Announcement

Homework 10

Neal Kar (ink2105)

Announcement

Please do not add code folding (code_folding: hide) to your YAML Header or echo = FALSE to your RMD code chunk options. In order to accurately grade your HTML files we need to be able to see all of your code. Thank you!

Question 1

a)

The `south_korea-_covid19.csv` dataset contains information on the number of confirmed cases in different cities in South Korea. Read in the data and report the total number of cases in the Seoul province.

```
#Read in the dataset
skorea_covid_wide <- read_csv("data/south_korea_covid19.csv")

#Find out number of cases in Seoul province
skorea_covid_seoul <- skorea_covid_wide %>%
  group_by(province) %>%
  summarize(tot_cases = sum(confirmed)) %>%
  filter(province == "Seoul") %>%
  pull(tot_cases)
```

The total number of cases in Seoul province is 360.

b)

The variable group refers to whether the reported cases were due to group infection or not. Calculate the mean number of cases in group infections and not group infections. Does this appear to be a meaningful variable to predict the number of cases?

```
#Find mean number of cases per group infection status
skorea_covid_groupinfec <- skorea_covid_wide %>%
  group_by(group) %>%
  summarize(mean_cases = mean(confirmed))

#Pull mean cases in group infections rounded to nearest whole nu
mber
num_group_infec <- skorea_covid_groupinfec %>%
  filter(group == "TRUE") %>%
  mutate(mean_cases = round(mean_cases, 0)) %>%
  pull(mean_cases)

#Pull mean cases in non-group infections rounded to nearest whol
e number
num_nongroup_infec <- skorea_covid_groupinfec %>%
  filter(group == "FALSE") %>%
  mutate(mean_cases = round(mean_cases, 0)) %>%
  pull(mean_cases)
```

There mean number of cases from group infections is about 112 whereas the mean number of cases from non-group infections was about 53. So, the “group” variable appears to be a meaningful variable to predict the number of cases.

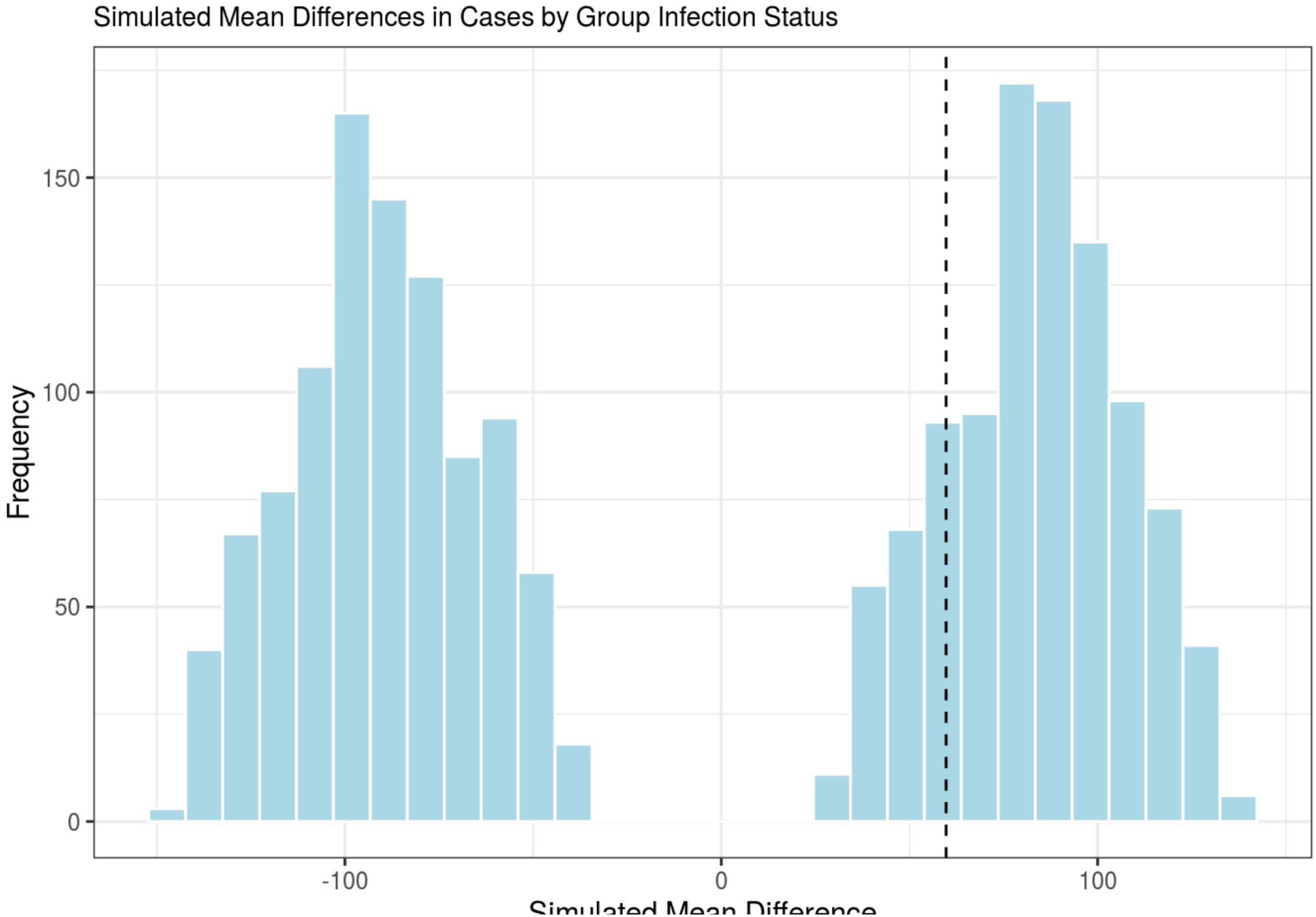
c)

Verify your hypothesis in (b) by performing a one sided permutation test to observe whether the mean cases was larger for group infection than not group infections. Perform 2000 permutations for the test. Report your p-value and conclusion.

```r
#test stat function
calculate_ts_group <- function(df){
  summary <- df %>%
    group_by(group) %>%
    summarize(mean_cases = mean(confirmed))

  group_infec <- summary %>%
    filter(group == "TRUE") %>%
    pull(mean_cases)

  nongroup_infec <- summary %>%
    filter(group == "FALSE") %>%
    pull(mean_cases)

  difference <- group_infec - nongroup_infec
  return(difference)
}

group_obs <- calculate_ts_group(skorea_covid_wide)
```

```r
#permutation function
perform_permutation_group <- function(df){
  permuted <- df %>%
    mutate(confirmed = sample(confirmed))
  return(permuted)
}

#get permuted test statistic
get_permuted_ts_group <- function(df){
  permed <- perform_permutation_group(df)
  ts <- calculate_ts_group(permed)
  return(ts)
}

#perform permutation test
group_results <- map_dbl(1:2000, function(x) get_permuted_ts_gro
up(skorea_covid_wide))
```

```
#plot results
group_tibble <- tibble(sim_stat_group = group_results)

ggplot(data = group_tibble) +
  geom_histogram(aes(x = sim_stat_group),
                 bins = 30, fill = "light blue", color = "whit
e")+
  geom_vline(aes(xintercept = group_obs),
             linetype = "dashed") +
  theme_bw() +
  theme(plot.title = element_text(size=10)) +
  labs(title="Simulated Mean Differences in Cases by Group Infec
tion Status", x="Simulated Mean Difference", y="Frequency")
```


Simulated Mean Differences in Cases by Group Infection Status
Frequency
150
100
50
0
-100
0
100
Simulated Mean Difference

Simulated Mean Difference

```
#Find p-value
greater_than_group <- group_tibble %>%
  mutate(val_greater = if_else(sim_stat_group >= group_obs, 1,
0)) %>%
  filter(val_greater == 1) %>%
  summarize(n = n()) %>%
  pull(n)

permute_pval_group <- greater_than_group/2000
```

The p-value is 0, which means there isn't enough evidence to reject the null that the mean number of cases from group infections is greater than the mean number of cases from non-group infections. In other words, it seems group infections had a greater number of mean cases than non-group infections. This supports the notion that the "group" variable would be a good predictor of cases.

d)

Create a new dichotomous variable named ‘seoul’ that identifies if a observation was in the Seoul province or outside of it.

```
skorea_covid_wide <- skorea_covid_wide %>%
  mutate(seoul = if_else(province == "Seoul", 1, 0))
```

e)

