Problem Set 2: Summarize global cesarean delivery rates and GDP across 137 countries

Your name and student ID

February 01, 2023

Instructions

- Solutions will be released by Wednesday, February 1st.
- This semester, problem sets are for practice only and will not be turned in for marks.

Helpful hints:

- Every function you need to use was taught during lecture! You may need to revisit the lecture code to help you along by opening the relevant files on Datahub. Alternatively, you may wish to view the code in the condensed PDFs posted on the course website. Good luck!
- Knit your file early and often to minimize knitting errors! If you copy and paste code for the slides, you are bound to get an error that is hard to diagnose. Typing out the code is the way to smooth knitting! We recommend knitting your file each time after you write a few sentences/add a new code chunk, so you can detect the source of knitting errors more easily. This will save you and the GSIs from frustration!
- If your code runs off the page of the knitted PDF, be sure to fix this! If it doesn't look right, go back to your .Rmd file and add spaces (new lines) using the return or enter key so that the code runs onto the next line.

Summarizing global cesarean delivery rates and GDP across 137 countries

Introduction

Recall from this week's lab that we constructed bar charts and histograms to explore a data set that looked at global rates of cesarean delivery and GDP. If you need a refresher, you can view your knitted file from lab and remind yourself about what you found.

In this week's problem set, you will describe these distributions using numbers. You will investigate the **mean** and **median** of the distribution of GDP. You will also examine the distribution of cesarean delivery separately for countries of varying income levels. Lastly, you will describe the **spread** of the distributions using **quartiles** and make a **box plot**.

Execute this code chunk to load the required libraries:

```
library(readr)
```

```
##
## Attaching package: 'readr'

## The following objects are masked from 'package:testthat':
##
##     edition_get, local_edition
```

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following object is masked from 'package:testthat':
##
##     matches

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(ggplot2)
```

Just like in lab, read in the data that is stored as a .csv file and assign it to an object called `CS_data`. We will also use dplyr's `mutate()` to create the new cesarean delivery variable that ranges between 0 and 100:

```r
CS_data <- read_csv("data/cesarean.csv")
```

```
## Rows: 137 Columns: 7
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (4): Country_Name, CountryCode, Income_Group, Region
## dbl (3): Births_Per_1000, GDP_2006, CS_rate
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
# The code below reorders the Factor variable `Income_Group` in the
# order specified in this function. This will affect the order the ggplot
# panels are shown in question 8 when we use `facet_wrap()`.
CS_data$Income_Group <- forcats::fct_relevel(CS_data$Income_Group,
                                             "Low income", "Lower middle income",
                                             "Upper middle income", "High income: nonOECD",
                                             "High income: OECD")

CS_data <- CS_data %>% mutate(CS_rate_100 = CS_rate*100)
```

1. [1 point] Fill in the blanks indicated by "—-" by saving the answer to each blank in the code chunk below. Make sure you capitalize correctly, as R is case-sensitive.

The function `mutate()` takes the old variable called –**(aa)**– and multiplies it by –**(bb)**– to make a new variable called –**(cc)**–.

```
. = " # BEGIN PROMPT
aa <- 'your answer here'
bb <- 'your answer here'
cc <- 'your answer here'
" # END PROMPT

# BEGIN SOLUTION
aa <- "CS_rate"
bb <- 100
cc <- "CS_rate_100"
# END SOLUTION
```

```
test_that("p1a", {
  expect_true(aa == "CS_rate")
  print("Correct! Try to understand what each part of the code on line 82 does!")
})
```

```
## [1] "Correct! Try to understand what each part of the code on line 82 does!"
## Test passed
```

```
test_that("p1b", {
  expect_true(bb == "100" | bb == 100)
  print("Correct! Try to understand what each part of the code on line 82 does!")
})
```

```
## [1] "Correct! Try to understand what each part of the code on line 82 does!"
## Test passed
```

```
test_that("p1c", {
  expect_true(cc == "CS_rate_100")
  print("Correct! Try to understand what each part of the code on line 82 does!")
})
```

```
## [1] "Correct! Try to understand what each part of the code on line 82 does!"
## Test passed
```

Investigate what would have happened had we not assigned the data using <- to CS_data. Re-run the code without the assignment operator and see what happens.

```
# First, let's re-read in the data as we did in the previous chunk
CS_data <- read_csv("data/cesarean.csv")
```

```
## Rows: 137 Columns: 7
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (4): Country_Name, CountryCode, Income_Group, Region
## dbl (3): Births_Per_1000, GDP_2006, CS_rate
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
CS_data$Income_Group <- forcats::fct_relevel(CS_data$Income_Group,
                                             "Low income", "Lower middle income",
                                             "Upper middle income", "High income: nonOECD",
                                             "High income: OECD")


# Now, we try mutating again without the re-assignment to CS_data
CS_data %>% mutate(CS_rate_100 = CS_rate*100)
```

```
## # A tibble: 137 x 8
##    Country_Name CountryCode Births_Per_1000 Income_Group Region GDP_2006 CS_rate
##    <chr>        <chr>                <dbl> <fct>        <chr>     <dbl>   <dbl>
##  1 Albania      ALB                     46 Upper middl~ Europ~    3052.   0.256
##  2 Andorra      AND                      1 High income~ Europ~   42417.   0.237
##  3 United Arab~ ARE                     63 High income~ Middl~   42950.   0.1
##  4 Argentina    ARG                    689 High income~ Latin~    6649.   0.352
##  5 Armenia      ARM                     47 Lower middl~ Europ~    2127.   0.141
##  6 Australia    AUS                    267 High income~ East ~   36101.   0.303
##  7 Austria      AUT                     76 High income~ Europ~   40431.   0.271
##  8 Azerbaijan   AZE                    166 Upper middl~ Europ~    2473.   0.076
##  9 Belgium      BEL                    119 High income~ Europ~   38936.   0.159
## 10 Benin        BEN                    342 Low income   Sub-S~     557.   0.036
## # ... with 127 more rows, and 1 more variable: CS_rate_100 <dbl>
```

