, Grout, Jason, Corlay, Sylvain, Ivanov, Paul, Avila, Damián, Abdalla, Safia, Willing, Carol, & Jupyter development team. 2016. Jupyter Notebooks – a publishing format for reproducible computational workflows. *Pages 87–90 of: Positioning and power in academic publishing: players, agents and agendas: proceedings of the 20th international conference on electronic publishing. Amsterdam. IOS Press.*

Knuth, Donald E. 1984. Literate Programming. *The Computer Journal*, **27**, 97–111.

Knuth, Donald E. 1992. *Literate Programming.* Center for the Study of Language and Information.

Koenker, Roger, & Zeileis, Achim. 2009. On Reproducible Econometric Research. *Journal of Applied Econometrics*, **24**(5), 833–847.

Köster, Johannes, & Rahmann, Sven. 2012. Snakemake - a scalable bioinformatics workflow engine. *Bioinformatics*, **28**(19), 2520–2522.

LabsExplorer. 2019. *2019 review of the best electronic laboratory notebooks.* `https://www.labsexplorer.com/c/2019-review-of-the-best-electronic-laboratory-notebooks_197`.

Lagoze, Carl, & Vilhuber, Lars. 2017. O Privacy, Where Art Thou? Making Confidential Data Part of Reproducible Research. *CHANCE*, **30**(3), 68–72.

Leek, Jeffrey T, & Peng, Roger D. 2015. Opinion: Reproducible research can still be wrong: Adopting a prevention approach. *Proceedings of the National Academy of Sciences*, **112**(6), 1645–1646.

Leisch, Friedrich. 2002. Sweave: Dynamic Generation of Statistical Reports Using Literate Data Analysis. *Compstat 2002 - Proceedings in Computational Statistics*, 575–580.

Leisch, Friedrich. 2006. *Sweave User Manual.* `https://stat.ethz.ch/R-manual/R-devel/library/utils/doc/Sweave.pdf`.

Lenth, Russell V. 2012. *StatWeave Users' Manual.* `http://homepage.divms.uiowa.edu/~rlenth/StatWeave/StatWeave-manual.pdf`.

Lenth, Russell V, & Højsgaard, Søren. 2007. SASweave: Literate Programming Using SAS. *Journal of Statistical Software*, **19**(8), 1–20.

Lenth, Russell V, & Højsgaard, Søren. 2011. Reproducible statistical analysis with multiple languages. *Computational Statistics*, **26**(3), 419–426.

LeVeque, Randall J. 2009. Python Tools for Reproducible Research on Hyperbolic Problems. *Pages 19–27 of: Special issue on Reproducible Research*. Computing in Science and Engineering (CiSE).

Levin, Lois. 2006. SAS Programming Guidelines. *SUGI 31 Proceedings - Paper 123-31*.

Levitt, Steven D. 1997. Using Electoral Cycles in Police Hiring to Estimate the Effect of Police on Crime. *The American Economic Review*, **87**(3), 270–290.

Levitt, Steven D. 2002. Using Electoral Cycles in Police Hiring to Estimate the Effects of Police on Crime: Reply. *American Economic Review*, **92**(4), 1244–1250.

Long, J Scott. 2009. *The workflow of data analysis using Stata*. Stata Press College Station, TX.

MacFarlane, John. 2016. *Pandoc User's Guide*. `https://pandoc.org/README.pdf`.

Martinson, Brian C, Anderson, Melissa S, & de Vries, Raymond. 2005. Scientists behaving badly. *Nature*, **435**, 737–738.

McCarthy, Ed. 2018. *Foundations of Computational Finance With MATLAB*. Wiley.

McCrary, Justin. 2002. Using Electoral Cycles in Police Hiring to Estimate the Effect of Police on Crime: Comment. *American Economic Review*, **92**(4), 1236–1243.

McCullough, B. D. 2009. Open Access Economics Journals and the Market for Reproducible Economic Research. *Economic Analysis and Policy*, **39**(1), 117–126.

McCullough, B. D. 2018. Quis Custodiet Ipsos Custodes? Despite Evidence to the Contrary, the American Economic Review Concluded That All Was Well with Its Archive. *Economics: The Open-Access, Open-Assessment E-Journal*, **12**(2018-52), 1–13.

McCullough, B. D., & Vinod, H. D. 2003. Verifying the Solution from a Nonlinear Solver: A Case Study. *American Economic Review*, **93**(3), 873–892.