Run a new two sided permutation test to determine if the median number of confirmed cases was different between those inside the Seoul province and outside of the Seuol provide. Perform 2000 permutations for the test. Report your p-value and conclusion.

```r
#Start by creating a function to calculate the test statistic

calculate_ts_seoul <- function(df){
  summary <- df %>%
    group_by(seoul) %>%
    summarize(median_cases = median(confirmed))

  seoul_med <- summary %>%
    filter(seoul == 1) %>%
    pull(median_cases)

  not_seoul_med <- summary %>%
    filter(seoul == 0) %>%
    pull(median_cases)

  difference <- seoul_med - not_seoul_med
  return(difference)
}
```

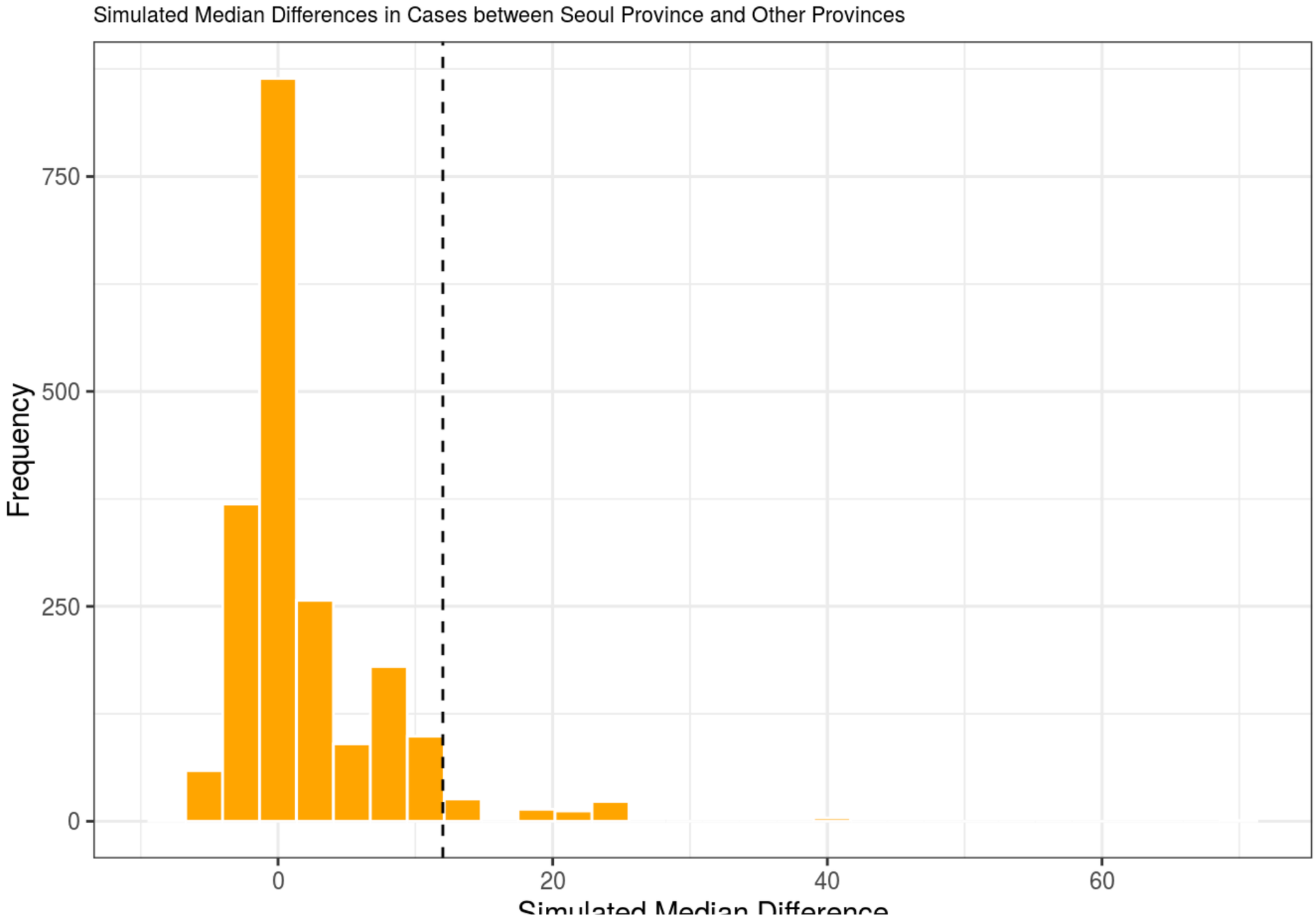
```r
obs_stat_seoul <- calculate_ts_seoul(skorea_covid_wide)

# Now create a function that performs a single permutation.

perform_permutation_seoul <- function(df){
  permuted <- df %>%
    mutate(confirmed = sample(confirmed))
  return(permuted)
}

permuted_seoul <- perform_permutation_seoul(skorea_covid_wide)


# Combining the permutation function and the calculation of the
test statistic
do_permutation_test_seoul <- function(df){
  permed <- perform_permutation_seoul(df)
  ts <- calculate_ts_seoul(permed)
  return(ts)
```

```
}

do_permutation_test_seoul(skorea_covid_wide)
```

```
## [1] 0
```

```
# Now we want to do this 2000 times:
# To repeatedly apply a function, use the map function

results_seoul <- map_dbl(1:2000, function(x) do_permutation_test
_seoul(skorea_covid_wide))

# Put into a tibble for graphing:
res_tibble_seoul <- tibble(sim_stat_seoul = results_seoul)

# Compare all of these permuted test statistics to the observed
difference in means

ggplot(data = res_tibble_seoul) +
  geom_histogram(aes(x = sim_stat_seoul), bins = 30, fill = "ora
nge", color = "white") +
  geom_vline(aes(xintercept = obs_stat_seoul), linetype = "dashe
d") +
  theme_bw() +
  theme(plot.title = element_text(size=8)) +
```

```
  labs(title="Simulated Median Differences in Cases between Seou
l Province and Other Provinces", x="Simulated Median Differenc
e",
       y="Frequency")
```


Simulated Median Differences in Cases between Seoul Province and Other Provinces
Frequency
750
500
250
0
0
20
40
60
Simulated Median Difference

Simulated Median Difference

```
# Compare simulated test statistics to observed test statistic t
o calculate p-value. The proportion of simulated test statistics
whose absolute value is greater than or equal to the observed te
st statistic will be our p-value:

greater_than_seoul <- res_tibble_seoul %>%
  mutate(abs_val_greater = if_else(abs(sim_stat_seoul) >= abs(ob
s_stat_seoul), 1, 0)) %>%
  filter(abs_val_greater == 1) %>%
  summarize(n = n()) %>%
  pull(n)

permute_pval_seoul <- greater_than_seoul/2000
```

The p-value is 0.0615, indicating there isn't enough evidence to reject the null of the median number of cases being equal between Seoul province and provinces outside of Seoul. In other words, the median number of cases is not significantly different between Seoul province and provinces outside of Seoul.

Question 2

In this question we will practice automating linear regression code using str_c(). We will use the body measures dataset which includes many continuous variables to practice with.

a)

Load the body measures data (`data/body_measures.csv`) into your R environment. Then fit a linear regression model with weight as the outcome and height and gender as predictors. Report the parameter estimates using `tidy()` and `kable()` and interpret the slope coefficient for height.

```
#Read in file
body_measures_wide <- read_csv("data/body_measures.csv")

#Create linear regression model
body_measures_lm <- lm(weight ~ height + gender,
                       data=body_measures_wide)

#Put paramater estimates into a tibble
body_measures_lm_param1 <- broom::tidy(body_measures_lm)

#Report paramater estimates
kable(body_measures_lm_param1)
```

term	estimate	std.error	statistic	p.value
(Intercept)	-56.9494889	9.4244377	-6.042747	0
height	0.7129752	0.0570661	12.493852	0

term	estimate	std.error	statistic	p.value
genderMale	8.3659935	1.0729582	7.797129	0

Controlling for gender, a 1 centimeter increase in height is associated with a 0.71 kilogram increase in weight, on average.

b)

Now write a function which will fit a linear regression model with weight as the outcome and gender and an input variable as predictors. You'll need to use `str_c()` *and* `as.formula()` *as shown in Lab 09. Your function should return a tidy table of parameter estimates from your regression.*

```
run_regression <- function(input_var) {

  outcome <- "weight"

  lm_covariate <- str_c(outcome, " ~ gender")

  lm_full <- str_c(lm_covariate, "+", input_var)

  model <- as.formula(lm_full)

  model_est <- lm(model, data = body_measures_wide) %>%
    broom::tidy()

  return(model_est)

}
```

c)

