# Run Directory Automatically for a Research Project

Qichao Wang*

25 December 2024

A research project involves a lot of steps. Generally, there are the following steps:

- Data cleaning
- Data transformation
- Data analysis
- Literature review
- Paper writing

However, each step requires at least some manual work. Sometimes the data steps involve multiple statistical software. For example, a project in macroeconomics possibly involves processing raw data in Stata before further analysis in MATLAB. When the number of software used increases, manual work can be laborious. To liberate researchers from manual work, automation at a higher degree is desirable.

I was illuminated by Gentzkow and Shapiro (2014) to adopt the complete automatic process for a research project as an additional step from a manual pipeline from data processing and plotting using Stata, R or MATLAB to typesetting the research paper using LaTeX. The Python program is based on Hofman (2018)'s work with modifications.

The report is automatically generated down the line from the python script file `rundirectory.py`. I include the functions in another file `rundirectory_function.py` with necessary pre-settings, including the directories to call the software in the Mac terminal. All the contents in the handbook are self-contained in the code files.

## 1 Stata

The automation can handle Stata files. The caveat is that the only plot format the terminal version of Stata supports is eps.

Currently, I optimize the compatibility of the Stata processing, so it works in both a Mac environment and a Windows environment.

*I am a PhD student in Economics in the Department of Economics at The Hong Kong University of Science and Technology. Email: qwangcq@connect.ust.hk.

I run the following Stata code.[1]

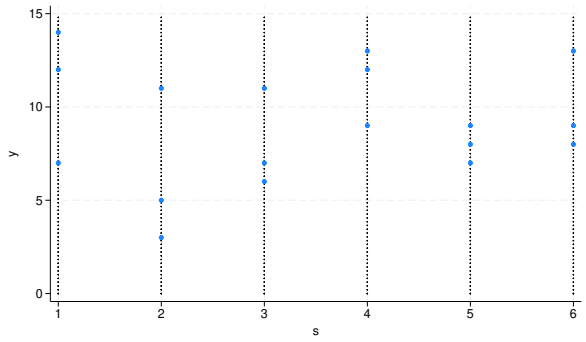
```
* This is a sample file.

cd "/Users/WilliamWang/Library/Mobile Documents/com~apple~
   CloudDocs/Documents/GitHub/rundir"

use "co3.dta", clear

twoway dot y s
graph export "sample_stata_graph.eps", replace
```

The function to run Stata code, `run_stata`, is as follow.

```
def run_stata(fileloc):
    """Run Stata do-file in batch mode, delete the log file, then
        go back to the original directory"""
    fileloc = "/".join([dir_origin, fileloc])
    script, dir_script = parse_location(fileloc)
    os.chdir(dir_script)

    if sys.platform == "win32":
        subprocess.call([loc_stata_win, "-e", "do", script])
    else:
        subprocess.call([loc_stata_mac, "-b", "do", script])

    err = re.compile("^r\([0-9]+\);$")
    with open("{}.log".format(re.sub("(.*)\\.(.*)$", "\\1",
       script)), 'r') as logfile:
        for line in logfile:
            if err.match(line):
                print(line)
                sys.exit("Stata Error code {line} in {fileloc}".
                   format(line = line[0:-2], fileloc = fileloc))
                lastline = line
                print(lastline)

    os.remove("{}.log".format(re.sub("(.*)\\.(.*)$", "\\1",
       script)))
    os.chdir(dir_origin)
```

The output plot is as follows.

[1] I use the first data file from: https://stats.oarc.ucla.edu/stata/examples/kirk/experimental-design-procedures-for-the-behavioral-sciences-third-edition-by-roger-e-kirk/ here for illustrative purposes.




# 2 R

The automation can handle R files. I have not yet found the solution to use the Mac terminal to open a project in RStudio, run an R script within the project, and then close the project. However, I can still run R scripts using all functions generated in the clean-and-install process of the project. Just load the library named after the project, and the functions are available. To use the processing on an RStudio project, add a line at the top of the R script to set the working directory as the root folder where the `.Rproj` file is located.

