Lecture 9: Intro to Python

What *is* R?

- R is a programming language specifically designed for statistics and data analysis
  - Objects for storing data, and functions for interacting with data, are fundamental
  - R is very good at graphics and visualization
  - R is easily extended. Users can write and share their own functions and packages
- We can interact with R through IDEs like RStudio

Who uses R?

- R is widely used in statistical research and academia
- R is also widely used in applications of statistics, such as
  - Biology and bioinformatics
  - Ecology, forestry, and environmental science
  - Psychology
  - Sociology
- R is used in a variety of areas outside research, by government agencies, pharmaceutical companies, insurance companies, etc.

What other options exist?

- SAS ← biostat, clinical
- Stata ← econ
- SPSS ← psychology
- Excel ← business
- Python ← data science
- Julia
- Matlab ← applied math, engineering
- Many others...

What is Python

- Python is a general-purpose programming language
- Like R, python has a wide range of packages to extend functionality
- Certain Python packages allow for sophisticated data analysis and modeling
  - `SciPy`, `NumPy`
  - `scikit-learn`, `statsmodels`, `pytorch`
  - `pandas`
  - `matplotlib`

R vs. Python

My own, *personal*, preferences:

R is good for

- Data visualization and wrangling
- Classical statistics
- Statistical inference
- New statistical methods

Python is good for

- General-purpose programming
- Challenging data types (e.g. images)
- Prediction and machine learning

Why do we teach R?

- Excellent support for the material covered in a statistics degree
- Benefits to using a consistent language across courses
- Valuable for a wide variety of future careers
- The primary research tool for most (if not all) the faculty

A taste of Python

Recall our code from the first class:

```r
M <- 10 # number of people at the party
hats <- 1:M # numbered hats
nsim <- 10000 # number of simulations
results <- rep(0, nsim) # vector to store the results

for(i in 1:nsim){
  randomized_hats <- sample(hats, M, replace = FALSE)
  results[i] <- sum(randomized_hats == hats) > 0
}

mean(results)
```

```
[1] 0.6296
```

A taste of Python

Here is the same code, written in Python

```
1  import numpy as np

2

3  M = 10 # number of people at the party
4  hats = np.arange(M) # numbered hats
5  nsim = 10000 # number of simulations
6  results = np.zeros(nsim) # to store the results
7
8  for i in range(nsim):
9      randomized_hats = np.random.choice(hats, M, replace = False)
10     results[i] = np.sum(randomized_hats == hats) > 0
11
12 np.mean(results)
```

```
0.634
```

What similarities and differences do you notice?

Python: = for assignment R: <- (or =) for assignment

Step 1: representing the hats

```
1 import numpy as np
2
3 M = 10 # number of people at the party
4 hats = np.arange(M) # numbered hats
5
6 hats
```

0, 1, 2, ..., M-1

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
1 hats[0]
```

```
0
```

```
1 hats[1]
```

```
1
```

- `hats` is a 1-dimensional array (similar to a vector in R)
- Python is *0-indexed*: the first entry is `hats[0]`

Step 2: everyone draws a random hat

```
1  import numpy as np
2
3  M = 10 # number of people at the party
4  hats = np.arange(M) # numbered hats
5
6  randomized_hats = np.random.choice(hats, M, replace = False)
7
8  randomized_hats
```

sample size

what replace

```
array([6, 8, 5, 2, 1, 0, 4, 7, 9, 3])
```

- `np.random.choice` works like R's `sample` function
- Booleans in Python are `True` and `False` (as opposed to TRUE and FALSE, or T and F)

package

collection of objects / functions in the numpy library

a function in np.random

Step 3: check who got their original hat

```
1  import numpy as np
2
3  M = 10 # number of people at the party
4  hats = np.arange(M) # numbered hats
5
6  randomized_hats = np.random.choice(hats, M, replace = False)
7  randomized_hats
```

```
array([5, 6, 0, 3, 1, 8, 9, 7, 2, 4])
```

```
1  randomized_hats == hats
```

```
array([False, False, False,  True, False, False, False,  True, False,
       False])
```

```
1  np.sum(randomized_hats == hats)
```

```
2
```

- NumPy arrays allow for “vectorized” operations, like in R

a = [. .]

Step 4: iteration

```
1  import numpy as np
2
3  M = 10 # number of people at the party
4  hats = np.arange(M) # numbered hats
5  nsim = 10000 # number of simulations
6  results = np.zeros(nsim) # to store the results
7
8  for i in range(nsim):
9      randomized_hats = np.random.choice(hats, M, replace = False)
10     results[i] = np.sum(randomized_hats == hats) > 0
11
12 np.mean(results)
```

rep(0, nsim) in R

ith entry in results is True if at least one person got their original hat. False otherwise.

```
0.6282
```

- `range(nsim)` is similar to `1:nsim` in R
- We don't use the curly braces `{ }`. Instead we use whitespace (four spaces is standard, you just have to be consistent)

Using Python through RStudio

- You can make Python chunks in Quarto documents, just like R chunks:

````
```{python}

```
````

Class activity

https://sta279-f23.github.io/class_activities/ca_lecture_9.html