McCullough, B. D., McGeary, Kerry Anne, & Harrison, Teresa D. 2006. Lessons from the JMCB Archive. *Journal of Money, Credit and Banking*, **38**(4), 1093–1107.

Meredith, Evan, & Racine, Jeffrey S. 2009. Towards Reproducible Econometric Research: The Sweave Framework. *Journal of Applied Econometrics*, **24**(2), 366–374.

Miara, Richard J, Musselman, Joyce A, Navarro, Juan A, & Shneiderman, Ben. 1983. Program indentation and comprehensibility. *Communications of the ACM*, **26**(11), 861–867.

Millman, K Jarrod, & Pérez, Fernando. 2014. Developing open source scientific practice. *Chap. 6, pages 149–183 of: Implementing Reproducible Research*. CRC Press, Stodden, Victoria and Leisch, Friedrich and Peng, Roger D (ed.).

Morrisson, Shannon M, & Karafa, Matthew T. 2012. *Reproducible Research Two Ways: SASweave vs StatRep*. `https://www.mwsug.org/proceedings/2012/PH/MWSUG-2012-PH09.pdf`.

Nagler, Jonathan. 1995. Coding style and good computing practices. *Political Science and Politics*, **28**(3), 488–492.

Nosek, B. A., Alter, G., Banks, G. C., Borsboom, D., Bowman, S. D., Breckler, S. J., Buck, S., Chambers, C. D., Chin, G., Christensen, G., Contestabile, M., Dafoe, A., Eich, E., Freese, J., Glennerster, R., Goroff, D., Green, D. P., Hesse, B., Humphreys, M., Ishiyama, J., Karlan, D., Kraut, A., Lupia, A., Mabry, P., Madon, T., Malhotra, N., Mayo-Wilson, E., McNutt, M., Miguel, E., Paluck, E. Levy, Simonsohn, U., Soderberg, C., Spellman, B. A., Turitto, J., VandenBos, G., Vazire, S., Wagenmakers, E. J., Wilson, R., & Yarkoni, T. 2015. Promoting an open research culture. *Science*, **348**(6242), 1422–1425.

Pérez, Fernando, & Granger, Brian E. 2007. IPython: a System for Interactive Scientific Computing. *Computing in Science and Engineering*, **9**(3), 21–29.

Pérignon, Christophe, Gadouche, Kamel, Hurlin, Christophe, Silberman, Roxane, & Debonnel, Eric. 2019. Certify reproducibility with confidential data. *Science*, **365**(6449), 127–128.

Pesaran, Bahram, & Pesaran, M Hashem. 2010. *Time Series Econometrics Using Microfit 5.0: A User's Manual*. Oxford University Press, Inc.

Pesaran, Hashem. 2003. Introducing a Replication Section. *Journal of Applied Econometrics*, **18**(1), 111.

Peters, Tim. 2004. *Pep 20–the zen of Python*. `https://www.python.org/dev/peps/pep-0020/#id3`.

Racine, Jeffrey S. 2019. Energy, economics, replication & reproduction. *Energy Economics*, **82**, 264–275.

Reinhart, Carmen M., & Rogoff, Kenneth S. 2010. Growth in a Time of Debt. *American Economic Review*, **100**(2), 573–78.

Rodriguez, German. 2017. Literate data analysis with Stata and Markdown. *Stata Journal*, **17**(3), 600–618.

Rothstein, Jesse. 2007. Does Competition Among Public Schools Benefit Students and Taxpayers? Comment. *American Economic Review*, **97**(5), 2026–2037.

Rougier, Nicolas P., Hinsen, Konrad, Alexandre, Frédéric, Arildsen, Thomas, Barba, Lorena A., Benureau, Fabien C.Y., Brown, C. Titus, de Buyl, Pierre, Caglayan, Ozan, Davison, Andrew P., Delsuc, Marc-André, Detorakis, Georgios, Diem, Alexandra K., Drix, Damien, Enel, Pierre, Girard, Benoît, Guest, Olivia, Hall, Matt G., Henriques, Rafael N., Hinaut, Xavier, Jaron, Kamil S., Khamassi, Mehdi, Klein, Almar, Manninen, Tiina, Marchesi, Pietro, McGlinn, Daniel, Metzner, Christoph, Petchey, Owen, Plesser, Hans Ekkehard, Poisot, Timothée, Ram, Karthik, Ram, Yoav, Roesch, Etienne, Rossant, Cyrille, Rostami, Vahid, Shifman, Aaron, Stachelek, Joseph, Stimberg, Marcel, Stollmeier, Frank, Vaggi, Federico, Viejo, Guillaume, Vitay, Julien, Vostinar, Anya E., Yurchak, Roman, & Zito, Tiziano. 2017. Sustainable computational science: the ReScience initiative. *PeerJ Computer Science*, **3**(Dec), e142.

