Paper 876-2017

Streamline Your Workflow: Integrating SAS, $\LaTeX$, and R into a Single Reproducible Document

Lucy D'Agostino McGowan, Vanderbilt University

ABSTRACT

There is an industry-wide push toward making workflows seamless and reproducible. Incorporating reproducibility into the workflow has many benefits; among them are increased transparency, time saving, and accuracy. We walk through how to seamlessly integrate SAS®, $\LaTeX$, and R into a single reproducible document. We also discuss best practices for general principles such as literate programming and version control.

INTRODUCTION

This paper will give you some tips and tools for streamlining your workflow. We will be creating a single document that can be compiled into a beautiful report, integrating SAS, $\LaTeX$, and R. Figure 1 displays a general summary of the streamlined workflow we are creating.

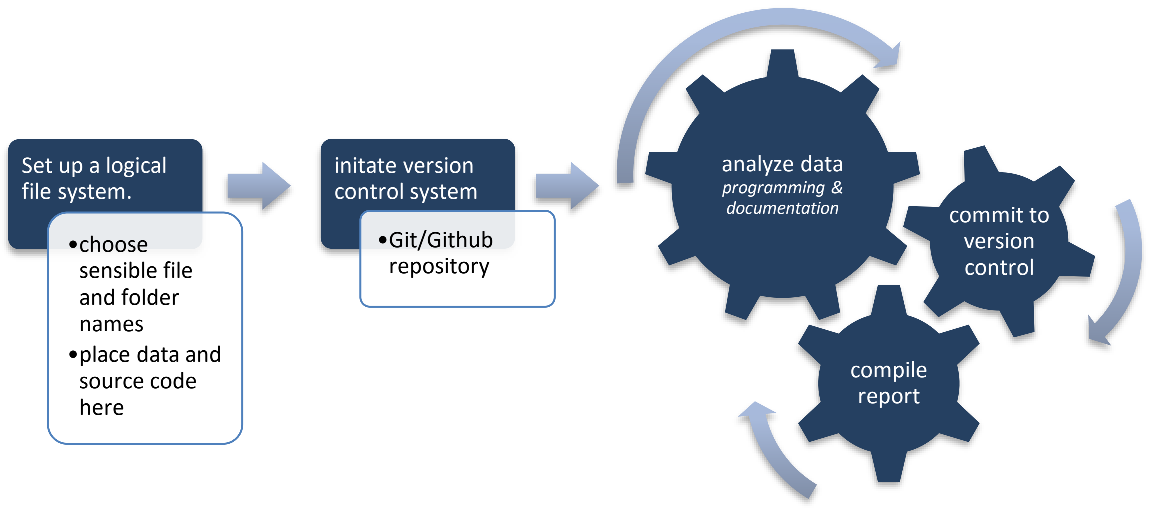

Set up a logical file system.
•choose sensible file and folder names
•place data and source code here
initate version control system
•Git/Github repository
analyze data
programming & documentation
commit to version control
compile report

Figure 1. Streamlined workflow summary

Before we dive in, here are some definitions of words I will be using.

`shell`	This can be referred to as the terminal, bash, Command Prompt, etc. We will be navigating to different folders and running commands here. To change folders, use the `cd` command. For example: `cd ~/documents` will take me to the documents folder.
`chunk`	A section of a programming language, in this case SAS or R.
Git	A version control system for tracking file changes
Github	Web based Git repository

LaTeX	A type-setting system designed for scientific documentation. *Pronounced lah-tek*
SAS University Edition	A free web interface to many SAS products (more here)

Table 1. Definitions

This paper will be a bit of a "choose your own adventure". The first section will give some tips for best practices and lay out the principles of literate programming and version control. If that doesn't interest you, you can skip straight to the Setup section. This will go through some setup and installations for the tools you will need to complete the task at hand. If you already have everything installed and would like to dive right into the step-by-step process, you can skip to the Methods section.

