N° TSE - 933

July 2018

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

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

How To Make A Pie: Reproducible Research for Empirical Economics & Econometrics[*]

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

June 2018[††]

Abstract

Empirical economics and econometrics (EEE) research now relies primarily on the application of code to datasets. Handling the workflow linking datasets, programs, results and finally manuscript(s) is essential if one wish to reproduce results, which is now increasingly required by journals and institutions. We underline here the importance of "reproducible research" in EEE and suggest three simple principles to follow. We illustrate these principles with good habits and tools, with particular focus on their implementation in most popular software and languages in applied economics.

Keywords : Reproducibility; workflow; replication; literate programming; software

[*]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.

[†]Toulouse School of Economics, INRA, University of Toulouse Capitole, Toulouse, France

[‡]Observatoire du Développement Rural, INRA, Toulouse, France

[§]CESAER, Agrosup Dijon, INRA, Université Bourgogne-Franche-Comté, Dijon, France

[¶]CEE-M, Univ Montpellier, CNRS, INRA, Montpellier SupAgro, Montpellier, France

[‖]Univ. Grenoble Alpes, INRA, CNRS, INP (*Grenoble Institute of Engineering Univ. Grenoble Alpes*), GAEL, 38000 Grenoble, France

[**]SMART-LERECO, INRA, Rennes, France

[††]The authors wish to thank Christine Boizot-Szantai for her helpful comments and suggestions, Steve Lawford, Chantal Le Mouël, Christophe Bisière, Sylvain Chabe-Ferret, Fabrice Etilé, Tim Richards, Pascal Lavergne, Olivier de Mouzon, and Philippe Bontems, as well as all participants of the JRSS INRA-SFER-CIRAD conferences in Rennes (2010) and Lyon (2017) and the food economics seminar at the Toulouse School of Economics (2016). We also are grateful to L. Frappier and A. Coppolino for their help collecting journal publication data. We gratefully acknowledge the support and funding for this research from CATI CITISES, a program of the National Institute for Agronomic Research.

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 cases, we end up with a detailed list of ingredients, such as the source of the data, their contents or the datasets themselves and some code, attached to the publication as supplementary material.[1] However, the recipe used in making a paper is typically not entirely known to the reader. 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 the 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 tastes different, depending on the cook, as gourmets but not researchers appreciate. Finally, in other cases, the code works but is poorly written and 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 (see Chang & Li, 2017).[2]

Like anyone, researchers occasionally 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] Thus, most professional journal archives provide updates and sections devoted to the diffusion of corrections of published papers. Publishers do their best to minimize the impact of uncorrected results, but their efforts appear insufficient. 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*

[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).

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.

These previously unnoticed weaknesses and errors could 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., Hoffler & Kneib, 2013), have not been fully learned even if some initiatives have recently emerged. First, although few economic journals have an online archive for data and code and/or strict rules for ensuring replicability, this situation is slowly changing.[4] Some journals' policies, e.g., the *Journal of Applied Econometrics* and the *American Journal of Agricultural Economics* are leading the way toward improving data and code availability online and impose conditions related to such availability prior to final publication.[5] Second, platforms such as *RunMyCode* and *ExecAndShare* (Stodden *et al.* , 2014) have recently been created and allow for an online replication of papers. There is also a replication wiki in economics (Höffler, 2017).[6] *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). The *International Journal for Re-Views in Empirical Economics* (IREE) was founded in 2017 and is the first journal that intends to publish replication studies based on microeconomic data.[7] 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 non-reproducibility 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.[8] 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*". The objectives of the paper are threefold:

First, we propose three main principles to guide the actions of any researcher in EEE toward a higher degree of replicability for his work. The research process has, in recent decades, been driven

[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. See Appendix A for more details on the evolution of economic journal policies.

[5]The *American Journal of Political Science* (AJPS) claims that *"submitted replication materials will be verified to confirm that they do, in fact, reproduce the analytic results reported in the article. For quantitative analyses, the verification process is carried out by the Archive Staff at the Odum Institute for Research in Social Science, at the University of North Carolina"*.

[6]Launched by the Institute for New Economic Thinking, this wiki is a database for replication articles (published or working papers) and promotes reproducible research. Each replication study is described on a page that describes the availability of raw data, type and degree of replication, a link to the replication article, and so forth. Anyone can join and contribute to the wiki: `http://replication.uni-goettingen.de/wiki/index.php/Main_Page`.

[7]`https://www.iree.eu/`

[8]There is a spectrum of definitions embedded in the terms replication and reproducible in the literature (see Playford *et al.* , 2016). We will provide more details on these definitions in Section 2.

by new data, tools, and methods and by extensive use of computers and codes (see Butz & Torrey, 2006).[9] Therefore, most, if not every, researcher in EEE is now also a programmer, but probably one who lacks the skills, best practices and methods that are now standard in other disciplines such as computer science.[10] These principles are general enough to cover the entire research process. Applying them should help improve the research processes and can be done with different levels of effort. Researchers involved in workpackages embedded in funded research projects may identify that some principles are already implicit in their project milestones, but many smaller scale research projects involving only a couple of researchers should also benefit from a more structured approach of 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". Many researchers are unaware of the tools and best practices for efficiently writing code, and learning-by-doing is a common practice (Gentzkow & Shapiro, 2014; Millman & Pérez, 2014). Following LeVeque (2009), we argue that "*constructing a computer program isn't so different from constructing a formal proof*", and a greater knowledge of tools developed and used in other scientific communities could do no harm while improving the impact of any research.

Third, we address the problem of research programs done in collaboration and/or with co-authors. The average number of authors per paper published in the top-five economic journals has increased over time, and research is mostly done within research projects.[11] This evolution induces interactions between co-authors that need an improved work organization and potentially creates the need to document each step of the research process. We should also take into account the unusual length of the publishing process in EEE compared to other disciplines.[12] The length of the process affects a researcher's memory of how and where programs, data and results are located on his 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 the methods and tools used in EEE. We will focus on

[9]Note also that there is a significant decrease in the share of theoretical articles in the top-three economic 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.

[10]Typically, coding activity represents a large part of research activity in many sciences, including social science (Wilson *et al.* , 2014).

[11]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).

[12]According to Bjork & 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.

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).[13] We will also mention software that are less statistically oriented (Python, Julia). Some papers and books have described and illustrated reproducible practices using specific software such as R (Meredith & Racine, 2009; Allaire *et al.* , 2017; Gandrud, 2015; Xie, 2015), Stata (Gentzkow & Shapiro, 2014; Jann, 2016, 2017; Rodriguez, 2017), SAS (Arnold & Kuhfeld, 2012; Lenth & Højsgaard, 2007), Mathematica (Varian, 2013) or 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 and preferred software.

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). In this definition, the "other scientist" is either the researcher himself or a colleague, a student, or any "stranger".[14] This idea 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 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 dataset 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

[13]The CRAN website (the reference repository of R packages) has a Reproducible Research section listing all packages devoted or linked to this topic (https://cran.r-project.org/web/views/ReproducibleResearch.html).

