Phil on Learning R: The Training Sequence

How to approach your early learning phase, orient toward long-run skills and understanding, and harness programming in R beyond mere commands.

Written especially for my psych grad student buddies coming from SPSS, those who are not fully trained programmers but want to know how to build up skills and why I obsess about elegant code.

Phil Enock
Student
Department of Psychology
Harvard University
Email address: p.enock on the domain of google's email system

Version 0.2, July 2014

Table of Contents

Intro

I was gonna write a bit of advice in an email, and then I remembered I made a Word template last year for writing docs in a cool format. And then I wrote 17 pages. Weird. I'd love to hear back from anyone about whether any of this guide is helpful and which parts specifically have been useful to you or are rubbish.

This material is going to be shareable with anyone, but for now, please ask me before you pass it on, in case I want to update it before you send it.

Anyway, I made this because:

1. I was going to share reference materials, and some of my most useful functions, and then I realized I couldn't resist sharing advice. I do that a lot, give the world according to Phil, on many topics. And I mean, it's conceivable (small to medium probability) people might want to hear a bit of this, but definitely not a 6000-word version. Fortunately for you, you can just choose to read or not read this, and when talking to people about R, I will henceforth *try* to just say "see guide" so I don't take up their time.

2. I don't think the R beginner's guides out there or workshops do enough… they tend to say, "look, R is easy, just model <- lm(y ~ x1 + x2, data=data) and your regression's done yay!" and I'm like yeah okay, thanks so much so now I know data analysis sure is easy and not a complex art that needs our deep respect attention and devotion as well as the humility to recognize how little we have just learned from that command which I could have googled, thank goodness they can next tell me how to change more parameters of regression instead of stopping to think about the big picture and talk about *how to get on track* for a persistent fruitful process learning a complex, rewarding skill fundamental to our research lives. (I do appreciate that they're giving these workshops… sorry for being militant on this, I'm just particular about didactics I guess.)

Anyway, a lot of the stuff in this guide will *not* be readily understandable unless you're experienced in R. Yet it's for beginners? Crazy! I guess it's for beginners or intermediates. It's for you go-getter self-learners. Here's how:

Read this guide, and for all the places you don't understand something, that's why you pull out the other guides—they're the real meat of learning. **I'm just trying to talk *process* more than *content,* since the other guides will give you content.**

The PH ("Phil's Helpful") functions

Mainly you should just read through the comments in the code, where I explain each in this very small set of utilities. I'm trying to prepare more functions to be ready to share, but I'm not done revising them and making them more logical, easy to understand, and elegant.

There are 3 types of functions in here:

1. Convenience shortcut functions that I use all over the place, mainly to save typing time, and it also makes your code clearer when you "omit needless words" (which happens to be Rich's favorite advice)

2. Output functions to send data frames[1] to Excel and SPSS. I use the Excel one all the time to look at my data -- it's the way I browse my data whenever I just want to see things.

3. The sandbox() function. Makes a data frame and list to play with. I always bring it up when I'm learning how stuff works involving those, trying new things out. Go read and run sandbox() now or when you reach the parts of this guide below where I talk about it and start writing "sdf$" a lot.

I've only run them on Windows PCs. Please, someone try them on Mac and let me know if they work.

Also, **always check your results.** Pretty sure my code does work but um…

1. Disclaimer—I can't be responsible for whatever happens
2. Don't trust me, I make mistakes
3. I've never tested on *your* data

If you're exporting to SPSS for example, check every single variable and how it comes through in SPSS. Check missing data handling, and nominal vs. numeric, dates, that kind of thing. If you see something in my code that looks off and you're like "is that a mistake in his code?" I'd appreciate if you check with me so we can clear it up. Also tell me if something's wrong in this guide.

[1] Quick term definition in case you're totally new: a "data frame" in R is the term for what you'd call a matrix in MatLab or a spreadsheet in Excel, or some might call a table.

