Lab 2: Visualization of global rates of cesarean delivery

Felicia Liu

09/04/2020

Instructions

- Due date: Friday, September 4 at 11:59pm PST.
- Late penalty: 50% late penalty if submitted within 24 hours of due date, no marks for assignments submitted thereafter.
- This assignment is graded on **correct completion**, all or nothing. You must pass all public tests and submit the assignment for credit.
- Submission process: Follow the submission instructions on the final page. Make sure you do not remove any `\newpage` tags or rename this file, as this will break the submission.

Visualizing global cesarean delivery rates and GDP across 137 countries

The proportion of pregnant women who undergo cesarean deliveries varies dramatically across countries. In the US, the cesarean delivery rate is around 33%. A 2018 study in the British Medical Journal found that cesarean delivery rates ranged between 0.6% in South Sudan and 58.9% in the Dominican Republic.[1]

Over the course of this lab and the first assignment, we will study the relationship between countries' cesarean delivery rates and their gross domestic products (GDPs).[2,3]

In today's lab we will:

- import the data,
- familiarize ourselves with the dataset,
- explore the univariate distributions of the key variables.

If you haven't already, begin by knitting this document by clicking the "Knit" button above. As you fill it out, you can re-knit (clicking the button again) and see how the document changes.

Let's get started!

Click the green arrow icon in the top right corner of the gray chunk below to execute the three lines of code in the code chunk, or execute them line by line by placing your cursor on the first line and hitting cmd + enter on Mac or ctrl + enter on PC. If you do not want messages produced by loading `dplyr` to print in your knitted document, toggle `message = TRUE` to `message = FALSE` in the code chunk options. Re-knit your file and notice the different in the output:

```r
library(readr)
library(dplyr)
library(ggplot2)
```

- The `library` command loads the packages into memory.
- The `readr` library contains function for reading in data.
- The `dplyr` library contains functions we will use to manipulate data.
- The `ggplot2` library contains functions to visualize data.

The data on each country's cesarean delivery rate is stored in a comma-separated-values (csv) file. We can load that data and assign it to an object called `CS_data` by running the code below. Click the green arrow in the line below to run the code. (Note that the green arrow will run all the code in this code chunk, not just the first line of code.):

```
CS_data_raw <- read_csv("cesarean.csv")
```

```
## Parsed with column specification:
## cols(
##   Country_Name = col_character(),
##   CountryCode = col_character(),
##   Births_Per_1000 = col_double(),
##   Income_Group = col_character(),
##   Region = col_character(),
##   GDP_2006 = col_double(),
##   CS_rate = col_double()
## )
```

```
# This code re-orders the variable Income_Group in the specified order.
# Note that it *does not* change the order of the data frame (like arrange() does)
# Rather, it specifies the order the data will be plotted.
# This will make more sense when we plot the data using Income_Group, and then
# again using Income_Group_order
CS_data_raw <- CS_data_raw %>%
  mutate(Income_Group_order = forcats::fct_relevel(Income_Group, "Low income",
                                            "Lower middle income", "Upper middle income",
                                            "High income: nonOECD", "High income: OECD"))
```

Look over at the environment tab (on the top right pane) and make sure you see your dataset there. Click on `CS_data` in that pane. This will open the data viewer tab. Take a quick look at the data and then switch back to the "lab02-cesarean-delivery.Rmd" file and move to the next step.

Often, when you receive a dataset, you also receive a data dictionary that tells you what each variable means. Below, is an example of such a dictionary for this dataset:

Variable	Description
Country_Name	The name of the country
CountryCode	3-digit unique country identifier
Births_Per_1000	The number of births per 1000 people in the country
Income_Group	The income group the country is categorized into
Region	The geographic region the country belongs to
GDP_2006	The gross domestic product (GDP) of the country in 2006
CS_rate	The percent of births that were delivered by cesarean section
Income_Group_order	The "ordered" version of Income_Group

Look at your newly imported data

In four separate code chunks, type `str(CS_data_raw)`, `dim(CS_data_raw)`, `names(CS_data_raw)`, and `head(CS_data_raw)`. Execute each code chunk and study the output.

```
str(CS_data_raw)
```

```
## Classes 'spec_tbl_df', 'tbl_df', 'tbl' and 'data.frame': 137 obs. of  8 variables:
##  $ Country_Name      : chr  "Albania" "Andorra" "United Arab Emirates" "Argentina" ...
##  $ CountryCode       : chr  "ALB" "AND" "ARE" "ARG" ...
##  $ Births_Per_1000   : num  46 1 63 689 47 267 76 166 119 342 ...
##  $ Income_Group      : chr  "Upper middle income" "High income: nonOECD" "High income: nonOECD" "Hig
##  $ Region            : chr  "Europe & Central Asia" "Europe & Central Asia" "Middle East & North Afr
##  $ GDP_2006          : num  3052 42417 42950 6649 2127 ...
##  $ CS_rate           : num  0.256 0.237 0.1 0.352 0.141 0.303 0.271 0.076 0.159 0.036 ...
##  $ Income_Group_order: Factor w/ 5 levels "Low income","Lower middle income",..: 3 4 4 4 2 5 5 3 5 1
```

```
dim(CS_data_raw)
```

```
## [1] 137   8
```

```
names(CS_data_raw)
```

```
## [1] "Country_Name"       "CountryCode"        "Births_Per_1000"
## [4] "Income_Group"       "Region"             "GDP_2006"
## [7] "CS_rate"            "Income_Group_order"
```

```
head(CS_data_raw)
```

```
## # A tibble: 6 x 8
##   Country_Name CountryCode Births_Per_1000 Income_Group Region GDP_2006
##   <chr>        <chr>                 <dbl> <chr>        <chr>     <dbl>
## 1 Albania      ALB                      46 Upper middl~ Europ~    3052.
## 2 Andorra      AND                       1 High income~ Europ~   42417.
## 3 United Arab~ ARE                      63 High income~ Middl~   42950.
## 4 Argentina    ARG                     689 High income~ Latin~    6649.
## 5 Armenia      ARM                      47 Lower middl~ Europ~    2127.
## 6 Australia    AUS                     267 High income~ East ~   36101.
## # ... with 2 more variables: CS_rate <dbl>, Income_Group_order <fct>
```

