Data analysis tools

David Pham

March 14, 2020

Contents

1 Introduction

A friend ask me what I would recommend to learn if someone wanted to join the field of quantitative finance, machine learning or statistics (the basic skills are similar). He also asked to make some small remarks on why these tools were relevant.

First of all, this is only a matter of personal taste and they are a lot of reason why someone would learn and act differently, but my hope is that I can provide some justification on why I invested time to learn these tools.

I personally use all the following tools on recurrent basis and I would advise to really to learn the basics and use the programs that really help you in your daily work. Obviously, this list does miss few programs/languages, the reason is probably that I can not assert that I used these other languages enough to have a decent opinion of them.

1.1 General advice

If I may give only one advice, it would be the following:

DO NOT REINVENT THE WHEEL.

Search hard for a solution of a problem, or try to shape a problem into a solved problem. Even if you are the best scientist and coder, great professionals almost always win time by only copy-pasting and understanding solutions that already exists. However, during your training, retry to rewrite all the scripts that you learn: one learns about typos and subtle mistakes by doing so.

2 Quick overview

This is a list of tools I genuinely recommend to learn and use in our daily life when interacting with a computer.

A good point to remember about the suggestions is they originate from my (little) experience and my concern about tractability of the analysis, reproducible research, and productivity.

Let's get started! Here is a short version:

- **Ubuntu**. Just do yourself a favor and work in an UNIX environment. Ubuntu Gnome homepage
- **Emacs** as your environment: write and code in any language without ever without leaving your f/j keys in the same consistent environment. Emacs homepage (or here for a more friendly-user version).
- **Git** and **Github**: modern way for archiving, collaborating and working with files. Git homepage
- **Pandoc, markdown**: write content and not the formatting. Markdown and pandoc automate most of the editing tasks. Pandoc homepage
- **Python**: your first and most fun programming language. Learn to create programs (not class) in a funny and interactive fashion. Python homepage
- **R**: your friend for visualizing and analyzing data. R CRAN homepage and RStudio IDE (always handy when starting with R).

3 Links

3.1 Emacs

- Start with `alt-x help-with-tutorial`, to get the first introduction.
- Emacs with R
- Some nice introduction from Udacity.

3.2 Data analysis

- Guidelines for Statistical Projects: Coding and Typography: Great reading and advice for writing LaTeX documents and R Code.

3.3 Git and Github

- Gentle introduction to git and github.
- Udacity guideline for editing git commits.
- Git official website

3.4 Python

- Best introduction ever to python.
- Scipy
- Scikit
- Seaborn

3.5 R

- Simsalapar documentation
- RStudio blog: a good point for looking at best computer practices.

3.6 Blog post

- Language to master, when beginning to learn software engineering.
- Spaghetti code, what to avoid in bigger projects.
- IDE vs Unix philosophy, or why Emacs is awesome. Make small things, that do one thing well.
- Unicode: If you don't want to smash your head against the wall because of accent.

4 Details

The goal of this section is to provide a basic justification on why I am using these programs or framework.

4.1 Ubuntu

Any Linux distribution would be fine. However, I would recommend Ubuntu as the first point of entry to the linux world. The advantage over Windows are following.

- Open source software: Most advanced tools for data analysis rely on some tools of the UNIX world and their installation is extremely simplified.
- Ubiquitous terminal: interacting with several languages, tools is the favored way of creating analysis (small modular pieces). Using the shell allow us to easily automate the boring tasks with shell scripts.
- Remote computations: Most cloud computing services are based on a UNIX distribution.

4.2 Text editor/IDE: Emacs

This topic is probably one of the most controversial. Most of my work goes around reading and writing. I am unfortunately paid to write and authors things: code for analysis, automating tasks, formatting or cleaning data, and last but not the least, writing documentations and reports. One of the source of inefficiency when writing documents is the time we move our hands from the keyboard to the mouse in order to move the cursor around a document or characters. Emacs provide a rich amount of shortcuts to help the user to write everything without leaving the keyboard and it also allows to completely modify them as well.

- Truly personal editor: One of the most interesting feature is also that one can really suits Emacs to its need. For example, I have a Swiss keyboard, and deleting backwards is a task I often have to do. However, with my keyboard layout, I have to move little finger to backspace, which disturb my right hand position. Hence, to solve this problem, I simply bound *CTRL-é* and *ALT-é* to `delete-char-backwards` and `delete-word-backward`.
- One editor for all languages: With Emacs, you will develop code (python, R, C++ and clojure are supported), edit text (LaTeX and Markdown) and use the shell, all with a consistent set of shortcuts that you set.

Note that I do not claim, it will be easy at the beginning. However, after a few days, Emacs will save you thirty minutes to a hour each day!

4.3 Document generation: Pandoc (markdown, LaTeX)

Stop using Word or Libre Office Writer. Just stop it right now! The flaw with these document editors is the user has only one interface with the program for two distinct tasks: content creation and formatting.

