6

Basic graphs

This chapter covers

- Bar, box, and dot plots
- Pie charts and tree maps
- Histograms and kernel density plots

Whenever we analyze data, the first thing we should do is *look* at it. For each variable, what are the most common values? How much variability is present? Are there any unusual observations? R provides a wealth of functions for visualizing data. In this chapter, we'll look at graphs that help you understand a single categorical or continuous variable. This topic includes

- Visualizing the distribution of a variable
- Comparing the distribution of a variable across two or more groups

In both cases, the variable can be continuous (for example, car mileage as miles per gallon) or categorical (for example, treatment outcome as none, some, or marked). In later chapters, we'll explore graphs that display more complex relationships among variables.

The following sections explore the use of bar charts, pie charts, tree maps, histograms, kernel density plots, box plots, violin plots, and dot plots. Some of these may be familiar to you, whereas others (such as tree charts or violin plots) may be new to you. The goal, as always, is to understand your data better and to communicate this understanding to others. Let's start with bar charts.

6.1 Bar charts

A bar plot displays the distribution (frequency) of a categorical variable through vertical or horizontal bars. Using the `ggplot2` package, we can create a bar chart using the code

```
ggplot(data, aes(x=catvar) + geom_bar()
```

where *data* is a data frame and *cat*`var` is a categorical variable.

In the following examples, you'll plot the outcome of a study investigating a new treatment for rheumatoid arthritis. The data are contained in the `Arthritis` data frame distributed with the `vcd` package. This package isn't included in the default R installation, so install it before first use (`install.packages("vcd")`). Note that the `vcd` package isn't needed to create bar charts. You're installing it in order to gain access to the `Arthritis` dataset.

6.1.1 Simple bar charts

In the Arthritis study, the variable `Improved` records the patient outcomes for individuals receiving a placebo or drug:

```
> data(Arthritis, package="vcd")
> table(Arthritis$Improved)

 None   Some Marked
   42     14     28
```

Here, you see that 28 patients showed marked improvement, 14 showed some improvement, and 42 showed no improvement. We'll discuss the use of the `table()` function to obtain cell counts more fully in chapter 7.

You can graph these counts using a vertical or horizontal bar chart. The code is provided in the following listing, and the resulting graphs are displayed in figure 6.1.

Listing 6.1 Simple bar charts

```
library(ggplot2)
ggplot(Arthritis, aes(x=Improved)) + geom_bar() +     #A
 labs(title="Simple Bar chart",                       #A
      x="Improvement",                                #A
      y="Frequency")                                  #A

ggplot(Arthritis, aes(x=Improved)) + geom_bar() +     #B
 labs(title="Horizontal Bar chart",                   #B
      x="Improvement",                                #B
      y="Frequency") +                                #B
 coord_flip()                                         #B
```

#A Simple bar chart
#B Horizontal bar chart

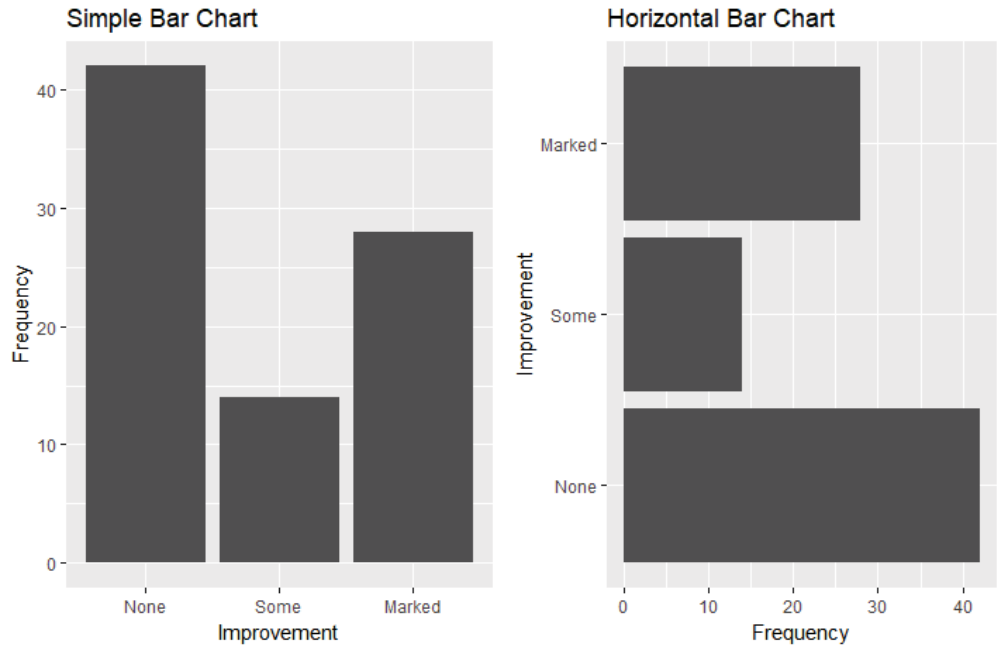


Figure 6.1 Simple vertical and horizontal bar charts

What happens if you have long labels? In section 6.1.4, you'll see how to tweak labels so that they don't overlap.

6.1.2 Stacked, grouped and filled bar charts

The central question in the Arthritis study is "How does the level of improvement vary between the placebo and treated conditions?". The `table()` function can used to generate a cross-tabulation of the variables.

```
> table(Arthritis$Improved, Arthritis$Treatment)

        Treatment
Improved Placebo Treated
  None       29      13
  Some        7       7
  Marked      7      21
```

While the tabulation is helpful, the results are easier to grasp with a bar chart. The relationship between two categorical variables can be plotted using *stacked*, *grouped*, or *filled* bar charts. The code is provided in listing 6.2 and the graph is displayed in figures 6.2.

Listing 6.2 Stacked, grouped, and filled bar charts

```
library(ggplot2)
ggplot(Arthritis, aes(x=Treatment, fill=Improved)) +   #A
  geom_bar(position = "stack") +                       #A
  labs(title="Stacked Bar chart",                      #A
       x="Treatment",                                  #A
       y="Frequency")                                  #A

ggplot(Arthritis, aes(x=Treatment, fill=Improved)) +   #B
  geom_bar(position = "dodge") +                       #B
  labs(title="Grouped Bar chart",                      #B
       x="Treatment",                                  #B
       y="Frequency")                                  #B

ggplot(Arthritis, aes(x=Treatment, fill=Improved)) +   #C
  geom_bar(position = "fill") +                        #C
  labs(title="Stacked Bar chart",                      #C
       x="Treatment",                                  #C
       y="Frequency")                                  #C
```

#A Stacked bar chart
#B Grouped bar chart
#C Filled bar chart

In the stacked bar chart, each segment represents the frequency or proportion of cases within in a given Treatment (Placebo, Treated) and Improvement (None, Some, Marked) level combination. The segments are stacked separately for each Treatment level. The grouped bar chart places the segments representing Improvement side by side within each Treatment level. The filled bar chart is a stacked bar chart rescaled so that the height of each bar is 1 and the segment heights represent proportions.

Filled bar charts are particularly useful for comparing the proportions of one categorical variable over the levels of another categorical variable. For example, the filled bar chart in figure 6.2 clearly displays the larger percentage of treated patients with marked improvement compared with patients receiving a placebo.

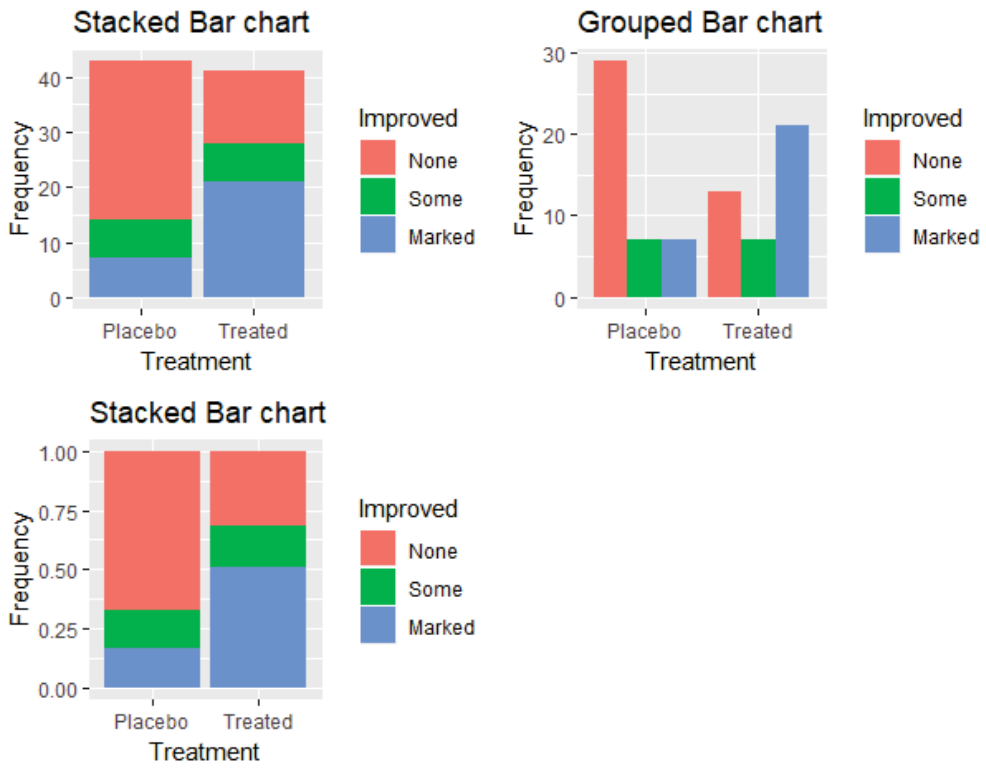


Figure 6.2 Stacked, grouped, and filled bar charts

6.1.3 Mean bar charts

Bar plots needn't be based on counts or frequencies. You can create bar charts that represent means, medians, percents, standard deviations, and so forth by summarizing the data with an appropriate statistic and passing the results to `ggplot2`.

In the following graph, we'll plot the mean illiteracy rate for regions of the United States in 1970. The built-in R dataset `state.x77` has the illiteracy rates by state, and the dataset `state.region` has the region names for each state. The following listing provides the code needed to create the graph in figure 6.3.