In your own words, what do these functions do:

- `dim()`: gives the number of rows and the number of columns
- `str()`: summarizes head(), dim(), and names() all at once
- `names()`: lists out the variable names of the columns in the dataset
- `head()`: shows the first 6 rows of the dataset

Based on these functions' outputs, who/what are the individuals in this dataset? How many of them are there?

The individuals are the countries, and there are 137 of them.

Add a new variable

1. Right now, `CS_rate` is presented as a number between 0 and 1. For plotting, it will be nice to have the proportion displayed as a number between 0 and 100. We can use `dplyr`'s `mutate()` function to add a new variable called `CS_rate_100` to the dataset that takes this desired format. Write one line of code to add this variable to the dataset and assign this to the new object `CS_data`:

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

check_problem1()
```

```
## [1] "Checkpoint 1 Passed: Correct!"
## [1] "Checkpoint 2 Passed: Correct!"
## [1] "Checkpoint 3 Passed: Correct!"
##
## Problem 1
## Checkpoints Passed: 3
## Checkpoints Errored: 0
## 100% passed
## --------
## Test: PASSED
```

Examine the distribution of income and geographic regions

These countries represent a wide range of income and geographic regions.

Which variable groups the countries by income regions? Income_Group

Which variable groups the countries by geographic regions? Region

These variables are categorical variables. So far in lecture, we've seen categorical variables stored as `chr` variables when we've examined `str(data)`. Here however, `Income_Group_order` is stored as a `Factor` variable, which is another way to store categorical data. One benefit of storing it as `Factor` is that we can re-order the `Factor` variable in a way that makes sense for plotting and later analysis. To see the values and order of a factor variable, we can use the following code.

STOP: Remove eval = F from this chunk header before continuing on.

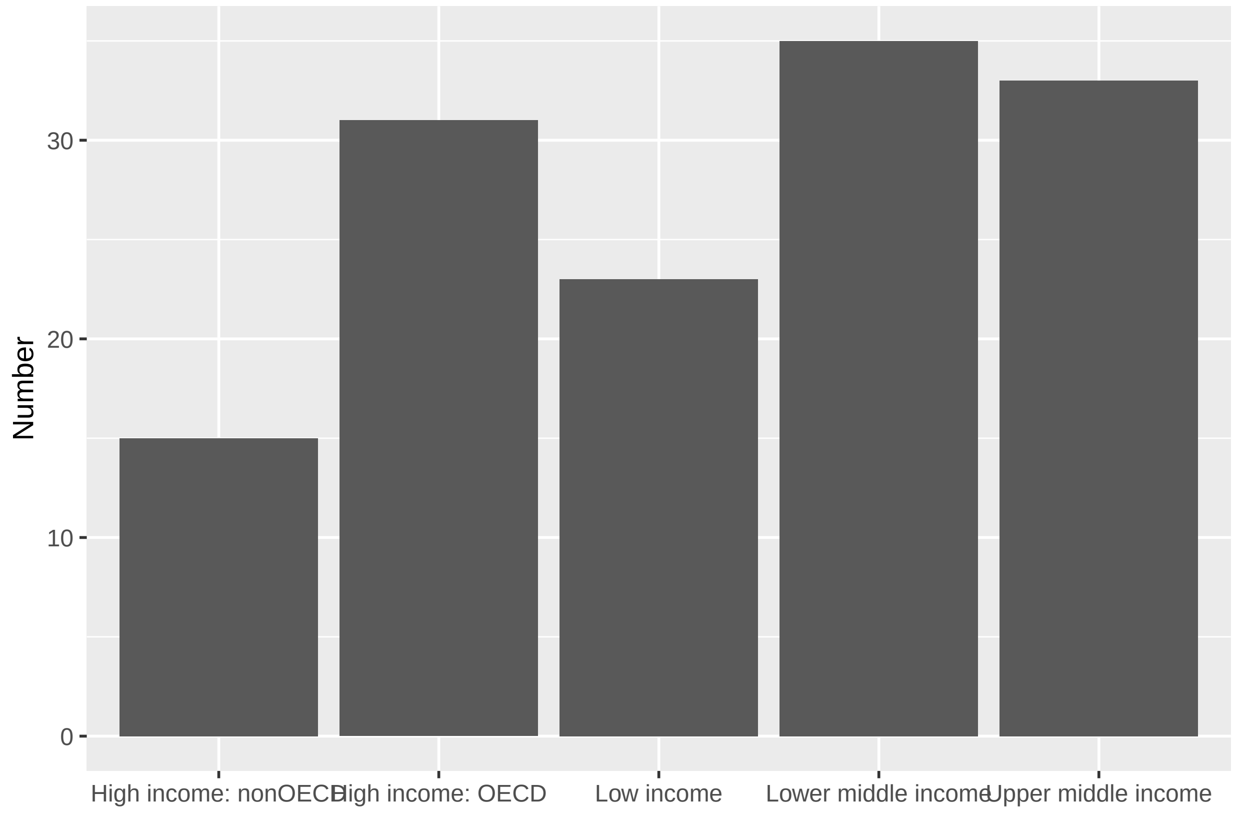
```
CS_data %>% pull(Income_Group_order) %>% levels
```

