# Methods Camp Assignment 3 Answers

**Day 3**

2025 Methods Camp Instructors

For today's assignment, we will be practicing using functions and alternatives to for loops to perform data manipulation tasks you're likely to encounter in your own work. We will also be reviewing probability density functions (PDFs) and how to manually create them in R.

## Structuring data in a way useful for plotting

We're going to be creating a function that creates and arrays plots of an *individual state's* trends in the counts of parent reports of vaccine events over time.

1. To make the next steps easier, restrict the data to exclude observations that are missing the year ("Unknown Date")
2. Before moving to the function, we're going to get the data in a format that is easier to feed the function. Create a new data.frame, *stateyearcounts*, that indicates the number of cases per year for each state. Practice doing this using the group_by / summarize workflow.

## Using functions to plot

3. Load the data: `stateyearreports.csv`. Is this data tidy? Why or why not? If not, run the following code to pivot your data so that it is in "long format".

What is "names_to"? What is "values_to"?

```
# Load data
# wide <- read_csv("stateyearreports.csv")

# Pivot your data to be "long"
# stateyear <- wide %>% pivot_longer(-c("id", "year"), # ignore id columns
```

```
                                    #names_to = "state",
                                    #values_to = "reports" )

# head(stateyear)
```

4. We're going to plot in two steps: 1. Creating two plots outside a function to get the code correct. 2. Generalizing to a function that will plot the counts by year for any group of states you choose

First, use ggplot to create separate plots for the counts of autism-related vaccine reports by year for two states– New Jersey and New York–side by side. You can either do a bar or line graph. Make sure the title indicates which state it is.

(Bonus - optional) Make sure the two plots have the same y axis range for comparability purposes (0 to the maximum reports out of the two).

5. Now generalize into a function that can do the following:

- Take in a state name
- For that state, create and store a plot of that state's autism reports per year
- (Bonus - challenging) Set the maximum ylim of the plot to the maximum number of reports reported across several states (in a `statesofinterest` vector)

*Hint*: Look at the above code for the two states. What did you change when copying and pasting? How can you subset the vector to give you the name of the state for the title and plot?

## Using alternatives to for loops

6. Using `map`, run the function with different groups of states that you're interested in comparing (e.g., you could create a character vector called `statesofinterest` with the state where you grew up, the state where you went to college, and the state you're in now, and plot graphs for each).

Try with at least two different `statesofinterest` vectors and store the results.

Your output should be a list containing all the stored plots.

7. Now, let's modify the function to return the plot and the average number of reports for that state. Your function should return a list returning these elements as `state_plot` and `state_avg_reports`. Copy and paste the function below, and modify it accordingly.
8. Finally, let's use `do` to run the function for the following states: Illinois, California, New Jersey.

Inside the loop, you should call the function and retrieve the average number of reports for each state. Store the results in a vector called `state_averages` (hint, use `.combine = c` argument).

**Bonus**: Modify your code to run the loop in parallel using `dopar`. Make sure to load the necessary libraries and register a parallel backend before running the loop.

## Reviewing probability density functions (PDFs)

9. Now, we will create a probability density function (PDF) from our data. We will define our random variable $X$ as the number of reports for a given state in a given year (so, the `reports` column in our data frame).

Since $X$ is a discrete random variable, our PDF ($f(x)$) tells us the probability of $X$ taking on a specific value $x$:

$$f(x) = P(X = x)$$

To create a PDF, we need to calculate the frequency of each unique value of `reports` in our data frame. Use `summarize()` to create a new data frame that contains these frequency counts:

10. Now, write a function `pdf` that takes in a a value `x` and returns `f(x)`, i.e. the probability of `x` reports. Verify your function outputs 0.207 for `x=1`.
11. What happens when you run `pdf(500)` or `pdf(18)`? Why? Modify your function using conditional statements to return 0 if `x` is not a report value in your data:
12. Now, instead of just returning the probability value, modify your function to return a data frame with one row and two columns: `x` and `f_x`. The `x` column should contain the input value, and the `f_x` column should contain the probability of that value.
13. Lastly, use `map_dfr` to create a data frame that contains the probabilities for all values from 0 to the maximum number of reports in your data frame. Use `head()` to print the first 8 rows of the resulting data frame.