HW3

110077443

3/5/2022

Please note that all code in this document is presented in a grey box and the output reflected below each box

- The below code allows lengthy lines of code to display neatly within the grey box (wrapping it)

```
knitr::opts_chunk$set(tidy.opts = list(width.cutoff = 60), tidy = TRUE)
```

1) Standardize Our Data

a) Create a normal distribution (mean=940, sd=190) and standardize it.

```
norm_dist <- rnorm(n = 10000, mean = 940, sd = 190)  #Create normal distribution
norm_dist_mean <- mean(norm_dist)  #Mean of normal data to be used for standardizing formula
norm_dist_sd <- sd(norm_dist)  #Standard deviation of normal data to be used for standardizing formula
rnorm_std <- (norm_dist - norm_dist_mean)/norm_dist_sd  # formula for vector of standardizing values
```

i) What should we expect the mean and standard deviation of rnorm_std to be, and why?

- We should expect the **mean = 0** and the **standard deviation = 1** because we are centering the variables at zero and standardizing the variance at 1 by subtracting the mean of each observation and then dividing by the standard deviation. Thus, the values that were exactly 1 standard deviation away from the mean will equal to 1 after standardizing the data.
- Proof of values from code below;

```
mean(rnorm_std)
```

```
## [1] 6.351067e-17
```

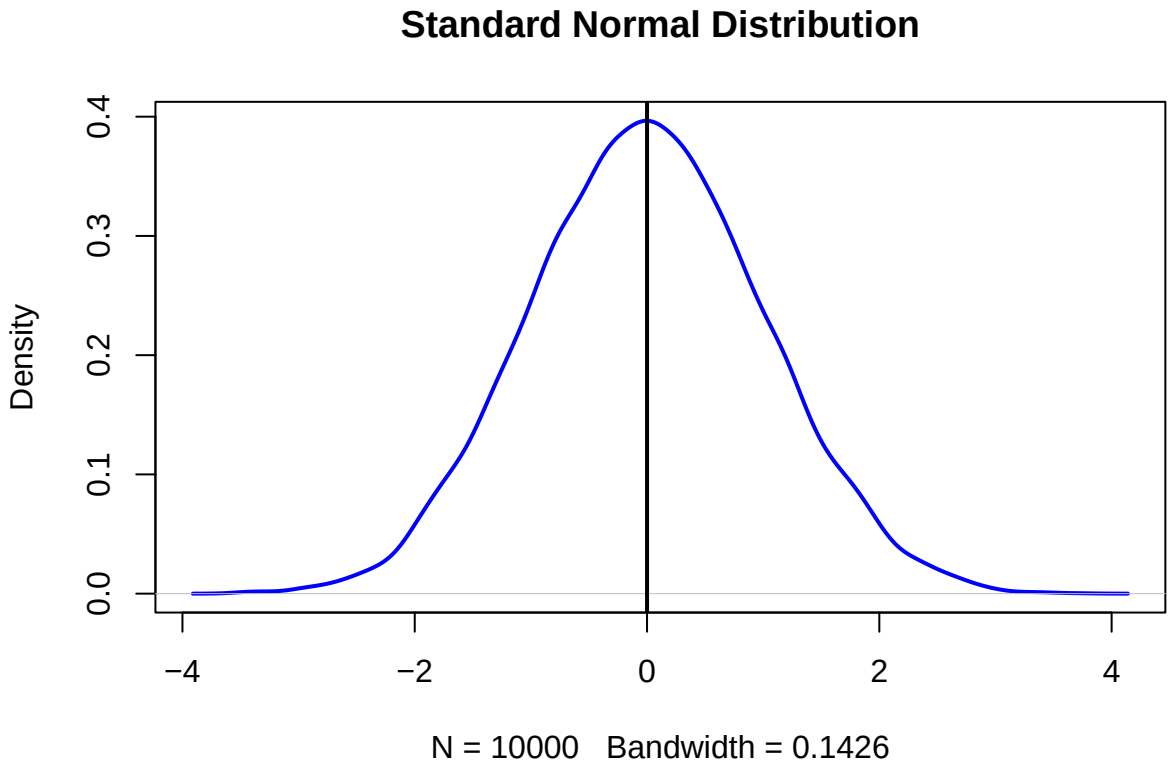
```
sd(rnorm_std)
```

```
## [1] 1
```

ii) What should the distribution (shape) of rnorm_std look like, and why?

- The shape of rnorm_std should be a **Bell Curve Shape** because the data is normally distributed, meaning the data near the mean is more frequent in occurrence than data far from the mean. The data is **symmetric** from the mean as seen below:

```
plot(density(rnorm_std), lwd = 2, col = "blue", main = "Standard Normal Distribution")
abline(v = mean(rnorm_std), lwd = 2)
```



iii) What do we generally call distributions that are normal and standardized?

- We generally refer to normal and standardized distributions as **Standard Normal Distribution** or **z-distribution**.

b) Create a standardized version of minday discussed in question 3

- Let's first import the data and format it accordingly:

```
bookings <- read.table("first_bookings_datetime_sample.txt",
    header = TRUE)
bookings$datetime[1:9]
```

```
## [1] "4/16/2014 17:30"  "1/11/2014 20:00"  "3/24/2013 12:00"  "8/8/2013 12:00"
```

```
## [5] "2/16/2013 18:00"  "5/25/2014 15:00"  "12/18/2013 19:00" "12/23/2012 12:00"
## [9] "10/18/2013 20:00"
```

```
hours <- as.POSIXlt(bookings$datetime, format = "%m/%d/%Y %H:%M")$hour
mins <- as.POSIXlt(bookings$datetime, format = "%m/%d/%Y %H:%M")$min
minday <- hours * 60 + mins
```

- Now we will standardize minday and call it "minday_std".

```
minday_mean <- mean(minday)  #Mean of minday data to be used for standardizing formula
minday_sd <- sd(minday)  #Standard deviation of normal data to be used for standardizing formula
minday_std <- (minday - minday_mean)/minday_sd  # formula for vector of standardizing values
```

i) What should we expect the mean and standard deviation of minday_std to be, and why?