```
## [1] "Low income"            "Lower middle income"  "Upper middle income"
## [4] "High income: nonOECD" "High income: OECD"
```

```
# You won't be tested on writing the line of code above.
# Try and gain an understanding of what it is showing you
```

2. Because `Income_Group` and `Region` are categorical, we make bar charts to represent their distributions. Using the `ggplot()` code you learnt during lecture, make a bar chart showing the number of countries per income group using `Income_Group`.

```
p2 <- ggplot(CS_data, aes(x = Income_Group)) + geom_bar() + labs (y = "Number", x = "")
p2
```

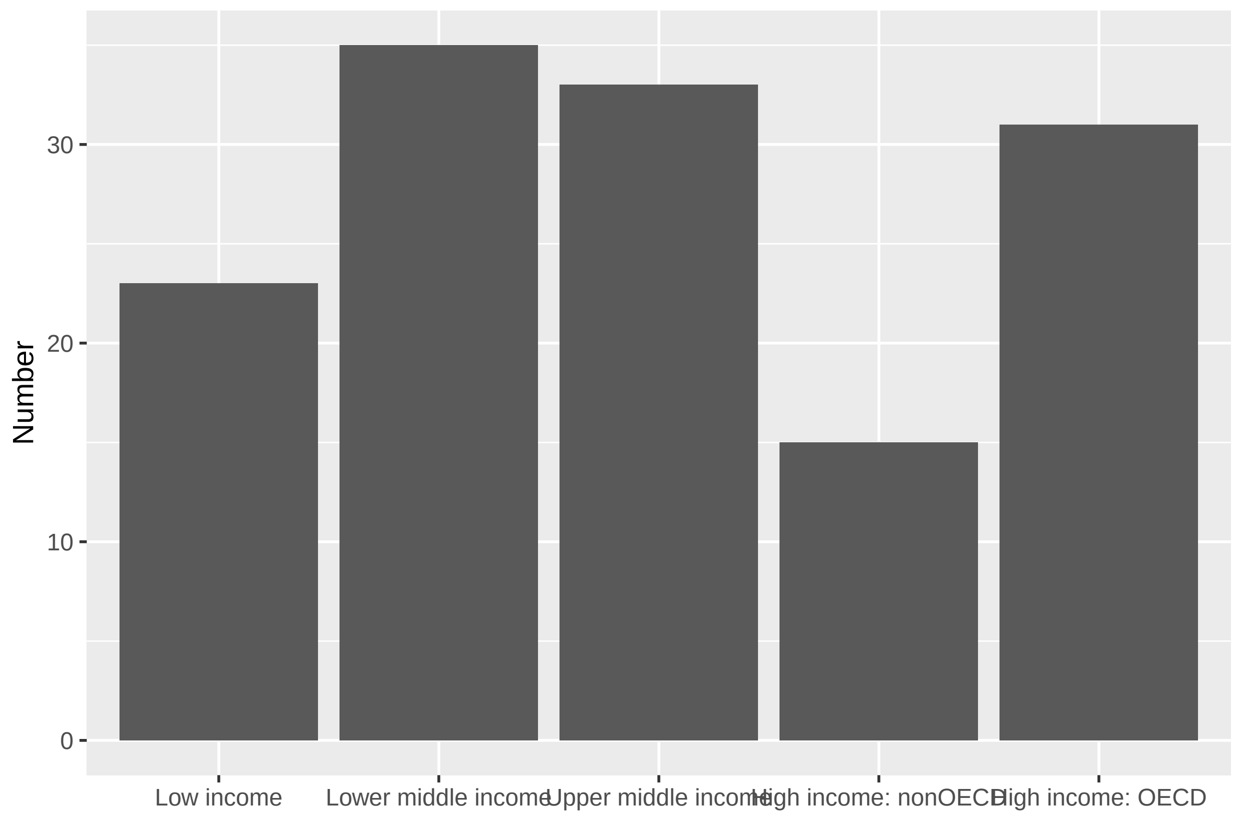


```
check_problem2()
```

```
## [1] "Checkpoint 1 Passed: A ggplot has been defined"
## [1] "Checkpoint 2 Passed: Correct!"
## [1] "Checkpoint 3 Passed: Correct!"
## [1] "Checkpoint 4 Passed: A bar chart has been defined in ggplot"
##
## Problem 2
## Checkpoints Passed: 4
## Checkpoints Errored: 0
## 100% passed
## --------
## Test: PASSED
```

3. Now, make the plot using `Income_Group_order`.