Pick 6 continuous variables from the body measures dataset and show that your function from part (b) works by using it to run separate linear regressions using each of these 6 continuous variables as predictors of weight (also adjusting for gender). Report your results in one large table or a series of smaller tables. It is not required to use map functions for this problem, but give them a try!

```
#Create vector of predictors
predictors <- c("age", "shoulder_girth", "waist_girth",
                "hip_girth", "thigh_girth", "bicep_girth_flexe
d")

#Run function on selected predictors
purrr::map_dfr(predictors,
               function(x) run_regression(input_var = x))
```

```
## # A tibble: 18 x 5
##    term              estimate std.error statistic   p.value
##    <chr>                <dbl>     <dbl>     <dbl>     <dbl>
##  1 (Intercept)          56.2      1.48      38.0   3.39e-150
##  2 genderMale           17.1      0.896     19.1   1.57e- 61
##  3 age                   0.153    0.0466     3.29  1.07e-  3
##  4 (Intercept)         -60.2      4.39     -13.7   1.23e- 36
##  5 genderMale           -1.97     0.904     -2.18  2.99e-  2
##  6 shoulder_girth        1.20     0.0436    27.6   4.86e-103
##  7 (Intercept)         -10.5      2.17      -4.83  1.83e-  6
##  8 genderMale            2.55     0.674      3.78  1.74e-  4
##  9 waist_girth           1.02     0.0306    33.2   4.06e-129
## 10 (Intercept)         -68.6      3.02     -22.7   6.07e- 79
## 11 genderMale           14.7      0.420     35.0   7.26e-137
## 12 hip_girth            1.35     0.0315    42.9   2.31e-170
## 13 (Intercept)         -44.5      3.38     -13.2   3.39e- 34
## 14 genderMale           18.8      0.524     36.0   2.98e-141
## 15 thigh_girth          1.84     0.0587    31.3   3.87e-120
## 16 (Intercept)         -13.9      2.96      -4.70  3.39e-  6
```

```
## 17 genderMale              0.824    0.885      0.931 3.52e-  1
## 18 bicep_girth_flexed      2.65     0.104     25.4   1.98e- 92
```

d)

Which variables from (c) were significantly associated with weight? If any of them were associated with weight, pick one and write an interpretation of its slope estimate here.

All of the selected predictors were significantly associated with weight. Controlling for gender, a one year increase in age is associated with a 0.15 kilogram increase in weight, on average.

e)

Use `str_c()` and `as.formula()` to write a new function which takes arguments for a predictor and an outcome and fits a regression model with the specified outcome and predictor (also adjusting for gender) using the body measures dataset.

Your function should output a tidy data frame of parameter estimates. Test your function twice, using different outcomes and predictors (make sure to only use continuous variables as outcome or predictor).

```r
#Write function
run_regression_v2 <- function(outcome, predictor) {

  lm_covariate <- str_c(outcome, " ~ gender")

  lm_full <- str_c(lm_covariate, "+", predictor)

  model <- as.formula(lm_full)

  model_est <- lm(model, data = body_measures_wide) %>%
    broom::tidy()

  return(model_est)

}

#Test function
```

```
run_regression_v2(outcome = "bicep_girth_flexed",
                  predictor = "wrist_girth")
```

```
## # A tibble: 3 x 5
##   term        estimate std.error statistic  p.value
##   <chr>          <dbl>     <dbl>     <dbl>    <dbl>
## 1 (Intercept)    -3.36     1.67      -2.02 4.42e- 2
## 2 genderMale      1.85     0.304      6.09 2.20e- 9
## 3 wrist_girth     2.09     0.110     18.9  7.95e-61
```

```
run_regression_v2(outcome = "thigh_girth",
                  predictor = "ankle_girth")
```

```
## # A tibble: 3 x 5
##   term        estimate std.error statistic  p.value
##   <chr>          <dbl>     <dbl>     <dbl>    <dbl>
## 1 (Intercept)    24.8      2.23      11.1  1.01e-25
## 2 genderMale     -3.69     0.390     -9.46 1.20e-19
## 3 ankle_girth     1.53     0.105     14.6  1.48e-40
```

Question 3

a)

Download the latest US State-level COVID-19 dataset from the New York Times data portal from https://github.com/nytimes/covid-19-data (https://github.com/nytimes/covid-19-data). Try this neat trick: go to the website in your browser and right-click on the "Raw CSV" link and select "Copy Link Address".

This will copy a URL for the data into your clipboard, which you can paste directly into the `read_csv()` function. If you can't get this to work, you can download the data manually and upload it into RStudio Cloud.

```
state_covid_long <- read_csv("https://raw.githubusercontent.com/
nytimes/covid-19-data/master/us-states.csv")
```

b)

Calculate the mean number of new COVID cases reported in California from 03/31/20 to 04/06/20. In order to do this:

(1) First filter the COVID State dataset to contain only information from California. (2) Next, arrange the observations according to date. (3) Create a new variable called `new_cases` by using `mutate(new_cases = cases - lag(cases))`. The `lag()` function will look backward one observation to obtain cases from the previous day. (4) Filter the data frame to only include dates from 03/31/20 to 04/06/20.

```
#Manipulate dataframe and pull new_cases variable
covid_CA_long <- state_covid_long %>%
  filter(state == "California") %>%
  arrange(date) %>%
  mutate(new_cases = cases - lag(cases)) %>%
  filter(date >= "2020-03-31" & date <= "2020-04-06")

#Calculate the mean
covid_CA_mean <- covid_CA_long %>%
  pull(new_cases) %>%
  mean()
```

c)

Compute and report the mean and the 95% confidence interval (using the standard formulate for SE) for new cases in California over the time period from 03/31/20 and 04/06/20.

```
### 95% CI = mean +/- t*SE ###

#Compute critical value
t_CA <- qt(1 - (0.05 / 2), nrow(covid_CA_long) - 1)

#Compute standard deviation of new_cases
sd_CA <- sd(covid_CA_long$cases)

#Compute standard error of new_cases
se_CA <- sd_CA / sqrt(nrow(covid_CA_long))

#Compute CI
lower_95_ci_CA <- covid_CA_mean - (t_CA * se_CA)

upper_95_ci_CA <- covid_CA_mean + (t_CA * se_CA)
```

The mean daily number of new cases in California from 3/31/2020-4/6/2020 was about 1266 with a 95% confidence interval of (-1320, 3852).

d)

