Homework 6 Solution

Disclaimer

This homework solution is for the sole benefit of students taking Stat 220 from Prof. Bastola during Winter term 2024. Dissemination of this solution to people who are not registered for this course is not permitted and will be considered grounds for Academic Dishonesty for the all individuals involved in the giving and receiving of the solution.

Assignment prompt

Problem 1: Crimes

Scrape the table of data found at https://en.wikipedia.org/wiki/List_of_United_States_cities_by_crime_rate (https://en.wikipedia.org/wiki/List_of_United_States_cities_by_crime_rate) and create a plot showing property crime rate (total property crime) vs. violent crime rate (total violent crime). Identify outlier cities by using a plotting command similar to the one below. (Don't blindly use this without thinking about the column names.)

```
> ggplot(crimes, aes(x = violent_crime, y = property_crime, label = city)) +
+     geom_point() +
+     geom_text(
+       data = filter(crimes, violent_crime > 1500 | property_crime > 6500),
+       check_overlap = TRUE, size = 2.5, nudge_y = 40
+     )
```

Hints:

- After reading in the table using `html_table()`, create a data frame with just the columns you want using column numbers. Otherwise, R gets confused (and will likely crash) since it appears as if several columns all have the same column name. It may also be useful to use `tibble::as_tibble(.name_repair = "unique")` for duplicate column names and `janitor::clean_names()` for clean names. Use informative column names, get rid of unneeded rows, parse columns into proper format, etc.

answer:

```
> city_crimes <- read_html("https://en.wikipedia.org/wiki/List_of_United_States_cities_by_crime_rate")
>
> table_list <- city_crimes %>%
+   html_nodes("table") %>%
+   html_table()
>
> str(table_list[[1]])
tibble [102 × 14] (S3: tbl_df/tbl/data.frame)
 $ State                                : chr [1:102] "State" "State" "Alabama" "Alaska" ...
 $ City                                 : chr [1:102] "City" "City" "Mobile3" "Anchorage" ...
 $ Popul.                               : chr [1:102] "Popul." "Popul." "248,431" "296,188" ...
 $ Yearly Crime Rates per 100,000 people: chr [1:102] "Total" "Total" "6217.02" "6640.04" ...
 $ Yearly Crime Rates per 100,000 people: chr [1:102] "Violent crime" "Murder andNonnegligentmanslaughter" "20.1
3" "9.12" ...
 $ Yearly Crime Rates per 100,000 people: chr [1:102] "Violent crime" "Rape1" "58.16" "132.01" ...
 $ Yearly Crime Rates per 100,000 people: chr [1:102] "Violent crime" "Robbery" "177.11" "262.67" ...
 $ Yearly Crime Rates per 100,000 people: chr [1:102] "Violent crime" "Aggravatedassault" "485.85" "799.49" ...
 $ Yearly Crime Rates per 100,000 people: chr [1:102] "Violent crime" "Total" "740.25" "1,203.29" ...
 $ Yearly Crime Rates per 100,000 people: chr [1:102] "Arson²" "Arson²" "22.94" "20.93" ...
 $ Yearly Crime Rates per 100,000 people: chr [1:102] "Property crime" "Burglary" "1,216.84" "748.17" ...
 $ Yearly Crime Rates per 100,000 people: chr [1:102] "Property crime" "Larcenytheft" "3,730.21" "3,619.66" ...
 $ Yearly Crime Rates per 100,000 people: chr [1:102] "Property crime" "Motorvehicletheft" "506.78" "1,047.98"
...
 $ Yearly Crime Rates per 100,000 people: chr [1:102] "Property crime" "Total" "5,453.83" "5,415.82" ...
```

The first table is the target, but upon closer inspection the header was read in as a header and the first row. Also, we don’t have unique columns names.

```
> head(table_list[[1]])
```

State	**City**	**Popul.**	**Yearly Crime Rates per 100,000 people**
<chr>	<chr>	<chr>	<chr>
State	City	Popul.	Total
State	City	Popul.	Total
Alabama	Mobile3	248,431	6217.02

State <chr>	City <chr>	Popul. <chr>	Yearly Crime Rates per 100,000 people <chr>
Alaska	Anchorage	296,188	6640.04
Arizona	Chandler	249,355	2589.08
Arizona	Gilbert	242,090	1483.75

6 rows | 1-4 of 14 columns

To avoid issues with non-unique column names, let’s first extract the columns of interest:

```
> crimes <- table_list[[1]][, c(1:3, 9, 13)] %>%
+     tibble::as_tibble(.name_repair = "unique") %>%
+   janitor::clean_names() %>%
+   rename(violent_crime = yearly_crime_rates_per_100_000_people_4,
+          property_crime = yearly_crime_rates_per_100_000_people_5)
>
> crimes
```

state <chr>	city <chr>	popul <chr>	violent_crime <chr>	property_crime <chr>
State	City	Popul.	Violent crime	Property crime
State	City	Popul.	Total	Motorvehicletheft
Alabama	Mobile3	248,431	740.25	506.78
Alaska	Anchorage	296,188	1,203.29	1,047.98
Arizona	Chandler	249,355	259.47	149.18
Arizona	Gilbert	242,090	85.51	55.76
Arizona	Glendale	249,273	488.22	466.56
Arizona	Mesa	492,268	415.83	179.58
Arizona	Phoenix	1,608,139	760.93	465.46
Arizona	Scottsdale	251,840	157.24	97.68

