

Categorical x Categorical data (1 of 2)

Aug 7, 2023

Exploring Bivariate Categorical Data

THIS CHAPTER explores how to summarize and visualize *Bivariate, categorical* data.

1. Bivariate analysis involves examining the **relationship between two variables**. For instance, we might examine the relationship between a person's gender (male, female, or non-binary) and whether they own a car (yes or no). By exploring these two categorical variables together, we can discern potential correlations or associations.
2. As an extension, multivariate analysis involves the simultaneous observation and analysis of more than two variables. We study multivariate data in the next chapter.
3. **Contingency Table:** A contingency table, also known as a **cross-tabulation** or **crosstab**, is a type of table in a matrix format that displays the frequency distribution of the variables. In the case of a univariate factor variable, a contingency table is essentially the same as a frequency table, as there's only one variable involved. In more complex analyses involving two or more variables, contingency tables provide a way to examine the interactions between the variables. [1]
4. **Data:** Suppose we run the following code to prepare the `mtcars` data for subsequent analysis and save it in a tibble called `tb`.

```
# Load the required libraries, suppressing annoying startup messages
library(tibble)
suppressPackageStartupMessages(library(dplyr))
# Read the mtcars dataset into a tibble called tb
data(mtcars)
tb <- as_tibble(mtcars)
# Convert relevant columns into factor variables
tb$cyl <- as.factor(tb$cyl) # cyl = {4,6,8}, number of cylinders
tb$am <- as.factor(tb$am) # am = {0,1}, 0:automatic, 1: manual transmission
tb$vs <- as.factor(tb$vs) # vs = {0,1}, v-shaped engine, 0:no, 1:yes
tb$gear <- as.factor(tb$gear) # gear = {3,4,5}, number of gears
```

```
# Directly access the data columns of tb, without tb$mpg
attach(tb)
```

Frequency Tables for Bivariate Categorical Data

1. As an illustration, let us investigate the bivariate relationship between the number of cylinders (`cyl`) and whether the car has an automatic or manual transmission, (`am=1` for manual, `am=0` for automatic).
2. `table()`: We can use this function to generate a contingency table of these two variables.

```
table(tb$cyl, tb$am)
```

	0	1
4	3	8
6	4	3
8	12	2

- In this code, a two-way frequency table of `am` and `cyl` is created using the `table()` function. The frequency of each grouping of categories is displayed in the table that results. As an illustration, there are 8 cars with a manual gearbox and 4 cylinders.
- `addmargins()`: The `addmargins()` function is used to add row and/or column totals to a table.

```
t0 <- table(tb$cyl, tb$am)
addmargins(t0)
```

	0	1	Sum
4	3	8	11
6	4	3	7
8	12	2	14
Sum	19	13	32

```
t0 <- table(tb$cyl, tb$am)
addmargins(t0,1)
```

	0	1	
4	3	8	
6	4	3	
8	12	2	
Sum	19	13	

- In this variation, the 1 in the function call indicates that we want to add row totals. So, this command adds the totals (Sum) of each row as a row in the contingency table.

```
t0 <- table(tb$cyl, tb$am)
addmargins(t0, 2)
```

	0	1	Sum
4	3	8	11
6	4	3	7
8	12	2	14

- In this variation, the 2 specifies that we want to add column totals. Thus, it adds the totals (Sum) of each column to the contingency table.
3. **xtabs()**: This function provides a more versatile way to generate cross tabulations or contingency tables. It differs from the **table()** function by allowing the use of weights and formulas. Here's how we can use it:

```
xtabs(~ cyl + am,
      data = tb)
```

	am	
cyl	0	1
4	3	8
6	4	3
8	12	2

- In the code, we have used **xtabs()** to construct a cross-tabulation of **am** and **cyl**.
- The syntax **~ cyl + am** is interpreted as a formula, signifying that we aim to cross-tabulate these variables. The output is a table akin to what we obtain with **table()**, but with the added advantage of accommodating more intricate analyses.
- An important advantage of **xtabs()** over **table()** is its superior handling of missing values or NAs; it doesn't automatically exclude them, which is beneficial when dealing with real-world data that often includes missing values. [1]

4. **fable()**: This function is a powerful tool that offers an advanced way to create and display contingency tables. Here's an example of its use:

```
fable(tb$cyl, tb$am)
```

```
0 1
4 3 8
6 4 3
8 12 2
```

- In this code, we've employed the **fable()** function to create a contingency table of **am** and **cyl**. The output of this function is similar to what we get using **table()**, but it presents the information in a flat, compact layout, which can be particularly helpful especially when dealing with more than two variables.
- One key advantage of **fable()** is that it creates contingency tables in a more readable format when dealing with more than two categorical variables, making it easier to visualize and understand complex multivariate relationships.
- Do note, like **xtabs()**, **fable()** also handles missing values or NAs effectively, making it a reliable choice for real-world data that might contain missing values.

5. We can also use package **dplyr** to generate contingency tables.

```
library(dplyr)
tb %>%
  group_by(cyl, am) %>%
  summarise(Frequency = n())
```

``summarise()`` has grouped output by 'cyl'. You can override using the ``.groups`` argument.

```
# A tibble: 6 x 3
# Groups:   cyl [3]
  cyl   am Frequency
<fct> <fct>     <int>
1 4     0         3
2 4     1         8
3 6     0         4
4 6     1         3
5 8     0        12
6 8     1         2
```

Proportions Table for Bivariate Categorical Data

6. `prop.table()`: This function is an advantageous tool to understand the relative proportions rather than raw frequencies. It converts a contingency table into a table of proportions. Here is how we could utilize this function:

```
freq <- table(tb$cyl, tb$am)
prop <- prop.table(freq)
round(prop,3)
```

	0	1
4	0.094	0.250
6	0.125	0.094
8	0.375	0.062

- In this code, we first generate a frequency table with the `table()` function, using `cyl` and `am` as our variables. Then, we employ the `prop.table()` function to convert this frequency table (`freq_table`) into a proportions table (`prop_table`).
- This resulting `prop_table` reveals the proportion of each combination of `cyl` and `am` categories relative to the total number of observations. This can provide insightful context, allowing us to see how each combination fits into the overall distribution. For instance, we could learn what proportion of cars in our dataset have 4 cylinders and a manual transmission.
- A noteworthy benefit of `prop.table()` is its ability to normalize the data, providing a perspective based on relative proportions instead of absolute numbers, which can often provide a clearer view of the underlying patterns in the data.

