Python vs R

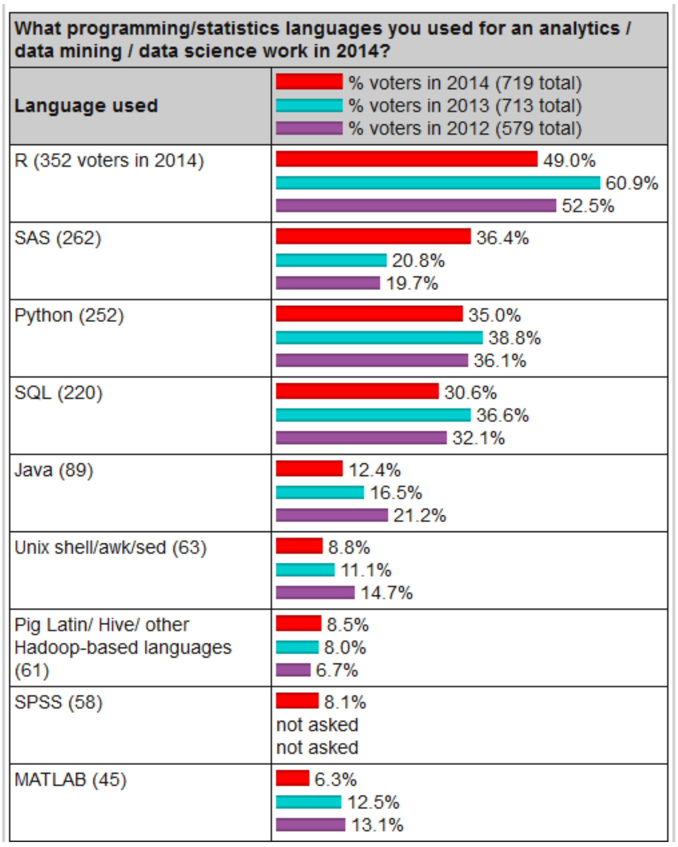
George Khalilieh

10/26/2017

Introduction

In the data science world, many languages and tools are used. Some use R, Python, SAS, MATLAB, SPSS, My SQL and Java.

Take a look below at a survey of languages used, from the above link.



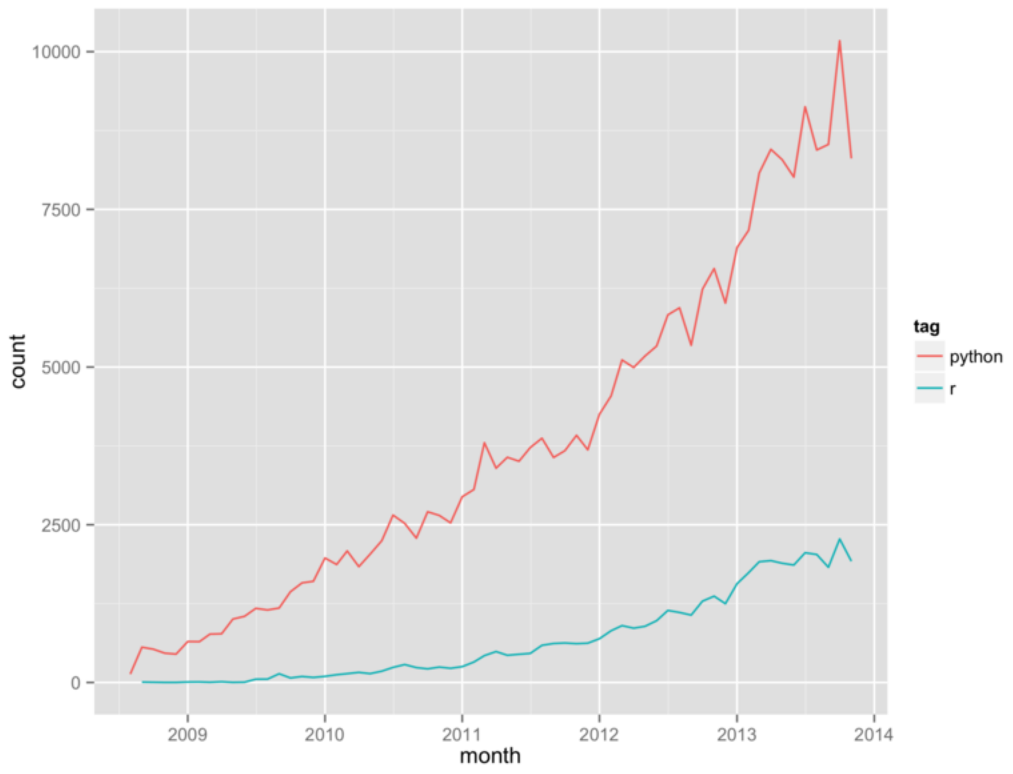
What programming/statistics languages you used for an analytics / data mining / data science work in 2014?