BEST PRACTICES

There are many principles to consider when creating a streamlined workflow. We want our workflow to (1) be easy to implement, (2) minimize errors, (3) be easy for collaborators to understand and contribute to. Here we will discuss three ways to address these three considerations, file organization, version control, and literate programming.

FILE ORGANIZATION

Consistent file organization is key. I prefer the following setup:

🗁 project-name

↳ 🗁 code

↳ 🗁 data

↳ 🗁 reports

Figure 2. File organization

The **data** folder will contain any data I need for my project. The **code** folder will contain my analysis code. Ideally, there will be a single reproducible document here that I can run to generate a report, saved in the **reports** folder. This single reproducible document will be the primary focus of this paper. This is the key to the three considerations for the implementation of a streamlined workflow. Only having to edit a single document adds to ease of implementation, minimizes errors, and is very useful for collaborators, even if that collaborator is just future you (Wickham 2017).

If you are following along, go ahead and create this folder tree on your local computer.

An additional consideration is your file naming convention. This is a personal preference, but consistency will make you and your collaborators much happier. Some principles I like to stick to:

- All lower case
- Separated by underscores (_) for field separation and dashes (-) for term/word separation (for example if I wanted to save a document with a date and an explanation, I'd separate the date and name with an underscore, and within each field I'd separate with dashes: 2017-01-01_new-years-resolutions).
- If including a date, begin file name with YYYY-MM-DD (this ensures easy sorting)
- Avoid things like analysis_final_final_this-is-really-final_2.tex (this brings us to our next section, Version Control!)

VERSION CONTROL

Version control, a management system of file changes, is paramount for a streamlined workflow. Good version control systems avoid messy file organization, such as analysis_final_final_this-is-really-final_2.tex. Instead, you can rely on the version control system for keeping track of the file version, allowing the file to remain named something sensible, such as analysis.tex. For this paper, we will use

GitHub. GitHub is a free online file hosting system for the Git version control system. You can find this paper's examples on my GitHub account here.

GitHub

This will be a very brief introduction to Git and GitHub. If you are interested in learning more, there are numerous resources (Philp 2012, Hu 2013, Bryan 2017).

0. If you do not have a GitHub account, you can create one here. If you do not have Git installed, install it.
1. Log into your GitHub account and create a repository with the same name as the parent directory you created above (from Figure 2, **project-name**).
2. In the `shell`, navigate to your **project-name** folder and type (or copy and paste) the following, changing **your-user-name** to your GitHub username and **project-name** to your project name

```
git init
git add .
git commit -m "first commit ⍰"
git remote add origin https://github.com/your-user-name/project-name.git
git push -u origin master
```

Code 1. Shell commands to initialize a GitHub repository

Every time you make a change to a file, repeat the following commands in the `shell`, changing the bold text as appropriate.

```
git add filename
git commit -m "describe your file change here"
git push -u origin master
```

Code 2. Shell commands to add, commit, and push code to GitHub

LITERATE PROGRAMMING

Literate programming is a concept conceived and propagated by Donald Knuth, the creator of TeX. Knuth states,

> "I believe that the time is ripe for significantly better documentation of programs, and that we can best achieve this by considering programs to be *works of literature*. Hence, my title: "Literate Programming."
>
> Let us change our traditional attitude to the construction of programs: Instead of imagining that our main task is to instruct a *computer* what to do, let us concentrate rather on explaining to *human beings* what we want a computer to do." (Knuth 1984).

Essentially, the concept entails writing programs with more than just short comments, but full explanations of why and how this program should be used. It is the intertwining of some markup language that is easier on the human eye, and a programming language organized in `chunks` of code.

We can achieve this by incorporating LaTeX and the StatRep package.

SETUP

This paper requires the installation of LaTeX, StatRep, SAS University Edition, and R. If you already have these installed, you can jump to the Methods section.