Listing 6.3 Bar chart for sorted mean values

```
> states <- data.frame(state.region, state.x77)
> library(dplyr)                          #1
> plotdata <- states %>%
   group_by(state.region) %>%
   summarize(mean = mean(Illiteracy))
 plotdata
```

```
# A tibble: 4 x 2
  state.region   mean
  <fct>         <dbl>
1 Northeast     1
2 South         1.74
3 North Central  0.7
4 West          1.02

> ggplot(plotdata, aes(x=reorder(state.region, mean), y=mean)) +   #2
    geom_bar(stat="identity") +
    labs(x="Region",
         y="",
         title = "Mean Illiteracy Rate")
```

#1 Generate means by region
#2 Plot means in a sorted bar chart

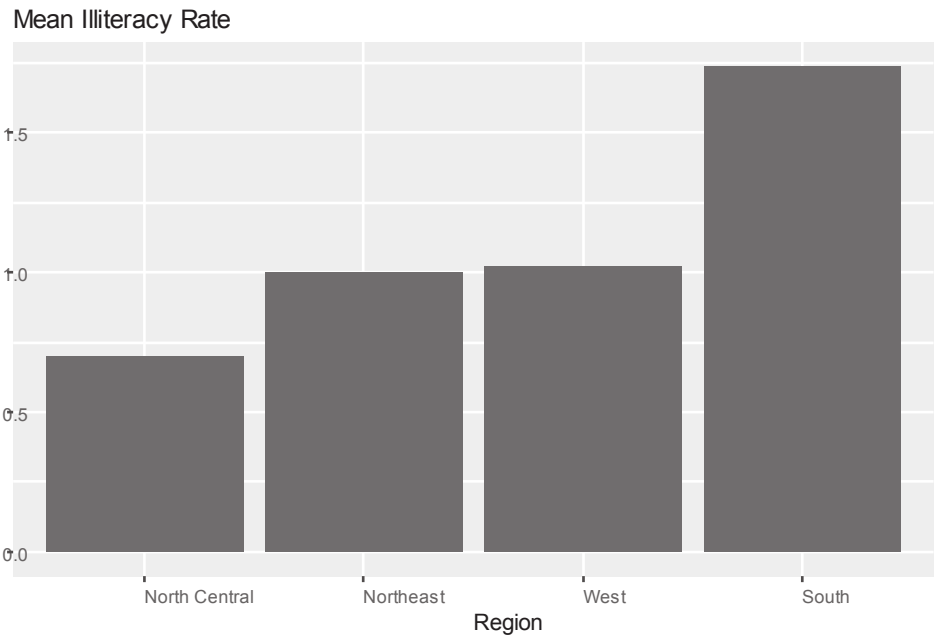


Figure 6.3 Bar chart of mean illiteracy rates for US regions sorted by rate

First, the mean illiteracy rate is calculated for each region #1. Next, the means are plotted in sorted in ascending order as bars #2. Normally, the `geom_bar()` function calculates and plots cell counts but adding the `stat="identity"` option forces the function to plot the numbers provided (means in this case). The `reorder()` function is used to order the bars by increasing mean illiteracy.

When plotting summary statistics such as means, it's good practice to indicate the variability of the estimates involved. One measure of variability is the standard error of the statistic – an estimate of the expected variation of the statistic across hypothetical repeated samples. The following plot adds error bars using the standard error of the mean.

Listing 6.4 Bar chart of mean values with error bars

```
> plotdata <- states %>%                    #1
   group_by(state.region) %>%
   summarize(n=n(),
             mean = mean(Illiteracy),
             se = sd(Illiteracy)/sqrt(n))

> plotdata

# A tibble: 4 x 4
  state.region     n  mean    se
  <fct>        <int> <dbl>  <dbl>
1 Northeast        9  1    0.0928
2 South           16  1.74 0.138
3 North Central   12  0.7  0.0408
4 West            13  1.02 0.169

> ggplot(plotdata, aes(x=reorder(state.region, mean), y=mean)) +     #2
   geom_bar(stat="identity", fill="skyblue") +
   geom_errorbar(aes(ymin=mean-se, ymax=mean+se), width=0.2) +       #3
   labs(x="Region",
        y="",
        title = "Mean Illiteracy Rate",
        subtitle = "with standard error bars")
```

#1 Generate means and standard errors by region
#2 Plot means in a sorted bar chart
#3 Add error bars

The means and standard errors are calculated for each region #1. The bars are then plotted in order of increasing illiteracy. The color is changed from a default dark grey to a lighter shade (sky blue) so that error bars to be added in the next step will stand out #2. Finally, the error bars are plotted #3. The `width` option in the `geom_errorbar()` function controls the horizontal width of the error bars and is purely aesthetic – it has no statistical meaning. In addition to displaying the mean illiteracy rates, we can see that the mean for the North Central region is the most reliable (least variability) and the West region is least reliable (largest variability).

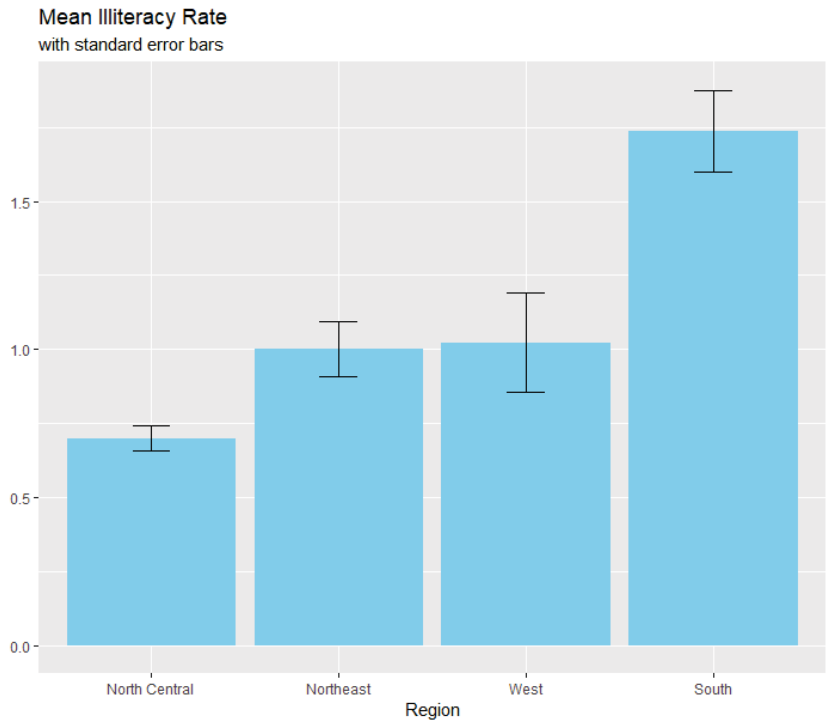


Figure 6.4 Bar chart of mean illiteracy rates for US regions sorted by rate. The standard error of the mean has been added to each bar.

6.1.4 Tweaking bar charts

There are several ways to tweak the appearance of a bar chart. The most common are customizing the bar colors and labels. We'll look at each in turn.

BAR CHART COLORS

Custom colors can be selected for the bar areas and borders. In the geom_bar() function the option `fill="color"` assigns a color for the area, while `color="color"` assigns a color for the border.

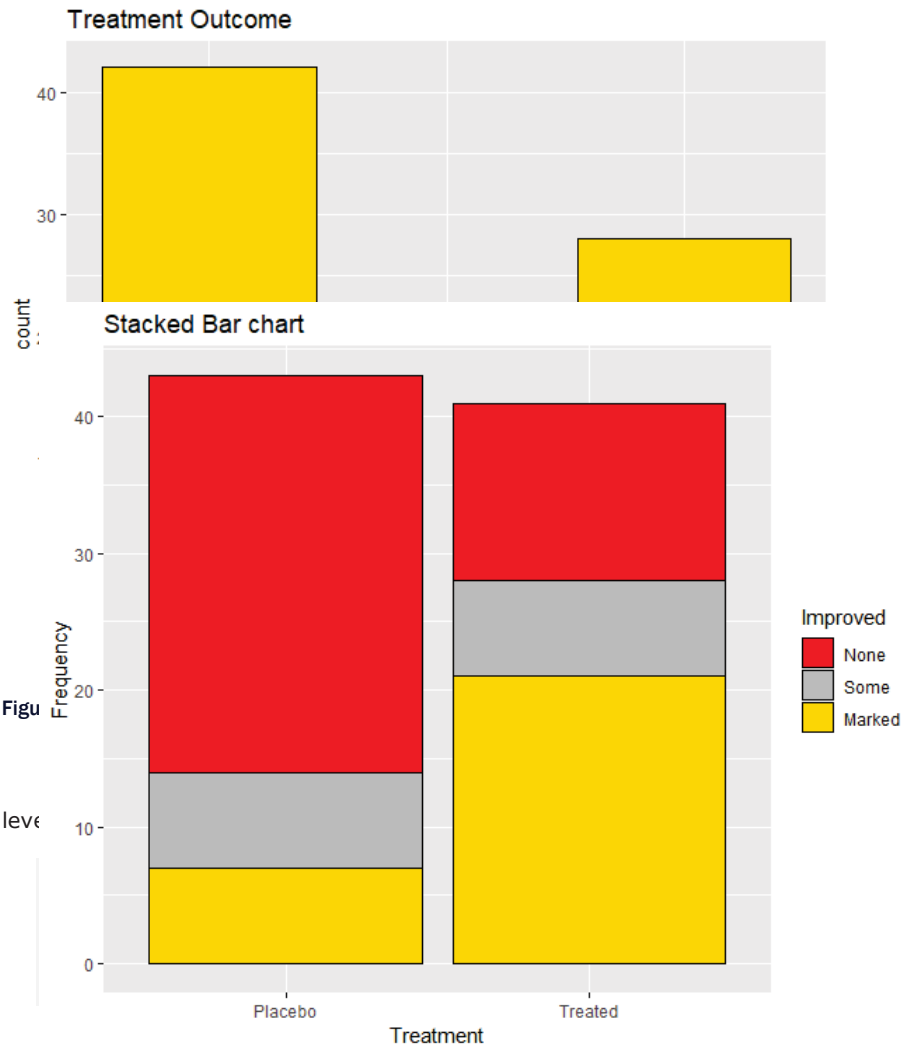
Fill vs. Color

In general, `ggplot2` uses *fill* to specify the color of geometric objects that have area (such as bars, pie slices, boxes), and the term *color* to when referring to the color of geometric objects without area (such as lines, points, and borders).

For example, the code

```
data(Arthritis, package="vcd")
ggplot(Arthritis, aes(x=Improved)) +
  geom_bar(fill="gold", color="black") +
  labs(title="Treatment Outcome")
```

produces the graph in figure 6.5.



Figu

ed to the

leve

produces the graph in figure 6.6.