Language used	🟥 % voters in 2014 (719 total)	🟦 % voters in 2013 (713 total)	🟪 % voters in 2012 (579 total)
R (352 voters in 2014)	49.0%	60.9%	52.5%
SAS (262)	36.4%	20.8%	19.7%
Python (252)	35.0%	38.8%	36.1%
SQL (220)	30.6%	36.6%	32.1%
Java (89)	12.4%	16.5%	21.2%
Unix shell/awk/sed (63)	8.8%	11.1%	14.7%
Pig Latin/ Hive/ other Hadoop-based languages (61)	8.5%	8.0%	6.7%
SPSS (58)	8.1%	not asked	not asked
MATLAB (45)	6.3%	12.5%	13.1%

Survey Results

SAS is closed source, so of the open source languages, R and Python were the two results in the survey. As the popularity of data science and analytics increases, interest in R and Python skills has increased greatly over the past few years. Another look at the popular data science programming languages here. The internet loves to compare Python versus R, as shown here and here.

Here's a comparison of the number of StackOverflow open questions asked per month about each language, from here.



Motivation + Vectors vs Lists

I have prior experience in Python, so I will outline some of my thoughts on the differences between the two languages. They are both interpreted languages that can run in a console. They both can read in data and compute new informations based on input data. However, they are quite different in fundamental ways. In R, vectors are the bread and butter of the language. Computations are done using vectors of characters, boolean logical values, integers, and floating point numbers for example. In native Python, values are stored in lists, which are collections of items that do not need to all be the same type.

```
#defining a mixed type vector below, containing numeric and character data
example_vector <- c(1, 3, 4, 5, "R")
print(example_vector)
```

```
## [1] "1" "3" "4" "5" "R"
```

Above, I defined a vector of numbers and one character value. When mixed data types are used in making a vector, R coerces everything into one type. In this case, the character type.

In Python, you can mix data types when making a list.

```
#the same as above, in a python list type
example_list = [1, 3, 4, 5, "R"]
print(example_list)
```

```
## [1, 3, 4, 5, 'R']
```

The types of each list element do not change in the presence of other data types.

However, the catch is that you cannot natively apply bulk transformations on lists as a whole.

```
num_list = [1, 4, 5, 8, 24, 43]
print(num_list + 2)
```

```
## Traceback (most recent call last):
##   File "<string>", line 2, in <module>
## TypeError: can only concatenate list (not "int") to list
```

What happened there? Python returned an error because you cannot add numbers to list objects, even though all the elements are numeric.

In R, this is not a problem at all.

```
num_vector <- c(1, 4, 5, 8, 24, 43)
print(num_vector + 2)
```

```
## [1]  3  6  7 10 26 45
```

To accomplish this in Python, it looks like this. In R, there are multiple approaches of doing a list comprehension like above described here.

```
num_list = [1, 4, 5, 8, 24, 43]
print([i + 2 for i in num_list])
```

```
## [3, 6, 7, 10, 26, 45]
```

R Lists vs Python Dictionaries

Another difference is that in Python we also have a data structure called a dictionary. It lets you store values that vary according to an input that the user provides.

```
colors = {"red": "rojo", "white":"blanco", "green":"verde"}
print("The color red in Spanish is " + colors["red"] + '.')
print("The color white in Spanish is " + colors["white"] + '.')
```

```
## The color red in Spanish is rojo.
## The color white in Spanish is blanco.
```

In R, the closest thing we have is a list. Lists in R are a lot like dictionaries in Python, and are like python lists.

```
colors = list("red" = "rojo", "white" = "blanco")
print(paste0("The color red in Spanish is ",colors['red'], "."))
```

```
## [1] "The color red in Spanish is rojo."
```

```
print(paste0("The color white in Spanish is ",colors['white'], "."))
```

```
## [1] "The color white in Spanish is blanco."
```

Lists in R can also act like Python lists.

```
like_python <- list(1, 3, 4, 5, "dog", "cat")
print(like_python)
```

```
## [[1]]
## [1] 1
## 
## [[2]]
## [1] 3
## 
## [[3]]
## [1] 4
## 
## [[4]]
## [1] 5
## 
## [[5]]
## [1] "dog"
## 
## [[6]]
## [1] "cat"
```

Lists display very differently, though. They show up as indices and show the indices of the element at that position.

Conditionals

Now that we have established some basic data types and data structures, let's look at how conditional logic and control flow is handled in both languages. Formal primer on conditions for Python here A similar resource for R is here

The logic that you will end up using in Python or R is ultimately the same, so the difference is purely syntactical.