LaTeX

There are many ways to install LaTeX (a list of free implementations can be found here). The most important pieces that you need for this tutorial are the following LaTeX packages:

1. pdflatex (version 1.3 or later)
2. StatRep

Most LaTeX downloads will automatically include pdflatex. You can install the StatRep package by downloading statrep.zip from here (Arnold & Kuhfeld 2012). Be sure to follow the instruction in the README file. As a quick summary:

1. Copy the StatRep macros to a local directory. *Remember where you are saving these macros, you will need this **filepath** soon.*
2. Install the LaTeX package

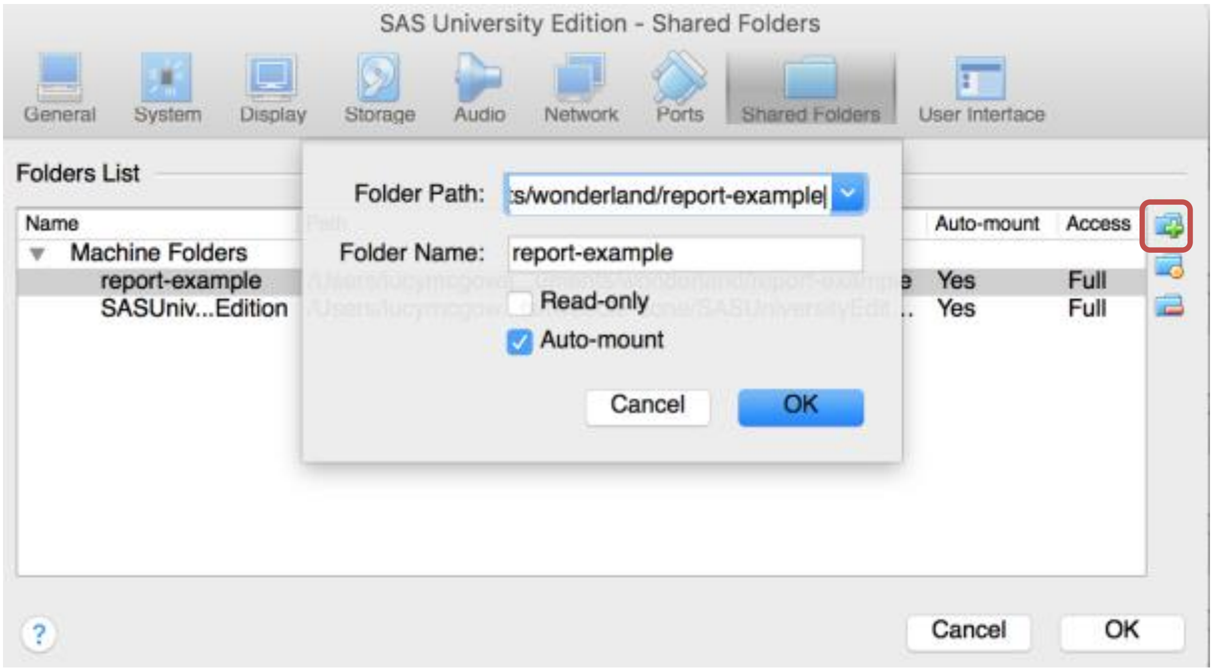
SAS UNIVERSITY EDITION

SAS University Edition is a free web interface to many SAS products. Since it is a web interface, it is accessible for both Mac and PC users. Essentially, you run a virtual machine on your local computer and create a shared folder that SAS can access. Step-by-step instructions for downloading and implementing SAS University Edition can be found here.

Once SAS University Edition has been installed, we need to setup a shared folder to save our output. As discussed in the Best Practices section, a good setup can be something like the following:

1. Create a folder with the title of your project, mine will be called *report-example*
2. Within that folder create 3 folders: **data**, **code**, **reports**.