- We should expect the **mean = 0** and the **standard deviation = 1** because we are standardizing the minday data, meaning that the values that were exactly 1 standard deviation away from the mean will now eual to 1.
- Proof of values from code below:

```
mean(minday_std)
```

```
## [1] -4.25589e-17
```

```
sd(minday_std)
```

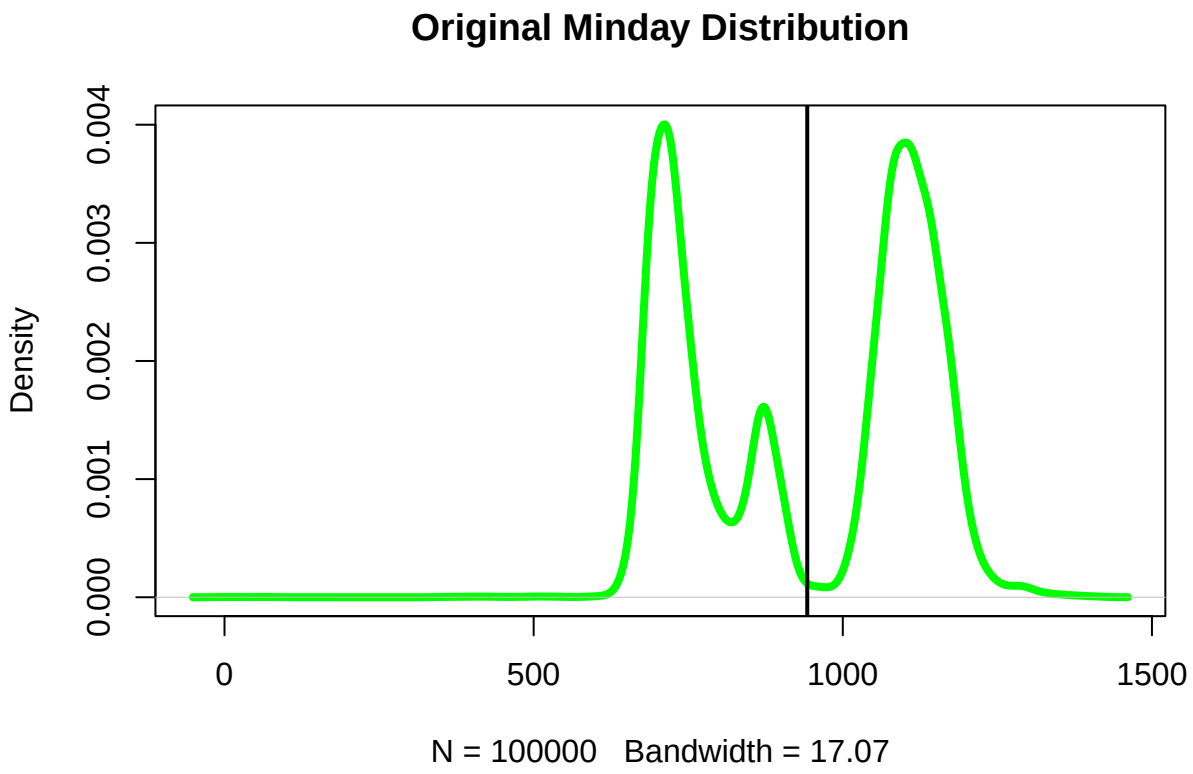
```
## [1] 1
```

ii) What should the distribution of minday_std look like compared to minday, and why?

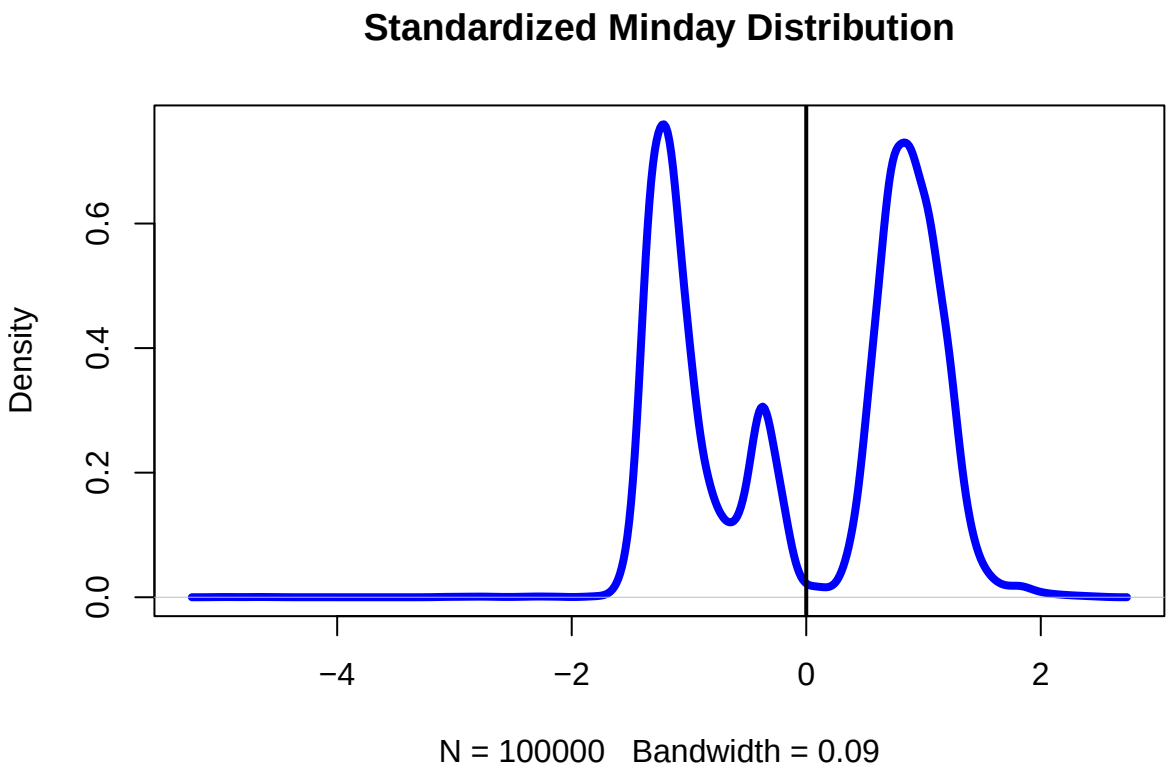
The shape of minday_std should be **identical** to the minday because standardizing the data only shifts and scales the distribution, but does not change it's shape.

