Walking the Python-R Bridge

Anuja Kelkar Follow
Mar 7, 2017 · 6 min read

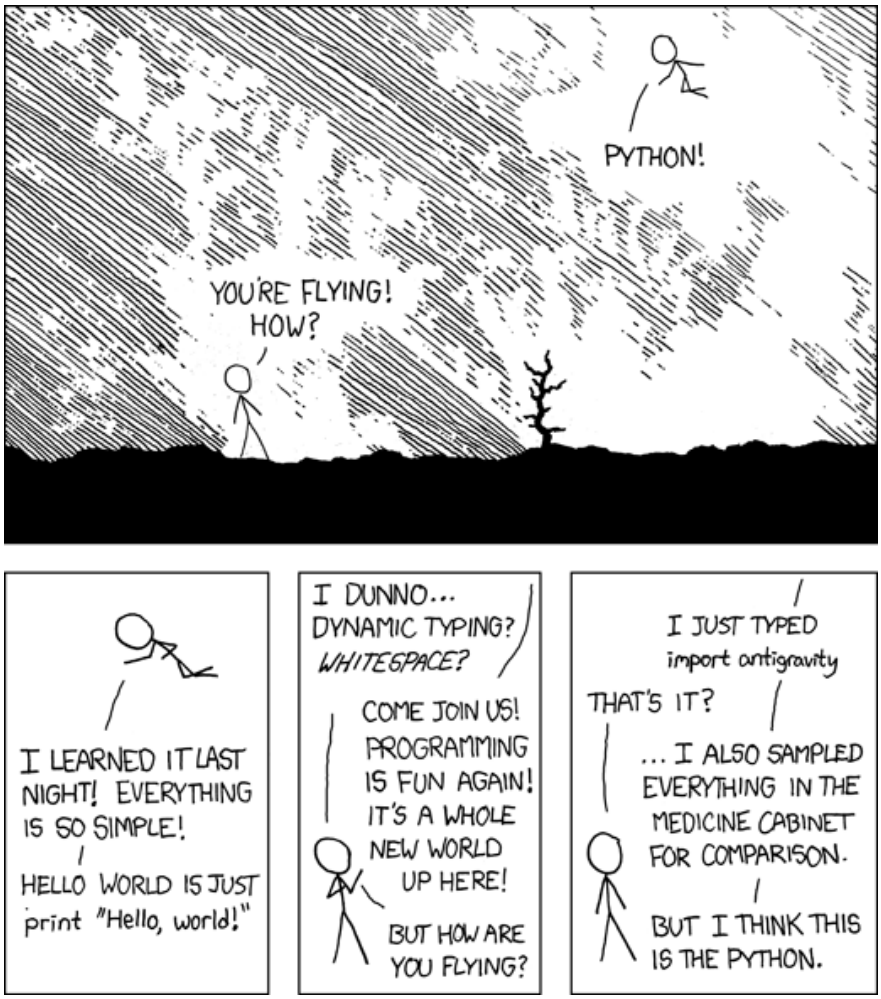

Image source: xkcd.com

For those who want to use the best of Python and R together!

Background

For one of the projects at work, in the Data Science team, we have most of our analytics capabilities and scripts written in R. We wrote an R package for our analytics use case. We use AWS Step functions for coordination of the different services and components in our project. We have a series of processing functions applied to data ingested via a

custom upload tool. These functions include partitioning the file, inserting partitions into a database and running custom and compute-intensive analyses on the normalized data in the database. Having this kind of an architecture in mind, we wrote AWS Lambda functions that call out to our R package functions (running in a docker container) and run analyses on the data ingested. However, at present, AWS Lambda does not support R as an executing environment. Hence, we needed to talk to R from Python.

For most data scientists and data engineers who use Python and R in their day-to-day lives and would like to use complementing capabilities (maybe, use some R plotting and stats packages in Python) of these two languages together, this post will help you understand how to wield the R sword in a Python battle.

Which Python module should I use?

There were a number of Python module choices to do this: rpy2, pyRserve and PypeR. My requirement from the module was that of providing me with a high level interface that could allow me to call functions from R packages. After reading up on all three, rpy2 seemed like the better of the lot, owing to multiple reasons.

One of the important factors was it plays nicely with numpy objects in Python. Also, it has a more Pythonic style in using R. I will outline the limitations I found in the module towards the end.

rpy2 Setup/Installation

rpy2 installation is very easy with pip.

```
pip install rpy2
```

However, to run rpy2, you must have **R version >3.2 and Python version 2.7 or >3.3** installed. Please upgrade to newer version of R/Python as required, to match the recommendations, before installing rpy2.

Working with rpy2

rpy2 runs embedded R in a Python process. rpy2 module provides two interfaces: a low-level interface (***rpy2.rinterface***) and a high-level interface (***rpy2.robjects***). I used the high-level interface, rpy2.robjects and I will only be talking about rpy2.robjects interface in this article.

