StatWire: Visual Flow-based Statistical Programming

Krishna Subramanian
RWTH Aachen University
52056 Aachen, Germany
krishna@cs.rwth-aachen.de

Chat Wacharamanotham
University of Zürich
8050 Zürich, Switzerland
chat@ifi.uzh.ch

Johannes Maas
RWTH Aachen University
52056 Aachen, Germany
johannes.maas1@rwth-aachen.de

Simon Voelker
RWTH Aachen University
52056 Aachen, Germany
voelker@cs.rwth-aachen.de

Michael Ellers
RWTH Aachen University
52056 Aachen, Germany
michael.ellers@rwth-aachen.de

Jan Borchers
RWTH Aachen University
52056 Aachen, Germany
borchers@cs.rwth-aachen.de

CHI'18 Extended Abstracts, April 21–26, 2018, Montreal, QC, Canada.

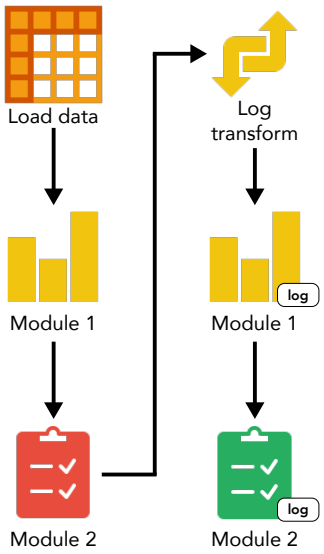
ACM ISBN 978-1-4503-5621-3/18/04.
https://doi.org/10.1145/3170427.3188528

Abstract
Statistical analysis is a frequent task in several research fields such as HCI, Psychology, and Medicine. Performing statistical analysis using traditional textual programming languages like R is considered to have several advantages over GUI applications like SPSS. However, our examination of 40 analysis scripts written using current IDEs for R shows that such scripts are hard to understand and maintain, limiting their replication. We present StatWire, an IDE for R that closely integrates the traditional text-based editor with a visual data flow editor to better support statistical programming. A preliminary evaluation with four R users indicates that this hybrid approach could result in statistical programming that is more understandable and efficient.

Author Keywords
Statistical Analysis IDE; Hybrid Programming; Visual Programming.

ACM Classification Keywords
H.5.2 [User Interfaces]: Graphical user interfaces (GUI); G.3. [Probability and Statistics]: Statistical software.

Introduction
Statistical analysis is an important task in several research fields. While analysis can be performed using text-based programming (e.g., R, Python) or GUIs (e.g., SPSS, JMP),



Module 1: compute descriptive statistics (mean, sd), and plot histograms.

Module 2: fit a model, get residuals, and perform Shapiro-Wilk's test on the residuals.

Figure 1: An example workflow of statistical analysis. Analysis is iterative and non-sequential with 'modules' acting upon modified data with each iteration.

the statistical community considers text-based programming to be a *"much more productive, accurate, and reproducible way of performing (analysis) tasks"* than GUIs [17].

However, our analysis shows that scripts written using current IDEs for R, the most commonly used text-based statistics programming language [13], are hard to understand and maintain. We speculate that current IDEs for R do not support the data-driven, iterative, and non-sequential nature of statistical analysis [10, 16], and we explore integrating the text-based environment in current IDEs with a visual data flow environment as a possible solution. As a realization of this concept, we present StatWire, an IDE for R that tightly integrates the two environments.

Motivation

Current IDEs for R

R [6] is the most commonly used statistical programming language in academia and research. Prominent IDEs for R include RStudio[1], the official R GUI, and Microsoft's Visual Studio Plugin for R. Such IDEs provide an interactive console for line-by-line execution of code snippets and script files for storing analysis code, which can then be executed in a sequential manner via the interactive console.

However, the workflow of statistical analysis is iterative and non-sequential [16, 10], as shown in Fig. 1. This could therefore be in conflict with the workflow supported by current IDEs for R that use a sequential representation to store and execute the analysis.