```
# check the variables in CS_data
names(CS_data)
```

```
## [1] "Country_Name"    "CountryCode"     "Births_Per_1000" "Income_Group"
## [5] "Region"          "GDP_2006"        "CS_rate"
```

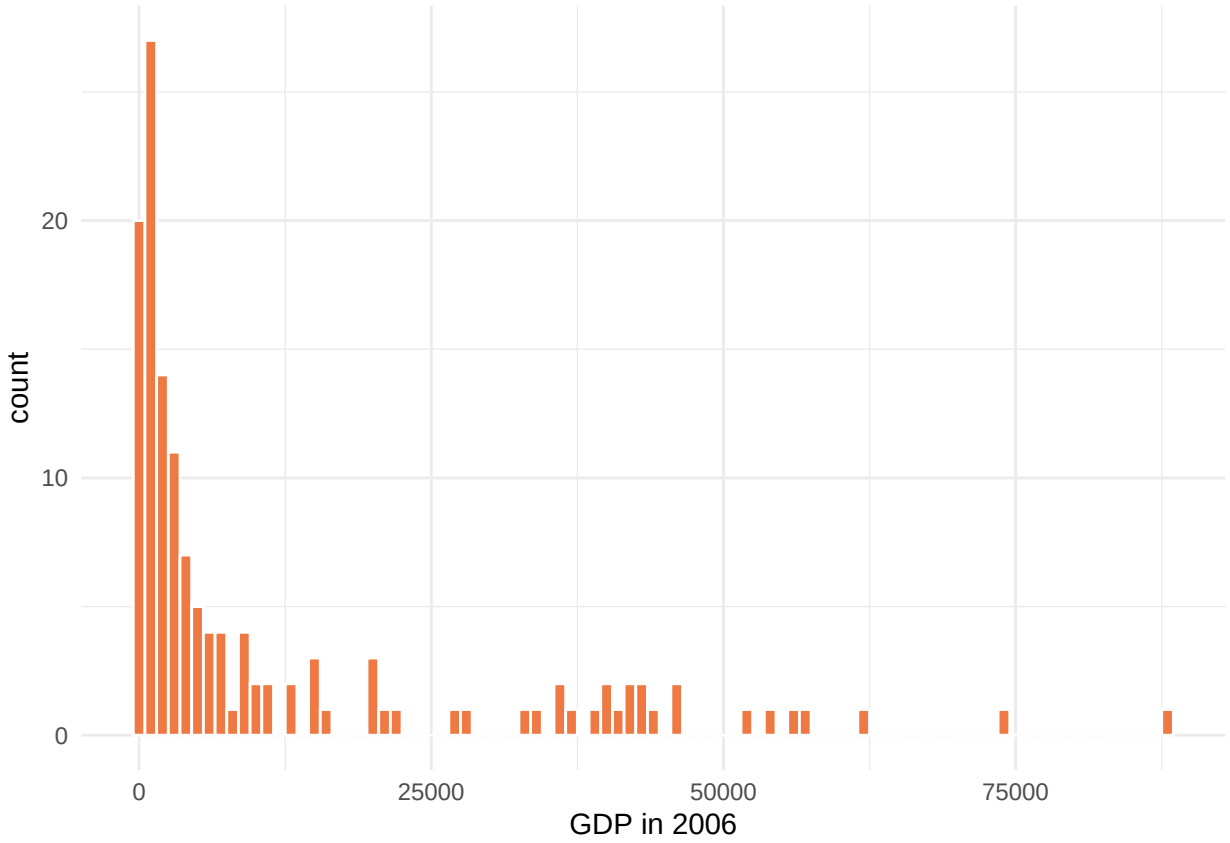
Did `CS_rate_100` get added to the data set? No. You can tell by using `head(CS_data)` to view the first few rows and notice that the variable hasn't been added. This is because when we don't assign the output to anything, it just prints it out for us to see. Nothing is saved. So, we want to save the output by assigning the result of the code to an object, which is `CS_data` in this case. In general, you want to use **new object names** at every significant step in your analysis as you work with your data, so that you have access to the specific data at all of those significant stages. However, if you are performing multiple small operations on the same dataset, you can overwrite the original object since you know you won't be needing the in-between steps. See the example below.

```r
# This overwrites the original CS_data object
CS_data <- CS_data %>% mutate(CS_rate_100 = CS_rate*100)
```

GDP: Summarizing measures of centrality

Recall your histogram of GDP from this week's lab:

```
ggplot(data = CS_data, aes(x = GDP_2006)) +
  geom_histogram(col = "white", fill = "sienna2", binwidth = 1000) +
  xlab("GDP in 2006") +
  theme_minimal()
```



2. [1 point] Describe the shape of this distribution. Is it "skewed left", "skewed right", "symmetric", or "bimodal"? Uncomment one of the possible choices.

```
. = " # BEGIN PROMPT
# p2 <- 'skewed left'
# p2 <- 'skewed right'
# p2 <- 'symmetric'
# p2 <- 'bimodal'
" # END PROMPT

# BEGIN SOLUTION
p2 <- 'skewed right'
# END SOLUTION
```

```
test_that("p2a", {
  expect_true(p2 == 'skewed right')
```

```
  print("Checking: chose the correct selection")
})
```

```
## [1] "Checking: chose the correct selection"
## Test passed
```

3. [1 point] Based on your answer, will the mean be approximately the "same", "larger than", or "smaller than" the median? Uncomment one of the possible choices.

