Python vs. R

Kyle Cho

10/31/2017

Introduction

As someone deeply interested in the field of computer science and programming, I have come across a variety of languages - Java, C, Go, PHP, JavaScript, you name it. When it comes to data analysis and building simple softwares, I would say that Python is the strongest tool in my possession. Not only do I love its syntax and minimality, but the vast arsenal of libraries is also one of the language's biggest selling points. With libraries like *NumPy* and *pandas*, Python is probably the go-to solution for data analysis for thousands of programmers out there, especially for those who feel like R is a bit outdated.

Coming from this background, I have found that learning R from scratch has been a quite meaningful experience. Just like any other language, R has pros and cons and I will compare them to Python and discuss what aspects of R I like (or dislike).

`import` and `library`

In R, to use an external library or package - ggplot2 for example - you do so by calling the following function:

```
library(ggplot2)
```

Then you can use all the features that the library provides. This is different for Python. You would use the following to import an imaginary ggplot2 package:

```
from ggplot2 import *
```

This means from ggplot2 package import everything. You might think this is rather verbose compared to the R version, but there is a reason for keeping this syntax. Imagine you import ggplot2, then declare a function `ggplot()`.

```
library(ggplot2) # import ggplot2 package
ggplot <- function(n) { ## declare function
  print(n)
}
```

This causes a collision, and to prevent this you would have to look up all the functions in a package. In Python, you can only import the functions you actually need and give them aliases.

```
from ggplot2 import ggplot as thisisalias
```

In my opinion for this part Python is the clear winner, since it is easier to track which external functions are present in the script I am working on.

Two kinds of `list`

Almost every programming language features a data structure that contains a certain number of elements and is usually called an array. This is called a list in both Python and R. However, the way Python's list works is drastically different from that of R. Note that I am comparing Python's list to R's list, not the vector - a Python list can have different types of elements.

One thing in common is that they both use bracket notation to extract values, but the R's way of indexing elements are not like any other mainstream language out there, of course including Python - the first element of a R list is located at index 1.

```
lst <- list("a", "b", "c")
print(lst[1]) # print element at index 1
```

```
## [[1]]
## [1] "a"
```

To achieve the same result for Python, you should look at index 0.

```
lst = ["a", "b", "c"]
print(lst[0])
"a"
```

This confuses people who are used to other languages, but I think for a novice R's indexing makes much more sense.

Another cool thing about R's list is that you could 'name' an element in a list.

```
john <- list(name="John", age=20) # list where name is John and age is 20
print(john$name) # print the name
```

```
## [1] "John"
```

To do this in Python you need to use a different data structure called dictionary.

```
john = {'name': "John", 'age': 20}
print(john['name'])
"John"
```

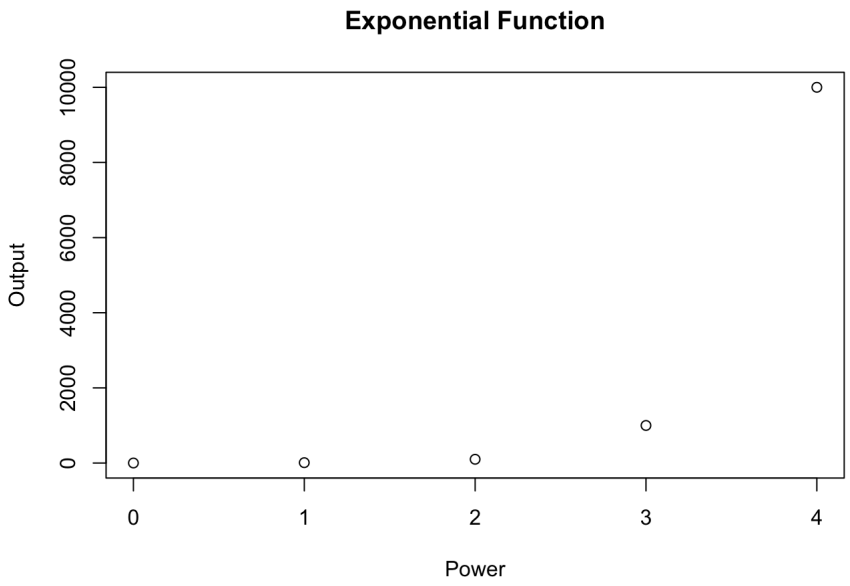
For some people R's list might feel like a swiss army knife - it can do a lot of things but looks complicated. I think this is just a matter of preference and don't think one is objectively better than the other, but I prefer the Python system solely because I like to keep things separated.