```
p3 <- ggplot(CS_data, aes(x = Income_Group_order)) + geom_bar() + labs (y = "Number", x = "")
p3
```



```
check_problem3()
```

```
## [1] "Checkpoint 1 Passed: A ggplot has been defined"
## [1] "Checkpoint 2 Passed: Correct!"
## [1] "Checkpoint 3 Passed: Correct!"
## [1] "Checkpoint 4 Passed: A bar chart has been defined in ggplot"
##
## Problem 3
## Checkpoints Passed: 4
## Checkpoints Errored: 0
## 100% passed
## --------
## Test: PASSED
```

Do you prefer the plot using `Income_Group` or `Income_Group_order`? Why?

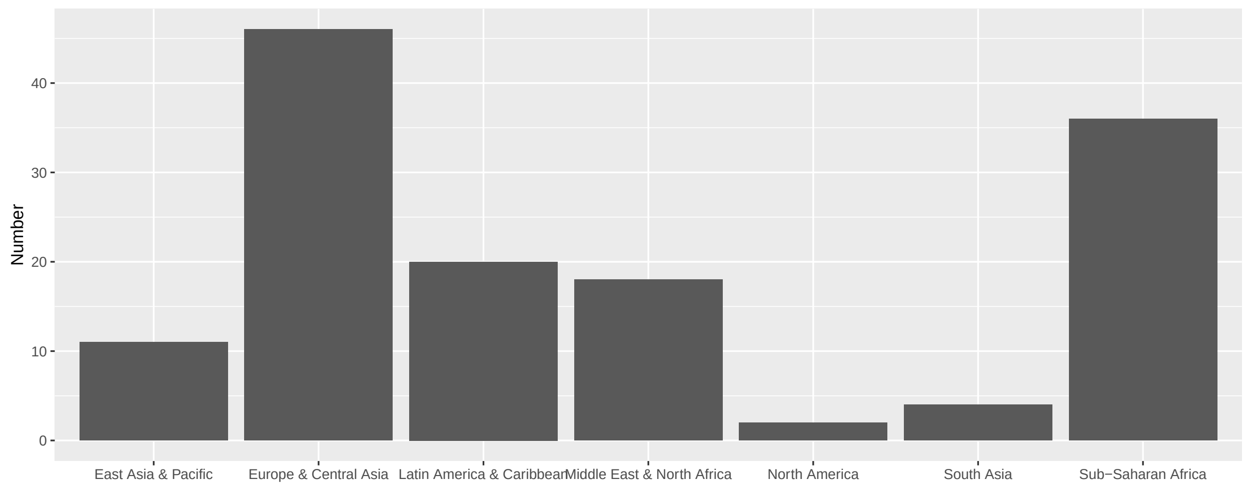
I prefer the plot using 'Income_Group_order' because the income groups are already sorted in a manner that makes the graph easier to read and understand.

Did you use the argument `stat="identity"` in the code to make this bar chart? Why or why not?

I did not use 'stat="identity" ' because I did not supply a y-variable that is exactly what I want to plot. Since I needed geom_bar() to calculate the number of countries on the y-axis, I did not include the argument.

4. Make a bar chart showing the number of countries per geographic region.

```r
p4 <-  ggplot(CS_data, aes(x = Region)) + geom_bar() + labs (y = "Number", x = "")
p4
```