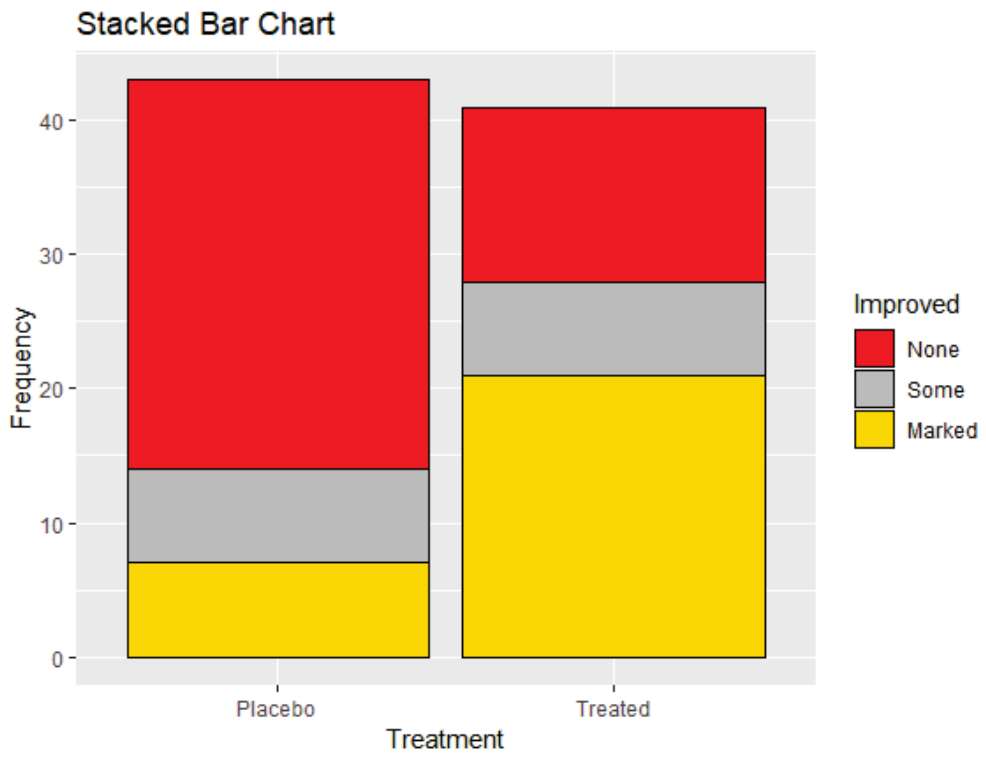


Figure 6.6 Stacked bar chart with custom fill colors mapped to Improvement

Here, bar fill colors are mapped to the levels of the variable `Improved`. The scale_fill_manual() function specifies red for `None`, grey for `Some`, and gold for `Marked` improvement. Color names can be obtained from http://www.stat.columbia.edu/~tzheng/files/Rcolor.pdf . Other methods of selecting colors are discussed in chapter 19 (Advanced Graphics with ggplot2).

BAR CHART LABELS

When there are many bars or long labels, bar chart labels tend to overlap and become unreadable. Consider the following example. The dataset `mpg` in the `ggplot2` package describes fuel economy data from for 38 popular car models in 1999 and 2008. Each model has several configurations (transmission type, number of cylinders, etc.). Let's say that we want a count of how many instances of each model are in the dataset. The code

```
ggplot(mpg, aes(x=model)) +
  geom_bar() +
  labs(title="Car models in the mpg dataset",
```

```
    y="Frequency", x="")
```

produces the graph in figure 6.7.

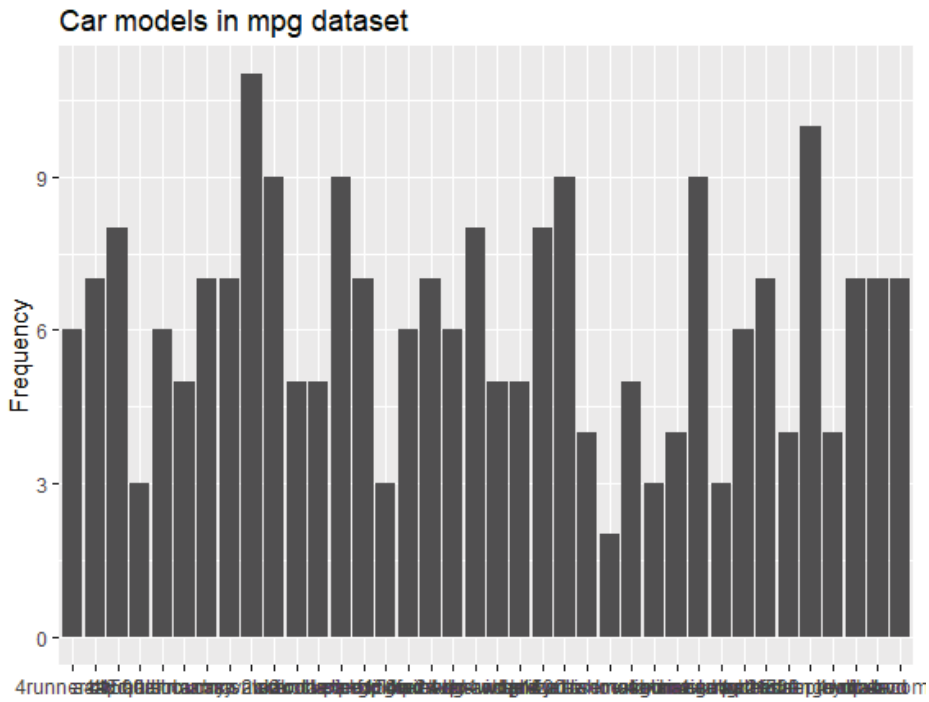


Figure 6.7 Bar chart with overlapping labels

Even with my glasses (or a glass of wine), I can't read this. Two simple tweaks will make the labels readable. First, we can plot the data as a horizontal bar chart.

```
ggplot(mpg, aes(x=model)) +
  geom_bar() +
  labs(title="Car models in the mpg dataset",
      y="Frequency", x="") +
  coord_flip()
```

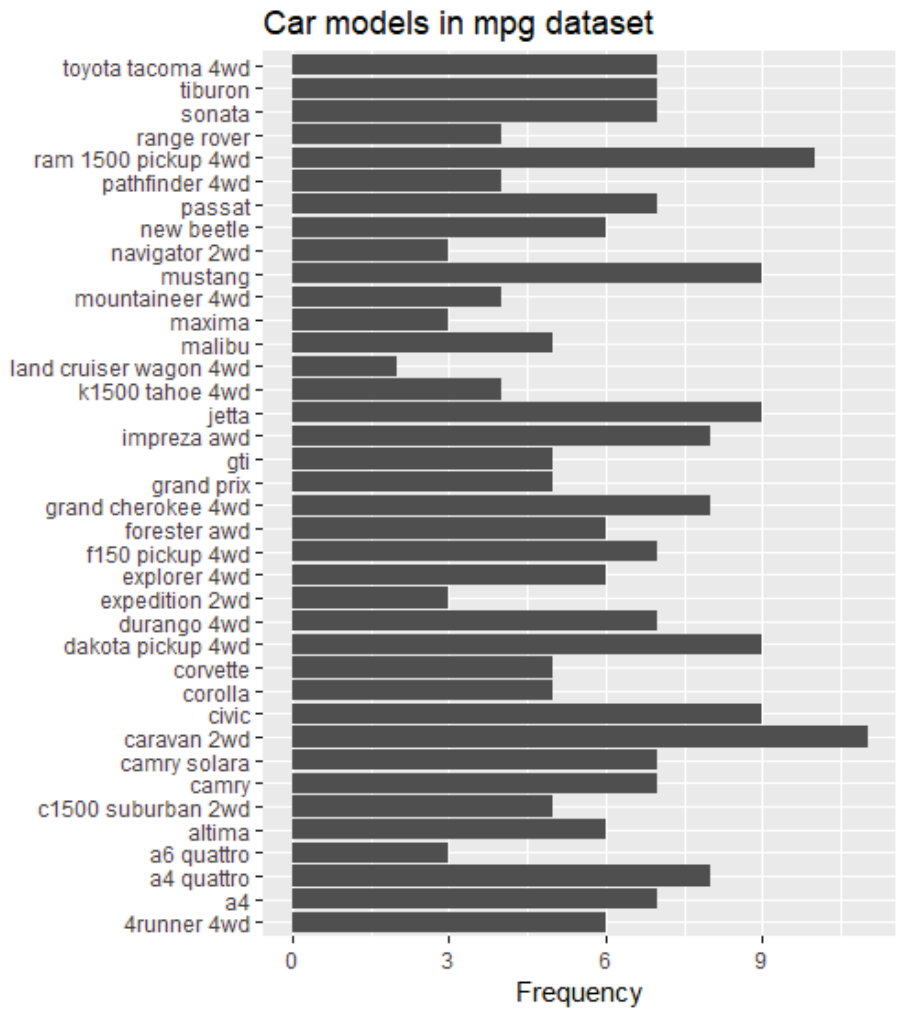


Figure 6.8 A horizontal bar chart avoids label overlap.

Second, we can angle the label text and use a smaller font.

```
ggplot(mpg, aes(x=model)) +
  geom_bar() +
  labs(title="Model names in the mpg dataset",
      y="Frequency", x="") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1, size=8))
```

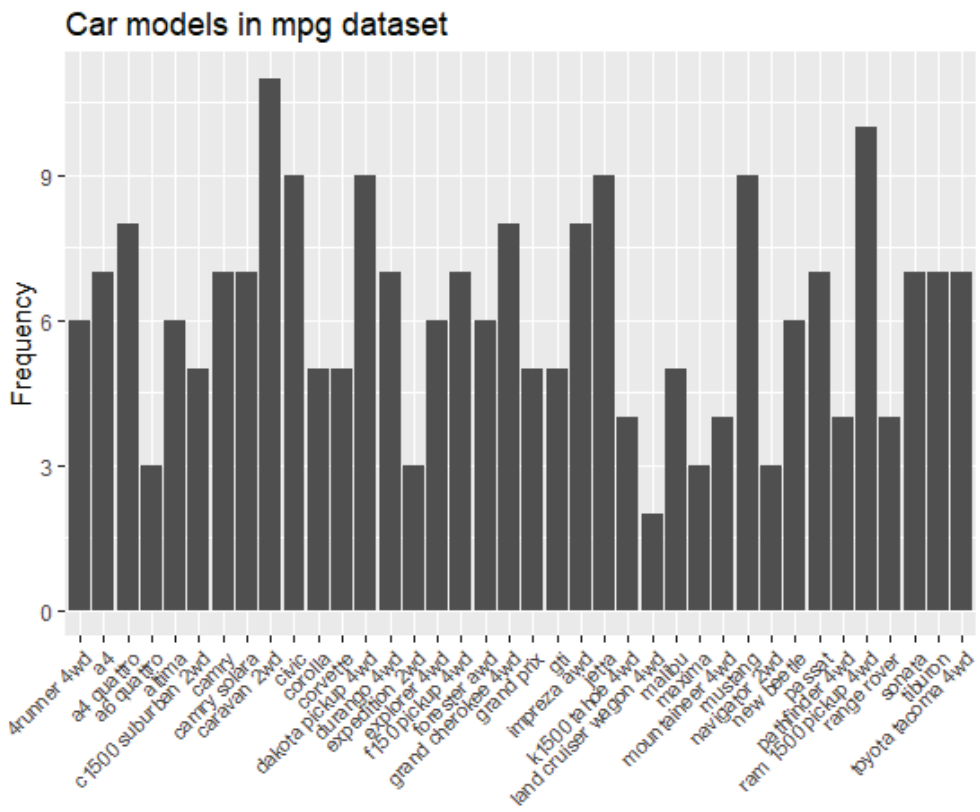


Figure 6.9 Bar chart with angled labels and a smaller label font.