- We can see the two plots in comparison below:

```
plot(density(minday), lwd = 4, col = "green", main = "Original Minday Distribution")
abline(v = mean(minday), lwd = 2)
```




```
plot(density(minday_std), lwd = 4, col = "blue", main = "Standardized Minday Distribution")
abline(v = mean(minday_std), lwd = 2)
```



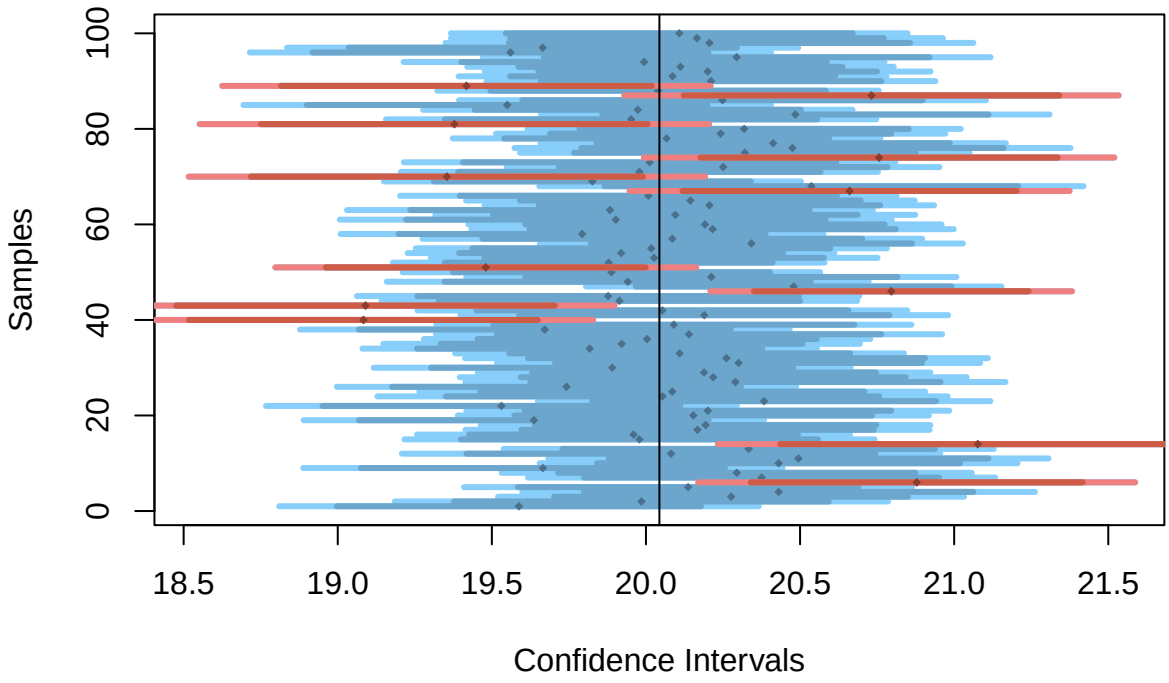
2) Simulate samples drawn randomly from a population using the code given from the following source: "https://gist.github.com/soumyaray/285296600b8712b04b52201010bbbd9f"

```
# Visualize the confidence intervals of samples drawn from
# a population e.g., visualize_sample_ci(sample_size=300,
# distr_func=rnorm, mean=50, sd=10)
# visualize_sample_ci(sample_size=300, distr_func=runif,
# min=17, max=35)

visualize_sample_ci <- function(num_samples = 100, sample_size = 100,
    pop_size = 10000, distr_func = rnorm, ...) {
    # Simulate a large population
    population_data <- distr_func(pop_size, ...)
    pop_mean <- mean(population_data)
    pop_sd <- sd(population_data)

    # Simulate samples
    samples <- replicate(num_samples, sample(population_data,
        sample_size, replace = FALSE))

    # Calculate descriptives of samples
```

```
    sample_means = apply(samples, 2, FUN = mean)
    sample_stdevs = apply(samples, 2, FUN = sd)
    sample_stderrs <- sample_stdevs/sqrt(sample_size)
    ci95_low <- sample_means - sample_stderrs * 1.96
    ci95_high <- sample_means + sample_stderrs * 1.96
    ci99_low <- sample_means - sample_stderrs * 2.58
    ci99_high <- sample_means + sample_stderrs * 2.58

    # Visualize confidence intervals of all samples
    plot(NULL, xlim = c(pop_mean - (pop_sd/2), pop_mean + (pop_sd/2)),
        ylim = c(1, num_samples), ylab = "Samples", xlab = "Confidence Intervals")
    add_ci_segment(ci95_low, ci95_high, ci99_low, ci99_high,
        sample_means, 1:num_samples, good = TRUE)

    # Visualize samples with CIs that don't include
    # population mean
    bad = which(((ci95_low > pop_mean) | (ci95_high < pop_mean)) |
        ((ci99_low > pop_mean) | (ci99_high < pop_mean)))
    add_ci_segment(ci95_low[bad], ci95_high[bad], ci99_low[bad],
        ci99_high[bad], sample_means[bad], bad, good = FALSE)

    # Draw true population mean
    abline(v = mean(population_data))
}

add_ci_segment <- function(ci95_low, ci95_high, ci99_low, ci99_high,
    sample_means, indices, good = TRUE) {
    segment_colors <- list(c("lightcoral", "coral3", "coral4"),
        c("lightskyblue", "skyblue3", "skyblue4"))
    color <- segment_colors[[as.integer(good) + 1]]

    segments(ci99_low, indices, ci99_high, indices, lwd = 3,
        col = color[1])
    segments(ci95_low, indices, ci95_high, indices, lwd = 3,
        col = color[2])
    points(sample_means, indices, pch = 18, cex = 0.6, col = color[3])
}
```

a) Simulate 100 samples (each of size 100), from a normally distributed population of 10,000

```
visualize_sample_ci(num_samples = 100, sample_size = 100, pop_size = 10000,
    distr_func = rnorm, mean = 20, sd = 3)
```



i) How many samples do we expect to NOT include the population mean in its 95% CI?