Guides / resources for R

Guide	Role	How to use
Lam Introduction to R (google it)	I love this because it has a lot about how to use variables and manipulate your data.	Read the whole thing and test out some of what it's teaching you as you read.
Verzani SimpleR (google it)	Covers how to do the type of stuff we learned in stats class.	Consult when you want to run stats, for the basic commands.
R Cookbook ($17 in kindle format, amazon.com/Cookbook-OReilly-Cookbooks-Paul-Teetor/dp/0596809158)	A whole book... tons of info. Some people might have this book in PDF format. Anyway, it's easy to buy the Kindle version from Amazon and have it on your computer.	If you want to know how to do something, you might look here first before trying Google… certainly this is way better than R's documentation. Digital format is better than paper book because you can search in it.
Andy Field's Discovering Statistics Using R (priceless, amazon.com/Discovering-Statistics-Using-Andy-Field/dp/1446200469)	is.god(Andy) == TRUE So, he covers all the stats in the best possible balance of theory, practice, and personality. (He doesn't cover what I focus on below though, like making code reusable and structuring your code.)	While you're deciding if R is right for you, you can wait to invest in this. If you decide you really want to use R for and are working on a paper, best to buy Andy's book. Consult a given chapter when you need solid understanding as well as commands for whatever area of stats. The multilevel linear models chapter helped me, and I saw it cited in a psych paper btw.
R help	Quickest way to look up a function	In RStudio, use the keyboard shortcut… if you just typed the aforementioned "lm" in your code, you can be on that word and hit F1, and the help appears within RStudio.
Google	Quickest way to look up anything in life	A glimpse of Phil's shortcut world: Open your browser, hit Ctrl-L to get to the address bar, and put in your search. You may see 5 results that are potentially useful. Ctrl-click each one to open tabs in background (works in Chrome and others), and they'll be loading. Then you can Ctrl-Tab and Ctrl-W through your tabs.
Rseek.org	An R-focused version of Google	Try it... I usually just use Google but thought this was worth mentioning.

R is good

Why R is powerful

A real programming language

Some people say R is "powerful" because you can download any stats package into it and run fancy modeling, etc. Totally true. But I say it's powerful because it's a real programming language.

R is a real programming language, as are MatLab and Python. SPSS is not. I believe SAS is not, and Stata used to not be but introduced proper programming in 2008, and also Nate Silver uses Stata so I have to respect it. When I say "not," it's because they merely have syntax commands. When I started learning SPSS syntax in depth, I started trying to do my usual programming jumping around, ducking and weaving, and I slammed up against the enclosing walls of SPSS of which I became suddenly aware and so bashed my head and limbs

against it and had to find something better. You can't properly do basics like variables, functions, and loops, so all of those are dealbreakers. That said, if you feel like variables, functions, and loops will always disagree with you, **it would not be a crazy choice to stick with SPSS** even though I'm an R fan and how could you possibly have different preferences from me. Some people also use RCommander, kind of a way of going halfway, as it has graphical dialogs like SPSS. Andy Field is the right guide for using that.

Anyway, so going from SPSS to R increases infinitely the amount of good structure, automation, flexibility, and reusability you can get with your code. Infinitely. It's like trying to become a sketch artist with a big fat marker, and then R gives you fine-tipped implements (pencils, pens), and you have to work more to fill the page but there's no limit to how good you can get. (You still might want the marker [SPSS] for certain works, too, but you don't want to be limited by its fat face.)

You can do anything. Last week, to get a handle on 400 JPG files for face stimuli all named IMG_0235.JPG and so on, I wrote a script to rename the files to use info about them (emotion, sex, age) from a data table included with the set. R can create, edit, delete files, use the internet, all sorts of stuff. My SPSS export function (included) is in some ways the most badass function I've written (and op() is the most useful).

Statisticians use R

Seriously, they all use it. They also look at psych researchers all the time and say, "Oh my goodness these people know nothing about stats." So thus I live in fear with plenty of imposter syndrome, but anyway might as well try to do my best and use what they use.

They use R in actuarial science, a domain where forecasting is probably good because they're accountable for the probabilities that play out in a reality that actually happens (e.g., people getting sick out of insurance company's 10 million patients). Unlike in many fields, including psych, where with all the replication problems—wait, you signed up for my R rants not my science rants?? Okay phew you don't have to listen to all that.

Still working a bit with other stats programs too

Well, folks in most realms of the MBA/business world typically work only in Excel, which suggests limitations to that entire field. That's my little jab at them because I have an attitude problem.

Excel is still great for being able to move data around while looking at it in front of you. Also using color and space to present things for one-time use without working hard to set it up. Also for data entry. And Nate Silver uses Excel for presenting data so I would never knock it. I use my op() function all the time to send data out to Excel to look at it, maybe hide some columns as I go… so I find myself in R 90% of the time; Excel very frequently but just for a few seconds at a time looking at my data, benefitting from the keyboard shortcuts there and ability to nicely display data frames with a lot of columns and text; and SPSS 5% of the time, to get comprehensive ANOVA outputs I guess was the main thing for me and also sometimes double-check their output vs. R as a safety check.

No limits

If you use SPSS, you can reach many avenues where you just can't do some analyses. People will tell you, well SPSS just can't do that, and they'll be right. Whereas if anyone says "I'm not sure you can do that analysis in R," hmm, well, that doesn't really happen, and if it does, it's more like you can't do it *their way* in R.

Or in SPSS land, you might reach a point where you're like, "Okay I have this analysis process, but it always involves running these many steps… I can't make it faster and I always have to cut and paste. And I pray I never need to go back to the first data processing step because I don't remember how I got here exactly through all the

steps and different SAV files I went through along the way." I did that before, lots. This doesn't really happen with R if you get on the right track with it. Take the time to make the right structure, in logical steps, functions, and code files, save and organize your code, and keep it traceable.

