Prof. Kahle
STA 6376 - Computational Statistics II

STA 6376 Homework 6

1 Overview and Instructions.

This problem set is intended to introduce you to the bootstrap and its variants and exercise your R coding skills. Your styling must conform to the tidyverse style guide. Echo all your code so that the code and output are present. You will work in teams of two on this assignment, but you must each turn in your own assignment. You may not look online for any aid apart from the direct links provided in this document. You also may not use past assignments or solutions. You may use textbooks, and you may use RStudio Cheat Sheets. Please come to me with any other questions!

```
library("tidyverse"); theme_set(theme_minimal())
library("parallel"); options(mc.cores = detectCores())
```

Please compile your document into a PDF using RMarkdown and upload the PDF to Canvas with the filename in the following format: `[surname]_hw6.pdf`. For example, I would turn in `kahle_hw6.pdf`. You only need to include the answers in your turn-in; you don't need to type out the questions.

2 Questions.

1. **Bootstrap CIs.** In this exercise we consider various bootstrap strategies. Recall the following points.

   (a) Let $F_{\boldsymbol{X}}$ be a distribution on $\mathbb{R}^p$. If we can freely generate IID observations from $F_{\boldsymbol{X}}$, we can determine up to a arbitrary degree of accuracy any quantity related to $F_{\boldsymbol{X}}$ for which we have a consistent estimator.

   (b) A *statistical functional* is simply any quantity that can be calculated given a distribution $F_{\boldsymbol{X}}$; i.e. a statistical functional $\mathtt{T}(F_{\boldsymbol{X}})$ is a function of $F_{\boldsymbol{X}}$. Many common summaries of distributions can be written this way. For example, if g is a function and $\boldsymbol{\mathcal{X}}$ is the sample space of $\boldsymbol{X}$, the expectation is defined

   $$\mathbb{E}\left[g(\boldsymbol{X})\right] = \int g(\boldsymbol{x})\ dF_{\boldsymbol{X}} = \begin{cases} \sum_{\boldsymbol{x}\in\boldsymbol{\mathcal{X}}} g(\boldsymbol{x}) f_{\boldsymbol{X}}(\boldsymbol{x}) & \text{if } \boldsymbol{X} \text{ is discrete, and} \\ \int_{\boldsymbol{x}\in\boldsymbol{\mathcal{X}}} g(\boldsymbol{x}) f_{\boldsymbol{X}}(\boldsymbol{x})\ d\boldsymbol{x} & \text{if } \boldsymbol{X} \text{ is continuous,} \end{cases}$$

   so any quantity that can be represented as an expectation is a functional of $F_{\boldsymbol{X}}$; this includes means, (co)variances, probabilities, etc. Note that the integral $\int g(\boldsymbol{x})\ dF_{\boldsymbol{X}}$ used here is the Lebesgue-Stieltjes integral used in integration and probability theory. It is a generalization of the Riemann integral you met in ordinary calculus in the sense that it allows you to integrate a much broader class of functions than the Riemann integral – specifically, functions where the limit of Riemann sums definition breaks down. The concept of the Riemann-Stieltjes integral, which also generalizes the Riemann integral but not as strongly as the Lebesgue variant, is more accessible if you're just seeing these things for the first time.

   (c) If $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_n \overset{iid}{\sim} F_{\boldsymbol{X}}$ are random p-vectors, the empirical CDF (ECDF) $\widehat{F}_n(\boldsymbol{x})$ is defined on $\mathbb{R}^p$ to be

   $$\widehat{F}_n(\boldsymbol{x}) = \frac{1}{n}\sum_{i=1}^{n} 1[\boldsymbol{X}_i \leq \boldsymbol{x}],$$

where $1[\boldsymbol{X}_i \leq \boldsymbol{x}]$ is the indicator function that assumes the value 1 when $\boldsymbol{X}_i \leq \boldsymbol{x}$, where $\leq$ is understood element-wise and all are true, and 0 otherwise. The distribution that corresponds to this CDF (the ECDF) is called the empirical distribution; it is a discrete distribution that puts $\frac{1}{n}$ probability at each of the observed random vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$.