The `if` statement in R is structured as follows:

```
# The condition is something that you test, such as equality, difference, or comparison of objects to each other.
condition <- 100 > 3 ^ 3
if (condition == TRUE) {
  print("The condition is true.")
}else {
  print("The condition is false.")
}
```

```
## [1] "The condition is true."
```

The Pythonic version of the above reads similarly, minus the `{}` symbols, and using `:` instead.

```
## exponentiation is done with ** in python
condition = 100 > 3 ** 3

if condition == True:
    print("The condition is true.")
else:
    print("The condition is false.")
```

```
## The condition is true.
```

Why no curly braces? Because in Python, indentation works the same as the curly braces do in R. Lines on the same indentation level are run together, similarly to how expressions are grouped with curly braces in R.

Iteration

In both languages it is very easy to loop through objects and perform operations using their contents. In R, a for loop looks like this.

```
example_sequence <- 1:5
for (i in example_sequence) {
  print(i + 1)
}
```

```
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
```

```
example_sequence = list(range(1, 5 + 1))
for i in example_sequence:
    print(i + 1)
```

```
## 2
## 3
## 4
## 5
## 6
```

The body of the for loops is actually identical in the case of the above. The syntax for modifying the contents of a list or vector is the same as well.

```
to_modify <- numeric(10)
fun_sequence <- seq(from = 5, to = 50, by = 5)

for (i in 1:10) {
  to_modify[i] <- i + fun_sequence[i]
}
print(to_modify)
```

```
##  [1]  6 12 18 24 30 36 42 48 54 60
```

```
to_modify = [0] * 10
fun_sequence = range(5, 50 + 1, 5)
for i in range(10):
    # i added 1 below because python indexes starting at 0 and R starts at 1
    to_modify[i] = i + fun_sequence[i] + 1

print(to_modify)
```

```
## [6, 12, 18, 24, 30, 36, 42, 48, 54, 60]
```

You can also do while loops in both languages.

```
counter <- 1
while (counter <= 5) {
  print(sin(counter * pi / 2) + 1)
  counter <- counter + 1
}
```

```
## [1] 2
## [1] 1
## [1] 0
## [1] 1
## [1] 2
```

```
import math
counter = 1
while counter <= 5:
    print(math.sin(counter * math.pi / 2))
    counter += 1
```

```
## 1.0
## 1.22464679915e-16
## -1.0
## -2.44929359829e-16
## 1.0
```

The major difference here, again, is that the curly braces denote what the indendation in python denotes, the code expressions and the order in which they are evaluated in groups and in compound expression statements. Having considered all the above, are the two languages really all the different next to each other?

Functions

A pretty big difference between the two is how functions are defined.

```
add_ten <- function(x) {
  x + 10
}
add_ten(1)
```

```
## [1] 11
```

```
def add_ten(x):
    return x + 10
print(add_ten(1))
```

```
## 11
```

In Python, you can also define shorter functions called lambda functions.

```
add_lambda = lambda x: x + 10
print(add_lambda(1))
```

```
## 11
```

To me, it just seems like the curly braces are the main syntactic difference between the two languages. You use colons in Python at the end of `if` or `while` statements and function definitions. Notice the general fact that the functions take in parameters and return a result. In R, the last line of a function is what is returned. You can also specify `return` to return a specific output.

Final Thoughts and Take Home Message

Python vs R is always brewing on the internet, but there is no clear winner for everybody. For statisticians, R usually wins. For a software engineer, Python wins. By and large almost any task or feature you would like in a programming language has been covered in some form or another with a library or reading into the documentation. I like Python. I also like R. I think both are awesome in their own ways, and I have clear use cases for each of them, so I am not forced to just use one or the other. I have shown that they are very similar for the most part if you know where to look for what you need. Programming languages at their heart are just implementations of logical thinking. I have had a fantastic experience using both, and am ecstatic that I am gaining skills in both of these programming languages. Personally, I was very much in favor of Python at the beginning of this course, but now I lean towards R for working with Data Science and Analysis. I find that it is better designed to handle analysis, whereas Python is better designed to scale into bigger solutions or handle more of the nuts and bolts of a project or tool. I believe that every Data Scientist should know both languages, because it only helps to understand the languages, and enables you to do more types of work than simply knowing one or the other.

References

- http://bigdata-madesimple.com/how-i-chose-the-right-programming-language-for-data-science/
- http://www.analyticbridge.datasciencecentral.com/profiles/blogs/blog-r-vs-python-which-one-has-higher-demand-on-the-job-market-a
- https://dzone.com/articles/which-are-the-popular-languages-for-data-science
- https://www.dataquest.io/blog/python-vs-r/
- https://www.kdnuggets.com/2015/05/r-vs-python-data-science.html
- http://blog.revolutionanalytics.com/2013/12/r-and-python.html
- http://apps.fishandwhistle.net/archives/1010
- http://www.openbookproject.net/books/bpp4awd/ch04.html
- https://www.programiz.com/r-programming/if-else-statement