Chapter 2

Introduction to ggplot2

This section provides an brief overview of how the ggplot2 package works. If you are simply seeking code to make a specific type of graph, feel free to skip this section. However, the material can help you understand how the pieces fit together.

2.1 A worked example

The functions in the ggplot2 package build up a graph in layers. We'll build a a complex graph by starting with a simple graph and adding additional elements, one at a time.

The example uses data from the 1985 Current Population Survey to explore the relationship between wages (*wage*) and experience (*expr*).

```
# load data
data(CPS85 , package = "mosaicData")
```

In building a ggplot2 graph, only the first two functions described below are required. The other functions are optional and can appear in any order.

2.1.1 ggplot

The first function in building a graph is the ggplot function. It specifies the

- data frame containing the data to be plotted
- the mapping of the variables to visual properties of the graph. The mappings are placed within the aes function (where *aes* stands for aesthetics).

```
# specify dataset and mapping
library(ggplot2)
ggplot(data = CPS85,
       mapping = aes(x = exper, y = wage))
```

Why is the graph empty? We specified that the *exper* variable should be mapped to the *x*-axis and that the *wage* should be mapped to the *y*-axis, but we haven't yet specified what we wanted placed on the graph.

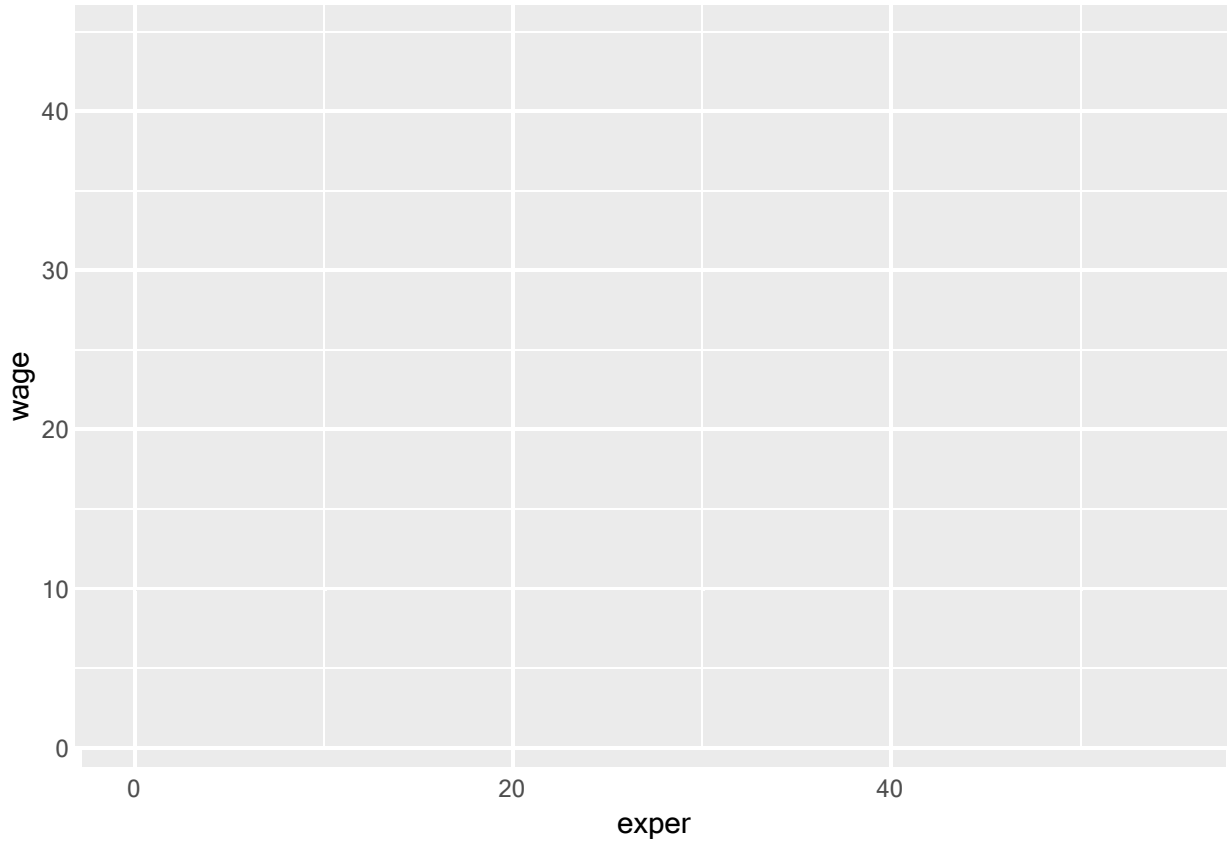


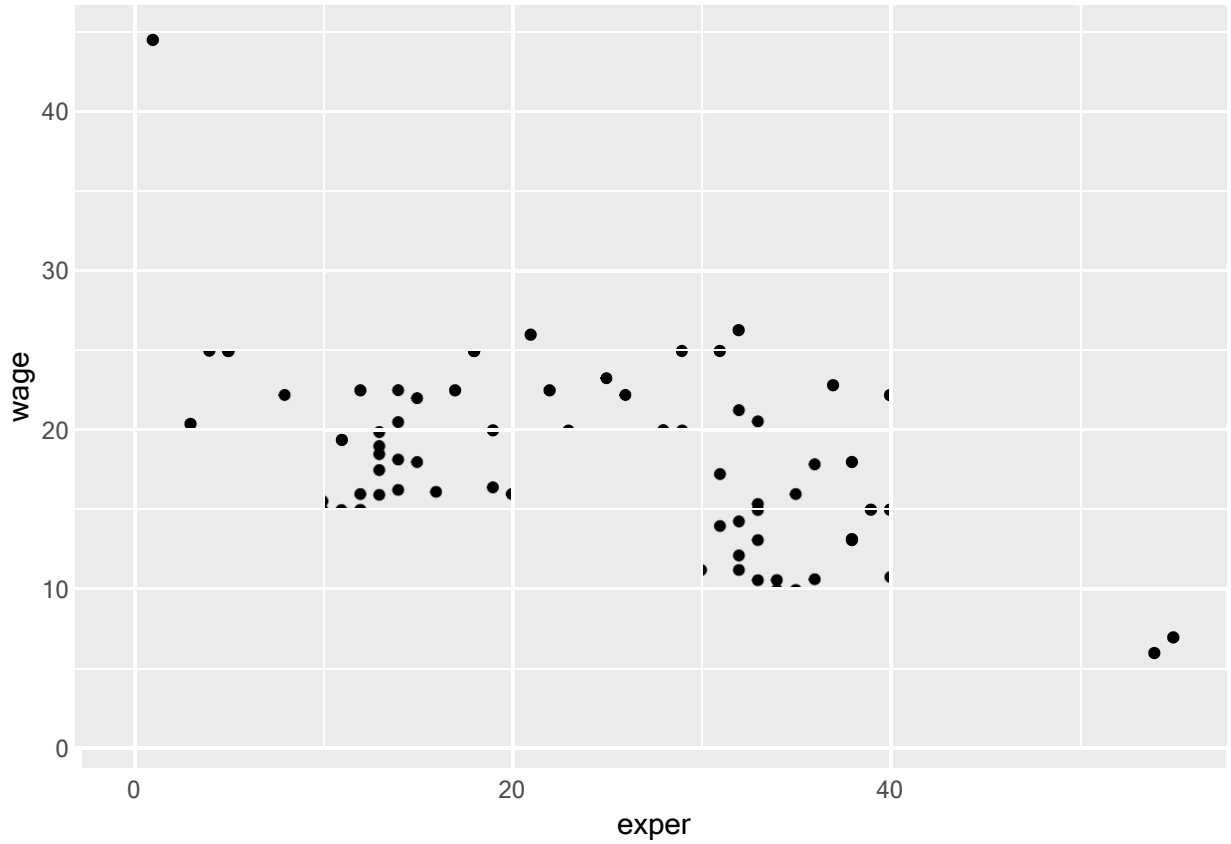
Figure 2.1: Map variables

2.1.2 geoms

Geoms are the geometric objects (points, lines, bars, etc.) that can be placed on a graph. They are added using functions that start with `geom_`. In this example, we'll add points using the `geom_point` function,creating a scatterplot.

In `ggplot2` graphs, functions are chained together using the `+` sign to build a final plot.

```
# add points
ggplot(data = CPS85,
       mapping = aes(x = exper, y = wage)) +
  geom_point()
```



The graph indicates that there is an outlier. One individual has a wage much higher than the rest. We'll delete this case before continuing.

```
# delete outlier
library(dplyr)
plotdata <- filter(CPS85, wage < 40)

# redraw scatterplot
ggplot(data = plotdata,
       mapping = aes(x = exper, y = wage)) +
  geom_point()
```

A number of parameters (options) can be specified in a `geom_` function. Options for the `geom_point` function include `color`, `size`, and `alpha`. These control the point color, size, and transparency, respectively. Trans-

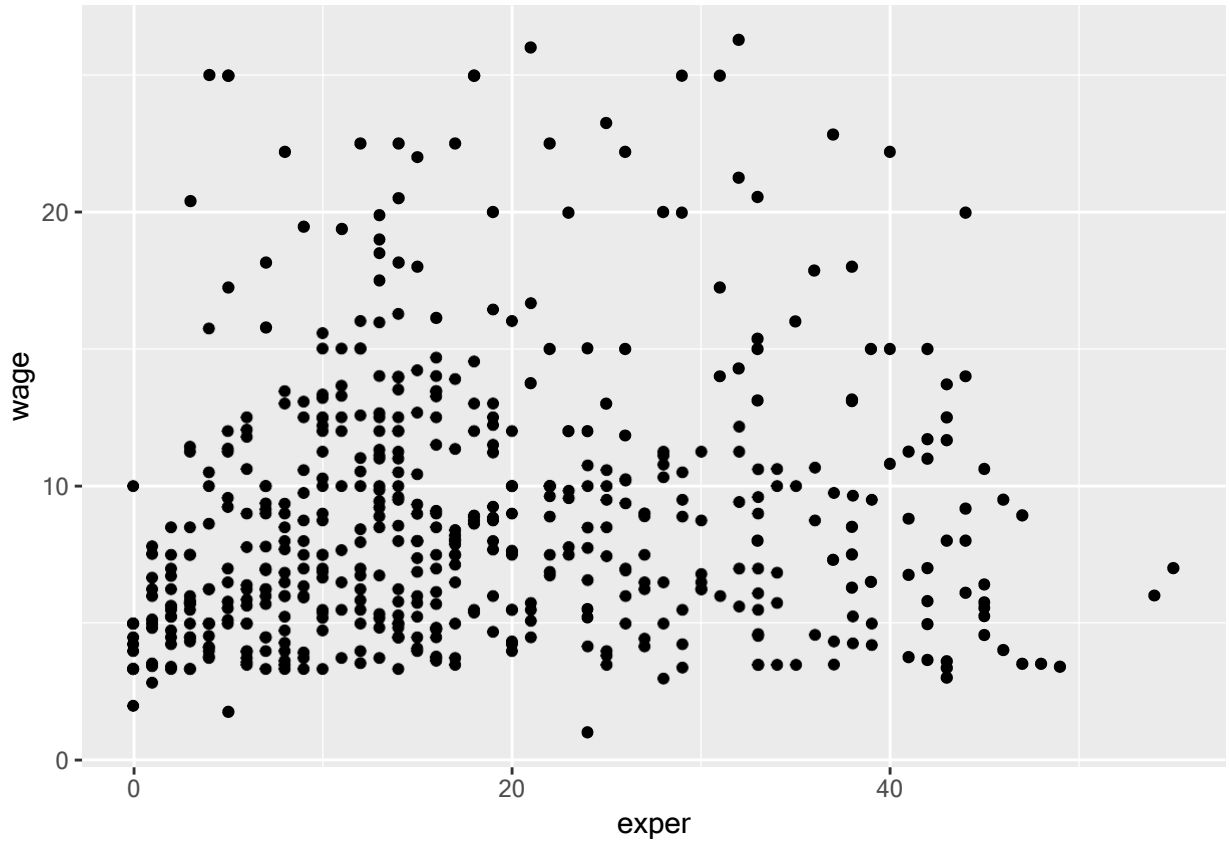


Figure 2.2: Remove outlier

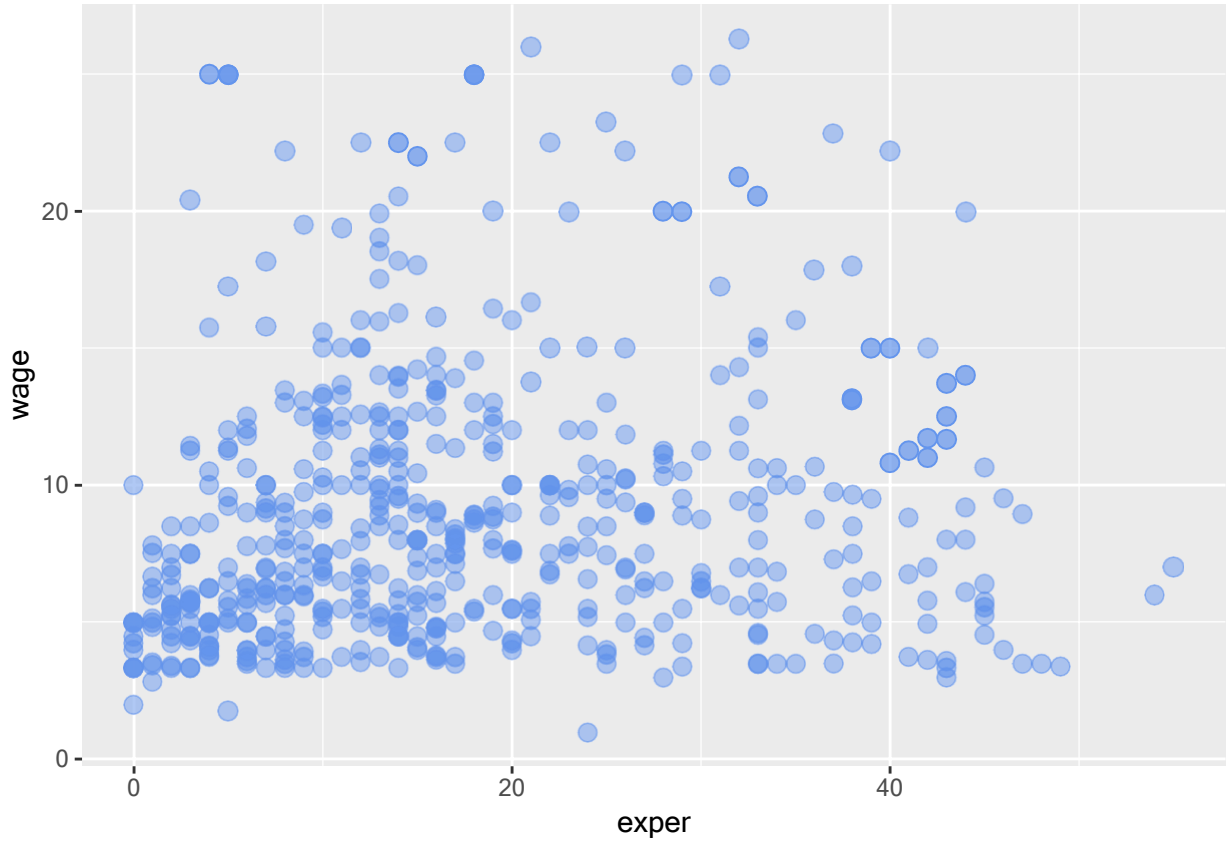


Figure 2.3: Modify point color, transparency, and size

parency ranges from 0 (completely transparent) to 1 (completely opaque). Adding a degree of transparency can help visualize overlapping points.

```
# make points blue, larger, and semi-transparent
ggplot(data = plotdata,
       mapping = aes(x = exper, y = wage)) +
  geom_point(color = "cornflowerblue",
             alpha = .7,
             size = 3)
```

Next, let's add a line of best fit. We can do this with the `geom_smooth` function. Options control the type of line (linear, quadratic, nonparametric), the thickness of the line, the line's color, and the presence or absence of a confidence interval. Here we request a linear regression (`method = lm`) line (where *lm* stands for linear model).

```
# add a line of best fit.
ggplot(data = plotdata,
       mapping = aes(x = exper, y = wage)) +
  geom_point(color = "cornflowerblue",
             alpha = .7,
             size = 3) +
  geom_smooth(method = "lm")
```

Wages appears to increase with experience.

