BIOS-584 Python Programming (Non-Bios Student)

Week 05

Instructor: Tianwen Ma, Ph.D.

Department of Biostatistics and Bioinformatics,

Rollins School of Public Health,

Emory University

Lecture Overview

- Use random numbers and simulations to solve problems
- Draw random samples from different distributions
- Create subplots to compare different simulations
- Practice nested loops
- Write for loops with if/else statements
  - Simulate discrete random variables
- week-05-application-simulation.ipynb

Modifications of this slide

- For this lecture, I only give hints on the code.
- You write your own code to fill in the blanks.
- The slide is longer because I show them step-by-step.
- Pages with jupyter are related to in-class coding.
- Raise hands if you have questions.
  - You can simply say “I don’t understand this step.”

Simulate random variables

- We use random numbers to simulate different distributions
- Previously we have learned random module to generate random numbers
- We can use functions within numpy module to simulate different distributions.
- np.random.distribution_name(parameters)
  - Normal, uniform, binomial, Chi-square, etc.
- We will start from continuous random variables.

numpy documentation:
https://numpy.org/doc/stable/reference/random/generator.html#distributions

Continuous random variables

- We import relevant packages.

Continuous random variables

- We import relevant packages.
- Set seed for reproducibility (seed value=100)

Continuous random variables

- We import relevant packages.
- Set seed for reproducibility (seed value=100)
- The output of each simulation is a sample (vector) with **n** observations.
  - n=10000

Continuous random variables

- We import relevant packages.
- Set seed for reproducibility (seed value=100)
- The output of each simulation is a sample (vector) with **n** observations.
  - n=10000
- Suppose we perform three simulations:
  - Normal: mean=1, variance=4
  - Chi-square: df=1
  - Uniform: [-5, 5].

Continuous random variables

- We import relevant packages.
- Set seed for reproducibility (seed value=100)
- The output of each simulation is a sample (vector) with **n** observations.
  - n=10000
- Suppose we perform three simulations:
  - Normal: mean=1, variance=4
  - Chi-square: df=1
  - Uniform: [-5, 5].

Continuous random variables

- We import relevant packages.
- Set seed for reproducibility (seed value=100)
- The output of each simulation is a sample (vector) with **n** observations.
  - n=10000
- Suppose we perform three simulations:
  - Normal: mean=1, variance=4
  - Chi-square: df=1
  - Uniform: [-5, 5].
- Define each random vector as vec_normal, vec_chisq, and vec_uniform.

Continuous random variables

- Compute mean, std, median, Q1, and Q3 for each random vector.

Continuous random variables

- Compute mean, std, median, Q1, and Q3 for each random vector.
- For example, for normal vector, name their summary statistics as
    - norm_mean_val, norm_std_val
    - norm_median_val, norm_q1_val, norm_q3_val

Continuous random variables

- Compute mean, std, median, Q1, and Q3 for each random vector.
- For example, for normal vector, name their summary statistics as
  - norm_mean_val, norm_std_val
  - norm_median_val, norm_q1_val, norm_q3_val
- Print out all summary statistics in one sentence per distribution using .format() syntax.

Continuous random variables

- Compute mean, std, median, Q1, and Q3 for each random vector.
- For example, for normal vector, name their summary statistics as
  - norm_mean_val, norm_std_val
  - norm_median_val, norm_q1_val, norm_q3_val
- Print out all summary statistics in one sentence per distribution using .format() syntax.
- Avoid duplicating print functions by using a for loop.
  - Involve indexing the element of a list (and a nested list).

Multiple plots in a row (subplots)

- Previously, we learned how to create multiple plots in multiple figures.
- We can create multiple plots in the same figure using plt.subplots() function.
- This function returns a figure and an array of axes
  - fig, axes = plt.subplots(nrows, ncols)
  - You can change the output with other names
  - nrows: the number of rows of subplots
  - ncols: the number of columns of subplots
  - Within each axis, using the plot() function.

https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.subplots.html

Subplots with 1x3 dimension

- Initialize the subplots with correct dimension size and figure size and export the figure names and axes (list_subfig in our case).

Subplots with 1x3 dimension

- Initialize the subplots with correct dimension size and figure size and export the figure names and axes (list_subfig in our case).
- Plot each subplot list_subfig[2]

Subplots with 1x3 dimension

- Initialize the subplots with correct dimension size and figure size and export the figure names and axes (list_subfig in our case).
- Plot each subplot list_subfig[2]
- Label x and y axes as well as the title.

