Create a Mock Dataframe and Plotting Graphs

POL-501

2024-09-16

Table of Contents

## 0.1 Introduction

This tutorial demonstrates how to create a mock public opinion dataset using dplyr and generate visualizations using ggplot2. We’ll cover the steps involved in creating a dataset with different opinion scales, age groups, and gender categories, and we’ll create visualizations such as histograms, boxplots, and scatterplots. You’ll also learn how to adjust the functions and variables for future uses.

## 0.2 Setup

First, we’ll load the necessary packages: dplyr for data manipulation and ggplot2 for visualization.

## 0.3 Creating the Mock Data

The dataset will include the following columns: - **opinion\_100**: Opinion measured on a 100-point scale (feeling thermometer). - **opinion\_7**: Opinion measured on a 7-point scale. - **age\_group**: Age groups divided into four categories: ‘18-29’, ‘30-44’, ‘45-64’, and ‘65+’. - **gender**: Gender categories with three groups: ‘Male’, ‘Female’, and ‘Non-binary’.

### 0.3.1 Creating the Dataset

We use the tibble() function from dplyr to create a dataset. The runif() function generates random values for opinion\_100, and sample() is used to randomly assign values for opinion\_7, age\_group, and gender.

# Set seed for reproducibility  
set.seed(123)  
  
# Create a mock dataset  
mock\_data <- tibble(  
 opinion\_100 = runif(1000, min = 0, max = 100), # Random values for 100-point scale  
 opinion\_7 = sample(1:7, 1000, replace = TRUE), # Random values for 7-point scale  
 age\_group = sample(c('18-29', '30-44', '45-64', '65+'), 1000, replace = TRUE), # Age groups  
 gender = sample(c('Male', 'Female', 'Non-binary'), 1000, replace = TRUE) # Gender categories  
)  
  
# Display the first few rows of the dataset  
head(mock\_data)

## # A tibble: 6 × 4  
## opinion\_100 opinion\_7 age\_group gender   
## <dbl> <int> <chr> <chr>   
## 1 28.8 5 30-44 Male   
## 2 78.8 2 65+ Male   
## 3 40.9 3 18-29 Non-binary  
## 4 88.3 6 45-64 Male   
## 5 94.0 7 18-29 Female   
## 6 4.56 1 65+ Non-binary

### 0.3.2 Explanation of Functions Used to Create the DataFrame

In this section, we’ll explain the key R functions used to create the mock public opinion dataset.

#### 0.3.2.1 1. tibble()

The tibble() function is part of the dplyr package and is used to create data frames in a more modern and flexible way compared to data.frame(). Tibble is particularly useful because it prints cleaner in the console and is more forgiving with variable types.

mock\_data <- tibble(  
 opinion\_100 = runif(1000, min = 0, max = 100), # Random values for 100-point scale  
 opinion\_7 = sample(1:7, 1000, replace = TRUE), # Random values for 7-point scale  
 age\_group = sample(c('18-29', '30-44', '45-64', '65+'), 1000, replace = TRUE), # Age groups  
 gender = sample(c('Male', 'Female', 'Non-binary'), 1000, replace = TRUE) # Gender categories  
)

* **Key Characteristics**:
  + **Compact printing**: Tibble limits the number of rows and columns printed by default, which makes it cleaner for large datasets.
  + **Variable Handling**: Tibbles don’t change variable types (e.g., converting strings to factors) unless explicitly told to do so.
* **How to Modify for Future Use**: To change the number of rows or variables in the dataset, simply modify the arguments passed within the tibble() function. For example, if you need more observations, change 1000 to a larger number.

#### 0.3.2.2 2. runif()

The runif() function generates random numbers from a uniform distribution. In this case, it’s used to create the opinion\_100 variable, which represents a score on a 100-point scale.

opinion\_100 = runif(1000, min = 0, max = 100)

* **Parameters**:
  + n: The number of random numbers to generate (in this case, 1000).
  + min: The minimum value of the generated numbers (0 in this case).
  + max: The maximum value of the generated numbers (100 in this case).
* **How to Modify for Future Use**: To change the range of the opinion scale, adjust the min and max values. For example, runif(1000, min = 1, max = 10) would generate values between 1 and 10.