[14]Donoho *et al.* (2008) extensively uses the notion of "strangers", noting that strangers are people we know or do not (co-authors, future or current students, referees, future employers). We should add the future-self, as the authors recall that "*it is not uncommon for a researcher who does not follow reproducibility to forget how the software in some long-ago project is used, or what its limitations are, or even how to generate an example of its application*".

refinements and subtle distinctions (Stodden, 2014). 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. Strictly speaking, this means that a replication dataset exists and is available and that managing the entire process, beginning with data pre-processing and ending with the paper, including all the steps in the descriptive analysis, modelization and results handling, can be repeated. Thus, this definition applies to a research project, composed of many elements, and not solely to a paper. In a sense, 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*" (Koenker & Zeileis, 2009).

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 code, of course, but also software and OS version. It all depends on the project and on the expected or intended level of precision or re-usability 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, Forthcoming). 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 (Reinhart & Rogoff, 2010).

We are not focusing here on a particular definition, and on the contrary, we examine all practical issues linked to any of these notions. Our goal here is to promote reproducibility in all its dimensions by providing advice, methods and tools. Hence, we will 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.[15]

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 (Gentzkow & Shapiro, 2014; Sandve *et al.* , 2013; 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's practices.[16] 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.

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 that is 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

[15]The open-science movement is not a reproducible research initiative, as it serves many other goals such as transparency, public data availability, and the efficiency of public effort.

[16]Useful habits, practices and tools concerning the data management preceding any empirical research project can be found at `http://odr.inra.fr/intranet/carto_joomla/index.php/qualite-donnees-INRA-SAE2`. Some principles and practices for empirical research that we describe here can also be applied to data management.

valid for the most common situation of a research done by a single author and leading to an output made of 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 of the recipe (phase and task of the project) which ingredients (e.g., data, methods) and what recipe (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 of managing a project consists of knowing in advance all the project's tasks and their contents, outcomes, organization and goals. It is useful to know the different persons 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 todo 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 material recording the research process and contains very useful information. However, from a long run 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.[17] 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 easy to search the document, without losing precious time turning a notebook's pages. Finally, unlike a physical notebook, the document,

[17]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.

if stored on a network drive, may be accessible even when out of 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).[18] 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. Evernote, OneNote and Etherpad are the most common ELNs used today. 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.

Finally, task management systems (TMSs) offer features for a more general collaborative and centralized task organization.[19] The usual way to share information with co-authors is to exchange e-mails 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 mixing different topics and possibly leading 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 to "*collaborate and share knowledge for the accomplishment of collective goals*".[20] In the design of TMSs, each task is represented as a card that can contain a description and attached exchanges with collaborators.[21] 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.[22] Several TMSs exist and are available on the web, e.g., Trello, Asana, MS Project and Wrike.[23]

[18]Baiocchi (2007) lists elements that can be found in Readme files. Additional materials from Dupas & Robinson (2013) contain a complete example.

[19]In the literature, TMSs are also called "project management systems" or "project portfolio management". The management of tasks through TMSs follows the Kanban methodology (Anderson, 2010).

[20]`https://en.wikipedia.org/wiki/Task_management`

[21]Usual text format is supported, but it is also possible to add web links and additional documents.

[22]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).

[23]Usually, the standard version is free, but additional services are offered in a paid version. As with any online

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.[24] Documenting tasks is an important aspect of documentation and should be considered one of the first things to do. During a project, regular updates should be planned and implemented to avoid any loss of implicit, and unwritten, information.[25]

Schematically, a task documentation is composed of a brief description of things to do, the people involved, the scheduled tasks and its state (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 (hypothesis 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 dataset describe the variables and their definitions and are familiar objects. However, a more global description of the data is needed to precisely characterize the dataset, its origin and its evolution over time 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. This information can be recorded using standard metadata (Dublin Core, or Data Documentation Initiative — DDI — for social sciences) that provides more precise information on the dataset itself.

Programs also need to be documented, and it is true that most element-documenting programs

service, these tools raise security and confidentiality concerns. Some commercial products that run locally also exist (for example, Jira as cited in Gentzkow & Shapiro (2014)).

[24]However, Gentzkow & Shapiro (2014) emphasize that while documentation has to be maintained, excessive documentation may be difficult to handle.

[25]This article does not address the question of documenting early steps that obviously also need to be recorded for future work (e.g., the grant proposal, human resources, data collection).

can be embedded in the code.[26] 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 state 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.[27]

3.2 Organizing files

Most, if not all, of the documents produced by researchers essentially consist of files organized in very different and personal ways. Undoubtedly, there is no perfect organization, but there are some elements to consider when organizing a research project, just as there are some tricks to know when organizing a library or a 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 unintentionally deleting 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 many 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 outputs (figures, tables, estimations, ..). Some practitioners also add a temporary folder (sandbox), saving temporary versions of code or documents

[26]Guidelines on how to write programs will be presented later on in the paper (see Section 4.1).
[27]For example, see Section 5 (computational details) in Koenker & Zeileis (2009).

for a small 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).

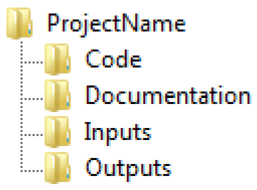


Figure 1: Example of well organized directory structure.

In the computer science community, file organization also uses simple ingredients such as file naming conventions. We suggest using these ideas, meaning that 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 co-authors, if any.

Before naming programs, one should have a clear idea of their goal. 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" dataset and for the statistical analyses based on that dataset (Nagler, 1995). This practice allows running new analyses without regenerating the dataset from scratch. The name of the program should therefore reflect its purpose. For example, we know immediately, and 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 for every file, not only programs.[28] An output can be named according to the name of the program that generated it and followed by a suffix. This simple rule allows us to locate 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 softwares. Following Long (2009), 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.

Using a naming convention for files can also help to manually manage file versions. It is standard

[28]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.

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.[29] This organization is still quite basic as it does not really help to follow the research process, generates many similar files and may still lead to confusions. We will see in Section 3.4.2 that more powerful tools exist and automatically manage file versions.

When working with others, having explicit conventions for directory structure and naming files has proven very helpful and allows co-authors to be able to understand and find what has been done and 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 (Bjork & 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 linking all the files and following 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 the files, all the links and the whole workflow should be as clear as possible and explicit. This process is key for the researcher and "strangers" to understand how programs (the circles in Figure 2) and data (rectangles) are related, how programs are related to one another, and their running order.

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, requiring either that one manually draw the workflow (e.g., Dia) or allowing one to code the elements of the workflow (boxes and links) and automatically generate the figure (e.g., GraphViz, RGraphViz),[30] and finally, some tools directly produce the workflow (SAS Entreprise Miner).

[29]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!

[30]A detailed example of GraphViz code is available in Appendix B.