Given a dataset $\mathbf{X} \in \mathbb{R}^{n \times p}$ of size n from some distribution $F_{\boldsymbol{X}}(\boldsymbol{x})$, "bootstrapping the statistic $\boldsymbol{T}(\boldsymbol{X})$" refers to the process of approximating the distribution of $\boldsymbol{T}$ using the following resampling strategy:

1. Take a simple random sample with replacement of size n from $\widehat{F}_n(\boldsymbol{x})$; denote the sample $\mathbf{X}^\star$. $\mathbf{X}^\star$ is referred to as a *bootstrap replicate* of $\mathbf{X}$.
2. Compute $\boldsymbol{T}(\mathbf{X}^\star)$ and record its value.
3. Repeat B times and look at the empirical distribution of the $\boldsymbol{T}$'s, referred to as the *bootstrap distribution of $\boldsymbol{T}$*. (It's actually a Monte Carlo approximation of that distribution, but it is common to hear it referred to the distribution itself.)

Very often in practice you will see people using the bootstrap to estimate the standard error of the statistic $\boldsymbol{T}$. Recall that the standard error of a statistic is simply its standard deviation, a measure of how much it may change from sample to sample. Estimating standard errors of a statistic is just one application of bootstrapping; the procedure is much more flexible in general. In particular, the bootstrap can be used to estimate the entire sampling distribution of any statistical functional.

This particular bootstrapping procedure is referred to as the *nonparametric bootstrap*. The idea follows from the so-called *plug-in principle*: to estimate a functional $\mathtt{T}$ of an unknown distribution $F_{\boldsymbol{X}}$, you should apply the same functional to the empirical distribution $\widehat{F}_n$: $\mathtt{T}(\widehat{F}_n)$ estimates $\mathtt{T}(F_{\boldsymbol{X}})$. While the distribution of $\mathtt{T}(F_{\boldsymbol{X}})$ may be difficult to obtain or even approximate (think: the exact distribution of the sample mean or correlation coefficient), arbitrarily good approximations to the exact distribution of $\mathtt{T}(\widehat{F}_n)$ are almost trivial via Monte Carlo methods: generate observations from the empirical distribution $\widehat{F}_n$, compute the statistic $\mathtt{T}(\widehat{F}_n)$, and repeat as many times as needed to get the desired level of accuracy. This is precisely what the nonparametric bootstrap does:

Resampling with replacement from the data is sampling from the empirical distribution $\widehat{F}_n$.

Let's look at an example. Suppose $X_1, \ldots, X_n \overset{iid}{\sim} \mathcal{N}(\mu, \sigma^2)$. By the above discussion, the mean $\mu = \mathbb{E}[X] = \int_{\mathbb{R}} x \, dF_X = \mathtt{T}_\mu(F_X)$ is a functional of the distribution of a random variable. Since $\widehat{F}_n$ is a discrete distribution, this same functional $\mathtt{T}_\mu$ applied to the empirical CDF yields

$$\mathtt{T}_\mu(\widehat{F}_n) = \int_{\mathbb{R}} x \, d\widehat{F}_n = \sum_{i=1}^{n} X_i \frac{1}{n} = \frac{1}{n} \sum_{i=1}^{n} X_i = \overline{X}_n,$$

Thus, the plug-in principle results in the estimator $\overline{X}_n$ for μ, since it dictates $\mathtt{T}_\mu(\widehat{F}_n)$ approximates $\mathtt{T}_\mu(F_X)$.

(a) Write a function `boot_pct_ci()` that accepts a vector `x` (considered a univariate dataset), `conf = .95`, and `B = 1e3`, the number of bootstrap replicates to use, and returns the *percentile bootstrap confidence interval*: the estimated 2.5 and 97.5 percentiles of the bootstrap distribution of the mean (for a 95% CI). Then, run your function on the code below to show it works.

Hint: recall `sample()` and `quantile()`.

```
set.seed(1)
(data <- rnorm(20, mean = 5, sd = 3)) %>% round(2)

#  [1]  3.12  5.55  2.49  9.79  5.99  2.54  6.46  7.21  6.73  4.08  9.54
# [12]  6.17  3.14 -1.64  8.37  4.87  4.95  7.83  7.46  6.78
```

```
data %>% boot_pct_ci() %>% round(2)

# [1] 4.23 6.69
```

(b) The standard interval for this scenario would be the t-interval $\overline{X}_n \pm t_{n-1,\alpha/2}\frac{S}{\sqrt{n}}$. Write a function `t_ci()` with arguments `x` and `conf = .95` that computes the t-interval for a given vector `x`. Do not use the function `t.test()`. Then, run your function on the code below to show it works.

```
data %>% t_ci() %>% round(2)

# [1] 4.29 6.85

data %>% t.test() %>% pluck("conf.int") %>% round(2)

# [1] 4.29 6.85
# attr(,"conf.level")
# [1] 0.95
```