- If we created 100 confidence intervals of the same size from the same population, we would expect 95 of them to contain the true parameter (the population mean weight). Thus, we expect around **5** of the intervals **NOT** to contain the population mean as seen below:

```
num_samples95 <- 100
num_samples95 * 0.05
```

```
## [1] 5
```

ii) How many samples do we expect to NOT include the population mean in their 99% CI?

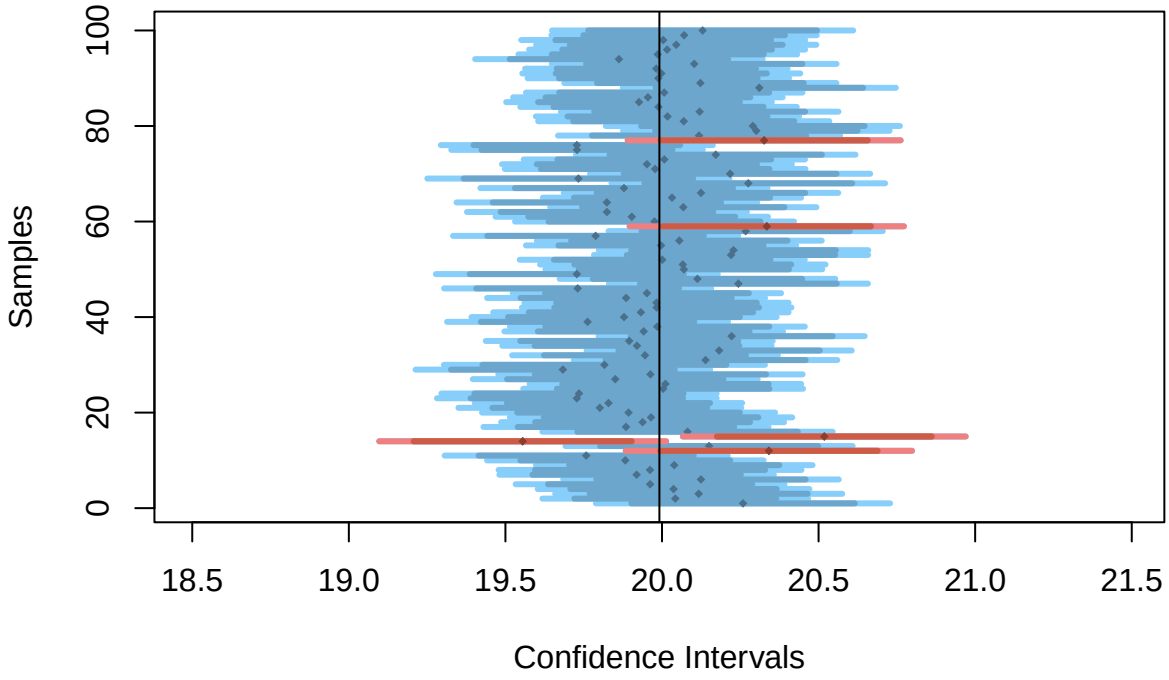
- This would be 1% of the number of samples which in this case is **1** as seen below:

```
num_samples99 <- 100
num_samples99 * 0.01
```

```
## [1] 1
```

b) Rerun the previous simulation with a larger sample size of 300:

```
visualize_sample_ci(num_samples = 100, sample_size = 300, pop_size = 10000,
    distr_func = rnorm, mean = 20, sd = 3)
```



i) Do we expect their 95% and 99% CI to become wider or narrower than before?

- We can expect their 95% and 99% CI to become **narrower** because as the sample size increases, the more accurately we can estimate the unknown parameters. Thus, if we can be more confident in our estimates, the narrower our confidence interval.

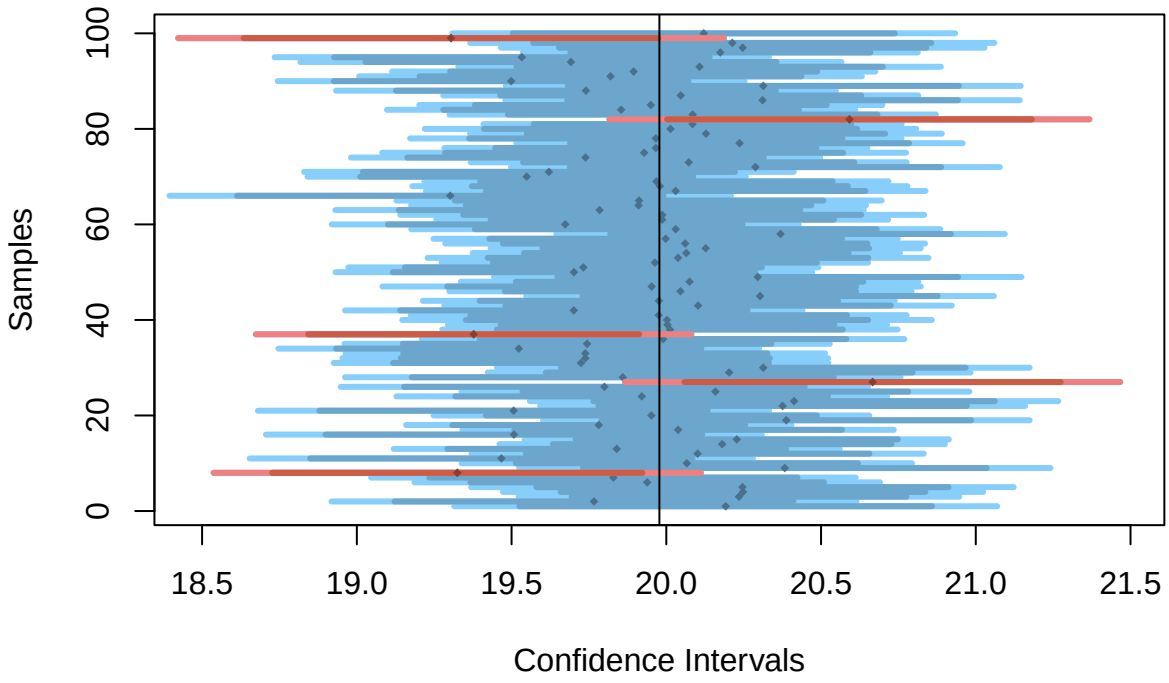
ii) how many samples would we expect to NOT include the population mean in its 95% CI?