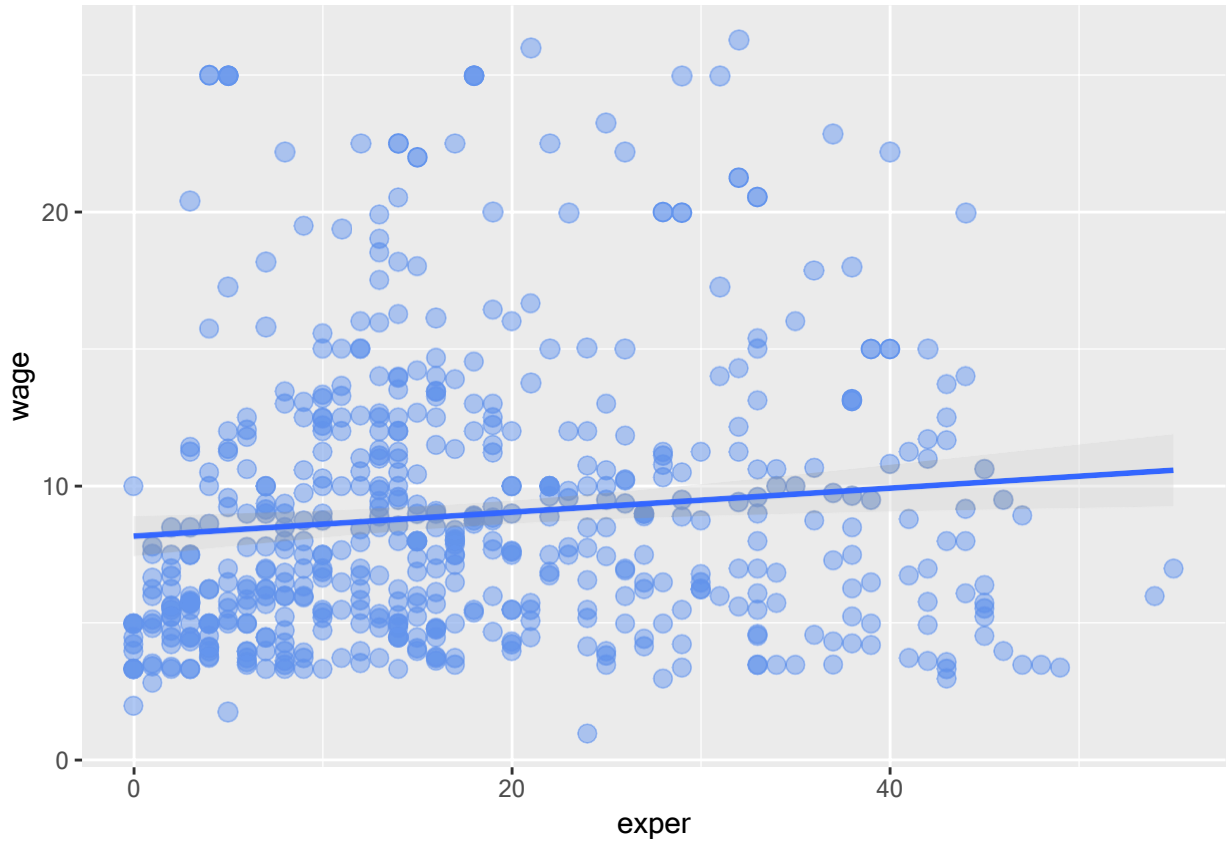


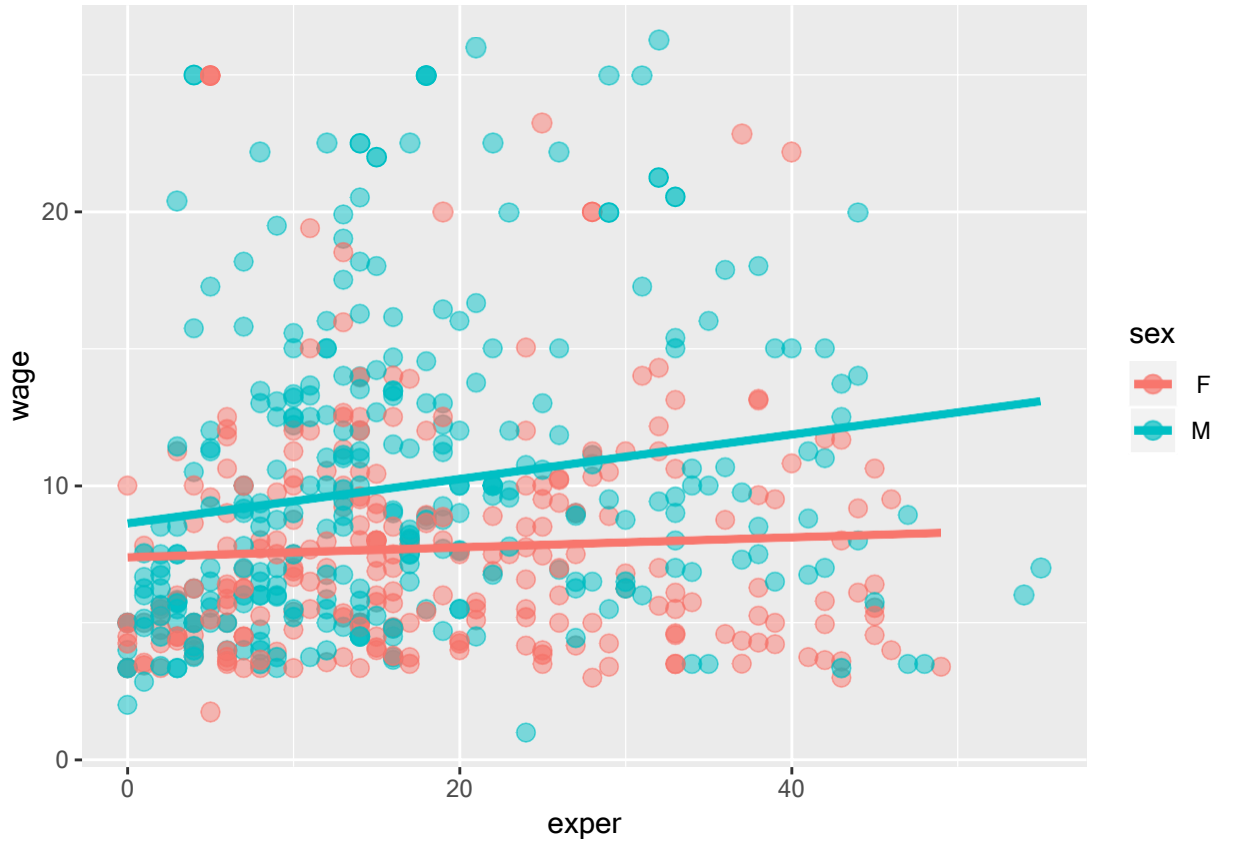
Figure 2.4: Add line of best fit

2.1.3 grouping

In addition to mapping variables to the x and y axes, variables can be mapped to the color, shape, size, transparency, and other visual characteristics of geometric objects. This allows groups of observations to be superimposed in a single graph.

Let's add sex to the plot and represent it by color.

```
# indicate sex using color
ggplot(data = plotdata,
       mapping = aes(x = exper,
                     y = wage,
                     color = sex)) +
  geom_point(alpha = .7,
             size = 3) +
  geom_smooth(method = "lm",
              se = FALSE,
              size = 1.5)
```



The `color = sex` option is placed in the `aes` function, because we are mapping a variable to an aesthetic. The `geom_smooth` option (`se = FALSE`) was added to suppresses the confidence intervals.

It appears that men tend to make more money than women. Additionally, there may be a stronger relationship between experience and wages for men than than for women.

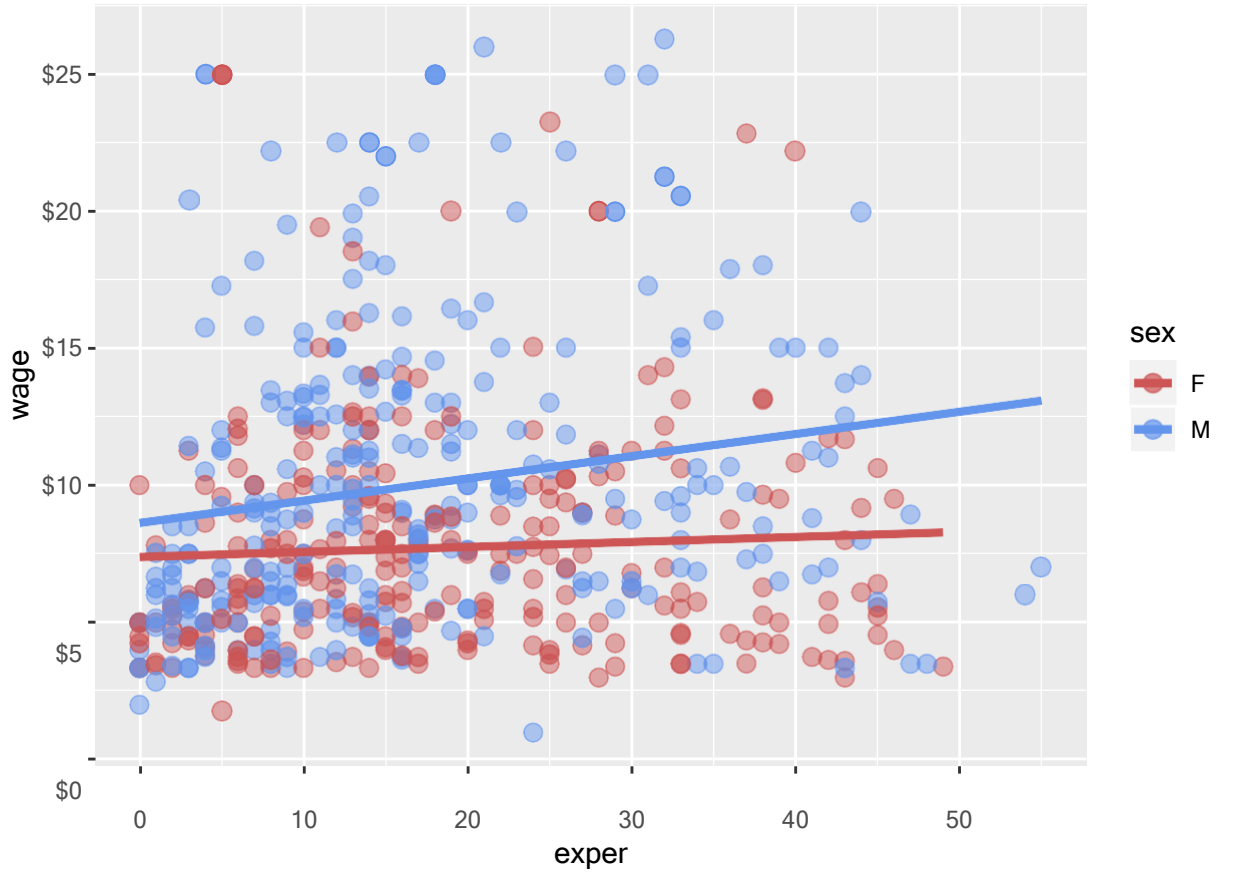


Figure 2.5: Change colors and axis labels

2.1.4 scales

Scales control how variables are mapped to the visual characteristics of the plot. Scale functions (which start with `scale_`) allow you to modify this mapping. In the next plot, we'll change the x and y axis scaling, and the colors employed.

```
# modify the x and y axes and specify the colors to be used
ggplot(data = plotdata,
       mapping = aes(x = exper,
                     y = wage,
                     color = sex)) +
  geom_point(alpha = .7,
             size = 3) +
  geom_smooth(method = "lm",
              se = FALSE,
              size = 1.5) +
  scale_x_continuous(breaks = seq(0, 60, 10)) +
  scale_y_continuous(breaks = seq(0, 30, 5),
                     label = scales::dollar) +
  scale_color_manual(values = c("indianred3",
                                "cornflowerblue"))
```

We're getting there. The numbers on the x and y axes are better, the y axis uses dollar notation, and the

colors are more attractive (IMHO).

Here is a question. Is the relationship between experience, wages and sex the same for each job sector? Let's repeat this graph once for each job sector in order to explore this.

2.1.5 facets

Facets reproduce a graph for each level a given variable (or combination of variables). Facets are created using functions that start with `facet_`. Here, facets will be defined by the eight levels of the *sector* variable.

```
# reproduce plot for each level of job sector
ggplot(data = plotdata,
       mapping = aes(x = exper,
                     y = wage,
                     color = sex)) +
  geom_point(alpha = .7) +
  geom_smooth(method = "lm",
              se = FALSE) +
  scale_x_continuous(breaks = seq(0, 60, 10)) +
  scale_y_continuous(breaks = seq(0, 30, 5),
                     label = scales::dollar) +
  scale_color_manual(values = c("indianred3",
                                "cornflowerblue")) +
  facet_wrap(~sector)
```

It appears that the differences between mean and women depend on the job sector under consideration.

2.1.6 labels

Graphs should be easy to interpret and informative labels are a key element in achieving this goal. The `labs` function provides customized labels for the axes and legends. Additionally, a custom title, subtitle, and caption can be added.

```
# add informative labels
ggplot(data = plotdata,
       mapping = aes(x = exper,
                     y = wage,
                     color = sex)) +
  geom_point(alpha = .7) +
  geom_smooth(method = "lm",
              se = FALSE) +
  scale_x_continuous(breaks = seq(0, 60, 10)) +
  scale_y_continuous(breaks = seq(0, 30, 5),
                     label = scales::dollar) +
  scale_color_manual(values = c("indianred3",
                                "cornflowerblue")) +
  facet_wrap(~sector) +
  labs(title = "Relationship between wages and experience",
       subtitle = "Current Population Survey",
       caption = "source: http://mosaic-web.org/",
       x = " Years of Experience",
       y = "Hourly Wage",
       color = "Gender")
```

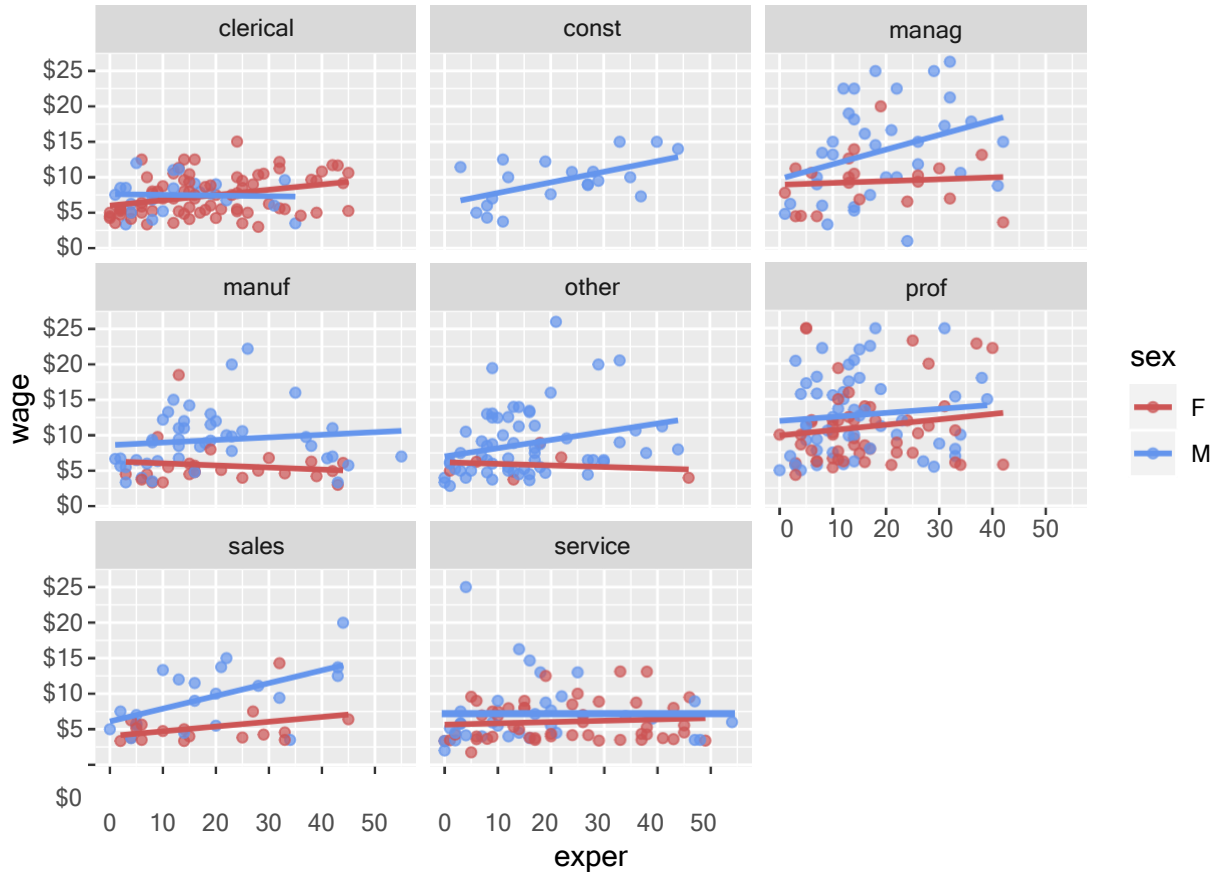


Figure 2.6: Add job sector, using faceting

Now a viewer doesn't need to guess what the labels *expr* and *wage* mean, or where the data come from.