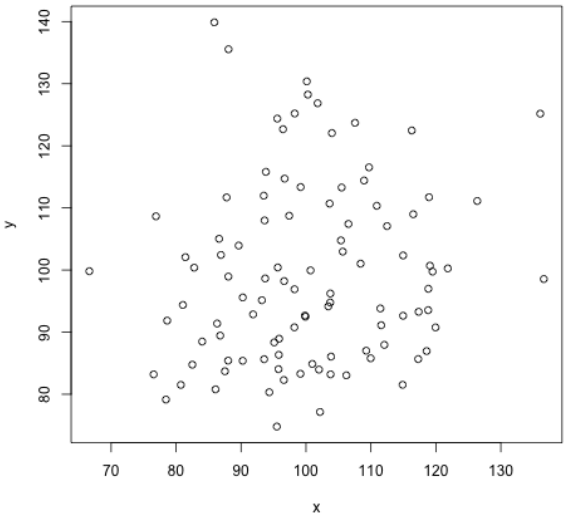
I run the following R code.

```
# This is a sample file.

setwd('/Users/WilliamWang/Library/Mobile Documents/com~apple~
   CloudDocs/Documents/GitHub/rundir')

set.seed(0)
x <- rnorm(100, mean = 100, sd = 15)
y <- rnorm(100, mean = 100, sd = 15)

png(filename = 'sample_r_graph.png')
plot(x, y)
dev.off()
```

The function to run R code, `run_python`, is as follow.

```
def run_r(fileloc):
    """Run R script, then go back to the original directory"""
    fileloc = "/".join([dir_origin, fileloc])
    script, dir_script = parse_location(fileloc)
    os.chdir(dir_script)
    subprocess.call([loc_r, "--vanilla", script])
    os.chdir(dir_origin)
```

The output plot is as follows.




# 3 MATLAB

The automation can handle MATLAB files. Running scripts within a project environment is supported for MATLAB processing. Nesting batch processing with parallel computing within a script substantially boosts efficiency. Dynare codes can also be nested within the script.

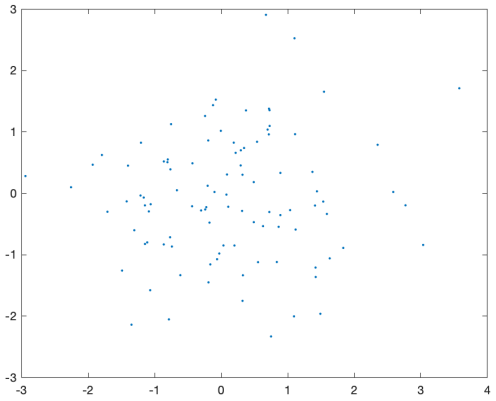
I run the following MATLAB code.

```
% This is a sample file.

rng(0, 'twister')
x = randn(100, 2);

plot(x(:, 1), x(:, 2), '.')
saveas(gcf, 'sample_matlab_graph.png');
close;
```

The function to run MATLAB code, `run_matlab`, is as follow.

```
def run_matlab(fileloc):
    """Run Matlab script, then go back to the original directory
       """
    fileloc = "/".join([dir_origin, fileloc])
    script, dir_script = parse_location(fileloc)
    os.chdir(dir_script)
    subprocess.call([loc_matlab, "-nodisplay", "-batch", script.
       split(".")[0]])
    os.chdir(dir_origin)
```

The output plot is as follows.




# 4 Python

The automation can handle Python files. Therefore, we can implement the automation process recursively.

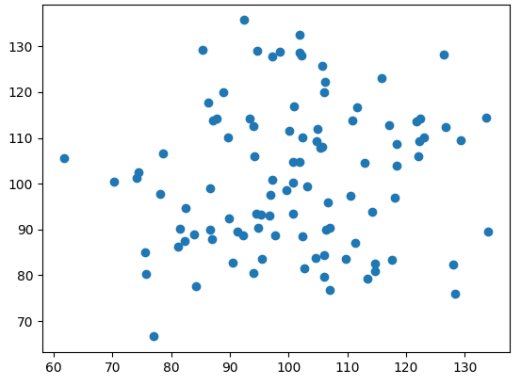
I run the following Python code.

```
# This is a sample file.

import matplotlib.pyplot as plt
import numpy as np

np.random.seed(0)
x, y = np.random.normal(loc = 100, scale = 15, size = (2, 100))

plt.scatter(x, y)
plt.savefig('sample_python_graph.png')
```

The function to run Python code, `run_python`, is as follow.

```
def run_python(fileloc):
    """Run Python script, then go back to the original directory
       """
    fileloc = "/".join([dir_origin, fileloc])
    script, dir_script = parse_location(fileloc)
    os.chdir(dir_script)
    subprocess.call([loc_python, script])
    os.chdir(dir_origin)
```

The output plot is as follows.




# 5 C++

Although C++ is less used in economic research, its speed renders it useful in computation-consuming projects.