1-10 of 102 rows Previous **1** 2 3 4 5 6 ... 11 Next

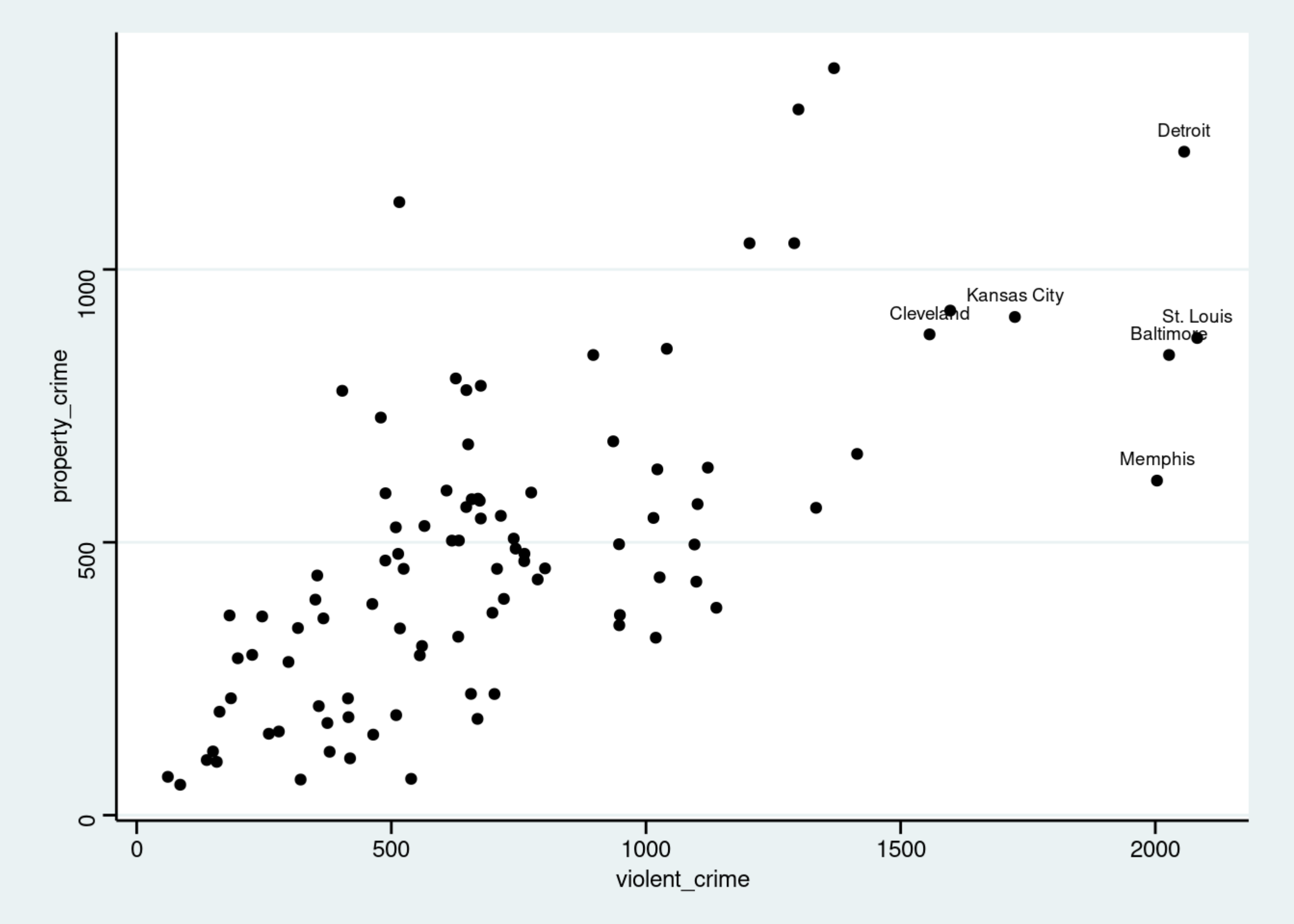
Now, let’s do the rest and plot the data:

```
> crimes <- crimes %>%
+   slice(-c(1:2)) %>%
+   mutate_at(vars(3:5), parse_number)
> crimes
```

state <chr>	**city** <chr>	**popul** <dbl>	**violent_crime** <dbl>	**property_crime** <dbl>
Alabama	Mobile3	248431	740.25	506.78
Alaska	Anchorage	296188	1203.29	1047.98
Arizona	Chandler	249355	259.47	149.18
Arizona	Gilbert	242090	85.51	55.76
Arizona	Glendale	249273	488.22	466.56
Arizona	Mesa	492268	415.83	179.58
Arizona	Phoenix	1608139	760.93	465.46
Arizona	Scottsdale	251840	157.24	97.68
Arizona	Tucson	532323	801.77	452.17
California	Anaheim	353400	354.56	439.16