We can make *report-example* a shared folder for SAS University Edition to access. Open your virtual machine and click on *Shared Folders.* Then click the icon of a document folder icon with a (+) to add a shared folder (in Figure 3 outlined in a red box). Fill in the path of your project folder and click *Auto-mount.* Notice here my *Folder Name* is the same as the name on my local computer. SAS University Edition will now recognize will know this folder by as /folders/myshortcuts/report-example. (Arnold 2015).



Folder Path: s/wonderland/report-example
Folder Name: report-example
Read-only
[✓ Auto-mount]

Name	Path	Auto-mount	Access
Machine Folders			
report-example	[illegible]	Yes	Full
SASUniv...Edition	[illegible]	Yes	Full

Figure 3. Add a Shared Folder to SAS University Edition

We also need SAS University to recognize your macros folder (*remember in step 1 of the StatRep setup (above) when you remembered where you saved those macros? We're going to use the location again here*.) Create another shared folder by clicking the document folder icon with a (+) to add a shared folder (in Figure 3 outlined in a red box). Fill in the path of your macros folder and click *Auto-mount.*

R

If you are interested in incorporating R code along with SAS, you will need to download R (here), as well as the kntir package. The knitr package can be installed by opening R and running the following code:

```
install.packages('knitr')
```

METHODS

There are three main steps to making this process seamless.

1. **Document Setup.** This involves setting up a reproducible document.
2. **Programming and Documentation.** This step entails writing code in the programming languages and writing text for humans, insuring we are not actively participating in *illiterate* programming.
3. **Rendering.** Here we will run all of the code and render the final document.

We will complete step 1 once per project and repeat steps 2 and 3 many times as part of our analysis workflow.

DOCUMENT SETUP

Now that we have the proper software installed, we need to setup our document. Open your favorite text editor and paste the following in, updating the bold text as indicated below:

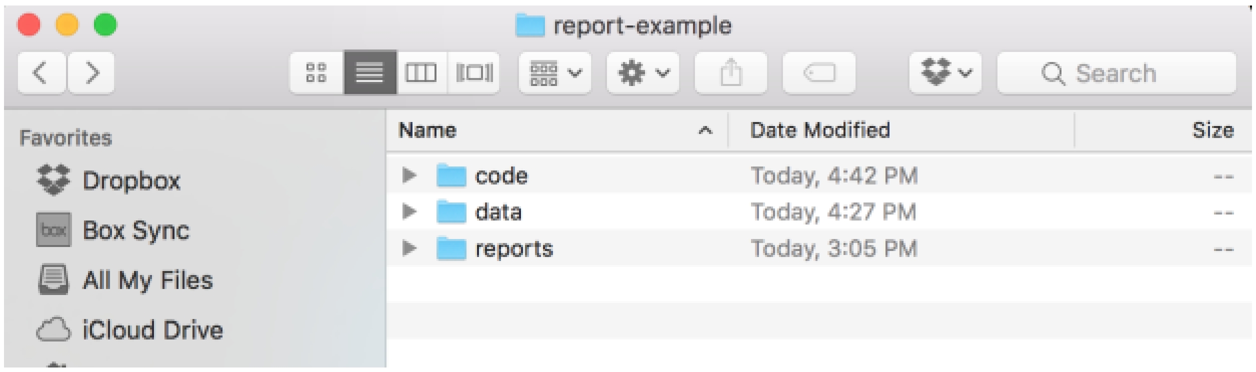
```
\documentclass{article}
\usepackage{statrep}
\usepackage{parskip,xspace,hyperref}
\def\SRrootdir{/folders/myshortcuts/report-example/code}
\def\SRmacropath{/folders/myshortcuts/sas-macros/statrep_macros.sas}
\title{Your title}
\author{Your name}
\date{\today}
\begin{document}
\maketitle

[we will add code here]

\end{document}
```

Code 3. Text for initiating analysis.Rnw

- Update **report-example/code** to the name of the folder where you are keeping this document. I recommend keeping code in a separate folder. If you set up your project as mentioned in the File organization section, this will be your **project-name/code**.
- Update **sas-macros** to the name of the folder where you placed your SAS macro (remember from above).
- Update **Your title** and **Your name.**

If you are planning on including R code, save this document as **analysis.Rnw** (if you will only be incorporating SAS code, you can save this as **analysis.tex**) locally in your **code** folder. Notice in the document we just saved, we are telling SAS University Edition where these folders are on the virtual machine rather than where they are locally.