Reflections on my R adventures

So, I'm not sure if I expended more time than I gained in my first 6-12 months of R. But I know that if you, like me, plan to learn and grow in data analysis over a period of 6-12 years… well, with SPSS or any non-programming language, you'd hit so many limits you have to work around. (I mean if you only want to do data stuff for 5 years in grad school, probably stick with SPSS unless you're already a programmer.)

With R, or another programming language, dedicating yourself to improving your code and skills, you'll continually work out the kinks in the way you operate, learn how to reuse more of your code across new studies, and never have to worry that you're not on the best stats platform.

Your choice on how far to go, how soon

You could also analyze data in R without using the programming stuff I'm advocating. I have a feeling that many or most R users do this, though actually I don't know, because not many of us in my neighborhood are really using R so I haven't seen other people's data analysis R code.

Maybe it does make sense to just do that at first and ignore this guide. **Maybe I'm trying to get you to learn too much too fast, trying to do everything at once when a more gradual approach is wiser. This is very possible, as I have this problem myself all the time.** But I say, try to level up your skills including functions and programming stuff as you go, and you'll progress to agility, deep-level understanding, and a long-run upward skill curve instead of plateau.

Pain in the Rs: How R is bad

Bad help files

Ugh, so hard to read them. Anyway, to mitigate, try to use the guides I attached, and use trial and error yourself.

Debugging

You're gonna spend time dealing with errors and weird behaviors so on. Just test as you go as much as possible. Don't write a whole bunch of code and then test—write a tiny bit then test. Debugging is its own skill, a fundamental of programming in general, deserving of a guide… I suggest googling for it, and please tell me if you find a good guide to debugging. Techniques have to do with purposely repeating the error, finding out how to unbreak your code and re-break it in whatever possible ways, looking live at the state of your data during each line of your code, and not moving on until you know why it happened. This can be painful. But you learn from it, and doing it well reduces real scientific errors. You'll have to debug more in R than in SPSS.

The "browser()" R function is the key tool I use for debugging.

Having to request every single test you want

e.g., SPSS will show you Levene's test, and Greenhouse-Geisser corrections. In R, you need to check Andy Field or Google for how to do them, click to install a package, learn how to use the command, and add that to your analyses. But hey, you could write a combo function that does your t-test AND Levene's test all in one.

Sending data to others

I say this because emailing SPSS output to others—who are already familiar with that format—is easy. But what exactly do you send to other people to show what you did in R? It may take time to set up something presentable, if they're not ready to just see what you see.

Building up a data frame and putting it into CSV -> Excel to send to them is pretty good I guess, and sending your data frame over to SPSS users used to be hard until I wrote my op.oe.SPSS() function, now it's easy.

But what I'm saying is, if others aren't using R, it can be hard to send them stuff.

R is very good at graphs, if you want to send those. Though these things take time to learn and are not automatically done via checkbox like SPSS. Look up the "ggplot2" library, that's what everyone uses.

The possibilities, the choices

Sometimes the decision itself, of how organized and elegant to make your code, is a burden. I haven't explained enough yet for this to make sense maybe… BELOW, I'm going to make a case for making your code elegant and reusable. The downside is, having to decide how far to go with that is itself a burden, as decisionmaking takes cognitive resources.

Use RStudio

Download and use RStudio (RStudio.com) now. When you download the R itself alone (cran.us.r-project.org), you get the code window and console window, which is barebones, but the creators of R always intended for other people to make a cooler interface. And RStudio did. What wonderful people, all these folks.

Explore RStudio

You could just start using it, quite intuitive. But don't! I urge you to initially, and continually, explore all aspects of it, including:

1. Read their intro help
2. Look up the keyboard shortcuts
3. Click all around to explore what it can do (such as listing functions)
4. Press tab to autocomplete… like for functions, variable names, and even filenames if you're importing a file or something

Did you know you can open multiple completely separate sessions of RStudio and the R processing just by using RStudio "Open Project in New Window"? This is so crazy powerful … so if your analysis takes a long time to process, you can leave it running and open a new project to continue working.

There are tons of cool RStudio shortcuts and tricks. One quick one: Clear the console (Ctrl-L) every once in a while. Once you've outputted like 1000 lines, it can get really slow.

Use a code file, not the console

Try to avoid typing in the console. It seems super cool at first, and make sense for temporary code. But then you realize you're trying to scroll up to what you typed 20 lines ago… it's actually bad.

Best is to always work in a code file. When I'm just playing around and am tempted to use the console because it's just for a minute, I make a play.R file. If you really need an even more temporary scratch file, hit Ctrl-Shift-N and use an Untitled. Run bits of code from there.

