Intro to Data Science:

Learning about Python (and retroactively, R)

Ed Podojil Data Engineer/Scientist, Animoto

Agenda

I. Python and R History

II. Comparing Python and R

III. Exercise: Python Workshopping

IV. Exercise: Experimenting with Scikit-Learn

I. Python and R: History

Goals

- Learn about the histories of each Python and R

S: A history lesson

S (later S-Plus) developed at Bell Labs by John Chambers, 1975

Previously: Statisticians used FORTRAN subroutines for their work

Goal: develop a more interactive statistical language

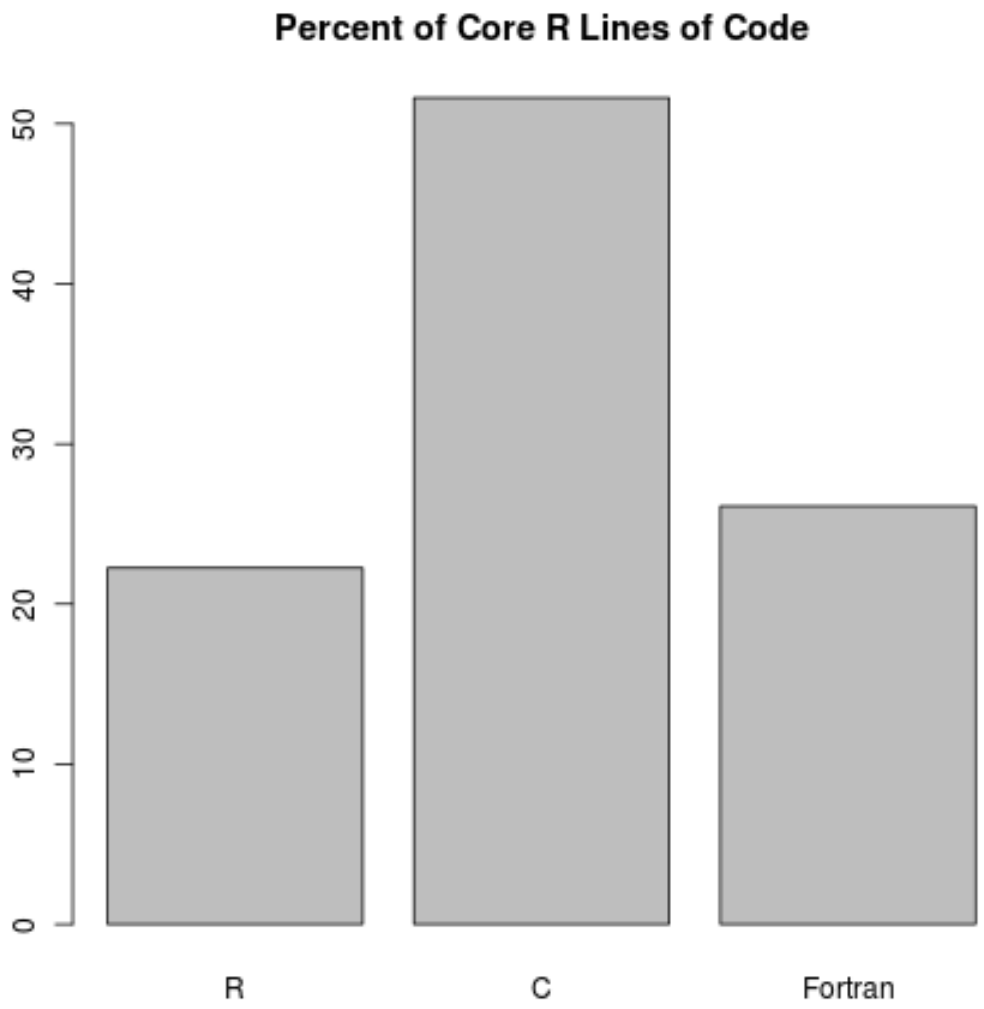
R vs S

S-PLUS is proprietary (owned and sold by Tibco)

R developed by Ross Ihaka and Robert Gentleman

Freely availabled by GNU Public License!