```
. = " # BEGIN PROMPT
# p3 <- 'same'
# p3 <- 'larger than'
# p3 <- 'smaller than'
" # END PROMPT

# BEGIN SOLUTION
p3 <- 'larger than'
# END SOLUTION
```

```
test_that("p3a", {
  expect_true(p3 == 'larger than')
  print("Checking: chose the correct selection")

})
```

```
## [1] "Checking: chose the correct selection"
## Test passed
```

4. [3 points] Describe, in words, how the mean and median are calculated.

- [1 point] Mean: add up all the measurement values and divide by their total observation count.
- [2 points] Median: First, order the measurements by their value. If the total observation count is an odd number, the middle observation is the median. If an even number, add the two middle measurement values and divide by two to calculate the median.

To calculate the mean and median in R, we can use the `summarize()` function from the `dplyr` package. The `summarize()` function is used any time we want to take a variable and use one number to summarize something about it, like with the mean or median. Below is the code to summarize the mean of all countries' `GDP_2006` and to print it to the screen. In the code, we name the mean `mean_GDP` and output the result:

```
GDP_summary <- CS_data %>% summarize(mean_GDP = mean(GDP_2006))
GDP_summary
```

```
## # A tibble: 1 x 1
##   mean_GDP
##      <dbl>
## 1   11791.
```

5. [1 point] Extend the above code to also summarize the median. Call the median summary `median_GDP`. Assign this summary to a dataframe called `GDP_summary` (it will overwrite the previous version of the dataframe). Then type `GDP_summary` on its own line to see your results.

```
. = " # BEGIN PROMPT
GDP_summary <- NULL # YOUR CODE HERE
GDP_summary
" # END PROMPT

# BEGIN SOLUTION
GDP_summary <- CS_data %>% summarize(mean_GDP = mean(GDP_2006),
                                     median_GDP = median(GDP_2006))
# END SOLUTION
```

```
test_that("p5a", {
  expect_true(all.equal(GDP_summary$mean_GDP, 11790.67, tol = 0.01))
  print("Checking: GDP_summary has a column called `mean_GDP` with the correct value")
})
```

```
## [1] "Checking: GDP_summary has a column called ‘mean_GDP‘ with the correct value"
## Test passed
```

```
test_that("p5b", {
  expect_true(all.equal(GDP_summary$median_GDP, 3351.305, tol = 0.01))
  print("Checking: GDP_summary has a column called `median_GDP` with the correct value")
})
```

```
## [1] "Checking: GDP_summary has a column called ‘median_GDP‘ with the correct value"
## Test passed
```

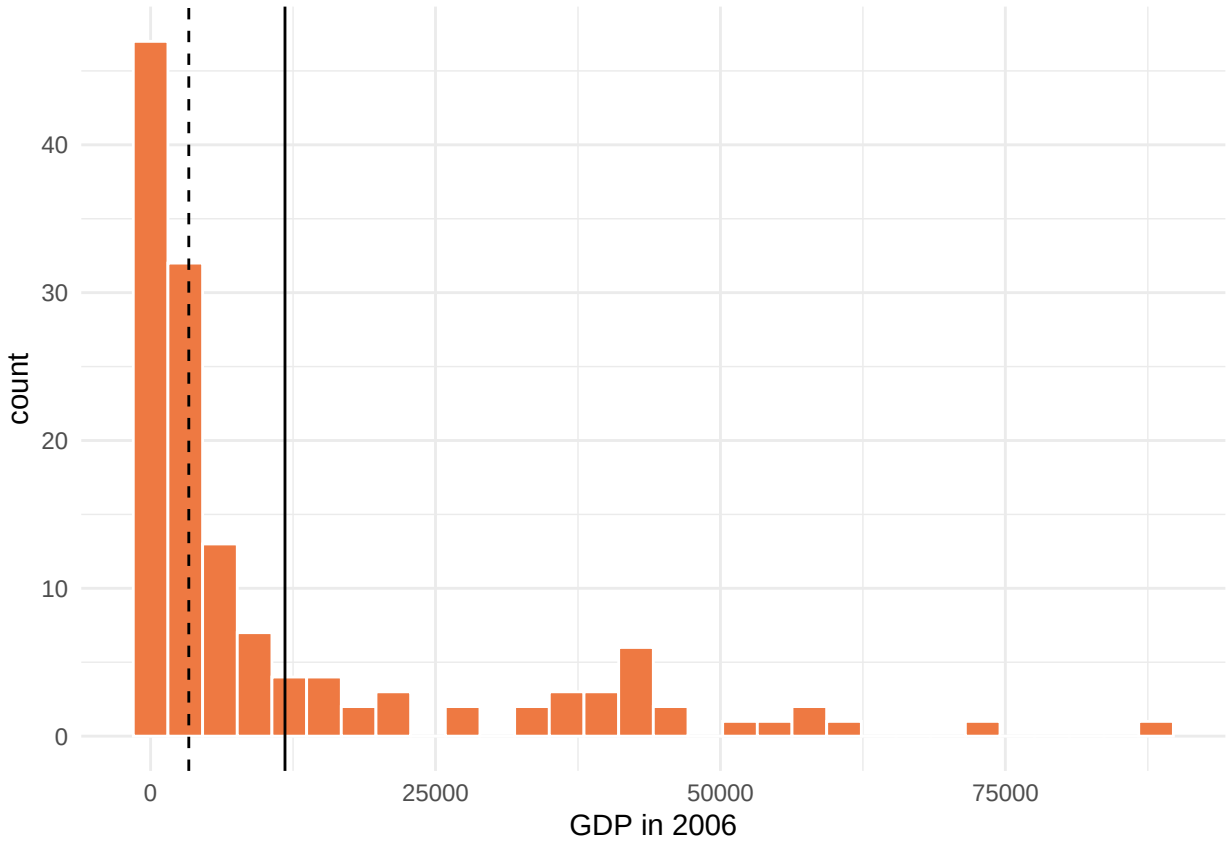
6. [2 points] `geom_vline()` can be used to add the mean and the median to the histogram shown above. This `geom_vline()` adds a vertical line to the graph. You need to specify where to add the line by passing it an "x-intercept" argument. Remove the comments (i.e., the three "#") from the code below and update the `geom_vline()` code to plot lines at the mean and median by telling it the mean and median estimates. The argument `lty=1` (line type) will plot a solid line and `lty=2` will plot a dashed line.

For the purposes of this question, please assign the x-intercept to a plain NUMERIC value, not a variable or expression.