- This time we still expect around **5** of the intervals **NOT** to contain the population mean because we are still drawing all the samples of same size. The difference with a greater sampling size is that the CI will be narrower.

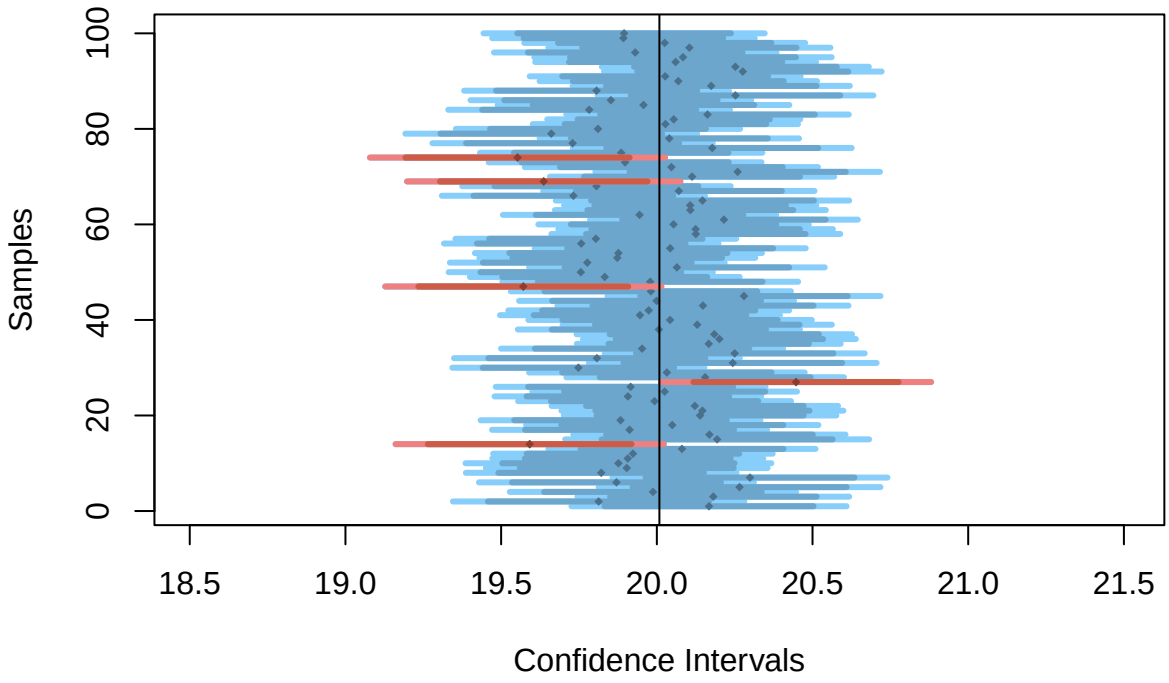
c) Using a uniformly distributed population, what changes do we see?

- Let's first look at the normally distributed population of 10,000 with different sample sizes:

```
visualize_sample_ci(num_samples = 100, sample_size = 100, pop_size = 10000,
    distr_func = rnorm, mean = 20, sd = 3)
```

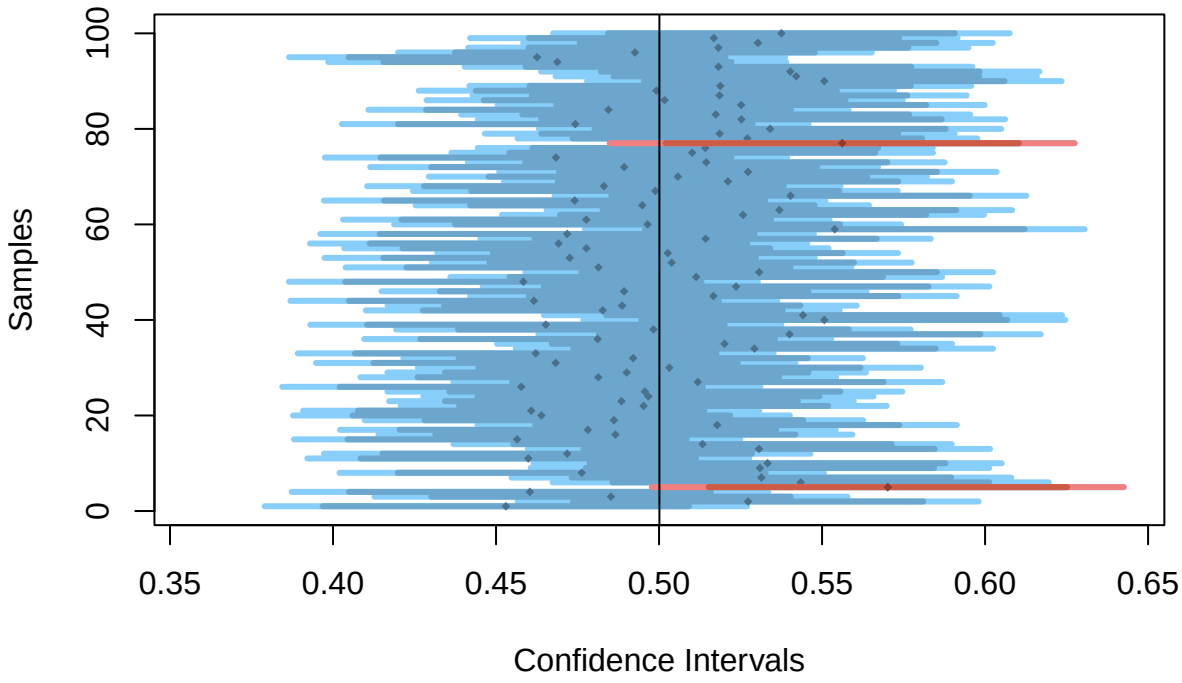



```
visualize_sample_ci(num_samples = 100, sample_size = 300, pop_size = 10000,
    distr_func = rnorm, mean = 20, sd = 3)
```