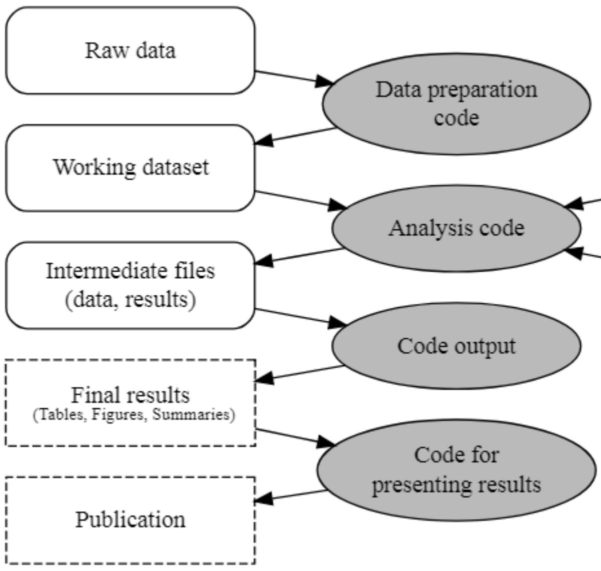


Figure 2: A simple example of a workflow, generated by GraphViz (code in Figure 13).

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.

Another solution without graphical representation will be presented in Section 5.3 and consists of managing the workflow directly and automatically using a dedicated software.

3.4 Handling files

Undoubtelly, a good organization of files is always a good thing for reproducibility, and can improve productivity. But the need of a 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 co-authors and time, it demands meaningful growth in the file handling strategy 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 e-mail 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. We propose in Table 1 a non-exhaustive 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 secured external server devoted to sending heavy files, such as FileSender available from RENATER[31] or WeTransfer. 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. 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.

We believe that the best way is to directly share a workspace and, if possible, have precise control of who is able to access to what (such as reading, writing, and erasing). Free tools exist to transform any computer into a server using FTP protocol (for example, FileZilla). This makes it possible to share a directory on a computer and to give access to collaborators for downloading files. Others tools such as Joomla, Agora, and SharePoint are designed to build and update websites, 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.[32]

[31] `https://filesender.renater.fr`. RENATER (National telecommunications network for Technology, Education and Research) is the French public broadband provider. The service allows people to transfer huge files that are downloadable through a time-limited link (Wetransfer) or only while your own web browser is open (as in JustBeamIt).

[32] In France, several research institution have developed hosting solutions. The INRA proposes a solution based on Microsoft's SharePoint to its members. The CNRS provides access to MyCoRe platform, which is open more broadly to the scientific research community while RENATER proposes also its widely open "PARTAGE" platform (`http://partage.renater.fr`).

Tool	Pros	Cons
Temporary exchange		
WeTransfer	Easy	Registration mandatory, 2 GB max, security concern (US based servers)
JustBeamIt	Easy (drag and drop), peer to peer	2 GB max
FileSender (RENATER)	Easy, up to 20GB, 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, off-line availability	Security concern (US-based servers), synchronization after off-line changes hazardous.
Google Drive	Online editor (docs, sheets, slides), browser access, easy sharing, comments in document, cloud-based storage	Security & privacy concern (advertising), Google account mandatory, no editing off-line
Joomla	Web based access, total control of user rights (private, public), unconstrained space	Content Management System (CMS) to be installed on a server, no off-line editing
Agora	Web based access, easy control of user rights, unconstrained space	Content management System (CMS) that needs to be installed on a server and administrated, no off-line editing
SharePoint	Easy integration with MS Office software	Needs to be installed on a server. Transparent use on an intranet, off-line editing hazardous.
PARTAGE (RENATER)	Webmail + shared task manager & calendar, messenger	Restricted to French institutions

Table 1: Tools for sharing files.

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 to apply very strict rules on file naming and file management to work properly. Visualizing the differences between two versions of a file and understanding or undoing changes made is not easy and requires specific tools. Software such as Notepad, Baloo, Total Commander, WinMerge, KDiff3 or muCommander are quite effective for text or Ascii files and allow 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, automatic versioning 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 share files with co-authors (see Appendix C for details on the key concepts of these tools).

Version management tools were developed and are mainly used by computer scientists and developers, for whom code is too precious to be wasted or re-written. In EEE, too, code is central to the research, and these tools could be used (Gentzkow & Shapiro, 2014; Millman & Pérez, 2014). A version management tool can follow the evolution of a project by memorizing its various modifications (Koenker & Zeileis, 2009), allowing one not only to have one or more backup copies of the files but also to work with different computers, to trace who has done what and when, and to eliminate any risk of overwriting changes due to a false manipulation. There are currently several programs for versioning with features that are more or less equivalent (see the comparison in Appendix C).

All version control software is associated with a centralized server that hosts all versions of the project. This server can be located on one's own personal computer, on a local server, or on an external server. At present, the most popular hosting servers are Sourceforge, GitHub, BitBucket, and CloudForge (Wilson *et al.* , 2014).[33]

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 that can be very difficult and time-consuming: People with different work habits (such as using Microsoft Word, OpenOffice, or LaTeX), problems associated with differences in software versions, and bibliography management are only a few examples.

Two common 'schools' still coexist, people working with MS Word, OpenOffice or Google docs on a WYSIWYG[34] basis and those working with LaTeX.[35] People using Word like it for 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. But, even with that feature, it is still difficult to cope with multiple changes made by multiple co-authors 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 and ShareLaTeX, that allow users, even those unfamiliar

[33]To guaranty intellectual property, confidentiality, that the software is up-to-date and legally compliant, in France, the scientific research community is recommended to use the Renater forge called Sourcesup.

[34]What You See Is What You Get.

[35]This distinction is usually found in the literature even if some front-end exist for LaTeX classifying it also in the WYSIWYG cases.

with LaTeX, to work (even simultaneously) with LaTeX files on a single online document.[36] The file is accessible through a 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.[37] The present paper was produced with Overleaf.

Basic recommendations can be provided: The use of a stable (over time), portable and unique file format (text, LaTeX) allows easy file sharing between co-authors 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, allowing researchers to share references and bibliographical files.

4 Code for others (including your future self)

For many, the feature of any piece of code is that it be 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; it should also be written in a clear way such that it is understandable by humans: such as the future self, coauthor, programmer, collaborators, or students. To quote Wilson *et al.* (2014), we recommend that one "*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 his own style based on his 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 program 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

[36] Overleaf is available at `http://www.overleaf.com/` while ShareLaTeX is available at `https://www.sharelatex.com/`. Note that ShareLaTeX joined OverLeaf recently.

[37] Overleaf has an history function that makes it possible to go back to previous saved versions and compare the actual version with older ones.

involved (single program, complete application) and regardless of the language used.[38] 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.

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 human's limited 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[39], "*check*" function for writing R packages[40] and "*formatR*" to adapt the code to the standard R layout.[41]

4.1.2 Conventions on naming

As for files, variable names have to be chosen to facilitate the reader's understanding and should also not contradict 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 "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 value 0 for men and 1 for

[38]Each programming community often provides its own principles (e.g., the "zen of Python" (Peters, 2004)).
[39]Pylint checks PEP 8 Python's conventions, see `https://www.pylint.org/`.
[40]`http://r-pkgs.had.co.nz/check.html`.
[41]`https://yihui.name/formatr/`.