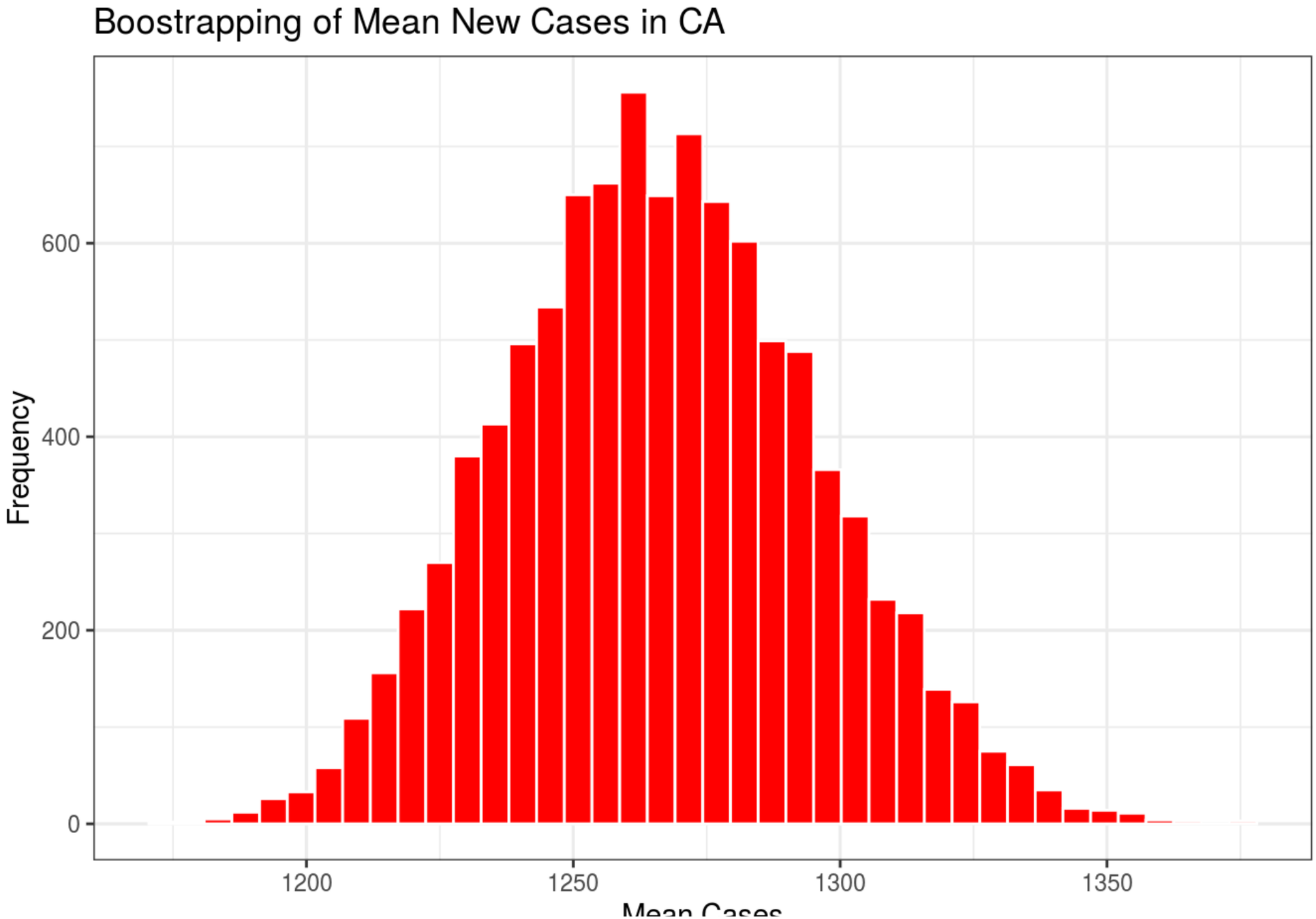
Compute and report the mean and the 95% confidence interval (using bootstrapping to obtain the SE) for new cases in California over the time period from 03/31/20 and 04/06/20. Use 10000 bootstrapped datasets to obtain the SE, and include a histogram of the distribution of bootstrapped mean case values.

```r
#First, create the dataframe from which to sample
covid_CA_filtered_long <- state_covid_long %>%
  filter(state == "California") %>%
  arrange(date) %>%
  mutate(new_cases = cases - lag(cases)) %>%
  filter(date >= "2020-03-31" & date <= "2020-04-06")


# Create function that gets statistic we want from a data frame
get_statistic <- function(df){

  test_df <- df %>%
    pull(new_cases)

  mean_new_cases <- mean(test_df)

  return(mean_new_cases)

}
```

```r
#Create bootstrapping function
create_boot_sample <- function(df){

  boot_df <- df %>%
    mutate(new_cases = sample(new_cases, replace = TRUE))

  return(boot_df)

}


#Create function to obtain a statistic from the bootstrapped sam
ple (combine previous two functions)
get_bootstrap_stat <- function(df){

  boot_df <- create_boot_sample(df)

  boot_stat <- get_statistic(boot_df)
```

```
  return(boot_stat)
}

#Run the function 10,000 times on the data frame
bootstrapped_stats_CA <- map_dbl(1:10000, function(x) get_bootst
rap_stat(covid_CA_filtered_long))

#Put the bootstrapped results into a tibble
stats_CA_tib <- tibble(run = 1:10000,
                       mean = bootstrapped_stats_CA)

#Compute bootstrapped mean
mean_bootstrap_CA <- stats_CA_tib %>%
  pull(mean) %>%
  mean()

#Compute bootstrapped 95% CI
t_bootstrap_CA <- qt(1 - (0.05/2),
```

```
                        nrow(covid_CA_filtered_long) - 1)

se_boostrap_CA <- sd(bootstrapped_stats_CA)

lower_95_ci_bootstrap_CA <- mean_bootstrap_CA -
  t_bootstrap_CA*se_boostrap_CA

upper_95_ci_bootstrap_CA <- mean_bootstrap_CA +
  t_bootstrap_CA*se_boostrap_CA

#Produce graph of distribution of means
ggplot(data = stats_CA_tib) +
  geom_histogram(aes(x = mean), bins = 40, fill = "red",
                 color = "white") +
  theme_bw() +
  labs(title="Boostrapping of Mean New Cases in CA",
       x="Mean Cases", y="Frequency")
```


Boostrapping of Mean New Cases in CA
Frequency
600
400
200
0
1200
1250
1300
1350
Mean Cases

The mean number of new cases from 3/31/2020-4/6/2020 in California after bootstrapping is about 1266 with a 95% confidence interval of (1195, 1337).

e)

Compare the confidence intervals in parts (c) and (d). Are they the same? Different? If they are different, which confidence interval is wider?

The mean and confidence interval after bootstrapping is close to the mean and confidence interval prior to bootstrapping. In fact, the mean is almost the same after bootstrapping, and the confidence interval before bootstrapping is only a little wider.

Question 4

The following two problems will walk you through how to create a map of COVID confirmed case data using ggplot(). Creating maps is a complicated topic, but there are some straightforward tools available for creating simple chloropleth maps in R.

a)

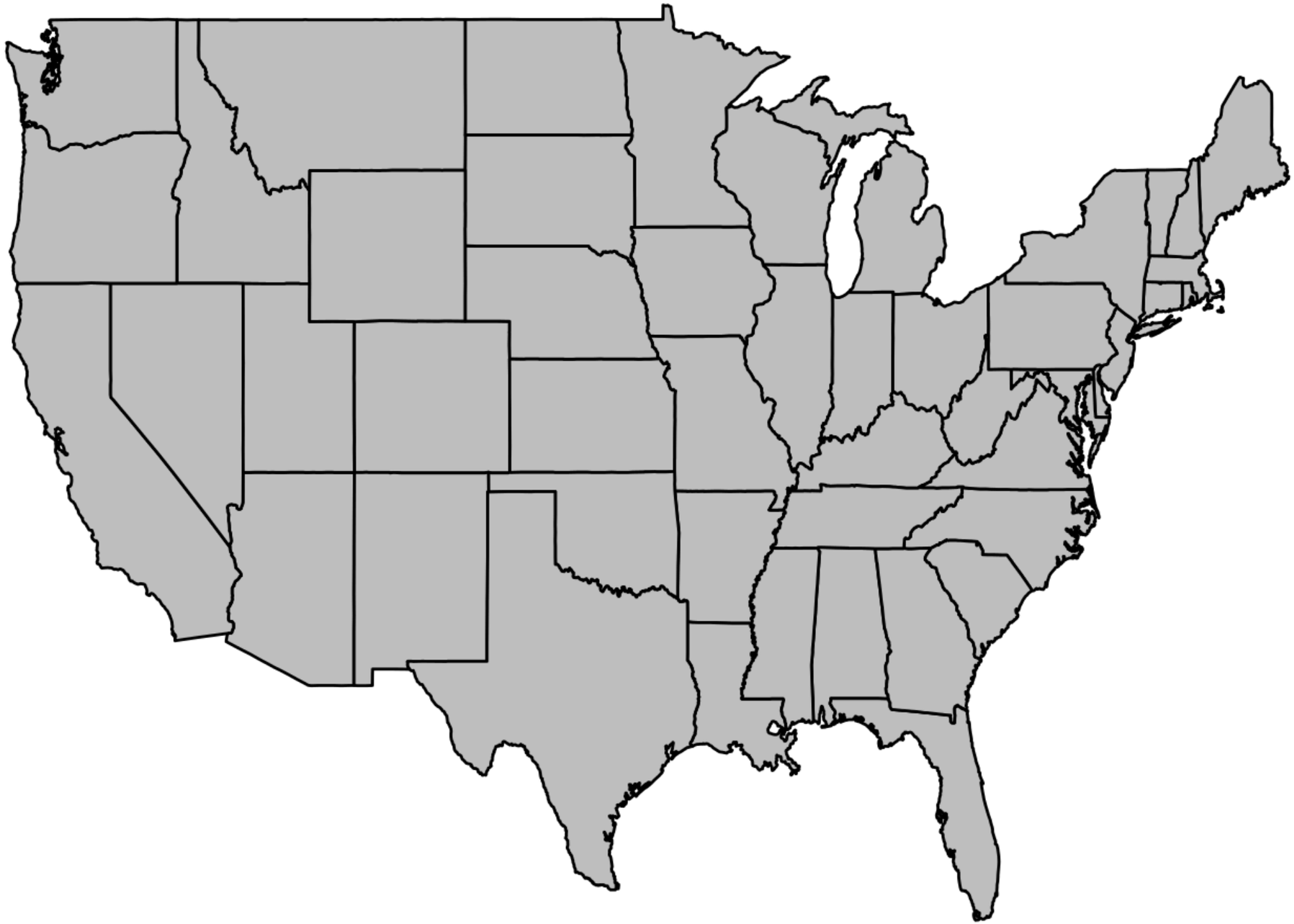
Begin by running the code below to load in data about states into `state_maps` using the `map_data("state")` function.