- Press Ctrl-Enter to run a line
- Press Ctrl-Alt-B to run all code from top of file up to where you are now
    - (By the way, as a peek into my own process, the "run from the top" is often what I do when working on my actual projects. It might even be running every line of code I've written for this project, if it's a small dataset. For bigger datasets, I do a save() after some of the meat-y processing, and then start up the next re-run with a load() command)

Learn the keyboard shortcuts

One more time: Learn them. Force yourself to use them. Take the time to stop, respect their importance, write them down on your own notes, post them by your computer, re-learn them next week, whatever you need to do.

My advice for a life with R

The best things you can do for your data analysis life with R are, in my opinion, learning and practicing:

1. **Moving data around**
2. **Organizing your code**
3. **Orienting yourself toward learning, not getting analyses done quickly**

Quick summary of my advice

1. **Moving data around**
    - You already spend tons of time on this without realizing it
    - Learn to flexibly process/transform your data in R… don't do it by hand in Excel
    - This is like training your general agility, which will help in all future analyses in your life

2. **Organizing your code**
    - You want: Organized, concise, logical, and easy-to-understand
    - It takes time and effort up front—you must really commit to this and it does hurt to spend the time—but saves time and effort later and reduces blunders (the scary kind) by a lot
    - You have many tools to do this, like creating your own functions, spreading across multiple code files (".R" code text files), using comments, and code folding
    - The *best* of those is creating your own functions

3. **Orienting yourself toward learning, not getting analyses done quickly**
    - Learning R well takes a long time
    - Respect the value of investing time in your learning
    - This means going beyond what you have to do to get the job done, and going into how you could more **flexibly and elegantly** do the exact same thing
        i. From a getting-this-one-analysis-done standpoint, it's probably a waste of time
        ii. From a project-long standpoint, might be a wash
        iii. But from a learning, reusing code for *next* project, and even growing-as-a-data-analyst, standpoints, the elegant route is gold!

By the way, #3 I haven't written a section about, I just wrote the whole guide emphasizing this in various ways.

I'll say one little tidbit about this though, well, a total tangent off on Word. I try to learn deeply in all domains I have time for, especially in computers, where I obsess over efficiency. Few people ever learn Microsoft Word deeply. Few people take time to stop, learn, and explore—but it's a piece of software you're gonna spend hundreds of hours in, you might wanna get to know it.

You can always just bold-italic-underline and tab your way through to make heading-ish formatting, and just deal. But I decided to learn properly how to customize Heading 1, Heading 2, paragraph margins, etc. And that's how I got to improving up this template, presentable and quite usable, making it easy to put together this nice-looking doc. (I'll share it if you want.) Plus I borrowed from my friend Donal's doc as a starting point, thanks Donal. But the point is, the exploring, learning approach, the elegant reusability—this approach is in action right in this Word doc, because the red doc template I've now used in 3 different projects for myself. Much like I've reused so much R code I wrote. It's nice.

1. Moving data around

What it is

Being able to import from raw data you're given, then remove unneeded columns, transform its format, create new variables, aggregate, process it, save it, export it to send to colleagues... this type of stuff we do all the time, but not everyone realizes how important this is and how you can do it many different ways. Automated and reusable ways are superior.

We never learned this!

I swear, we never learned how to manage data in stats classes. No one seems to talk about the key skill of moving data around. Just manipulating what you already have. People see this as just something that happens while you're trying to accomplish something else, as a timesink... but *because* it's a timesink, let's get on top of it. *Focus* on this meat-y subject and try mastering it.

Why it matters

Because you constantly have to do this stuff! It's tempting not to stop and think of it.

You're always getting data from various sources. You're dealing with renaming columns, getting rid of useless ones, reformatting, recoding, subsetting the data, saving it, moving it, reloading it.

Some RAs and grad students spend hours copying and pasting things in Excel. This is never the right way to do it. First of all, that may work for an N=30 study, but don't you want to run an N=200+ study someday? Also, doing things manually is error-prone, untraceable, and oh so hard to modify. For keeping errors low from computer and human, use both: Do some manual checking of cases against your code's output. This is generally stronger than the all-manual approach, even if you are worried about programming mistakes.

If you do your transformations in R, you can keep your original data file, keep the code for all the manipulations (such as questionnaires' summing and reverse scoring), and even years later you could go back and re-run in a modified way because of something new you may have learned about theory. Don't just modify something and get rid of the original. You'll notice R as a language is designed so it generally doesn't destroy original data when doing an operation to modify data (even at times where you might think it would). Good policy.

How to get started

This guide

So, might as well read this guide. There are exercises later, like the Code Ninja progression and the other exercises. These will be useful, but you might also read the Lam guide first.

Things you should master (by looking up in the guides)

- subset() function
- All the ways you can subset just using the brackets [] or [[]] (and knowing the difference between single and double brackets)
- How to work with "lists", which can store anything and everything, and take quite a while to understand fully
- How to save an R variable (the save() function) and reload it
- read.csv() and write.csv()
- Data types (character, numeric, factor)

Practice, practice, practice

So, when learning R, get curious! Don't just learn what you have to for getting stats done, learn all your options for manipulating data.