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.[42]

Conventions can also help to label variables (Nagler, 1995): uppercase names for constants (`MY_CONSTANT`), 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 a "I_" or "i_" prefix (e.g., i_strawberry).[43] 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 the maintainability of the code, leading to greater re-usability 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 dataset. 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 path are stored in either local or global variables that are used as the reference directory. In the rest of the program, all the references to any path are relative (hence the use of dos-like commands with `../` and `cd`). Once as co-authors begin to follow the same directory structure, this practice will enhance compatibility and portability of the whole code on another machine.

The importance of keeping code as generic as possible is illustrated by the counterexample program 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 (Long, 2009). The code in Figure 6 provides a better solution.

[42]Length variables restricted to eight characters has long been an issue for naming variables. In this case, the CamelCaseNaming convention offers more alternatives for naming variables.

[43]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)

  # Example of use in a path used to save a graph
  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 in 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 in 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 dataset derived from input datasets ;
  data fin.ApplePie;
       set ini.sugar ini.apple;
  run;
```

Figure 4: Implementation and use of relative paths in code.

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

Figure 5: Example of a 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 a 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.[44] For example, the code in Figure 7 is a modular version of that in Figure 6.

[44]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.

```
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 a 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, explaining the purpose of the block, or 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 is "self-documenting" 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 a 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, 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

```
Program for pie cooking technology

Goal: Generate the Chocolate Foam estimations
Date: 2017/01/05
Author: Jamie Oliver
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.

avoid such disagreements.[45] 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 the 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.[46] 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-updating when modifying code (Wilson *et al.* , 2014).

4.3 Programing with pairs

In the programming phase, pair programming, where two researchers sit together while writing code, is recognized as a good way to improve the code review (Wilson *et al.* , 2014). This method can be particularly helpful in the first programming stage to define conventions (such as naming rules, comments, style, and file directories) among several programmers. It can also be valuable for debugging code.

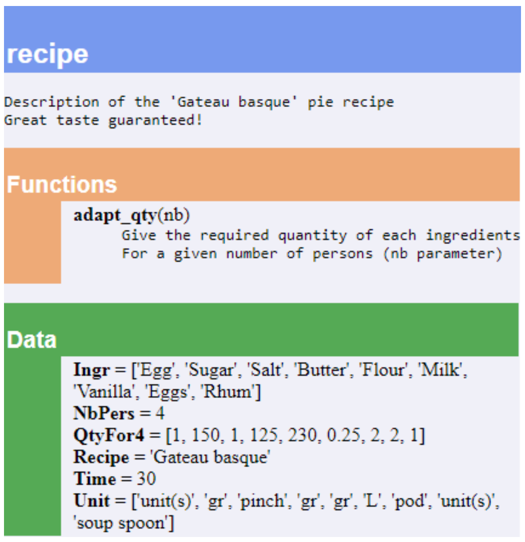
When pair programming is impossible because developers work in different places, one solution is to use a screen sharing platform (e.g., Skype, TeamViewer). Working at different times requires another type of organization that entails scheduling tasks (see 3.1) and versioning tools (see 3.4.2)

[45]Some software programs have commands or packages to address this issue, allowing the user to automatically save and load the computing environment. See, in this respect, the “checkpoint” R package (Racine, 2017) and “packrat” (Ushey *et al.* , 2016). For Stata, the “version” command indicates which Stata version is needed to run the code.

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

Python program	Automatic html documentation with pydoc

```python
# -*- coding: utf-8 -*-
"""
Description of the 'Gateau basque' pie recipe
Great taste guaranteed!
"""

RECIPE = 'Gateau basque'
TIME = 30 #minutes

INGR = ['Egg', 'Sugar', 'Salt', 'Butter', 'Flour',
        'Milk', 'Vanilla', 'Eggs', 'Rhum']

QTY_FOR_4 = [1, 150, 1, 125, 230,
             0.25, 2, 2, 1]

UNIT = ['unit', 'gr', 'pinch', 'gr', 'gr',
        'L', 'pod', 'unit', 'soup spoon']

def adapt_qty(nbpers):
    """Give the required quantity of each ingredients
    For a given number of persons (nb parameter)
    """
    for i, j, k in zip(INGR, QTY_FOR_4, UNIT):
        print('Ingredient', i, ': ',
              float(j)*nbpers/4, k,
              'required for a ', nbpers, 'person pie')

adapt_qty(4)
adapt_qty(6)
```



recipe

```
Description of the 'Gateau basque' pie recipe
Great taste guaranteed!
```

Functions

adapt_qty(nb)
```
    Give the required quantity of each ingredients
    For a given number of persons (nb parameter)
```

Data

Ingr = ['Egg', 'Sugar', 'Salt', 'Butter', 'Flour', 'Milk', 'Vanilla', 'Eggs', 'Rhum']
NbPers = 4
QtyFor4 = [1, 150, 1, 125, 230, 0.25, 2, 2, 1]
Recipe = 'Gateau basque'
Time = 30
Unit = ['unit(s)', 'gr', 'pinch', 'gr', 'gr', 'L', 'pod', 'unit(s)', 'soup spoon']

Figure 9: Example of automatic documentation generation (with Python docstring).

to avoid duplicated efforts.

5 Automate as much as you can

If any research workflow, such as the standard one represented in Figure 2 (Section 3.3), were conceived such that all elements (programs, data, files) are clearly linked within programs, it would be easy to automatize the entire process. In terms of reproducibility, it would greatly help any end user ("strangers" or "future-self") to use and reproduce, even partially, the research outputs. Unfortunately, this ideal vision of a 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 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 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 computation that can still be done without typing a single line of code. The tedious search for syntax errors and command names is 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 dataset. 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 dataset. 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 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.[47] The Reinhart & Rogoff (2010) case 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 hours of work. Thus, despite its great econometric features, a reproducible research approach is not feasible using Microfit.

Other programs, such as Stata (StataCorp *et al.* , 2007) 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 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,

[47] 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 potential compatibility issues across different versions of MS-Excel.

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 in order to generate results tables (see for example Table 2) consists of reporting the results cell by cell or by rearranging manually a raw output copied from a software output console and pasted somewhere else. The automatic generation of all the estimations, numbers, graphics and tables produced for an article is a minimal requirement for ensuring the traceability of any results from the raw dataset to the final paper. Even when carefully done, copy-paste practices do not guarantee that the results printed in a paper could be obtained again and should be seldom done or even avoided.

	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 Stata *esttab* command.

Many programs have included features (commands, packages) to export different types of outputs in a portable format (txt, rtf, LaTeX, and html, or PNG, JPEG and WMF for graphics; see Table 3).[48] Most programs also provide log files that report the executed code and the output, albeit without incorporating them.[49] 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 D) to create *RegressionTable.tex*.

Once properly labelled and named according to their source, these external files can simply

[48]For MS Word users, some statistical software can export results to .doc or .odt documents.

[49]Stata and Matlab have one log file with both executed instructions and results. For SAS, the log file contains only the executed instructions, and one *ODS* instruction can export all output (analysis results and figures).

	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).

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, 2014). That program can be written for and within the software used (R, Stata, Gams, . . .) with successive calls to external programs and datasets and is written in the software language (see the Stata example in the left panel of Figure 10). A more powerful alternative is to write a shell script (a batch file in Windows operating systems) or a Makefile, as in the right panel of Figure 10.[50] One great advantage is that batch files can successively call various programs (R and Stata in the example).[51]

Makefile is very popular in many disciplines and, as mentioned by Wilson *et al.* (2014) and Millman & Pérez (2014), provides researchers with two major benefits. First, the dependencies be-

[50] Batch files and Makefiles 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.

[51] IPython can also be used as a system shell (Pérez & Granger, 2007).

