STA 032 Homework 2

CHANGE YOUR NAME HERE

DUE: April 26 2023, 12PM

Instructions

- Upload a PDF file, named with your UC Davis email ID and homework number (e.g., `xjw18_hw2.pdf`), to Gradescope (accessible through Canvas). You will give the commands to answer each question in its own code block, which will also produce output that will be automatically embedded in the output file. When asked, answer must be supported by written statements as well as any code used.
- All code used to produce your results must be shown in your PDF file (e.g., do not use `echo = FALSE` or `include = FALSE` as options anywhere). `Rmd` files do not need to be submitted, but may be requested by the TA and must be available when the assignment is submitted.
- Students may choose to collaborate with each other on the homework, but must clearly indicate with whom they collaborated. Every student must upload their own submission.
- Start to work on it as early as possible
- When you want to show your result as a vector that is too long, slice the first 10 objects. When you want to show your result as a data frame, use `head()` on it. Failure to do so may lead to point deduction.
- Directly knit the Rmd file will give you an html file. Open that file in your browser and then you can print it into a PDF file.

Level 1 Parts

Problem 1: Manipulating data frames

1. Load the **dplyr** package and the murders dataset. (In future courses before your project you need to write these code yourself to include package and dataset)

```
library(dplyr)
library(dslabs)
data(murders)
```

You can add columns using the **dplyr** function `mutate`. This function is aware of the column names and inside the function you can call them unquoted:

```
murders <- mutate(murders, population_in_millions = population / 10^6)
```

We can write `population` rather than `murders$population`. The function `mutate` knows we are grabbing columns from `murders`.

Use the function `mutate` to add a murders column named `rate` with the per 100,000 murder rate as in the example code above. Make sure you redefine `murders` as done in the example code above (murders <- [your code]) so we can keep using this variable.

2. If `rank(x)` gives you the ranks of `x` from lowest to highest, `rank(-x)` gives you the ranks from highest to lowest. Use the function `mutate` to add a column `rank` containing the rank, from highest to lowest murder rate. Make sure you redefine `murders` so we can keep using this variable.

3. With **dplyr**, we can use `select` to show only certain columns. For example, with this code we would only show the states and population sizes:

Instructions

Level 1 Parts

Level 2 parts

```
select(murders, state, population) |> head()
```

Use `select` to show the state names and abbreviations in `murders`. Do not redefine `murders`, just show the results.

4. The **dplyr** function `filter` is used to choose specific rows of the data frame to keep. Unlike `select` which is for columns, `filter` is for rows. For example, you can show just the New York row like this:

```
filter(murders, state == "New York")
```

```
     state abb    region population total population_in_millions
1 New York  NY Northeast   19378102   517                19.3781
```

You can use other logical vectors to filter rows.

Use `filter` to show the top 5 states with the highest murder rates. After we add murder rate and rank, do not change the murders dataset, just show the result. Remember that you can filter based on the `rank` column.

5. We can remove rows using the `!=` operator. For example, to remove Florida, we would do this:

```
no_florida <- filter(murders, state != "Florida")
```

Create a new data frame called `no_south` that removes states from the South region. How many states are in this category? You can use the function `nrow` for this.

6. We can also use `%in%` to filter with **dplyr**. You can therefore see the data from New York and Texas like this:

```
filter(murders, state %in% c("New York", "Texas"))
```

```
     state abb    region population total population_in_millions
1 New York  NY Northeast   19378102   517               19.37810
2    Texas  TX     South   25145561   805               25.14556
```

Create a new data frame called `murders_nw` with only the states from the Northeast and the West. How many states are in this category?

7. Suppose you want to live in the Northeast or West **and** want the murder rate to be less than 10. We want to see the data for the states satisfying these options. Note that you can use logical operators with `filter`. Here is an example in which we filter to keep only small states in the Northeast region.

```
filter(murders, population < 5000000 & region == "Northeast")
```

```
          state abb    region population total population_in_millions
1   Connecticut  CT Northeast    3574097    97               3.574097
2         Maine  ME Northeast    1328361    11               1.328361
3 New Hampshire  NH Northeast    1316470     5               1.316470
4  Rhode Island  RI Northeast    1052567    16               1.052567
5       Vermont  VT Northeast     625741     2               0.625741
```

Make sure `murders` has been defined with `rate` and `rank` and still has all states. Create a table called `my_states` that contains rows for states satisfying both the conditions: it is in the Northeast or West and the murder rate is less than 10. Use `select` to show only the state name, the rate, and the rank.