The `theme()` function is discussed more fully in chapter 19 (Advanced Graphics with ggplot2). In addition to bar charts, pie charts are a popular vehicle for displaying the distribution of a categorical variable. We'll consider them next.

6.2 Pie charts

Pie charts are ubiquitous in the business world, but they're denigrated by most statisticians, including the authors of the R documentation. They recommend bar or dot plots over pie charts because people are able to judge length more accurately than volume. Perhaps for this reason, the pie chart options in R are severely limited when compared with other statistical platforms.

However, there are times when pie charts can be useful. In particular, they can capture part-whole relationships well. For example, a pie chart can be used to display the percentage of tenured faculty at a university who are female.

You can create a pie chart in base R using the `pie()` function, but as I've said, the functionality is limited and the plots are unattractive. To address this, I've created a packaged called `ggpie` that allows you to create a wide variety of pie charts using ggplot2 (no flame emails please!). You can install it with the following code.

```
if(!require(devtools) install.packages("devtools")
devtools::install_github("rkabacoff/ggpie")
```

The basic syntax is

```
ggpie(data, x, by, offset, percent, legend, title)
```

where

- `data` is a data frame
- `x` is the categorical variable to be plotted
- `by` is an optional second categorical variable. If present, a pie will be produced for each level of this variable.
- `offset` is the distance of the pie slice labels from the origin. A value of 0.5 will place the labels in the center of the slices, and a value greater than 1.0 will place them outside the slice.
- `percent` is logical. If `FALSE`, percentage printing is suppressed.
- `legend` is logical. If FALSE, the legend is omitted and each pie slice is labeled.
- `title` is a an option title.

Additional options (described on the `ggpie` website) allow you to customize the pie chart's appearance.

Let's create a pie chart displaying the distribution of car classes in the `mpg` data frame.

```
library(ggplot2)
library(ggpie)
ggpie(mpg, class)
```

The results are given in figure 6.10. From the graph, we see that 26% percent of cars are SUVS, while only 2% are two-seaters.

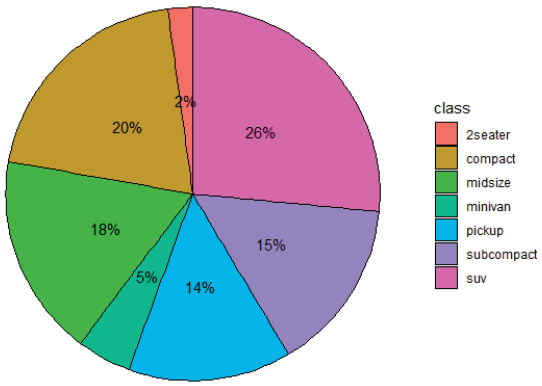


Figure 6.10. Pie chart displaying the percentage of each car class in the mpg data frame.

In the next version, the legend is removed and each pie slice is labeled. In addition, the labels are place outside the pie area, and title is added.

```
ggpie(mpg, class, legend=FALSE, offset=1.3,
      title="Automobiles by Car Class")
```

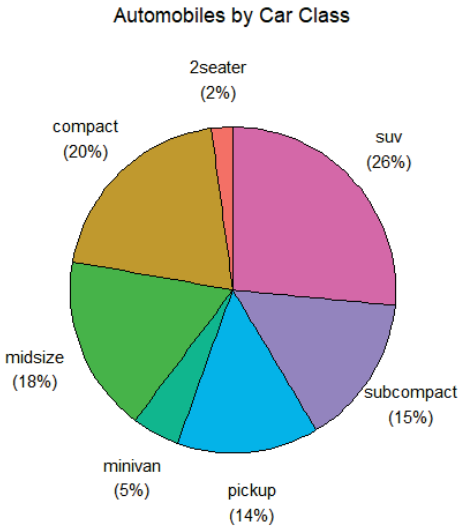


Figure 6.11. Pie chart with labels displayed outside the pie.

In the final example, the distribution of car class is displayed by year.

```
ggpie(mpg, class, year,
    legend=FALSE, offset=1.3, title="Car Class by Year")
```

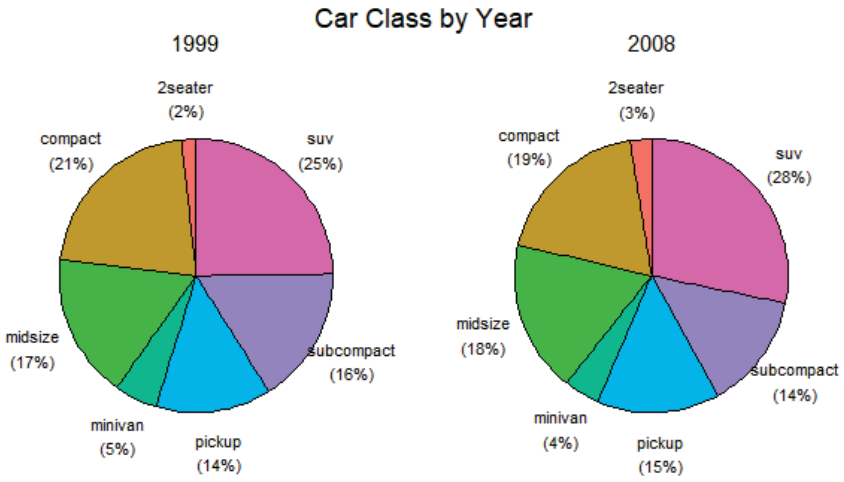


Figure 6.12. Pie charts displaying the distribution of car classes by year

Between 1999 and 2008, the distribution of car classes appears to have remained rather constant. The `ggpie` package can create more complex and customized pie charts. See the documentation (http://rkabacoff.github.io/ggpie) for details.

6.3 Tree maps

An alternative to a pie chart is a tree map. A tree map displays the distribution of a categorical variable using rectangles that are proportional to variable levels. Unlike pie charts, tree maps can handle categorical variables with *many* levels. We'll create tree maps using the `treemapify` package. Be sure to install it before proceeding (`install.packages("treemapify")`).

We'll start by creating a tree map displaying the distribution of car manufacturers in the `mpg` data frame. The code is given in listing 6.6. The resulting graph is giving fin figure 6-12.

Listing 6.6 Simple Tree Map

```
library(ggplot2)
library(dplyr)
library(treemapify)
```

```
plotdata <- mpg %>% count(manufacturer) #1

ggplot(plotdata,                    #2
    aes(fill = manufacturer,
        area = n,
        label = manufacturer)) +
geom_treemap() +
geom_tree_text() +
theme(legend.position = FALSE)
```

#1 Summarize the data
#2 Create the tree map

First we calculate the frequency counts for each level the `manufacturer` variable #1. This information is passed to `ggplot2` to create the graph #2. In the `aes()` function, `fill` refers to the categorical variable, `area` is the count for level, and `label` is the option variable used to label the cells. The `geom_treemap()` function creates the tree map and the `geom_tree_text()` function adds the labels to each cell. The `theme()` function is used to suppress the legend, which is redundant here, since each cell is labeled.

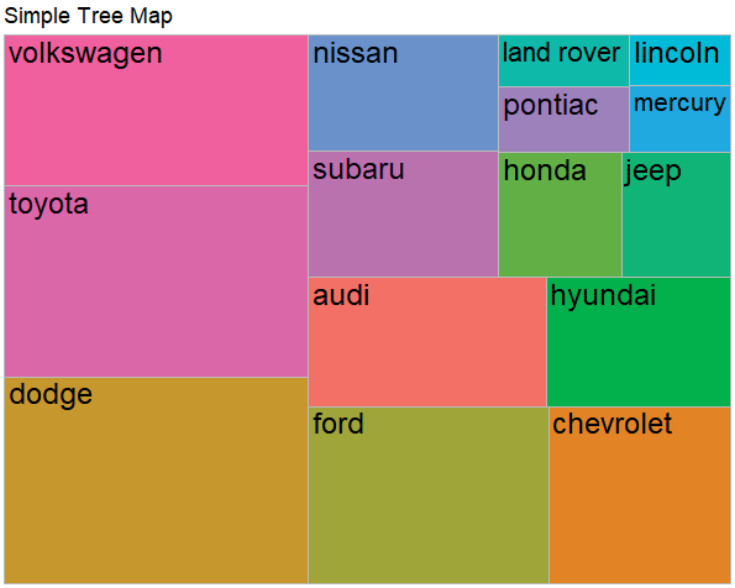


Figure 6.13. Tree map displaying the distribution of car manufacturers in the mpg data set. Rectangle size is proportional to the number of cars from each manufacturer.

In the next example, a second variable is added – drivetrain. The number of cars by manufacturer is plotted for front-wheel, rear-wheel, and four-wheel drives. The code is provided in listing 6.7 and the plot is displayed in figure 6.14.

Listing 6.7 Tree Map with Subgrouping

```
plotdata <- mpg %>%                                    #1
 count(manufacturer, drv)
  plotdata$drv <- factor(plotdata$drv,                          #2
                levels=c("4", "f", "r"),
                labels=c("4-wheel", "front-wheel", "rear"))

ggplot(plotdata,                                       #3
      aes(fill = manufacturer,
          area = n,
          label = manufacturer,
          subgroup=drv)) +
 geom_treemap() +
 geom_treemap_subgroup_border() +
 geom_treemap_subgroup_text(
   place = "middle",
   colour = "black",
   alpha = 0.5,
   grow = FALSE) +
 geom_treemap_text(colour = "white",
              place = "centre",
              grow=FALSE) +
 theme(legend.position = "none")
```

#1 Compute cell counts
#2 Provide better labels for drivetrains
#2 Create tree map

First, the frequencies for each manufacturer-drivetrain combination is calculated #1. Next, better labels are provided for the drivetrain variable #2. The new data frame is passed to `ggplot2` to produce the tree map #3. The subgroup option in the `aes()` function creates separate subplots for each drivetrain type. The `geom_treemap_border()` and `geom_treemap_subgroup_text()` add borders and labels for the subgroups respectively. Options in each function control their appearance. The subgroup text is centered and given some transparency (`alpha=0.5`). The text font remains a constant size, rather than growing to fill the area (`grow=FALSE`). The tree map cell text is print in a white font, centered in each cell, and does not grow to fill the boxes.