(c) You can read about the "bias corrected and adjusted (for acceleration)," or BCa for short, bootstrap interval in Rizzo Section 7.5. Two common bootstrapping packages are **boot** and **resample**. Written by Tim Hesterberg at Google, the **resample** package contains several functions for resampling in general, see `ls("package:resample")`. The function `CI.bca()` is what you want to use, but you can't use it directly; see `?CI.bca` to see how you use it. Write a wrapper function `boot_bca_ci()` that accepts the same arguments as `boot_pct_ci()` and passes them to `resample::CI.bca()` to compute the interval of interest. Be sure that the output is exactly the same kind of data structure as that returned by `boot_pct_ci()`, a `numeric(2)` with no attributes. Then, run your function on the code below to show it works.

```
data %>% boot_bca_ci() %>% round(2)

# [1] 4.20 6.77
```

(d) Construct a simulation study to address the following question: For a sample of size 250 from the Weibull $(1/5, 5)$ (mean 600), what is the coverage of a (nominal) 95% confidence interval for μ using the bootstrap percentile, standard t, and BCa intervals? Use at least 10,000 Monte Carlo iterations.

Hint: As always, start by writing code that does it once: generate a dataset, compute the interval, and check if it contains the true value of μ for the Weibull $(1/5, 5)$ distribution (600). Then do that a bunch of times, and compute the proportion of the intervals that contain μ. You may parallelize the computation.

There are many variants of the bootstrap. The *parametric bootstrap* is a resampling procedure that combines aspects of the bootstrap with parametric modeling. Assuming a parametric model $\mathcal{M} = \{F_{\boldsymbol{\theta}}\}$ indexed by $\boldsymbol{\theta} \in \boldsymbol{\Theta} \subset \mathbb{R}^p$, given the dataset $\mathbf{X}$ the parametric bootstrap approximates the distribution of a statistic $\boldsymbol{T}(\boldsymbol{X})$ with the following procedure:

1. Fit the model, i.e. estimate $\boldsymbol{\theta}$ with $\widehat{\boldsymbol{\theta}}$, using your estimator of choice.
2. Take a IID sample of size n from $F_{\widehat{\boldsymbol{\theta}}}(\boldsymbol{x})$; denote the sample $\mathbf{X}^\star$. $\mathbf{X}^\star$ is referred to as a *parametric bootstrap replicate* of $\mathbf{X}$.

(a) Compute $\boldsymbol{T}(\mathbf{X}^\star)$ and record its value.

(b) Repeat B times to generate the (approximate) parametric bootstrap distribution of $\boldsymbol{T}$; this estimates the true unknown distribution of $\boldsymbol{T}$.

Thus, the parametric bootstrap is just like the nonparametric bootstrap, but instead of sampling from the empirical distribution $\widehat{F}_n$ (i.e. resampling the data), it samples from an estimate of $F_{\boldsymbol{X}}$ that is in the model, namely $F_{\widehat{\boldsymbol{\theta}}}$.

Another variant of the bootstrap is called the *smooth bootstrap*. The smooth bootstrap is used in situations where $F_{\boldsymbol{X}}$ is known to be a continuous distribution. It tries to address the oddity of the idea that the nonparametric bootstrap estimates the distribution $F_{\boldsymbol{X}}$, which is continuous, with the empirical distribution $\widehat{F}_n$, which is discrete, when trying to determine the distribution of $\boldsymbol{T}$. It does this by smoothing the empirical distribution into a continuous distribution. One way to do this is to estimate the joint PDF $f_{\boldsymbol{X}}(\boldsymbol{x})$ with a kernel density estimator and then let $\widehat{F}_S(\boldsymbol{x}) = \int_{-\infty}^{\boldsymbol{x}} \widehat{f}(\boldsymbol{t})\ d\boldsymbol{t} = \int_{-\infty}^{x_1} \cdots \int_{-\infty}^{x_p} \widehat{f}(t_1, \ldots, t_p)\ dt_p \cdots dt_1$ be the CDF corresponding to that KDE.