It should cross your mind, as you're going through the guides and doing your first project (and your tenth…): "I wonder if I could convert this to that," or, "What data types [character, numeric, factor] or what data structures [vector, list, data frame] would be my options for storing these data?" So, not just one option—learn ALL the options.

A lot of times you can do almost the same thing in 3 different ways in R, and you just need 1… but actually, it always turns out you get some deep-level understanding if you actively say, "Well, I'm gonna learn all 3 ways just for fun." Then, do "compare and contrast" for yourself, and you'll know which one to pick at the moment. And that process is fairly useless for that momentary task, but the understanding will endure and pay dividends.

Plus, you no longer feel lost, you feel found ☺. Though possibly fatigued... shows you spent effort on learning, congrats.

Make some reference code files for your learning

Make a folder for your reference code or learning code.

Make a file like "how to do stuff with data frames.R" and then add to it as you play around (*practice*) and read the guides. With # comments.

The first thing you can put in it is the code I included in my "PH demo" .R file, with "sandbox()". That's a good demo to keep for yourself and play with. Then add to it some sample code on how to use subset and the brackets I mentioned above under "Things you should master". Add extensive comments with #. Organize your note-taking via RStudio code folding (google those 3 words to see their guide to it).

More reference code files you'd make for yourself would be: "how to run basic stats.R", "how to load and save data.R", "fun with 'for' loops.R"

Envision the end goal

The level of agility you want to gain is that when you get data in front of you, and you're like "I wish it was set up in this other way," being able to just do that in a few minutes. At this stage, becomes easier. And it's nice if you work with people who might thank you because "whoa if I had had to do that via copy/paste it would have taken a few hours."

2. Organizing and structuring your code

We never learned this!

"How do you keep track of your all your different analyses you've run?"

"How do you take notes so you can return to a project a year later and still understand what you did?"

"[One year later…] How the heck did I do all these analyses that I'm now supposed to tell someone who asked about them?"

I just don't think these questions have been addressed, not in workshops, not in class, not in any R guides. I've worked a lot on my own organizing systems, and how to keep track of past analyses, and I've found some great methods. But I'm always trying to, uh, organize them more. And revise. I basically overhauled it a year ago and am doing it again this year. Hopefully once I settle on the design this year, I'll be able to share.

But anyway, I suggest you work consciously on your own systems for you. I'm drawing your attention to this because no one ever told us how to do it or even that it's important, though you probably noticed well before reading this that it is. It's worth thinking about, worth losing sleep over, and then resting easier when you have a reliable setup.

What do you mean, organizing?

Mainly, **structuring**.

In a fully abstracted sense, organizing is about putting things where they belong, according to their classifications.

So what are the "things"? Chunks of code.

The code might be for alternative ways to analyze the data, ways to slice it (subgroups), or running the same analyses on different variables.

The ever-present choice – just get it done, or stop and learn

You have an opportunity, pretty much *any time* you're working on your data in R, to choose. You will confront the choice many times, honestly in life as much as in R.

Option 1 -- "**quick and dirty**" approach: Run your stats as fast as possible, stick with what works

Option 2 -- "**elegant and reusable**" approach: Think about what you're doing. Write some code. Re-write it so it's more concise. Re-write it as a function. Automate what's being done so you do the least copy-pasting or nearly-identical blocks of code.

#2 could be called the "road less traveled." But all seasoned programmers, like computer science majors or pros, know the virtues of #2. They know. I keep proselytizing about elegance / reusability because most of my colleagues aren't programmers by nature. So, you must learn the virtues. And you will do so in the following ninja training sequence. (For accompanying music, listen to the 2006 album, "The Training Sequence EP" by me and my former band, myspace.com/themonkery , or turn on some kung fu movie I guess.)

An in-depth example/exercise in iteratively making your code more organized, elegant, and reusable

Say you're analyzing data frame "sdf", with columns (variables) i and n. You're only interested in whether p is significant. Run sandbox() to get that sdf.

Code Ninja… Level 1:

First, you write it like this:

```
t.test(sdf$i) # Compares i column to 0

t.test(sdf$n) # Compares n column to 0
```

It works, and you could stop.

Code Ninja… Level 2:

But since you're learning R, you want to explore. Plus, you want to organize your output, so why have all that t.test info popping out when you just want a "yes or no" about significance. So, you press F1 on your t.test command and see the R help. You figure out to do this:

```
tResult <- t.test(sdf$i)

ca('p = ', tResult$p.value, ' for column i')

if (tResult$p.value < .05) {

        cal(", so it's significant!")

} else {

        cal(", meh, it's nonsignificant.")

}
```

And then you copy-paste the same code for the n column instead of i.