1-10 of 100 rows Previous **1** 2 3 4 5 6 ... 10 Next

```
> ggplot(crimes, aes(x = violent_crime, y = property_crime, label = city)) +
+     geom_point() +
+     geom_text(
+       data = filter(crimes, violent_crime > 1500 | property_crime > 6500),
+       check_overlap = TRUE, size = 2.5, nudge_y = 40
+     )
```



Problem 2: Movie scraping

The web site Box Office Mojo (http://www.boxofficemojo.com) gives statistics on box office earnings of movies. In addition to daily earnings, the web site also maintains lists of yearly and all time record holders.

We will start with a look at the movies in the top 100 of all time movie worldwide grosses in box office receipts. In particular, we will scrape the data from Box Office Mojo: All Time Box Office (https://www.boxofficemojo.com/chart/ww_top_lifetime_gross/?offset=0&area=XWW). The dollar amounts are in millions of dollars and the years marked with “^” indicate that the movie had multiple releases.

a.

Read in the data from page 1 using the `read_html` command, extract the html tables, then parse them into data frames. How many HTML tables are on the page? Which table contains the box office earnings?

```
> url <- "https://www.boxofficemojo.com/chart/ww_top_lifetime_gross/?offset=0&area=XWW"
>
> movie_data <- read_html(url) %>%
+   html_table() %>%
+   .[[1]] %>%
+   janitor::clean_names() %>%
+   tibble::as_tibble(.name_repair = "unique")
>
> length(movie_data)
[1] 8
```

Answer: There is one table on the page.

```
> url <- "https://www.boxofficemojo.com/chart/ww_top_lifetime_gross/?offset=0&area=XWW"
> movieHtml<- read_html(url)
> movieHtmlTables <- html_nodes(movieHtml, "table")
> movieHtmlTables
{xml_nodeset (1)}
[1] <table class="a-bordered a-horizontal-stripes a-size-base a-span12 mojo-b ...
> movieTables <- html_table(movieHtmlTables)
> glimpse(movieTables[[1]])
Rows: 200
Columns: 8
$ Rank                       <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, …
$ Title                      <chr> "Avatar", "Avengers: Endgame", "Avatar: The…
$ `Worldwide Lifetime Gross` <chr> "$2,923,706,026", "$2,799,439,100", "$2,320…
$ `Domestic Lifetime Gross`  <chr> "$785,221,649", "$858,373,000", "$684,075,7…
$ `Domestic %`               <chr> "26.9%", "30.7%", "29.5%", "29.8%", "45.2%"…
$ `Foreign Lifetime Gross`   <chr> "$2,138,484,377", "$1,941,066,100", "$1,636…
$ `Foreign %`                <chr> "73.1%", "69.3%", "70.5%", "70.2%", "54.8%"…
$ Year                       <int> 2009, 2019, 2022, 1997, 2015, 2018, 2021, 2…
```

b.

Extract the box office earnings data frame from the list found in (a). Clean up variable names by renaming columns to be: “rank”, “title”, “world_dollars”,“domestic_dollars”, “domestic_percentage”, “overseas_dollars”, “overseas_percentage”, “year”.

Answer:

Pulling just table 3 in the html list to get the data frame:

```
> moviesTable <- movieTables[[1]] %>% as_tibble()
> names(moviesTable) <- c("rank", "title", "world_dollars","domestic_dollars", "domestic_percentage", "overseas_d
ollars", "overseas_percentage", "year")
> moviesTable
```

rank	title	world_dollars	domestic_dollars
<int>	<chr>	<chr>	<chr>
1	Avatar	$2,923,706,026	$785,221,649
2	Avengers: Endgame	$2,799,439,100	$858,373,000
3	Avatar: The Way of Water	$2,320,250,281	$684,075,767
4	Titanic	$2,264,750,694	$674,292,608
5	Star Wars: Episode VII - The Force Awakens	$2,071,310,218	$936,662,225
6	Avengers: Infinity War	$2,052,415,039	$678,815,482
7	Spider-Man: No Way Home	$1,921,847,111	$814,115,070
8	Jurassic World	$1,671,537,444	$653,406,625
9	The Lion King	$1,663,079,059	$543,638,043
10	The Avengers	$1,520,538,536	$623,357,910

1-10 of 200 rows | 1-4 of 8 columns

Previous **1** 2 3 4 5 6 ... 20 Next

c.

Most columns with numeric type variables are actually character columns because of extra characters (dollar or percent signs, commas, other random characters). Clean up the columns with these issues and change their type to numeric rather than character.

