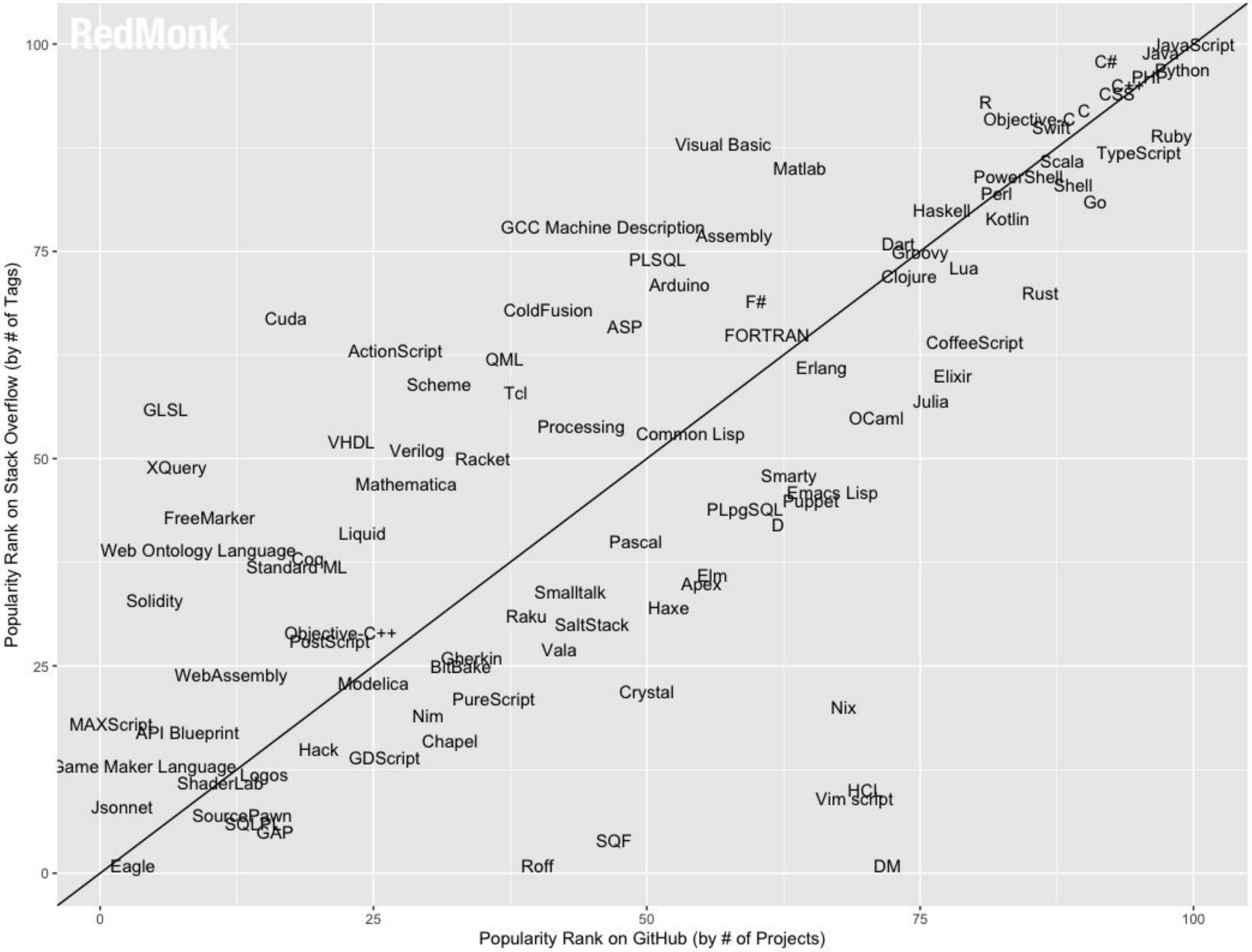
Introduction to python - along with the implementation in R

Asst. Prof. Ashwini Mathur



RedMonk Q120 Programming Language Rankings

Overview of R and Python

Python

Since its release in 1991, Python has been extremely popular and is widely used in data processing. Some of the reasons for its wide popularity are:

- Object-oriented language.
- General Purpose.
- Has a lot of extensions and incredible community support.
- Simple and easy to understand and learn.
- packages like pandas, numpy and scikit-learn, make Python an excellent choice for machine learning activities.