2.1.7 themes

Finally, we can fine tune the appearance of the graph using themes. Theme functions (which start with `theme_`) control background colors, fonts, grid-lines, legend placement, and other non-data related featuresof the graph. Let's use a cleaner theme.

```
# use a minimalist theme
ggplot(data = plotdata,
       mapping = aes(x = exper,
                     y = wage,
                     color = sex)) +
  geom_point(alpha = .6) +
  geom_smooth(method = "lm",
              se = FALSE) +
  scale_x_continuous(breaks = seq(0, 60, 10)) +
  scale_y_continuous(breaks = seq(0, 30, 5),
                     label = scales::dollar) +
  scale_color_manual(values = c("indianred3",
                                "cornflowerblue")) +
  facet_wrap(~sector) +
  labs(title = "Relationship between wages and experience",
       subtitle = "Current Population Survey",
       caption = "source: http://mosaic-web.org/",
       x = " Years of Experience",
```

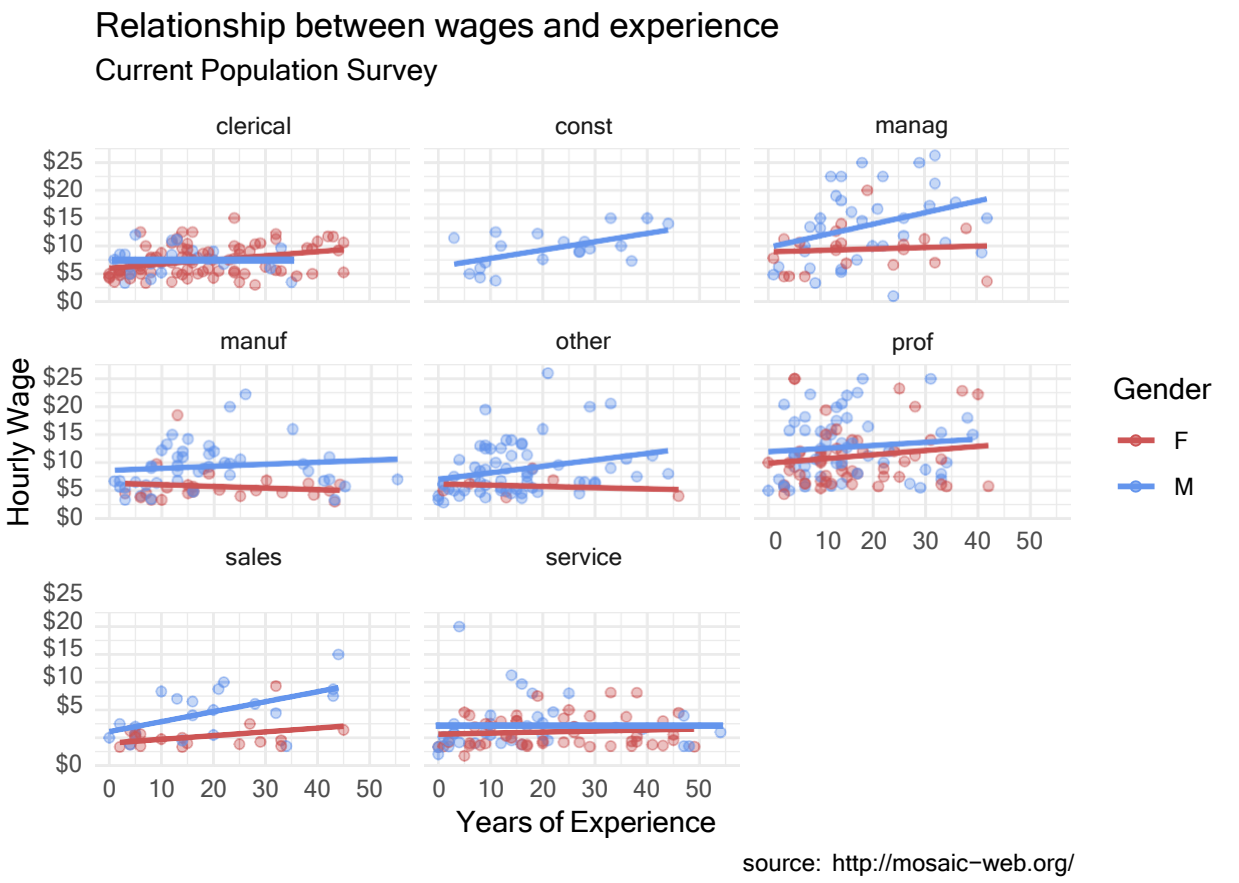


Figure 2.7: Use a simpler theme

```
      y = "Hourly Wage",
      color = "Gender") +
theme_minimal()
```

Now we have something. It appears that men earn more than women in management, manufacturing, sales, and the "other" category. They are most similar in clerical, professional, and service positions. The data contain no women in the construction sector. For management positions, wages appear to be related to experience for men, but not for women (this may be the most interesting finding). This also appears to betrue for sales.

Of course, these findings are tentative. They are based on a limited sample size and do not involve statistical testing to assess whether differences may be due to chance variation.

2.2 Placing the data and mapping options

Plots created with `ggplot2` always start with the `ggplot` function. In the examples above, the `data` and `mapping` options were placed in this function. In this case they apply to each `geom_` function that follows.

You can also place these options directly within a `geom`. In that case, they only apply only to that specific geom.

Consider the following graph.

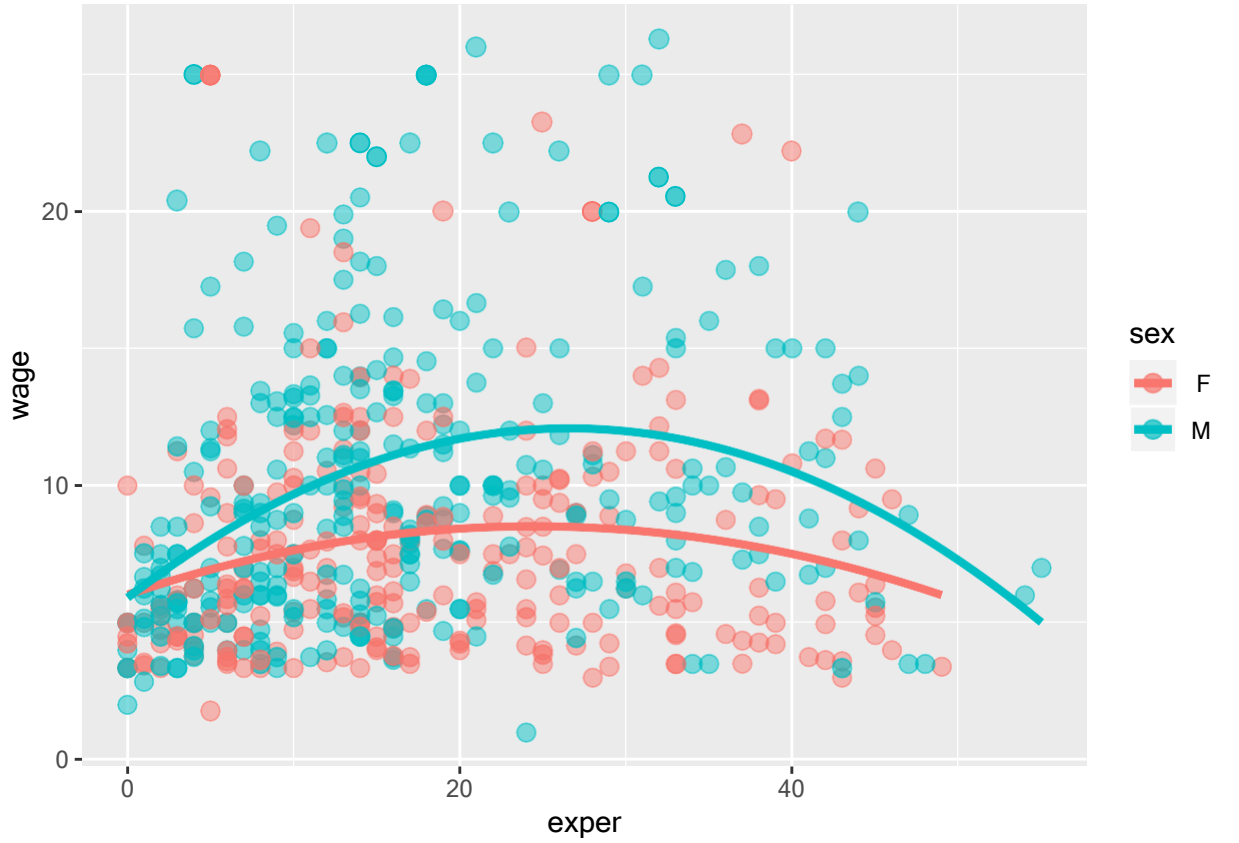


Figure 2.8: Color mapping in ggplot function

```
# placing color mapping in the ggplot function
ggplot(plotdata,
       aes(x = exper,
           y = wage,
           color = sex)) +
  geom_point(alpha = .7,
             size = 3) +
  geom_smooth(method = "lm",
              formula = y ~ poly(x,2),
              se = FALSE,
              size = 1.5)
```

Since the mapping of sex to color appears in the `ggplot` function, it applies to *both* `geom_point` and `geom_smooth`. The color of the point indicates the sex, and a separate colored trend line is produced for men and women. Compare this to

```
# placing color mapping in the geom_point function
ggplot(plotdata,
       aes(x = exper,
           y = wage)) +
  geom_point(aes(color = sex),
             alpha = .7,
             size = 3) +
```

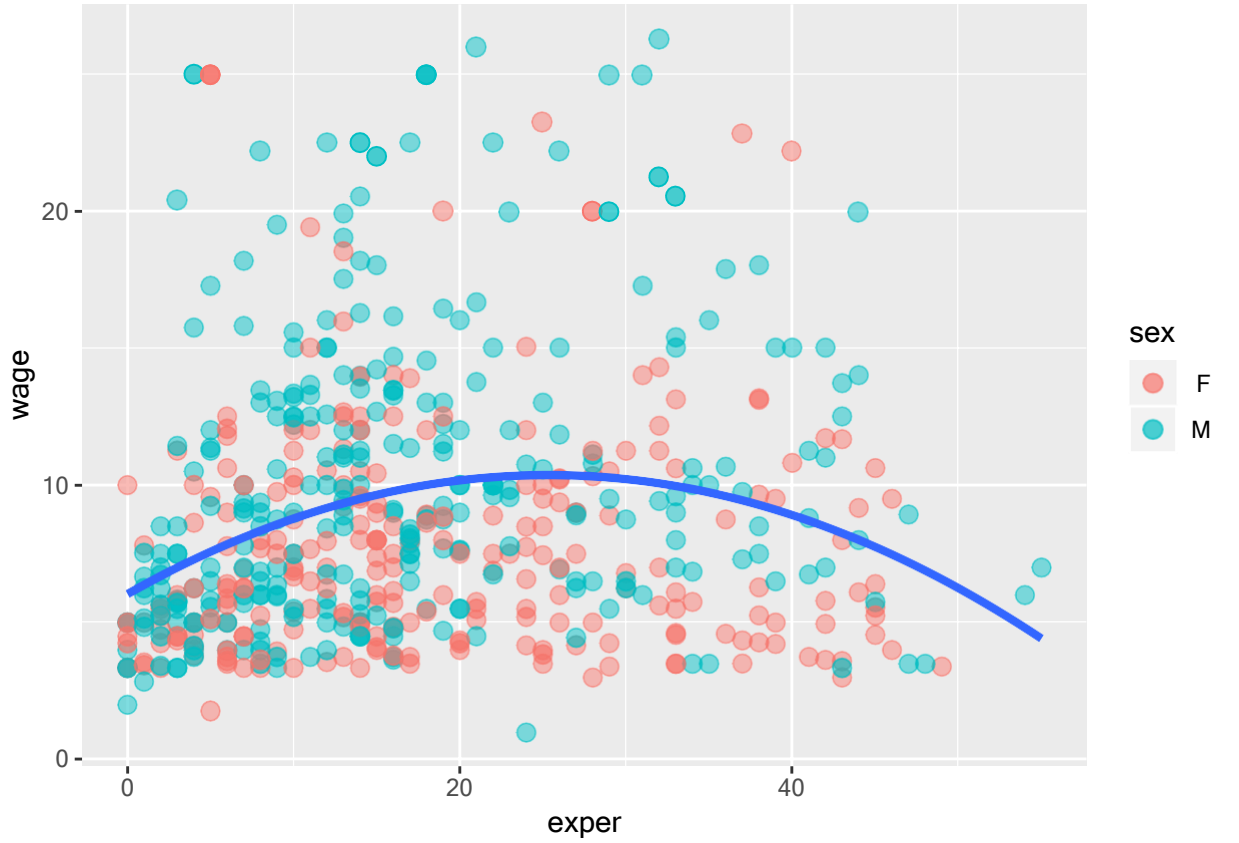


Figure 2.9: Color mapping in ggplot function

```
  geom_smooth(method = "lm",
              formula = y ~ poly(x,2),
              se = FALSE,
              size = 1.5)
```

Since the sex to color mapping only appears in the `geom_point` function, it is only used there. A single trend line is created for all observations.

Most of the examples in this book place the data and mapping options in the `ggplot` function. Additionally, the phrases *data=* and *mapping=* are omitted since the first option always refers to data and the second option always refers to mapping.

2.3 Graphs as objects

A `ggplot2` graph can be saved as a named R object (like a data frame), manipulated further, and thenprinted or saved to disk.

```
# prepare data
data(CPS85 , package = "mosaicData")
plotdata <- CPS85[CPS85$wage < 40,]
```

```
# create scatterplot and save it
myplot <- ggplot(data = plotdata,
                 aes(x = exper, y = wage)) +
            geom_point()

# print the graph
myplot

# make the points larger and blue
# then print the graph
myplot <- myplot + geom_point(size = 3, color = "blue")
myplot

# print the graph with a title and line of best fit
# but don't save those changes
myplot + geom_smooth(method = "lm") +
  labs(title = "Mildly interesting graph")

# print the graph with a black and white theme
# but don't save those changes
myplot + theme_bw()
```

This can be a real time saver (and help you avoid carpal tunnel syndrome). It is also handy when saving graphs programmatically.

Now it's time to try out other types of graphs.

Chapter 3

Univariate Graphs

Univariate graphs plot the distribution of data from a single variable. The variable can be categorical (e.g., race, sex) or quantitative (e.g., age, weight).