Subplots with 1x3 dimension

- Initialize the subplots with correct dimension size and figure size and export the figure names and axes (list_subfig in our case).
- Plot each subplot list_subfig[2]
- Label x and y axes as well as the title.
- Display the plot and set other optional parameters

Subplots with 2x2 dimension

- Initialize the subplots with correct dimension size and figure size and export the figure names and axes (list_subfig2 in our case).

Subplots with 2x2 dimension

- Initialize the subplots with correct dimension size and figure size and export the figure names and axes (list_subfig2 in our case).
- Plot each subplot list_subfig[0,0], list_subfig[0,1], and list_subfig[1,0]
  - Label x and y axes as well as the title.

Subplots with 2x2 dimension

- Initialize the subplots with correct dimension size and figure size and export the figure names and axes (list_subfig2 in our case).
- Plot each subplot list_subfig[0,0], list_subfig[0,1], and list_subfig[1,0]
  - Label x and y axes as well as the title.
- Display the plot and set other optional parameters

Create sequences of numbers

- We met range() function and np.arange() function to create a sequence of numbers.
- Let's introduce it formally!
- Syntax: list(range(start, stop, step)) to create a sequence of numbers.
    - start: The first number in the sequence
    - stop: The last number in the sequence (not included)
    - step: difference between adjacent numbers in the sequence.
    - If start is omitted, it defaults to 0.
    - If step is omitted, it defaults to 1.

Create a sequence of numbers

- For range(), it only supports integers, while np.arange(), it supports floating numbers.
- The boundary setting is a little weird, but it follows the principle of half-open intervals (including the starting value but excludes the stop value).
- Since Python's index starts from 0, this design makes it easier to work with lists and arrays.
  - list[0:3] or list[:3] return the first three elements in the list.
  - list[1:] returns elements excluding the first one.

Create a sequence of numbers

- Create a list with the numbers from 0 to 100, denoted as int_ls.

Create a sequence of numbers

- Create a list with the numbers from 0 to 100, denoted as int_ls.
- Create a list with all even numbers from 1 to 100, denoted as even_ls.

Create a sequence of numbers

- Create a list with the numbers from 0 to 100, denoted as int_ls.
- Create a list with all even numbers from 1 to 100, denoted as even_ls.
- Find the first 10 elements of int_ls and even_ls.

Create a sequence of numbers

- Create a list with the numbers from 0 to 100, denoted as int_ls.
- Create a list with all even numbers from 1 to 100, denoted as even_ls.
- Find the first 10 elements of int_ls and even_ls.
- Find the elements excluding the first two elements of int_ls and even_ls.

Nested Loops

- We can nest loops inside each other to iterate over multiple dimensions.
- The inner loop runs to completion for each iteration of the outer loop
- This is useful when we need to iterate over a matrix or a list of lists.
- The syntax is simple: Write a loop inside another loop

Nested Loops

- The inner loop is indented twice: once for the outer loop and once for the inner loop.
- Given i=0, j iterates 0, 1, 2.
- Repeat for i=1,2,3.
- You can modify the code section.

```
for i in range(4):
    for j in range(3):
        print("i = {}, j = {}".format(i, j))
```

```
i = 0, j = 0
i = 0, j = 1
i = 0, j = 2
i = 1, j = 0
i = 1, j = 1
i = 1, j = 2
i = 2, j = 0
i = 2, j = 1
i = 2, j = 2
i = 3, j = 0
i = 3, j = 1
i = 3, j = 2
```

Central Limit Theorem (CLT)

- The CLT is a fundamental concept in statistics.
- It states that the distribution of the mean (or sum) of many independent, identically distributed random variables approaches to a normal distribution, regardless of the original distribution
- This is true even if the original distribution is NOT normal.

Real-world examples

- Quality control in manufacturing:
- A manager can take a random sample of products to assess the proportion of defective items produced.
- The CLT allowed them to use this sample to make an informed estimate of the defect rate for the entire plant.

Real-world examples

- Epidemiologists study how common illnesses or health behaviors are in populations using surveys.
- For example, to see how many adults in an area are obese, they select a random sub-population and measure people's height and weight and calculate the BMI.
- The CLT allowed them to use this sample to make an informed estimate of the obesity rate for the entire poupation.

Central Limit Theorem (CLT)

- The larger the sample size is, the closer the distribution of the sample mean is to a normal distribution.
- We use simulation studies to understand the central limit theorem.
- To simplify, let's simulate the CLT using the uniform distribution.

