

Lecture 10: 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

What other options exist?

- SAS
- Stata
- SPSS
- Excel
- Python
- Julia
- Matlab
- 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

A taste of 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)
```

0.6299

- What is this code doing?
- What similarities and differences do you notice, compared to R?

A taste of Python

Recall our code from the first class:

```
1 M <- 10 # number of people at the party
2 hats <- 1:M # numbered hats
3 nsim <- 10000 # number of simulations
4 results <- rep(0, nsim) # vector to store the results
5
6 for(i in 1:nsim){
7   randomized_hats <- sample(hats, M, replace = FALSE)
8   results[i] <- sum(randomized_hats == hats) > 0
9 }
10
11 mean(results)
```

```
[1] 0.625
```

A taste of Python

Here is the same code, written in Python

```
1 import numpy as np    ← similar to library(...)
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):  range(nsim) instead of 1: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)
```

no parens

comments are the same!

looks like indexing a vector in R

use "sum" function from numpy (abbreviated as np)

write space/indentation used to specify scope

0.6259

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
```

array containing 0, 1, 2, ..., M-1 (length M)

```
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
```

array to sample from
sample size
w/out replacement

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

- `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 package

a function in
`np.random`

Step 3: check who got their original hat

```
1 import numpy as np
2 np.random.seed(3)
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([9, 4, 0, 5, 3, 6, 2, 8, 1, 7])

```
1 randomized_hats == hats
```

to check for equality

array([False, False, False, False, False, False, False, False, False, False])

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

0

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

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           0 0 0 ... 0           rep(0, nsim) in R
8 for i in range(nsim):           0, 1, 2, ..., nsim-1
9     randomized_hats = np.random.choice(hats, M, replace = False)
10    results[i] = np.sum(randomized_hats == hats) > 0
11
12 np.mean(results) ← fraction of times at least one person got their hat
```

0.6361

- `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:

```
1  ```{python}
2
3  ```
```

Class activity

https://sta279-s24.github.io/class_activities/ca_lecture_10.html

