R news and tutorials contributed by (573) R bloggers

Welcome!

Here you will find daily **news and tutorials about R**, contributed by over 573 bloggers. There are many ways to **follow us -**

By e-mail:

On Facebook:

If you are an R blogger yourself you are invited to add your own R content feed to this site (**Non-English** R bloggers should add themselves- here)

Jobs for R-users

- Principal Data Engineer @ London
- Insights Data Analyst
- Research Assistant
- Data Scientist for facebook @ Tel AViv
- Data Scientist for ARMUS @ California

Popular Searches

- web scraping
- heatmap
- maps
- twitter
- time series

- boxplot
- animation
- shiny
- hadoop
- how to import image file to R
- ggplot2
- ggplot
- trading
- excel
- finance
- latex
- eclipse
- googlevis
- rattle
- pca
- quantmod
- Rstudio
- sql
- knitr
- market research
- regression
- tutorial
- coplot
- rcmdr
- Map

Recent Posts

- Battle Maps Using R and Leaflet
- ShinyDevCon videos now available
- Shiny JavaScript Tutorials
- R Tools for Visual Studio 3.0 now available
- Announcing new forecastHybrid package
- RcppArmadillo 0.6.700.6.0
- Student-Run Conference in Data Science
- a Simpson paradox of sorts
- RZabbix – easy and direct communication with 'Zabbix API'
- vtreat cross frames
- R 3.3.0 now available
- Manipulate(d) Regression!
- BioC 3.3: NEWS of my packages
- Need user feedback? Send it directly from R
- Classify gender based on danish first names

Other sites

- Jobs for R-users
- SAS blogs
- Statistics of Israel

Integrating Python and R Part II – Executing R from Python and Vice Versa

October 26, 2015
By Mango Blogger

(This article was first published on **Mango Solutions**, and kindly contributed to R-bloggers)

By Chris Musselle

In a previous article we went over why you might want to integrate both R and Python into a single pipeline, and how to do so via the use of a flat file air-gap. In doing so we covered how to run a Python or R script from the command line, and how to access any additional arguments that are parsed in. In this post we complete the integration process by showing how the two scripts can be linked together by getting R to call Python and vice versa.

Command Line Execution and Executing Subprocesses

To better understand what's happening when a subprocess is executed, it is worth revisiting in more detail what happens when a Python or R process is executed on the command line. When the following command is run, a new Python process is started to execute the script.

```
python path/to/myscript.py arg1 arg2 arg3
```

During executing, any outputs that are printed to the standard output and standard error streams are displayed back to the console. The most common way this is achieved is via a built in function (`print()` in Python and `cat()` or `print()` in R), which writes a given string to the `stdout` stream. The Python process is then closed once the script has finished executing.

Running command line scripts in this fashion is useful, but can become tedious and error prone if there are a number of sequential but separate scripts that you wish to execute this way. However it is possible for a Python or R process to execute another directly in a similar way to the above command line approach. This is beneficial as it allows, say a parent Python process to fire up a child R process to run a specific script for the analysis. The outputs of this child R process can then be passed back to the parent Python process once the R script is complete, instead of being printed to the console. Using this approach removes the need to manually execute steps individually on the command line.

Examples

To illustrate the execution of one process by another we are going to use two simple examples: one where Python calls R, and one where R calls Python. The analysis performed in each case is trivial on purpose so as to focus on the machinery around how this is achieved.

Sample R Script

Our simple example R script is going to take in a sequence of numbers from the command line and return the maximum.

```
# max.R
# Fetch command line arguments
myArgs <- commandArgs(trailingOnly = TRUE)

# Convert to numerics
nums = as.numeric(myArgs)

# cat will write the result to the stdout stream
cat(max(nums))
```

Executing an R Script from Python

To execute this from Python we make use of the subprocess module, which is part of the standard library. We will be using the function, `check_output` to call the R script, which executes a command and stores the output of stdout.