I have used rpy2 in my project at work, for calling our R analytics package functions in the analytics services (Python services) that run as steps of an AWS state machine, which once created, handles the coordination of the services and components and lets me design the micro-services to handle analysis tasks.

The snippets of code I will use in this article are similar to those I used in my code at work, in spirit and meaning. However, I have changed variable/package/function names to conceal the domain of the project work.

The following is a small snippet of code that demonstrates how to use rpy2.robjects interface.

The code assumes that data has been extracted from the data source into a pandas dataframe already.

```
import rpy2.robjects as ro
from rpy2.robjects.packages import importr
import pandas.rpy.common as com


#not including the lines of code that create the pandas
dataframe #data_pd_df
#convert pandas dataframe to R dataframe
data_r_df = com.convert_to_r_dataframe(data_pd_df)
```

```
#importing R package custom.analytics and replacing . with _
in #package names to ensure no conflicts
d = {'package.dependencies': 'package_dot_dependencies',
     'package_dependencies': 'package_uscore_dependencies'}
custom_analytics = importr('custom.analytics',
                   robject_translations = d)
```

```
#calling functions from our R package custom.analytics in
Python to #generate results
stage1_processed_data =
custom_analytics.process_filetype1(data_r_df)
summary_stats =
custom_analytics.summarize_filetype1(stage1_processed_data)
```

```
#similarly, you can import any other R package, for example,
'base' #package in R
base = importr('base', robject_translations=d)
```

```
#saving R object, summary_stats to file
base.saveRDS(summary_stats, file="/tmp/summary_stats.rds")
```

Variable naming conventions and styles in R and Python are very different. in R, a variable with multiple words is usually period-separated, i.e. read.table vs a Pythonic read_table. An important thing to note is that function *importr* (used to load R packages when required) from *rpy2.robjects.packages,* has an optional argument that allows you to check that the translation to Python does not mask any symbols in the R package. For example, in our R package ***custom.analytics,*** we have functions named *process.filetype1() and summarize.filetype1(),* which follow the period-separated style of naming. This is handled by the *robject_translations* argument to the *importr* function. You pass it a *dict* of your translations and you are done!

You can create R dataframes using rpy2 as follows:

```
import rpy2.robjects as ro

#c_data_pd_df is a pandas dataframe
#with name and characteristics columns in addition to others
name_R_obj = ro.StrVector(tuple(c_data_pd_df['name']))
char_R_obj =
ro.StrVector(tuple(c_data_pd_df['characteristics']))
```

```
#creating R dataframe from the R objects(StrVectors)
c_data_R_df = ro.DataFrame({'name': name_R_obj,
                'characteristics': char_R_obj})
```

You can create R objects (StrVector, or any other Vectors) from lists/tuples. In my case, I had to extract the columns from pandas data frame to create those R vectors.

One issue I came across when I ran the analytics worker was that of the fact that, annoyingly, R data frame constructor converts character

vectors to factors, by default. R users, you know what I am talking about! In addition, the R error trace might not directly point to the actual error, and it makes it even more difficult to debug this issue later on. So, please be cognizant of this subtle but important factor. More on factors here, for those curious.

To successfully run my analytics worker, I had to make sure that my character vectors stay character vectors and are not converted to factors. The *rpy2.robjects* interface function does not expose the argument *stringsAsFactors* that is used in the R version to handle coercion of strings to factors. I was happy to find that this is an issue reported to rpy2 developers in January 2017, and a feature to expose this variable in robjects interface DataFrame constructor will be released soon. Till this feature comes out, one must resort to handling this in the R code itself (if it is your own R package and you can do it easily) or handle it using the *r* class in *rpy2.robjects,* which is an entry point to the embedded R process. The following snippet shows and example of how to avoid the string to factor coercion in data frames by sending the command directly to the R embedded process.

```
import rpy2.robjects as ro
```

```
#preserving character vectors by setting stringsAsFactors to
False
ro.r('df<- data.frame(x=c(1,2,3,4,5),
y=c("a","b","c","d","e")
         , stringsAsFactors=F)')
```

Limitations

- The rpy2 module is well written. However, I found the documentation to be scarce.
- The high-level interface, *robjects* is very convenient to use if you are getting something quick and easy done. However, if you are going to walk the Python-to-R bridge multiple times with large objects in your code, then performance will take a hit. In such cases, it might be wiser to use the low-level interface (I am already contemplating using the low-level interface) as you will be closer to the R's C-level interface and you will be able to run your code faster.

- Also, you need to know both R and Python to be able to debug any issues/errors. rpy2 does not help if you know Python only and are looking for a Pythonic interface that helps you to talk to R, without delving into R.

My intention with this post is to help out those who need to talk to R from Python code, to get things working easily, without much pain from the scarce documentation of rpy2.

R has a rich community and ecosystem of powerful packages across different fields and disciplines. The visualization capabilities of R are pretty awesome and you can use rpy2 as a bridge between Python and R to get the power of R's stats and visualization with Python's ease of readability and ease of integration into web apps and cloud stacks!

> *If you like this post, don't forget to recommend and share it.*