Code Ninja… Level 3:

When you copy paste it, you ninja with RStudio: use the Find and Replace capability of RStudio to turn all instances of sdf$i into sdf$n, because you've explored RStudio and know all about that. Pretty quick really.

Code Ninja… Level 4:

But you're realizing that copying and pasting is kind of a hack, a quick and dirty way to do it, and you want to proceed to the next level of "elegant code." Well, first you figure, I only need to do one variable at a time, so I can just choose which variable I want, and re-run the code whenever I need to do another variable. This is a reasonable improvement, and I do stuff like this a lot when I'm playing with ad hoc analyses.

```
currentVar <- 'i' # Not bad, because you need only change this line to test a new variable
```

Then use the same code, but Find and Replace all instances of "sdf$i" with "sdf[[currentVar]]".

Also replace the line: `ca('p = ', tResult$p.value, ' for column i')`

With this: `ca('p = ', tResult$p.value, ' for column ', currentVar)`

Code Ninja… Level 5:

Then you realize, you can loop through as many variables as you want! So, if your data has 5 other numeric variables, you could go ahead and test those too.

```
currentVars <- c('i', 'n') # can put as many columns (aka variables) as you want

for (var in currentVars) {

    tResult <- t.test(sdf[[var]])

    # Here, insert the whole if statement with the "so it's significant" and everything

}
```

Code Ninja… Level 6:

Then you figure, why not just do it on ALL columns in the data frame? This will work if all columns are numeric or integer. Well, names(sdf) gives us all column names.

So just change the currentVars to: `currentVars <- names(sdf)`

Code Ninja… Level 7:

Okay this is cool code. You can use it for your next project easily, because whatever the data frame of your next project is, you can just do sdf <- thatDataFrame and then run this code. So, maybe your ways of organizing this code are to keep it in its own "one-sample t-tests.R" file, and write comments at the top. Not bad, you're at Level 7.

Code Ninja… Level 8:

What about making your own function?

```
runttests <- function (df, currentVars) {

    # Put the exact same code you got up to from Level 5 here, but replace "sdf" with just "df", because we
```

kind of want a more generic variable name, df, not sdf which I intended to mean "sandbox df".

}

I can now run runttests(df=sdf, currentVars=c('i', 'n')) like Level 5 and that's cool.

The other thing it'll do is, it will not clutter your lines of console output with all the code you've run. It'll just put that one function runttests call line on the console, so the other code stays hidden. This is appropriate and organized and elegant in many cases when you want to present just the info (to yourself or someone else) that belongs as "output". You have now created a function that has a clear purpose, so the call to the function is all it should display really. Except for your results, which it WILL show because you did explicit ca() calls to do that.

Code Ninja… Level 9:

What about running all columns, like in Level 6? This one is for you to figure out…

Hint 1: You don't change the function's code itself. So what should you change then?

Hint 2: What does names(sdf) get converted into when R interprets it?

So, you'll see.

Code Ninja… Level 10:

Okay, now it's 1 year later, and you're working on a new project. You want to t-test all the columns to see which are significantly different from 0.

Your new dataframe is called dissertationDataFrame. Ohhh, scary, dissertation time. But this re-use of your code part is really not scary:

source("C:/Dropbox/myfolder/one-sample t-tests.R") # Assuming you saved your Level 9 function in there…

runttests(df=dissertationDataFrame, currentVars=names(dissertationDataFrame))

(P.S. I just gave away the Level 9 answer.)

Code Ninja… Grand Master Level 1:

Grand Master ones are harder but very cool. This one would be really useful.

Your mission: Instead of using ca() to print the output, build up a data frame with the desired output. So, this special data frame you're making contains p values and significance—you're creating it for the purpose of printing and saving it. Almost like it's an output frame not a data frame.

Each row of the data frame should have the name of the variable you tested, a p value, and a "sig" or "nonsig."

Hints: use rbind() and probably data.frame(), adding one row each cycle of the loop

One more step: Up until now, your runttests has been "outputting" by ca(). Now, make it just output the newly built dataframe using return(). This is the fundamental way functions are supposed to output data, as hopefully you're learning from some other guides about functions.

Now you can run this code to see your p values:

cooldf <- runttests(df=dissertationDataFrame, currentVars=names(dissertationDataFrame))

op(cooldf)

Or, all in one line: op(runttests(df=dissertationDataFrame, currentVars=names(dissertationDataFrame)))

And then, in Excel, it's easy to do stuff like display only 2 decimal places, or display a lot of decimal places, or copy paste into an email. Or use pretty colors.

Code Ninja… Grand Master Level 2:

Let's try the same thing, but with a different code structure. With your ninja ears you heard from the guides about this command called "lapply" (which happens to run faster if you happen to need to process like thousands of columns).

So, I call upon you, as your final challenge, go forth and find out how you could replace the whole for() loop with a use of lapply or apply. (Well, after you figure it out with one of those, use the other, just for fun.) Hint: You'll have to make your own simple function that just does ONE currentVar, and instruct lapply to make use of that function applied to all columns of your data frame.