To execute the `max.R` script in R from Python, you first have to build up the command to be executed. This takes a similar format to the command line statement we saw in part I of this blog post series, and in Python terms is represented as a list of strings, whose elements correspond to the following:

```
['<command_to_run>', '<path_to_script>', 'arg1' , 'arg2', 'arg3', 'arg4']
```

An example of executing an R script form Python is given in the following code.

```
# run_max.py
import subprocess

# Define command and arguments
command = 'Rscript'
path2script = 'path/to your script/max.R'

# Variable number of args in a list
args = ['11', '3', '9', '42']

# Build subprocess command
cmd = [command, path2script] + args

# check_output will run the command and store to result
x = subprocess.check_output(cmd, universal_newlines=True)

print('The maximum of the numbers is:', x)
```

The argument `universal_newlines=True` tells Python to interpret the returned output as a text string and handle both Windows and Linux newline characters. If it is omitted, the output is returned as a byte string and must be decoded to text by calling `x.decode()` before any further string manipulation can be performed.

Sample Python Script

For our simple Python script, we will split a given string (first argument) into multiple substrings based on a supplied substring pattern (second argument). The result is then printed to the console one substring per line.

```
# splitstr.py
import sys

# Get the arguments passed in
string = sys.argv[1]
pattern = sys.argv[2]

# Perform the splitting
ans = string.split(pattern)

# Join the resulting list of elements into a single newline
# delimited string and print
print('n'.join(ans))
```

Executing a Python Script from R

When executing subprocess with R, it is recommended to use R's system2 function to execute and capture the output. This is because the inbuilt system function is trickier to use and is not cross-platform compatible.

Building up the command to be executed is similar to the above Python example, however `system2` expects the command to be parsed separately from its arguments. In addition the first of these arguments must always be the path to the script being executed.

One final complication can arise from dealing with spaces in the path name to the R script. The simplest method to solve this issue is to double quote the whole path name and then encapsulate this string with single quotes so that R preserves the double quotes in the argument itself.

An example of executing a Python script from R is given in the following code.

```
# run_splitstr.R
command = "python"

# Note the single + double quotes in the string (needed if paths have spaces)
path2script='"path/to your script/splitstr.py"'

# Build up args in a vector
string = "3523462---12413415---4577678---7967956---5456439"
pattern = "---"
args = c(string, pattern)

# Add path to script as first arg
allArgs = c(path2script, args)

output = system2(command, args=allArgs, stdout=TRUE)

print(paste("The Substrings are:n", output))
```

To capture the standard output in a character vector (one line per element), `stdout=TRUE` must be specified in `system2`, else just the exit

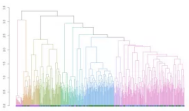
status is returned. When `stdout=TRUE` the exit status is stored in an attribute called "status".

Summary

It is possible to integrate Python and R into a single application via the use of subprocess calls. These allow one parent process to call another as a child process, and capture any output that is printed to stdout. In this post we have gone through examples of using this approach to get an R script to call Python and vice versa.

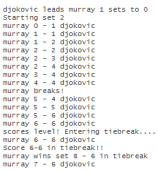
In a future upcoming article will draw on the material of this post and part I, to show a real world example of using Python and R together in an application.

Related



Integrating Python and R Part III: An Extended Example
In "R bloggers"

Integrating Python and R into a Data Analysis Pipeline – Part 1
For a conference in the R language, EARL London 2015 saw a surprising number of discussions about
In "R bloggers"



Fun simulating Wimbledon in R and Python
In "R bloggers"

Comments are closed.

Recent popular posts

- How to write the first for loop in R
- R tutorials
- R – GPU Programming for All with 'gpuR'
- ShinyDevCon videos now available

Most visited articles of the week

1. How to write the first for loop in R
2. Installing R packages
3. R 3.3.0 is released!
4. R tutorials
5. How to perform a Logistic Regression in R
6. In-depth introduction to machine learning in 15 hours of expert videos
7. Using apply, sapply, lapply in R
8. Cross-Validation for Predictive Analytics Using R
9. Computing and visualizing PCA in R

Sponsors

Jobs for R users

- Principal Data Engineer @ London
- Insights Data Analyst
- Research Assistant