Note: **However, Python doesn't have specialized packages for statistical computing, unlike R**.

R

R’s first release came in 1995 and since then it has gone on to become one of the most used tools for data science in the industry.

- Consists of packages for almost any statistical application one can think of. CRAN currently hosts more than 10k packages.
- Comes equipped with excellent visualization libraries like ggplot2.
- Capable of standalone analyses

Note: **Performance wise R is not the fastest language and can be a memory glutton sometimes when dealing with large datasets.**

R	Python	Examples
Single-element vector	Scalar	`1`, `1L`, `TRUE`, `"foo"`
Multi-element vector	List	`c(1.0, 2.0, 3.0)`, `c(1L, 2L, 3L)`
List of multiple types	Tuple	`list(1L, TRUE, "foo")`
Named list	Dict	`list(a = 1L, b = 2.0)`, `dict(x = x_data)`
Matrix/Array	NumPy ndarray	`matrix(c(1,2,3,4), nrow = 2, ncol = 2)`
Data Frame	Pandas DataFrame	`data.frame(x = c(1,2,3), y = c("a", "b", "c"))`
Function	Python function	`function(x) x + 1`
Raw	Python bytearray	`as.raw(c(1:10))`
NULL, TRUE, FALSE	None, True, False	`NULL`, `TRUE`, `FALSE`

1. Import library
2. Data Frame
    - A. Creating a DataFrame
    - B. Viewing Data
    - C. Data Manipulation
        - a. Selection by Label
        - b. Selection by Position
    - D. Dealing with Missing Data
    - E. Statistical Operations
    - F. String Methods
    - G. Data frames Merge operation
        - a. Join's
3. Import dataset
4. Data Manipulation
5. Exploratory Data Analysis
6. Data Visualization
    - A. Correlation map
    - B. Line Plot
    - C. Scatter Plot
    - D. Histogram
    - E. Box plots

Import library

In [1]:

```
# Both langugares have library to perform set of functions.
# In R
library(readr) # CSV file I/O, e.g. the read_csv function
library(ggplot2) # Data visualization
library(dplyr) # Data Manupulation



# In Python

# # First, we'll import pandas, a data processing and CSV file I/O library
# import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
# import numpy as np   # linear algebra
# # We'll also import seaborn, a Python graphing library
# import warnings # current version of seaborn generates a bunch of warnings that we'll ignore
# warnings.filterwarnings("ignore")
# import seaborn as sns
# import matplotlib.pyplot as plt
# sns.set(style="white", color_codes=True)
```