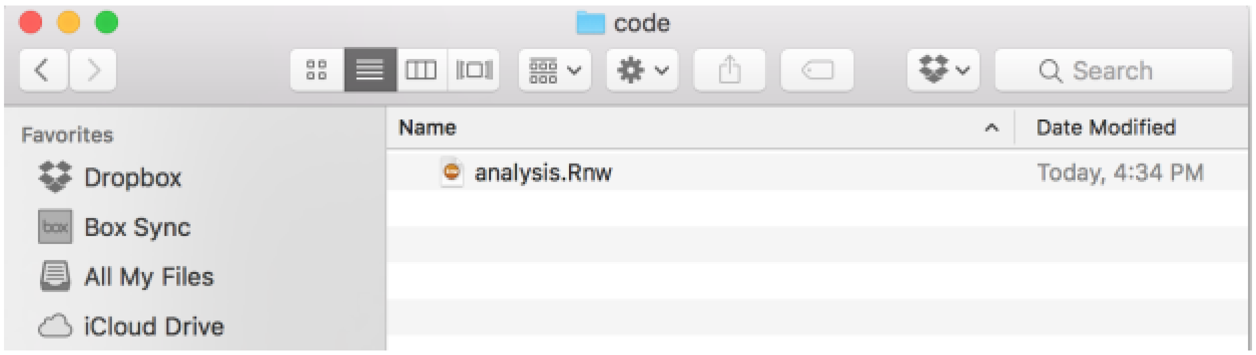
If you are all set, move on in your adventure to the Programming and Documentation section. Otherwise here is a quick recap. We have a local folder with our **project-name**, mine is called **report-example** (Figure 4).



Name	Date Modified	Size
code	Today, 4:42 PM	--
data	Today, 4:27 PM	--
reports	Today, 3:05 PM	--

Figure 4. File setup demonstration: project-name

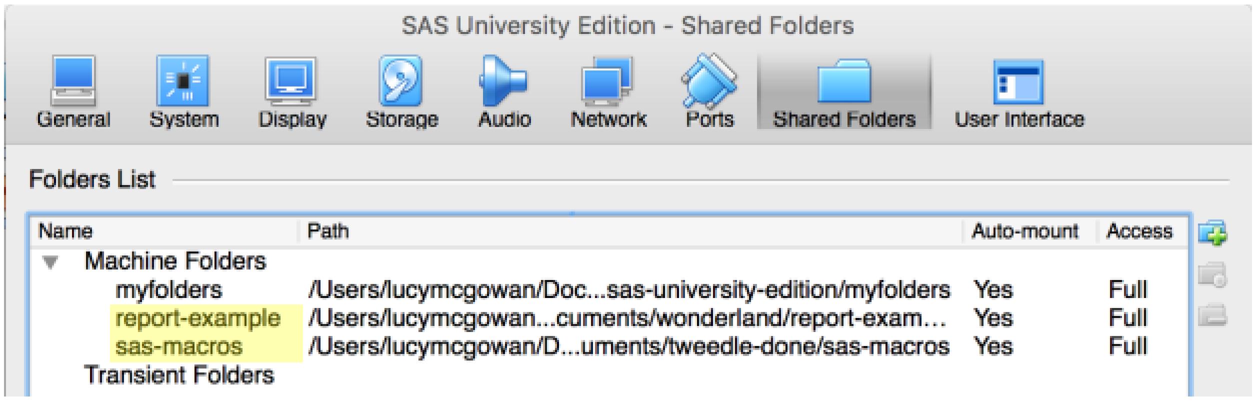
Within this folder, there are 3 folders: code, data, and reports. We have just created a new file called **analysis.Rnw** that includes the updated text from Code 3 (Figure 5).