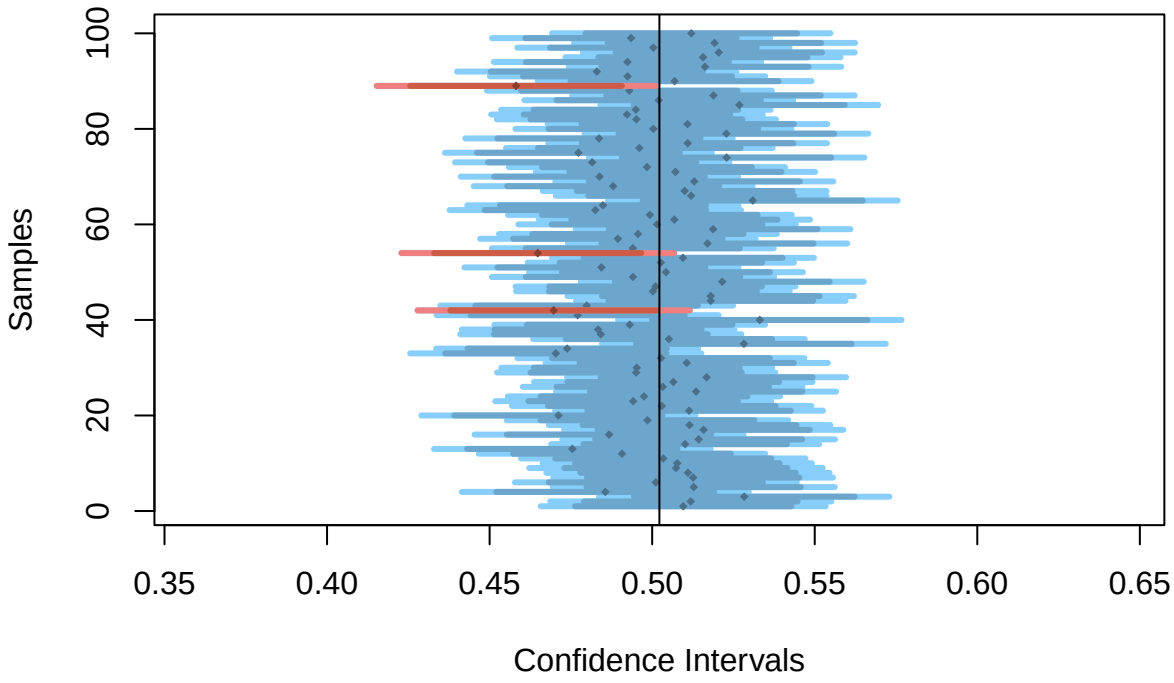


- We can see that the larger the sample size, the narrower the CI gets.
- Now let's look at a uniformly distributed population of 10,000 with different sample sizes:

```
visualize_sample_ci(num_samples = 100, sample_size = 100, pop_size = 10000,
    distr_func = runif)
```




```
visualize_sample_ci(num_samples = 100, sample_size = 300, pop_size = 10000,
    distr_func = runif)
```



- We can see that the same principle applies with the uniformly distributed population.
- There is no clear difference between rnorm and runif distributions.
- Central Limit Theorem states that regardless of the type of distribution, the larger the sampling size gets, the more the parameters behave like that in a normal distribution.

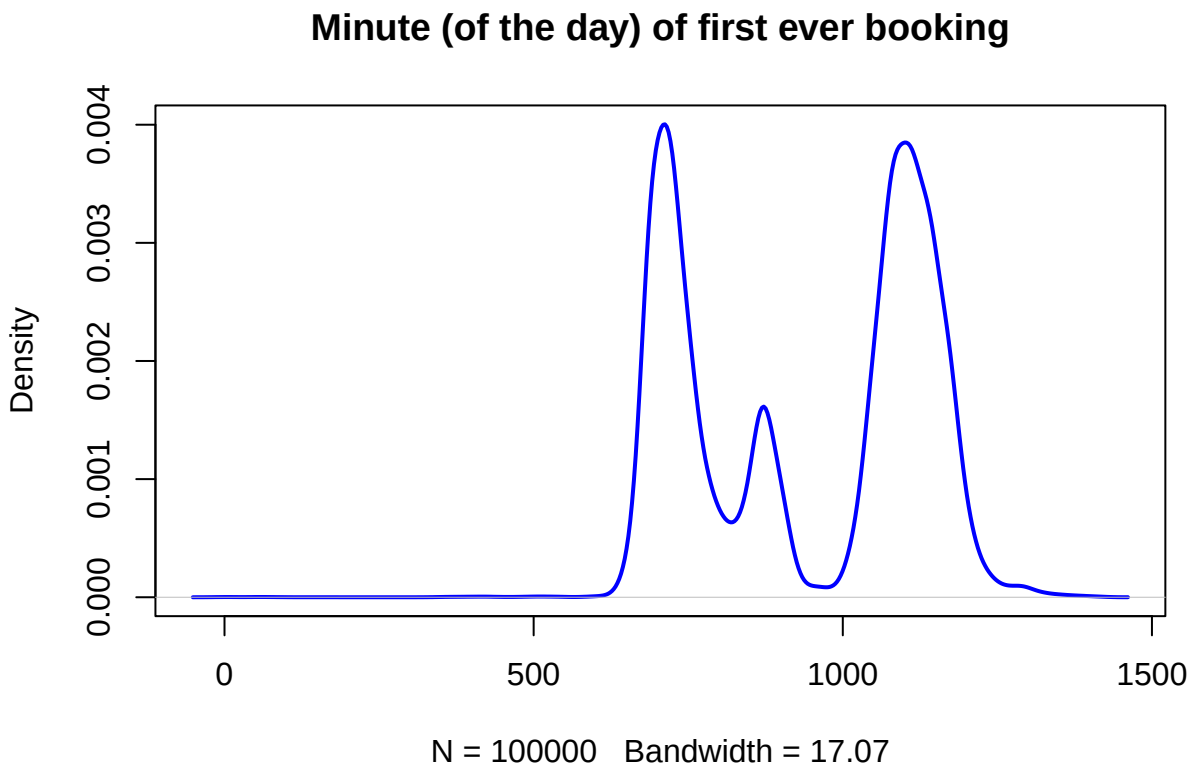
3) EZTABLE sample data (date and time) of new members first ever booking.