#### 0.3.2.3 3. sample()

The sample() function randomly selects elements from a vector. It’s used to assign random values to opinion\_7, age\_group, and gender in the mock dataset.

opinion\_7 = sample(1:7, 1000, replace = TRUE)  
age\_group = sample(c('18-29', '30-44', '45-64', '65+'), 1000, replace = TRUE)  
gender = sample(c('Male', 'Female', 'Non-binary'), 1000, replace = TRUE)

* **Parameters**:
  + x: The vector from which to sample (e.g., 1:7 for a 7-point scale, or the age groups in age\_group).
  + size: The number of elements to select. In this case, we are selecting 1000 elements to match the number of observations.
  + replace: Logical value indicating whether sampling is done with replacement. Here, we use replace = TRUE, which allows repeated values (since each category can appear multiple times).
* **How to Modify for Future Use**:
  + To change the values being sampled (e.g., to use a different opinion scale), modify the x argument. For example, to change opinion\_7 to a 5-point scale, use sample(1:5, 1000, replace = TRUE).
  + To change the categories in age\_group or gender, modify the vectors being passed to the x argument (e.g., add more age categories or change the gender options).

#### 0.3.2.4 4. set.seed()

The set.seed() function ensures that random number generation produces the same results each time the code is run. This is useful for reproducibility, especially in tutorials or research.

set.seed(123)

* **How to Modify for Future Use**: You can change the number passed to set.seed() to generate different sets of random numbers. However, using the same seed will always generate the same random numbers, which is crucial if you want your analysis to be reproducible by others.

### 0.3.3 Customizing the Dataset

To adjust the dataset for future uses, you can: - **Change the number of observations** by modifying the value in runif(1000) or sample(1:7, 1000). For example, to generate 500 observations, change 1000 to 500. - **Modify the opinion scales** by adjusting the ranges in runif(). For example, change min = 0 and max = 100 to use a different range of opinion values. - **Change the age groups** by modifying the list c('18-29', '30-44', '45-64', '65+') to include different or more categories.

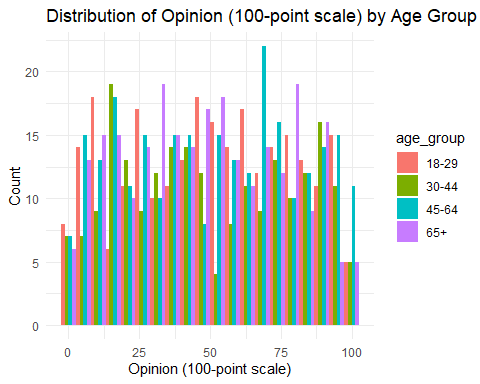
## 0.4 Creating Visualizations

Now that we have the dataset, we can create some visualizations using ggplot2.

### 0.4.1 Histogram: Distribution of Opinion (100-point scale)

We’ll start with a histogram to show the distribution of the 100-point opinion scale, grouped by age.

# Histogram of 100-point opinion scale by age group  
ggplot(mock\_data, aes(x = opinion\_100, fill = age\_group)) +  
 geom\_histogram(bins = 20, position = 'dodge') +  
 labs(title = "Distribution of Opinion (100-point scale) by Age Group",  
 x = "Opinion (100-point scale)",  
 y = "Count") +  
 theme\_minimal()



### 0.4.2 Explanation:

* **ggplot()**: The base function for creating plots. It defines the dataset (mock\_data) and aesthetics (aes()) such as the x-axis (opinion\_100) and fill color (age\_group).
* **geom\_histogram()**: Creates the histogram, with bins = 20 specifying the number of bins.
* **labs()**: Adds a title and labels to the x- and y-axes.
* **theme\_minimal()**: Applies a clean, minimal theme to the plot.