Evaluation of R Analysis Scripts

To investigate potential consequences of this mismatch, we examined analysis scripts written in R by researchers. 40 R analysis scripts, used in HCI and Psychology publications at top-tier conferences such as CHI '13 and ISWC '16, were collected from two sources: 32 scripts from 23 projects on the Open Science Framework (OSF) platform[2], and 8 scripts from 3 researchers at our university in Aachen. We randomly selected no more than 3 scripts from the same project or researcher to avoid biasing our analysis to a particular analyst. We analyzed a total of 20,303 source lines of code (SLOC), with an average of 508 SLOC per file, and found two major issues:

1. Excessive Code Cloning

Code cloning is a common programming practice in which the user copies, pastes, modifies, and executes code [8, 15]. Excessive code cloning can make analysis scripts difficult to maintain, understand and modify [9, 11, 12], limiting reproduction and reuse of the code for other analyses.

We followed a standard procedure [15] to identify instances of code cloning. First, we preprocessed the analysis scripts (remove blank lines, comments, etc.) to identify Lines of Interest (LOI). Then, we used fingerprints as both the intermediate representation of the code and the match detection technique to compute instances of code cloning. We identified that out of 4098 LOI from all scripts, 2861 LOI, or **70%**, were parametric clones [15].

2. Prevalence of Non-Modular Code Using modules is generally considered to lead to more productive programming [5]. Non-modular code is inefficient and does not match the nature of statistical analysis, since a typical analysis involves iterative execution of a module with modified input, as shown in Fig. 1. This issue is further aggravated in long analyses in which a module is reused several times.

In our analysis, we classified code snippets into the three

[1] http://www.rstudio.com

[2] http://osf.io

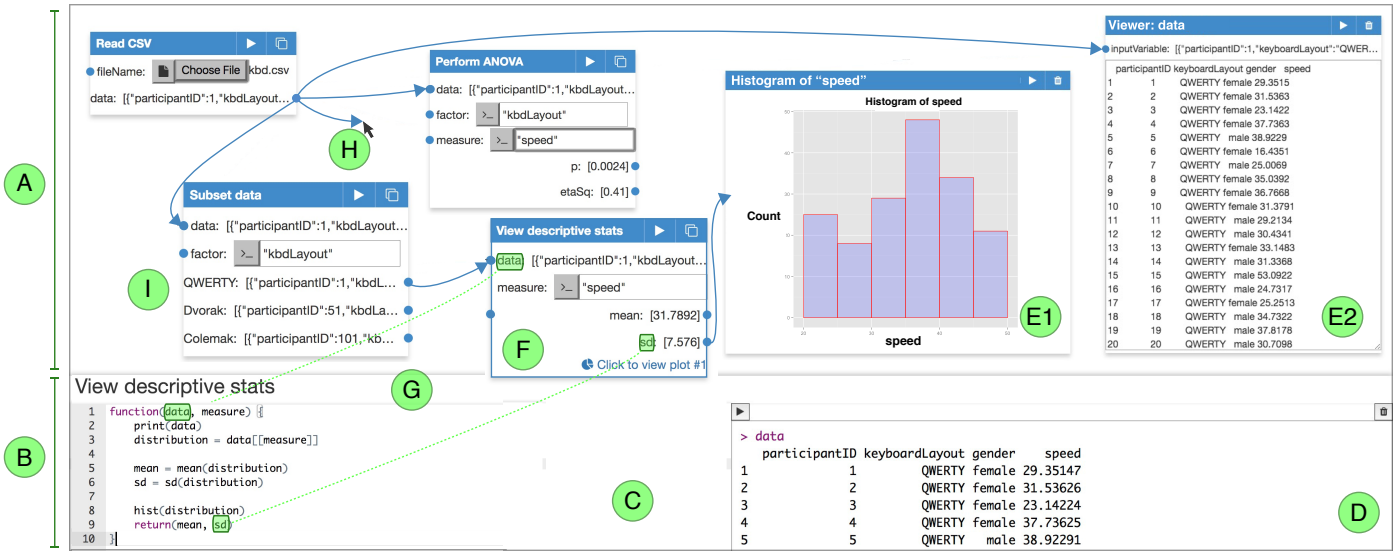


Figure 2: User interface of StatWire, a novel IDE for performing statistical analysis using the R programming language. StatWire tightly integrates a visual data flow editor (A) with a text-based editor (B) to better support code authoring.