```
. = " # BEGIN PROMPT
p6 <- ggplot(data = CS_data, aes(x = GDP_2006)) +
        geom_histogram(col = 'white', fill = 'sienna2') +
        xlab('GDP in 2006') +
        theme_minimal() #+
        #geom_vline(aes(xintercept = ), lty=1) +
        #geom_vline(aes(xintercept = ), lty=2)
p6
" # END PROMPT


# BEGIN SOLUTION
p6 <- ggplot(data = CS_data, aes(x = GDP_2006)) +
        geom_histogram(col = "white", fill = "sienna2") +
        xlab("GDP in 2006") +
        theme_minimal() +
        geom_vline(aes(xintercept = 11790.67), lty=1) +
        geom_vline(aes(xintercept = 3351.305), lty=2)
p6
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
# # ALTERNATE SOLUTION 1 (base R)
# p6 <- ggplot(data = CS_data, aes(x = GDP_2006)) +
#         geom_histogram(col = "white", fill = "sienna2") +
#         xlab("GDP in 2006") +
#         theme_minimal() +
#         geom_vline(aes(xintercept = GDP_summary$mean_GDP), lty=1) +
#         geom_vline(aes(xintercept = GDP_summary$median_GDP), lty=2)

# # ALTERNATE SOLUTION 2 (tidyverse)
# p6 <- ggplot(data = CS_data, aes(x = GDP_2006)) +
#         geom_histogram(col = "white", fill = "sienna2") +
#         xlab("GDP in 2006") +
#         theme_minimal() +
#         geom_vline(aes(xintercept = GDP_summary %>% pull(mean_GDP)), lty=1) +
#         geom_vline(aes(xintercept = GDP_summary %>% pull(median_GDP)), lty=2)
#
# # END SOLUTION
```

```
test_that("p6a", {
  expect_true("ggplot" %in% class(p6))
  print("Checking: p6 is a ggplot")
})
```

```
## [1] "Checking: p6 is a ggplot"
## Test passed
```

```
test_that("p6b", {
  expect_true(all.equal(p6$layers[[2]]$mapping$xintercept, 11790.67, tol = 0.01))
  print("Checking: First vline (mean) is set to the correct value")
})
```

```
## [1] "Checking: First vline (mean) is set to the correct value"
## Test passed
```

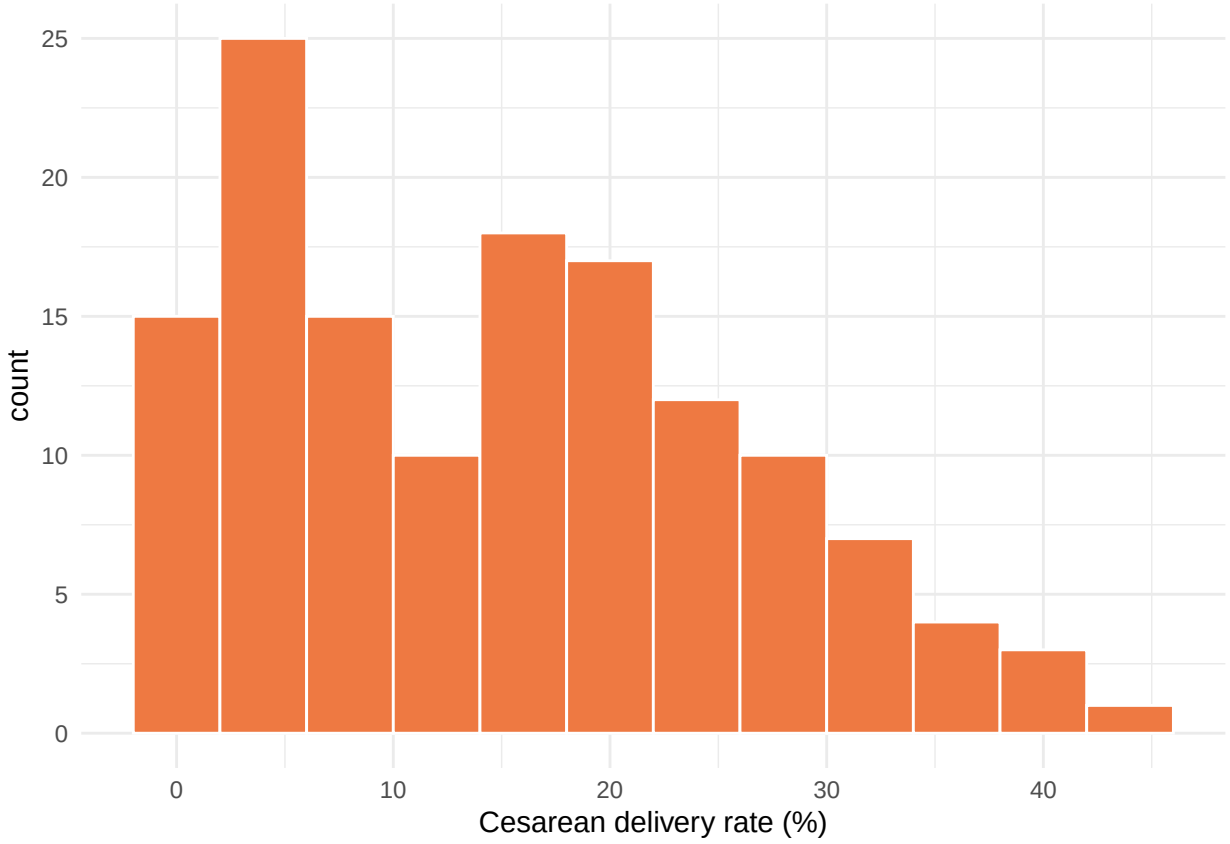
```
test_that("p6c", {
  expect_true(all.equal(p6$layers[[3]]$mapping$xintercept, 3351.305, tol = 0.01))
  print("Checking: Second vline (median) is set to the correct value")
})
```