7. We can alternately use package `dplyr` to achieve this as follows.

```
library(dplyr)
tb %>%
  group_by(cyl, am) %>%
  summarise(Frequency = n()) %>%
  mutate(Proportion = Frequency / sum(Frequency))
```

``summarise()`` has grouped output by 'cyl'. You can override using the ``groups`` argument.

```
# A tibble: 6 x 4
# Groups:   cyl [3]
  cyl   am  Frequency Proportion
<fct> <fct>    <int>     <dbl>
1 4     0         3      0.273
2 4     1         8      0.727
3 6     0         4      0.571
4 6     1         3      0.429
5 8     0        12      0.857
6 8     1         2      0.143
```

- In this code, `group_by(cyl, am)` groups the data by `cyl` and `am`, `summarise(Frequency = n())` calculates the frequency for each group, and `mutate(Proportion = Frequency / sum(Frequency))` calculates the proportions by dividing each frequency by the total sum of frequencies.
- The `mutate()` function adds a new column to the dataframe, keeping the original data intact.
- **Rounding:** If we wanted to round-off the proportion up to 4 decimal places, we could write the following code.

```
library(dplyr)
tb %>%
  group_by(cyl, am) %>%
  summarise(Frequency = n()) %>%
  mutate(Proportion = round(Frequency / sum(Frequency), 4))
```

``summarise()`` has grouped output by 'cyl'. You can override using the ``.groups`` argument.

```
# A tibble: 6 x 4
# Groups:   cyl [3]
  cyl   am  Frequency Proportion
<fct> <fct>    <int>     <dbl>
1 4     0         3      0.273
2 4     1         8      0.727
3 6     0         4      0.571
4 6     1         3      0.429
5 8     0        12      0.857
6 8     1         2      0.143
```

Margins in Proportions Tables

1. Different proportions provide various perspectives on the relationship between categorical variables in our dataset. We can calculate the i) Proportions for **Each Cell**; (ii) **Row-Wise* Proportions**; (iii) Column-Wise** Proportions. This forms a crucial part of exploratory data analysis.
2. **Proportions for Each Cell**: This calculates the ratio of each cell to the overall total.

```
freq <- table(tb$cyl, tb$am)
# Compute cell-wise proportions
cellprop <- prop.table(freq)
# Add totals for each row and column
cellprop <- addmargins(cellprop)
# Round off the results to three decimal places and display as percentages
100*round(cellprop, 3)
```

	0	1	Sum
4	9.4	25.0	34.4
6	12.5	9.4	21.9
8	37.5	6.2	43.8
Sum	59.4	40.6	100.0

3. **Row-Wise Proportions**: Here, we compute the proportion of each cell relative to the total of its row.

```
# Compute row-wise proportions
rowprop <- prop.table(freq, margin = 1)
# Append totals for each row and column
rowprop <- addmargins(rowprop, 2)
# Round off the results to three decimal places and display as percentages
100*round(rowprop, 3)
```

	0	1	Sum
4	27.3	72.7	100.0
6	57.1	42.9	100.0
8	85.7	14.3	100.0

4. **Column-Wise Proportions**: Here, we determine the proportion of each cell relative to the total of its column.

```
# Compute column-wise proportions
colprop <- prop.table(freq, margin = 2)
# Append totals for each row and column
colprop <- addmargins(colprop,1)
# Round off the results to three decimal places and display as percentages
100*round(colprop, 3)
```

	0	1
4	15.8	61.5
6	21.1	23.1
8	63.2	15.4
Sum	100.0	100.0

- In this code, the `prop.table()` function is initially applied with the `margin` parameter to calculate cell, row, and column proportions from the contingency table. We then invoke `addmargins()` to sum up and include the marginal totals for each row and column. Lastly, the `round()` function aids in displaying the proportions with one decimal place.

Visualizing Bivariate Categorical Data

1. Grouped Barplots and Stacked Barplots serve as powerful tools for representing and understanding bivariate categorical data, where both variables are categorical in nature.
2. Grouped Barplots, often referred to as side-by-side bar plots, illustrate the relationship between two categorical variables by placing bars corresponding to one category of a variable next to each other, differentiated by color or pattern. This layout facilitates a direct comparison between categories of the second variable. Grouped bar plots are particularly effective when we are interested in comparing the distribution of a categorical variable across different groups
3. On the other hand, stacked bar plots present a similar relationship between two categorical variables, but rather than aligning bars side by side, they stack bars on top of one another. This results in a single bar for each category of one variable, with the length of different segments in each bar corresponding to the counts or proportions of the categories of the other variable. Stacked bar plots are advantageous when we're interested in the total size of groups as well as the distribution of a variable across groups. [2]

4. Grouped Barplot

```
# Create a table with count by transmission and number of cylinders
freq <- table(tb$cyl, tb$am)
```