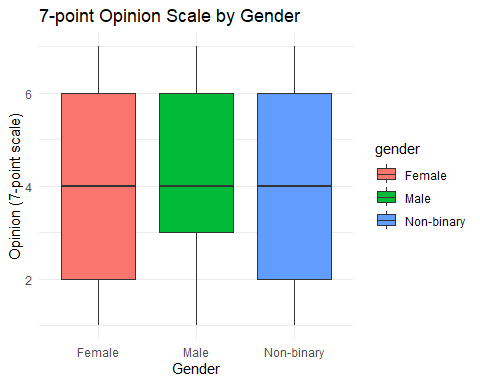
### 0.4.3 Customizing the Histogram

To customize the histogram for future use: - **Change the number of bins** by modifying the bins argument in geom\_histogram(). - **Group by a different variable** by changing fill = age\_group to another variable, such as gender.

### 0.4.4 Boxplot: 7-point Opinion Scale by Gender

Next, we create a boxplot to visualize the 7-point opinion scale across gender categories.

# Boxplot of 7-point opinion scale by gender  
ggplot(mock\_data, aes(x = gender, y = opinion\_7, fill = gender)) +  
 geom\_boxplot() +  
 labs(title = "7-point Opinion Scale by Gender",  
 x = "Gender",  
 y = "Opinion (7-point scale)") +  
 theme\_minimal()



### 0.4.5 Explanation:

* **geom\_boxplot()**: Creates a boxplot. The x axis represents gender, and the y axis represents the opinion\_7 scale.
* **fill = gender**: Colors the boxes by gender.

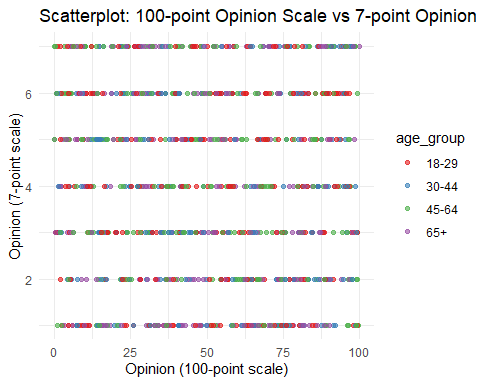
### 0.4.6 Customizing the Boxplot

To modify the boxplot: - **Change the variable on the x-axis** by replacing gender with another categorical variable, such as age\_group. - **Change the variable on the y-axis** by replacing opinion\_7 with another numeric variable, such as opinion\_100.

### 0.4.7 Scatterplot: Opinion (100-point scale) vs. Opinion (7-point scale)

Lastly, we create a scatterplot to examine the relationship between the two opinion variables.

# Scatterplot of 100-point opinion scale vs 7-point opinion scale  
ggplot(mock\_data, aes(x = opinion\_100, y = opinion\_7, color = age\_group)) +  
 geom\_point(alpha = 0.6) + # Add scatter points with some transparency  
 labs(title = "Scatterplot: 100-point Opinion Scale vs 7-point Opinion Scale",  
 x = "Opinion (100-point scale)",  
 y = "Opinion (7-point scale)") +  
 theme\_minimal() +  
 scale\_color\_brewer(palette = "Set1") # Optional: set a color palette



### 0.4.8 Explanation:

* **geom\_point()**: Creates the scatterplot. Points are plotted based on opinion\_100 (x-axis) and opinion\_7 (y-axis). The points are colored by age\_group.
* **alpha = 0.6**: Sets the transparency of the points to prevent overplotting.
* **scale\_color\_brewer()**: Applies a color palette from the RColorBrewer package. You can remove or change the palette to modify the colors.

### 0.4.9 Customizing the Scatterplot

For future uses: - **Change the transparency** by modifying the alpha argument in geom\_point(). - **Use a different color scheme** by modifying or removing scale\_color\_brewer(). - **Group by a different variable** by changing color = age\_group to another categorical variable like gender.

## 0.5 Conclusion

In this tutorial, we created a mock public opinion dataset and explored several visualization techniques using ggplot2. You can easily modify the dataset generation and visualizations for different purposes by adjusting the parameters in the code. Feel free to experiment with the settings to suit your needs!