Sandve, Geir Kjetil, Nekrutenko, Anton, Taylor, James, & Hovig, Eivind. 2013. Ten Simple Rules for Reproducible Computational Research. *PLoS Comput Biol*, **9**(10), 1–4.

Santaguida, Vincent. 2010. *Folder and File Naming Convention - 10 Rules for Best Practice*. `http://www.exadox.com/en/articles/file-naming-convention-ten-rules-best-practice`.

Schulte, E., & Davison, D. 2011. Active Documents with Org-Mode. *Computing in Science Engineering*, **13**(3), 66–73.

Schulte, Eric, Davison, Dan, Dye, Thomas, & Dominik, Carsten. 2012. A Multi-Language Computing Environment for Literate Programming and Reproducible Research. *Journal of Statistical Software*, **46**(1), 1–24.

Schwab, Matthias, Karrenbach, Martin, & Claerbout, Jon. 2000. Making scientific computations reproducible. *Computing in Science & Engineering*, **2-6**, 61–67.

Staniland, Mark. 2018. *Nature Research journals trial new tools to enhance code peer review and publication*. `http://blogs.nature.com/ofschemesandmemes/2018/08/01/nature-research-journals-trial-new-tools-to-enhance-code-peer-review-and-publication`.

Stata News. 2018. *StataCorp LLC and Code Ocean partner to accelerate reproducible research with code*. `https://codeocean.com/press-release/stata`.

Stodden, V., Bailey, D.H., Borwein, J., LeVeque, R.J., Rider, W., & Stein, W. 2013. Setting the Default to Reproducible: Reproducibility in Computational and Experimental Mathematics.

Stodden, Victoria. 2014. The reproducible research movement in statistics. *Statistical Journal of the IAOS*, **30**(2), 91–93.

Stodden, Victoria, McNutt, Marcia, Bailey, David H., Deelman, Ewa, Gil, Yolanda, Hanson, Brooks, Heroux, Michael A., Ioannidis, John P.A., & Taufer, Michela. 2016. Enhancing reproducibility for computational methods. *Science*, **354**(6317), 1240–1241.

Sweeney, L, Crosas, M, & Bar-Sinai, M. 2015. Sharing Sensitive Data with Confidence: The Datatags System. *Technology Science*.

Toomey, Dan. 2016. *Learning Jupyter*. Packt Publishing, Limited.

Ushey, Kevin, McPherson, Jonathan, Cheng, Joe, Atkins, Aron, & Allaire, JJ. 2016. *packrat: A Dependency Management System for Projects and their R Package Dependencies*. R package version 0.4.8-1.

Van Noorden, Richard. 2011. The troubles with retractions. *Nature*, **478**, 26–28.

van Rossum, Guido, Warsaw, Barry, & Coghlan, Nick. 2001. PEP 8–style guide for Python code. *python.org*.

Vilhuber, Lars, Turrito, James, & Welch, Keesler. 2020. Report by the AEA Data Editor. *AEA Papers and Proceedings*, **110**(May), 764–75.

Vlaeminck, Sven, & Herrmann, Lisa-Kristin. 2015. Data Policies and Data Archives: A New Paradigm for Academic Publishing in Economic Sciences? *Pages 145–155 of:* Schmidt, Birgit, & Dobreva, Milena (eds), *New Avenues for Electronic Publishing in the Age of Infinite Collections and Citizen Science*. IOS Press.

Wilson, Greg, Aruliah, D. A., Brown, C. Titus, Chue Hong, Neil P., Davis, Matt, Guy, Richard T., Haddock, Steven H. D., Huff, Kathryn D., Mitchell, Ian M., Plumbley, Mark D., Waugh, Ben, White, Ethan P., & Wilson, Paul. 2014. Best Practices for Scientific Computing. *PLOS Biology 12(1)*, **e1001745**.

Wolfram Research, Inc. 2008. *Wolfram Mathematica Tutorial Collection: Notebooks and Documents*. Wolfram Research, Inc.