- Let's first import the data and format it accordingly:

```
bookings <- read.table("first_bookings_datetime_sample.txt",
    header = TRUE)
bookings$datetime[1:9]
```

```
## [1] "4/16/2014 17:30"  "1/11/2014 20:00"  "3/24/2013 12:00"  "8/8/2013 12:00"
## [5] "2/16/2013 18:00"  "5/25/2014 15:00"  "12/18/2013 19:00" "12/23/2012 12:00"
## [9] "10/18/2013 20:00"
```

```
hours <- as.POSIXlt(bookings$datetime, format = "%m/%d/%Y %H:%M")$hour
mins <- as.POSIXlt(bookings$datetime, format = "%m/%d/%Y %H:%M")$min
minday <- hours * 60 + mins
plot(density(minday), main = "Minute (of the day) of first ever booking",
    col = "blue", lwd = 2)
```



a) The "average" booking time for new members making their first booking

```
minday_mean  #Average booking time
```

```
## [1] 942.4964
```

###i) Use traditional statistical methods to estimate population mean, SE, and CI.

```
minday_mean <- sum(minday)/length(minday)  #Mean
minday_mean  #print
```

```
## [1] 942.4964
```

```
minday_se <- sd(minday)/(sqrt(length(minday)))  #Standard error
minday_se  #print
```

```
## [1] 0.5997673
```

```
minday_ci95 <- minday_mean + c(1.96, -1.96) * minday_se  #95% CI
minday_ci95  #print
```

```
## [1] 943.6719 941.3208
```

- We are 95% confident that the population mean is between **941.3208 and 943.6719.**
- Checking to see if traditional methods match with pastecs package:

```
require(pastecs)
```

```
## Loading required package: pastecs
```

```
stat.desc(minday)  #print
```

```
##      nbr.val     nbr.null       nbr.na          min          max        range
## 1.000000e+05 6.000000e+00 0.000000e+00 0.000000e+00 1.410000e+03 1.410000e+03
##          sum       median         mean      SE.mean CI.mean.0.95          var
## 9.424964e+07 1.040000e+03 9.424964e+02 5.997673e-01 1.175537e+00 3.597208e+04
##      std.dev     coef.var
## 1.896631e+02 2.012348e-01
```

- Values match

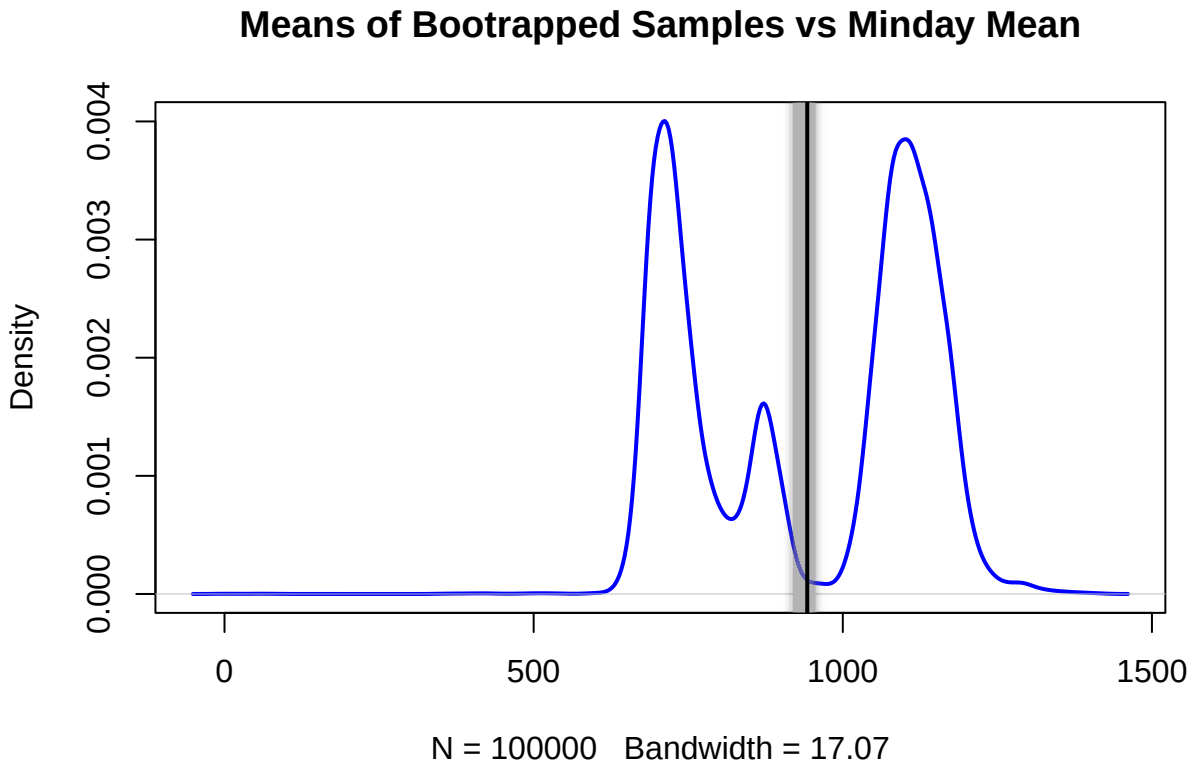
ii) Bootstrap to produce 2000 new samples from the original sample

```
# Preparing and creating new sample
compute_sample_mean <- function(sample0) {
    resample <- sample(sample0, length(sample0), replace = TRUE)
    mean(resample)
}
mindays_mean <- compute_sample_mean(minday)  #New sample mean

# Take one sample from minday' sample
minday_sample_size <- 300  #Creating sample size
minday_sample0 <- sample(minday, minday_sample_size)  #Creating 1st sample from population

# Bootstrapping to produce 2000 new samples
minday_resamples <- replicate(2000, sample(minday_sample0, length(minday_sample0),
    replace = TRUE))
```