major steps of statistical analysis: *preprocessing*, *exploratory analysis*, and *confirmatory analysis* [16]. We then identified how many of these steps were part of each script. We found that 97.5% of all scripts contained two or more steps, and 60% contained all three steps. This issue was also prevalent in long analysis scripts.

StatWire

Since existing IDEs for R result in issues with code reproduction and comprehension, we wanted to explore the tight integration of a visual data-flow editor with the traditional text editor as a possible solution. The following walkthrough explains the interaction design of our current prototype:

Hazel is an HCI researcher who wants to analyze data from a text entry study using StatWire. She is presented two views: a visual data flow editor (Fig. 2 A) and a textual programming environment (B). Each node in the visual editor can be either a *statlet*, which is a processing step (F), or a *viewlet*, which shows plots (E1) or data (E2). The edges (H) represent the flow of data across the nodes. The visual editor is initially empty, and Hazel creates a statlet by right-clicking on the empty canvas. When a statlet is created, the text editor (C) is shown with a default function template, encouraging her to think in a modular fashion and to write functions. During code authoring, Hazel uses the output shown next to the code (D) for debugging. She authors statlets to read in the CSV file and then subsets the data according to different levels of an independent variable.

When she adds an input or output argument to the function header, the corresponding node in the visual data flow editor is *updated automatically* (G). This lets Hazel focus on the flow of data (i.e., overview) in the visual editor, and on the processing code (i.e., details) in the text editor.

When Hazel wants to view the descriptive statistics of the QWERTY distribution using another statlet, she 'wires' its output to the 'View descriptive stats' statlet. She runs the statlet to view the histogram (in a viewlet, Fig. 2 E1) and descriptive statistics (Fig. 2 F). To repeat this for the other two distributions, Hazel simply 'rewires' the input of the statlet, leading to an efficient analysis workflow.

StatWire was developed in a user-centered, iterative fashion, during which we explored several design alternatives to support a tight integration between the visual and text-based environments. E.g., in an earlier prototype of StatWire, users added input or output arguments via the visual programming environment. However, evaluations with existing users of R and an expert in statistical analysis who teaches introductory courses in R revealed that users preferred to update the arguments in the textual programming environment. This resulted in live updates to the visual programming environment when changes were made to the textual programming environment.

Related Work

Visual programming environments have been shown to help with comprehension [2], programmer performance [1], and code navigation [3]. For a comprehensive overview of visual programming environments and their benefits, see [7]. StatWire uses a data flow visual programming environment, which better suits the data-driven nature of statistical analysis. Such data flow environments have been successfully applied to other data-driven domains [18].

Existing visual programming environments such as ViSta for statistical analysis [19] and Orange for machine learning [4] do not expose the underlying source code of their modules. Blender, a 3D modeling software, allows the user to view and edit the underlying source code of modules, but they have to initially be authored in a different environment and then imported. RapidMiner and KNIME are visual programming environments that allow authoring code using a text editor, but treat the visual editor as a passive element (i.e., the visual environment does not reflect changes in the textual code), resulting in a lack of integration between the two environments. The Gestalt system provides textual and visual environments, and allows the user to easily switch between them [14]. However, it uses a linear list as its visual representation, which does not suit the non-linear nature of statistical analysis. Further, Gestalt does not provide a tight integration between the two environments.