For example, suppose `data` contains continuous data assumed to have come from a normal distribution $\mathcal{N}(\mu, \sigma^2)$, with both parameters unknown, and you are interested in the distribution of the sample median defined

$$\boldsymbol{T}(\mathbf{X}) = T(X_1, \ldots, X_n) = \begin{cases} X_{(\frac{n+1}{2})} & n \text{ is odd} \\ \frac{1}{2}\left(X_{(\frac{n}{2})} + X_{(\frac{n}{2}+1)}\right) & n \text{ is even,} \end{cases}$$

where $X_{(k)}$ is the *k*th order statistic. Without going through the theory, you can approximate the distribution with bootstrap resampling. Here's how the three strategies will do this:

1. The nonparametric bootstrap resamples the data directly, computing the median every time, and then uses the empirical distribution of those sample medians.
2. The parametric bootstrap fits the normal distribution, for example with $\mu = \overline{X}_n$ and $\sigma^2 = S^2$ (here $=$ denotes "is estimated with"), draws samples from that normal distribution, computes the median of each, and uses the empirical distribution of those sample medians.
3. The smooth bootstrap resamples the data a bunch of times, each time adding a small amount of normal noise to every value in the resampled dataset, computes the median of each, and looks at the distribution of those medians. Note that the [resample the data + add noise] part is equivalent to sampling from the distribution given by the kernel density estimate of the data, where the distribution of the noise is the kernel. This is a mixture distribution; typically a mixture of normals, since the kernel is typically a normal.

With these in mind, answer the following questions.

(a) Write a function called `kdefun()` with formal arguments `x` and `sd = .1` and returns a function, the kernel density estimate $\widehat{f}_S(x)$. (Here assume $p = 1$; the data is univariate.)
Hint: This is at most a two-liner. Use `density()` to compute a set of values (x_k, y_k) with $y_k = \widehat{f}_S(x_k)$ and then `approxfun()` to create a function from those values that simply interpolates linearly between neighboring x_k's. The result of the latter won't be a PDF exactly, but it'll be close enough for us. Be sure to use the `yleft` and `yright` arguments of `approxfun()`.

(b) Using `kdefun()` and `stat_function()`, plot the kernel density estimate of `data` over the range $(-4, 14)$. Don't use `geom_density()`.

(c) Integrate the KDE returned by `kdefun()` on the dataset `data` to show that it's (roughly) a PDF.
Hint: `integrate()`.

(d) The function `stats::ecdf()` accepts a numeric vector `x` and returns a vectorized function that is the empirical CDF of `x`. Write your own function `ecdf2()`, also with formal argument `x` (a numeric vector), that returns a vectorized function that computes the empirical CDF. Do not use `approxfun()`. Then, run your function on the code below to show it works.

```
s <- seq(.5, 2.5, .5)
ecdf(1:2)(s)

# [1] 0.0 0.5 0.5 1.0 1.0

ecdf2(1:2)(s)

# [1] 0.0 0.5 0.5 1.0 1.0
```

(e) Create a function `secdf()` with formal argument `x` that returns a smoothed version of the empirical CDF, $\widehat{F}_S(x)$. The function should be vectorized across `x`. Then, run your function on the code below to show it works.

```
pnorm(-3:3) %>% round(4)

# [1] 0.0013 0.0228 0.1587 0.5000 0.8413 0.9772 0.9987

secdf(rnorm(1e6))(-3:3) %>% round(4)

# [1] 0.0013 0.0228 0.1591 0.5012 0.8423 0.9782 0.9996
```

(f) Graph the three CDFs ($\widehat{F}_n(x)$, $\widehat{F}_S(x)$, and $\widehat{F}_{\widehat{\boldsymbol{\theta}}}(x)$) on the same plot, superimposed, over the range $(-4, 14)$.

Hint : To use **ggplot2** to plot only functions over a given range, you can use `ggplot(data_frame(x = c(-4, 14)), aes(x)) + ...`, then use `stat_function()`.

(g) Write a function `boot_param_ci()` with formal arguments `x`, `conf = .95`, and `B = 1e3` that creates a parametric bootstrap percentile confidence interval for the population mean μ assuming a normal model. Then, run your function on the code below to show it works.

```
data %>% boot_param_ci() %>% round(2)

# [1] 4.38 6.73
```

(h) Write a function `boot_smooth_ci()` with formal arguments `x`, `conf = .95`, and `B = 1e3` that creates a smooth bootstrap percentile confidence interval for the population mean μ. For the noise distribution, use a mean-zero normal with standard deviation equal to `bw.nrd0(x)`. (That's what `density()` uses by default.) Then, run your function on the code below to show it works.

```
data %>% boot_smooth_ci() %>% round(2)

# [1] 4.26 6.74
```

(i) Repeat the simulation in (d) above using all five intervals and datasets of size 250 from the Weibull $(1/5, 5)$ distribution.