Answer:

We can use the `readr` function `parse_number`.

```
> moviesTable <- moviesTable %>%
+   mutate_at(3:7, parse_number)
> glimpse(moviesTable)
Rows: 200
Columns: 8
$ rank                <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,…
$ title               <chr> "Avatar", "Avengers: Endgame", "Avatar: The Way of…
$ world_dollars       <dbl> 2923706026, 2799439100, 2320250281, 2264750694, 20…
$ domestic_dollars    <dbl> 785221649, 858373000, 684075767, 674292608, 936662…
$ domestic_percentage <dbl> 26.9, 30.7, 29.5, 29.8, 45.2, 33.1, 42.4, 39.1, 32…
$ overseas_dollars    <dbl> 2138484377, 1941066100, 1636174514, 1590458086, 11…
$ overseas_percentage <dbl> 73.1, 69.3, 70.5, 70.2, 54.8, 66.9, 57.6, 60.9, 67…
$ year                <int> 2009, 2019, 2022, 1997, 2015, 2018, 2021, 2015, 20…
```

d.

Using `html_nodes` to pull the anchor tags `a` from the html table of box office earnings that you found in part (a) (before turning it into a data frame table). Then use `html_attr` to get the url link (`href`) for the movie Titanic. (Note: the link is a page on `http://www.boxofficemojo.com`) Report the position number in the anchor or url vector that contains the Titanic URL and report the entire URL.

Answer:

```
> anchors <- html_nodes(movieHtmlTables[[1]], "a")
> head(anchors)
{xml_nodeset (6)}
[1] <a class="a-link-normal" href="/title/tt0499549/?ref_=bo_cso_table_1">Ava ...
[2] <a class="a-link-normal" href="/year/world/2009/?ref_=bo_cso_table_1">200 ...
[3] <a class="a-link-normal" href="/title/tt4154796/?ref_=bo_cso_table_2">Ave ...
[4] <a class="a-link-normal" href="/year/world/2019/?ref_=bo_cso_table_2">201 ...
[5] <a class="a-link-normal" href="/title/tt1630029/?ref_=bo_cso_table_3">Ava ...
[6] <a class="a-link-normal" href="/year/world/2022/?ref_=bo_cso_table_3">202 ...
> hrefs <- html_attr(anchors,"href")
> names <- html_text(anchors)
> index <- which(str_to_lower(names) == "titanic")
> index
[1] 7
> hrefs[index]
[1] "/title/tt0120338/?ref_=bo_cso_table_4"
> url_titanic <- str_c("http://www.boxofficemojo.com", hrefs[index])
> url_titanic
[1] "http://www.boxofficemojo.com/title/tt0120338/?ref_=bo_cso_table_4"
```

Row number 7 in the anchor vector contains the Titanic movie url: http://www.boxofficemojo.com/title/tt0120338/?ref_=bo_cso_table_4 (http://www.boxofficemojo.com/title/tt0120338/?ref_=bo_cso_table_4).

e.

The website contains 5 pages of the “top slightly more than 1000” grossing movies (about 200 per page). The basic format for their url links is shown in `tempUrl` where `#` is just a placeholder for starting movie rank where a `#` of 0 starts with top ranking of 1, a `#` of 200 starts with a top ranking of 201. (Fill in a 200 in the `#` spot and verify that the url works to get movies 201-400.)

```
> temp_url <- "https://www.boxofficemojo.com/chart/ww_top_lifetime_gross/?offset=#&area=XWW"
```

Write a function called that `pull_table` that returns a data frame constructed from one of these pages. This function should have input and outputs:

- input: a url (like `temp_url` but with a # plugged in)
- output: a parsed data frame with column names as defined in part (c) and with *all* columns parsed as numbers *except* for `title`
    - you can directly parse columns 3-7 into numbers

  - one of the pages will result in a character data type for `rank` (because of commas) but others will be integer. Create a condition in your function that checks for a character data type for `rank` and parses it to a number (with `parse_number`) if needed. (Note: `parse_integer` can't parse a number like "1,000" so we need to use `parse_number`)

Test your function on the first page with # of 800 in `temp_url`

Answer:

The function `pull_table` takes a url like `tempUrl` and ranking. It replaces the `#` in the temp url with the page number, reads the link, extracts the third table (which is always the box office stats) and returns it as a data frame.

```
> pull_table <- function(url)
+ {
+   movieHtml<- read_html(url)
+   movieHtmlTables <- html_nodes(movieHtml, "table")
+   moviesTable <- html_table(movieHtmlTables[[1]], header = TRUE) %>% as_tibble()
+   names(moviesTable) <- c("rank", "title", "world_dollars","domestic_dollars", "domestic_percentage", "overseas_dollars", "overseas_percentage", "year")
+   moviesTable <- moviesTable %>% mutate_at(3:7, parse_number)
+   if (is.character(moviesTable$rank)){
+     moviesTable$rank <- parse_number(moviesTable$rank)
+   }
+   return(moviesTable)
+ }
> pull_table("https://www.boxofficemojo.com/chart/ww_top_lifetime_gross/?offset=800&area=XWW")
```