3.1 Categorical

The distribution of a single categorical variable is typically plotted with a bar chart, a pie chart, or (less commonly) a tree map.

3.1.1 Bar chart

The Marriage dataset contains the marriage records of 98 individuals in Mobile County, Alabama. Below, a bar chart is used to display the distribution of wedding participants by race.

```
library(ggplot2)
data(Marriage, package = "mosaicData")

# plot the distribution of race
ggplot(Marriage, aes(x = race)) +
  geom_bar()
```

The majority of participants are white, followed by black, with very few Hispanics or American Indians.

You can modify the bar fill and border colors, plot labels, and title by adding options to the `geom_bar` function.

```
# plot the distribution of race with modified colors and labels
ggplot(Marriage, aes(x = race)) +
  geom_bar(fill = "cornflowerblue",
           color="black") +
  labs(x = "Race",
       y = "Frequency",
       title = "Participants by race")
```

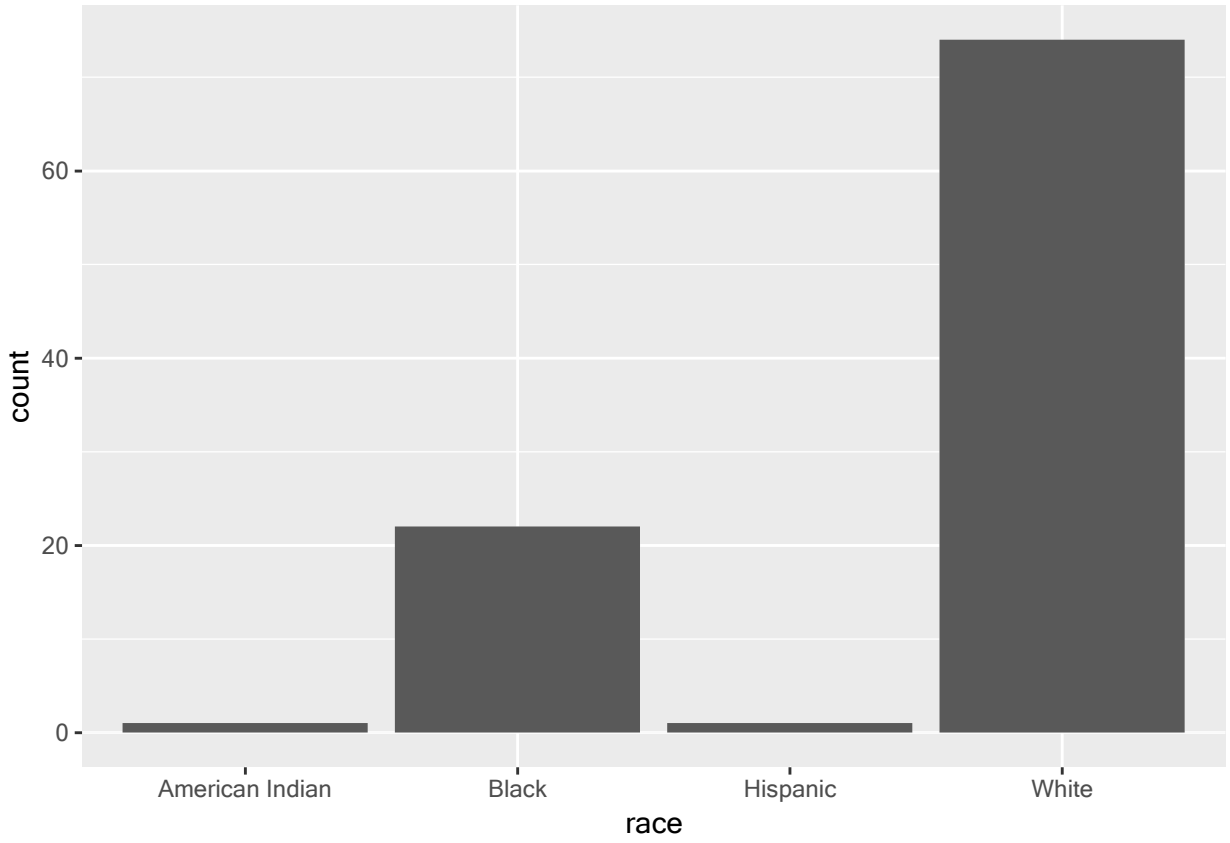


Figure 3.1: Simple barchart

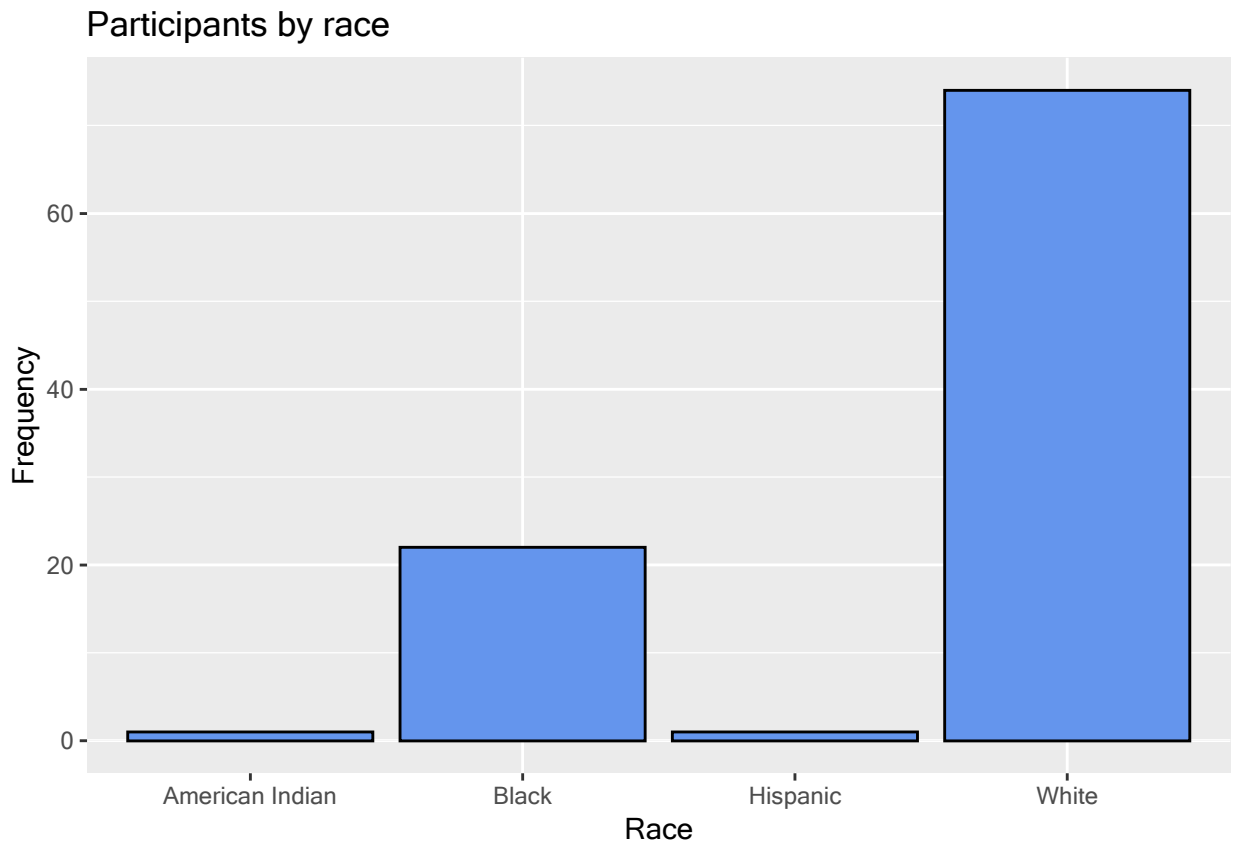


Figure 3.2: Barchart with modified colors, labels, and title

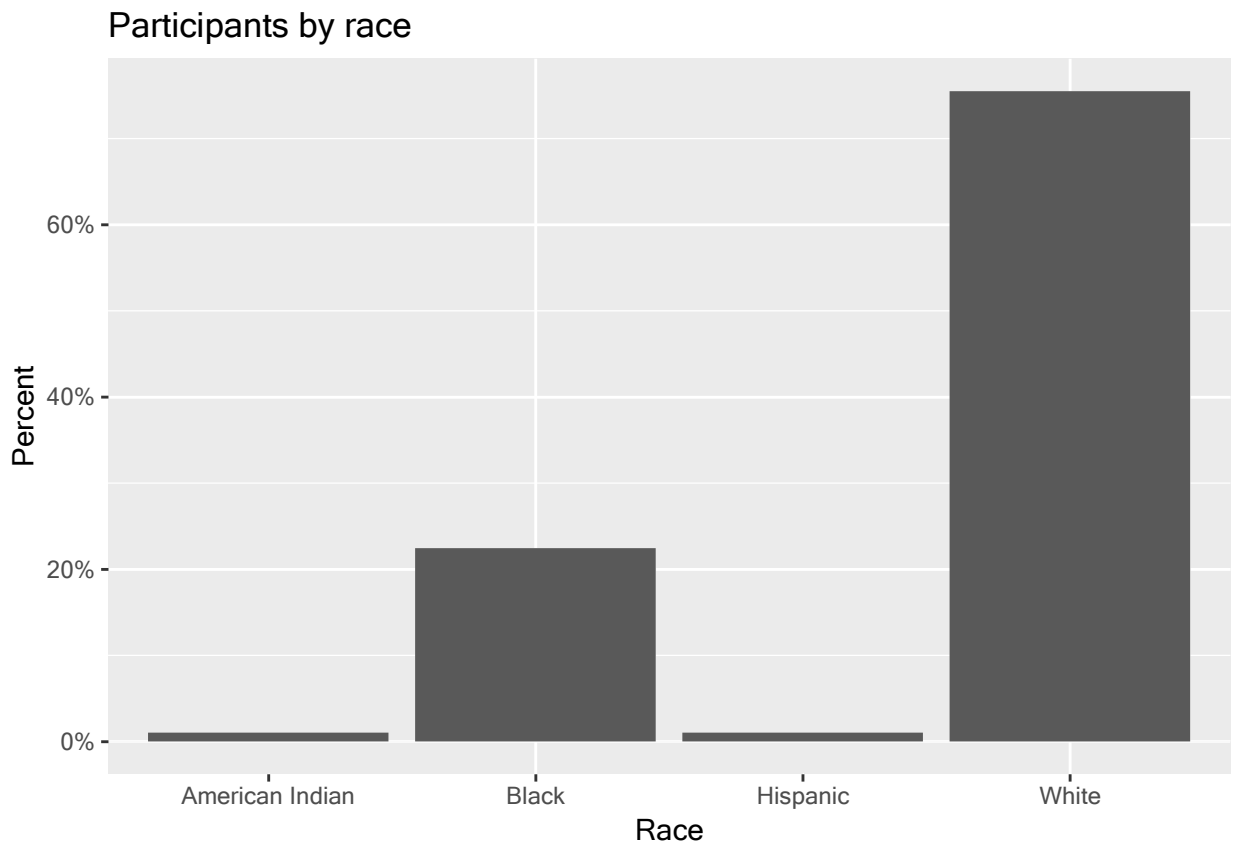


Figure 3.3: Barchart with percentages

3.1.1.1 Percents

Bars can represent percents rather than counts. For bar charts, the code `aes(x=race)` is actually a shortcut for `aes(x = race, y = ..count..)`, where `..count..` is a special variable representing the frequency within each category. You can use this to calculate percentages, by specifying the `y` variable explicitly.

```
# plot the distribution as percentages
ggplot(Marriage,
       aes(x = race,
           y = ..count.. / sum(..count..))) +
  geom_bar() +
  labs(x = "Race",
       y = "Percent",
       title = "Participants by race") +
  scale_y_continuous(labels = scales::percent)
```

In the code above, the `scales` package is used to add % symbols to the y-axis labels.

3.1.1.2 Sorting categories

It is often helpful to sort the bars by frequency. In the code below, the frequencies are calculated explicitly. Then the `reorder` function is used to sort the categories by the frequency. The option `stat="identity"` tells the plotting function not to calculate counts, because they are supplied directly.

Table 3.1: plotdata

race	n
American Indian	1
Black	22
Hispanic	1
White	74

```
# calculate number of participants in
# each race category
library(dplyr)
plotdata <- Marriage %>%
 count(race)
```

The resulting dataset is give below.

This new dataset is then used to create the graph.

```
# plot the bars in ascending order
ggplot(plotdata,
       aes(x = reorder(race, n),
           y = n)) +
  geom_bar(stat = "identity") +
  labs(x = "Race",
       y = "Frequency",
       title  = "Participants by race")
```

The graph bars are sorted in ascending order. Use `reorder(race, -n)` to sort in descending order.

3.1.1.3 Labeling bars

Finally, you may want to label each bar with its numerical value.

```
# plot the bars with numeric labels
ggplot(plotdata,
       aes(x = race,
           y = n)) +
  geom_bar(stat = "identity") +
  geom_text(aes(label = n),
            vjust=-0.5) +
  labs(x = "Race",
       y = "Frequency",
       title  = "Participants by race")
```

Here `geom_text` adds the labels, and `vjust` controls vertical justification. See Annotations for more details.

Putting these ideas together, you can create a graph like the one below. The minus sign in `reorder(race, -pct)` is used to order the bars in descending order.

```
library(dplyr)
library(scales)
```