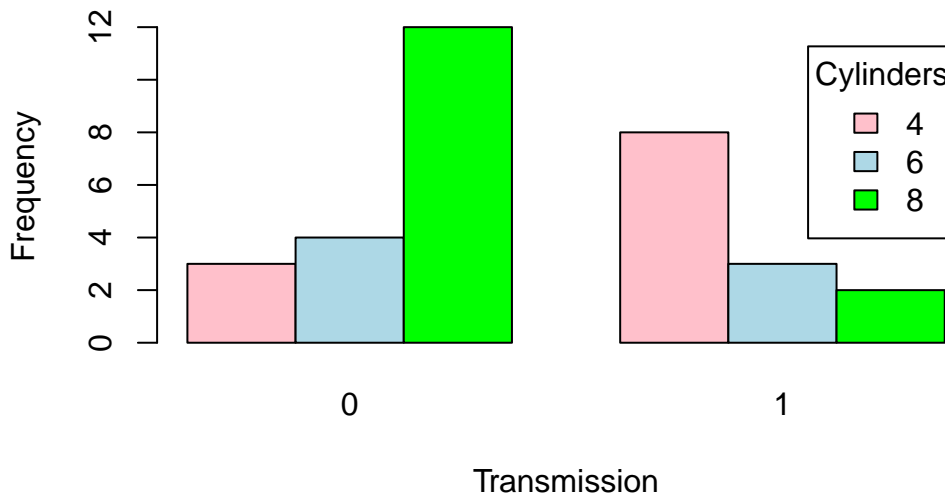


```
freq
```

```
      0  1  
4 3 8  
6 4 3  
8 12 2
```

```
# Create a Grouped bar plot  
barplot(freq,  
        beside = TRUE,  
        col = c("pink", "lightblue", "green"),  
        xlab = "Transmission", ylab = "Frequency",  
        main = "Grouped Barplot of Frequency by transmission and cylinders",  
        legend.text = rownames(freq),  
        args.legend = list(title = "Cylinders"))
```

Grouped Barplot of Frequency by transmission and cylind



5. Here's a detailed explanation of each argument passed to the `barplot()` function:

- **freq**: This is the dataset being visualized, which we anticipate to be a contingency table of `am` and `cyl` variables.
- **beside = TRUE**: This argument is specifying that the bars should be positioned next to each other, which means that for each level of `am`, there will be a distinct bar for each level of `cyl`.

- `col = c("pink", "lightblue", "green")`: Here, we are setting the colors of the bars to pink, light blue, and green.
- `xlab = "Transmission"` and `ylab = "Frequency"`: These arguments set the labels for the x and y-axes, respectively.
- `main = "Grouped Barplot of Frequency by transmission and cylinders"`: This argument assigns a title to the plot.
- `legend.text = rownames(freq)`: This creates a legend for the plot, using the row names of `freq` as the legend text.
- `args.legend = list(title = "Cylinders")`: This sets the title of the legend to "Cylinders".

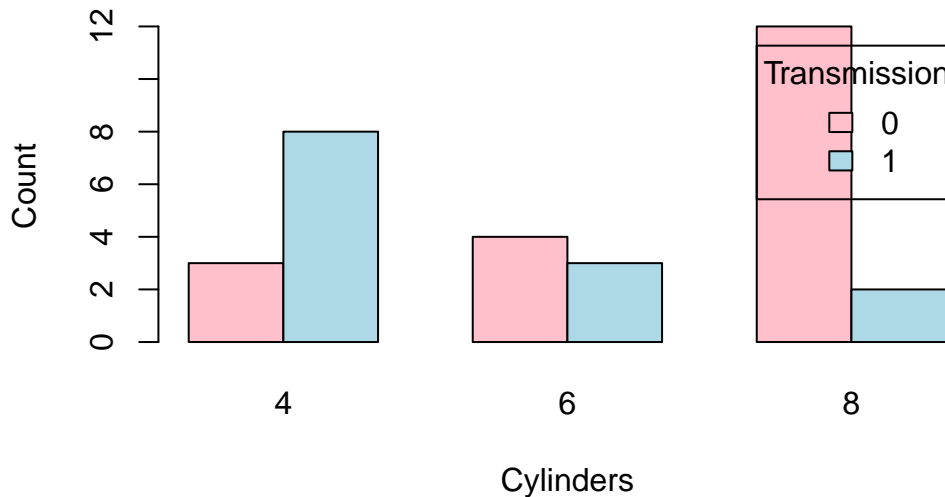
6. Consider this alternate barplot.

```
# Create a table with count by transmission and number of cylinders
freqInverted <- table(tb$am, tb$cyl)
freqInverted
```

```
      4  6  8
0    3  4 12
1    8  3  2
```

```
# Create the bar plot
barplot(freqInverted,
        beside = TRUE,
        col = c("pink", "lightblue"),
        xlab = "Cylinders", ylab = "Count",
        main = "Grouped Barplot of Frequency by cylinders and transmission",
        legend.text = rownames(freqInverted),
        args.legend = list(title = "Transmission"))
```

Grouped Barplot of Frequency by cylinders and transmiss



The most significant differences from the previous Grouped Barplot and this one are as follows:

- `freqInverted`: The contingency table's axes have been swapped or inverted. Hence, the table's rows now correspond to the `am` variable (transmission), and its columns correspond to the `cyl` variable (cylinders).
- `xlab = "Cylinders"` and `ylab = "Count"`: These arguments set the labels for the x and y-axes, respectively. This is a departure from the previous plot where the x-axis represented 'Transmission'. In this case, the x-axis corresponds to 'Cylinders'.
- `legend.text = rownames(freqInverted)` and `args.legend = list(title = "Transmission")`: In the legend, the roles of 'Transmission' and 'Cylinders' are reversed compared to the previous plot.
- To put it succinctly, the main distinction between the two plots is the swapping of the roles of the `cyl` and `am` variables. In the second plot, 'Cylinders' is on the x-axis, which was occupied by 'Transmission' in the first plot. This perspective shift helps to understand the data in a different light, adding another dimension to our exploratory data analysis. [2]