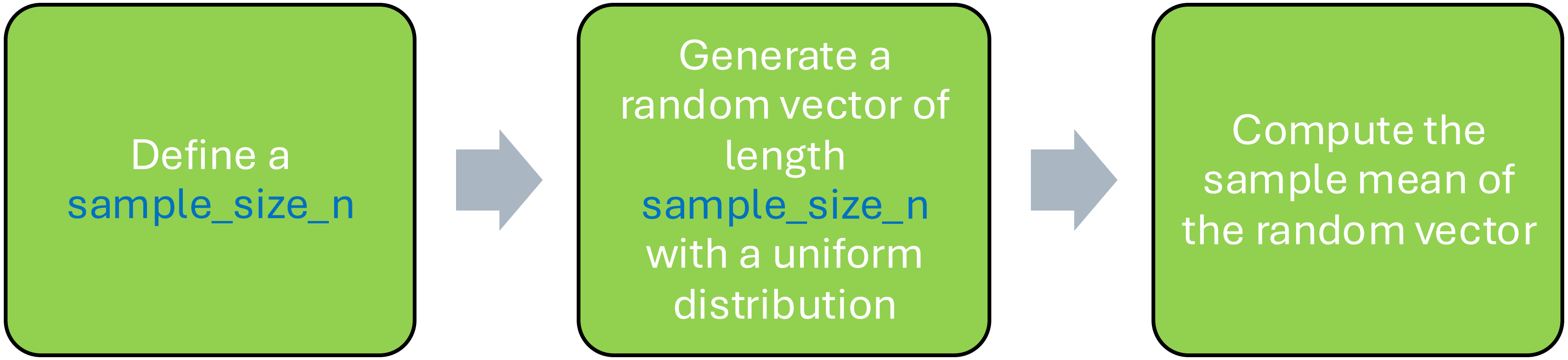
Logic Flow (before iterations)

- We pre-specify the number of iterations
  - iteration_num: The number of repetitions

Logic Flow (before iterations)

- We pre-specify the number of iterations
  - iteration_num: The number of repetitions
- We initialize an empty list to save the sample mean
  - sample_size_n: within each repetition, the length of a random vector with a uniform distribution

Logic Flow within each iteration



Logic Flow within each iteration

Define a sample_size_n → Generate a random vector of length sample_size_n with a uniform distribution → Compute the sample mean of the random vector

The above process is repeated iteration_num times.
You append the sample mean to the vec_unif1 list.

Subplot after four scenarios

- Repeat the steps in the previous two slides to create unif_vec10, unif_vec50, and unif_vec100.

Subplot after four scenarios

- Repeat the steps in the previous two slides to create unif_vec10, unif_vec50, and unif_vec100.
- Create a 2x2 subplot to display histograms
  - The first row: unif_vec1 (left) and unif_vec10 (right)
  - The second row: unif_vec50 (left) and unif_vec100 (right)

Subplot after four scenarios

- Repeat the steps in the previous two slides to create unif_vec10, unif_vec50, and unif_vec100.
- Create a 2x2 subplot to display histograms
  - The first row: unif_vec1 (left) and unif_vec10 (right)
  - The second row: unif_vec50 (left) and unif_vec100 (right)
- Display the final subplot

Simplify with a nested for loop

- We create a list sample_size_ls to store the sample size, i.e, 1, 10, 50, and 100

Simplify with a nested for loop

- We create a list sample_size_ls to store the sample size, i.e, 1, 10, 50, and 100
- We create an empty list unif_vec_ls_ls to store the random vector for each scenario of sample_size_ls

Simplify with a nested for loop

- We create a list sample_size_ls to store the sample size, i.e, 1, 10, 50, and 100
- We create an empty list unif_vec_ls_ls to store the random vector for each scenario of sample_size_ls
- The inner for loop is to create a random vector to store the sample mean of uniform vectors per sample size

Simplify with a nested for loop

- We create a list sample_size_ls to store the sample size, i.e, 1, 10, 50, and 100
- We create an empty list unif_vec_ls_ls to store the random vector for each scenario of sample_size_ls
- The inner for loop is to create a random vector to store the sample mean of uniform vectors per sample size
- The outer for loop is to repeat for each sample size, i.e., 1, 10, 50, and 100.