```r
check_problem4()
```

```
## [1] "Checkpoint 1 Passed: A ggplot has been defined"
## [1] "Checkpoint 2 Passed: Correct!"
## [1] "Checkpoint 3 Passed: Correct!"
## [1] "Checkpoint 4 Passed: Correct!"
##
## Problem 4
## Checkpoints Passed: 4
## Checkpoints Errored: 0
## 100% passed
## --------
## Test: PASSED
```

Based on your plot, which world region has the most countries in the data set?

Europe & Central Asia

Which geographic region had the fewest number of countries in the data set

North America

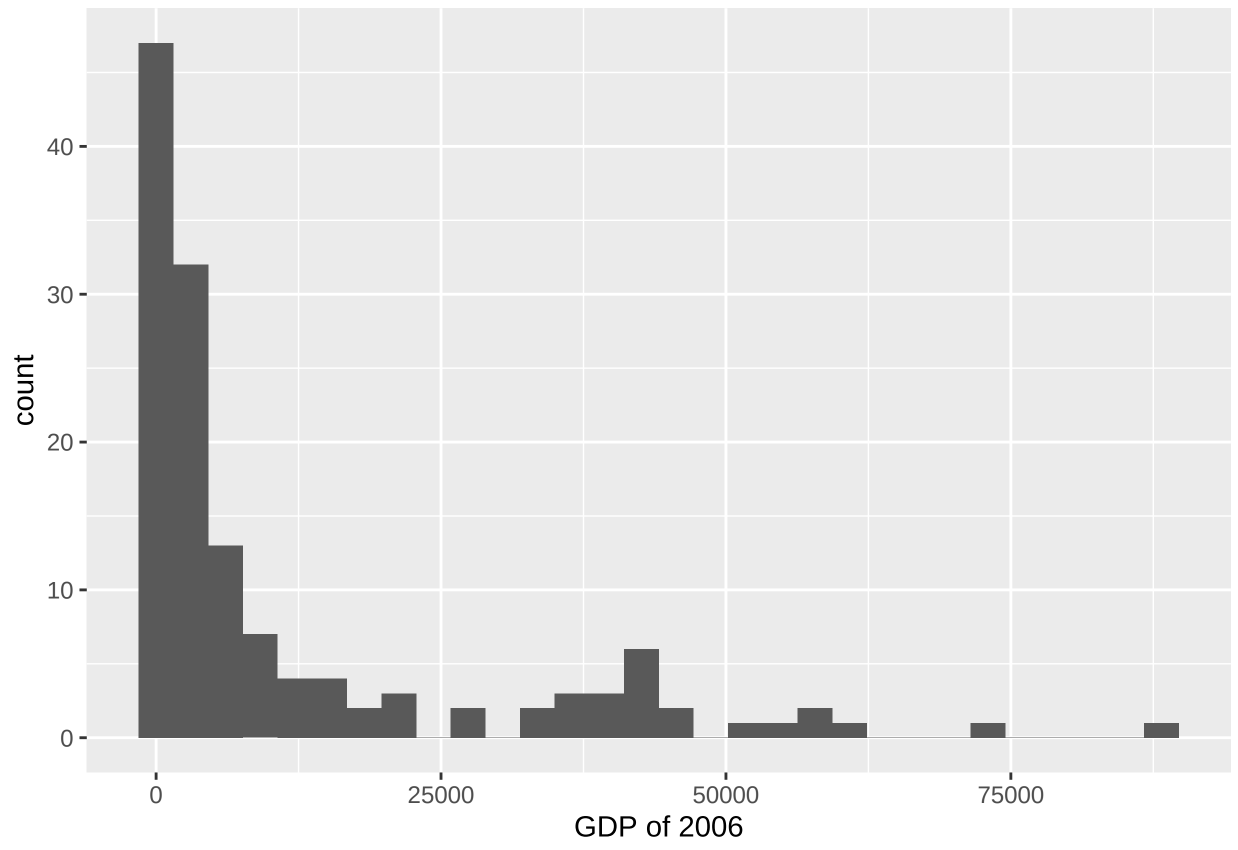
Note that the above code chunk had the chunk option "fig.width = 11". Try changing the setting to a lower number, like 9 and re-running the chunk. Notice how the width of the rendered plot changes. Change it back to 11 so that the x-axis labels don't look too cramped.

Examine the distribution of `GDP_2006` and `CS_rate_100` across countries

5. `GDP_2006` and `CS_rate_100` are quantitative, meaning that we use histograms rather than bar charts to examine their distributions. Using code you learnt in class, make a histogram of `GDP_2006`. Pay attention to the message that R outputs when running your code. It is telling you that by default the data is grouped into 30 bins for plotting.

```r
p5 <- ggplot(data = CS_data, aes(x = GDP_2006)) + geom_histogram()+ labs(x = "GDP of 2006")
p5
```

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



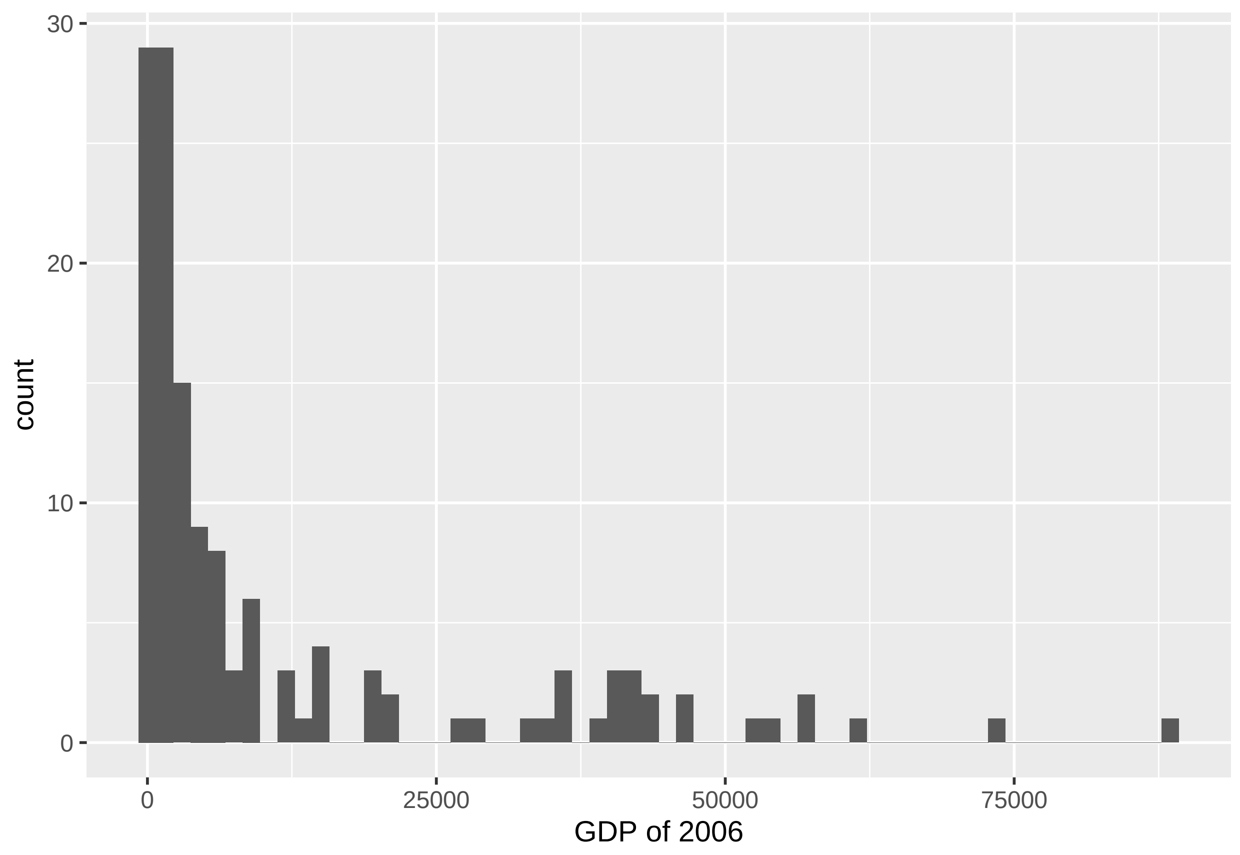
```r
check_problem5()
```