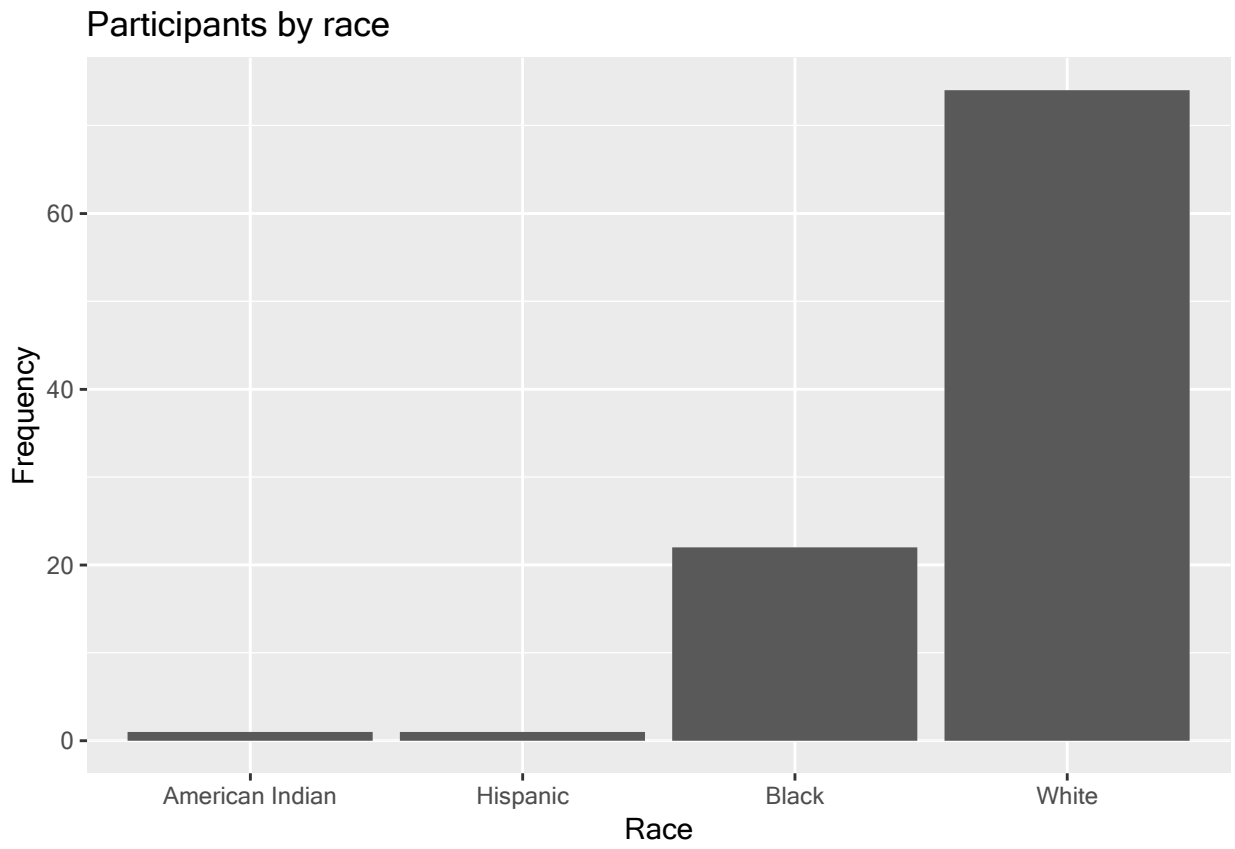


Figure 3.4: Sorted bar chart

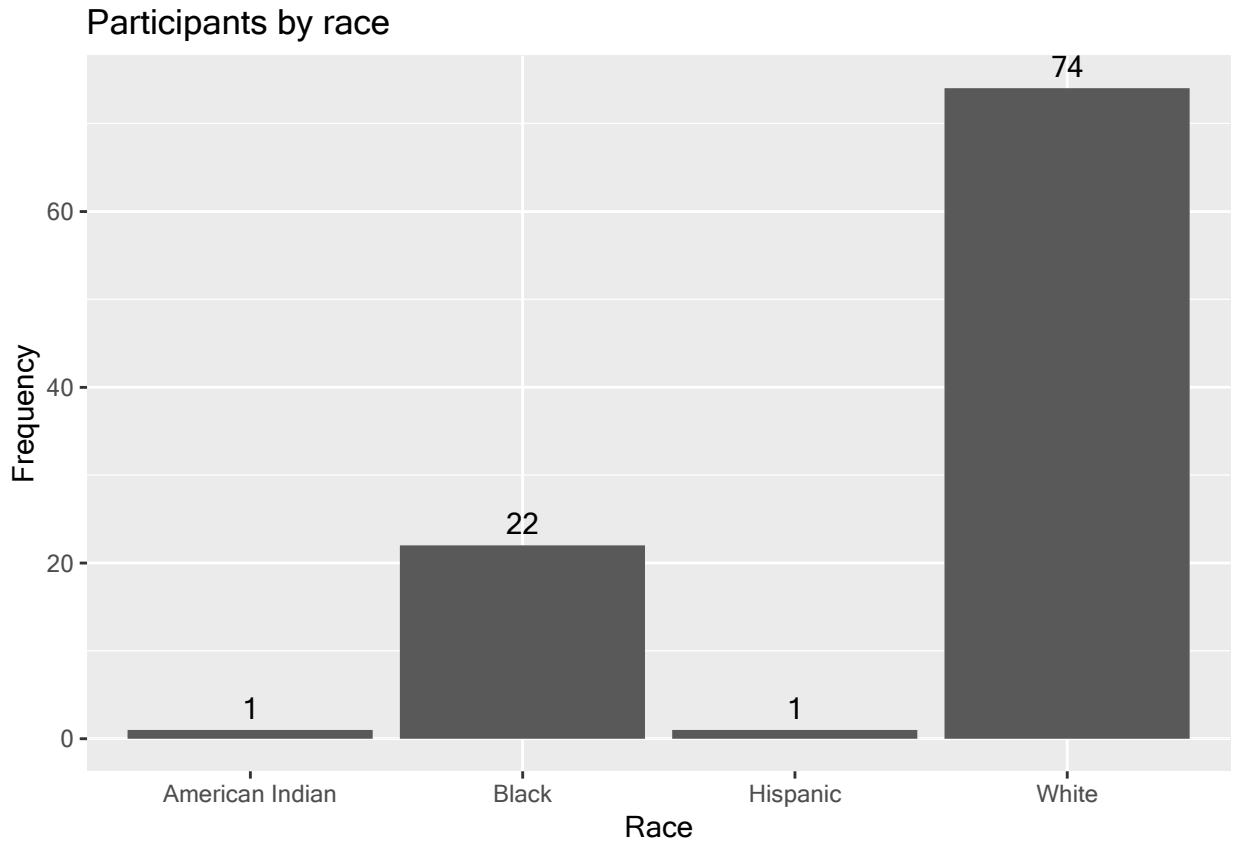


Figure 3.5: Bar chart with numeric labels

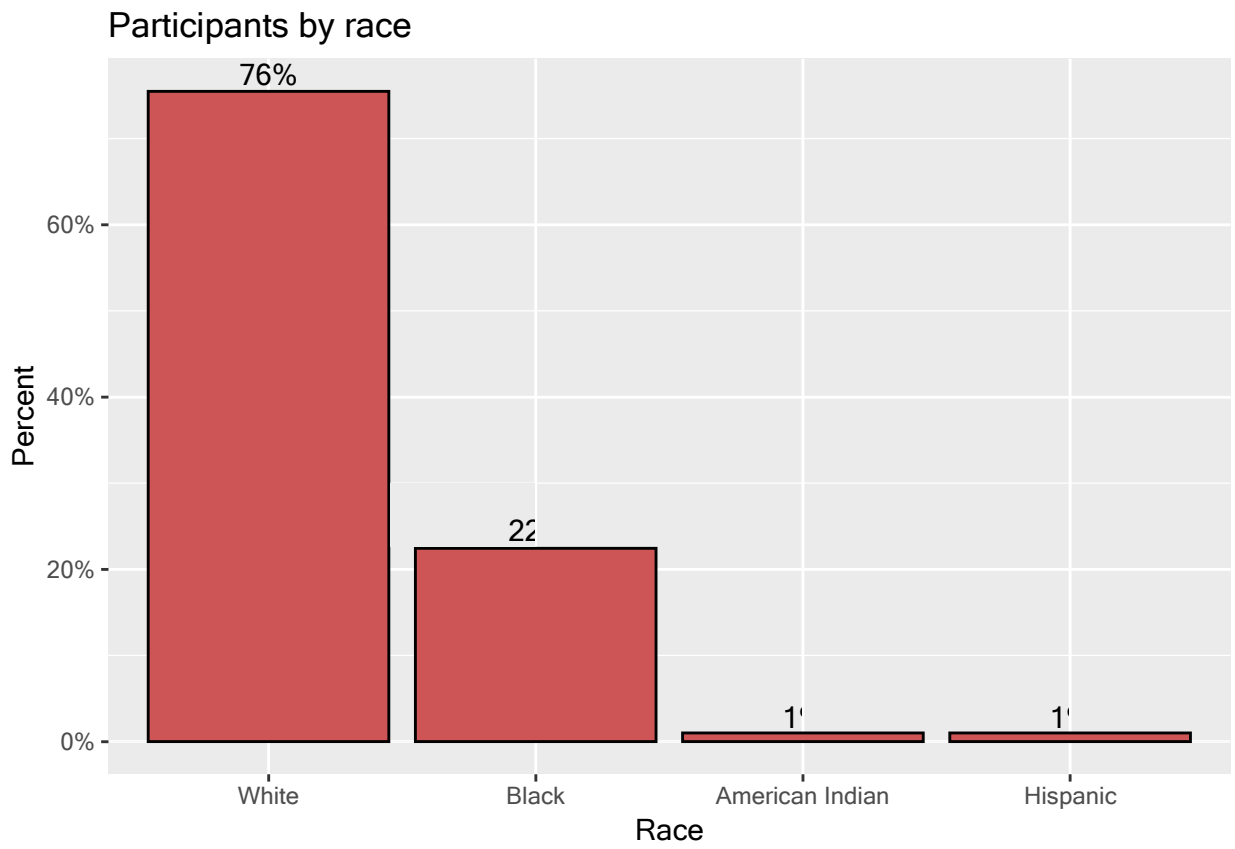


Figure 3.6: Sorted bar chart with percent labels

```
plotdata <- Marriage %>%
  count(race) %>%
  mutate(pct = n / sum(n),
         pctlabel = paste0(round(pct*100), "%"))

# plot the bars as percentages,
# in decending order with bar labels
ggplot(plotdata,
       aes(x = reorder(race, -pct),
           y = pct)) +
  geom_bar(stat = "identity",
           fill = "indianred3",
           color = "black") +
  geom_text(aes(label = pctlabel),
            vjust = -0.25) +
  scale_y_continuous(labels = percent) +
  labs(x = "Race",
       y = "Percent",
       title  = "Participants by race")
```

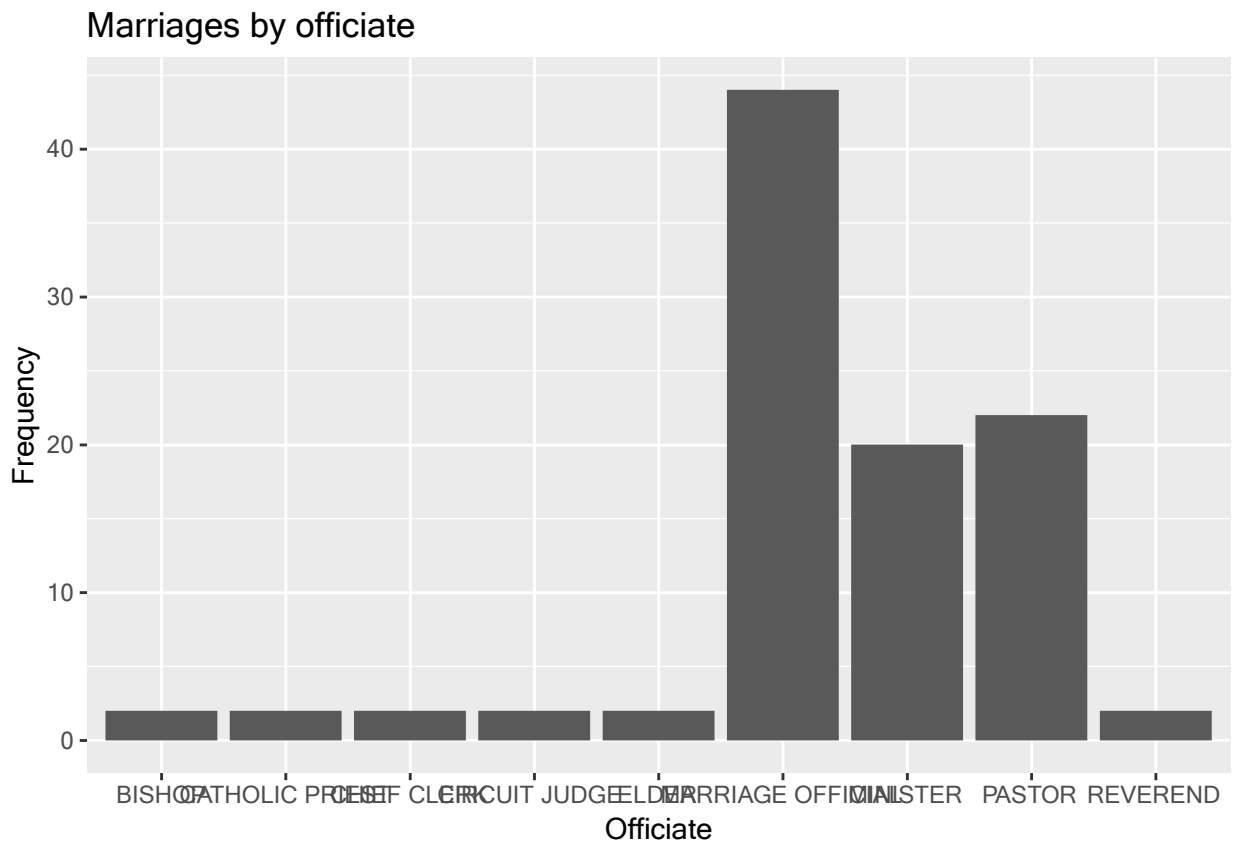


Figure 3.7: Barchart with problematic labels

3.1.1.4 Overlapping labels

Category labels may overlap if (1) there are many categories or (2) the labels are long. Consider the distribution of marriage officials.

```
# basic bar chart with overlapping labels
ggplot(Marriage, aes(x = officialTitle)) +
  geom_bar() +
  labs(x = "Officiate",
       y = "Frequency",
       title = "Marriages by officiate")
```

In this case, you can flip the x and y axes.

```
# horizontal bar chart
ggplot(Marriage, aes(x = officialTitle)) +
  geom_bar() +
  labs(x = "",
       y = "Frequency",
       title = "Marriages by officiate") +
  coord_flip()
```

Alternatively, you can rotate the axis labels.