```
EXAMPLE 1 (Stata code): global.do
local CodeFolder "c:/ApplePie/Progs"
cd "‘CodeFolder’"
do DataPreparation.do
do AnalysisCode.do
do OutputCode.do
do MakingPaper.do
```

```
EXAMPLE 2 (Batch file): global.bat
set CodeFolder="C:\ApplePie\Progs"
cd CodeFolder
R CMD BATCH DataPreparation.R
stata /e do AnalysisCode.do
stata /e do OutputCode.do
R CMD BATCH MakingPaper.R
```

Figure 10: Examples of scripts in Stata (left panel) or in batch (right panel).

tween 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 re-run.

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 dataset `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.[52] This parsimonious feature will be greatly appreciated when a program's runtime becomes long.[53]

```
# 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.

[52]The consistency between the files included in `MyZipFile.zip` is guaranteed.

[53]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 (Claerbout & Nichols, 1989)).

5.4 Creating reproducible documents

Automating the entire production process, including the generation of external files (datasets, 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.

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, 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, combining the power of a programming language with the readability of a documentation language.

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, here, 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).

For people using LaTeX and R, it is straightforward to do literate programming using Sweave (Meredith & Racine, 2009), a package embedded as a native package in R. For people less familiar with LaTeX or willing to produce documents in various output formats (e.g., html, MS Word, pdf) there is a more recent tool, using a simplified markup language, called R Markdown.[54] It uses Markdown as narrative language,[55] knitr (Xie, 2015) for compilation, and pandoc for output format conversion (MacFarlane, 2016).[56] By using Sweave or R Markdown, one can create a document written in LaTeX or Markdown that includes the statistical analysis within R chunks.

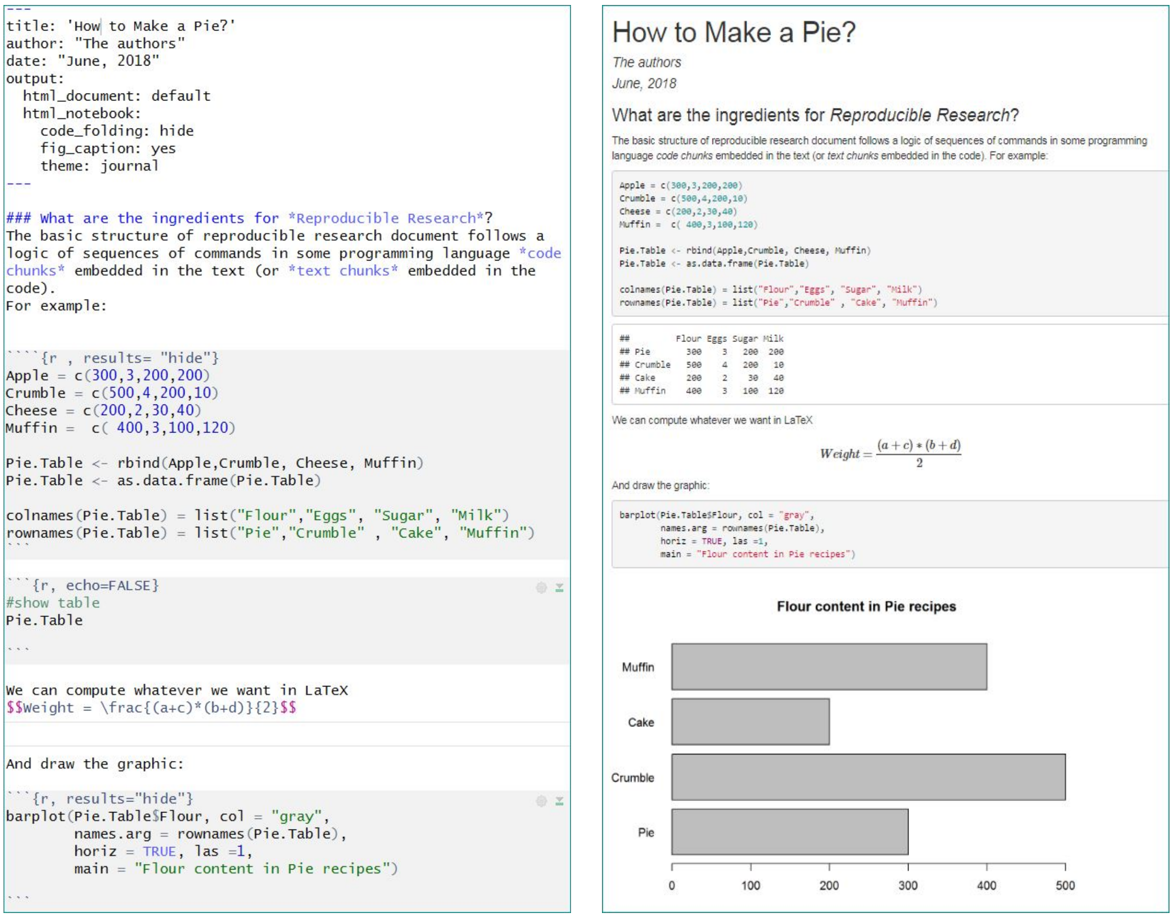
Other popular statistical software have included, more or less recently, the same type of literate programming tools. At last, Stata 15 (launched in June 2017) offers new native commands (*dyndoc*, *putdocx*, *putpdf*) that allow one to create html, Word or pdf documents (respectively) with text, code and embedded results, from usual do files.[57] Recently Rodriguez (2017) released

[54]Rmarkdown is a tool developed by Rstudio based on the knitr package (`http://rmarkdown.rstudio.com/`).

[55]Markdown has a plain text appearance with simple visual markup (Millman & Pérez, 2014).

[56]Pandoc converts files from one markup format to another (e.g., Markdown, LaTeX, html, Microsoft Word docx, LibreOffice odt).