Users often want standard format and focus on the content. LaTeX and markdown are markup language that abstract the formatting process from the user. Then, the computer takes care of the references, the layout of listings, the exact amount of spaces around words.

Markdown is simplified version of LaTeX (but still really capable) and **pandoc** is a program that can freely convert your markdown or LaTeX file into pdf, html or even docx.

A good argument in favor of these pure text file is that version control come almost for free and UNIX diff functions (to display difference between documents) works seamlessly.

4.4 Programming languages for data analysis: python and R

There is no clear winner between the two languages when analyzing data and each of them has its own advantages. Both provide great advantages:

- REPL (interactive console for coding);
- Open source;
- A wonderful open source community;
- An uncountable set of tutorials for beginning;
- Object-oriented and functional programming paradigm;
- Easy interaction with most used compiled languages (C, C++, Java).
- Exceptional documentation for the base package.

Advice: if you have little experience in coding, I totally support learning **python** first and then quickly pick up with **R**, because **python** also emphasis data structures (aka speed of your code) and software engineering (maintainability and reliability).

Main differences are emphasized in the next subsections. Note, should you choose to use **python**, please try to learn **python 3** instead of 2.7 as it will be depreciated in 2020, and no 2.8 is planed.

4.4.1 Python

Python is often favored by the machine learning community.

- All-battery included programming languages: there is a module for almost everything in the base installation (csv, itertools, regular expression).
- Explicit data structure (list, tuple, set, frozenset, hash-map/dictionary).
- There is only ONE true way of writing code (so most people tend to have the same style).
- It has one of the most beautiful syntax among programming languages.
- There is a default support for modules (i.e. writing independent code in other files and reusing them) and it is trivial to use (`import my_other_file`, wher `my_other_file.py` contains what you want to reuse).

4.4.2 R

R is the standard programming language for most data analysts.

- Almost any statistical learning algorithm has already been implemented and is available on CRAN.
- Installation of new packages is a lot easier, when the user has no **admin rights**. This is really important as most companies do not provide them to their employees.
- Programming language created by statisticians for statisticians.
- Syntax is really targeted for performing data analysis.
- Fairly high level language. Packages should take care of most system error (segmentation fault or out of bound memory).
- Amazing data visualization capabilities.
- Interactive data visualization with javascript is definitively more advanced than with python.
- *Vectorized* operations are favored to `for` loop.

4.5 Version control: git and github

I have honestly used them since only recently. Hence my opinion and my experience might not be really significant. But version control definitively helps. It avoids having archive folders everywhere, and it tracts the development of our work.

4.6 Finally, if you want to get a corporate job

4.6.1 SQL for databases

I am honestly not a specialist of databases, especially as I prefer simple format such as csv. But as data grow larger, it is clear that database have advantage. I suppose we will only remain user, so it does help to know the basics of SQL. From there, a lot of language are similar but if one has to write queries often, some packages from python or R will help to not become crazy.

4.6.2 Excel, VBA

Excel and VBA are the most dangerous tools in the corporate world. Anyone can make extremely advanced data analysis and plots. However, it is almost impossible to reproduce the research and to avoid mistakes. Hence my advice is that, if possible, try everything to avoid using with Excel files:

- Use some python module to interact with Excel.
- Create vba macros to export/import data in `.csv` and call a python script with the `Shell` vba function to work with the data.

If anyhow, you still MUST use Excel, here are some tricks:

- Excel
  - Pivot table allows to create simple "*-by" summary of your data (e.g. "sum-by region", "count-by country").
  - Filter function and deleting duplicate elements are great built-in tools.
  - `vlookup`: the best way to reproduce the behavior of hash-map in Excel.
- VBA
  - Record macro is your best friend.

- Turn off display updating when running macros.
- If possible, work with arrays in memory and not cells on the spreadsheet.
- Use Excel functions whenever possible.
- Include Microsoft.Scripting module to get hash-maps (associative arrays).
- When filtering data, use the `filter` function from Excel with the correct parameters.

5 Intermediate topics

I consider here some intermediate topics

5.1 Ubuntu

I don't have much to add about which OS you should use. You can argue that most paid job require some knowledge of C# and the .Net platform. Nonetheless, most open source tools have been thoroughly tested on Linux machines, hence you will avoid some subtle bugs by adopting Ubuntu.

5.2 Emacs

I would like use the same tool at work and at home, as I want to be able to improve my skills continuously, especially when I am working (and we do use Emacs at my working place now). The alternative solution is to use the IDEs. Nevertheless, most of them are quite complicated (which is also true for Emacs) and the possibilities of customizing them are quite limited.

Moreover, using a bare text editor force professionals to understand the engineering process behind their products when compiling.

One of the best argument, I have heard about using Emacs are the following:

- Emacs has been here for the past 30 years, and will probably still be there in the next 30 years. One of the reason why it could disappear is the non-integration of webkit (but we will figure out how to make it).
- True integration with UNIX tools: You will find yourself using grep, find, top without knowing.
- A wonderful way to learn to code as well.