iii) Visualize the means of the 2000 bootstrapped samples

```
# Plot population and original sample densities
plot(density(minday), col = "blue", lwd = 2, main = "Means of Bootrapped Samples vs Minday Mean")

# Draws lines for each sampling mean
plot_resample_mean <- function(sample_i) {
    abline(v = mean(sample_i), col = rgb(0.7, 0.7, 0.7, 0.01))
    return(mean(sample_i))
}
```

```r
# Plot and get means of all bootstrapped samples
sample_means <- apply(minday_resamples, 2, FUN = plot_resample_mean)

# Draw lines of original sample mean
abline(v = mean(minday), lwd = 2)
```



iv) Estimate the 95% CI of the bootstrapped means

```r
quantile(sample_means, probs = c(0.05, 0.95))
```

```
##       5%      95%
## 920.3325 954.7217
```

b) By what time of day, have half the new members of the day already arrived?

- The median will provide the time of day for half the new members of the day arriving. The answer is **1040**

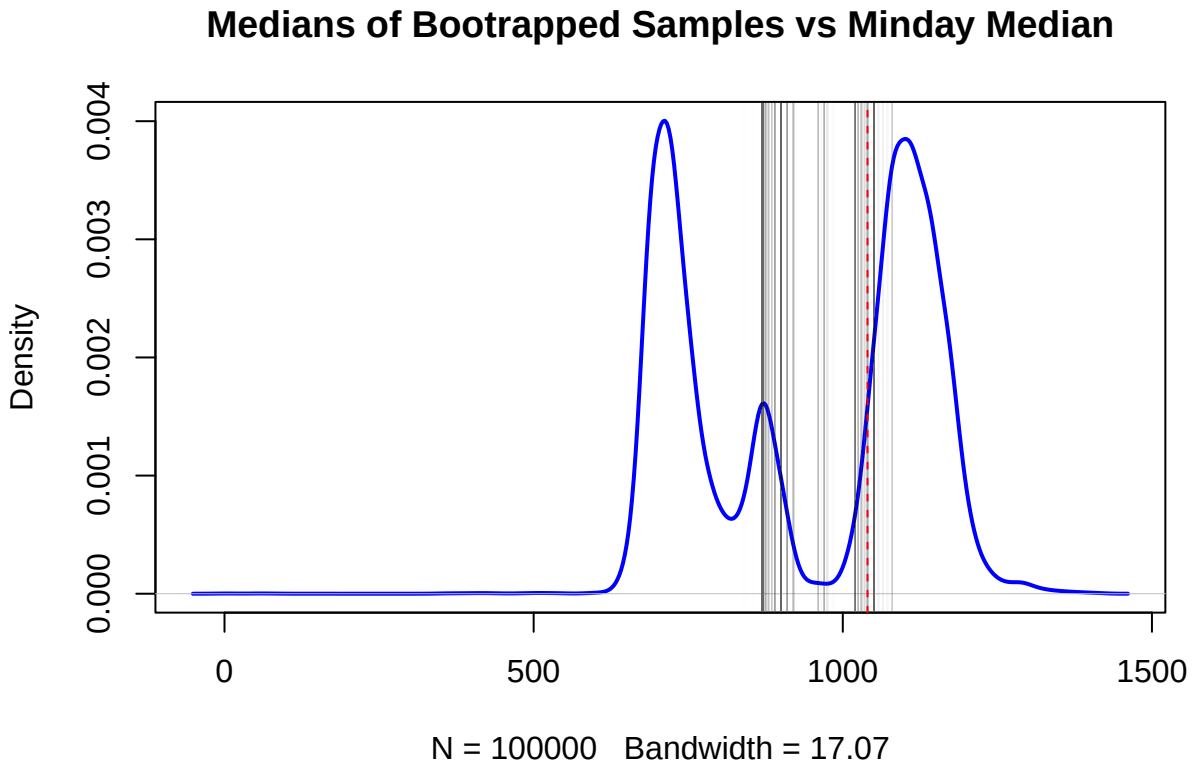
i) Estimate the median of minday

```r
minday_sorted <- sort(minday)
median(minday_sorted)
```

```
## [1] 1040
```

ii) Visualize the medians of the 2000 bootstrapped samples

```r
# Plot population and original sample densities
plot(density(minday), col = "blue", lwd = 2, main = "Medians of Bootrapped Samples vs Minday Median")

# Draws lines for each sampling mean
plot_resample_median <- function(sample_i) {
    abline(v = median(sample_i), col = rgb(0.4, 0.4, 0.4, 0.01))
    return(mean(sample_i))
}

# Plot and get means of all bootstrapped samples
sample_median <- apply(minday_resamples, 2, FUN = plot_resample_median)

# Draw lines of original sample mean
abline(v = median(minday), lty = "dashed", col = "red")
```



iii) Estimate the 95% CI of the bootstrapped medians

```
quantile(sample_median, probs = c(0.05, 0.95))
```

```
##       5%      95%
## 920.3325 954.7217
```