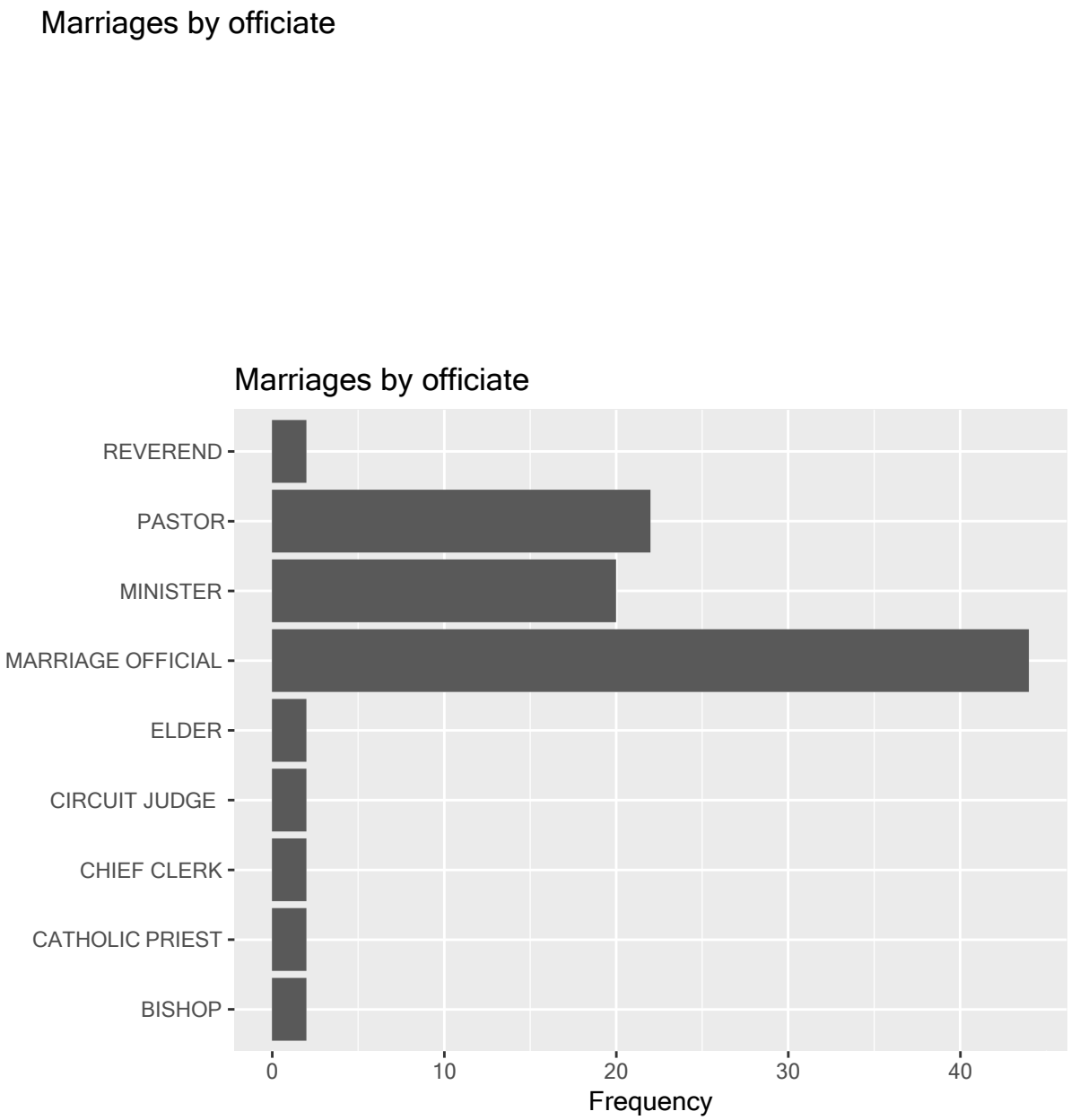


Figure 3.8: Horizontal barchart

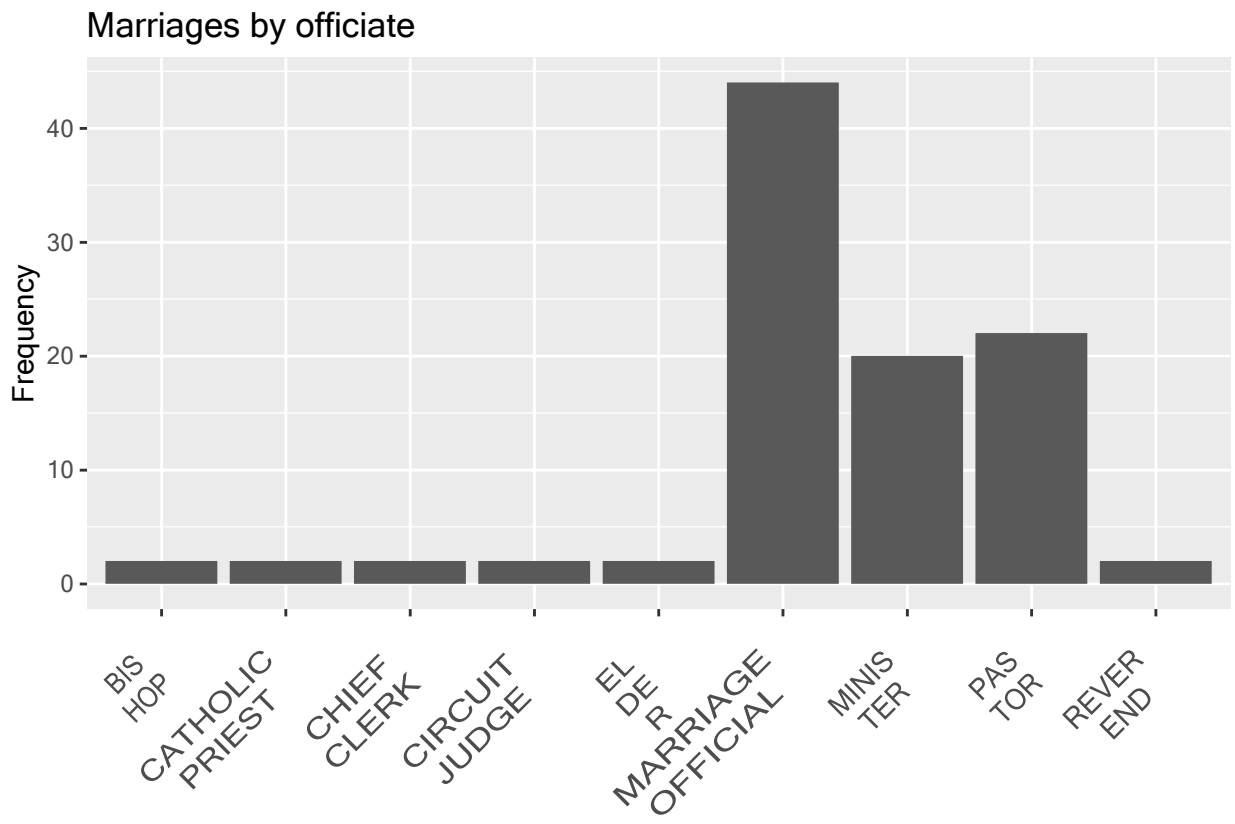
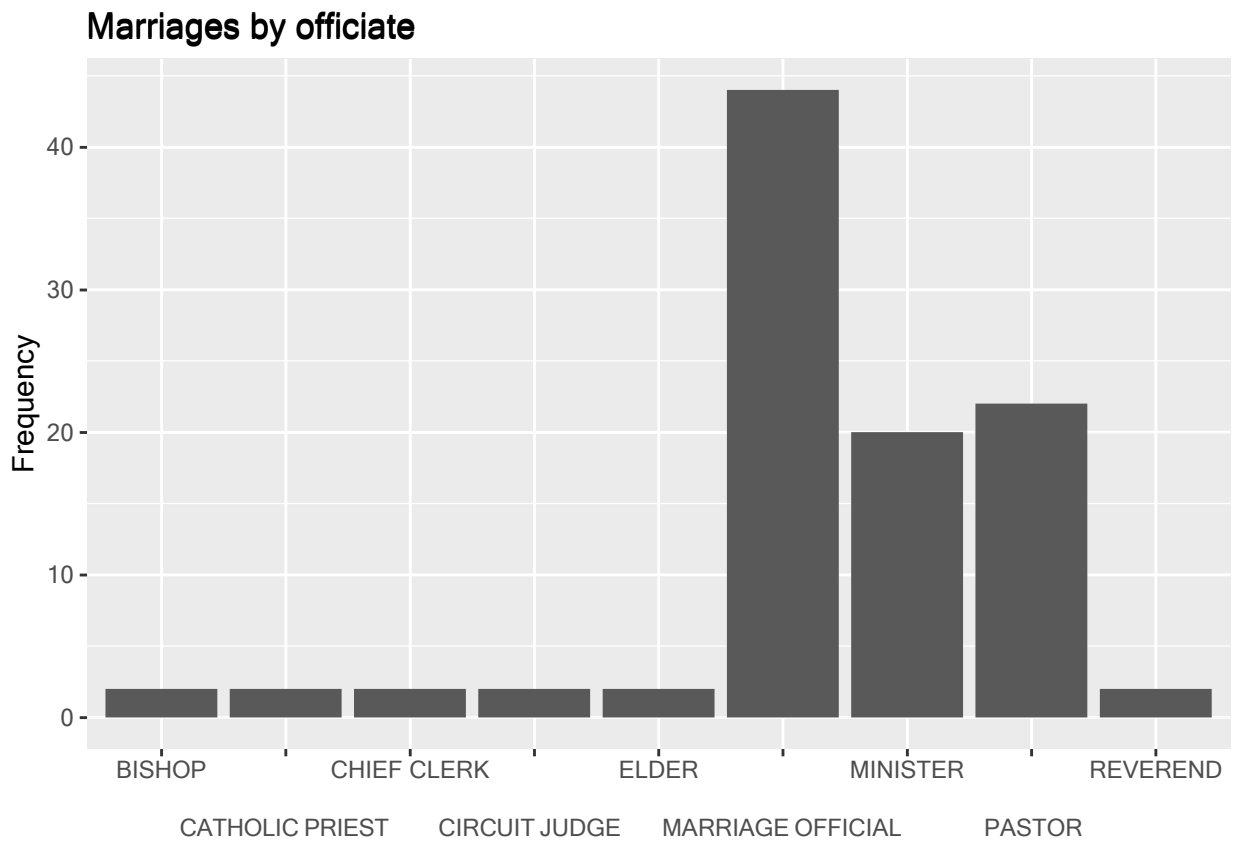


Figure 3.9: Barchart with rotated labels

```
# bar chart with rotated labels
ggplot(Marriage, aes(x = officialTitle)) +
  geom_bar() +
  labs(x = "",
       y = "Frequency",
       title = "Marriages by officiate") +
  theme(axis.text.x = element_text(angle = 45,
                                   hjust = 1))
```

Finally, you can try staggering the labels. The trick is to add a newline `\n` to every other label.

```
# bar chart with staggered labels
lbls <- paste0(c("", "\n"),
               levels(Marriage$officialTitle))
ggplot(Marriage,
       aes(x=factor(officialTitle,
                    labels = lbls))) +
  geom_bar() +
  labs(x = "",
       y = "Frequency",
       title = "Marriages by officiate")
```



Pie chart

Pie charts are controversial in statistics. If your goal is to compare the frequency of categories, you are better off with bar charts (humans are better at judging the length of bars than the volume of pie slices). If your goal is compare each category with the the whole (e.g., what portion of participants are Hispanic compared to all participants), and the number of categories is small, then pie charts may work for you. It takes a bit more code to make an attractive pie chart in R.

```
# create a basic ggplot2 pie chart
 plotdata <- Marriage %>%
    count(race) %>%
    arrange(desc(race)) %>%
    mutate(prop = round(n * 100 / sum(n), 1),
           lab.ypos = cumsum(prop) - 0.5 *prop)

  ggplot(plotdata,
         aes(x = "",
             y = prop,
             fill = race)) +
    geom_bar(width = 1,
             stat = "identity",
             color = "black") +
    coord_polar("y",
                start = 0,
                direction = -1) +
    theme_void()
```

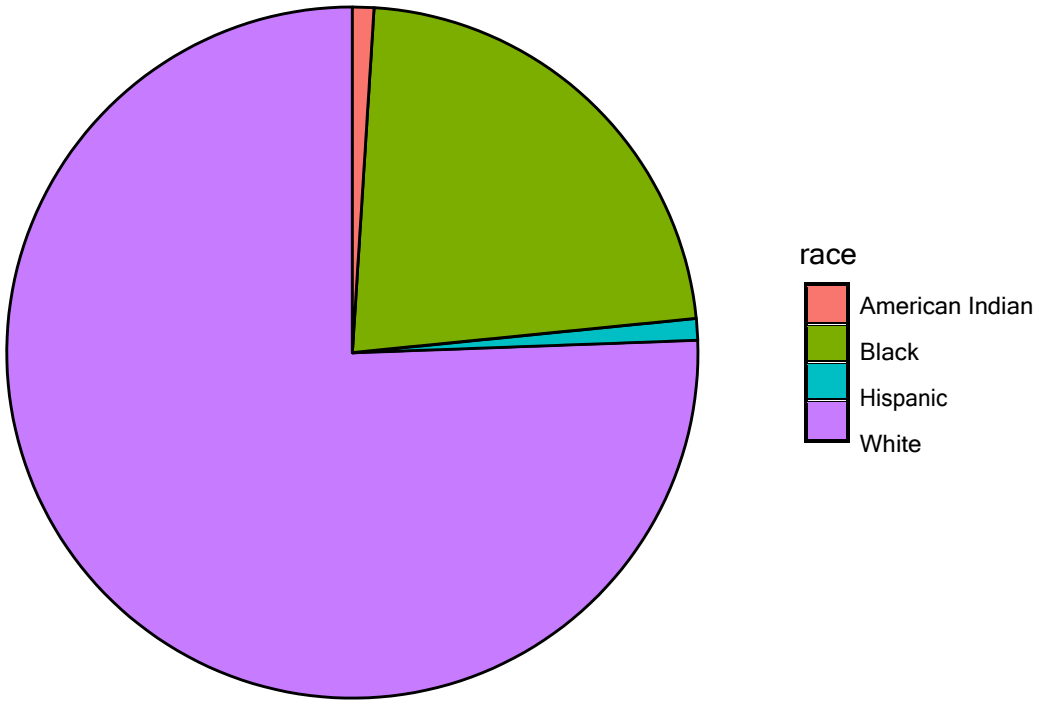


Figure 3.10: Basic pie chart

Now let's get fancy and add labels, while removing the legend.

```
# create a pie chart with slice labels
plotdata <- Marriage %>%
  count(race) %>%
  arrange(desc(race))  %>%
  mutate(prop = round(n*100/sum(n), 1),
         lab.ypos = cumsum(prop) - 0.5*prop)

plotdata$label <- paste0(plotdata$race, "\n",
                         round(plotdata$prop), "%")

ggplot(plotdata,
       aes(x = "",
           y = prop,
           fill = race)) +
  geom_bar(width = 1,
           stat = "identity",
           color = "black") +
  geom_text(aes(y = lab.ypos, label = label),
            color = "black") +
  coord_polar("y",
              start = 0,
              direction = -1) +
  theme_void() +
  theme(legend.position = "FALSE") +
  labs(title = "Participants by race")
```

The pie chart makes it easy to compare each slice with the whole. For example, Back is seen to roughly a quarter of the total participants.

3.1.2 Tree map

An alternative to a pie chart is a tree map. Unlike pie charts, it can handle categorical variables that have *many* levels.

```
library(treemapify)

# create a treemap of marriage officials
plotdata <- Marriage %>%
  count(officialTitle)

ggplot(plotdata,
       aes(fill = officialTitle,
           area = n)) +
  geom_treemap() +
  labs(title = "Marriages by officiate")
```

Here is a more useful version with labels.

```
# create a treemap with tile labels
ggplot(plotdata,
```