[57]To fill the gap, several user-developed initiatives were developed, making tools available on the web, but their syntax was not straightforward, and some functionalities were limited. We can cite the *webdoc* and *texdoc* commands



~~~
---
title: 'How to Make a Pie?'
author: "The authors"
date: "June, 2018"
output:
  html_document: default
  html_notebook:
    code_folding: hide
    fig_caption: yes
    theme: journal
---

### What are the ingredients for *Reproducible Research*?
The basic structure of reproducible research document follows a
logic of sequences of commands in some programming language *code
chunks* embedded in the text (or *text chunks* embedded in the
code).
For example:


````{r , results= "hide"}
Apple = c(300,3,200,200)
Crumble = c(500,4,200,10)
Cheese = c(200,2,30,40)
Muffin =  c( 400,3,100,120)

Pie.Table <- rbind(Apple,Crumble, Cheese, Muffin)
Pie.Table <- as.data.frame(Pie.Table)

colnames(Pie.Table) = list("Flour","Eggs", "Sugar", "Milk")
rownames(Pie.Table) = list("Pie","Crumble" , "Cake", "Muffin")
~~~

```{r, echo=FALSE}
#show table
Pie.Table

```

We can compute whatever we want in LaTeX
$$Weight = \frac{(a+c)*(b+d)}{2}$$

And draw the graphic:

```{r, results="hide"}
barplot(Pie.Table$Flour, col = "gray",
        names.arg = rownames(Pie.Table),
        horiz = TRUE, las =1,
        main = "Flour content in Pie recipes")
```
```

How to Make a Pie?

The authors

June, 2018

What are the ingredients for *Reproducible Research*?

The basic structure of reproducible research document follows a logic of sequences of commands in some programming language *code chunks* embedded in the text (or *text chunks* embedded in the code). For example:

```
Apple = c(300,3,200,200)
Crumble = c(500,4,200,10)
Cheese = c(200,2,30,40)
Muffin =  c( 400,3,100,120)

Pie.Table <- rbind(Apple,Crumble, Cheese, Muffin)
Pie.Table <- as.data.frame(Pie.Table)

colnames(Pie.Table) = list("Flour","Eggs", "Sugar", "Milk")
rownames(Pie.Table) = list("Pie","Crumble" , "Cake", "Muffin")
```

```
##         Flour Eggs Sugar Milk
## Pie       300    3   200  200
## Crumble   500    4   200   10
## Cake      200    2    30   40
## Muffin    400    3   100  120
```

We can compute whatever we want in LaTeX

$$Weight = \frac{(a+c)*(b+d)}{2}$$

And draw the graphic:

```
barplot(Pie.Table$Flour, col = "gray",
        names.arg = rownames(Pie.Table),
        horiz = TRUE, las =1,
        main = "Flour content in Pie recipes")
```

Figure 12: Example of a reproducible research document (left), and its resulting report (right)

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.[58] 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 E. 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). The available output formats are linked to the markup language used and vary according to the tools employed. Covering the largest set of output formats and markup languages is the current challenge in developing literate programming

that allows to compile html, Markdown and LaTeX documents (Jann, 2016, 2017) and the *MarkDoc* command (Haghish, 2016a,b).