```
## [1] "Checking: Second vline (median) is set to the correct value"
## Test passed
```

Summarizing the distribution of cesarean delivery rates

Recall the distribution of cesarean delivery rates across countries:

```
ggplot(data = CS_data, aes(x = CS_rate_100)) +
  geom_histogram(binwidth = 4, col = "white", fill = "sienna2") +
  xlab("Cesarean delivery rate (%)") +
  theme_minimal()
```



7. [1 point] Describe the shape of this distribution. Is it "skewed left", "skewed right", "symmetric", or "bimodal"? Uncomment one of the possible selections.

```
. = " # BEGIN PROMPT
# p7 <- 'skewed left'
# p7 <- 'skewed right'
# p7 <- 'symmetric'
# p7 <- 'bimodal'
" # END PROMPT

# BEGIN SOLUTION
p7 <- 'bimodal'
# END SOLUTION
```

```
test_that("p7", {
  expect_true(p7 == 'bimodal' | p7 == 'skewed right')
  print("Checking: p7 is one of 2 possible correct choices")
})
```

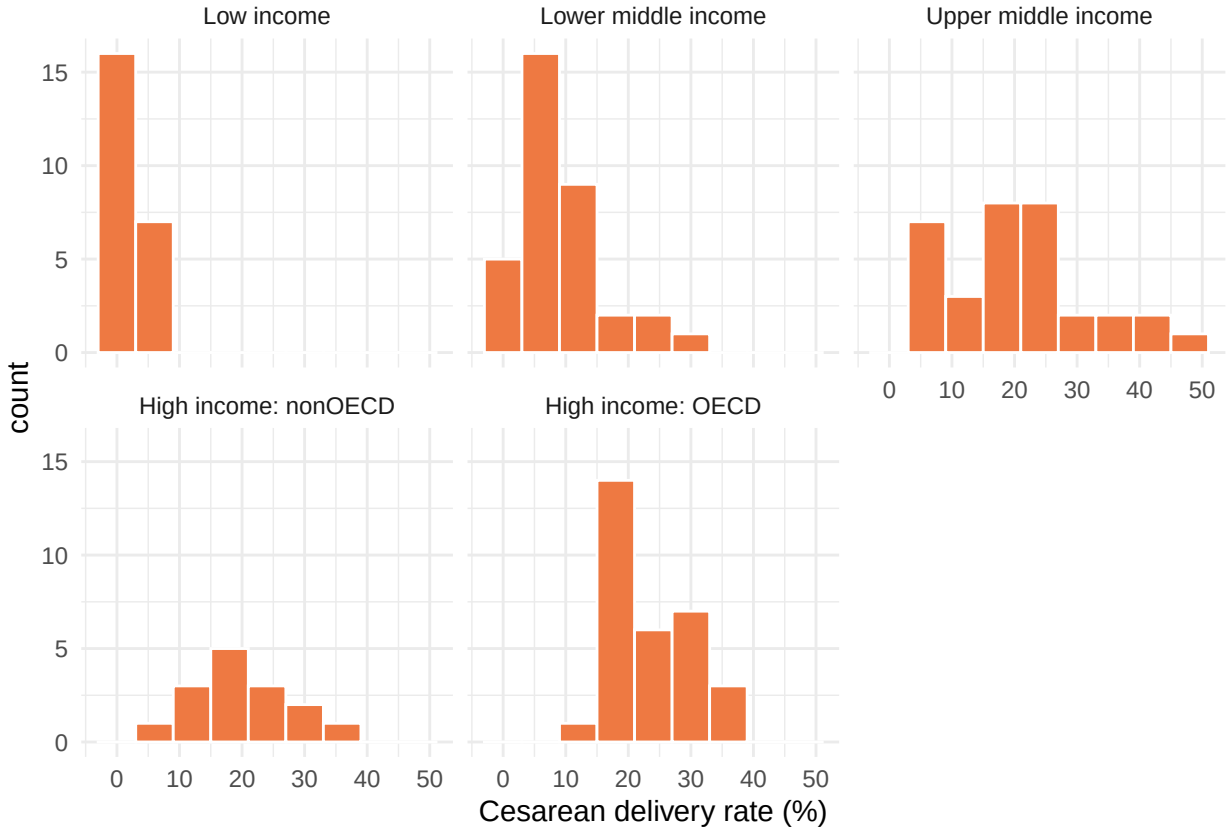
```
## [1] "Checking: p7 is one of 2 possible correct choices"
## Test passed
```

There appears to be multiple peaks in the cesarean delivery rate histogram which may point to another variable influencing the distribution. We can make a separate histogram for each income group using the `facet_wrap()` function.

8. [1 point] Extend the ggplot code given below using the `facet_wrap()` statement to make a separate histogram for each level of the `Income_Group` variable.