rank <dbl>	title <chr>	world_dollars <dbl>	domestic_dollars <dbl>
801	The Prince of Egypt	218613188	101413188
802	Jack Reacher	218340595	80070736
803	Kingdom of Heaven	218237071	47398413
804	Smallfoot	218015531	83315531
805	The Emoji Movie	217776646	86089513
806	Smile	217408513	105935048
807	Too Cool to Kill	217254604	185882

rank	title	world_dollars	domestic_dollars
<dbl>	<chr>	<dbl>	<dbl>
808	Dracula Untold	217124280	56280355
809	Central Intelligence	216940871	127440871
810	Million Dollar Baby	216763646	100492203

1-10 of 200 rows | 1-4 of 8 columns Previous **1** 2 3 4 5 6 ... 20 Next

f.

Create a vector containing the URLs for the 5 pages of the “top slightly more than 1000” grossing movies, then use a `purrr` mapping function to create a data frame of all top movies. Do not use a `for` loop for this question.

- use a function from `stringr` to create the vector of URLs with the # replaced by the values 0, 200, 400, 600, and 800. (don’t type out all URLS by hand)
- make sure your data frame is a `tibble` and print out a `glimpse` of it

answer:

First, get a vector of URLs by replacing the # with a ranking:

```
> pages <- c(0, 200, 400, 600, 800)
> urls <- str_replace(temp_url, "#", as.character(pages))
> urls
[1] "https://www.boxofficemojo.com/chart/ww_top_lifetime_gross/?offset=0&area=XWW"
[2] "https://www.boxofficemojo.com/chart/ww_top_lifetime_gross/?offset=200&area=XWW"
[3] "https://www.boxofficemojo.com/chart/ww_top_lifetime_gross/?offset=400&area=XWW"
[4] "https://www.boxofficemojo.com/chart/ww_top_lifetime_gross/?offset=600&area=XWW"
[5] "https://www.boxofficemojo.com/chart/ww_top_lifetime_gross/?offset=800&area=XWW"
```

The we pass these pages to our `pull_table` function using `map_df` :

Since there are 5 pages to pull we use `map`

```
> all_df <- map_df(urls, pull_table)
> glimpse(all_df)
Rows: 1,000
Columns: 8
$ rank                <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,…
$ title               <chr> "Avatar", "Avengers: Endgame", "Avatar: The Way of…
$ world_dollars       <dbl> 2923706026, 2799439100, 2320250281, 2264750694, 20…
$ domestic_dollars    <dbl> 785221649, 858373000, 684075767, 674292608, 936662…
$ domestic_percentage <dbl> 26.9, 30.7, 29.5, 29.8, 45.2, 33.1, 42.4, 39.1, 32…
$ overseas_dollars    <dbl> 2138484377, 1941066100, 1636174514, 1590458086, 11…
$ overseas_percentage <dbl> 73.1, 69.3, 70.5, 70.2, 54.8, 66.9, 57.6, 60.9, 67…
$ year                <int> 2009, 2019, 2022, 1997, 2015, 2018, 2021, 2015, 20…
```

If we used `map`, we would need to use `bind_rows` to bind together each data frame by rows:

```
> all_list <- map(urls, pull_table)
> class(all_list)
[1] "list"
> all_df <- all_list %>% bind_rows()
> glimpse(all_df)
Rows: 1,000
Columns: 8
$ rank                <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,…
$ title               <chr> "Avatar", "Avengers: Endgame", "Avatar: The Way of…
$ world_dollars       <dbl> 2923706026, 2799439100, 2320250281, 2264750694, 20…
$ domestic_dollars    <dbl> 785221649, 858373000, 684075767, 674292608, 936662…
$ domestic_percentage <dbl> 26.9, 30.7, 29.5, 29.8, 45.2, 33.1, 42.4, 39.1, 32…
$ overseas_dollars    <dbl> 2138484377, 1941066100, 1636174514, 1590458086, 11…
$ overseas_percentage <dbl> 73.1, 69.3, 70.5, 70.2, 54.8, 66.9, 57.6, 60.9, 67…
$ year                <int> 2009, 2019, 2022, 1997, 2015, 2018, 2021, 2015, 20…
```

Note on warnings: the list shows some of the `problems` associated with tables 2 and 5 parsing. This is due to entries of “-” or “–” when a number was expected. These will be converted to NAs (which is the correct action to take).