From the graph in figure 6.14, it is clear for example, that Hyundai has front-wheel cars, but not rear-wheel or four-wheel cars. The manufacturers with rear-wheel cars are primarily Ford and Chevrolet. Many of the four-wheel cars are made by Dodge.

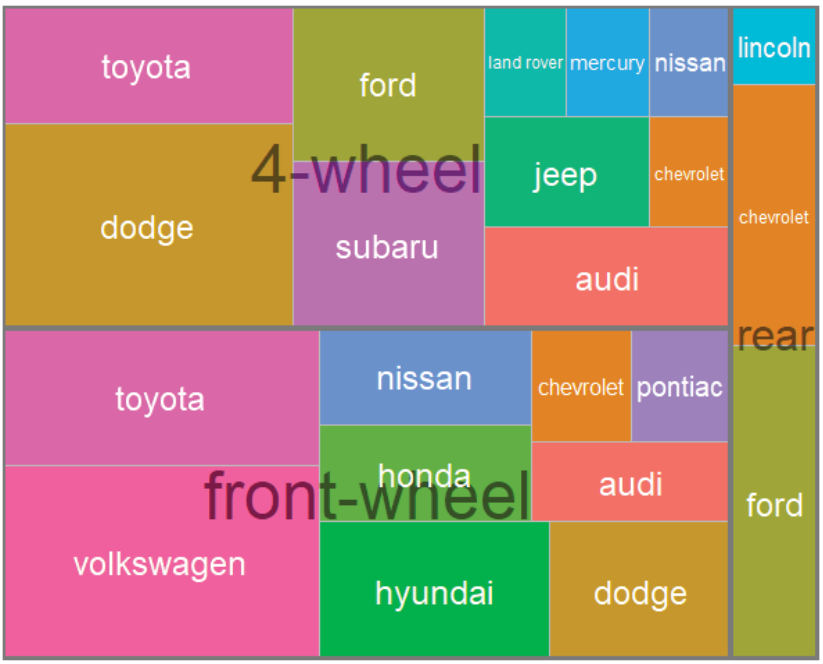


Figure 6.14 Tree map with car manufactures by drive-train type.

Now that we've covered pie charts and tree maps, let's move on to histograms. Unlike bar charts, pie charts, and tree maps, histograms describe the distribution of a continuous variable.

6.4 Histograms

Histograms display the distribution of a continuous variable by dividing the range of scores into a specified number of bins on the x-axis and displaying the frequency of scores in each bin on the y-axis. You can create histograms using

```
ggplot(data, aes(x = contvar)) + geom_histogram()
```

where *data* is a data frame and *contvar* is a continuous variable. Using the `mpg` data set in the ggplot package, we'll examine the distribution of city miles per gallon (`cty`) for 117 automobile configurations in 2008. Four variations of a histogram are created in lisiting 6.8 and the results graphs are presented in figure 6.15.

Listing 6.6 Histograms

```
library(ggplot2)
library(scales)
```

```
data(mpg)
cars2008 <- mpg[mpg$year == 2008, ]

ggplot(cars2008, aes(x=hwy)) +                  #1
  geom_histogram() +                          #1
  labs(title="Default histogram")              #1

ggplot(cars2008, aes(x=hwy)) +                              #2
  geom_histogram(bins=20, color="white", fill="steelblue") +    #2
  labs(title="Colored histogram with 20 bins",                #2
     x="City Miles Per Gallon",                             #2
     y="Frequency")


ggplot(cars2008, aes(x=hwy, y=..density..)) +                #3
  geom_histogram(bins=20, color="white", fill="steelblue") +    #3
  scale_y_continuous(labels=scales::percent) +                #3
 labs(title="Histogram with percentages",                     #3
     y= "Percent".                                          #3
     x="City Miles Per Gallon")                             #3

ggplot(cars2008, aes(x=hwy, y=..density..)) +                #4
  geom_histogram(bins=20, color="white", fill="steelblue") +    #4
  scale_y_continuous(labels=scales::percent) +                44
  geom_density(color="red", size=1) +                       #4
  labs(title="Histogram with density curve",                  #4
      y="Percent" ,                                         #4
     x="City Miles Per Gallon")                             #4
```

#1 Simple histogram
#2 Colored histogram with 20 bins
#3 Histogram with percentages
#4 Histogram with density curve

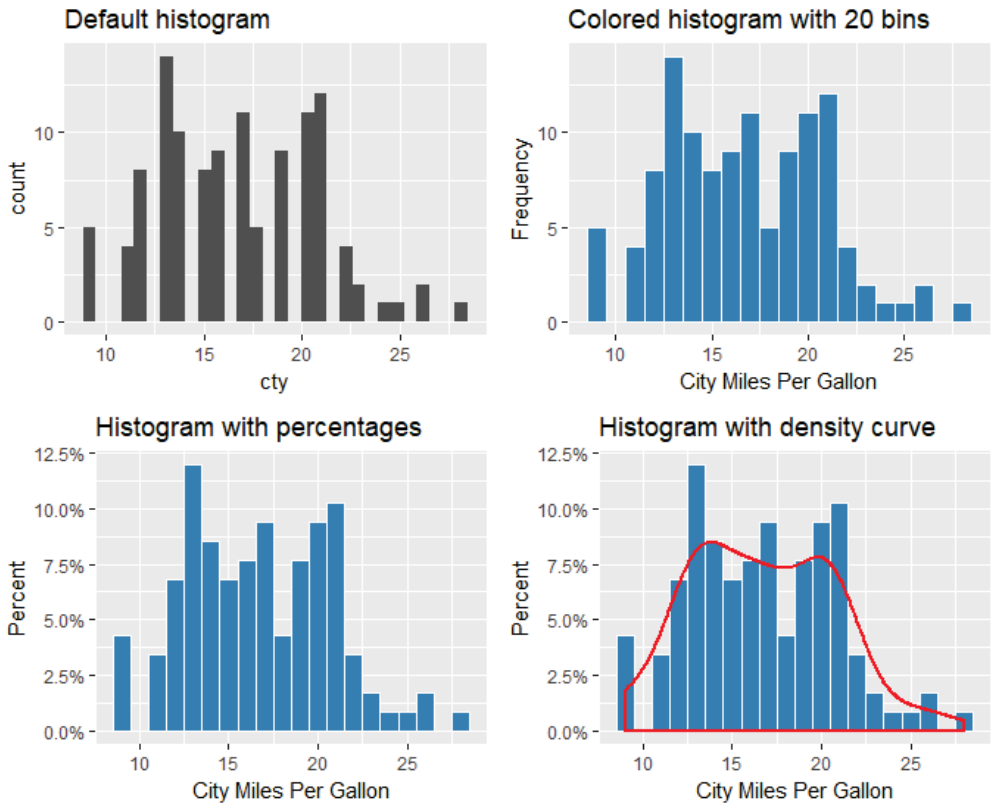


Figure 6.15 Histogram examples

The first histogram #1 demonstrates the default plot when no options are specified. In this case, 30 bins are created. For the second histogram #2, 20 bins, a steel blue fill, and a white border color are specified. In addition, more informative labels have been added. The number of bins can strongly influence the appearance of the histogram. It is a good idea to experiment with the `bins` value until you find one that captures the distribution well. With 20 bins, it appears that there are two peaks to the distribution – one around 13 mpg and one around 20.5 mpg.

The third histogram #3 plots the data as percents rather than frequencies. This is accomplished by assigning the built-in variable `..density..` to the y axis. The `scales` package is used to format the y-axis as percents. Be sure to install the package (`install.packages("scales")`) before running this part of the code.

The fourth histogram #4 is similar to the previous plot, but adds a density curve. The density curve is a kernel density estimate and is described in the next section. It provides a smoother description of the distribution of scores. The `geom_density()` function is used to plot

the kernel curve in a red color and a width that's slightly larger the default thickness for lines. The density curve also suggests a bimodal distribution (two peaks).

6.5 Kernel density plots

In the previous section, you saw a kernel density plot superimposed on a histogram. Technically, kernel density estimation is a nonparametric method for estimating the probability density function of a random variable. Basically, we're trying to draw a smoothed histogram, where the area under the curve equals one. Although the mathematics are beyond the scope of this text, density plots can be an effective way to view the distribution of a continuous variable. The format for a density plot is

```
ggplot(data, aes(x = contvar)) + geom_density()
```

where *data* is a data frame and *contvar* is a continuous variable. Again, let's plot the distribution of city miles per gallon (`cty`) for cars in 2008. Three kernel density examples are given in the next listing, and the results are provided in figure 6.16.

Listing 6.7 Kernel density plots

```
library(ggplot2)
data(mpg)
cars2008 <- mpg[mpg$year == 2008, ]

ggplot(cars2008, aes(x=cty)) +              #1
  geom_density() +                          #1
  labs(title="Default kernel density plot")   #1

ggplot(cars2008, aes(x=cty)) +              #2
  geom_density(fill="red") +                #2
  labs(title="Filled kernel density plot")    #2

> bw.nrd0(cars2008$cty)                     #3
1.408                                       #3

ggplot(cars2008, aes(x=cty)) +                    #4
  geom_density(fill="red", bw=.5) +               #4
  labs(title="Kernel density plot with bw=0.5")     #4
```

#1 Default density plot
#2 Filled density plot
#3 Print default bandwidth
#4 Density plot with smaller bandwidth

The default kernel density plot is given first #1. In the second example, the area under the curve is fill with red. The smoothness of the curve is control with a bandwidth parameter, which is calculated from the data being plotted. The code `bw.nrd0(cars2008$cty)` displays this value (1.408) #3. Using a larger bandwidth will give a smoother curve with less details. A smaller value will give a more squiggly curve (I don't squiggly an official term, but I couldn't

think of a better one). The third example uses a smaller bandwidth (`bw=`), allowing us to see more detail#4. As with the `bins` parameter for histograms, it is a good idea to try several bandwidth values to see which value helps you visualize the data most effectively.