```
## [1] "Checkpoint 1 Passed: A ggplot has been defined"
## [1] "Checkpoint 2 Passed: Correct!"
## [1] "Checkpoint 3 Passed: Correct!"
## [1] "Checkpoint 4 Passed: Correct!"
##
## Problem 5
## Checkpoints Passed: 4
## Checkpoints Errored: 0
## 100% passed
## --------
## Test: PASSED
```

6. Rather than letting R choose the number of `bins` we can set `binwidth = number` within the geom function to choose the width of the bins. Try setting the binwidth to different amounts and see how it changes the look of the plot.

To choose a binwidth, look at the x-axis of the histogram you just made. Because we are plotting GDP across a wide variety of countries, it might make sense to choose a bin width in the 1000s. Thus, the range of the variable matters when choosing a good bin width.

```r
p6 <- ggplot(data = CS_data, aes(x = GDP_2006)) + geom_histogram(binwidth = 1500) +
labs(x = "GDP of 2006")
p6
```

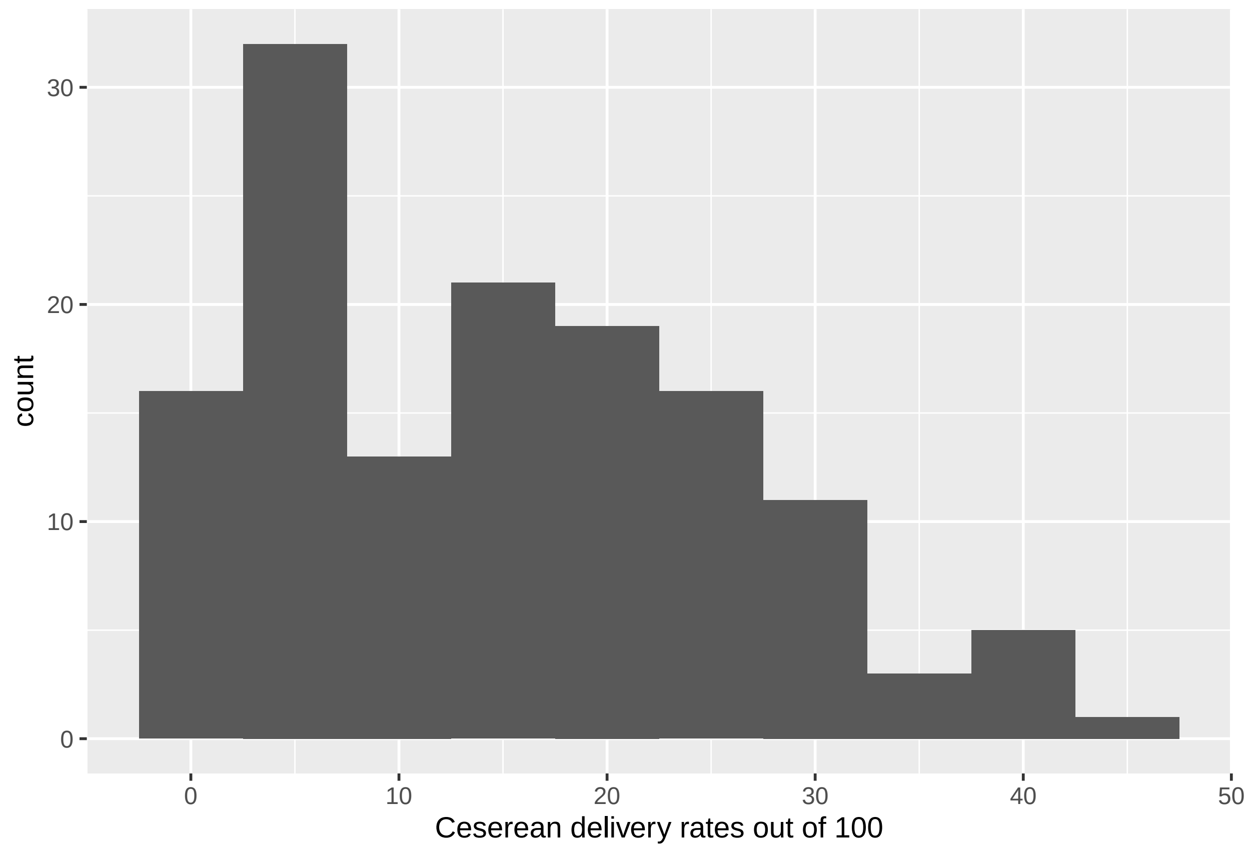