Primarily written in C, FORTRAN, and R


Percent of Core R Lines of Code
0
10
20
30
40
50
R
C
Fortran

Python: History

Written and implemented by Guido van Rossum in 1989

Designed around code readability

The Zen of Python

```
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

Python is a multiparadigm programming language

Imperitive Programming

Tell the computer what to do in some kind of sequence

Object Oriented Programming

Everything is an object; objects have fields

Imperitive Programming Example

```
print 'writing publisher counts to file...'
with open(output_file, 'w') as f:
    for k, v in pubs_counter.iteritems();
    try:
        f.write('{0}, {1}\n'.format(k, v))
    except Exception as details:
        print 'error: {0} -- {1}'.format(details, (k, v))
        continue
```

OOP Example

```
class MRWordCount(MRJob):

    def mapper(self, _, line):
        # self.set_status('mapper')
        self.increment_counter('mapper_group', 'items_mapped', 1)
        for word in line.split();
            yield word, 1

    def reducer(self, word, counts):
        # self.set_status('reducer')
        self.increment_counter('reducer_group', 'items_reduced', 1)
        yield word, sum(counts)
```

II. Comparing R and Python (data structures)

Goals

- Review: What are the data structures in R?
- What data structures exist in Python?

R: Numeric structures

```
## Generally, R does well working
## between and converting numbers
## as needed
class(1)
[1] "numeric"
class(as.integer(1))
[1] "integer"
```

Python

```
# Python has four basic numeric types
>>> type(1)
<type 'int'>
>>> type(2.5)
<type 'float'>
>>> type(True)
<type 'bool'>
>>> type(2+3j)
<type 'complex'>
```

R: Arrays (Vectors)

```r
## Arrays are an ordered collection
## of elements. R Vectors can only
## maintain one data structure
# Returns 'character'
str(c(1, 2, 3, 'a', 'asldj', TRUE))
# Returns 'numeric'
str(c(1, 2, 3, 4, 10, 1233))
# Returns 'numeric',
# TRUE converted to 1
# FALSE converted to 0
str(c(1, 2, 3, TRUE))
```

Python: Arrays (lists)

```python
# Lists in Python maintain
# their original data type
>>> k = [1, 'b', True]
>>> k
[1, 'b', True]
>>> type(k)
<type 'list'>
```

R: Lists

```
## in R, combining vectors
## generates one longer vector.
a <- c(1, 2, 3)
b <- c(4, 5, 6)
c <- c(a, b)
c
[1] 1 2 3 4 5 6
## use lists to maintain
## vector relationships
d <- list(a, b)
d
[[1]]
[1] 1 2 3

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

Python: Lists of Lists

```
# Lists in Python maintian original
# data type, including other lists
>>> k = [1, 'b', True, ['new', 'list',
  'here']]
>>> k
[1, 'b', True, ['new', 'list', 'here']]
>>> k[3]
['new', 'list', 'here']
```

Python: mutable vs immutable

```
# Tuples are like lists, but immutable (they can't be changed)
>>> k = (1, 'b', True)
>>> k[0]
1
>>> k[0] = 'a'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

Hashes: key/value objects

Not (well) supported in R, very useful in Python (and other scripting languages, like Ruby or JavaScript)

```
email = {'title': 'Good Morning!', 'from':'Bob Loblaw', 'date': 'Tue Mar 3 2013'}
email
{'date': 'Tue Mar 3 2013', 'from': 'Bob Loblaw', 'title': 'Good Morning!'}
email['title']
'Good Morning!'
```

Class break

III. Python Workshopping

Goals:

- Work through various flow control systems
- Refer to lesson_08.py for Python code

III. Python Wrap up

Goals:

- What are some advantages and disadvantages to working with Python?
- What are some different things people have done with Python?

Popularity

Python's popularity comes from the strength of its design.

The syntax looks like pseudocode, and it is explicitly meant to be clear, compact, and easy to read.

Python is an expressive language.

Versatility

Python is also an extremely versatile language, and it attracts fans from many different walks of life:

Django: Web Development

PANDAS: Data Analysis

Fabric: Systems Administration

Standard Library

Another great strength is the Python Standard Library.

This is a collection of packages that ships with the standard Python distribution, and “…covers everything from asynchronous processing to zip files”.

The advantages of the PSL are usually described by saying that Python comes with batteries included.

Weaknesses

Slower than lower level languages (such as FORTRAN or C)

Dynamic Typing (variable types are declared by whatever is passed, not by pre assignment)

Rather specific code format (though I digress...)

Group Discussion

Questions to ponder about:

How are Python and R different? How are they the same?

Where do you think Python would be more useful, or less useful?

Some Python today referred to "map" and "reduce." What do you think these mean in programming?