Problem 3: Penguins

Let’s revisit the Palmer penguins data. The following scatterplot compares bill length to body mass using a Shiny app structure. This isn’t an interactive graph yet, and below you will be modifying this basic app structure to create interactive versions of this graph.

```
> library(shiny)
> library(tidyverse)
> data(penguins, package = 'palmerpenguins')
>
> ui <- fluidPage(
+   plotOutput("plot", height = 500)
+ )
>
> server <- function(input, output){
+   output$plot <- renderPlot({
+     g <- ggplot(penguins, aes(x = bill_length_mm, y = body_mass_g))
+     g + geom_point()
+   })
+ }
>
> # you can modify the height to avoid scrolling
> shinyApp(ui, server, options = list(height = 600))
```

Shiny applications not supported in static R Markdown documents

a.

Copy the code above to your homework answers .Rmd, then modify the app to include a checkbox input that allows you to toggle between the graph with and without points colored by `species`. The main steps of this should be:

- Add the `checkboxInput` input object in the `ui` that will allow you to add (or omit) color. Run this app to make sure your input object works.
- Then using the input value from the checkbox to modify the graph rendered, to include color when checked and exclude color when unchecked.

answer:

```
> ui <- fluidPage(
+     checkboxInput(inputId = "color_check",
+                   label = "Check to color by species",
+                   value = FALSE),  # default is not checked
+     plotOutput("plot", height = 500)
+ )
>
> server <- function(input, output){
+   output$plot <- renderPlot({
+     g <- ggplot(penguins, aes(x = bill_length_mm, y = body_mass_g))
+     if (input$color_check){    # checked: add color
+       g + geom_point(aes(color = species))
+     } else{   # not checked: no color
+     g + geom_point()
+     }
+   })
+ }
>
> # you can modify the height to avoid scrolling
> shinyApp(ui, server, options = list(height = 600))
```

Shiny applications not supported in static R Markdown documents

b.

Copy your part (a) code here, then modify it to use `varSelectInput` to select x- and y-axis variables from columns 3-6 in `penguins` . Recall the Intro to Shiny example, you will need to use the `!!` (“bang-bang”) to substitute the server input value with a variable from the data frame used in a ggplot.

answer:

```
> ui <- fluidPage(
+     checkboxInput(inputId = "color_check",
+                   label = "Check to color by species",
+                   value = FALSE),  # default is not checked
+     varSelectInput(inputId = "x",
+                    label = "select your x variable",
+                    data = select(penguins, 3:6),
+                    selected = "bill_length_mm"),
+     varSelectInput(inputId = "y",
+                    label = "select your y variable",
+                    data = select(penguins, 3:6),
+                    selected = "body_mass_g"),
+   plotOutput("plot", height = 400)
+ )
>
> server <- function(input, output){
+   output$plot <- renderPlot({
+     g <- ggplot(penguins, aes(x = !!input$x, y = !!input$y))
+     if (input$color_check){    # checked: add color
+       g + geom_point(aes(color = species))
+     } else{   # not checked: no color
+     g + geom_point()
+     }
+   })
+ }
>
> # you can modify the height to avoid scrolling
> shinyApp(ui, server, options = list(height = 700))
```

Shiny applications not supported in static R Markdown documents

c.

Again, copy your app code from part (b) to this question. The goal of this part is to allow the app user to click on a point to see the data associated with the case (or cases) near the clicked region.

- Check the help file for `plotOutput` and look at the `click` argument option.
    - This option allows a user to *interact* with a rendered object (plot).
    - If we let `click = "my_click"` , then `input$my_click` will be an **input** value return returned by the click action that contains the co ordinations (x/y) of the clicked point.
    - The function `nearPoints(my_data, input$my_click)` will return a data frame of data cases “near” the input coordinates.
- Use the `dataTable` render and output commands to add a data table of data points near your click below your scatterplot.

answer:

```
> ui <- fluidPage(
+     checkboxInput(inputId = "color_check",
+                   label = "Check to color by species",
+                   value = FALSE),  # default is not checked
+     varSelectInput(inputId = "x",
+                    label = "select your x variable",
+                    data = select(penguins, 3:6),
+                    selected = "bill_length_mm"),
+      varSelectInput(inputId = "y",
+                    label = "select your y variable",
+                    data = select(penguins, 3:6),
+                    selected = "body_mass_g"),
+   plotOutput("plot", click = "plot_click", height = 400),
+   dataTableOutput("table")
+ )
>
> server <- function(input, output){
+   output$plot <- renderPlot({
+     g <- ggplot(penguins, aes(x = !!input$x, y = !!input$y))
+     if (input$color_check){    # checked: add color
+       g + geom_point(aes(color = species))
+     } else{   # not checked: no color
+     g + geom_point()
+     }
+   })
+   output$table <- renderDataTable({
+     nearPoints(penguins, input$plot_click)
+   })
+ }
>
> # you can modify the height to avoid scrolling
> shinyApp(ui, server, options = list(height = 1000))
```

Shiny applications not supported in static R Markdown documents

d.

Again, copy your app code from part (c) to this question. The goal of this part is to allow the app user to **brush** a region to see the data associated with the case (or cases) inside the brushed region.