```
check_problem6()
```

```
## [1] "Checkpoint 1 Passed: A ggplot has been defined"
## [1] "Checkpoint 2 Passed: Correct!"
## [1] "Checkpoint 3 Passed: Correct!"
## [1] "Checkpoint 4 Passed: Correct!"
## [1] "Checkpoint 5 Passed: Correct!"
##
## Problem 6
## Checkpoints Passed: 5
## Checkpoints Errored: 0
## 100% passed
## --------
## Test: PASSED
```

7. Next, make a histogram for cesarean delivery rates using the variable `CS_rate_100`. First run your code without binwidth set and then add in the binwidth argument to choose the size.

```
p7 <- ggplot(data = CS_data, aes(x = CS_rate_100)) + geom_histogram(binwidth = 5) +
labs(x = "Ceserean delivery rates out of 100")
p7
```



```
check_problem7()
```

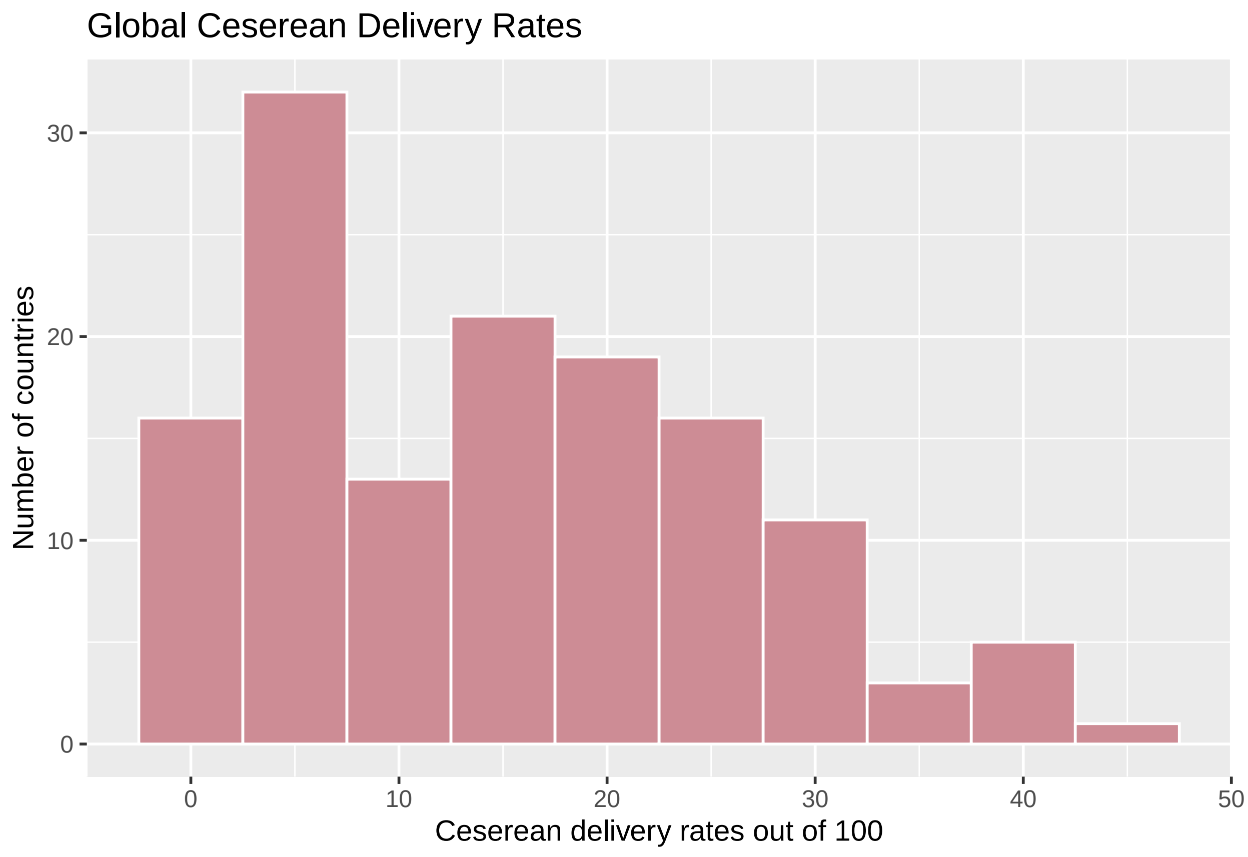
```
## [1] "Checkpoint 1 Passed: A ggplot has been defined"
## [1] "Checkpoint 2 Passed: Correct!"
## [1] "Checkpoint 3 Passed: Correct!"
## [1] "Checkpoint 4 Passed: Correct!"
## [1] "Checkpoint 5 Passed: Correct!"
##
## Problem 7
## Checkpoints Passed: 5
## Checkpoints Errored: 0
## 100% passed
## --------
## Test: PASSED
```

Polish up your plots

Your plots look great so far, but let's get them ready to present at a meeting or in a report. To do this please update the plot and axes titles (`labs`), and change the `fill` or `color` of the bars. Here is a big list of colors that you can set col and fill equal to. For example col = "orchid1" will change the color to a pinky-purple.

8. Below present your final plots with updated titles, labels, colors, and fills.To do this, extend the ggplot code you wrote above with the functions and arguments just described:

```
p8 <- ggplot(data = CS_data, aes(x = CS_rate_100)) +
geom_histogram(col = "white", binwidth = 5, fill = "lightpink3") +
labs(x = "Ceserean delivery rates out of 100", y = "Number of countries",
title = "Global Ceserean Delivery Rates")
p8
```



```
check_problem8()
```

```
## [1] "Checkpoint 1 Passed: A ggplot has been defined"
## [1] "Checkpoint 2 Passed: Correct!"
## [1] "Checkpoint 3 Passed: Correct!"
## [1] "Checkpoint 4 Passed: Correct!"
## [1] "Checkpoint 5 Passed: Correct!"
## [1] "Checkpoint 6 Passed: Correct!"
## [1] "Checkpoint 7 Passed: Correct!"
## [1] "Checkpoint 8 Passed: Correct!"
## [1] "Checkpoint 9 Passed: Correct!"
## [1] "Checkpoint 10 Passed: Correct!"
```

```
##
## Problem 8
## Checkpoints Passed: 10
## Checkpoints Errored: 0
## 100% passed
## --------
## Test: PASSED
```

Save your plots to separate files

Sometimes, you need to save plots as PNG or JPEG to include in a PowerPoint presentation or submit to a research journal for publication. The `ggsave()` function is used to save plots as separate files.

Quickly read the documentation of the `ggsave()` function by executing this code which opens a help window in the bottom right pane:

```
?ggsave
```

9. Try figuring out how to save a plot based on the documentation and running your attempts in the code chunk below. Don't worry about producing an error or saving your plot in the wrong place – that is all part of coding and learning new functions.

```r
ggsave(filename = "lab02", plot = last_plot(), device = jpeg)
```

```
## Saving 6.5 x 4.5 in image
```

When you are done, knit your R document one last time and ensure it looks good! Change anything you don't like. You can choose to knit to .docx or .pdf by selecting these options form the dropdown menu next to knit, in the case that you'd like to save the completed file onto your computer.

When you're happy with your work, please follow the instructions on the final page to submit.

```
# Just run this
total_score()
```

```
##              Test Points_Possible       Type
## Problem 1 PASSED               1 autograded
## Problem 2 PASSED               1 autograded
## Problem 3 PASSED               1 autograded
## Problem 4 PASSED               1 autograded
## Problem 5 PASSED               1 autograded
## Problem 6 PASSED               1 autograded
## Problem 7 PASSED               1 autograded
## Problem 8 PASSED               1 autograded
```

References

1. Boatin AA, Schlotheuber A, Betran AP, Moller AB, Barros AJ, Boerma T, Torloni MR, Victora CG, Hosseinpoor AR. Within country inequalities in caesarean section rates: observational study of 72 low and middle income countries. BMJ. 2018;24(360):k55. DOI: 10.1136/bmj.k55.

2. Gibbons L, Belizán JM, Lauer JA, Betrán AP, Merialdi M, Althabe F. The global numbers and costs of additionally needed and unnecessary caesarean sections performed per year: overuse as a barrier to universal coverage. World Health Organization; 2010. Available at: http://www.who.int/healthsystems/topics/financing/healthreport/30C-sectioncosts.pdf. Accessed October 4, 2015.

3. The World Bank. GDP per capita (current US$). Washington, DC; Available at: http://data.worldbank.org/indicator/NY.GDP.PCAP.CD. Accessed Sep 30, 2015.

Submission

For assignments in this class, you'll be submitting using the **Terminal** tab in the pane below. In order for the submission to work properly, make sure that:

1. Any image files you add that are needed to knit the file are in the `src` folder and file paths are specified accordingly.
2. You **have not changed the file name** of the assignment.
3. The file knits properly.

Once you have checked these items, you can proceed to submit your assignment.

1. Click on the **Terminal** tab in the pane below.
2. Copy-paste the following line of code into the terminal and press enter.

cd; cd ph142-fa20/lab/lab02; python3 turn_in.py

3. Follow the prompts to enter your Gradescope username and password. When entering your password, you won't see anything come up on the screen–don't worry! This is just for security purposes–just keep typing and hit enter.
4. If the submission is successful, you should see "Submission successful!" appear as output.
5. If the submission fails, try to diagnose the issue using the error messages–if you have problems, post on Piazza.

The late policy will be strictly enforced, **no matter the reason**, including submission issues, so be sure to submit early enough to have time to diagnose issues if problems arise.