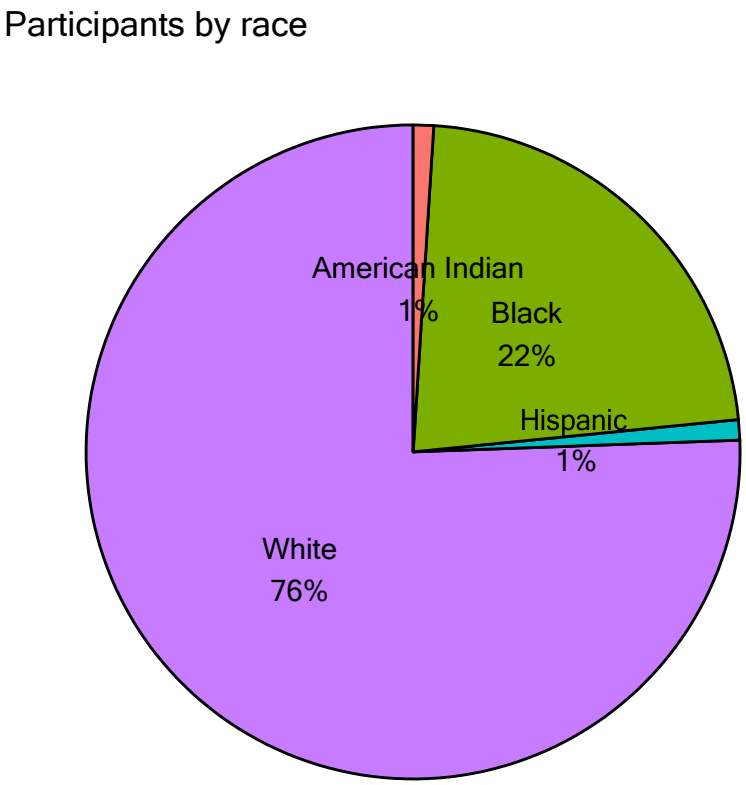


Figure 3.11: Pie chart with percent labels

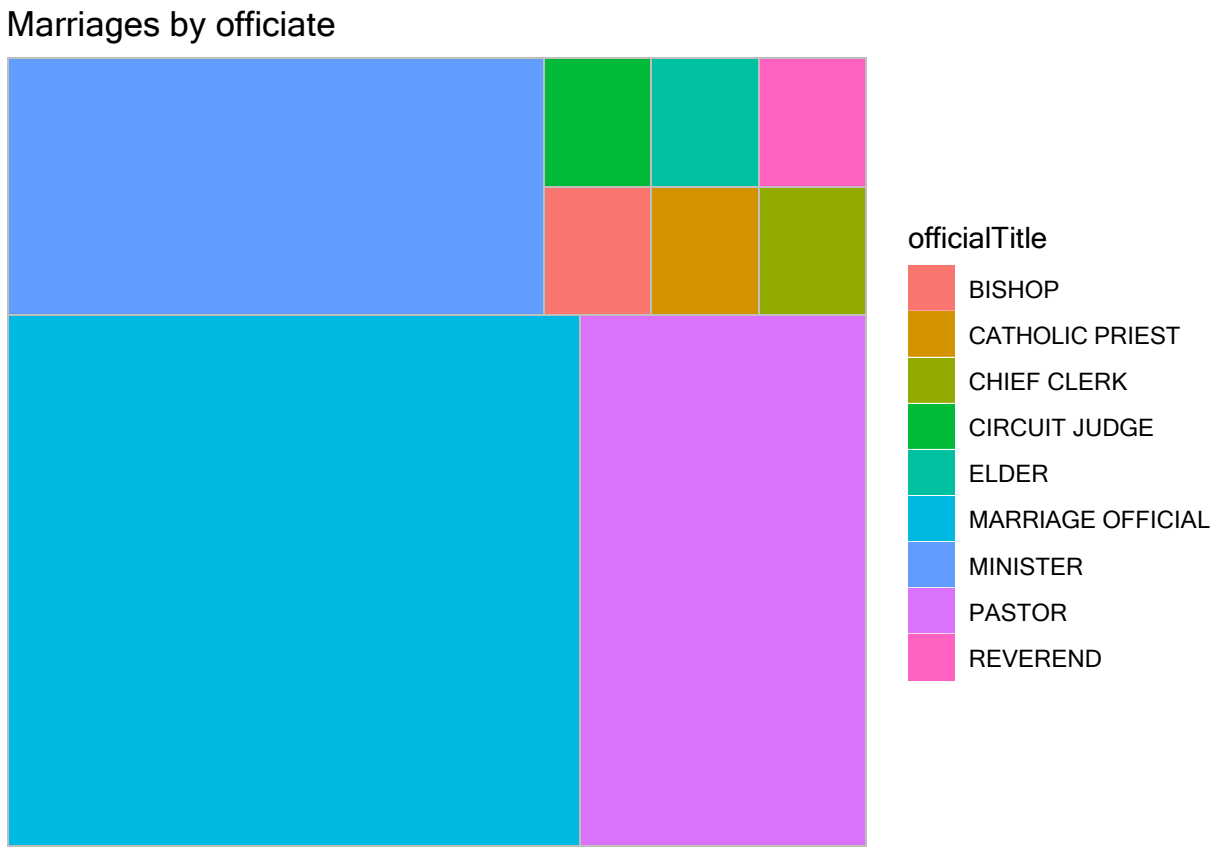


Figure 3.12: Basic treemap

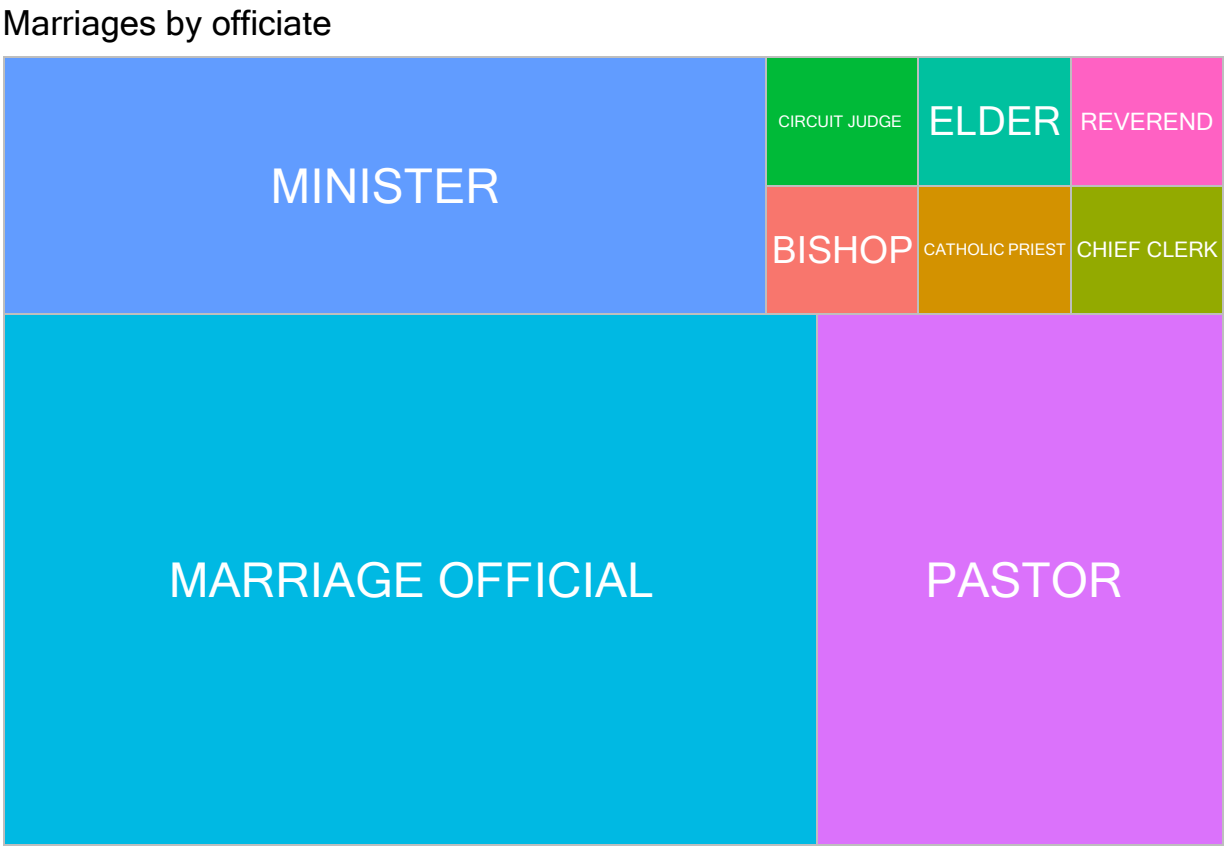


Figure 3.13: Treemap with labels

```
      aes(fill = officialTitle,
          area = n,
          label = officialTitle)) +
geom_treemap() +
geom_treemap_text(colour = "white",
                  place = "centre") +
labs(title = "Marriages by officiate") +
theme(legend.position = "none")
```

3.2 Quantitative

The distribution of a single quantitative variable is typically plotted with a histogram, kernel density plot, or dot plot.

3.2.1 Histogram

Using the Marriage dataset, let's plot the ages of the wedding participants.

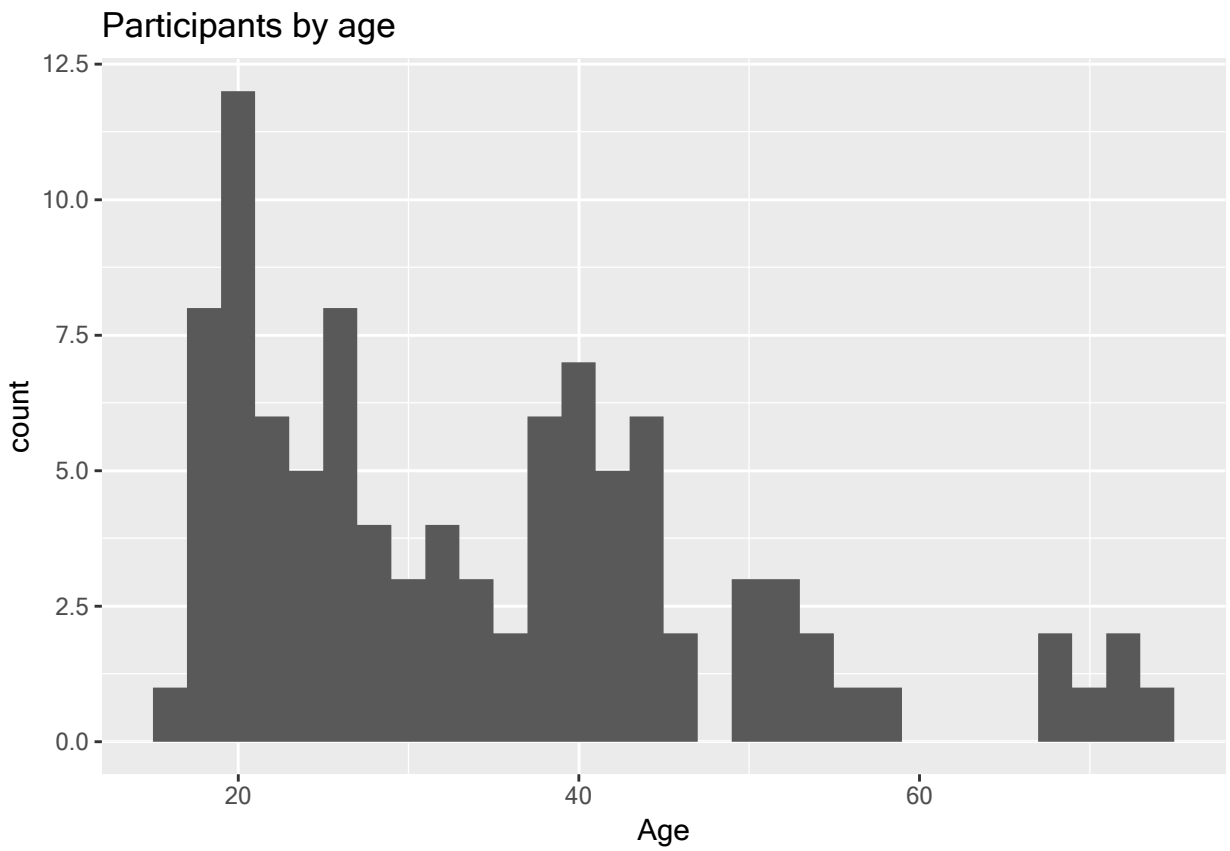


Figure 3.14: Basic histogram

```
library(ggplot2)

# plot the age distribution using a histogram
ggplot(Marriage, aes(x = age)) +
  geom_histogram() +
  labs(title = "Participants by age",
       x = "Age")
```

Most participants appear to be in their early 20's with another group in their 40's, and a much smaller group in their later sixties and early seventies. This would be a *multimodal* distribution.

Histogram colors can be modified using two options

- `fill` - fill color for the bars
- `color` - border color around the bars

```
# plot the histogram with blue bars and white borders
ggplot(Marriage, aes(x = age)) +
  geom_histogram(fill = "cornflowerblue",
                 color = "white") +
  labs(title="Participants by age",
       x = "Age")
```

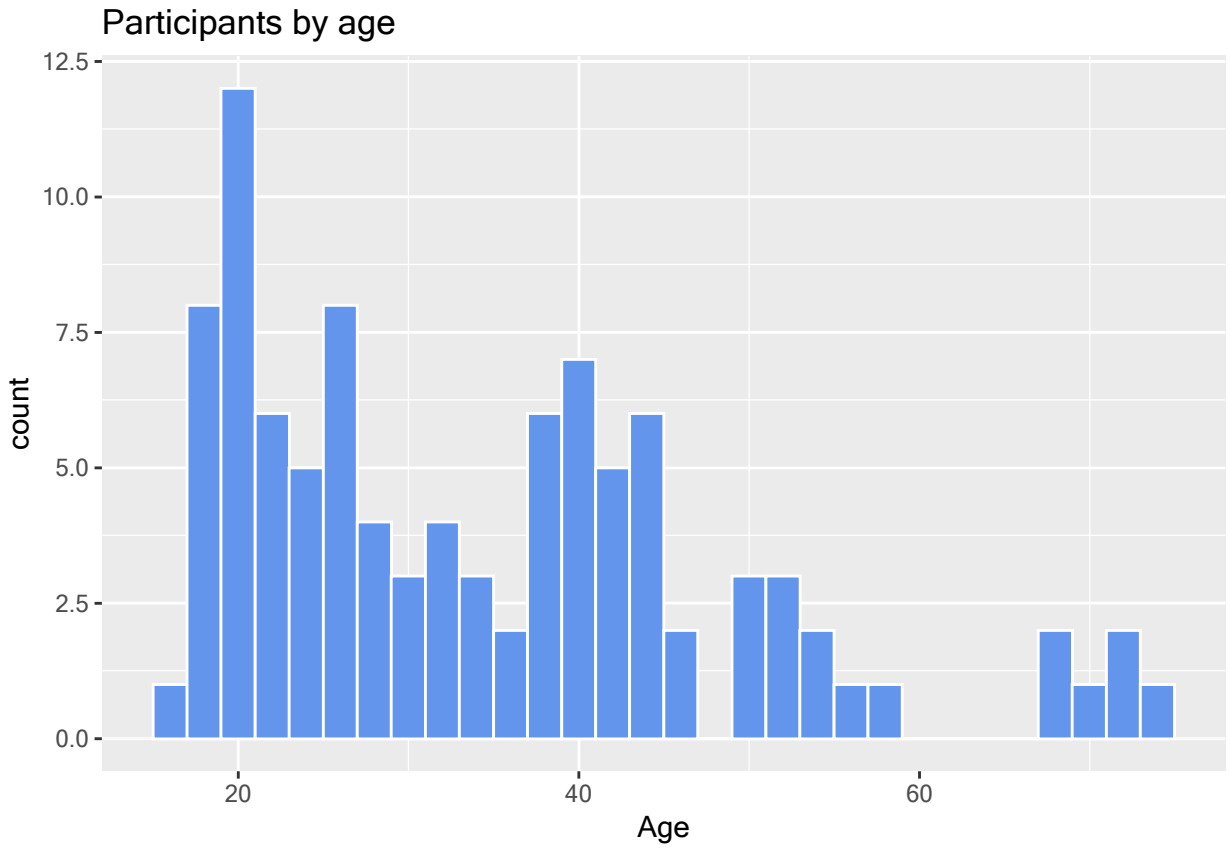


Figure 3.15: Histogram with specified fill and border colors

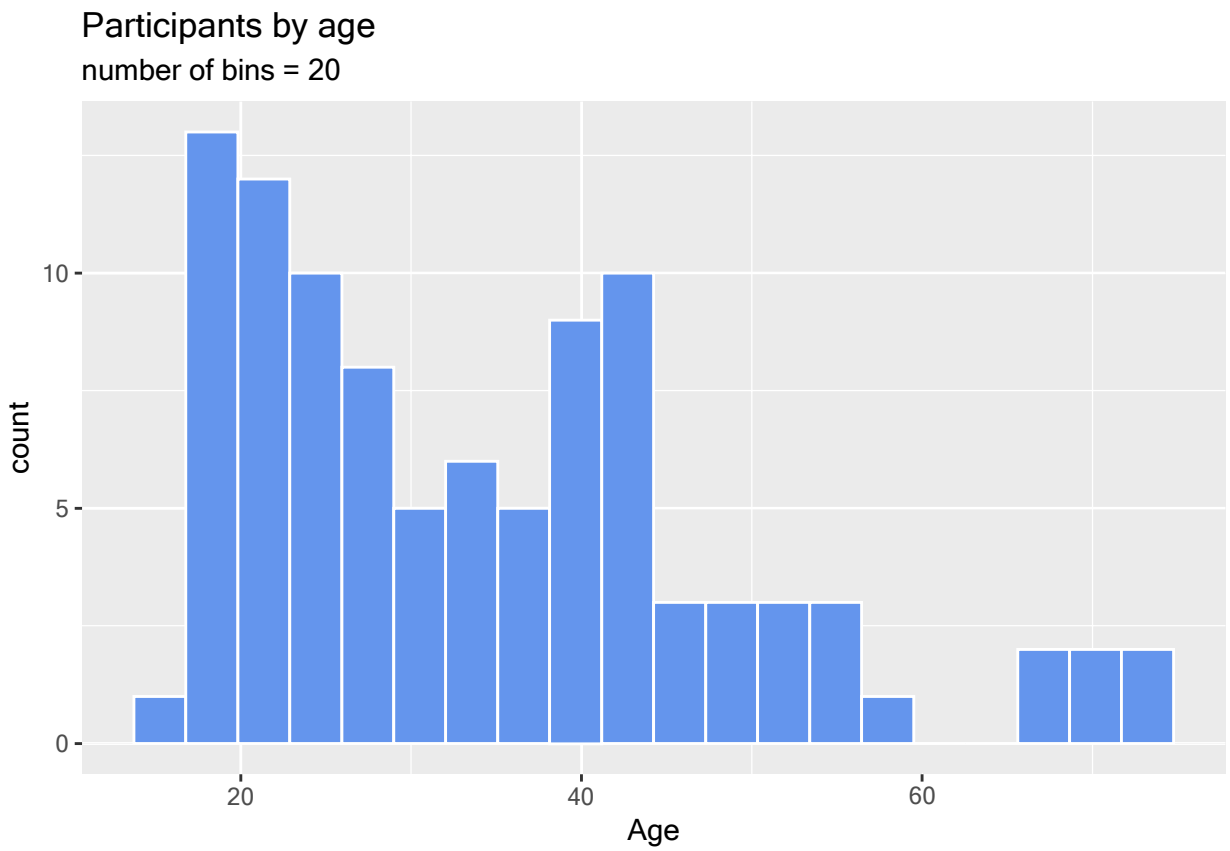


Figure 3.16: Histogram with a specified number of bins

3.2.1.1 Bins and bandwidths

One of the most important histogram options is `bins`, which controls the number of bins into which the numeric variable is divided (i.e., the number of bars in the plot). The default is 30, but it is helpful to try smaller and larger numbers to get a better impression of the shape of the distribution.

```
# plot the histogram with 20 bins
ggplot(Marriage, aes(x = age)) +
  geom_histogram(fill = "cornflowerblue",
                 color = "white",
                 bins = 20) +
  labs(title="Participants by age",
       subtitle = "number of bins = 20",
       x = "Age")
```