[58]On his website (`http://data.princeton.edu/stata/markdown/dyndoc`), Rodriguez provides examples written both with *dyndoc* (*putpdf*) and *markstat* to compare their syntax, demonstrating the simplicity of his command.
```

tools.[59]

| 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), Allaire *et al.* (2017) |
| R, SAS | LaTeX, noweb | SASWeave | TeX, PDF | Lenth & Højsgaard (2007), Morrisson & Karafa (2012) |
| R, SAS, Matlab, Stata, ... | LaTeX, OpenOffice | StatWeave[60] | 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 | Matlab documentation |
| R, Stata, Matlab, Python, ... | plain text markup | Org-mode | TeX, PDF, HTML, ODT, ... | Dominik (2010), Schulte *et al.* (2012) |
| **Notebooks** | | | | |
| Python, R, SAS, Stata,[61] Matlab, Julia, ... | Markdown | Jupyter Notebook | HTML, rST, PDF | LeVeque (2009), Kluyver *et al.* (2016) |
| Mathematica | Wolfram language | Mathematica Notebook | HTML, PDF, TeX, ... | Varian (2013) |
| R, Python, SAS, SQL, ... | Markdown | R Notebook | HTML, PDF, MS Word, Beamer, ... | Gandrud (2015) |
| Matlab | Formatted text | Live Scripts | HTML, PDF | Matlab documentation |

Table 4: Literate programming tools.

Due to the flexibility of the concept, not only will the chunk of output adjust to any change in the code, but also the text itself can 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 command‘` for R Markdown (R); `\Stataexpr{}` for

[59]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.

[60]Note that this software, independent from Stata, is no longer maintained.

[61]IPyStata enables the use of Stata together with Python via the Jupyter notebook (de Kok, 2016).

StatWeave (Stata, R, SAS, ...) ; and `‘s expression‘` for *markstat* (Stata). Leisch (2006) and Gentleman & 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 are software specific. Another interesting option (available in R Markdown) is the "cache=TRUE" option that 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, containing all the required materials (narrative, code, outputs). Literate programming is now extended into "*notebooks*" (see Table 4), inspired by the Mathematica notebooks initiative, which are growing in popularity.[62] 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 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 to interactively play with hypotheses, subsamples, estimators, or to test small chunks of code on the fly. Notebooks are also a good way to share programs and results to co-authors, even if Sweave-like tools are more suited for a final printing or for writing a companion paper. Organizing all the steps of the workflow within a single literate programming document can be difficult, though, and it may be more convenient to use several literate documents, devoting each one 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",

[62]Millman & Pérez (2014) use the terminology "literate computing" for this new generation of tools.

deals with the overall organization of our files, and the documentation of a research workflow. We provide elements and tools that can be highly beneficial for improving reproducibility and also to simplify day-to-day work. 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 by our future self later. 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 an injunction to avoid any manual treatment and to automatize most, if not all, steps used in a research process to reduce errors and increase reproducibility.

Despite all the tools available and illustrated here, reproducible research remains a current challenge for the scientific community. Many papers have emphasized the lack of reproducibility in EEE and have sent alerts to the community quite a long time ago (Dewald *et al.* , 1988; McCullough & Vinod, 2003; Koenker & Zeileis, 2009). Nevertheless non-reproducible papers are still as published as reproducible ones (Hamermesh, 2013; Höffler, 2017) and some are cited as seminal references.

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 his path toward more reproducible practices. A key element is that adopting, even partially, more reproducible practices is always better than carrying on with non-reproducible 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 big. Moreover, many statistical software packages are improving their coding interfaces, some are implementing notebooks (R, Python) or are compatible with Jupyter (Julia, Python, R, Stata, SAS). So reproducible research is no longer a technical issue and on-the-shelf tools are available for a great improvement in EEE's usual cooking practices.

Education also has a role to play in overcoming these obstacles. Many initiatives have emerged to improve common practices (see, e.g., Stodden, 2014; Duvendack *et al.* , 2017). Some universities have started to promote reproducible practices, provide examples and teach principles, methods and tools in their doctoral programs (Höffler, 2013). There is even a MOOC on Reproducible Research on the Coursera platform (`http://www.coursera.org/learn/reproducible-research`).[63]

Journals and institutions financing research, such as NSF, ERC, and ANR, as well as univer-

[63] One may also mention on-line programs such as the TIER program (`https://www.projecttier.org/about/about-project-tier/`).

sities and research centers, 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 and should ask that original data and code be evaluated together with the article during the review process. Such a mandatory pre-publication policy would clearly signal the minimum replication standards for publication. Institutions could also condition their research grants on mandatory policies requiring reproducibility of the financed research projects.

In EEE, as in many sciences, the patrimonial legacy of published papers structured in archives plays an important role. The JSTOR platform exhaustively compiles many EEE journals and documents but does not compile data or code, as if only the content of the papers was useful for future research. Initiatives to develop secure solutions for archiving code and data on permanent repository platforms should be encouraged; this archiving could be hosted by journals or by platforms such as *Zenodo* or *Runmycode*.[64]

Today, our community is facing difficulties to address methodological problems such as 'p-hacking' (Benjamin *et al.* , 2018) or 'HARking' (Hypothesizing After the Results are Known, Kerr (1998)), and some doubt has been casted on science, either due to errors (Reinhart & Rogoff, 2010), fraud (Duvendack *et al.* , 2017) or retraction.[65] As a response, we need more rigorous, more transparent and more reproducible scientific processes to assess our results.[66] Reproducible research may be the key element to tackle all these challenges and improve the way we create, comment and share our cooking recipes. Cherry on the cake, this process may also improve our productivity.

[64]The web-based platform *ExecAndShare* that allows for the direct execution of the code is also a promising solution.

[65]See the retraction watch website `http://retractionwatch.com/`

[66]We did not focus here on related Open Science debate, which focuses on achieving the FAIR (Findable Accessible Interoperable Reusable) principle. Obviously, research ingredients have to be shared, provided that the data are not subject to confidentiality issues.

## References

Allaire, JJ, Horner, Jeffrey, Bengtsson, Henrik, Hester, Jim, Qiu, Yixuan, Takahashi, Kohske, November, Adam, Caballero, Nacho, Ooms, Jeroen, Leeper, Thomas, Cheng, Joe, Oles, Andrzej, Marti, Vicent, Porte, Natacha, Xie, Yihui, & RStudio. 2017. Markdown Rendering for R. *R package version 1.8.*

Anderson, David J. 2010. *Kanban: Successful Evolutionary Change for Your Technology Business.* 1 edn. Blue Hole Press.

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

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

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

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, *et al.* . 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.

Bjork, 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.

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 S, & Miguel, Edward. Forthcoming. Transparency, reproducibility, and the credibility of economics research. *Journal of Economic Literature*, 93p.

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.

Cooper, Natalie, Hsing, Pen-Yuan, Croucher, Mike, Graham, Laura, James, Tamora, Krystalli, Anna, & Michonneau, Francois. 2017. *A guide to reproducible code in Ecology and Evolution.* Tech. rept. `http://www.britishecologicalsociety.org/wp-content/uploads/2017/12/guide-to-reproducible-code.pdf`.

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

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. 2010. *The Org Mode 7 Reference Manual - Organize Your Life with GNU Emacs*. Network Theory Ltd.

Donoho, David, Maleki, Arian, Rahman, Inam, Shahram, Morteza, & Stodden, Victoria. 2008. *15 Years of Reproducible Research in Computational Harmonic Analysis*. Tech. rept. Department of Statistics, Stanford University, Stanford.

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.

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*.

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

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*. Tech. rept. 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*. Tech. rept. University of Chicago mimeo.

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.

Hinsen, Konrad. 2015. ActivePapers: a platform for publishing and archiving computer-aided research [version 3; referees: 3 approved]. *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).

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

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

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, *et al.* . 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.

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.* Tech. rept. `https://stat.ethz.ch/R-manual/R-devel/library/utils/doc/Sweave.pdf`.

Lenth, Russell V. 2012. *StatWeave Users' Manual.* Tech. rept. `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.*

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

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

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

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., & 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.* Tech. rept. `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.

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

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`.

Playford, Christopher J, Gayle, Vernon, Connelly, Roxanne, & Gray, Alasdair JG. 2016. Administrative social science data: The challenge of reproducible research. *Big Data & Society*, **3**(2), 1–13.

Racine, Jeffrey. 2017. Energy, Economics & Replication. *McMaster University, Department of Economics Working Paper No. 2017-02.*

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.

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.* Tech. rept. `http://www.exadox.com/en/articles/file-naming-convention-ten-rules-best-practice`.

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.

StataCorp, LP, *et al.* . 2007. Stata Data Analysis and Statistical Software. *Special Edition Release*, **10**.

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

Stodden, Victoria, Leisch, Friedrich, & Peng, Roger D. 2014. *Implementing Reproducible Research.* Chapman & Hall/CRC, The R Series.

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 retractations. *Nature*, **478**, 26–28.

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

Varian, Hal R. 2013. *Economic and Financial Modeling with Mathematica®*. Springer.

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**.

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

# Appendices

## A Evolution of economic journal replication policies

Table 5 compiles information provided by McCullough (2009) for the years 2003 to 2009, augmented with data compiled by the authors from journal websites for the year 2017.

| **Rank** | **Journal** | **Mandatory replication policy** | | | |
|---|---|---|---|---|---|
| | | **2003** | **2009** | **2017** | actual policy |
| 1 | Am Econ Review | – | YES | YES | Mandatory data + code policy |
| 2 | J Finance | – | – | – | |
| 3 | Q J Economics | – | – | YES | Mandatory data + code policy[67] |
| 4 | Econometrica | – | YES | YES | Mandatory data + code policy |
| 5 | J Financial Econ | – | – | – | encourage data and code sharing |
| 6 | J Political Econ | – | YES | YES | replication policy |
| 7 | Rev Financial Stud | – | – | – | |
| 8 | J Econ Theory | – | – | – | |
| 9 | Rev Econ Studies | – | YES | YES | data + code |
| 10 | J Econometrics | – | – | – | |
| 11 | J Econ Literature | – | – | YES | Mandatory data + code policy |
| 12 | J Monetary Econ | – | – | – | encourage data and code sharing |
| 13 | J Econ Perspectives | – | YES | YES | Mandatory data + code policy |
| 14 | Rev Econ & Stat | – | YES | YES | Mandatory data + code policy |
| 15 | Eur Econ Review | – | – | – | encourage data and code sharing |
| 16 | Int Econ Review | – | – | – | |
| 17 | J Int Econ | – | – | – | encourage data and code sharing |
| 18 | Economic Journal | – | – | – | |
| 19 | J Public Econ | – | – | – | encourage data and code sharing |
| 20 | Game Econ Behav | – | – | – | encourage data and code sharing |
| 21 | RAND J Economics | – | – | – | |
| 22 | J Money Credit Bank | YES | YES | YES | Mandatory data + code policy |
| 23 | Economic Theory | – | – | – | encourage data sharing |
| 24 | J Bus & Econ Stat | – | – | – | encourage data and code sharing |
| 25 | Economics Letters | – | – | – | encourage data and code sharing |
| 41 | J Applied Econometrics | – | – | YES | data expected, code encouraged[68] |
| **Specialized journals** (data not available before 2017) | | | | | |
| | Eur Review of Agri Econ | n.a | | YES | Mandatory data + code policy |
| | Ecological Econ | n.a | | – | encourage data and code sharing |
| | Am J of Agri Econ | n.a | | YES | Mandatory data + code policy |
| | Food Policy | n.a | | – | encourage data and code sharing |
| | Applied Econ | n.a | | – | encourage data sharing |
| | Resource and Energy Econ | n.a | | – | encourage data and code sharing |

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

[67]The QJE has adopted the AER data availability policy.
[68]This journal also provides a replication section.

## B Example of GraphViz code to draw a workflow figure (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 dataset"];
    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;
}
```

Figure 13: Example of GraphViz code.