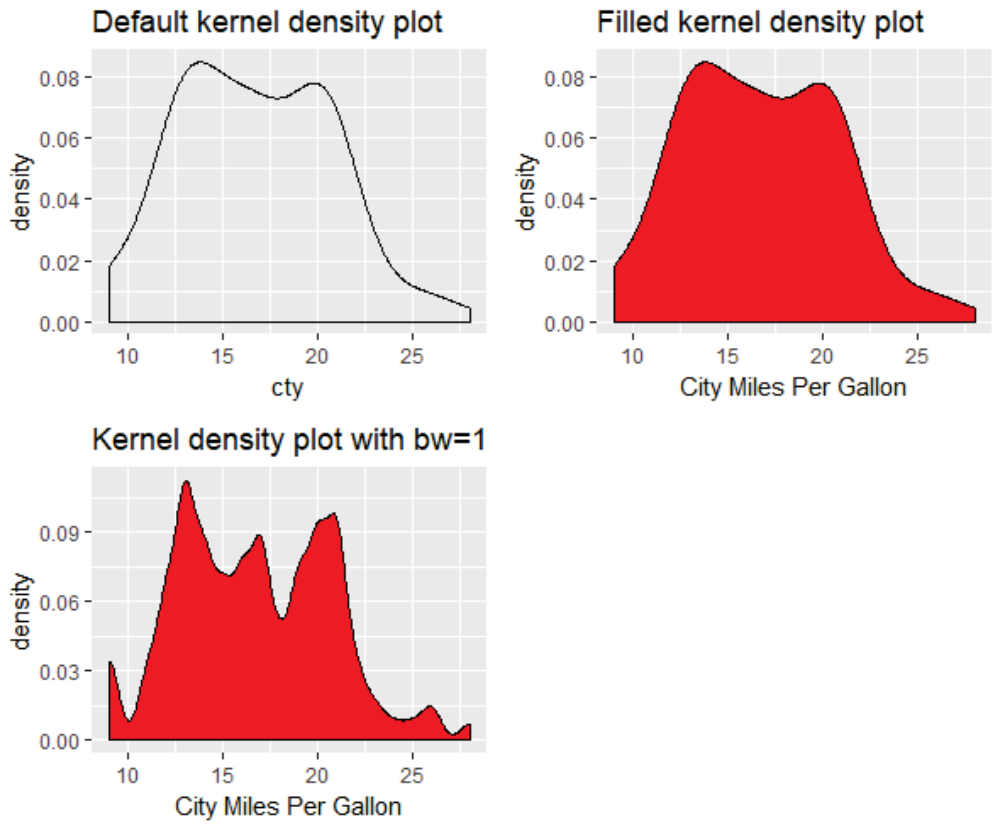


Figure 6.16 Kernel density plots

Kernel density plots can be used to compare groups. This is a highly underutilized approach, probably due to a general lack of easily accessible software. Fortunately, the `ggplot2` package fills this gap nicely.

For this example, we'll compare the 2008 city gas mileage estimates for 4-, 6-, and 8-cylinder cars. There are only a handful of cars with 5 cylinders so we will drop them from the analyses. The code is presented in listing 6.7. The resulting graphs are given in figures 6.17 and 6.18.

Listing 6.8 Comparative kernel density plots

```
data(mpg, package="ggplot2")                    #1
cars2008 <- mpg[mpg$year == 2008 & mpg$cyl != 5,]    #1
cars2008$Cylinders <- factor(cars2008$cyl)          #1

ggplot(cars2008, aes(x=cty, color=Cylinders, linetype=Cylinders)) +  #2
  geom_density(size=1)  +                           #2
  labs(title="Fuel Efficiecy by Number of Cylinders",          #2
       x = "City Miles per Gallon")                 #2

ggplot(cars2008, aes(x=cty, fill=Cylinders)) +
  geom_density(alpha=.4) +                           #3
  labs(title="Fuel Efficiecy by Number of Cylinders",          #3
       x = "City Miles per Gallon")                 #3
```

#1 Prepare the data
#2 Plots the density curves
#3 Plot filled density curves

First, a fresh copy of the data is loaded and 2008 data for cars with 4, 6, or 8 cylinders are retained #1. The number of cylinders (`cyl`) is saved as a categorical factor (`Cylinders`). The transformation is required because `ggplot2` expects the grouping variable to be categorical (and `cyl` is stored as a continuous variable).

A kernel density curve is plotted for each level of the Cylinders variable #2. Both the color (red, green, blue) and line type (solid, dotted, dashed) are mapped to the number of cylinders. Finally, the same plot is produced with filled curves #3. Transparency is added (`alpha=0.4`), since the filled curves overlap and we want to be able to see each one.

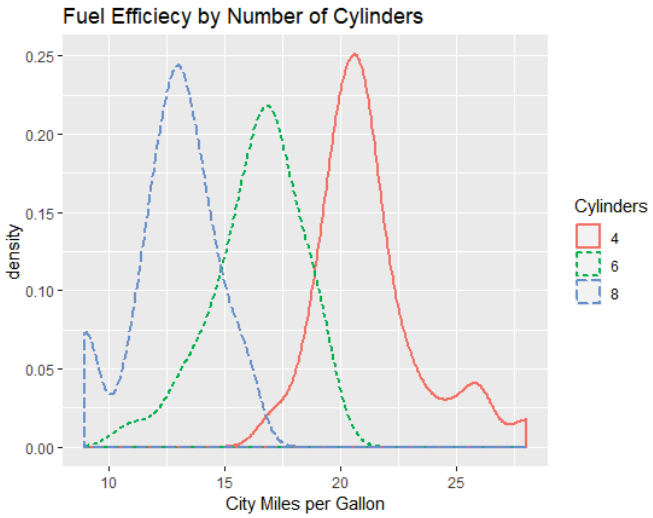


Figure 6.17 Kernel density curves of city mpg by number of cylinders

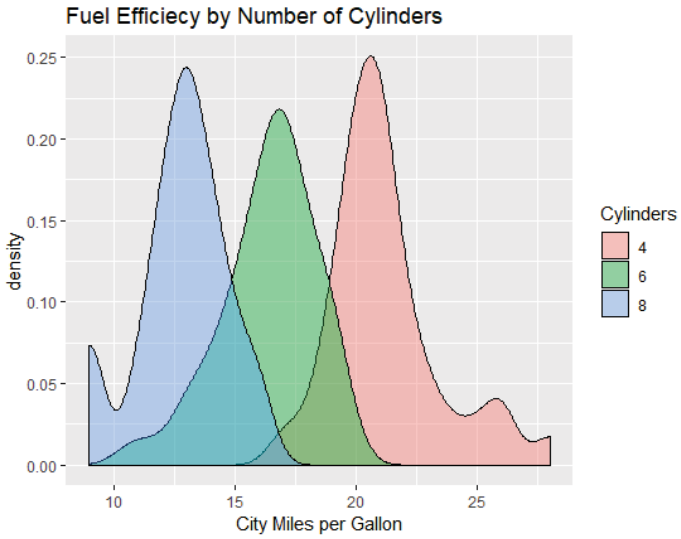


Figure 6.18 Filled kernel density curves of city mpg by number of cylinders.

Overlapping kernel density plots can be a powerful way to compare groups of observations on an outcome variable. Here you can see both the shapes of the distributions and the amount of overlap between groups. (The moral of the story is that my next car will have four cylinders—or a battery.)

Box plots are also a wonderful (and more commonly used) graphical approach to visualizing distributions and differences among groups. We'll discuss them next.

6.6 Box plots

A *box-and-whiskers plot* describes the distribution of a continuous variable by plotting its five-number summary: the minimum, lower quartile (25th percentile), median (50th percentile), upper quartile (75th percentile), and maximum. It can also display observations that may be outliers (values outside the range of ± 1.5*IQR, where IQR is the interquartile range defined as the upper quartile minus the lower quartile). For example, the following code produces the plot shown in figure 6.19:

```
ggplot(mtcars, aes(x="", y=mpg)) +
 geom_boxplot() +
 labs(y = "Miles Per Gallon", x="", title="Box Plot"))
```

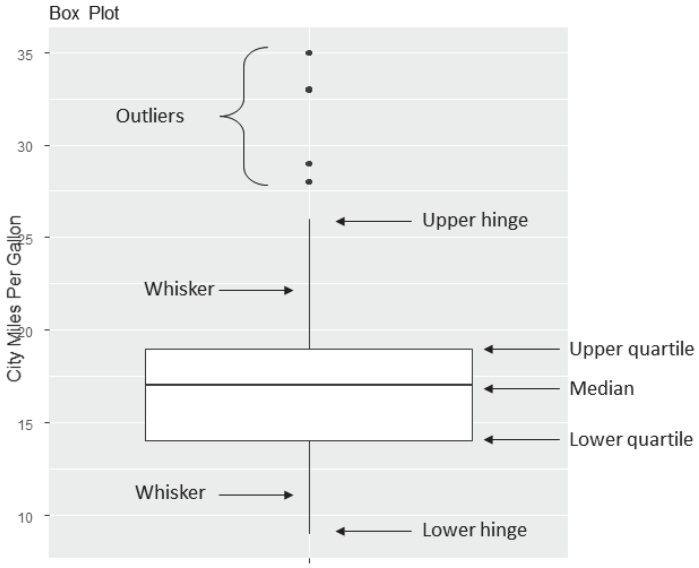


Figure 6.19 Box plot with annotations added by hand

I've added annotations by hand to illustrate the components. By default, each whisker extends to the most extreme data point, which is no more than 1.5 times the interquartile range for the box. Values outside this range are depicted as dots.

For example, in this sample of cars, the median mpg is 17, 50% of the scores fall between 14 and 19, the smallest value is 9, and the largest value is 35. How did I read this so precisely from the graph? Issuing `boxplot.stats(mtcars$mpg)` prints the statistics used to build the graph (in other words, I cheated). There is four outliers (greater than the upper hinge of 26). These values would be expected to occur less than 1% of the time in a normal distribution.

6.6.1 Using parallel box plots to compare groups

Box plots are a useful method of comparing the distribution of a quantitative variable across the levels of a categorical variable. Once again, let's compare city gas mileage for 3-, 6-, and 8-cylinder cars, but this time use both 1999 and 2008 data. Since there are only a few 5-cylinder cars, we will delete them. We'll also convert `year` and `cyl` from continuous numeric variables into categorical (grouping) factors.

```
library(ggplot2)
cars <- mpg[mpg$cyl != 5, ]
cars$Cylinders <- factor(cars$cyl)
cars$Year <- factor(cars$year)
```

```
The code
ggplot(cars, aes(x=Cylinders, y=cty)) +
 geom_boxplot() +
 labs(x="Number of Cylinders",
     y="Miles Per Gallon",
     title="Car Mileage Data")
```

produces the graph in figure 6.20. You can see that there's a good separation of groups based on gas mileage, with fuel efficiency dropping as the number of cylinders increases. There are also four outliers (cars with unusually high mileage) in the four-cylinder group.