```
. = " # BEGIN PROMPT
p8 <- ggplot(data = CS_data, aes(x = CS_rate_100)) +
        geom_histogram(binwidth = 6, col = 'white', fill = 'sienna2') +
        xlab('Cesarean delivery rate (%)') +
        theme_minimal()
p8
" # END PROMPT


# BEGIN SOLUTION
p8 <- ggplot(data = CS_data, aes(x = CS_rate_100)) +
        geom_histogram(binwidth = 6, col = "white", fill = "sienna2") +
        xlab("Cesarean delivery rate (%)") +
        theme_minimal() +
        facet_wrap(. ~ Income_Group) # facet_grid(Income_Group ~ .) is also fine
p8
```




```
# END SOLUTION
```

```
test_that("p8a", {
  expect_true("ggplot" %in% class(p8))
  print("Checking: p8 is a ggplot")
})
```

```
## [1] "Checking: p8 is a ggplot"
## Test passed
```

```
test_that("p8b", {
  expect_true(!is.null(p8$facet$params[1]$facets$Income_Group) ||
                !is.null(p8$facet$params[1]$facets$`"Income_Group"`))
  print("Checking: There is a separate histogram for each level of the `Income_Group` variable!")
})
```

```
## [1] "Checking: There is a separate histogram for each level of the ‘Income_Group‘ variable!"
## Test passed
```

9. [2 points] Based on both this plot and the previous plot, describe why the data had two peaks.

[+1pt for discussing low/lower income, +1pt for discussing higher income countries] Many of the low and lower-middle income countries had CS rates $< 10\%$, while the higher income counties have rates closer to 20%.

10. [1 point] Why might lower income countries have lower rates of cesarean delivery?

Solution: [+1pt for an answer that sounds reasonable, no pts if the answer doesn't make sense or is obviously incorrect] Lower income countries have reduced access to obstetrical care, especially surgical procedures, limiting the number of women who can receive cesarean deliveries.

11. [2 points] Calculate the `mean_CS` and `median_CS` of `CS_rate_100` using only one `summarize()` command. Assign this summary to a dataframe called `CS_summary` and then print the results by typing `CS_summary`.

```
. = " # BEGIN PROMPT
CS_summary <- NULL # YOUR CODE HERE
CS_summary
" # END PROMPT

# BEGIN SOLUTION
CS_summary <- CS_data %>% summarize(mean_CS = mean(CS_rate_100),
                                    median_CS = median(CS_rate_100))
# END SOLUTION
```

```
test_that("p11a", {
  expect_true(all.equal(CS_summary$mean_CS, 15.26642, tol = 0.01))
  print("Checking: CS_summary has a column called `mean_CS` with the correct value")
})
```

```
## [1] "Checking: CS_summary has a column called ‘mean_CS‘ with the correct value"
## Test passed
```

```
test_that("p11b", {
  expect_true(all.equal(CS_summary$median_CS, 15.6, tol = 0.01))
  print("Checking: CS_summary has a column called `median_CS` with the correct value")
})
```

```
## [1] "Checking: CS_summary has a column called ‘median_CS‘ with the correct value"
## Test passed
```

Measures of variation

12. [2 points] Use `ggplot2` to make a boxplot of the distribution of `CS_rate_100`.

```
. = " # BEGIN PROMPT
p12 <- NULL # YOUR CODE HERE
p12
" # END PROMPT


# BEGIN SOLUTION
p12 <- ggplot(data = CS_data, aes(y = CS_rate_100)) +
          geom_boxplot(col = "black", fill = "sienna2") +
          theme_minimal()
# END SOLUTION
```

```
test_that("p12a", {
  expect_true("ggplot" %in% class(p12))
  print("Checking: p12 is a ggplot")
})
```

```
## [1] "Checking: p12 is a ggplot"
## Test passed
```

```
test_that("p12b", {
  expect_true(rlang::quo_get_expr(p12$mapping$y) == "CS_rate_100")
  print("Checking: CS_rate_100 is on the x-axis")
})
```

```
## [1] "Checking: CS_rate_100 is on the x-axis"
## Test passed
```

```
test_that("p12c", {
  expect_true("GeomBoxplot" %in% class(p12$layers[[1]]$geom))
  print("Checking: Made a boxplot")
})
```

```
## [1] "Checking: Made a boxplot"
## Test passed
```

Recall that the box plot summarizes the distribution in five numbers: the minimum, the first quartile (with 25% of the data below it), the median, the third quartile (with 75% of the data below it), and the maximum. Each of these numbers has at least one corresponding R function:

Number	R function
Minimum	`min(variable)`
First quartile	`quantile(variable, probs = 0.25)`
Median	`median(variable)` or `quantile(variable, probs = 0.5)`
Third quartile	`quantile(variable, probs = 0.75)`
Maximum	`max(variable)`

13. [2 points] Use a combination of `dplyr`'s `summarize` function and the above functions to compute the five number summary of `CS_rate_100`. Assign the summary to the name `five_num_summary`, which should contain values for `min`, `Q1`,`median`, `Q3`, and `max` (in this order and with these names).

```
. = " # BEGIN PROMPT
five_num_summary <- NULL # YOUR CODE HERE
five_num_summary
" # END PROMPT

# BEGIN SOLUTION
five_num_summary <- CS_data %>% summarize(
  min = min(CS_rate_100),
  Q1 = quantile(CS_rate_100, 0.25),
  median = median(CS_rate_100),
  Q3 = quantile(CS_rate_100, 0.75),
  max = max(CS_rate_100)
)
# END SOLUTION
```

```
test_that("p13a", {
  expect_true(all.equal(five_num_summary$min, 0.4, tol = 0.01))
  print("Checking: five_num_summary has a column called `min` with the correct value")
})
```

```
## [1] "Checking: five_num_summary has a column called ‘min‘ with the correct value"
## Test passed
```

```
test_that("p13b", {
  expect_true(all.equal(five_num_summary$Q1[[1]], 5.1, tol = 0.01))
  print("Checking: five_num_summary has a column called `Q1` with the correct value")
})
```

```
## [1] "Checking: five_num_summary has a column called ‘Q1‘ with the correct value"
## Test passed
```