It's easier to start with the ninja level versions that do output via ca() instead of via return().

The apply() functions were quite hard for me to wrap my head around. But they've been clutch in several places I needed 'em. So, good luck.

Could you be a Code Ninja in SPSS?

Well, running t-tests on a large number of variables is super easy in SPSS. But you get this mass of output, and you get it SPSS's way. And if you want to clear up the output, you're gonna click on boxes in the output and press delete. You can't automate that.

But with our way, YOU chose what to write as output.

Also, you have a way to refer to the output, manipulate and store it. With SPSS, all you can do is copy-paste out of SPSS, and more importantly, there's no way in SPSS syntax to be like "let's take that table of p values and multiply them" or "tell me how many are significant"—for the latter you could look at how many stars there are. But that's SPSS's way, not yours… And ninjas do what they please.

Should I really bother putting things into functions? It seems to take so much time and… *effortful thinking*

Yes, as much as possible, you should.

Thinking is good

When you write a function, you're forced to think about what you're trying to accomplish, what you need for the job, and what the inputs and outputs for **this particular step** are.

The thinking gets crystallized

Since you've made a function, you've been forced to think, what *kind* of a step is this? What would I call it? (Feel free to use insanely long function names if you want. You can use Tab to autocomplete while typing them.)

You've been forced to logically decide exactly what is required as input—those are the parameters/arguments. And that input could be various input, and the function would do its job. And what goes as output should be the return() value ideally, but that can sometimes be hard to do fully[2], and it may be convenient to basically use ca() or print() as your form of output during the function. But using your own functions can help to *not* have masses of distracting output like in SPSS, or even the R t.test which shows you too much text about the test and takes up a bunch of lines.

A year later, when you look back at your code, well if you didn't use functions and you just have 200 lines of code, how will you know what's extractable to be used in your new project? Does this block of code rely on code above it? Which parts of the output are you supposed to pay attention to or ignore? Not cool to go relearn that stuff.

But if you made a function, if you programmed the function appropriately (a great programming standard: It should be kind of a self-contained module, not messing with the world outside the function), then you'll be very clear on whether you can just pull that function out and use it in your new project. Or what little tweaks it may need, and you'll have confidence that these tweaks won't break the system, because the function's purpose, inputs, and outputs have been well-defined.

That's why functions are good.

Other benefits of using functions

Clean output – no need to show your messy code in your output

Ideally, your output should contain *only* relevant info about your results. This is why I use cat() and my own opm() function system to output specific messages, not just dump all the code execution as my output.

If you're using straight code, without putting the code in a function or using source() on a file, you're getting this whole jumble of your code outputting along with the useful stuff like your t-test results. You also get every single return value in full, like when you Put it in a function, and it'll show nothing. You'll just add in code to specifically choose what goes in the output: e.g.:

```
cat("key p value of interest, from our t-test between active and control, is: ", t.test(etc…)$p.value)
```

Execution speed (this might surprise you)

So, R can run thousands of complex code and calculations in 1 second, but outputting *to the console* is super slow. If you're just running straight code, in your currently open file (as opposed to doing source("othercode.R") or calling one big function), odds are that when you run you're code, you're sitting there, and you *think* you're waiting for R to execute your code, but you're *actually* waiting for it to just print to the console! This applies to when you're outputting a lot to the console, too, which is why for outputting big data frames, best to output to CSV or something. (My ph.op() function is very useful.) To see how much time you can save by not outputting to the console at all, use sink("outfile.txt") before running your code, and all output will go straight into the file "outfile.txt", so you see how long that takes compared with what you've been doing.

[2] because to return more than one thing from a function, you have to return ONE list containing multiple things.

To test this, simply copy/paste all the code you're running into a brand new source file, then source the active file (RStudio Code->Source is Ctrl-Shift-S)

R wants you to use functions

It's all based around functions. So you often need your own functions to work with some of R's mechanisms, like by(), apply(), and various fancy stuff. You can even use functions inside your functions, or pass your functions as arguments into your functions, or whatever. It's all flexible once it's in a function.

Mediocre alternative: Splitting your ideas into separate .R files

Well you can do that… and I do use some of that. I have multiple hierarchical tiers of organization I guess, and the files are a higher level than the functions, and the two levels are helpful to have.

You can decide to make code .R files as self-contained modules too. I've done that.

Still, it's ideal to put things in a function whenever possible, so you're guaranteed to know the inputs and outputs, whereas things get hazier on that front sometimes if you use code files without putting stuff in functions. Also, with functions, if you create temporary variables, they're gone at the end of the function and won't clutter up anything thereafter. You also won't accidentally overwrite any variables that exist outside the function. For example, if you use "df" as your temporary data frame name all the time in all your code, that's okay if you use a function, because the "df" you create inside the function is in a separate scope from the rest of your code. If you use source(), the scope is all shared, so things are not cleanly separated, and to get rid of extra variables at the end of a code file, you need to use rm() on all the variables, which I recommend if you're using sourcing.

