Homework 1 Solutions

Overview

Instructions

The goal of this homework assignment is to introduce you to simulation-based data analysis.

- Problem 1 asks you to explore whether a difference between data collected from two groups might be statistically meaningful or the result of noise. This problem repeats the analysis from Statistics Without The Agonizing Pain by John Rauser (which is a neat watch!).
- Problem 2 asks you to evaluate an interview method for finding the level of cheating on a test to determine whether cheating was relatively high or low. This problem was adapted from Bayesian Methods for Hackers.

Load Environment

The following code loads the environment and makes sure all needed packages are installed. This should be at the start of most Julia scripts.

```
import Pkg
Pkg.activate(@__DIR__)
Pkg.instantiate()
```

The following packages are included in the environment (to help you find other similar packages in other languages). The code below loads these packages for use in the subsequent notebook (the desired functionality for each package is commented next to the package).

```
using Random # random number generation and seed-setting
using DataFrames # tabular data structure
using CSVFiles # reads/writes .csv files
using Distributions # interface to work with probability distributions
using Plots # plotting library
using StatsBase # adds some basic statistical functionality (means, medians,
 ↪  empirical cumulative densities, etc)
using StatsPlots # for qqplots
```

Problems

Problem 1

```
data = DataFrame(load("data/bites.csv")) # load data into DataFrame

# print data variable (semi-colon suppresses echoed output in Julia, which in
 ↪  this case would duplicate the output)
@show data;
```

```
data = 43×2 DataFrame
 Row   group   bites
       String  Int64

   1   beer       27
   2   beer       20
   3   beer       21
   4   beer       26
   5   beer       27
   6   beer       31
   7   beer       24
   8   beer       21
   9   beer       20
  10   beer       19
  11   beer       23
  12   beer       24
  13   beer       28
  14   beer       19
  15   beer       24
  16   beer       29
  17   beer       18
```

```
 18   beer       20
 19   beer       17
 20   beer       31
 21   beer       20
 22   beer       25
 23   beer       28
 24   beer       21
 25   beer       27
 26   water      21
 27   water      22
 28   water      15
 29   water      12
 30   water      21
 31   water      16
 32   water      19
 33   water      15
 34   water      22
 35   water      24
 36   water      19
 37   water      23
 38   water      13
 39   water      22
 40   water      20
 41   water      24
 42   water      18
 43   water      20
```

```julia
# split data into vectors of bites for each group
beer = data[data.group .== "beer", :bites]
water = data[data.group .== "water", :bites]

observed_difference = mean(beer) - mean(water)
@show observed_difference;
```

```
observed_difference = 4.37777777777778
```

In this problem:

- Conduct the above procedure to generate 50,000 simulated datasets under the skeptic's hypothesis.
- Plot a histogram of the results and add a dashed vertical line to show the experimental difference (if you are using Julia, feel free to look at the Making Plots with Julia tutorial on the class website).

- Draw conclusions about the plausibility of the skeptic's hypothesis that there is no difference? Feel free to use any quantitative or qualitative assessments of your simulations and the observed difference.

Solution:

First, we write a function (`simulate_differences()`) to generate a new data set and compute the group differences under the skeptic's hypothesis by shuffling the data across the two groups (this is called *the non-parametric bootstrap*, which we will talk about more later):

```
# simulate_differences: function which simulates a new group difference based
 ↪  on the skeptic's hypothesis of no "real" difference between groups by
 ↪  shuffling the input data across groups
# inputs:
#   y : vector of bite counts for the beer-drinking group
#   y : vector of bite counts for the water-drinking group
# output:
#   a simulated difference between shuffled group averages
function simulate_differences(y , y )
    # concatenate both vectors into a single vector
    y = vcat(y , y )

    # create new experimental groups consistent with skeptic's hypothesis
    y_shuffle = shuffle(y)   # shuffle the combined data
    n  = length(y )
    x  = y_shuffle[1:n ]
    x  = y_shuffle[(n +1):end]

    # compute difference between new group means
    diff = mean(x ) - mean(x )
    return diff
end
```

```
simulate_differences (generic function with 1 method)
```

Next, we evaluate this function 10,000 times and plot the resulting histogram of differences.

In Julia (and in Python), it is convenient to use a *comprehension* to automatically allocate the output of the `for` loop to a vector. The syntax for a comprehension is `[some_function(input) for input in some_range]`. In this case, the index `input` doesn't appear in the comprehension as we're just repeating the exact same calculation every time:

```
shuffled_differences = [simulate_differences(beer, water) for i in 1:50_000]
```

```
50000-element Vector{Float64}:
  0.173333333333332
 -0.7822222222222202
 -0.3999999999999986
  0.9377777777777787
  0.8422222222222224
  0.8422222222222224
  0.36444444444444457
  1.9888888888888907
 -2.884444444444444
 -0.4955555555555584
 -1.7377777777777794
  0.8422222222222224
 -0.9733333333333327

 -0.20888888888888957
  1.5111111111111093
 -0.20888888888888957
 -0.9733333333333327
 -0.3999999999999986
 -2.120000000000001
 -0.4955555555555584
 -0.4955555555555584
  0.9377777777777787
  2.1799999999999997
  1.4155555555555566
 -0.6866666666666674
```

Without a comprehension, this loop would look something like:

```
shuffled_diffs = zeros(10_000)
for i in 1:length(shuffled_diffs)
    shuffled_differences[i] = simulate_differences(beer, water)
end
```

Back to the problem. Now let's plot the histogram. When you see a `!` after a function name in Julia, it means that this is a *mutating function*, which changes the object that it acts on, rather than returning a new object and preserving the old one. In this case, the plot object is changed to add new elements to the original histogram.