Then create a new `ggplot()` using this data and add a `geom_polygon()` statement which maps x to `long` (longitude), y to `lat` (latitude) and group to `group`.

Finally, add a `theme_void()` statement to your `ggplot()`. You should see a nifty map of the US!

Now try setting the color and fill aesthetics for this map – what do these options do?

```
#Read in data
state_maps <- map_data("state")

#Create ggplot
ggplot(data = state_maps) +
  geom_polygon(aes(x=long, y=lat, group=group), fill="grey",
                color="black") +
  theme_void()
```



The fill aesthetic makes each state the specified color, and the color statement draws borders around each state in the specified color.

b)

Next filter the New York Times COVID-19 State case data (from Question 3) to only include the most recent day. Once you have done this, present this dataset using DT::datatable().

```
state_covid_recent <- state_covid_long %>%
  arrange(date, state) %>%
  filter(date == max(date))

DT::datatable(state_covid_recent)
```

Show 10 **entries** **Search:**

	date	state	fips	cases	deaths
1	2020-04-12	Alabama	01	3583	93
2	2020-04-12	Alaska	02	270	6
3	2020-04-12	American Samoa	60	0	0
4	2020-04-12	Arizona	04	3539	117
5	2020-04-12	Arkansas	05	1280	27
6	2020-04-12	California	06	23323	676
7	2020-04-12	Colorado	08	7303	290
8	2020-04-12	Connecticut	09	12035	554

	date	state	fips	cases	deaths
9	2020-04-12	Delaware	10	1625	35
10	2020-04-12	District of Columbia	11	1875	50

Showing 1 to 10 of 56 entries

Previous 1 2 3 4 5 6 Next

c)

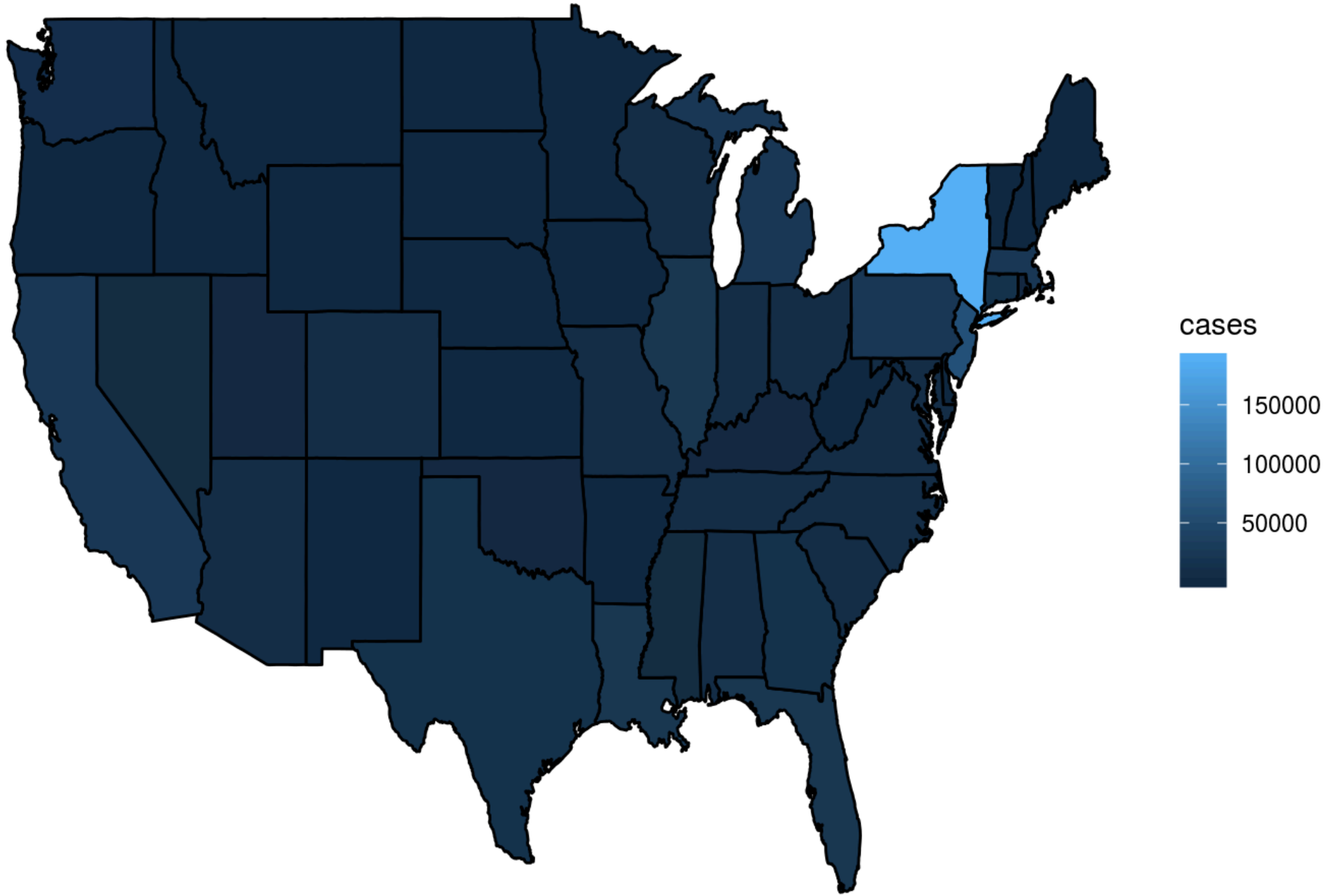
In order to create a map, we need to join our COVID-19 data from (b) to our `state_maps` *data frame. Do this by using a join that will preserve all the states in the* `state_maps` *data frame, regardless of whether they are present in our COVID-19 data. Take a close look at how the states are recorded in the* `state_maps` *data frame versus the COVID-19 data frame – you may have to do some string processing to get them to join correctly.*

```
#Make state variable in state_maps df match that of covid df
state_maps_updated <- state_maps %>%
  mutate(region = str_to_title(region)) %>%
  rename(state = region)

#Join data frames together
joined_df_state <- left_join(state_maps_updated,
                             state_covid_recent, by = "state")
```

d)

Now we can plot a map of most recent cumulative cases in each state! Use your `ggplot()` *code from (a) with your new joined data frame from (c). If you map the fill aesthetic to cases, you should see a map with each state filled in with a color corresponding to the number of cumulative cases. What's going on with this map? Are you seeing any real variation in cases?*

```
ggplot(data = joined_df_state) +
  geom_polygon(aes(x=long, y=lat, group=group, fill=cases),
               color="black") +
  theme_void()
```


cases
150000
100000
50000

Each state is colored blue, but the darkness of a state’s color depends on the number of cumulative cases in that state. A lighter blue color correlates with a higher number of cases. It’s hard to distinguish a variation in cases by state aside from New York having the most cases in the country.

e)

