Reticule: R & Python

Dakota Cintron

August 22, 2018

Set Defaults

Here we are setting the defaults. The opts_chunk function is called from the knitr library. The reticulate package in R has previously been installed and is called here using library. By default, reticulate uses the Python version found on your PATH (i.e. Sys.which("python"))

```r
knitr::opts_chunk$set(echo = TRUE)
library(reticulate)
library(ggplot2)
```

Why reticulate (from Rstudio website)?

From the Wikipedia article on the reticulated python:

> The reticulated python is a species of python found in Southeast Asia. They are the world's longest snakes and longest reptiles. The specific name, reticulatus, is Latin meaning "net-like", or reticulated, and is a reference to the complex colour pattern.

From the Merriam-Webster definition of reticulate:

> 1: resembling a net or network; especially : having veins, fibers, or lines crossing a reticulate leaf.
> 2: being or involving evolutionary change dependent on genetic recombination involving diverse interbreeding populations.

The package enables you to *reticulate* Python code into R, creating a new breed of project that weaves together the two languages.

Figure 1: Reticulated Python from Belize!

Simulating Data in R & Python

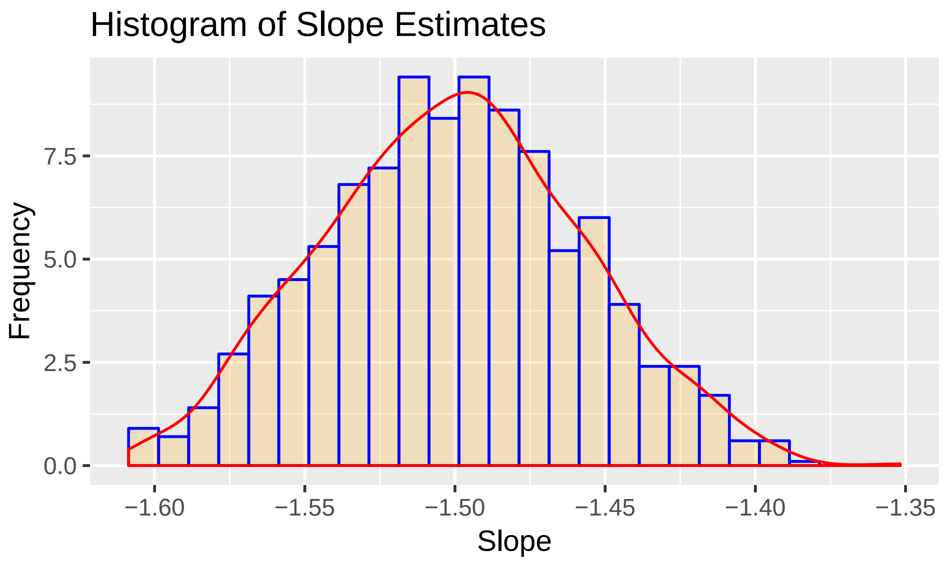
non-vectorized R approach for simulating data

The following simulates data for a simple OLS regression funtion. The X variable is normally distributed for 500 individuals with mean 0 and standard deviation 1. The Y variable is a function of X following the data generating process, which in this case is with an intercept of 2.0 and a slope of -1.5.

```
set.seed(782323)                   # Set the seed for reproducible results
reps <- 1000                       # replications
slope <- numeric(reps)             # Empty vector to store slope estimates
intercept <- numeric(reps)             # Empty vector to store slope estimates
b0 <- 2.0                          # True value for the intercept
b1 <- -1.5                         # True value for the slope of X
n <- 500                           # Sample size
X <- rnorm(n, mean = 0, sd = 1)    # Create a sample of n observations on the
                                   # independent variable X

for (i in 1:reps){                 # Start the simulation loop
        Y <- b0 + b1*X + rnorm(n, 0, 1)  # The true Data Generating Process (DGP),
                                         # with N(0, 1) error

        model <- lm(Y ~ X)               # Estimate the OLS model

        intercept[i] <- model$coef[1]      # Store estimates for intercept coefficients
        slope[i] <- model$coef[2]      # Store estimates for slope coefficients
}                                  # End the loop

# The mean and sd of the simulated intercept and slope values.
mean(intercept); mean(slope)
```

```
## [1] 1.999083
```

```
## [1] -1.500428
```



vectorized R approach for simulating data

A similar example to simulating data, using a vectorized approach in R. First, we write a function sim_function which takes the defaults for our parameters in the OLS equation. I use different names for slope and intercept to demonstrate that I am not overwriting the slope from the previous chunk.

```
sim_function <- function(X, b0=.2,b1=.5,n=500)
{
        X <- rnorm(n, mean = 0, sd = 1)
        Y <- b0 + b1*X + rnorm(n, 0, 1)  # The true DGP, with N(0, 1) error
        model <- lm(Y ~ X)               # Estimate the OLS model
        int <- model$coef[1]
        c("b0hat") -> names(int)
        slp <- model$coef[2]
        c("b1hat") -> names(slp)
        params <- cbind(int,slp)
}

set.seed(774987)
output <- replicate(1000,sim_function(X=X), simplify = "matrix" )
dim(output)
```

```
## [1]    2 1000
```

```
output <- t(output)
dim(output)
```

```
## [1] 1000    2
```

```
head(output)
```

```
##           [,1]      [,2]
## [1,] 0.2513949 0.5264440
## [2,] 0.1989534 0.5030985
## [3,] 0.2008878 0.4269342
## [4,] 0.2041265 0.4117166
## [5,] 0.1735151 0.4826162
## [6,] 0.1593361 0.4984751
```

```
mean(output[,1]);sd(output[,2])
```

```
## [1] 0.1987067
```

```
## [1] 0.04569008
```

```
mean(output[,2]);sd(output[,2])
```

```
## [1] 0.4983463
```

```
## [1] 0.04569008
```

Python approach to simulating data using numpy

First to verify that we are indeed using python within R we can check the version number of Python.

```
import sys
print(sys.version)
```

```
## 3.6.3 |Anaconda custom (64-bit)| (default, Oct 15 2017, 03:27:45) [MSC v.1900 64 bit (AMD64)]
```

Before simulating data with python, we can see how to call data from the "R environment" to use within Python. I will read the reps, n, intercept parameter, and slope parameter into the Python environment to use in Python.

```
print(r.b0)
```

```
## 2.0
```

```
print(r.b1)
```

```
## -1.5
```

```
print(r.n)
```

```
## 500.0
```

```
print(r.reps)
```

```
## 1000.0
```

```
print(type(r.X))
# convert X to pandas object not list
```

```
## <class 'list'>
```

```
import pandas as pd # importing pandas package in python
import numpy as np # importing numpy package in python
X = pd.DataFrame(np.array(r.X), columns = list("X"))
print(type(X))
```

```
## <class 'pandas.core.frame.DataFrame'>
```

```
print(X.shape)
```

```
## (500, 1)
```

```
print(X.head())
```

```
##            X
## 0  1.955224
## 1 -1.188715
## 2  0.771389
## 3 -0.914338
## 4  0.467658
```