Problem 2: Pipes

Reset `murders` to the original table by using `data(murders)`. Use a pipe to create a new data frame called `my_states` that considers only states in the Northeast or West which have a murder rate lower than 1, and contains only the state, rate and rank columns. The pipe should also have four components separated by three `|>`. The code should look something like this:

```
my_states <- murders |>
  mutate SOMETHING |>
  filter SOMETHING |>
  select SOMETHING
```

Problem 3: filter

We will using data from the nycflights13 package. So please install that package:

```
install.packages("nycflights13")
```

Now we can library the package and access the data `flights`

```
library(nycflights13)
library(tidyverse)
head(flights)
```

```
# A tibble: 6 × 19
   year month   day dep_time sched_dep…¹ dep_d…² arr_t…³ sched…⁴ arr_d…⁵ carrier
  <int> <int> <int>    <int>       <int>   <dbl>   <int>   <int>   <dbl> <chr>
1  2013     1     1      517         515       2     830     819      11 UA
2  2013     1     1      533         529       4     850     830      20 UA
3  2013     1     1      542         540       2     923     850      33 AA
4  2013     1     1      544         545      -1    1004    1022     -18 B6
5  2013     1     1      554         600      -6     812     837     -25 DL
6  2013     1     1      554         558      -4     740     728      12 UA
# … with 9 more variables: flight <int>, tailnum <chr>, origin <chr>,
#   dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
#   time_hour <dttm>, and abbreviated variable names ¹sched_dep_time,
#   ²dep_delay, ³arr_time, ⁴sched_arr_time, ⁵arr_delay
```

You might notice that this data frame prints a little differently from other data frames you might have used in the past: it only shows the first few rows and all the columns that fit on one screen. (To see the whole dataset, you can run `View(flights)` which will open the dataset in the RStudio viewer). It prints differently because it's a **tibble**. Tibbles are data frames, but slightly tweaked to work better in the tidyverse. For now, you don't need to worry about the differences.

You might also have noticed the row of three (or four) letter abbreviations under the column names. These describe the type of each variable:

- int stands for integers.
- dbl stands for doubles, or real numbers.
- chr stands for character vectors, or strings.
- dttm stands for date-times (a date + a time).
- lgl stands for logical, vectors that contain only TRUE or FALSE.
- fctr stands for factors, which R uses to represent categorical variables with fixed possible values.

- date stands for dates.

Problems starts here

Find all flights that

1. Had an arrival delay of two or more hours
2. Flew to Houston (IAH or HOU)
3. Were operated by United, American, or Delta
4. Departed in summer (July, August, and September)
5. Arrived more than two hours late, but didn't leave late
6. Were delayed by at least an hour, but made up over 30 minutes in flight

In each of the problem, only show the `dim(result)` where result is your result for each sub problem.

Problem 4: Data visualization

Start by loading the **dplyr** and **ggplot2** library as well as the `murders` and `heights` data.

```
library(dplyr)
library(ggplot2)
library(dslabs)
data(heights)
data(murders)
```

1. With **ggplot2** plots can be saved as objects. For example we can associate a dataset with a plot object like this

```
p <- ggplot(data = murders)
```

Because `data` is the first argument we don't need to spell it out

```
p <- ggplot(murders)
```

and we can also use the pipe:

```
p <- murders |> ggplot()
```

What is class of the object `p` ?

2. Remember that to print an object you can use the command `print` or simply type the object. Print the object `p` defined in exercise one and describe what you see.

a. Nothing happens.
b. A blank slate plot.
c. A scatterplot.
d. A histogram.

3. Using the pipe `|>`, create an object `p` but this time associated with the `heights` dataset instead of the `murders` dataset.

4. Now we are going to add a layer and the corresponding aesthetic mappings. For the murders data we plotted total murders versus population sizes. Explore the `murders` data frame to remind yourself what are the names for these two variables and select the correct answer. **Hint**: Look at `?murders` .

a. `state` and `abb` .

Instructions

Level 1 Parts

Level 2 parts

b. `total_murders` and `population_size`.
c. `total` and `population`.
d. `murders` and `size`.

5. To create the scatterplot we add a layer with `geom_point`. The aesthetic mappings require us to define the x-axis and y-axis variables, respectively. So the code looks like this:

```
murders |> ggplot(aes(x = , y = )) +
  geom_point()
```

except we have to define the two variables `x` and `y`. Fill this out with the correct variable names.

6. Note that if we don't use argument names, we can obtain the same plot by making sure we enter the variable names in the right order like this:

```
murders |> ggplot(aes(population, total)) +
  geom_point()
```

Remake the plot but now with total in the x-axis and population in the y-axis.

7. If instead of points we want to add text, we can use the `geom_text()` or `geom_label()` geometries. The following code

```
murders |> ggplot(aes(population, total)) + geom_label()
```

will give us the error message: `Error: geom_label requires the following missing aesthetics: label`

Why is this?

a. We need to map a character to each point through the label argument in aes.
b. We need to let `geom_label` know what character to use in the plot.
c. The `geom_label` geometry does not require x-axis and y-axis values.
d. `geom_label` is not a ggplot2 command.