Graph

R really shines when it comes to plotting graphs. As far as I know none of the major programming languages come with useful graphing tools included. While some people find R's default plot function somewhat lackluster and use external libraries like *ggplot2*, I think the fact that R comes with a plotting tool is a great selling point, as you don't have to search the Internet for which library to download.

Let's say we want to plot an exponential function, `10^x`. We can do this easily in R:

```
xs = c(0, 1, 2, 3, 4) # vector for x axis
ys = c(1, 10, 100, 1000, 10000) # vector for y axis
plot(xs, ys, xlab='Power', ylab='Output', main='Exponential Function') # plots graph
```



In python it is necessary to first install a package like `matploblib` and import it. What's worse is that to install a package you need to use a package manager called `pip`. If this doesn't work then you have to download an .exe file.

```
import matplotlib.pyplot as plt
plt.plot([0,1,2,3,4], [1,10,100,1000,10000])
plt.show()
```

Table

Just like graphing, this is a really nice feature that R provides and no other language does by deafult. To create a table - or a data frame in R - you just do the following:

```
name <- c("Alice", "Bob", "Eve") # name vector
age <- c(20, 21, 22) # age vector
major <- c("Statistics", "Computer Science", "Chemistry") # major vector
table <- data.frame(name, age, major) # combine them to create a data frame
print(table) # print the content of the data frame
```

```
##    name age            major
## 1 Alice  20       Statistics
## 2   Bob  21 Computer Science
## 3   Eve  22        Chemistry
```

And you can access each column using the dollar notation.

```
print(mean(table$age)) # print the average of the age column
```

```
## [1] 21
```

To do this in Python you need to first install a package, and as I mentioned above this could be really stressful if things don't work for some reason. In addition, since there are so many libraries out there it could be hard to choose which one to use. Here's an example using Berkeley's Data 8 library.

```
from datascience import Table
import numpy as np
table = Table().with_columns(
  'name', make_array('Alice', 'Bob', 'Eve'),
  'age', make_array(20, 21, 22),
  'major', make_array('Statistics', 'Computer Science', 'Chemistry')
)
print(np.mean(table.column('major')))
21
```

Conclusion

I do prefer Python's minimalistic syntax and I am not a fan of R's there-is-more-than-one-way-of-doing-it approach, especailly when it comes to subsetting elements in a data structure. In addition, the difference between `typeof()` and `mode()` could be difficult to understand for beginners who are used to Python's dynamic typing. However, R's 'batteries included' philosophy regarding graphs and data frames is hard to turn down for a quick and easy way of analyzing data, as you would have to install a bunch of dependencies for Python and other languages to do this. My message would be if you want cleaner syntax and more versatility in addition to simple data wrangling at the cost of managing external packages, Python would be a good choice. If you don't want to install too many libraries and just need a basic tool with a built-in plotting function and data frame manimpulation to analyze your data and nothing more, R will suit your purpose.

References

- https://matplotlib.org/users/pyplot_tutorial.html
- https://www.tutorialspoint.com/python/python_lists.htm
- https://www.tutorialspoint.com/python/python_dictionary.htm
- https://www.programcreek.com/2014/01/vector-array-list-and-data-frame-in-r/
- http://data8.org/datascience/tutorial.html#creating-a-table
- https://www.python.org/dev/peps/pep-0221/
- https://www.programiz.com/r-programming/plot-function