Another thing that's great about functions is, you have a name for them, and you can pile a ton of them into one code file and still navigate it because of the names. RStudio has various mechanisms for this, such as F2, or the little arrow thing in the bottom left of the code window, or Go To File/Function (Ctrl-.). And you can Tab to autocomplete your function name when typing it.

Fun R exercises for you

Do these once you've learned enough so they start to look maybe doable. Then try doing them, and as you do it, look up more stuff (in the guides, R help, and google) to accomplish it.

These are the types of exercises I did myself. This can get you started, and you can basically make up your own exercises on the fly, by being curious. It's about taking the opportunities to venture into how this stuff works.

Run the sandbox() function first in my attached R code.

11 Exercises… in order of increasing difficulty

1. Add two numbers and say the number is cool by using the ca() function

2. Do the same thing but create a variable called toPrint that you'll print via ca(toPrint) or similarly print(toPrint). In making this variable, use pa() (see my PH functions). Then, re-write it using my special syntax ("%_%").

3. Put a whole bunch of code you've so far written in R into one file. Use RStudio's code folding features to organize it. Make your own headings, and think about the best ways to name and show those headings. Code folding is not as cool as functions, but it's a decent tool. (Google for RStudio code folding and their guide to it comes up.)

4. Convert a data frame to a list and then back to a data frame. Write down your understanding: Compare and contrast data frames and lists. Lists can hold anything in them. But what limits are there to what data frames can hold?

5. Replace sdf$i with random numbers.

6. By the way, how many ways can you say "sdf$i" more or less equivalently? (Different syntaxes.)

7. Triple the number of rows in the "sdf" data frame. Do it by rbind() it over and over, by adding lines of NA…

8. "i" is the first column of sdf. Make it the last column. Hint: remove the column temporarily and then bind it back on.

9. Say you want to do stuff to some certain columns, like t.test on columns "n" and "i", but you're sick of typing that kind-of-ugly "sdf$" part all the time (like t.test(sdf$n))? (Often times, it's okay to just type it a lot. But for when you don't want to, look up in R help the with() and within() functions… kind of confusing at times, but very useful. You just set up the with(sdf, expr = { bracket deal, then you can t.test(n) in there. And you can use with in conjunction with subset, FYI. Don't use attach() ever.)

10. Make your own function, a t.testWithNiceOutput() for example… it takes the output of t.test and turns it into your own custom format of a one-row data frame. That way, it looks organized, fits on one line.

    - **If you're like me, you'll use this approach all the time forever, as your way of viewing R's t test and correlation outputs, and you'll make your own descriptives function like this instead of using summary()**

11. Take that function and make a new one that can run 10 t.tests. You can rbind() together the outputs, so you have just one output data frame with all the info, sort by p values and op() to Excel to play with.

Concluding remarks

If this guide has helped you, and you would indeed take the time to try learning more along these lines, and some more of my systems for functions and an organized system for the steps of data analysis, and a system for archiving every analysis you ever run… just tell me in an email and that will encourage me to do more. I'm trying to make it all more coherent and shareable, but fully doing that and writing is only worth it if I know at least a few people will try using it.

Anyway, go forth and develop your own systems, your own rules about how you structure around functions, code files. If you broadly generalize from the Code Ninja training sequence, that can get you there.

Yeah so, that Code Ninja example sure was long, right? A little obsessive, one gets. Well, that type of thing is what I've spent a lot of time doing in R. And about half my efforts in doing the elegant reusable versions have not paid off. Life may have been better is I did more of a quick and dirty version. But hey, I couldn't have known that without trying it out, and I still learned things from those experiences.

Now the *other* half of the attempts, it's been just peachy. I pride myself on the reusable code I've built up. The last 2 data analysis projects I did went really quickly because the data were similar to past projects, and I reused old code that was elegantly written, put into functions, documented by comments, and placed into a few separate .R files. I also did use RStudio's find-and-replace, I did use functions, and I did use some quick and dirty code here and there especially the night before it was due.

I can re-run all those analyses anytime, now or a year from now, even if we need to modify the very first step in the whole processing of data… this is a 5-minute job. That's cool. And not feasible in SPSS, or any process where cutting and pasting is heavily involved, or if my code wasn't organized and function-ized enough to see the clear logical step progression of it all.

So, I hope you get a bit of that satisfaction yourself. I hope some of this is wisdom. I really don't want to paint a rosy picture either and have tried to express frustrations foretold as well as triumphs promised. There's a road R has taken me personally on, a fruitful journey, not for everyone, but by getting this perspective as you start out, you can make the wisest of choices for your ninja path.