5. Stacked Barplot

```
# Create a table with count by transmission and number of cylinders
freq <- table(tb$cyl, tb$am)
freq
```

```

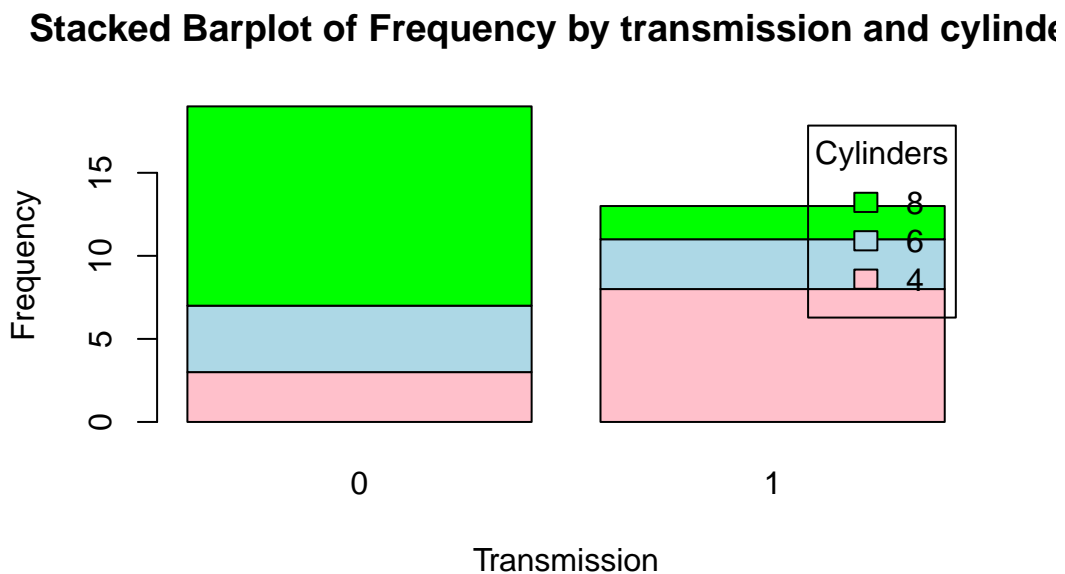
0 1
4 3 8
6 4 3
8 12 2

```

```

# Create a Stacked bar plot
barplot(freq,
        beside = FALSE,
        col = c("pink", "lightblue", "green"),
        xlab = "Transmission", ylab = "Frequency",
        main = "Stacked Barplot of Frequency by transmission and cylinders",
        legend.text = rownames(freq),
        args.legend = list(title = "Cylinders"))

```



There are a few key differences between this Stacked Barplot and the original Grouped Barplot:

- **beside = FALSE:** In the original code, `beside = TRUE` was used to generate a grouped bar plot, where each set of bars corresponding to each transmission type (automatic or manual) were displayed side by side. However, with `beside = FALSE`, we obtain a stacked bar plot. In this plot, the bars corresponding to each cylinder category (4, 6, or 8 cylinders) are stacked on top of one another for each transmission type.
- **main = "Stacked Barplot of Frequency by transmission and cylinders":** The title of the plot also reflects this change, mentioning now that it's a stacked bar plot instead of a grouped bar plot.

- Finally, here is a Stacked Barplot, corresponding to the second Grouped Barplot discussed above.

```
# Create a table with count by transmission and number of cylinders
freqInverted <- table(tb$am, tb$cyl)
freqInverted
```

```
      4  6  8
0     3  4 12
1     8  3  2
```

```
# Create the bar plot
barplot(freqInverted,
        beside = FALSE,
        col = c("pink", "lightblue"),
        xlab = "Cylinders", ylab = "Count",
        main = "Stacked Barplot of Frequency by cylinders and transmission",
        legend.text = rownames(freqInverted),
        args.legend = list(title = "Transmission"))
```

Stacked Barplot of Frequency by cylinders and transmissi



- The grouped bar plot helps in comparing the number of cylinders across transmission types side by side, while the stacked bar plot gives an overall comparison in terms of total number of cars, with the frequency of each cylinder type stacked on top of the other. The choice between a stacked and a grouped bar plot would depend on the specific aspects

of the data one would want to highlight. Taken together, grouped and stacked bar plots offer visually appealing and intuitive methods for presenting bivariate categorical data, allowing us to understand and analyze relationships between categorical variables in a meaningful way. [3]

Using ggplot2

Grouped Barplot using ggplot2

```
freq <- table(tb$cyl, tb$am)

# Load required library
library(ggplot2)
```

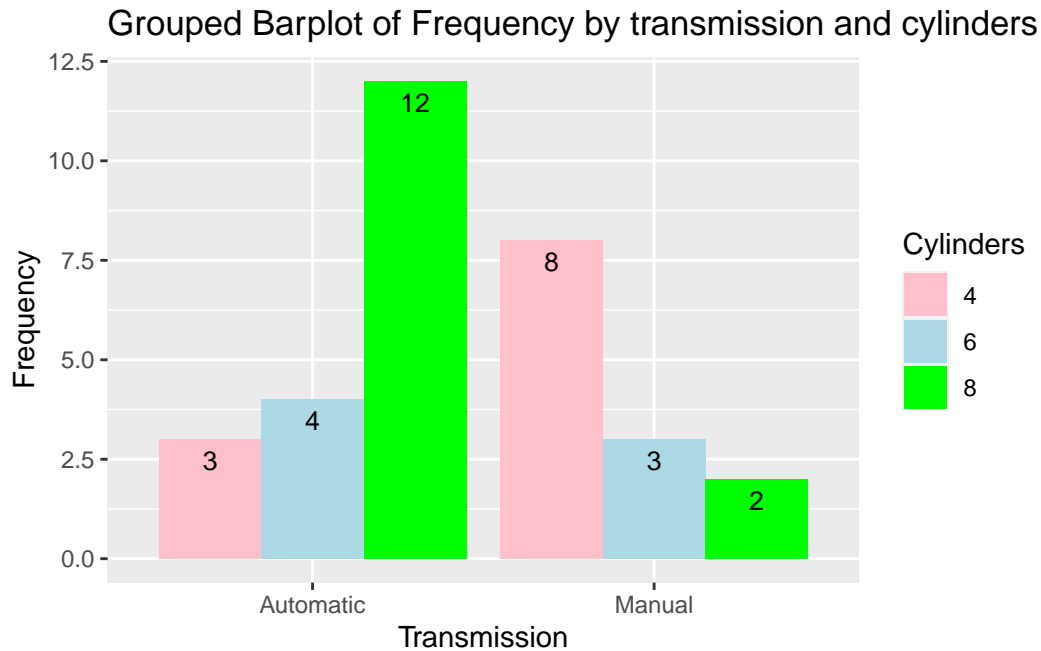
Attaching package: 'ggplot2'

