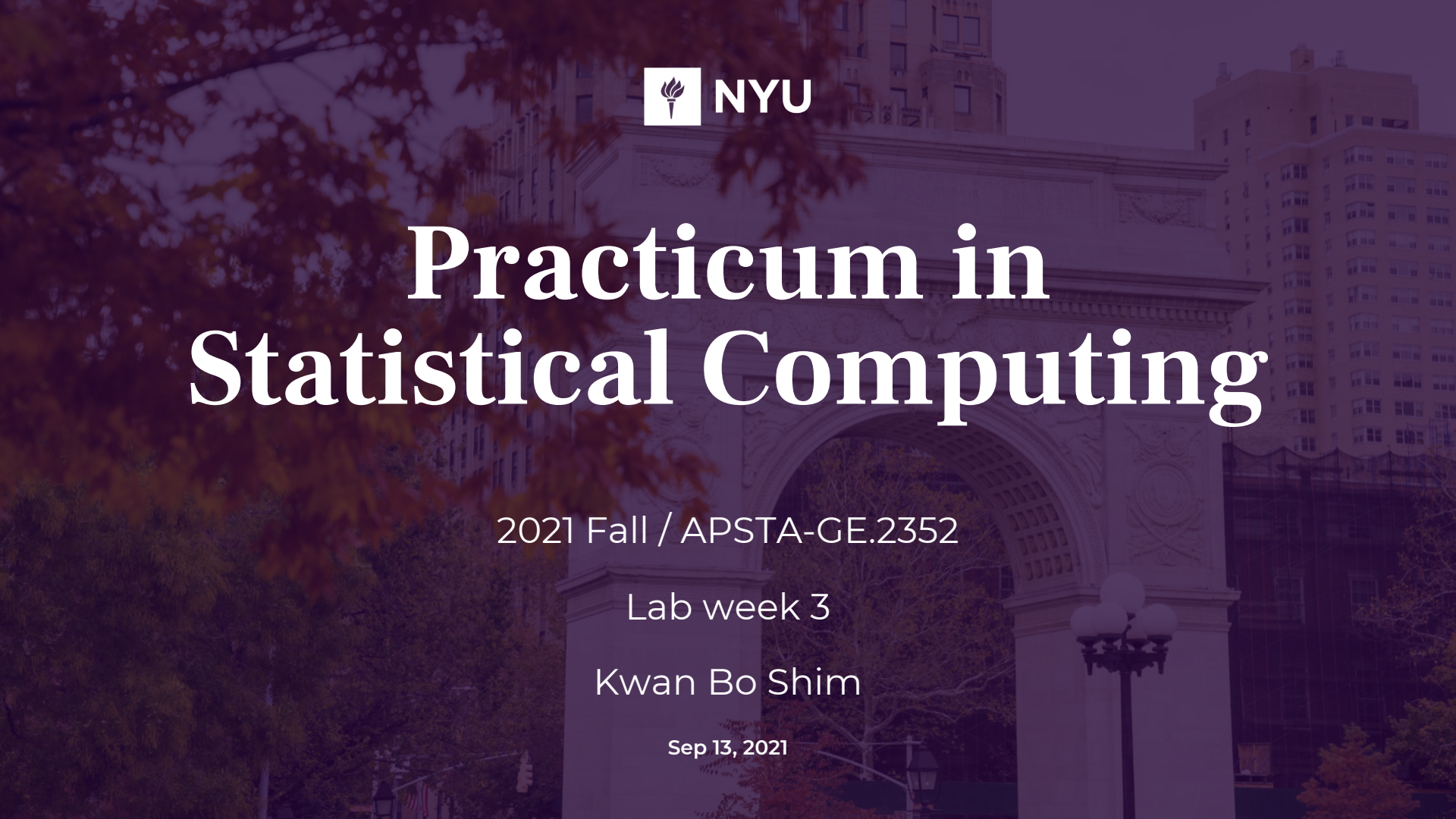

NYU

Practicum in Statistical Computing

2021 Fall / APSTA-GE.2352

Lab week 3

Kwan Bo Shim

Sep 13, 2021

PART 01

Week 3

PART 01

Week 3

1. Density plot

2. Functional programming

3. Loop and conditionals

4. Lab (practice)

NYU

Density estimation

1. Nonparametric **density estimation** is an approach to estimating the distribution of the population from which the sample was drawn. The density estimation is nonparametric in the sense that the sample data suggest the shape, rather than imposing the shape of a known population distribution with particular values of its parameters.
2. Nonparametric **kernel density estimation** can be thought of as a method of averaging and smoothing the density estimate that would be provided by a histogram
3. Kernel density estimation works by estimating the density at each observation, x, using a smooth, weighted function, known as a kernel.
4. Changing either the **kernel function or the smoothing parameter** would affect the overall density that is estimated. The former by changing the relative weights and the latter by changing the range of data used in the estimate.

```
density(vlss$Age, bw = "SJ", adjust = 0.5,
kernel="gaussian")
```

1. bw: smoothing bandwidth; The kernels are scaled such that this is the standard deviation of the smoothing kernel

2. adjust: Used with bw*adjust -> convenience

3. kernel: giving the smoothing kernel to be used. Default="gaussian".

Variability band

1. One method of capturing the uncertainty associated with a density estimate is to create a variability band or interval around the density estimate.

    - p(x) = estimate(p(x)) + 2 * SE(p(x))

2. We use sm package to generate the plot!

Functional programming / conditional and loop

1. **A function** is a statement to perform a defined set of commands. It allows for easy variation and repetition of those commands. As you write more code as scripts, you might find yourself repeating (cutting and pasting) the same code over and over and only changing one or two things. In this case, **a function is likely to save you a lot of time**, and those reading your code will have an easier time making sense of it.
2. **Conditional operators** are used to evaluate a condition that's applied to one or two boolean expressions. The result of the evaluation is either true or false.
3. **A for-loop** (or simply for loop) is a control flow statement for specifying iteration, which allows code to be executed repeatedly.

```
status = c("sleep", "awake")
mental_status = c("good", "tired")


If ( You == status[2] ){
  if( mental == mental_status[2]){
      print("You need coffee")
  } else {
      print("FoCuS on your lab")
  }
}
```

“Decision helper Bot”

Answer to these specific conditions

1. “I'm hungry”
2. “Maybe some boba”
3. “some desserts”

Data collection

```
You <- c(“I'm hungry” , “Maybe some boba” ,
“I don’t know”, “some desserts”, “some
desserts”, ”I'm hungry”, “stop asking me”,
“Maybe some boba”, “I need coffee”)
```

```
for (i in 1:length(You)){
  If ( You[i] == “I'm hungry”){
     print(“how about chipotle?”)
     } else if (You[i]== “Maybe some boba"){
     print(“go to bubble tea place”)
     } else if (You[i]== “some desserts”) {
     print(“Mille-Feuille Bakery”)
     } else {
     print(“think more and come back”)
     }
  }
}
```