Creating a DataFrame

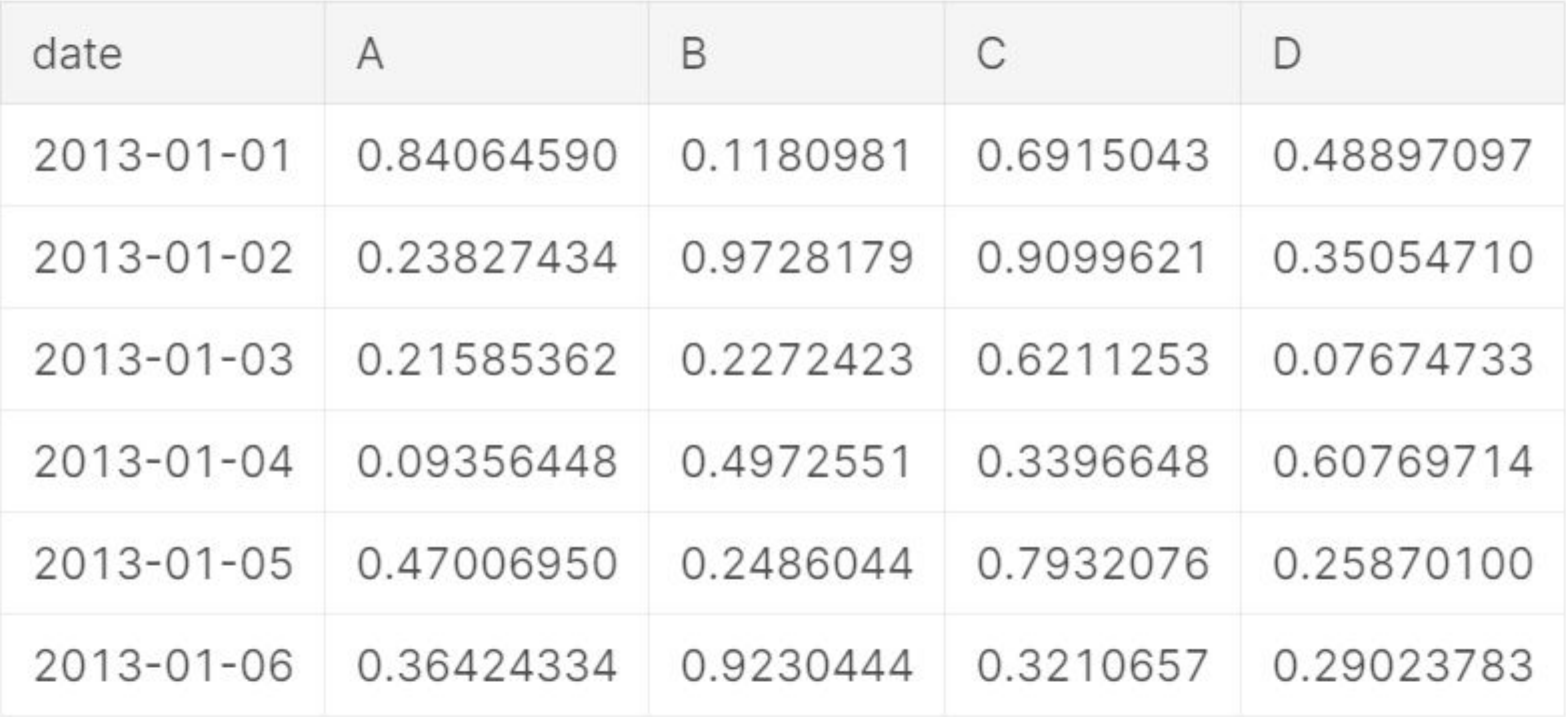
In [2]:

```
#Creating a DataFrame by passing a dict of objects that can be converted to series-like.
# In R
dates <-seq(as.Date("2013/01/01"), by = "day", length.out = 6)

df<- data.frame(date=dates,
                A=runif(6) ,
                B=runif(6) ,
                C=runif(6) ,
                D=runif(6)
               )
df

# In Python
# dates = pd.date_range('20130101', periods=6)
# df = pd.DataFrame(np.random.randn(6,4), index=dates, columns=list('ABCD'))
# data frames in python has indexing r won't have

# df
```



date	A	B	C	D
2013-01-01	0.84064590	0.1180981	0.6915043	0.48897097
2013-01-02	0.23827434	0.9728179	0.9099621	0.35054710
2013-01-03	0.21585362	0.2272423	0.6211253	0.07674733
2013-01-04	0.09356448	0.4972551	0.3396648	0.60769714
2013-01-05	0.47006950	0.2486044	0.7932076	0.25870100
2013-01-06	0.36424334	0.9230444	0.3210657	0.29023783

In [3]:

```
# In R
dates <-as.Date("2013/01/01")

df2<- data.frame(date=dates,
                A=1 ,
                B=runif(4) ,
                C=runif(4) ,
                D=runif(4) ,
                E=c("test","train","test","train") ,
                F="foo"
             )
df2



# In Python
# df2 = pd.DataFrame({ 'A' : 1.,
#                      'B' : pd.Timestamp('20130102'),
#                      'C' : pd.Series(1,index=list(range(4)),dtype='float32'),
#                      'D' : np.array([3] * 4,dtype='int32'),
#                      'E' : pd.Categorical(["test","train","test","train"]),
#                      'F' : 'foo' })
# df2
```

date	A	B	C	D	E	F
2013-01-01	1	0.2139692	0.004444704	0.2185053	test	foo
2013-01-01	1	0.3071877	0.918408287	0.9194862	train	foo
2013-01-01	1	0.5232002	0.577242457	0.4272279	test	foo
2013-01-01	1	0.6102341	0.515222023	0.6412675	train	foo