8. Rewrite the code above to use abbreviation as the label through `aes`

9. Change the color of the labels to blue. How will we do this?

a. Adding a column called `blue` to `murders`.
b. Because each label needs a different color we map the colors through `aes`.
c. Use the `color` argument in `ggplot`.
d. Because we want all colors to be blue, we do not need to map colors, just use the color argument in `geom_label`.

10. Rewrite the code above to make the labels blue.

11. Now suppose we want to use color to represent the different regions. In this case which of the following is most appropriate:

a. Adding a column called `color` to `murders` with the color we want to use.
b. Because each label needs a different color we map the colors through the color argument of `aes`.
c. Use the `color` argument in `ggplot`.
d. Because we want all colors to be blue, we do not need to map colors, just use the color argument in `geom_label`.

12. Rewrite the code above to make the labels' color be determined by the state's region.

13. Now we are going to change the x-axis to a log scale to account for the fact the distribution of population is skewed. Let's start by defining an object `p` holding the plot we have made up to now

```
p <- murders |>
  ggplot(aes(population, total, label = abb, color = region)) +
  geom_label()
```

To change the y-axis to a log scale we learned about the `scale_x_log10()` function. Add this layer to the object `p` to change the scale and render the plot.

14. Repeat the previous exercise but now change both axes to be in the log scale.

15. Now edit the code above to add the title “Gun murder data” to the plot. Hint: use the `ggtitle` function.

Level 2 parts

Problem 5 NHANES data

For these exercises, we will be using the data from the survey collected by the United States National Center for Health Statistics (NCHS). This center has conducted a series of health and nutrition surveys since the 1960’s. Starting in 1999, about 5,000 individuals of all ages have been interviewed every year and they complete the health examination component of the survey. Part of the data is made available via the **NHANES** package. Once you install the **NHANES** package, you can load the data like this:

```
library(NHANES)
data(NHANES)
```

The **NHANES** data has many missing values. The `mean` and `sd` functions in R will return `NA` if any of the entries of the input vector is an `NA`. Here is an example:

```
library(dslabs)
data(na_example)
mean(na_example)
```

```
[1] NA
```

```
sd(na_example)
```

```
[1] NA
```

To ignore the `NA`s we can use the `na.rm` argument:

```
mean(na_example, na.rm = TRUE)
```

```
[1] 2.301754
```

```
sd(na_example, na.rm = TRUE)
```

```
[1] 1.22338
```

Let’s now explore the NHANES data.

1. We will provide some basic facts about blood pressure. First let’s select a group to set the standard. We will use 20-to-29-year-old females. `AgeDecade` is a categorical variable with these ages. Note that the category is coded like ” 20-29”, with **a space in front**! What is the average and standard deviation of systolic blood pressure as saved in the `BPSysAve` variable? Save it to a variable called `ref`.

Hint: Use `filter` and `summarize` and use the `na.rm = TRUE` argument when computing the average and standard deviation. You can also filter the NA values using `filter`.

Instructions

Level 1 Parts

Level 2 parts

2. Using a pipe, assign the average to a numeric variable `ref_avg`. Hint: Use the code similar to above and then `pull`.

3. Now report the min and max values for the same group.

4. Compute the average and standard deviation for females, but for each age group separately rather than a selected decade as in question 1. Note that the age groups are defined by `AgeDecade`. Hint: rather than filtering by age and gender, filter by `Gender` and then use `group_by`.

5. We can actually combine both gender summaries into one line of code. This is because `group_by` permits us to group by more than one variable. Obtain one big summary table using `group_by(AgeDecade, Gender)`.

6. For males between the ages of 40-49, summarise the average systolic blood pressure across race as reported in the `Race1` variable. Order the resulting table from lowest to highest average systolic blood pressure. hint: `arrange()`

Problem 6: Data visualization 2, GDP and life expectancy

We'll be illustrating visualization process using a dataset comprising observations of life expectancies at birth for men, women, and the general population, along with GDP per capita and total population for 158 countries at approximately five-year intervals from 2000 to 2019.

- Observational units: countries.
- Variables: country, year, life expectancy at birth (men, women, overall), GDP per capita, total population, region (continent), and subregion.