Simplify with a nested for loop

- Initialize an empty list unif_vec_ls_ls

Simplify with a nested for loop

- Initialize an empty list unif_vec_ls_ls
- Each element of unif_vec_ls_ls is another list with iteration_num sample mean values

Simplify with a nested for loop

- Initialize an empty list unif_vec_ls_ls
- Each element of unif_vec_ls_ls is another list with iteration_num sample mean values
- Within each sample size, initialize an empty list unif_vec_sample_size_ls

Simplify with a nested for loop

- Initialize an empty list unif_vec_ls_ls
- Each element of unif_vec_ls_ls is another list with iteration_num sample mean values
- Within each sample size, initialize an empty list unif_vec_sample_size_ls
- Repeat iteration_num times and for each iteration, generate a uniform vector and calculate the sample mean

Simplify with a nested for loop

- Initialize an empty list unif_vec_ls_ls
- Each element of unif_vec_ls_ls is another list with iteration_num sample mean values
- Within each sample size, initialize an empty list unif_vec_sample_size_ls
- Repeat iteration_num times and for each iteration, generate a uniform vector and calculate the sample mean
- Append the sample mean to unif_vec_sample_size_ls

Simplify with a nested for loop

- After the completion of inner for loop, append unif_vec_sample_size_ls to unif_vec_ls_ls.

Simplify with a nested for loop

- After the completion of inner for loop, append unif_vec_sample_size_ls to unif_vec_ls_ls.
- Display the four histograms using a 1x4 subplot
  - Left to right order: 1, 10, 50, and 100

for loop with if/else statements

- In addition to nested for loop, we can write if/else statement within the for loop.
- The syntax looks like:

```
for iter_id in total_ls:
    if condition1:
        statement1
    else:
        statement2
```

- We illustrate this function via simulating a discrete random variable.
  - We start with a binary variable.

Discrete random variable from a uniform distribution

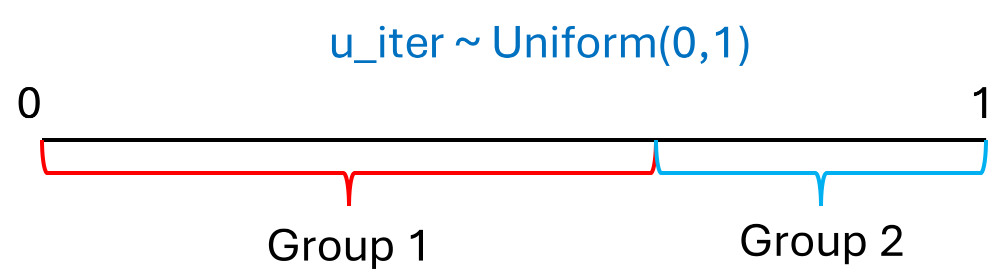
- Suppose we want to randomize a clinical trial of two treatment groups to a N_total patients with a 2:1 ratio
- How can we generate the treatment group id to make it **random** and approximately follow the **2:1** ratio?
- We define two treatment groups as 1 and 2 such that the number of group 1 is 2 times the number of group 2.

Discrete random variable from a uniform distribution

- We define N_total and initialize an empty list to store the treatment group id.

Discrete random variable from a uniform distribution

- We define N_total and initialize an empty list to store the treatment group id.
- Within each iteration
  - We generate a u_iter from Uniform(0, 1) .
  - If u_iter<2/3, we assign it to group 1, otherwise, we assign it to group 2.
  - The < or <= does not make any difference.



Discrete random variable from a uniform distribution

- We define N_total and initialize an empty list to store the treatment group id.
- Within each iteration
  - We generate a u_iter from Uniform(0, 1) .
  - If u_iter<2/3, we assign it to group 1, otherwise, we assign it to group 2.
  - The < or <= does not make any difference.
- We replicate it for N_total times and store the treatment id using .append().

Discrete random variable from a uniform distribution

- We define N_total and initialize an empty list to store the treatment group id.
- Within each iteration
  - We generate a u_iter from Uniform(0, 1) .
  - If u_iter<2/3, we assign it to group 1, otherwise, we assign it to group 2.
  - The < or <= does not make any difference.
- We replicate it for N_total times and store the treatment id using .append().
- Print out the actual numbers of patients who receive treatment 1 and 2 using .format().

Discrete random variable from a uniform distribution

- When N_total is moderate or small, the actual ratio may not be exactly 2:1.
- However, if you use bigger N_total, the actual ratio will approach to 2:1.

- Q: What if we have three treatment groups with a 2:2:1 ratio?
  - Expand your current one cutoff to two cutoffs.
- You will work on this in HW5.