In [4]:

```
#data types
# In R
sapply(df, class)

# In Python
# df2.dtypes
```

```
date  'Date'
   A  'numeric'
   B  'numeric'
   C  'numeric'
   D  'numeric'
```

In [5]:

```
#See the top & bottom rows of the frame
# In R
head(df)


# In Python
#df.head()
```

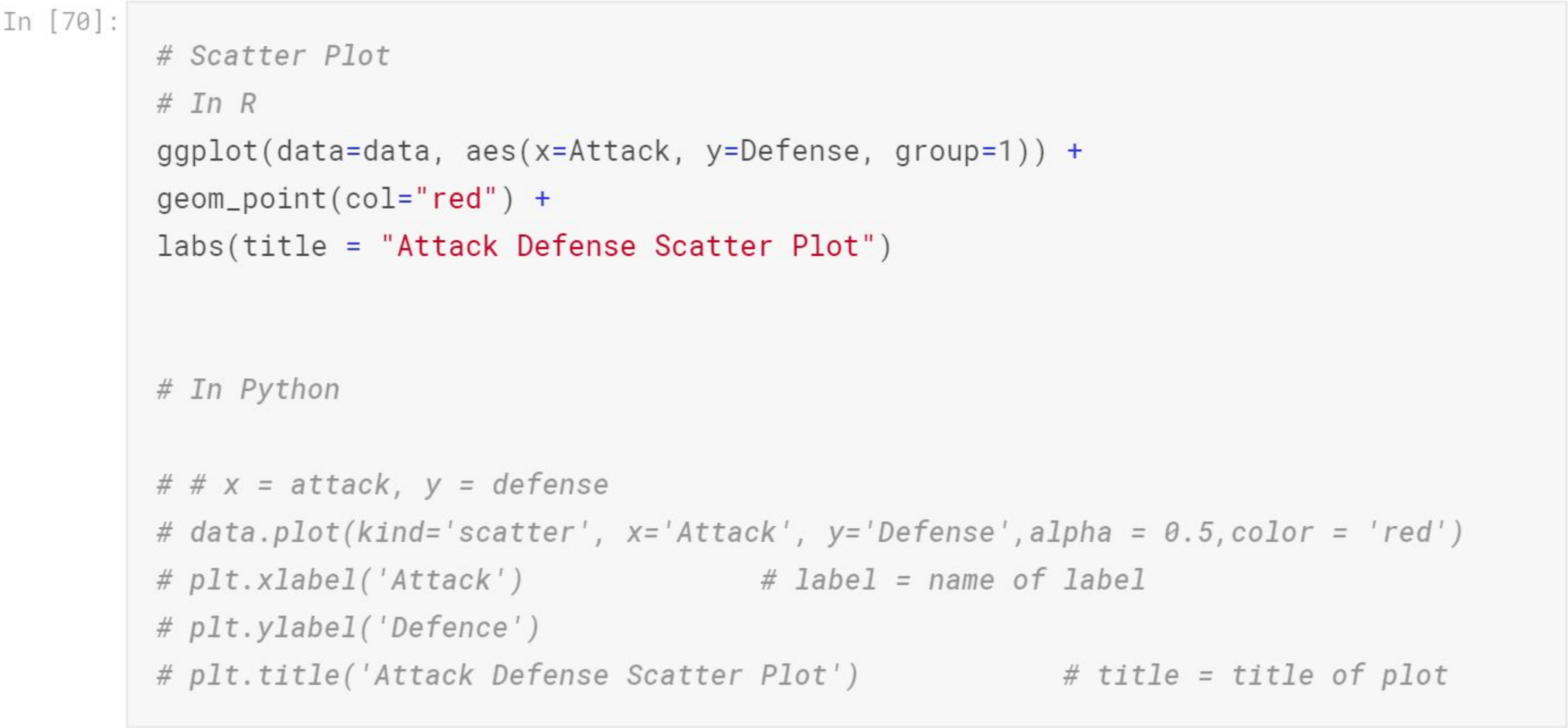
date	A	B	C	D
2013-01-01	0.84064590	0.1180981	0.6915043	0.48897097
2013-01-02	0.23827434	0.9728179	0.9099621	0.35054710
2013-01-03	0.21585362	0.2272423	0.6211253	0.07674733
2013-01-04	0.09356448	0.4972551	0.3396648	0.60769714
2013-01-05	0.47006950	0.2486044	0.7932076	0.25870100