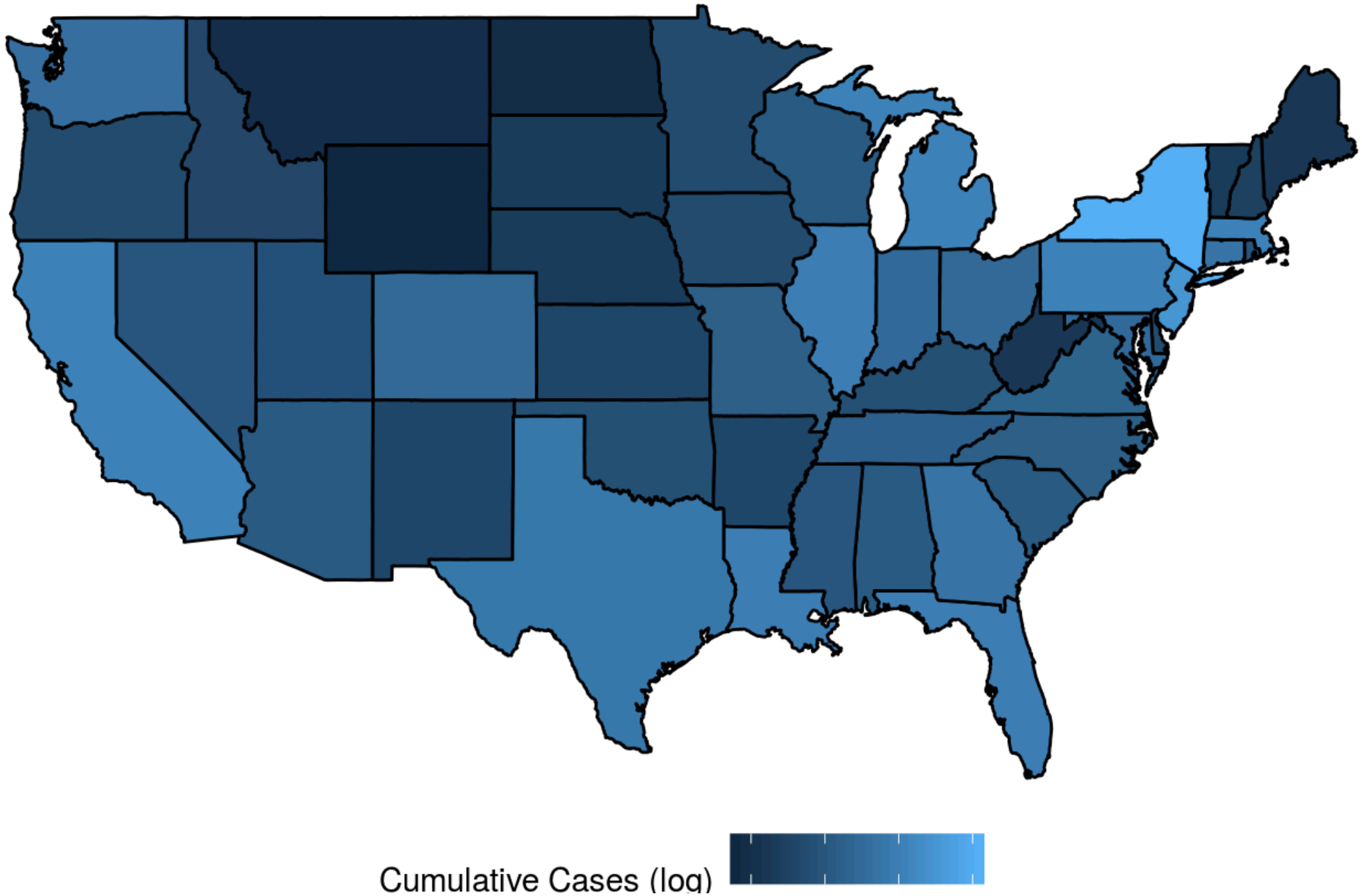
In order to visualize more variation between states, recreate the map from (d) using log cases. Can we see more variation in cases now?

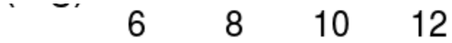
Add a title to your graph and to the legend.

```
#Create new variable with log(cases)
joined_df_state <- joined_df_state %>%
  mutate(log_cases = if_else(is.infinite(log(cases)),
                             NA_real_, log(cases)))



#Create ggplot
ggplot(data = joined_df_state) +
  geom_polygon(aes(x=long, y=lat, group=group, fill=log_cases),
               color="black") +
  theme_void() +
  theme(legend.position = "bottom") +
  labs(title="Cumulative COVID-19 Cases in the U.S.",
       subtitle = "Through 4/12/2020",
       fill="Cumulative Cases (log)")
```

Cumulative COVID-19 Cases in the U.S.

Through 4/12/2020





Yes, we can now see more variation between states.

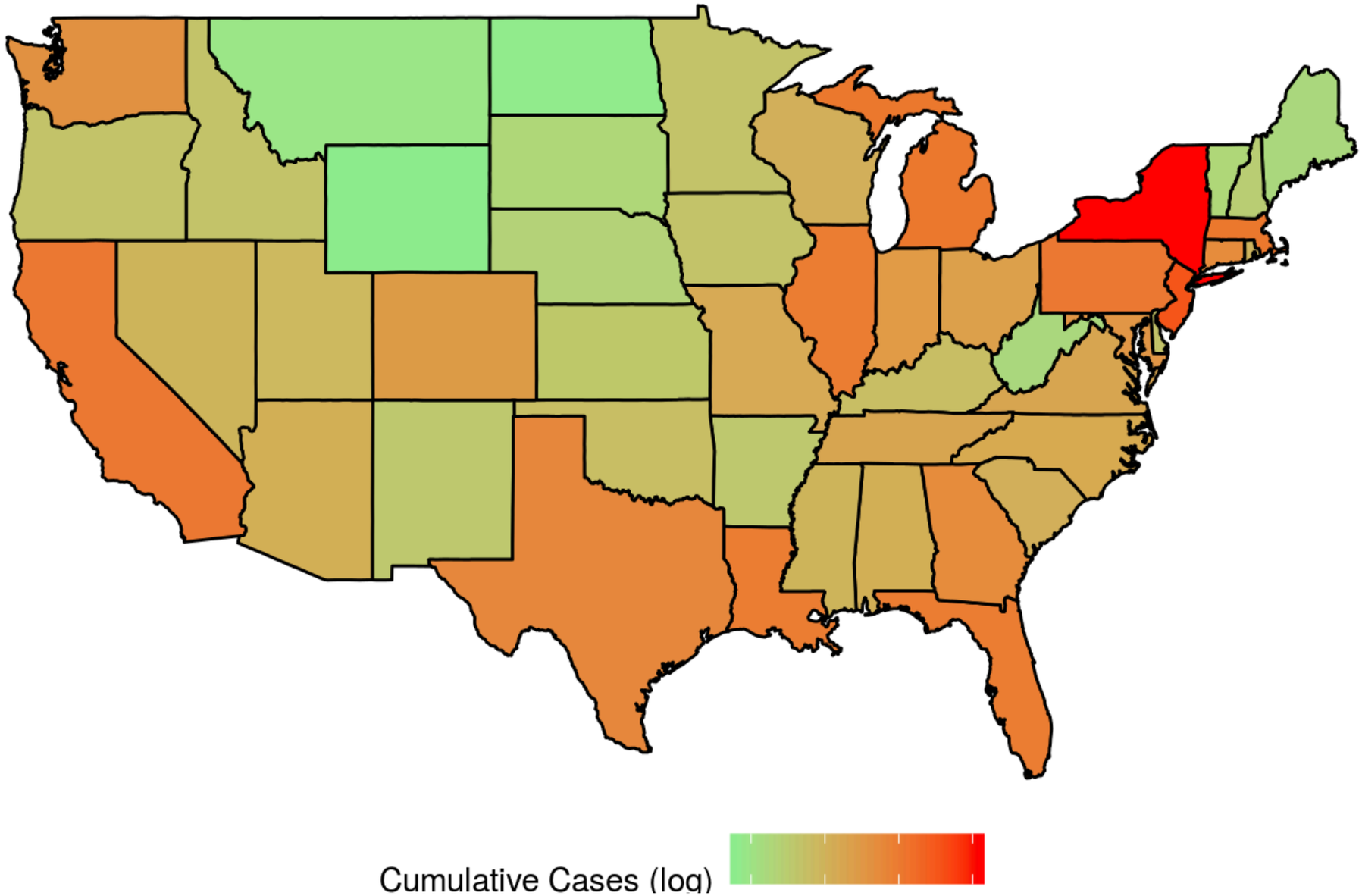
f)

Update your graph from (e) with a new color scheme. Use `scale_fill_gradient()` *with* `low` *and* `high` *arguments to create a gradient scale from one color to another. Check out https://colorbrewer2.org/#type=sequential&scheme=BuGn&n=3 (https://colorbrewer2.org/#type=sequential&scheme=BuGn&n=3) or https://htmlcolorcodes.com/ (https://htmlcolorcodes.com/) to select start and end colors. You can use hex codes like* `#99d8c9` *by putting them in quotes:* `low = "#99d8c9"`.

```
ggplot(data = joined_df_state) +
  geom_polygon(aes(x=long, y=lat, group=group, fill=log_cases),
               color="black") +
  theme_void() +
  theme(legend.position = "bottom") +
  scale_fill_gradient(low = "light green", high = "red") +
  labs(title="Cumulative COVID-19 Cases in the U.S.",
       subtitle = "Through 4/12/2020",
       fill="Cumulative Cases (log)")
```

Cumulative COVID-19 Cases in the U.S.