The following object is masked from 'tb':

mpg

```
# Convert the table to a data frame and reshape to 'long' format
df <- as.data.frame.table(freq)
names(df) <- c("Cylinders", "Transmission", "Frequency")
df$Cylinders <- factor(df$Cylinders)
df$Transmission <- factor(df$Transmission, labels = c("Automatic", "Manual"))

# Create the Grouped Barplot using ggplot2
ggplot(df,
  aes(x = Transmission, y = Frequency, fill = Cylinders)) +
  geom_bar(stat = "identity",
    position = position_dodge()) +
  geom_text(aes(label=Frequency,
    vjust=1.6, color="black", position = position_dodge(0.9), size=3.5) +
  labs(x = "Transmission", y = "Frequency", fill = "Cylinders",
    title = "Grouped Barplot of Frequency by transmission and cylinders") +
  scale_fill_manual(values = c("pink", "lightblue", "green"))
```



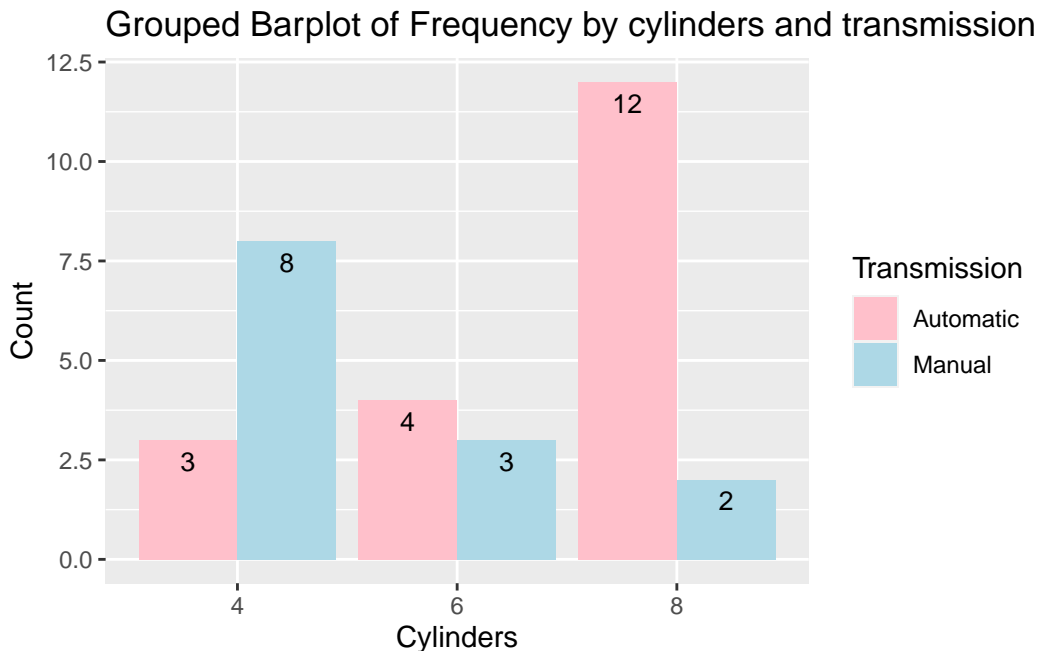
In this code:

- We first convert the frequency table into a data frame that `ggplot2` can use. This involves converting the table to a data frame and then reshaping it to long format.
- We then use the `ggplot()` function to initiate the plot and specify the aesthetic mappings. Here, we map 'Transmission' to the x-axis, 'Frequency' to the y-axis, and 'Cylinders' to the fill aesthetic which controls the color of the bars.
- `geom_bar()` with `stat = "identity"` is used to add the bar geometry to the plot.
- `position = position_dodge()` ensures the bars are placed side by side, which creates the grouped effect.
- `labs()` is used to add labels to the plot.
- `scale_fill_manual()` allows us to manually specify the colors for the bars.

```
# Convert the table to a data frame and reshape to 'long' format
df_inverted <- as.data.frame.table(freqInverted)
names(df_inverted) <- c("Transmission", "Cylinders", "Count")
df_inverted$Transmission <- factor(df_inverted$Transmission, labels = c("Automatic", "Manual"))
df_inverted$Cylinders <- factor(df_inverted$Cylinders)

# Create the Grouped Barplot using ggplot2
```

```
ggplot(df_inverted,
       aes(x = Cylinders, y = Count, fill = Transmission)) +
  geom_bar(stat = "identity",
          position = position_dodge()) +
  geom_text(aes(label = Count), vjust = 1.6, color = "black", position = position_dodge(0.5)) +
  labs(x = "Cylinders", y = "Count", fill = "Transmission",
       title = "Grouped Barplot of Frequency by cylinders and transmission") +
  scale_fill_manual(values = c("pink", "lightblue"))
```



In this plot:

- The `geom_text()` function is used in the same way as in the previous example to add frequency labels to the bars.
- The `scale_fill_manual()` function is used to specify the colors for the different transmission types.