Name	Date Modified
analysis.Rnw	Today, 4:34 PM

Figure 5. File setup demonstration: code folder

SAS University Edition is aware of the location of these files because we set up a shared folder for our **project-name** folder (Figure 4) and our **sas-macro** folder (remember from above) – the Shared folders should resemble Figure 6.



Name	Path	Auto-mount	Access
Machine Folders			
myfolders	/Users/lucymcgowan/Doc...sas-university-edition/myfolders	Yes	Full
report-example	/Users/lucymcgowan...cuments/wonderland/report-exam…	Yes	Full
sas-macros	/Users/lucymcgowan/D...uments/tweedle-done/sas-macros	Yes	Full
Transient Folders			

Figure 6. File setup demonstration: SAS University Edition shared folders

PROGRAMMING AND DOCUMENTATION

Add the data that you will be analyzing to the **data** folder. In my case, I am adding a .csv file from a 538 Star Wars Survey (found here, FiveThirtyEight 2014).

Open **analysis.Rnw** in your favorite text editor. Delete "[we will add code here]" – this is where we are going to add code as well as detailed explanations of our analyses.

As discussed in the

Literate **Programming** section, the programming languages are separated into `chunks` of code. SAS and R `chunks` look different. We wrap our SAS code with the following:

```
\begin{Sascode}

/* SAS CODE HERE */

\end{Sascode}
```

Code 4. SAS code chunk

We wrap our R code with the following:

```
<<>>=

#R CODE HERE

@
```

Code 5. R code chunk

For example, I want to read my .csv into SAS, so I will add the following to my **analysis.Rnw** file (Code 6).

```
I obtained data from a 538 Star Wars Survey (found here:
\url{https://github.com/fivethirtyeight/data/tree/master/star-wars-survey}
and will read it into SAS in order to analyze whether age, gender,
education level, or whether the individual is a Star Trek fan are
associated with incorrectly believing that Greedo shot first.

\begin{Sascode}
libname data "/folders/myshortcuts/report-example/data";

filename reffile '/folders/myshortcuts/report-example/data/star-wars-
survey-538.csv';

proc import datafile=reffile
    dbms=csv
    out=data.starwars;
    getnames=yes;
run;
\end{Sascode}
```

Code 6. Import file example. Update the libname and filename to match your file locations. Notice how this example begins with me describing where I obtained the data and what I will be doing with it (literate programming) as well as a chunk of SAS code wrapped with the \begin{Sascode} and \end{Sascode} markers.

Notice in Code 6 I included some descriptive text above the SAS code `chunk`. For conservation of space, I will not show this step every time, but be sure to include it – this is one of the aspects that makes these reproducible documents so appealing.

We can then add some data cleaning and analysis steps (Code 7).

```
\begin{Sascode}[store = logistic]
  ods graphics on;

  proc logistic data = data plots = oddsratio;
   class age (ref = FIRST) gender college star_trek_fan;
   model wrong (event = "1") = age gender college star_trek_fan;
  run;

  ods graphics off;
  \end{Sascode}

  \Listing[store = logistic,
      objects = OddsRatios,
      caption = {Wrong about who shot first odds ratios}]{logisticOR}

  \Graphic[store=logistic,
     objects=ORPlot,
     caption={Wrong about who shot first plots}]{ORplot}
```

Code 7. Analysis example. Notice here we use the \Listing and \Graphics commands to output tables and graphs from our analyses.