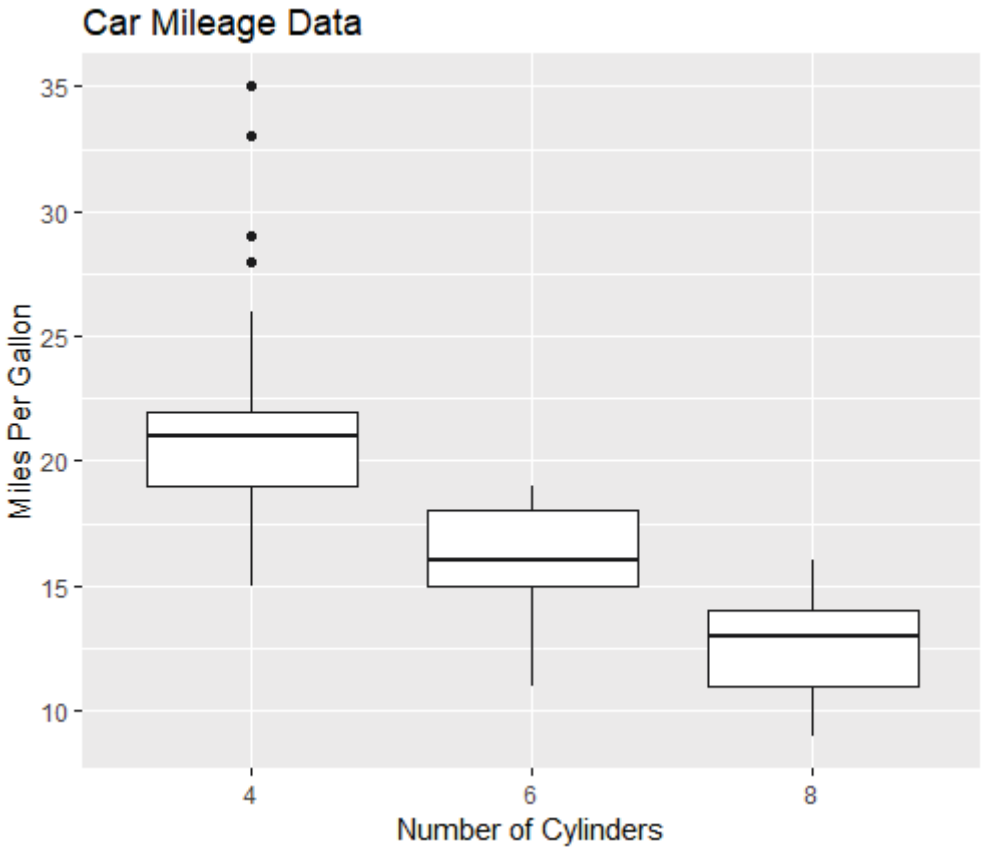


Figure 6.20 Box plots of car mileage vs. number of cylinders

Box plots are very versatile. By adding `notch=TRUE`, you get *notched* box plots. If two boxes' notches don't overlap, there's strong evidence that their medians differ (Chambers et al., 1983, p. 62). The following code creates notched box plots for the mileage example:

```
ggplot(cars, aes(x=Cylinder, y=cty)) +
 geom_boxplot(notch=TRUE,
              fill="steelblue",
              varwidth=TRUE) +
 labs(x="Number of Cylinders",
      y="Miles Per Gallon",
      title="Car Mileage Data")
```

The `fill` option fills the box plots with a red color. In a standard box plot, the box width has no meaning. Adding `varwidth=TRUE`, draws box widths proportional to the square roots of the number of observations in each group.

You can see in figure 6.21 that the median car mileage for four-, six-, and eight-cylinder cars differs. Mileage clearly decreases with number of cylinders. Additionally, there are fewer 8-cylinder cars, than 4- or 6-cylinder cars (although the difference is subtle).

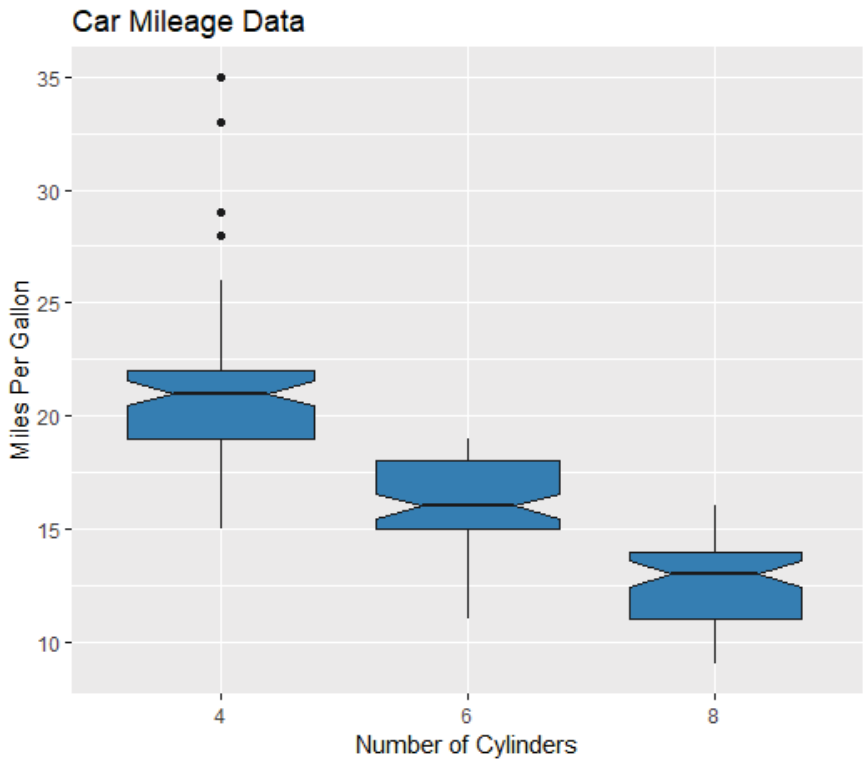


Figure 6.21 Notched box plots for car mileage vs. number of cylinders

Finally, you can produce box plots for more than one grouping factor. The following code provides box plots for city miles per gallon versus the number of cylinders by year (see figure 6.21). The `scale_fill_manual()` function has been added in order to customize the fill colors.

```
ggplot(cars, aes(x=Cylinders, y=cty, fill=Year)) +
 geom_boxplot() +
 labs(x="Number of Cylinders",
    y="Miles Per Gallon",
    title="City Mileage by # Cylinders and Year") +
 scale_fill_manual(values=c("gold", "green"))
```

From figure 6.22, it's again clear that median mileage decreases with number of cylinders. Additionally, for each group, mileage has increased between 1999 and 2008.

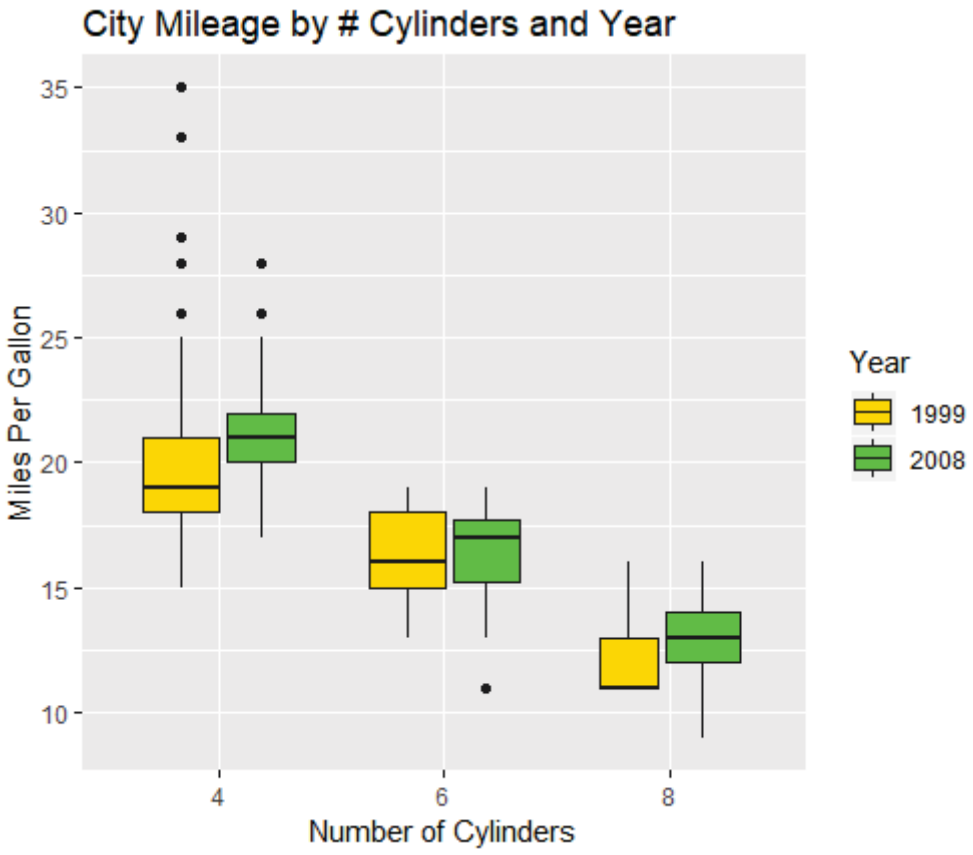


Figure 6.22 Box plots for car mileage vs. year and number of cylinders

6.6.2 Violin plots

Before we end our discussion of box plots, it's worth examining a variation called a *violin plot*. A violin plot is a combination of a box plot and a kernel density plot. You can create one using the `geom_violin()` function. In listing 6.9, we'll add violin plots to the box plots in figure 6.23.

Listing 6.9 Violin plots

```
library(ggplot2)
cars <- mpg[mpg$cyl != 5, ]
cars$Cylinders <- factor(cars$cyl)

ggplot(cars, aes(x=Cylinders, y=cty)) +
 geom_boxplot(width=0.2,
          fill="green") +
 geom_violin(fill="gold",
          alpha=0.3) +
 labs(x="Number of Cylinders",
     y="City Miles Per Gallon",
     title="Violin Plots of Miles Per Gallon")
```

The width of the box plots are set to 0.2 so that they will fit inside the violin plots. The violin plots are set with a transparency level of 0.3 so that the box plots are still visible.