In [11]:

```
#Describe shows a quick statistic summary of your data
# In R
summary(df)

# In Python
# df.describe()
```

```
     date                     A                   B                C          
 Min.   :2013-01-01   Min.   :0.09356   Min.   :0.1181   Min.   :0.3211  
 1st Qu.:2013-01-02   1st Qu.:0.22146   1st Qu.:0.2326   1st Qu.:0.4100  
 Median :2013-01-03   Median :0.30126   Median :0.3729   Median :0.6563  
 Mean   :2013-01-03   Mean   :0.37044   Mean   :0.4978   Mean   :0.6128  
 3rd Qu.:2013-01-04   3rd Qu.:0.44361   3rd Qu.:0.8166   3rd Qu.:0.7678  
 Max.   :2013-01-06   Max.   :0.84065   Max.   :0.9728   Max.   :0.9100  
       D          
 Min.   :0.07675  
 1st Qu.:0.26659  
 Median :0.32039  
 Mean   :0.34548  
 3rd Qu.:0.45437  
 Max.   :0.60770  
```



In [70]:

```
# Scatter Plot
# In R
ggplot(data=data, aes(x=Attack, y=Defense, group=1)) +
geom_point(col="red") +
labs(title = "Attack Defense Scatter Plot")



# In Python

# # x = attack, y = defense
# data.plot(kind='scatter', x='Attack', y='Defense',alpha = 0.5,color = 'red')
# plt.xlabel('Attack')              # label = name of label
# plt.ylabel('Defence')
# plt.title('Attack Defense Scatter Plot')          # title = title of plot
```

Calling Python from R

The **reticulate** package provides an R interface to Python modules, classes, and functions. For example, this code imports the Python `os` module and calls some functions within it:

```
library(reticulate)

os <- import("os")

os$listdir(".")
```

Reticulate package

Functions and other data within Python modules and classes can be accessed via the `$` operator (analogous to the way you would interact with an R list, environment, or reference class).

The **reticulate** package is compatible with all versions of Python >= 2.7. Integration with NumPy is optional and requires NumPy >= 1.6.

Python Version

By default, reticulate uses the version of Python found on your `PATH` (i.e. `Sys.which("python")`). The `use_python()` function enables you to specify an alternate version, for example:

```
library(reticulate)

use_python("/usr/local/bin/python" )
```

The `use_virtualenv()` and `use_condaenv()` functions enable you to specify versions of Python in virtual or conda environments, for example:

```
library(reticulate)

use_virtualenv("myenv")
```

Importing Modules

The `import()` function can be used to import any Python module. For example:

```
difflib <- import("difflib")

difflib$ndiff(foo, bar)
```

```
filecmp <- import("filecmp")

filecmp$cmp(dir1, dir2)
```

Sourcing Scripts

The `source_python()` function will source a Python script and make the objects it creates available within an R environment (by default the calling environment). For example, consider the following Python script:

```
def add(x, y):

return x + y
```

We source it using the `source_python()` function and then can call the `add()` function directly from R:

```
source_python('add.py')

add(5, 10)
```

Executing Code

You can execute Python code within the main module using the `py_run_file` and `py_run_string` functions. You can then access any objects created using the `py` object exported by reticulate:

```r
library(reticulate)

py_run_file("script.py")

py_run_string("x = 10")

# access the python main module via the 'py' object

py$x
```

Demonstration in R Studio : Calling Python Function in R