Xie, Yihui. 2015. *Dynamic Documents with R and knitr, Second Edition*. Chapman & Hall/CRC The R Series.

Xie, Yihui, Allaire, J., & Grolemund, Garrett. 2018. *R Markdown: The Definitive Guide*. Chapman & Hall/CRC The R Series.

Appendices for the paper:

"How To Make A Pie: Reproducible Research for Empirical Economics and Econometrics"

by V. Orozco, C. Bontemps, E. Maigné, V. Piguet,
A. Hofstetter, A. Lacroix, F. Levert, J.M. Rousselle

A Evolution of economics journal replication policies

Rank	Journal	Mandatory replication policy			
		2003	**2009**	**2020**	**2020 policy' details**
1	Am Econ Review	–	YES	YES	mandatory (data + code) sharing*
2	J Finance	–	–	YES	mandatory code, encourage data sharing*
3	Q J Economics	–	–	YES	mandatory (data + code) sharing*, AER
4	Econometrica	–	YES	YES	mandatory (data + code) sharing
5	J Financial Econ	–	–	–	encourage (data + code) sharing
6	J Political Econ	–	YES	YES	mandatory (data + code) sharing*, AER
7	Rev Financial Stud	–	–	–	
8	J Econ Theory	–	–	–	encourage (data + code) sharing
9	Rev Econ Studies	–	YES	YES	mandatory (data + code) sharing
10	J Econometrics	–	–	–	encourage (data + code) sharing
11	J Econ Literature	–	–	YES	mandatory (data + code) sharing*
12	J Monetary Econ	–	–	–	encourage (data + code) sharing
13	J Econ Perspectives	–	YES	YES	mandatory (data + code) sharing*
14	Rev Econ & Stat	–	YES	YES	mandatory (data + code) sharing
15	Eur Econ Review	–	–	–	encourage (data + code) sharing
16	Int Econ Review	–	–	–	
17	J Int Econ	–	–	–	mandatory data, encourage code sharing
18	Economic Journal	–	–	–	
19	J Public Econ	–	–	–	encourage (data + code) sharing
20	Game Econ Behav	–	–	–	encourage (data + code) sharing
21	RAND J Economics	–	–	–	
22	J Money Credit Bank	YES	YES	YES	mandatory (data + code) sharing*
23	Economic Theory	–	–	–	encourage data sharing
24	J Bus & Econ Stat	–	–	–	encourage (data + code) sharing
25	Economics Letters	–	–	–	mandatory data, encourage code sharing
...					
41	J Appl Econometrics	–	–	YES	mandatory data, encourage code sharing**
Specialized journals (data not available before 2019)					
	Eur Review of Agri Econ	n.a		YES	mandatory (data + code) sharing*
	Ecological Econ	n.a		–	encourage (data + code) sharing
	Am J of Agri Econ	n.a		YES	mandatory (data + code) sharing
	Food Policy	n.a		–	encourage (data + code) sharing
	Applied Econ	n.a		–	encourage data sharing
	Resource and Energy Econ	n.a		–	encourage (data + code) sharing

Ranks and columns for years 2003 and 2009 are from McCullough (2009).

Information for year 2020 has been verified by the authors on journal websites (last access 24-07-2020).

* Requested before publication. ^{AER} The journal follows the *American Economic Review* data availability policy.

** This journal also provides a replication section.

Table 5: Overview of EEE journals replication policies over time.

B Practitioner's checklist

Organize your work:	**Yes**	**No**	*see section*
Is your directory structure separating raw data from created ones, programs, documentation and outputs?	☐	☐	3.2
Do you have a naming convention for your files? More specifically for programs, are you able to quickly know their goal?	☐	☐	3.2
Is the whole pipeline of your programs and data clear (drawn or depicted somewhere)?	☐	☐	3.3 & 5.3
Have you defined a strategy for differentiating updated files from old ones?	☐	☐	3.4.2
Is it easy to compare different versions of your code?	☐	☐	3.4.2
Have you discussed with your co-authors (if any) how to share your documents and programs?	☐	☐	3.4.1 & 3.4.3
Do you keep track of your ideas, tests or record your notes and conversations with coauthors (if any)?	☐	☐	3.1.1
Do you have a backup procedure?	☐	☐	3.4.2
Code for others:			
Are your variable names explicit?	☐	☐	4.1.2
Do you use a convention for variables that you created vs original ones?	☐	☐	4.1.2
Do you use relative paths in your programs?	☐	☐	4.1.3
Could you run your code on another computer?	☐	☐	4.1.3
Is your code as generic as possible?	☐	☐	4.1.3
Do you have comments in your code?	☐	☐	3.1.2 & 4.2
Is your code understandable without any other documentation?	☐	☐	4.1 & 4.2
Automate as much as you can:			
Does your software record code in a readable format?	☐	☐	5.1
Did you avoid any cut-and-paste?	☐	☐	5.2 & 5.4
Is it easy to recall in which order you ran each of your programs?	☐	☐	5.3
Is it easy to track how each table, figure and result in your document was created?	☐	☐	3.3, 5.1 & 5.3
Is it easy to rerun all the programs (and find all the results)?	☐	☐	5.4