Here we will load the data for you. In this problem you just need to work on the dataset named `data`.

```
myfile <- "https://raw.githubusercontent.com/xjw1001001/xjw1001001.github.io/main/data/homework%20data/HW2data.csv"
data = read.csv(myfile) %>%
  select(-X) %>%
  mutate(Year = as.factor(Year))
head(data)
```

```
  Country.Name Year Life.Expectancy Male.Life.Expectancy Female.Life.Expectancy
1  Afghanistan 2019            63.2                 63.3                   63.2
2  Afghanistan 2015            61.7                 61.0                   62.3
3  Afghanistan 2010            59.9                 59.6                   60.3
4      Albania 2019            78.0                 76.3                   79.9
5      Albania 2015            77.8                 76.1                   79.7
6      Albania 2010            76.2                 74.2                   78.3
  GDP.per.capita region      sub.region Population
1       507.1034   Asia   Southern Asia   38041754
2       578.4664   Asia   Southern Asia   34413603
3       543.3030   Asia   Southern Asia   29185507
4      5353.2449 Europe Southern Europe    2854191
5      3952.8012 Europe Southern Europe    2880703
6      4094.3503 Europe Southern Europe    2913021
```

1. Construct a scatterplot of `Life.Expectancy` against `GDP.per.capita`; each point corresponds to one country in one year. Change the X axis into 'GDP per capita' and Y axis into 'Life Expectancy at Birth'. Change the title into 'Scatterplot of life expectancy at birth against GDP per capita'. Change the scale of X axis into log2 scale.

Name the plot object as `p`. Show the plot by call `p` like this:

```
p = ggplot... +
    geom_point... +
    ... +
    ...

# Show the plot
p
```

- Hint: `trans='log2'` inside some function's (). What's that function?

2. Map Year (as nominal) to color in part 1 plot. Change the opacity of all points into 0.8. (`alpha` =0.8 in `geom_point()`) Does this color give you any information?

- I have already transform the Year into factor by `mutate(Year = as.factor(Year))` . Without that Year will be a numeric variable and the color will be assigned continuous.
- Start from the part 1 code. Just make a small modification in `aes` part will be able to do that.

3. Instead, map region to color, population to size in the `aes()` . Change the opacity of all points into 0.8. Does this plot give you more information?

4. Now based on the part 3 plot, facet by the year. Use `facet_wrap` .

5. Apply a new theme from the ggthemes package.

```
library(ggthemes)
```

6. Use ggrepel to label the countries in the country list:

```
library(ggrepel)
country_list = c("Afghanistan","Belgium","Canada","Djibouti","France","Indonesia","Mexico","Sudan")
```

Here is the code that you can generate the label for these selected countries. Generate the plot. If you find the result plot texts are colored, it may because you put color and size as global aes mapping. You need to change it into local ones in `geom_point()` .

- You can try different values of the arguments for `box.padding` , `max.overlaps` , `force` to make it nicer.

```
geom_text_repel(data =
       subset(data, Country.Name %in% country_list),
       aes(label = Country.Name),
       box.padding = 0.5, max.overlaps = Inf,
       force = 1.5)
```

Include the person you coop with:

Names:

Appendix

```
sessionInfo()
```

```
R version 4.2.3 (2023-03-15 ucrt)
Platform: x86_64-w64-mingw32/x64 (64-bit)
Running under: Windows 10 x64 (build 22621)

Matrix products: default

locale:
[1] LC_COLLATE=English_United States.utf8
[2] LC_CTYPE=English_United States.utf8
[3] LC_MONETARY=English_United States.utf8
[4] LC_NUMERIC=C
[5] LC_TIME=English_United States.utf8

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

other attached packages:
 [1] ggrepel_0.9.3      ggthemes_4.2.4     NHANES_2.1.0       lubridate_1.9.2
 [5] forcats_1.0.0      stringr_1.5.0      purrr_1.0.1        readr_2.1.4
 [9] tidyr_1.3.0        tibble_3.2.0       ggplot2_3.4.1      tidyverse_2.0.0
[13] nycflights13_1.0.2 dslabs_0.7.4       dplyr_1.1.0

loaded via a namespace (and not attached):
 [1] Rcpp_1.0.10      pillar_1.8.1     bslib_0.4.2      compiler_4.2.3
 [5] jquerylib_0.1.4  tools_4.2.3      digest_0.6.31    timechange_0.2.0
 [9] jsonlite_1.8.4   evaluate_0.20    lifecycle_1.0.3  gtable_0.3.2
[13] pkgconfig_2.0.3  rlang_1.1.0      cli_3.6.0        rstudioapi_0.14
[17] yaml_2.3.7       xfun_0.37        fastmap_1.1.1    withr_2.5.0
[21] knitr_1.42       hms_1.1.3        generics_0.1.3   vctrs_0.6.0
[25] sass_0.4.5       grid_4.2.3       tidyselect_1.2.0 glue_1.6.2
[29] R6_2.5.1         fansi_1.0.4      rmarkdown_2.20   tzdb_0.3.0
[33] magrittr_2.0.3   scales_1.2.1     htmltools_0.5.4  colorspace_2.1-0
[37] utf8_1.2.3       stringi_1.7.12   munsell_0.5.0    cachem_1.0.7
```