Preliminary Evaluation

To identify how well StatWire supports statistical programming with R, we compared it to RStudio, a widely used IDE for R, as well as RapidMiner and KNIME, two visual programming environments with R integration.

We recruited four R users from our local university in Aachen. Three had taken a graduate-level introductory analysis course in R, all had used RStudio before (none had used KNIME or RapidMiner before), and all had at least three years of experience performing statistical analysis.

Procedure

Each user performed analysis with each tool for a minimum of 30 minutes with different datasets and research questions. Additional research questions or datasets were provided as needed. We balanced the order of conditions and randomized both the dataset-tool pairing and the character-

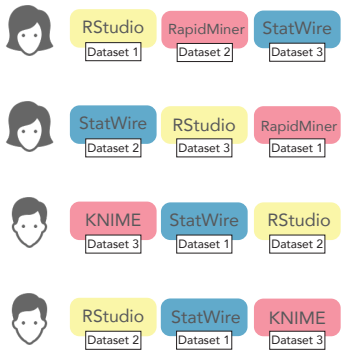


Figure 3: The experimental design of our preliminary study. We balanced the order of conditions and randomized data-tool pairing.

istics of datasets to minimize learning effects (Fig. 3. The analysis required typical, non-trivial analysis methods such as factor encoding, data transformation, and post-hoc tests.

Significant Findings

With RStudio, none of our users followed a modular programming approach. There were several instances of code cloning, and users used comments to structure the analysis. However, because users were familiar with RStudio, they reported feeling at ease using the tool.

With RapidMiner and KNIME, users had issues with the lack of integration between visual data flow editor and textual environment, which led to frustration. E.g., while authoring textual code, the visual data flow editor was not updated and did not provide any interaction. Of course, over time, users learn to work with these shortcomings.

With StatWire, users benefited from the tighter integration between the two programming environments: Two users reported understanding the analysis better because of the visual data flow editor, and during analysis with StatWire, 18 out of 35 statlets were reused, whereas no reuse was observed with other tools. Nevertheless, not all modules were reused, which indicates further work is necessary.

StatWire is open-source software, and available as a local web application from the StatWire project home page.[3]

Future Work

While our preliminary study is promising, it is limited by sample size and guiding hypotheses, and further longitudinal studies are required to understand StatWire's effects on code understanding, structuring, and navigation, as well as on the productivity of the analysis. Further, the artifact can be extended to allow transformation of existing analysis scripts into a more structured and reusable format by, e.g., automatically highlighting similar chunks of code and semi-automatically converting them to a reusable module.

[3] http://hci.rwth-aachen.de/statwire

REFERENCES

1. Ed Baroth and Chris Hartsough. 1995. Visual Object-Oriented Programming. Manning Publications Co., Greenwich, CT, USA, Chapter Visual Programming in the Real World, 21–42. http://dl.acm.org/citation.cfm?id=213388.213393
2. Andrew Bragdon, Robert Zeleznik, Steven P. Reiss, Suman Karumuri, William Cheung, Joshua Kaplan, Christopher Coleman, Ferdi Adeputra, and Joseph J. LaViola, Jr. 2010. Code Bubbles: A Working Set-based Interface for Code Understanding and Maintenance. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '10)*. ACM, New York, NY, USA, 2503–2512. DOI: http://dx.doi.org/10.1145/1753326.1753706
3. Robert DeLine, Mary Czerwinski, Brian Meyers, Gina Venolia, Steven Drucker, and George Robertson. 2006. Code Thumbnails: Using Spatial Memory to Navigate Source Code. In *Proceedings of the Visual Languages and Human-Centric Computing (VLHCC '06)*. IEEE Computer Society, Washington, DC, USA, 11–18. DOI: http://dx.doi.org/10.1109/VLHCC.2006.14
4. Janez Demšar, Blaž Zupan, Gregor Leban, and Tomaz Curk. 2004. Orange: From Experimental Machine Learning to Interactive Data Mining. In *Proceedings of the 8th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD '04)*. Springer-Verlag New York, Inc., New York, NY, USA, 537–539. http://dl.acm.org/citation.cfm?id=1053072.1053130

5. John Hughes. 1989. Why Functional Programming Matters. *Comput. J.* 32, 2 (1989), 98–107.

6. Ross Ihaka and Robert Gentleman. 1996. R: A Language for Data Analysis and Graphics. *Journal of Computational and Graphical Statistics* 5, 3 (1996), 299–314.