C GraphViz code used to draw the workflow of Figure 2

```
digraph G {
rankdir = RL;
node [width =2, height=0.7];
  subgraph cluster_data {
    style=invis;
    node [shape=box, style = rounded]

    rawdata [label = "Raw data"];
    working [label = "Working data set"];
    interm [label = <Intermediate files<BR />(data, results)>];

    node [style=dashed]
    final [label=<Final results<BR /><FONT POINT-SIZE="10">
    (Tables, Figures, Summaries)</FONT>>];
    publication [label = "Publication"];

    {rank=same; rawdata; working; interm; publication}
  }

  subgraph cluster_code {
    style=invis;
    node [shape = ellipse, fillcolor=gray73, style="filled"]

    dataprep [label = <Data preparation<BR />code>];
    analysis [label = "Analysis code"];
    codeout [label = "Code output"];
    coderes [label = <Code for<BR />presenting results>];

    node [fillcolor=gray93, style="filled"]
    method1 [label = "Method 1", width=1.5];
    method2 [label = "Method 2", width=1.5];

    {rank=same; dataprep; analysis; codeout; coderes}
    {rank=same; method1; method2}
    //dataprep -> analysis -> codeout -> coderes [invis];
    method1 -> analysis;
    method2 -> analysis;
  }

 rawdata -> dataprep -> working -> analysis -> interm -> codeout ->
 final -> coderes -> publication;
}
```

D Version control software details

Figure 13 shows that *repository* and *working copy* are the two key elements of the client/server architecture of distributed version control software. A *repository* is a database of changes; it contains all the historical versions of the files. It is possible to store code, text or image files, although versioning is designed to address files that people edit. The *working copy* is a directory that contains the project's files, but only the files that the user has chosen to track will have a history. All the modifications are made on this *working copy*. The *commit* function allows users to send the modifications to the *local repository* and to provide a description of the changes.

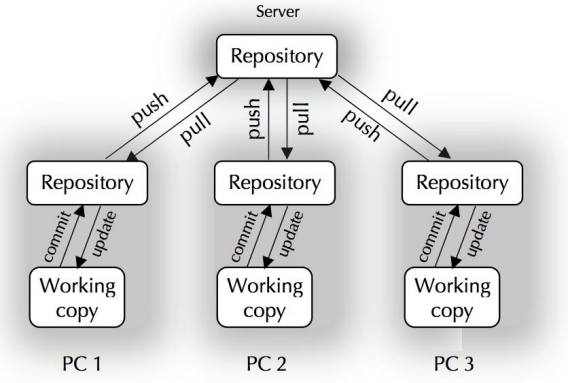

Figure 13: Distributed version control (Ernst, 2012).

To give other computers access to your project (and its history), you need to define a *central repository* and *push* your *local repository* to this *central repository*. The server where the *central repository* is located can be another personal computer, but most programmers use a web site to share their repositories. To obtain the latest version of a project, it is necessary to first *pull* it to retrieve a copy of the *central repository* to your *local repository* and then to *update* it in order to take into account the changes in your *working copy*. The most widely used distributed version control software is Git, and the Carpentry Software online lesson titled "Version Control with Git" is one of the numerous resources available for getting started.

While each computer has its own *repository* in distributed version control systems, it is not the case in centralized version control tools (Figure 14), where there is only one *repository* in the server. It is thus not possible to work offline with centralized version control software. Furthermore, as all changes are saved on the server, it does not provide the opportunity to take time to save and try changes before sharing them.