```
test_that("p13c", {
  expect_true(all.equal(five_num_summary$median, 15.6, tol = 0.01) |
                all.equal(five_num_summary$median[[1]], 15.6, tol = 0.01))
  print("Checking: five_num_summary has a column called `median` with the correct value")
})
```

```
## [1] "Checking: five_num_summary has a column called ‘median‘ with the correct value"
## Test passed
```

```
test_that("p13d", {
  expect_true(all.equal(five_num_summary$Q3[[1]], 23.3, tol = 0.01))
  print("Checking: five_num_summary has a column called `Q3` with the correct value")
})
```

```
## [1] "Checking: five_num_summary has a column called ‘Q3‘ with the correct value"
## Test passed
```

```
test_that("p13e", {
  expect_true(all.equal(five_num_summary$max, 45.9, tol = 0.01))
  print("Checking: five_num_summary has a column called `max` with the correct value")
})
```

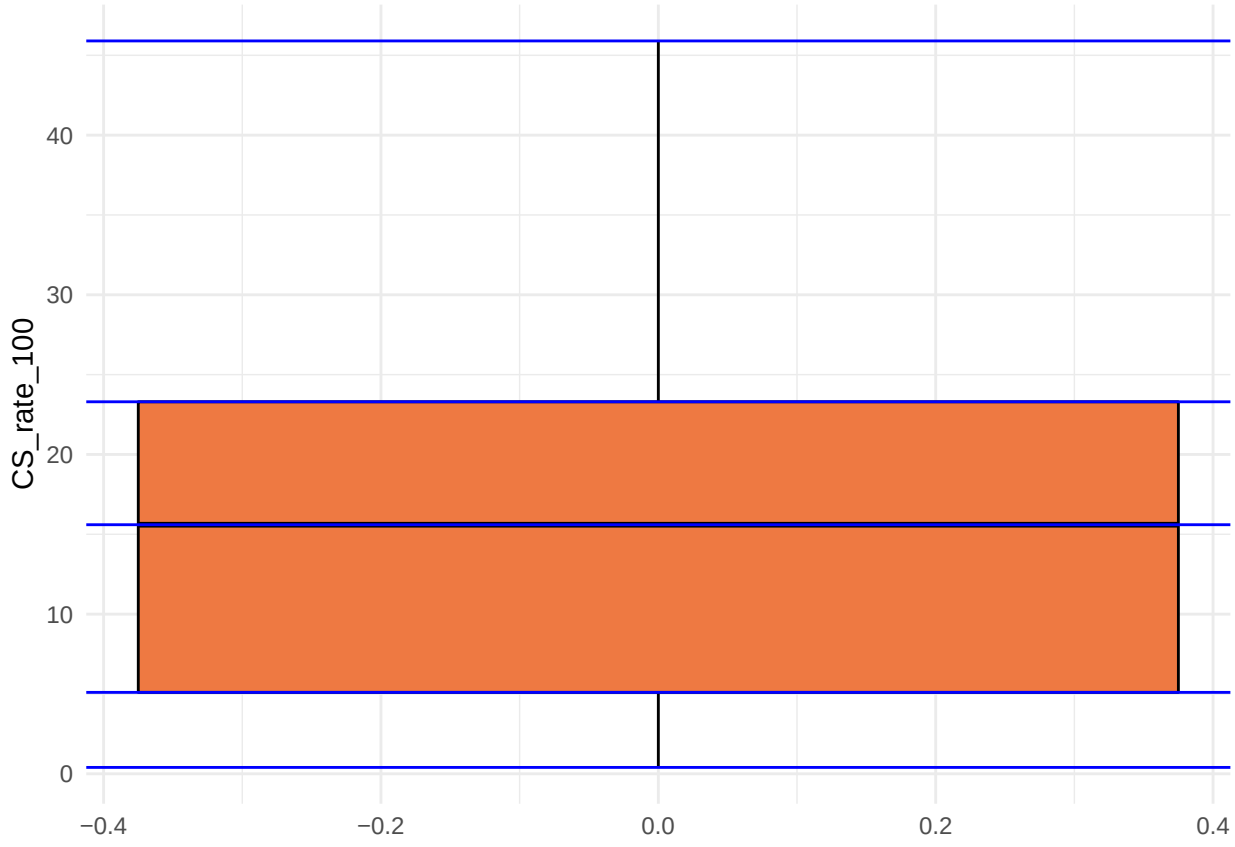
```
## [1] "Checking: five_num_summary has a column called ‘max‘ with the correct value"
## Test passed
```

Double check that `geom_boxplot()` is making the box plot correctly. You can do this by adding horizontal lines to the plot at each number in your five number summary using `geom_hline()`. Because horizontal lines intercept the y-axis, `geom_hline()` requires the `yintercept` argument that you can set to each number in your summary.

14. [2 points] The code below includes one horizontal line at the minimum value shown in blue. Add the rest of the lines in the order denoted in question 13.

```
. = " # BEGIN PROMPT
p14 <- ggplot(data = CS_data, aes(y = CS_rate_100)) +
          geom_boxplot(col = 'black', fill = 'sienna2') +
          theme_minimal() +
          geom_hline(aes(yintercept = 0.4), col = 'blue')
          # add more geom_hlines
p14
" # END PROMPT

# BEGIN SOLUTION
p14 <- ggplot(data = CS_data, aes(y = CS_rate_100)) +
        geom_boxplot(col = "black", fill = "sienna2") +
        theme_minimal() +
        geom_hline(aes(yintercept = 0.4), col = "blue") +
        geom_hline(aes(yintercept = 5.1), col = "blue") +
        geom_hline(aes(yintercept = 15.6), col = "blue") +
        geom_hline(aes(yintercept = 23.3), col = "blue") +
        geom_hline(aes(yintercept = 45.9), col = "blue")
p14
```