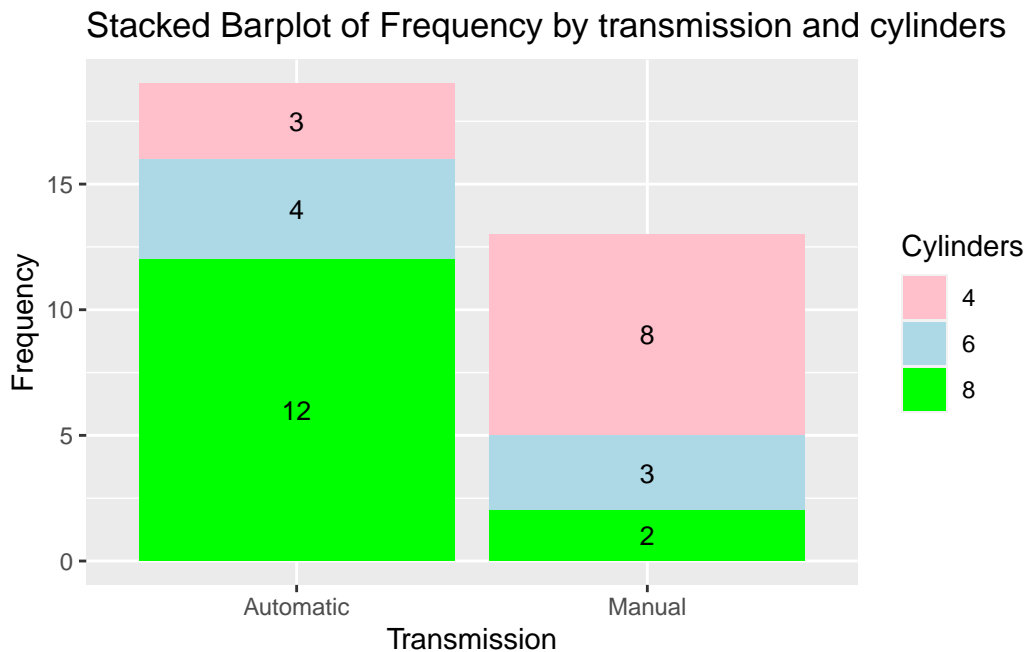
Stacked Barplots using ggplot2

- To replicate the stacked bar plot with `ggplot2`, we can utilize the same dataframe transformation as before. Then, we apply the `geom_bar()` function but without the dodge positioning this time, since we want the bars stacked. Here is how we can accomplish this:


```
# Load required library
library(ggplot2)

# Convert the table to a data frame and reshape to 'long' format
df <- as.data.frame.table(freq)
names(df) <- c("Cylinders", "Transmission", "Frequency")
df$Transmission <- factor(df$Transmission, labels = c("Automatic", "Manual"))
df$Cylinders <- factor(df$Cylinders)

# Create the Stacked Barplot using ggplot2
ggplot(df, aes(x = Transmission, y = Frequency, fill = Cylinders)) +
  geom_bar(stat = "identity") +
  geom_text(aes(label = Frequency), position = position_stack(vjust = 0.5), color = "black") +
  labs(x = "Transmission", y = "Frequency", fill = "Cylinders",
       title = "Stacked Barplot of Frequency by transmission and cylinders") +
  scale_fill_manual(values = c("pink", "lightblue", "green"))
```



In this code:

- The `geom_bar(stat = "identity")` is used to create a stacked bar plot.
- The `scale_fill_manual()` function is used to specify the colors for the different cylinder categories. Again, we changed the order of the Transmission and Cylinders columns to

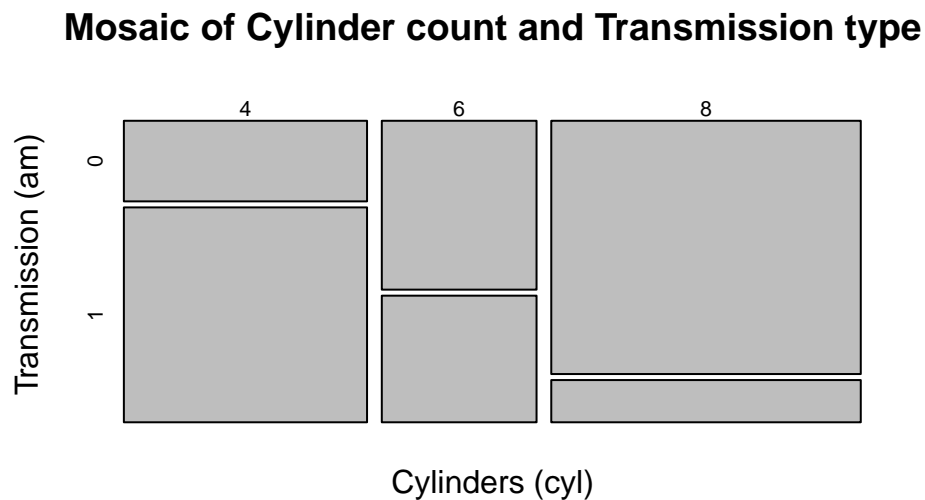
suit this specific plot.

- The `geom_text()` function is used to add the labels to the stacked bars. We use `position_stack()` to position the labels in the middle of the stacked sections.
- The `vjust` argument inside `position_stack()` controls the vertical positioning of the labels, and 0.5 puts them in the middle.
- As before, `scale_fill_manual()` allows us to specify the colors for the different cylinder categories.