Through 4/12/2020



6 8 10 12

Question 5

Now let's make some graphs of county-level COVID-19 cases.

a)

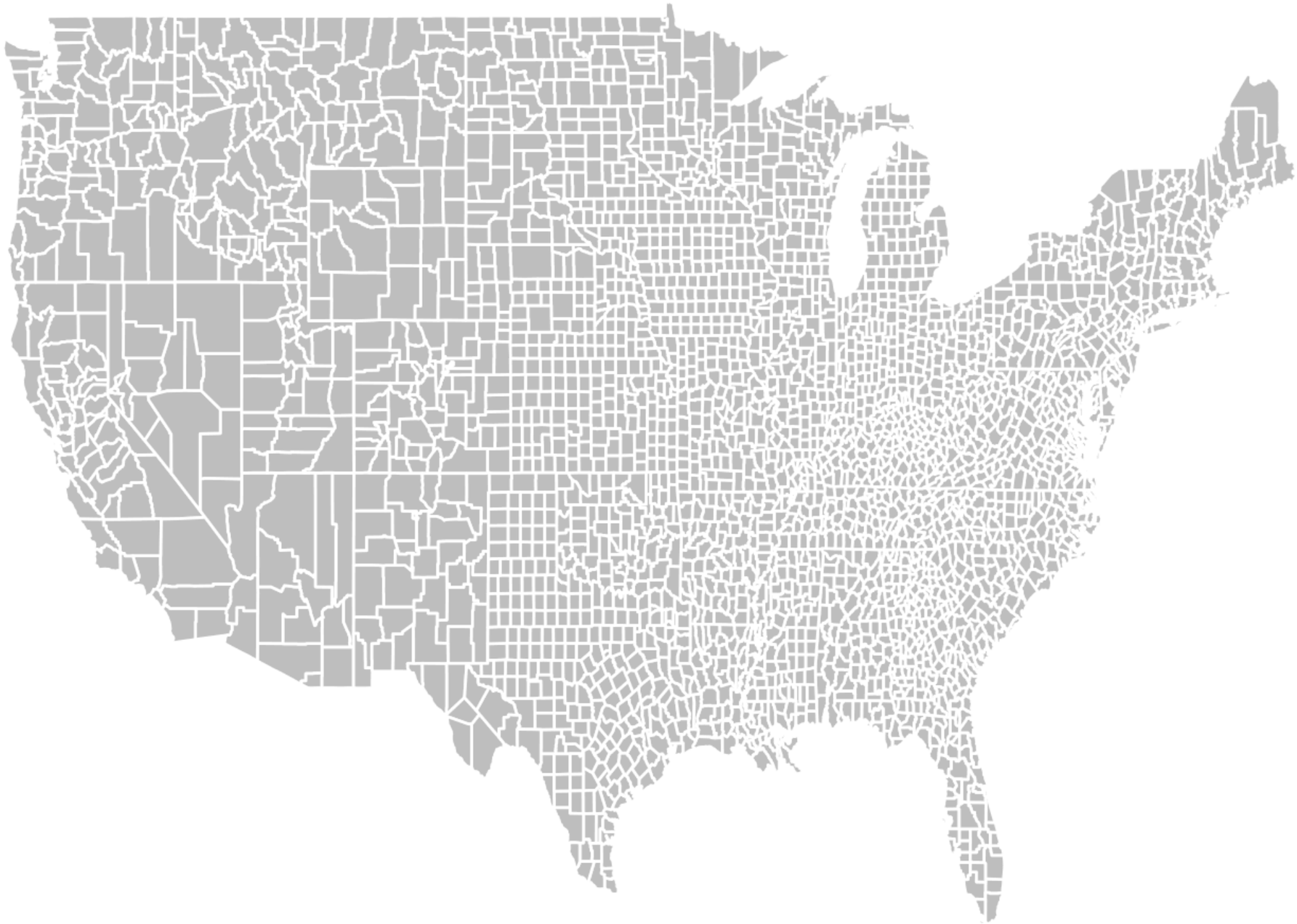
Download the county-level COVID data from the New York Times data repository: https://github.com/nytimes/covid-19-data (https://github.com/nytimes/covid-19-data). You can use the same direct-download from URL method used in problem (3). Additionally, load county map data into the `county_map` *object by running the* `map_data("county")` *code below.*

```
#Create COVID county level df
county_covid_long <- read_csv("https://raw.githubusercontent.co
m/nytimes/covid-19-data/master/us-counties.csv")

#Create map df
county_maps <- map_data("county")
```

b)

Use what you learned from problem (4) to create a map of all the counties in the US. Set the fill aesthetic to a color of your choice and the color aesthetic to white.

```
ggplot(data = county_maps) +
  geom_polygon(aes(x=long, y=lat, group=group), fill="grey",
                color="white") +
  theme_void()
```



c)

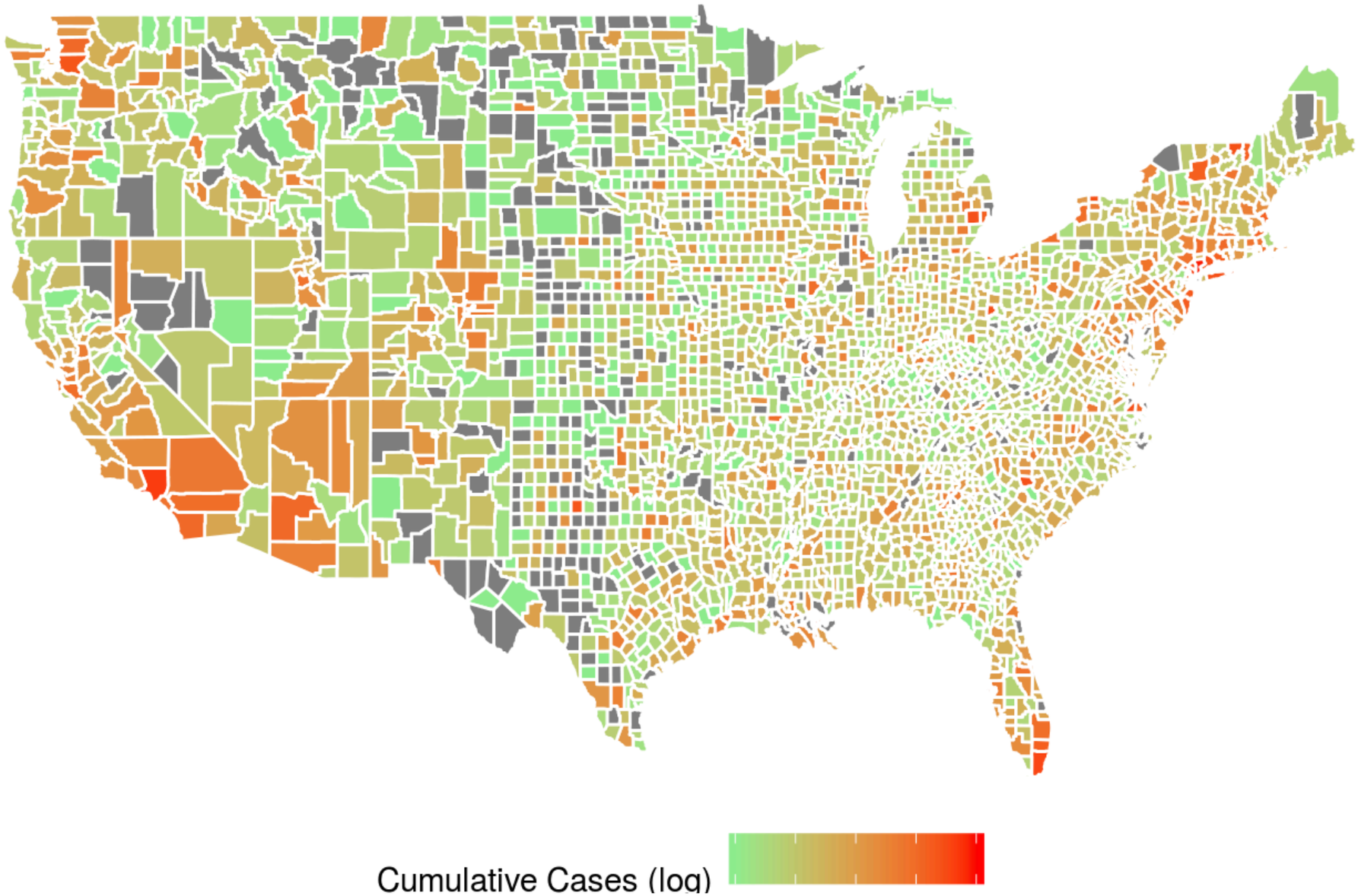
Join the COVID County-level data to the `county_maps` *data frame, just like you did in problem (4). Be careful about matching the county and state names! Then plot a map of the entire USA filled in by log(cases) for the most recent day you have data available. There will be a lot of empty space on your graph because many counties have no case data – you can make the graph look better by first layering a* `geom_polygon()` *statement with a set fill aesthetic like the one in part (b), and then adding your* `geom_polygon()` *statement with fill mapped to log cases.*