To do this, modify your part (c) code to change from a “click” to a “brush”. Use the `brushedPoints` function instead of `nearPoints` .

answer:

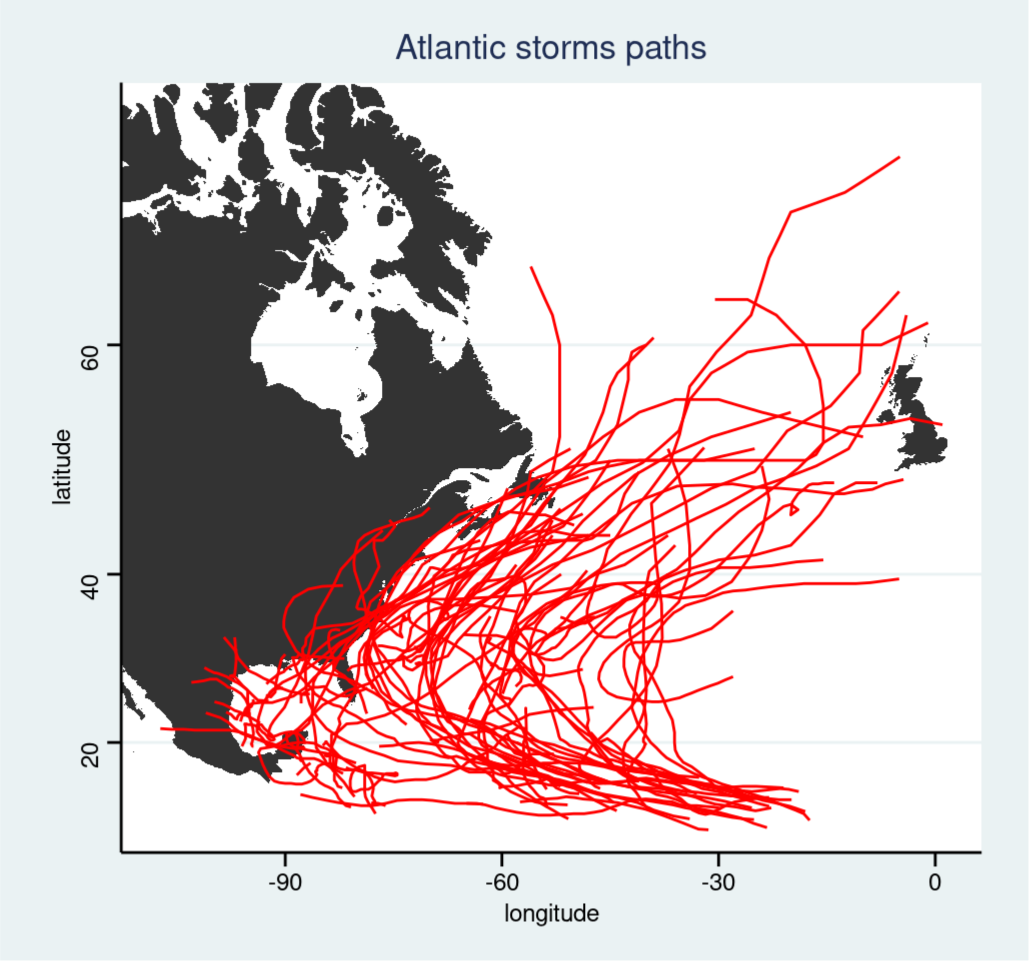
```
> ui <- fluidPage(
+     checkboxInput(inputId = "color_check",
+                   label = "Check to color by species",
+                   value = FALSE),  # default is not checked
+     varSelectInput(inputId = "x",
+                    label = "select your x variable",
+                    data = select(penguins, 3:6),
+                    selected = "bill_length_mm"),
+      varSelectInput(inputId = "y",
+                    label = "select your y variable",
+                    data = select(penguins, 3:6),
+                    selected = "body_mass_g"),
+   plotOutput("plot", brush = "plot_brush", height = 400),
+   dataTableOutput("table")
+ )
>
> server <- function(input, output){
+   output$plot <- renderPlot({
+     g <- ggplot(penguins, aes(x = !!input$x, y = !!input$y))
+     if (input$color_check){    # checked: add color
+       g + geom_point(aes(color = species))
+     } else{   # not checked: no color
+     g + geom_point()
+     }
+   })
+   output$table <- renderDataTable({
+     brushedPoints(penguins, input$plot_brush)
+   },
+   options = list(pageLength = 5))  # controls table length
+ }
>
> # you can modify the height to avoid scrolling
> shinyApp(ui, server, options = list(height = 1000))
```

Shiny applications not supported in static R Markdown documents

Probelm 4: Storm paths

Revisit the storm path problem from homework 2. Here is the basic map without a year facet and color.

```
> data(storms, package = "nasaweather")
>
> ctry <- map_data("world",
+                   region = c(
+                     "usa",
+                     "mexico",
+                     "canada",
+                     "uk"
+                   ))
>
> base_map <- ggplot(ctry) +
+   geom_polygon(aes(x = long,  y = lat, group = group)) +
+   labs(
+     x = "longitude",
+     y = "latitude",
+     title = "Atlantic storms paths"
+     )
>
> base_map +
+   geom_path(data = storms, aes(x = long, y = lat, group = name), color = "red") +
+   coord_map(xlim  = c(min(storms$long), max(storms$long)),
+             ylim  = c(min(storms$lat), max(storms$lat)))
```



a.