```
# END SOLUTION
```

```
test_that("p14a", {
  expect_true(all.equal(p14$layers[[2]]$mapping$yintercept, 0.4, tol = 0.01))
  print("Checking first line: a y-intercept was added for min at the correct value")
})
```

```
## [1] "Checking first line: a y-intercept was added for min at the correct value"
## Test passed
```

```
test_that("p14b", {
  expect_true(all.equal(p14$layers[[3]]$mapping$yintercept, 5.1, tol = 0.01))
  print("Checking second line: a y-intercept was added for Q1 at the correct value")
})
```

```
## [1] "Checking second line: a y-intercept was added for Q1 at the correct value"
## Test passed
```

```
test_that("p14c", {
  expect_true(all.equal(p14$layers[[4]]$mapping$yintercept, 15.6, tol = 0.01))
  print("Checking third line: a y-intercept was added for median at the correct value")
})
```

```
## [1] "Checking third line: a y-intercept was added for median at the correct value"
## Test passed
```

```
test_that("p14d", {
  expect_true(all.equal(p14$layers[[5]]$mapping$yintercept, 23.3, tol = 0.01))
  print("Checking fourth line: a y-intercept was added for Q3 at the correct value")
})
```

```
## [1] "Checking fourth line: a y-intercept was added for Q3 at the correct value"
## Test passed
```

```
test_that("p14e", {
  expect_true(all.equal(p14$layers[[6]]$mapping$yintercept, 45.9, tol = 0.01))
  print("Checking fifth line: a y-intercept was added for max at the correct value")
})
```

```
## [1] "Checking fifth line: a y-intercept was added for max at the correct value"
## Test passed
```

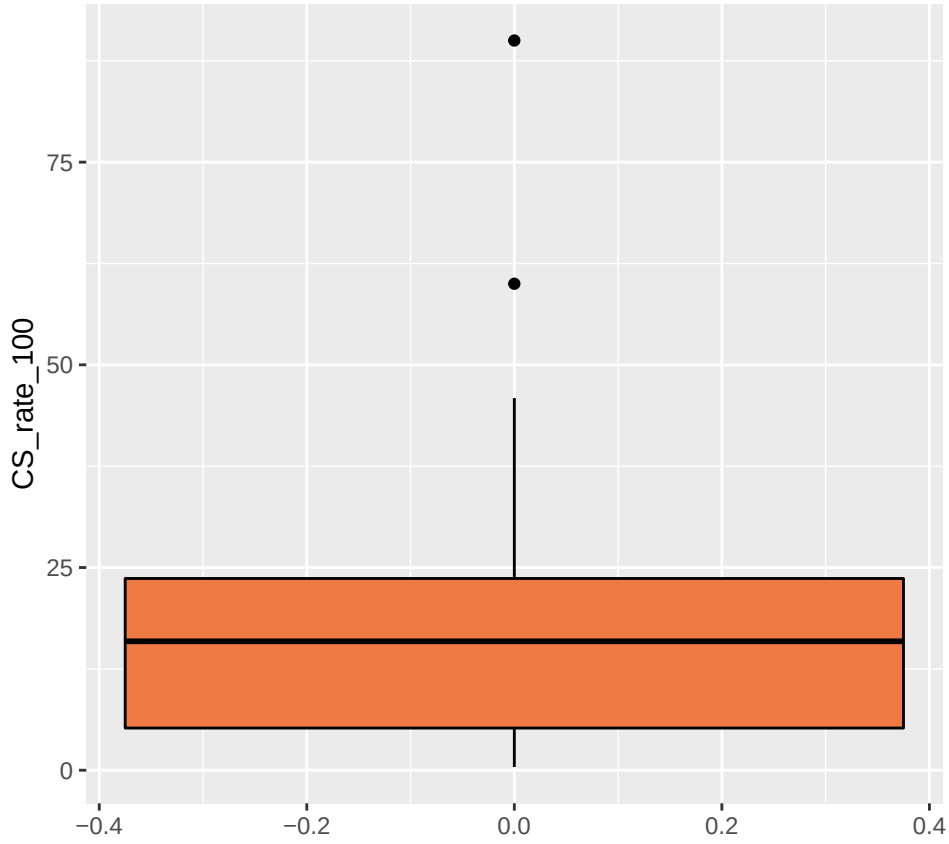
15. [4 points] Run the following code that adds two points to `CS_data`, makes a new dataset called `CS_data_plus_2`, and redraws the box plot. How did the box plot change? Perform a calculation to justify why it changed. What are the newly-added features on the plot called?

```
out_data <- tibble::tribble(
  ~Country_Name, ~CS_rate_100,
  "Point 1", 90,
  "Point 2", 60
)

CS_data_plus_2 <- dplyr::full_join(CS_data, out_data)
```

```
## Joining, by = c("Country_Name", "CS_rate_100")
```

```
ggplot(data = CS_data_plus_2, aes(y = CS_rate_100)) +
  geom_boxplot(col = "black", fill = "sienna2")
```




```
. = " # BEGIN PROMPT
# YOUR CALCULATIONS HERE
" # END PROMPT

# BEGIN SOLUTION
five_num_summary_new <- CS_data_plus_2 %>% summarize(
  min = min(CS_rate_100),
  Q1 = quantile(CS_rate_100, 0.25),
```

```
  median = median(CS_rate_100),
  Q3 = quantile(CS_rate_100, 0.75),
  max = max(CS_rate_100)
)
# END SOLUTION
```

- [1pt] There are two points above the top whisker on the revised box plot.
- [2pts calculation] These points must be larger than Q3 + 1.5*IQR.
  - The IQR = Q3-Q1 =23.65 - 5.2 = 18.45.
  - IQR times 1.5 = 27.675
  - Q3 + 27.675 = upper bound = 51.325
  - Both 60 and 90 are larger than 51.325, which is why they are suspected outliers.
- [1pt] The points are called suspected outliers (or outliers is fine).

END