- Most importantly: the feeling of mastery. Every day, I learn something new about Emacs and I am sure I can use it in my workflow. Otherwise I learn something about elisp (which is never bad). Overall, it made me a better programmer as well as it gave me the feeling about being able to tackle all challenge with a common set of tools.

5.3 Document generation

A good option in combination with Emacs is to use **Org-mode** for **literate programming**. Org-mode is an equivalent of markdown (and is also supported by pandoc). Nevertheless, it provides a better integration with Emacs and its default HTML and LaTeX output hare nicer. Bonus: it is a really good TODO list remainder.

Literate programming is also an interesting topic: usually, programmers write code, filled with comments, but this could or should be reverse. Authors should write text with code blocks describing their views. Org-mode totally support this type of document. For a good alternative, search for `rmarkdown`.

5.4 Python

The **Scipy** stack includes most of what you will ever need to replace matlab: numpy (approximately matrix calculation), scipy (optimization and numerical mathematics), ipython (a better repl/interactive console), pandas (data.frame like data structure), simpy (symbolic mathematics) and nose (for testing). **pandas** is the most important package to learn for data analysis.

Scikit-learn offers most machine learning algorithm for statistical prediction. **Seaborn** is a great abstraction over matplotlib for plotting *data.

5.5 R

There are several types of R packages: packages providing statistical models, others providing infrastructures for performing analysis. I will give you some advice for the second type of packages.

simsalapar abstract and provide a great framework for computing simulation over grids of parameters.

data.table is probably the most interesting packages. It offers a C++ implementation of the data.frame data structure and provides incredible speed up. The `fread` function also accelerate incredibly the loading of *csv* data. One of the drawback is the way it uses expression and is maybe not the most natural way for R programmers to use it.

ggplot2 and **lattice** make the plotting experience of R even sweeter. lattice gives quick visualization for faceting (e.g. xy-plot for x against y grouped by z), whereas ggplot2 wraps the **grid** package to make highly customized data visualization. lattice is often faster than ggplot2, but ggplot2 is maintained and its output can customized.

caret abstracts the best practices of data analysis to allow more flexibility. It offers a framework to compare and average models and perform the split of data (test-training-evaluation set) automatically.

parallel come by default in R. I suggest to learn how to write parallel code with clusters and not with multiple cores. The reason is that forking copies completely your master thread which could cause some problem if you have a finite amount of RAM.

timeDate from the RiskMetrics team is a really appreciable abstraction of calendar issue in the financial world.

regular expression (or **regex**) are supported by default in R, learn about them, they are life savers!

magrittr provides the pipe operator `%>%`. More or less, it allows to read expression from left to right, compared to inside to outside:

```
x %>% f %>% g(y) == g(f(x), y) # TRUE
```

5.6 Statistics: Repeat the basics

I love shiny statistical learning methods, I mean, I do love them (neural networks, lasso/ridge regression, random forest, copula modeling). However, I am convinced that for mastery, one do have to repeat the basics often and to have solid understanding of what *randomness* means in statistics.

6 Things that exists which I never used professionally

As any curious person, I try to surf on the net and see what could be interesting to add to my set of skills. Unfortunately, I could never professionally use the tools which I am describing below, but I wanted to point at them for the sake of completion.

6.1 Linear Algebra

Matrix notation is certainly helpful. Nevertheless, for advanced numerical computations and optimization, *vectorization* (using values locations in memory to perform calculation) is primordial. I unfortunately never had to deal with such constraints, but it would not hurt to have an overview about LAPACK, BLAS and ATLAS.

6.2 C, C++ and Cuda

C and **C++** are almost the *de facto* way to improve R or python code and **Cuda** gives access to the power of your graphic card for computing. I do not have a strong opinion of them, I may be too inexperienced.

What I do know is that for short function, these language are really manageable and quit elegant as well.

6.3 Java (JVM), Clojure, Scala and Jython

No one would argue that Java is used everywhere. However, the language has its constraints on its syntax that makes it quite not enjoyable.

I really appreciate **Clojure** for its simplicity and for its tradition for Lisp, however, one will more easily find a job with **Scala**.

As last resort, **Jython**, the JVM implementation of python, can also help.

6.4 Javascript and Clojurescript

Interactive data visualization is the future and your browser is probably one of the best tool to display data. **D3.js** is one of the best framework and **metricsgraphics.js** wraps around it in order to abstract some of the complexities.

Clojure is a wonderful lisp dialect, and Clojurescript uses the same semantics, but targets the Javascript instead of the JVM. Even though I am still in the process of experimenting Clojurescript, I am certain that it will enhance the maintainability and performance of my code.

6.5 Python

Cython seems to be a nice start for optimizing your python code. If you really want to go into new fields, **Tensorflow** from Google is a good start.

6.6 R

Rcpp is a wonderful way to interact R with C++.