C Version control software details

In version control software, the *repository* and *working copy* are the two key elements (Figure 14). A *repository* is a database of all the historical versions of the project. It is possible to store code, text or image files, although versioning is designed to address files that people edit. The *working copy* contains a copy of all the files in the project. All the modifications are made on this *working copy.* The *commit* function allows users to send the modifications to the *repository* and provides a description of the changes. An *update* of the *working copy* is necessary to take into account amendments made on other computers. It allows users to retrieve the latest version of the project from the *repository.*

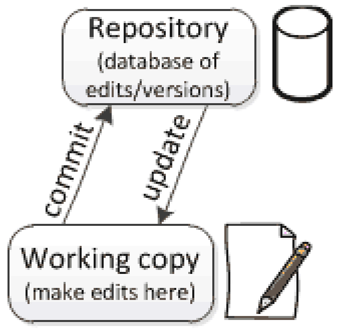


Figure 14: Principles of versioning (Ernst, 2012).

Version control systems can be classified according to the number of *repositories* that they use for each project (see Table 6). Figure 15 shows that centralized version control depends on only one *repository*, whereas a distributed version control uses multiple *repositories* (Figure 16). The latter case implies that each computer has its own *repository.* This means that a *commit* action acts only on your own *local repository.* To give other computers access to your changes, you need to *push* them to the *central repository.* Similarly, to obtain the last version of the project, it is necessary to *pull* it before you can *update* your *working copy.* According to Ernst (2012), "*distributed version control is more modern, runs faster, is less prone to errors, has more features, and is somewhat more complex to understand. You will need to decide whether the extra complexity is worthwhile for you*". An example for getting started with Git is also available in Cooper *et al.* (2017).

Software	Category	Comment
CVS	centralized	It is one of the oldest version management software. Although it works and is still used for some projects, it is better to use SVN (often presented as its successor), which fixes a number of its flaws, such as its inability to track renamed files.
SVN	centralized	Probably the most used tool at the moment. It is quite simple to use, although it requires a certain period of adaptation.
Git	distributed	Very powerful and recent, it was created by Linus Torvalds, who initiated Linux. It is distinguished by its speed and its management of branches, which allow parallel development of new functions.
Mercurial	distributed	More recent, it is complete and powerful. It appeared a few days after the beginning of the development of Git and is comparable to the latter in many aspects.
Bazaar	distributed	Another tool, complete and recent, like Mercurial. It is sponsored by Canonical, the company that publishes Ubuntu. It focuses on ease of use and flexibility.

Table 6: Main version control software.

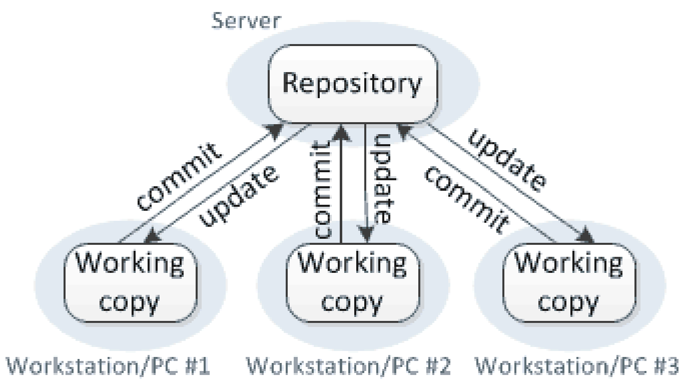


Figure 15: Centralized version control (Ernst, 2012).

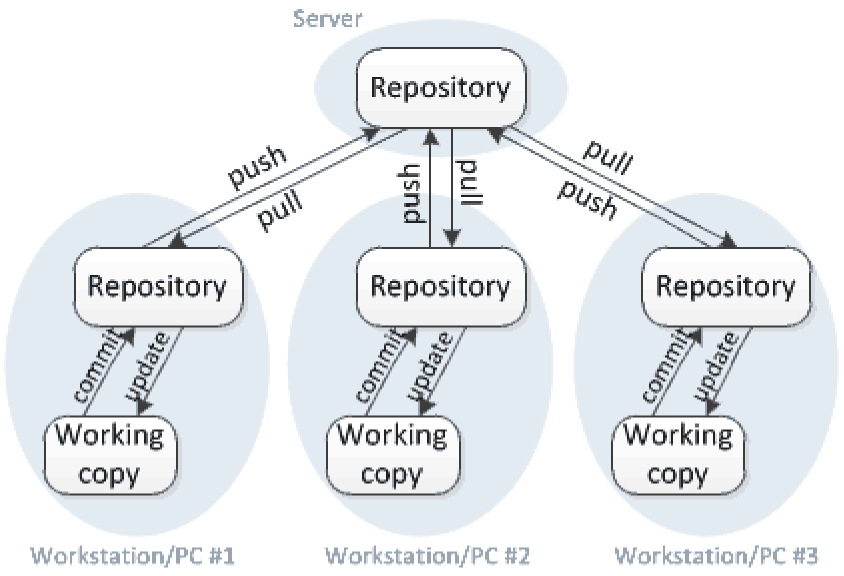


Figure 16: Distributed version control (Ernst, 2012).

In a single-user project, the most interesting feature of versioning is the ability to provide historical versions of the project. It is a guarantee against lost data because it will always be possible to retrieve either a previous version of a damaged file or previous estimation results. It is also suited to working with several computers, as may be the case when intensive computations are externalized on a server.

In the case of one user with one computer, the history is linear. However, when working with two computers or more, it is possible that changes are simultaneously made on the same file. The version control software is able to simultaneously take into account modifications made by different computers as long as no changes occur on the same lines. When conflicts arise, version control requests human intervention to resolve them.

To limit conflicts, it is possible to define *branches*. This implies the duplication of an object (such as a source code file or a directory tree), allowing the user to correct a program or add and test new features without affecting the initial version (parent branch). Manipulations on a branch are carried out without interfering with the parent branch. If these modifications are validated, they can then be integrated into the parent branch. Even a single developer can use this notion of branches and test new features on his program without risking the integrity of the exploited version of his software.

D 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)

/*****************************************************************************/
/* Exporting 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
```

E 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	`«chuncknawe»=`
						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 7: Source extension and code chunk syntax for literate programming tools.