Mosaic Plots for Bivariate Categorical Data

1. A mosaic plot is a graphical method for visualizing data from two or more qualitative variables / categorical data. It is a form of area plot that can provide a visual representation of the frequency or proportion of the different categories within the variables.
2. The following code generates a mosaic plot from a contingency table of two variables: `cyl` (cylinders) and `am` (transmission).

```
# Create a mosaic plot
mosaicplot(table(tb$cyl, tb$am),
            main = "Mosaic of Cylinder count and Transmission type",
            xlab = "Cylinders (cyl)",
            ylab = "Transmission (am)")
```



2. In a mosaic plot, we interpret two categorical variables on two axes. The **width** of each section on one axis signifies the proportion of that particular category in our dataset. Conversely, the **height** of a section on the other axis illustrates the proportion of that

category, contingent on the specific category from the first variable. Therefore, the **area** of each rectangle directly corresponds to the frequency or proportion of observations falling within that specific combination of categories (Hartigan & Kleiner, 1981).

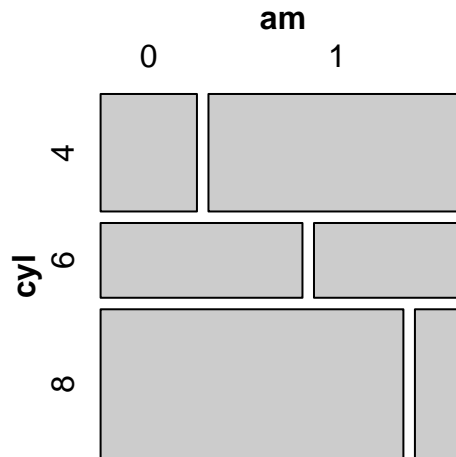
3. By combining both the height and width of the rectangles, a mosaic plot gives us a visual representation of the joint distribution of the two categorical variables. It helps us to identify patterns, associations, and dependencies between the two variables.
4. In the above code:
 - The `table()` function is used to create a contingency table of the `cyl` and `am` columns of the `tb` tibble. This contingency table represents the counts of all combinations of `cyl` and `am` in the data.
 - The `mosaicplot()` function then creates a mosaic plot from this contingency table. The `main`, `xlab`, and `ylab` arguments are used to set the main title, x-axis label, and y-axis label of the plot, respectively.
 - This code gives a mosaic plot that visualizes the distribution of the number of cylinders by the type of transmission. Each block's width in the plot would be proportional to the number of cylinders, and the height would be proportional to the transmission type.
5. They can be used to identify breaks in independence or test hypotheses regarding the connections between the variables. They are especially helpful for examining interactions between two or more categorical variables.
6. The `vcd` package, short for Visualizing Categorical Data, provides alternative visual and analytical methods for categorical data (Meyer, Zeileis, & Hornik, 2020). Following this, a mosaic plot is created from the variables `cyl` (cylinders) and `am` (transmission type) using the `mosaic()` function.

```
# Load the vcd package
library(vcd)
```

Loading required package: grid

```
# Create a mosaic plot of mpg (miles per gallon) vs. am (transmission)
vcd::mosaic(~ cyl + am,
  data = tb,
  main = "Mosaic of Cylinder count and Transmission type")
```

Mosaic of Cylinder count and Transmission type



7. In this code:

- `library(vcd)` is used to import the `vcd` package which includes the `mosaic()` function that is superior to the base R `mosaicplot()` in its handling of labeling and legends.
 - `mosaic(~ cyl + am, data = tb, main = "Mosaic of Cylinder count and Transmission type")` creates a mosaic plot.
 - In this command, the formula `~ cyl + am` tells R to plot `cyl` against `am`. The argument `data = tb` specifies that the variables for the plot come from the `tb` data frame. The `main` argument sets the title of the plot.
8. We can customize the shading color of the mosaic plot by setting the color using the `col` argument inside the `mosaic()` function. To specify shades of blue, we can use the `RColorBrewer` package and its `brewer.pal()` function to generate a color palette:

```
# Load necessary packages
library(vcd)
library(RColorBrewer)

# Define the color palette
cols <- brewer.pal(4, "Blues")

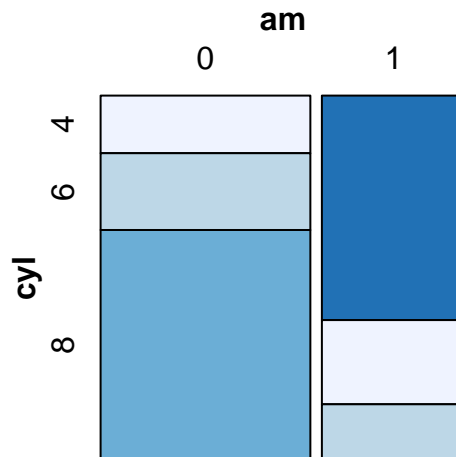
# Create a mosaic plot
vcd::mosaic(~ cyl + am,
            data = tb,
            main = "Mosaic of Cylinder count and Transmission type",
            col = cols)
```

```

shade = TRUE,
highlighting = "cyl",
highlighting_fill = cols)

```

Mosaic of Cylinder count and Transmission type



9. In this code:

- `brewer.pal(4, "Blues")` creates a color palette of 4 different shades of blue.
- `highlighting = "cyl"` means that the shading will differentiate the categories in the `cyl` variable.
- `highlighting_fill = cols` applies the `cols` color palette to the shading.