After completing our SAS analyses, we want to export our cleaned dataset so we can utilize it in R.

```
  \begin{Sascode}
  proc export data=data.starwars
     outfile =
       '/folders/myshortcuts/report-example/data/starwars_sasedit.csv'
     replace
     dbms = dlm;
     delimiter = ',';
  run;
  \end{Sascode}
```

Code 8. Export example. We export our cleaned data for analysis in R. Update the outfile to match where you would like to save your file in your project location.

For more tips and tricks for using StatRep, the manual is a great resource (found here).

Now we want to add some R code. We want to be sure this code will only run *after* we have exported our SAS data, so we have a small logical expression to ensure that the edited .csv file exists (Code 9).

```
  <<>>=
  filename = "../data/starwars_sasedit.csv"
  if (file.exists(filename)){

  starwars <- read.csv(filename)

  # more R code here
  }
  @
```

Code 9. R code example

RENDERING

Now that we have our **analysis.Rnw** document written up, complete with documentation, SAS code, and R code, let's render it. There are 4 steps to render this code:

1. **Compile the .Rnw file.** This is completed using knitr. In the `shell`, navigate to your **code** folder where **analysis.Rnw** is located and submit:

```
RScript -e "library(knitr); knit('analysis.Rnw')"
```

2. **Compile the pdflatex.** Run the following in the `shell`:

```
pdflatex analysis.tex
```

3. **Run the SAS code.** Open SAS University Edition. Navigate to your **code** folder (it will be under *Server Files and Folders*). Open and run **analysis_SR.sas.**

4. **Re-compile the pdflatex.** Run the following in the `shell`:

```
pdflatex analysis.tex
```

As you edit the code, you will repeat the above steps many times. The **only** document that you should be editing is the **analysis.Rnw** file. This ensures a streamlined workflow and safeguards against mistakes. If you are only including SAS code, you only need to complete steps 2-4.

Once you have compiled your document, you can set up an alias in the **reports** folder so you will have a clean .pdf to show to collaborators. To do so, run the following in the `shell`, updating the path to your local file path.

```
ln -s  ~/your-filepath/report-example/code/analysis.pdf
../reports/analysis.pdf
```

CONCLUSION

This paper walks through how to seamlessly integrate SAS®, $\LaTeX$, and R into a single reproducible document. We also discuss best practices for file organization, version control, and literate programming, demonstrating a workflow that is easy to implement, minimizes errors, and is easy for collaborators to understand and contribute to.

REFERENCES

Arnold, T., & Kuhfeld, W. F. (2012). Using SAS and LATEX to Create Documents with Reproducible Results. URL http://support.sas.com/resources/papers/proceedings12/324-2012.pdf.

Arnold, T. (2015). The StatRep System for Reproducible Research: A Note for SAS University Edition Users. URL http://support.sas.com/rnd/app/papers/statrep/statrepUE.pdf.

Bryan, J. (2017). Happy Git and GitHub for the useR. URL http://happygitwithr.com/

FiveThirtyEight. (2014). America's Favorite 'Star Wars' Movies (And Least Favorite Characters). URL https://fivethirtyeight.com/datalab/americas-favorite-star-wars-movies-and-least-favorite-characters/. Data found at: https://github.com/fivethirtyeight/data/tree/master/star-wars-survey

Hu, J. (2013). The Hitchhiker's Guide to Github: SAS Programming Goes Social. SESUG. URL http://analytics.ncsu.edu/sesug/2013/PA-04.pdf.

Knuth, D. E. (1984). Literate programming. The Computer Journal, 27(2), 97-111.

Philp, S. (2012). An Introduction to Git Version Control for SAS Programmers. WUSS. URL http://www.wuss.org/proceedings12/94.pdf.

Wickham, H. (2015). R packages. O'Reilly Media, Inc.

RECOMMENDED READING

- *The StatRep System for Reproducible Research*

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Lucy D’Agostino McGowan
Vanderbilt University
✉ ld.mcgowan@vanderbilt.edu
www.lucymcgowan.com
@LucyStats