I run the following C++ code.

```cpp
// This is a sample file.

#include <iostream>
#include <fstream>
#include <vector>
#include <random>

int main() {
    std::mt19937 generate(0);
    std::normal_distribution<> randnorm(100, 15);

    int n = 100;
    std::vector<double> x(n), y(n);
    for (int i = 0; i < n; ++i) {
        x[i] = randnorm(generate);
        y[i] = randnorm(generate);
    }

    std::ofstream output("sample_cpp_output.csv");
    output << "i,x,y" << std::endl;
    for (int i = 0; i < n; ++i) {
        output << i + 1 << "," << x[i] << "," << y[i] << std::
           endl;
    }
    output.close();

    return 0;
}
```

The function to run C++ code, `run_cpp`, is as follow. There is no need to specify the location of the application, as the function automatically uses g++, a conventional compiler for C++ programs.

```
def run_cpp(fileloc):
    """Run C++ script, then go back to the original directory"""
    fileloc = "/".join([dir_origin, fileloc])
    script, dir_script = parse_location(fileloc)
    os.chdir(dir_script)
    os.system("g++ " + script + " -o " + re.sub("(.*)\\.(.*)$", "
       \\1", script))
    os.system("./" + re.sub("(.*)\\.(.*)$", "\\1", script))
    os.chdir(dir_origin)
```

C++ is usually not used for producing graphs, and even the most used graphic library in C++ is inherited from the Matplotlib in Python. Therefore, the output files are usually data frames, with CSV being the most convenient type in most situations. The sample code outputs a CSV file.

# 6 LaTex

Finally, everything is poured into the LaTeX for final processing towards the research report. The function to run LaTeX, `run_latex`, is as follows.

```
def run_latex(fileloc, distribution = "pdflatex", num_typeset =
   2, shell_escape = False):
    """Run Tex script, then go back to the original directory"""
    # shell_escape: Enable when external tools are needed for
       compiling.
    if distribution == "pdflatex":
        loc_latex = loc_pdflatex
    else:
        loc_latex = "/Library/TeX/texbin/" + distribution
    fileloc = "/".join([dir_origin, fileloc])
    script, dir_script = parse_location(fileloc)
    os.chdir(dir_script)
    if shell_escape:
        subprocess_call_tex = [loc_latex, "-shell-escape", script
           ]
    else:
        subprocess_call_tex = [loc_latex, script]
    subprocess.call(subprocess_call_tex)
    subprocess.call([loc_bibtex, re.sub("(.*)\\.(.*)$", "\\1",
       script)])
    for i_num_typeset in range(num_typeset):
        subprocess.call(subprocess_call_tex)
    os.chdir(dir_origin)
```

Note that the function has four arguments. The second specifies the distribution to use, which is "pdflatex" by default. The third specifies the number of typesettings after loading the BibTex. The default setting is two, when the processing typesets the pdf file twice after loading the references. The fourth, a logical variable, specifies whether external tools are needed for the Tex compiling. It should be enabled (set to `True`) when the compile needs to access files outside the folder of the Tex file.

# 7 Automation

To run the whole process, just open the Python script of `rundirectory.py` and run. It automatically includes functions and pre-settings from the function script.

To change the location of the software used, open `rundirectory_function.py` and manually modify the section under "locations of the software". We can extend the terminal processing to any statistical software that supports running at the terminal.

The main script, `rundirectory.py`, is as follows.

```
# By Qichao Wang
# From source: "https://github.com/hofmanpaul/rundirectory.py"
# with modifications

# Setup

import os
import subprocess
import sys
import re
dir_origin = os.getcwd()

exec(open("rundirectory_function.py").read())

print("""
    Wait for the message 'DONE' to show up.
    The files are running in the background.
    """)

# Run

run_stata("sample_stata.do")
run_r("sample_r.r")
run_matlab("sample_matlab.m")
run_python("sample_python.py")
run_cpp("sample_cpp.cpp")
run_latex("sample_latex.tex", num_typeset = 1)

print("DONE")
```

For Mac users, click the folder that contains the main script, then click "Finder" $->$ "Services" $->$ "New Terminal at Folder". In the popped terminal, type:

```
python3 rundirectory.py
```

and return, and then the script starts to run automatically.

# References


Gentzkow, M., & Shapiro, J. M. (2014, March). Code and Data for the Social Sciences: A Practitioner's Guide. https://web.stanford.edu/~gentzkow/research/CodeAndData.pdf

Hofman, P. (2018, December). Github - hofmanpaul/rundirectory.py: A script that ties R, Latex, Stata and Python together. Handy for researchers. Retrieved December 16, 2022, from https://github.com/hofmanpaul/rundirectory.py