10. We can also recreate the previous mosaic plot using `ggplot2` and `ggmosaic` packages. Here is how to do it:

```

# Load the packages
library(ggplot2)
library(ggmosaic)

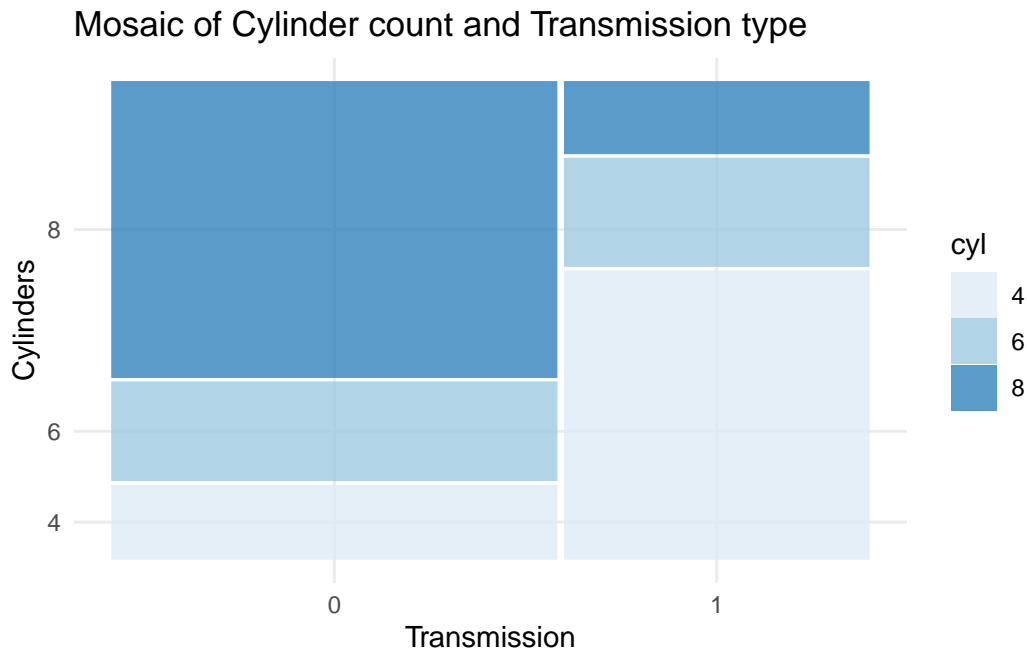
```

Attaching package: 'ggmosaic'

The following objects are masked from 'package:vcd':

`mosaic`, `spine`

```
# Create the mosaic plot
ggplot(data = tb) +
  geom_mosaic(aes(x = product(cyl, am), fill = cyl)) +
  theme_minimal() +
  labs(title = "Mosaic of Cylinder count and Transmission type",
       x = "Transmission",
       y = "Cylinders") +
  scale_fill_brewer(palette = "Blues")
```



11. In this code:

- `geom_mosaic(aes(x = product(cyl, am), fill = cyl))` creates the mosaic plot with `cyl` and `am` as the categorical variables.
- The `fill = cyl` part means that the color fill will differentiate the categories in the `cyl` variable.
- `theme_minimal()` applies a minimal theme to the plot.
- `labs()` is used to specify the labels for the plot, including the `title`, x-axis label, and y-axis label.
- `scale_fill_brewer(palette = "Blues")` specifies the color palette to be used for the fill color, in this case, a palette of blues.

Summary of Chapter 9 – Categorical Data (2 of 3)

In this chapter, we delve into an extensive exploration of various methods for visualizing bivariate categorical data using R, which include but are not limited to grouped bar plots, stacked bar plots, and mosaic plots.

An explanation is provided to distinguish between grouped and stacked bar plots, further supported by the inclusion of coding examples and variations in parameters. Both base R functions and the more advanced `ggplot2` library serve as instrumental tools to depict these techniques, furnishing readers with a comprehensive understanding of their practical usage.

The `ggplot2` library further elevates the possibilities by offering enhanced customization capabilities and superior control over aesthetics. Detailed coding examples are presented again to explain both the grouped and stacked bar plots within this library.

The chapter also introduces mosaic plots as another way of visualizing bivariate categorical data and discusses how mosaic plots can provide a visual representation of the frequency or proportion of different categories within variables. The use of the base R `mosaicplot()` function is exemplified, followed by the demonstration of the `mosaic()` function in the `vcd` package and the `ggmosaic` package, bringing full circle the spectrum of visualizations covered for bivariate categorical data.

References

- [1] Agresti, A. (2018). *An Introduction to Categorical Data Analysis* (3rd ed.). Wiley.
- Kabacoff, R. I. (2015). *R in Action: Data analysis and graphics with R* (2nd ed.). Manning Publications.
- Wickham, H., & Grommund, G. (2016). *R for Data Science: Import, Tidy, Transform, Visualize, and Model Data*. O'Reilly Media.
- Hair, J. F., Black, W. C., Babin, B. J., & Anderson, R. E. (2018). *Multivariate data analysis* (8th ed.). Cengage Learning.
- [2] Unwin, A. (2015). *Graphical data analysis with R*. CRC Press.
- Friendly, M. (2000). *Visualizing Categorical Data*. SAS Institute.
- Hartigan, J. A., & Kleiner, B. (1981). Mosaics for contingency tables. In *Computer Science and Statistics: Proceedings of the 13th Symposium on the Interface* (pp. 268-273).
- [3] Healy, K., & Lenard, M. T. (2014). *A practical guide to creating better looking plots in R*. University of Oregon. <https://escholarship.org/uc/item/07m6r>

[4] Meyer, D., Zeileis, A., & Hornik, K. (2020). vcd: Visualizing Categorical Data. R package version 1.4-8. <https://CRAN.R-project.org/package=vcd>

Friendly, M. (1994). Mosaic displays for multi-way contingency tables. *Journal of the American Statistical Association*, 89(425), 190-200.

Agresti, A. (2018). *An Introduction to Categorical Data Analysis* (3rd ed.). Wiley.

[5] R Core Team (2020). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. URL <https://www.R-project.org/>.