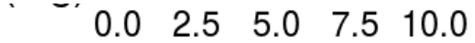
```
#Keep most recent date
county_covid_recent <- county_covid_long %>%
  arrange(date, state, county) %>%
  filter(date == max(date))

#Make state variable in state_maps df match that of covid df
county_maps_updated <- county_maps %>%
  mutate(subregion = str_to_title(subregion)) %>%
  rename(county = subregion)

#Join data frames together
joined_df_county <- left_join(county_maps_updated,
                              county_covid_recent, by = "count
y")

#Create new variable with log(cases)
joined_df_county <- joined_df_county %>%
  mutate(log_cases = if_else(is.infinite(log(cases)),
                              NA_real_, log(cases)))
```

```
#Create ggplot
ggplot(data = joined_df_county) +
  geom_polygon(aes(x=long, y=lat, group=group), fill="grey",
               color="white") +
  geom_polygon(aes(x=long, y=lat, group=group, fill=log_cases),
               color="white") +
  theme_void() +
  theme(legend.position = "bottom") +
  scale_fill_gradient(low = "light green", high = "red") +
  labs(title="Cumulative COVID-19 Cases in the U.S.",
       subtitle = "Through 4/12/2020",
       fill="Cumulative Cases (log)")
```

Cumulative COVID-19 Cases in the U.S.

Through 4/12/2020





d)

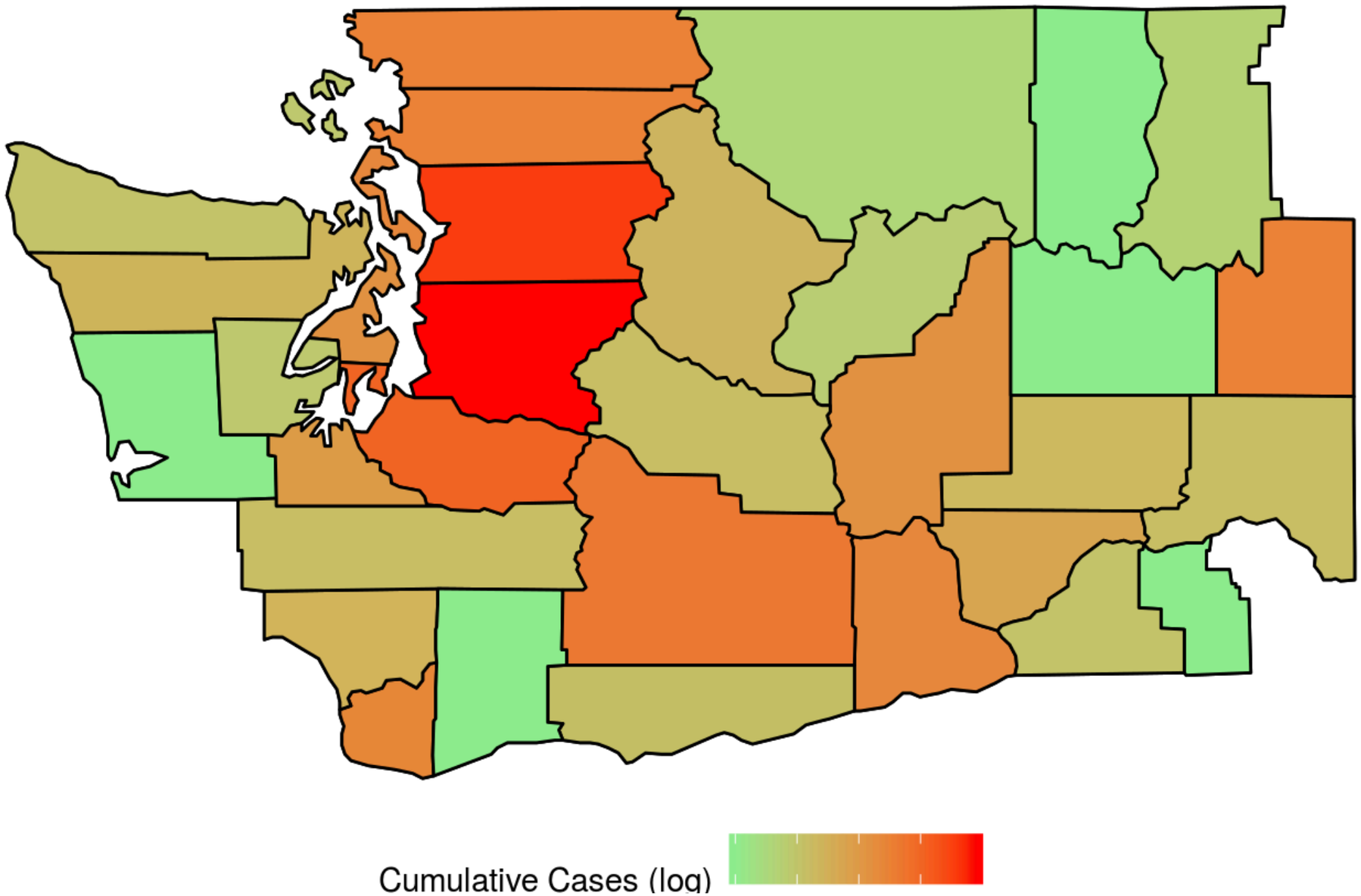
Use `filter()` and `ggplot()` to make a map of (log) cumulative cases by county in Washington State on 04/01/20. Remember to use `theme_void()`, and map the log cases to see variation across counties. Also add an informative title.

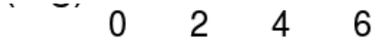
```
#Create new covid df
county_covid_WA <- county_covid_long %>%
  filter(state == "Washington" & date == "2020-04-01") %>%
  mutate(log_cases = if_else(is.infinite(log(cases)),
                             NA_real_, log(cases)))

#Create new maps df
county_map_WA <- county_maps_updated %>%
  mutate(region = str_to_title(region)) %>%
  rename(state = region) %>%
  filter(state == "Washington")

#Create new joined df
joined_df_WA <- inner_join(county_map_WA, county_covid_WA,
                           by = c("county", "state"))

#Create ggplot
ggplot(data = joined_df_WA) +
  geom_polygon(aes(x=long, y=lat, group=group), fill="grey",
```

```
              color="black") +
geom_polygon(aes(x=long, y=lat, group=group, fill=log_cases),
              color="black") +
theme_void() +
theme(legend.position = "bottom") +
scale_fill_gradient(low = "light green", high = "red") +
labs(title="Cumulative COVID-19 Cases in Washington State",
     subtitle = "Through 4/1/2020",
     fill="Cumulative Cases (log)")
```

Cumulative COVID-19 Cases in Washington State

Through 4/1/2020





e)

Write a function that will create a county-level map of (log) COVID cumulative cases for a given state on a given date. Use `str_c()` *to include the state in the title of your graph and the date in the subtitle of your graph. Try to come up with a clever way to keep your state title capitalized! Show us your function works with two states (and dates) of your choice.*

```
#Create function
plot_county_covid <- function(state, date) {

  date = lubridate::mdy(date)

  #Create new covid df
  county_covid <- county_covid_long %>%
    filter(county_covid, state == state & date == date) %>%
    mutate(county_covid,
           log_cases=if_else(is.infinite(log(cases)),
                             NA_real_, log(cases)))

  #Create new maps df
  county_map <- county_maps_updated %>%
    mutate(region = str_to_title(region)) %>%
    rename(state = region) %>%
    filter(state == state)

  #Create new joined df
```

```
  joined_df <- inner_join(county_map, county_covid,
                          by = c("county", "state"))

  #Create ggplot
  plot <- ggplot(data = joined_df) +
    geom_polygon(aes(x=long, y=lat, group=group), fill="grey",
                 color="black") +
    geom_polygon(aes(x=long, y=lat, group=group, fill=log_case
s),
                 color="black") +
    theme_void() +
    theme(legend.position = "bottom") +
    scale_fill_gradient(low = "light green", high = "red") +
    labs(title=str_c("Cumulative COVID-19 Cases in ", state),
         subtitle = str_c("Through ", date),
         fill="Cumulative Cases (log)")

  #Rteurn the plot
  return(plot)
```

```
}

#Test function

### Whenever I tested the function, R Studio would crash or it w
ould give some error message that didn't makes sense (e.g. "%>%"
is an invalid function)

#New York on 4/4/2020
#plot_county_covid("New York", "4/4/2020")

#Connecticut on 4/12/2020
#plot_county_covid("Connecticut", "4/12/2020")
```