Select one storm and filter your data to that storm and add in an elapsed time (hour) variable that measures the number of hours that has elapsed since the first lat/lon measurement. The trace the path of the storm with an animated `sliderInput` .

answer:

```
> base_map <- ggplot(ctry) +
+   geom_polygon(aes(x = long,  y = lat, group = group)) +
+   labs(
+     x = "longitude",
+     y = "latitude",
+     title = "Atlantic storms paths"
+     )
>
> storms_alison <- storms %>%
+   filter(name == "Allison") %>%
+   arrange(year, month, day, hour) %>%
+   mutate(hour_btw = 6,
+          elapsed_hour = cumsum(hour_btw) - 6)
>
>
> ui <- fluidPage(
+   sliderInput(inputId = "slider",
+               label = "plot a storm path",
+               min = 0,
+               max = max(storms_alison$elapsed_hour),
+               value = 0,
+               step = 6,
+               animate = TRUE),
+   plotOutput("map")
+ )
>
> server <- function(input, output){
+   output$map <- renderPlot({
+     base_map +
+     geom_path(data = filter(storms_alison, elapsed_hour <= input$slider),
+             aes(x = long, y = lat), color = "red") +
+     coord_map(xlim  = c(min(storms_alison$long), max(storms_alison$long)),
+             ylim  = c(min(storms_alison$lat), max(storms_alison$lat)))
+   })
+ }
>
> # you can modify the height to avoid scrolling
> shinyApp(ui, server, options = list(height = 600))
```

Shiny applications not supported in static R Markdown documents

b.

Use `selectInput` to draw a storm path map that lets the user select the storm of interest. You don’t need to animate this path. Note that the input value returned by this input option is a character string. (Use this info to filter the data based on the user selected storm.)

answer:

```
> base_map <- ggplot(ctry) +
+   geom_polygon(aes(x = long,  y = lat, group = group)) +
+   labs(
+     x = "longitude",
+     y = "latitude",
+     title = "Atlantic storms paths"
+     )
> storm_names <- storms %>%
+   select(name) %>%
+   distinct() %>%
+   arrange(name)
>
> ui <- fluidPage(
+      selectInput(inputId = "name",
+                    label = "select your storm",
+                    choices = storm_names),
+   plotOutput("map")
+ )
>
> server <- function(input, output){
+   output$map <- renderPlot({
+     base_map +
+     geom_path(data = filter(storms, name == input$name),
+             aes(x = long, y = lat), color = "red") +
+   coord_map(xlim  = c(min(storms$long), max(storms$long)),
+             ylim  = c(min(storms$lat), max(storms$lat)))
+   })
+ }
>
> # you can modify the height to avoid scrolling
> shinyApp(ui, server, options = list(height = 600))
```

Shiny applications not supported in static R Markdown documents

c.

Combine (a) and (b) into one graph! You may need to use `renderUI` and `uiOutput` to allow your slider max value to vary by storm selected. And you may need to create a reactive version of the data using `reactive` to complete this task.

answer:

```
> storms <- storms %>%
+   group_by(name, year) %>%
+   arrange(year, month, day, hour) %>%
+   mutate(hour_btw = 6,
+          elapsed_hour = cumsum(hour_btw) - 6) %>%
+   ungroup()
>
> ui <- fluidPage(
+      selectInput(inputId = "name",
+                    label = "select your storm",
+                    choices = storm_names),
+   uiOutput("slider"),
+   plotOutput("map")
+ )
>
> server <- function(input, output){
+   my_storm <- reactive({
+     filter(storms, name == input$name)
+   })
+
+   output$map <- renderPlot({
+     req(input$animation)
+
+     data_upto <- my_storm() %>%
+       filter(elapsed_hour <= input$animation)
+
+     base_map +
+         geom_path(data = data_upto,
+             aes(x = long, y = lat), color = "red") +
+         coord_map(xlim  = c(min(storms$long), max(storms$long)),
+             ylim  = c(min(storms$lat), max(storms$lat)))
+   })
+
+   output$slider <- renderUI({
+   sliderInput(inputId = "animation",
+               label = "plot a storm path",
+               min = 0,
+               max = max(my_storm()$elapsed_hour),
+               value = 0,
+               step = 6,
+               animate = animationOptions(interval=300, loop = FALSE)) })
+ }
>
> # you can modify the height to avoid scrolling
> shinyApp(ui, server, options = list(height = 600))
```

Shiny applications not supported in static R Markdown documents