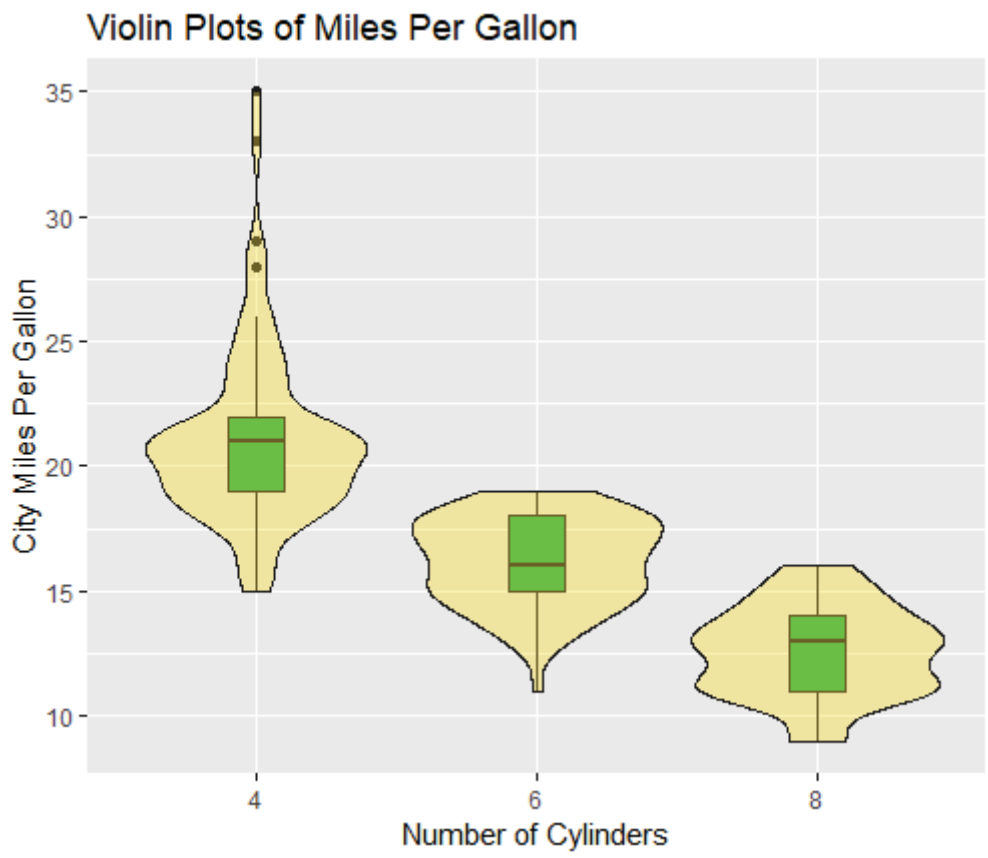


Figure 6.23 Violin plots of mpg vs. number of cylinders

Violin plots are basically kernel density plots superimposed in a mirror-image fashion over box plots. The middle lines are the medians, the black boxes range from the lower to the upper quartile, and the thin black lines represent the whiskers. Dots are outliers. The outer shape provides the kernel density plot. Here we can see that the distribution of gas mileage for 8-cylinder cars may be bimodal – a fact that is obscured by using box plots alone. Violin plots haven't really caught on yet. Again, this may be due to a lack of easily accessible software; time will tell.

We'll end this chapter with a look at dot plots. Unlike the graphs you've seen previously, dot plots plot every value for a variable.

6.7 Dot plots

Dot plots provide a method of plotting a large number of labeled values on a simple horizontal scale. You create them with the `dotchart()` function, using the format

```
ggplot(data, aes(x=contvar, y=catvar)) + geom_point()
```

where `data` is a data frame, *contvar* is a continuous variable, and *catvar* is a categorical variable. Here's an example using the highway gas mileage for the 2008 automobiles in the `mpg` dataset. Highway gas mileage is averaged by car model.

```
library(ggplot2)
library(dplyr)
plotdata <- mpg %>%
 filter(year == "2008") %>%
 group_by(model) %>%
 summarize(meanHwy=mean(hwy))

> plotdata

# A tibble: 38 x 2
  model            meanHwy
  <chr>              <dbl>
1 4runner 4wd         18.5
2 a4                  29.3
3 a4 quattro          26.2
4 a6 quattro          24
5 altima              29
6 c1500 suburban 2wd    18
7 camry               30
8 camry solara        29.7
9 caravan 2wd         22.2
10 civic              33.8
# ... with 28 more rows


ggplot(plotdata, aes(x=meanHwy, y=model)) +
 geom_point() +
 labs(x="Miles Per Gallon",
     y="",
     title="Gas Mileage for Car Models")
```

The resulting plot is given in figure 6.24.

This graph allows you to see the mpg for each car model on the same horizontal axis. Dot plots typically become most useful when they're sorted. The following code sorts the cars from lowest to highest mileage.

```
ggplot(plotdata, aes(x=meanHwy, y=reorder(model, meanHwy))) +
 geom_point() +
```

```
labs(x="Miles Per Gallon",
   y="",
   title="Gas Mileage for Car Models")
```

The resulting graph is given in figure 6.25. To plot in descending order, use `reorder(model, -meanHwy)`.

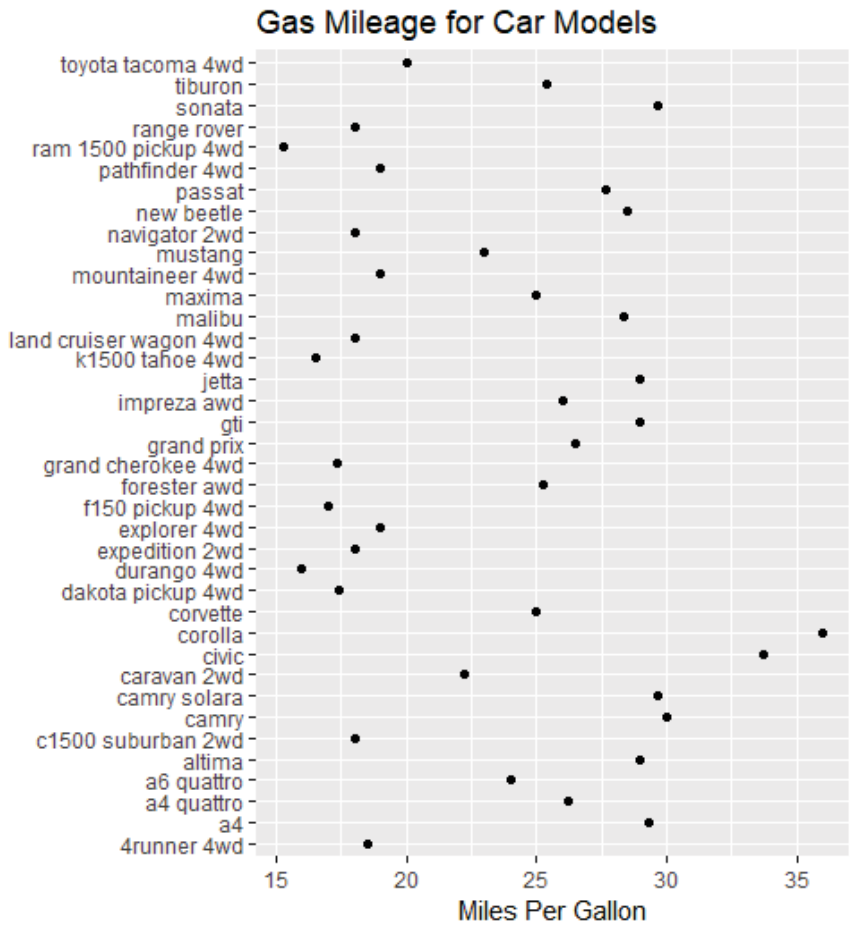


Figure 6.24 Dot plot of mpg for each car model

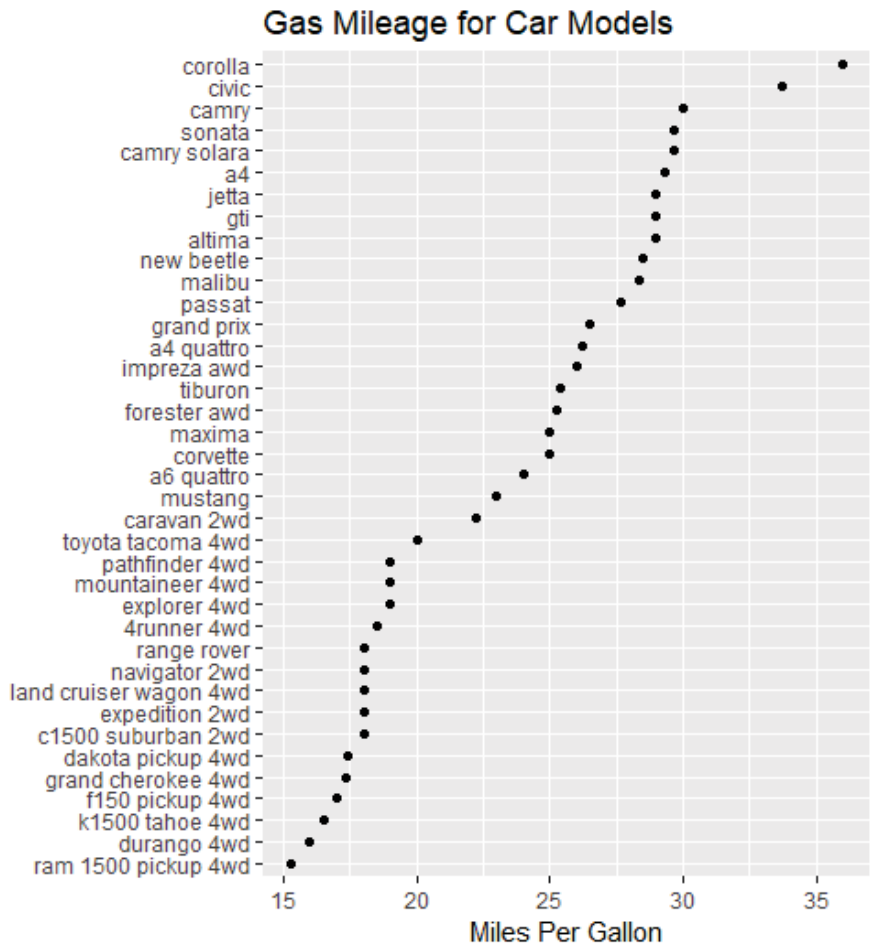


Figure 6.25 Dot plot of mpg for car models sorted by mileage

You can gain significant insight from the dot plot in this example because each point is labeled, the value of each point is inherently meaningful, and the points are arranged in a manner that promotes comparisons. But as the number of data points increases, the utility of the dot plot decreases.

6.8 Summary

- Bar charts (and to a lesser extent pie charts and tree maps) can be used to gain insight into the distribution of a categorical variable.
- Stacked, grouped, and filled bar charts can help you understand how groups differ on