Lab 2

Your name and student ID

today's date

Instructions

This lab uses the same data set as Lab 1 and Assignment 1. In those exercises, you investigated the distribution of global cesarean delivery rates and country's GDPs. In part I of this lab, we will look at the relationship between these variables using the skills we learned from Chapters 3 and 4. Part II is optional - this is included for those who want to learn more about simulating data in R but will not be tested.

Start by loading the required libraries, reading in the data and adding on a variable:

```r
library(dplyr)
```

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

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

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

```r
library(ggplot2)
library(readr)
library(broom)

CS_data <- read_csv("../Data/Cesarean.csv")
```

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

```r
# 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$Income_Group <- forcats::fct_relevel(CS_data$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. Make a scatter plot between `CS_rate_100` and `GDP_2006`:

```
# use this code chunk to write your code.
```

In your plot, you might notice that many of the points are condensed towards the lower left corner. And you might recall from the lab and assignment that the distributions of both cesarean delivery rate and GDP covered a wide range of values. Both of these variables are good candidates for log transformations to spread out the range of data at the lowest levels.

2. Using the `mutate()` function, add two new logged variables to the data set `CS_data`. Call the variables `log_CS` and `log_GDP`. Use base e, also know as natural logarithms, to create the logged variables:

```
# use this code chunk to write your code.
```

3. Remake the scatter plot using the logged variables:

```
# use this code chunk to write your code.
```

4. A `geom` that you have not yet learnt is `geom_smooth()`. This geom can fit a curve to the data. Extend you `ggplot()` code by adding geom__smooth() to it:

```
# use this code chunk to write your code.
```

5. Does the relationship between logged GDP and logged CS look linear?

<Replace this text with your answer.>

6. Modify your scatter plot by linking the color of the points to the variable `Income_Group`.

```
# use this code chunk to write your code.
```

Based on this colored scatter plot, it looks like the relationship is linear for those countries that are not categorized as one of the two high income categories.

7. For this lab, we would like to use linear regression. To do this, use a `dplyr` function to make a new data set called `CS_data_sub` that only contains the low-, lower-middle, and upper-middle income countries (hint: You might want to look at the data to see exactly what these levels are called in the data set):

```
# use this code chunk to write your code.
```

8. Remake the last scatter plot, this time using `CS_data_sub` to see if the relationship looks approximately linear between the logged variables:

```
# use this code chunk to write your code.
```

9. Given that the relationship is approximately linear, use linear regression to model the relationship between `log_CS` as the response variable and `log_GDP` as the explanatory variable. Don't forget to specify the correct data set!:

```
# use this code chunk to write your code.
```

10. Interpret the intercept estimate: <Replace this text with your answer.>

11. Estimate what the cesarean delivery rate would be for a country with a GDP of 2000. Outline the steps you take to calculate your answer and provide an interpretation. Round your final answer to one decimal place.

<Replace this text with your answer.>

12. Is it appropriate to use the model to predict the cesarean delivery rate for a country with a GDP of 50,000? Why or why not? Based on the relationship in the full data set, would you expect the linear model to over or under predict?

<Replace this text with your answer.>

Part II: Simulation as tool for understanding statistics

This section contains slightly advanced material for those interested in learning about using R for statistical simulations.

One very useful feature of R is the ability to create simulations, both as research and educational tools. More specifically, one can define a set of functions to generate "realistic" data, and then observe the behavior of algorithms used to estimate statistics based on the simulated data. The benefit of simulating data vs. using actual data, is that you now know the true values the statistics should take because you defined them yourself. (The downside of simulations, is they usually are an over-simplistic representation of the real process that generated the data). Simulations are used for a wide variety of reasons. For example, you could use a simulation to investigate whether the spread of an infectious disease that has occurred in the real world is aligned with a model that you have proposed.

This part of the lab begins to introduce you to the steps followed to simulate data using R. Later in the course, we will use simulations to discuss estimation, inference and sampling distributions.

For now, we generate data from a linear model $Y = a + bX + e$, where e is a random error (otherwise, all points would lie exactly on the line).

```r
# runif() makes n copies of random number between 0 and 2
# Run this line and then take a look at X by printing it to the screen
X <- runif(n = 100, min = 0, max = 2)
X # Notice that there are 100 values between 0 and 2.
```

```
##   [1] 0.69396202 0.81621646 1.73743953 1.86995750 1.04718951 1.05222157
##   [7] 0.68760881 1.98307113 0.78545096 1.39256245 0.52312851 1.63598708
##  [13] 1.19738259 0.05724475 0.41658238 1.32324030 1.87031805 0.09758983
##  [19] 1.64525351 0.23960104 0.12444077 1.47152976 1.07847021 0.38807238
##  [25] 1.49470023 1.66986651 0.02534620 1.65444502 1.33036807 0.72182572
##  [31] 1.54622898 1.12512992 0.14269930 0.82125358 0.06923359 0.37219166
##  [37] 1.47719105 1.18656679 1.30919606 1.89739052 1.93377891 1.03282789
##  [43] 1.73784949 1.77784098 0.84473036 0.28281549 0.51733539 1.83519035
##  [49] 0.45855222 0.02210140 1.09483654 0.73753599 1.94945252 1.21117697
##  [55] 1.49510079 1.69221627 1.95047565 0.75884946 1.24023138 1.85090068
##  [61] 0.66709193 0.13316609 0.79908489 0.03965545 1.58837638 0.94400963
##  [67] 1.15131744 1.28716643 1.58743529 1.79683194 0.20806298 0.28284697
##  [73] 1.16653816 0.13782095 0.16126221 1.10105080 0.81304855 0.09250481
##  [79] 0.50418043 1.91872128 0.99508854 1.02147019 1.24918331 0.35934532
##  [85] 0.08512781 0.29703720 0.14387433 0.96218668 1.69242171 0.78681022
##  [91] 1.03118725 1.17143435 1.19733737 0.93226833 1.34888780 0.41519905
##  [97] 0.80152246 0.12436230 1.86121885 0.54227487
```

```r
# Make a random error that is normally distributed
# We will learn more about the normal distribution in Chapter 11.
e <- rnorm(n = 100, mean = 0, sd = 0.25)

# Make Y, the reponse variable, a function of X and e
sim_intercept <- 0 # we arbitrarily set the intercept to 0 to begin.
sim_slope <- 0.5 # we arbitrarily set the slope to 0.5 to begin.
Y <- sim_intercept + sim_slope*X + e
dat_sim <- data.frame(Y, X)
```

11. Make a scatter plot of Y versus X, variables in the data set `dat_sim`.

```r
# use this code chunk to write your code.
```

12. Run a linear regression of Y on X.

```
# use this code chunk to write your code.
```

13. Investigate what happens if you specify `sim_slope` to equal -1 rather than 0.5. You can do this by overwriting the contents of `sim_slope` so that it equals -1 and then re-creating `Y` and `dat_sim` by copying and pasting the lines of code that define `Y` and `dat_sim`, making sure to run the new lines of code. Also re__create the plot and re-run the linear regression. What changes?

```
# use this code chunk to write your code.
```

<Replace this text with your answer.>