7. Wesley M. Johnston, J. R. Paul Hanna, and Richard J. Millar. 2004. Advances in Dataflow Programming Languages. *Comput. Surveys* 36, 1 (2004), 1–34. DOI: http://dx.doi.org/10.1145/1013208.1013209

8. Miryung Kim, L. Bergman, T. Lau, and D. Notkin. 2004. An Ethnographic Study of Copy and Paste Programming Practices in OOPL. In *Empirical Software Engineering, 2004. (ISESE '04)*. 83–92. DOI: http://dx.doi.org/10.1109/ISESE.2004.1334896

9. Rainer Koschke. 2008. Frontiers of Software Clone Management. In *Frontiers of Software Maintenance '08*. 119–128. DOI: http://dx.doi.org/10.1109/FOSM.2008.4659255

10. David Lubinsky and Daryl Pregibon. 1988. Data Analysis as Search. *Journal of Econometrics* 38, 1-2 (1988), 247–268. DOI: http://dx.doi.org/10.1016/0304-4076(88)90035-8

11. Jean Mayrand, Claude Leblanc, and Ettore M. Merlo. 1996. Experiment on the Automatic Detection of Function Clones in a Software System Using Metrics. In *1996 Proceedings of International Conference on Software Maintenance*. 244–253. DOI: http://dx.doi.org/10.1109/ICSM.1996.565012

12. Akito Monden, Daikai Nakae, Toshihiro Kamiya, Shin ichi Sato, and Ken ichi Matsumoto. 2002. Software Quality Analysis by Code Clones in Industrial Legacy Software. In *Proceedings Eighth IEEE Symposium on Software Metrics*. 87–94. DOI: http://dx.doi.org/10.1109/METRIC.2002.1011328

13. Robert A. Muenchen. 2017. The Popularity of Data Analysis Software. (2017). (Accessed: 08-31-2017) http://r4stats.com/articles/popularity/.

14. Kayur Patel, Naomi Bancroft, Steven M. Drucker, James Fogarty, Andrew J. Ko, and James A. Landay. 2010. Gestalt: Integrated Support for Implementation and Analysis in Machine Learning.. In *Proceedings of the 23rd Annual ACM Symposium on User Interface Software and Technology (UIST '10)*. ACM, New York, NY, USA, 37–46. DOI: http://dx.doi.org/10.1145/1866029.1866038

15. Dhavleesh Rattan, Rajesh Bhatia, and Maninder Singh. 2013. Software Clone Detection: A Systematic Review. *Information and Software Technology* 55, 7 (July 2013), 1165–1199.

16. John W. Tukey. 1977. *Exploratory Data Analysis*. Addison-Wesley Publishing Company. https://books.google.de/books?id=UT9dAAAAIAAJ

17. Pedro M. Valero-Mora and Rubén D. Ledesma. 2012. Graphical User Interfaces for R. *Journal of Statistical Software* 49, 1 (2012), 1–8.

18. Kanit Wongsuphasawat, Daniel Smilkov, James Wexler, Jimbo Wilson, Dandelion Mané, Doug Fritz, Dilip Krishnan, Fernanda B Viégas, and Martin Wattenberg. 2017. Visualizing Dataflow Graphs of Deep Learning Models in TensorFlow. *IEEE Transactions on Visualization and Computer Graphics* (2017).

19. Forrest W. Young and Carla M. Bann. 1996. ViSta: The Visual Statistics System. (1996).