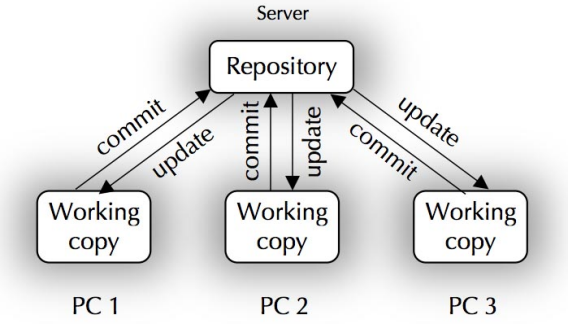

Figure 14: Centralized version control (Ernst, 2012).

E Stata code for Table 2

```
/*****************************************************************************/
/* Define dependent ("Y") and independent variables
(exogeneous ones "Xexo", endogeneous one "Xendo")
and instrumental variables ("IV")                                             */
/*****************************************************************************/
local Y      "Taste"
local Xexo   "Cooker_Level NbIngredients NbServers French Michelin"
local Xendo  "price"
local IV     "Eggs_Price Flour_Price Sugar_Price"

/*****************************************************************************/
/* Estimations : OLS and 2SLS                                                 */
/*****************************************************************************/
eststo OLS : reg `Y' `Xendo' `Xexo'
eststo IV  : ivreg2 `Y' `Xexo' (`Xendo' = `IV'), endog(`Xendo') first       ///
savefirst savefprefix(First_Stage)

/*****************************************************************************/
/* Export a nice table (LaTeX format) with both OLS and 2SLS estimation
results                                                                       */
/*****************************************************************************/
esttab OLS IV using RegressionTable.tex,                                      ///
scalar("N Observations"  "r2 R$^2$" "sargan Sargan statistic"                 ///
"sarganp Sargan p") b(3) not nonumber mtitle compress replace se              ///
star(* 0.10   ** 0.05    *** 0.01) label                                     ///
title(Regression table created using Stata \textit{esttab} command.           ///
\label{ExampleNiceReg})                                                       ///
addnote("Standard errors are in parentheses."                                 ///
"IV are input prices: sugar, flour and eggs prices."                          ///
"Sargan test is an overidentification test of all instruments."               ///
"This is a fictive example (no real interpretation).")                        ///
"\sym{*} p < 0.10, \sym{**} p < 0.05, \sym{***} p < 0.01."                    ///
mtitle("OLS" "2SLS") wide
```

F Details for Table 4

Language		Tool	Source extension	Output format	Chunk usage	Chunk syntax
Code	**Text**					
Sweave-like tools						
R	LaTeX	Sweave	.Rnw	TeX, Beamer, PDF	code	«chunckname»= *R code* @
R, Python, SAS, SQL, ...	Markdown	R Markdown	.Rmd	HTML, PDF, MS Word, Beamer, ...	code	```r *R code* ```
SAS	LaTeX	SASWeave	.SAStex	TeX, PDF	code	\begin{SAScode} *SAS code* \end{SAScode}
R			.Rtex			
SAS + R			.SASRtex, .RSAStex			
R	noweb		.Rnw			
SAS + R			.SASnw, .nwSAS			
R, SAS, MATLAB, Stata, ...	LaTeX, OpenOffice	StatWeave	.snw	TeX, ODT	code	\begin{Statacode} *Stata code* \end{Statacode}
Stata	Markdown	Markstat	.stmd	TeX, PDF, HTML	code	```s *Stata code* ```
SAS	LaTeX	StatRep	.tex	TeX, PDF	code	\begin{SAScode} *SAS code* \end{SAScode}
MATLAB	plain text markup	Publish	.m	MS Word, HTML, PDF, TeX	text	%%title %text
R, Stata, MATLAB, Python, ...	plain text markup	Org-mode	.org	TeX, PDF, HTML, ODT, ...	text	#+BEGIN_SRC <language> *code* #+END_SRC
Notebooks						
Python, R, SAS Stata, MATLAB, Julia, ...	Markdown	Jupyter Notebook	.ipynb	HTML, rST, PDF	code & text	
Mathematica	Wolfram language	Mathematica Notebook	.nb	HTML, PDF, TeX, ...	code & text	
R, Python, SAS, SQL, ...	Markdown	R Notebook	.Rmd	HTML, rST, PDF	code	```r *R code* ```
MATLAB	Formatted text	Live Scripts	.mlx	HTML, PDF	code & text	

A "chunk" is a block of code or text (see column "Chunk usage"). Its syntax is software specific (see column "Chunk syntax").

Table 6: Source extension and code chunk syntax for literate programming tools.