Alternatively, you can specify the `binwidth`, the width of the bins represented by the bars.

```
# plot the histogram with a binwidth of 5
ggplot(Marriage, aes(x = age)) +
  geom_histogram(fill = "cornflowerblue",
                 color = "white",
                 binwidth = 5) +
  labs(title="Participants by age",
```

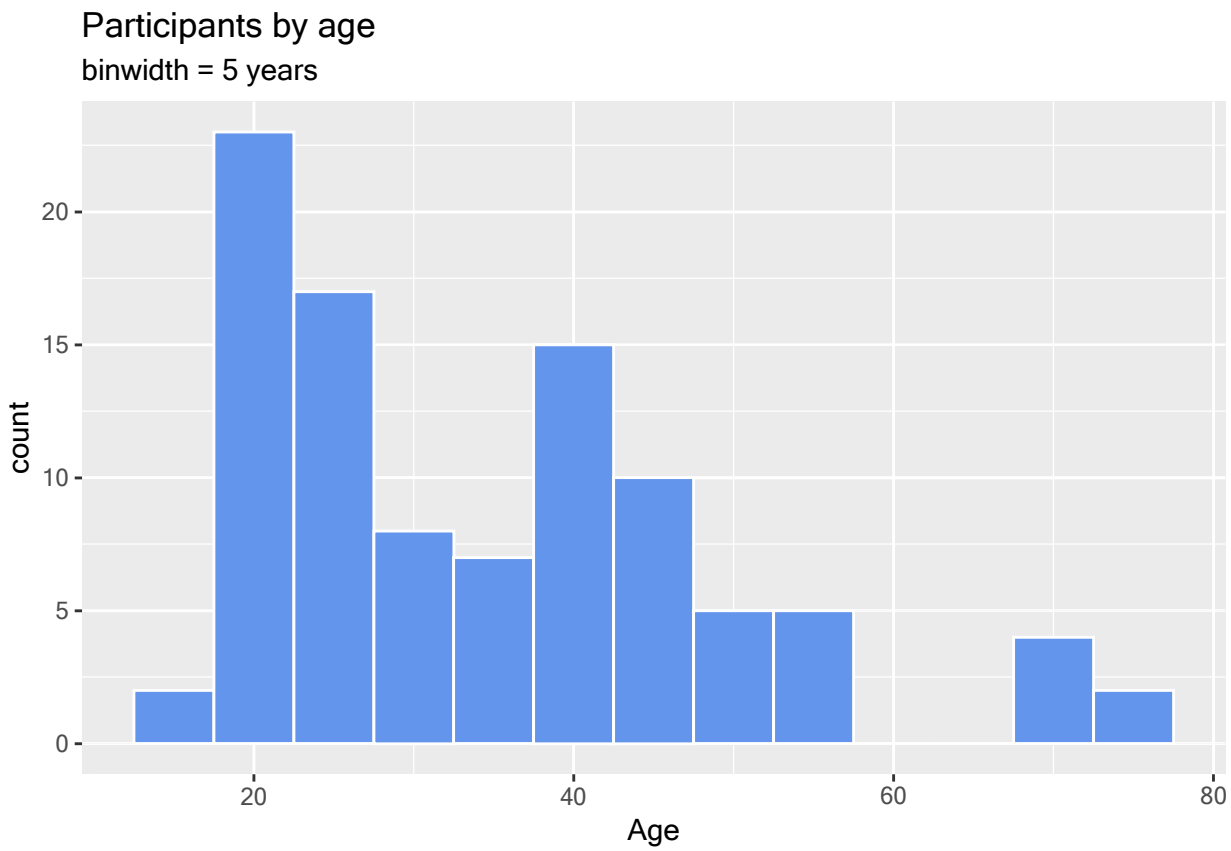
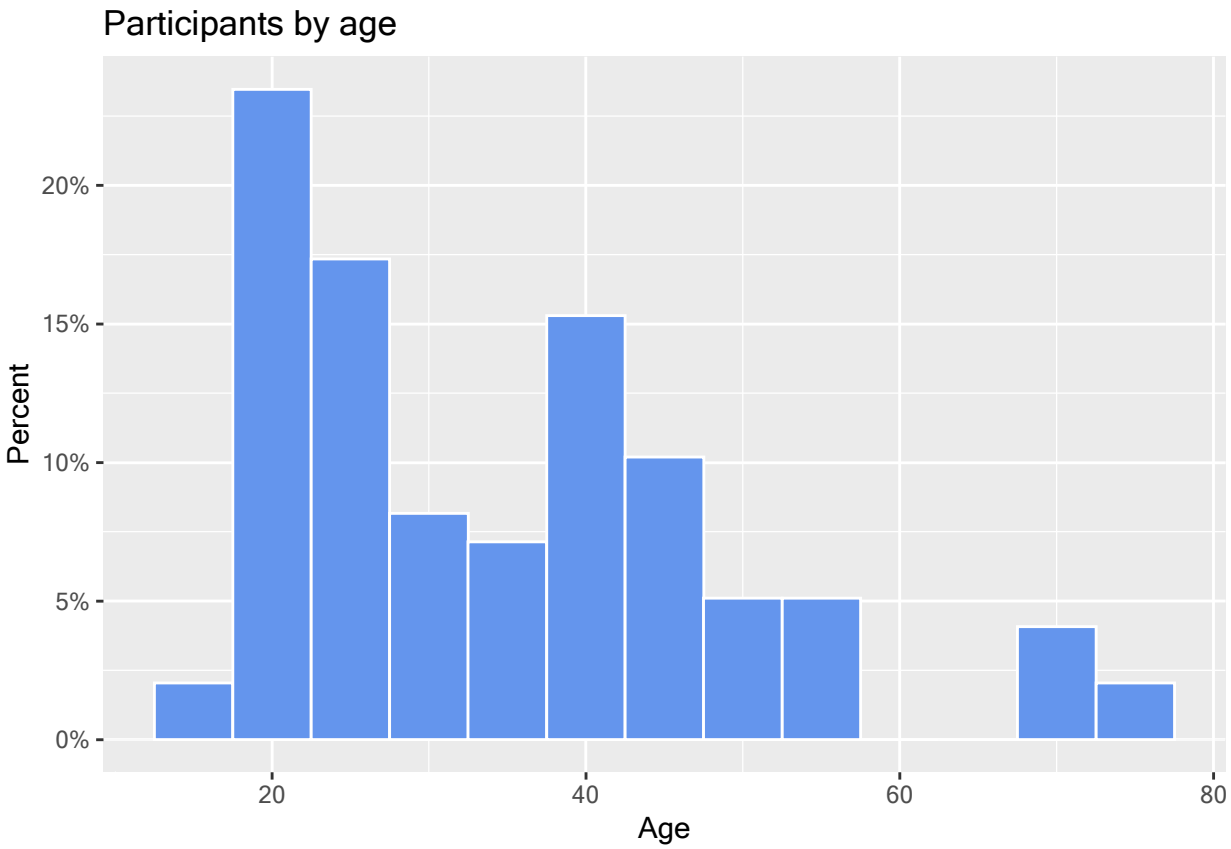


Figure 3.17: Histogram with specified a bin width

```
       subtitle = "binwidth = 5 years",
       x = "Age")
```

As with bar charts, the y-axis can represent counts or percent of the total.

```
# plot the histogram with percentages on the y-axis
library(scales)
ggplot(Marriage,
       aes(x = age,
           y= ..count.. / sum(..count..))) +
  geom_histogram(fill = "cornflowerblue",
                 color = "white",
                 binwidth = 5) +
  labs(title="Participants by age",
       y = "Percent",
       x = "Age") +
  scale_y_continuous(labels = percent)
```



Kernel Density plot {#Kernel}

An alternative to a histogram is the kernel density plot. Technically, kernel density estimation is a nonparametric method for estimating the probability density function of a continuous random variable. (What??) Basically, we are trying to draw a smoothed histogram, where the area under the curve equals one.

```
# Create a kernel density plot of age
ggplot(Marriage, aes(x = age)) +
  geom_density() +
  labs(title = "Participants by age")
```

The graph shows the distribution of scores. For example, the proportion of cases between 20 and 40 years old would be represented by the area under the curve between 20 and 40 on the x-axis.

As with previous charts, we can use `fill` and `color` to specify the fill and border colors.

```
# Create a kernel density plot of age
ggplot(Marriage, aes(x = age)) +
  geom_density(fill = "indianred3") +
  labs(title = "Participants by age")
```

3.2.1.2 Smoothing parameter

The degree of smoothness is controlled by the bandwidth parameter `bw`. To find the default value for a particular variable, use the `bw.nrd0` function. Values that are larger will result in more smoothing, while values that are smaller will produce less smoothing.

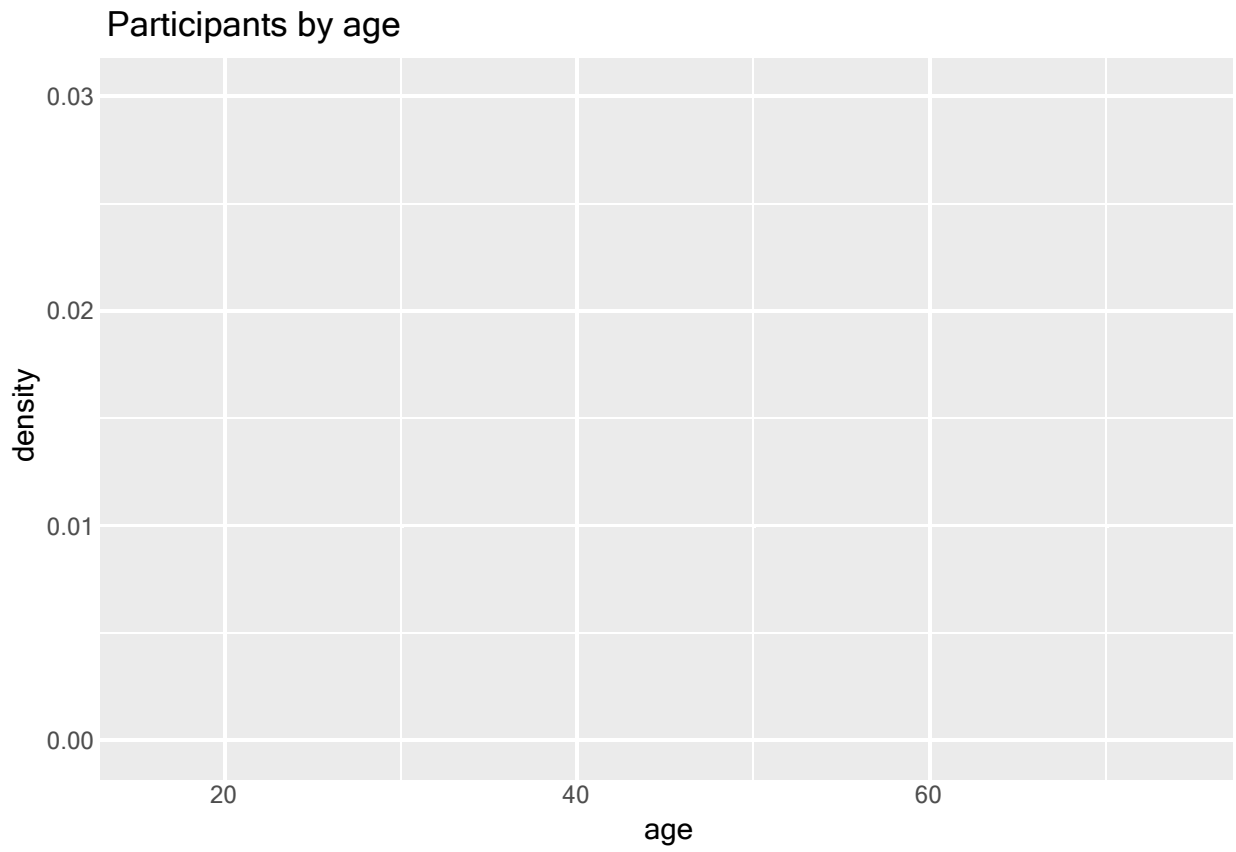


Figure 3.18: Basic kernel density plot

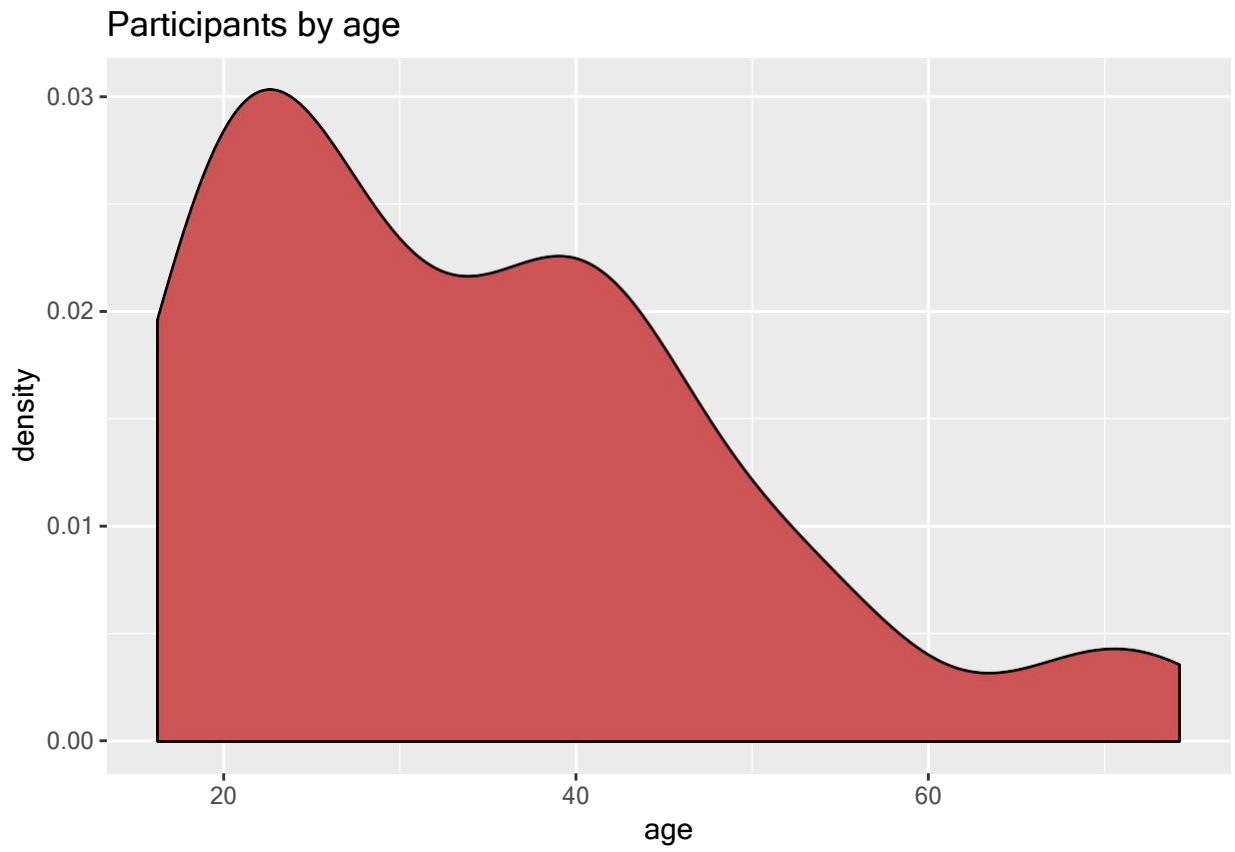


Figure 3.19: Kernel density plot with fill

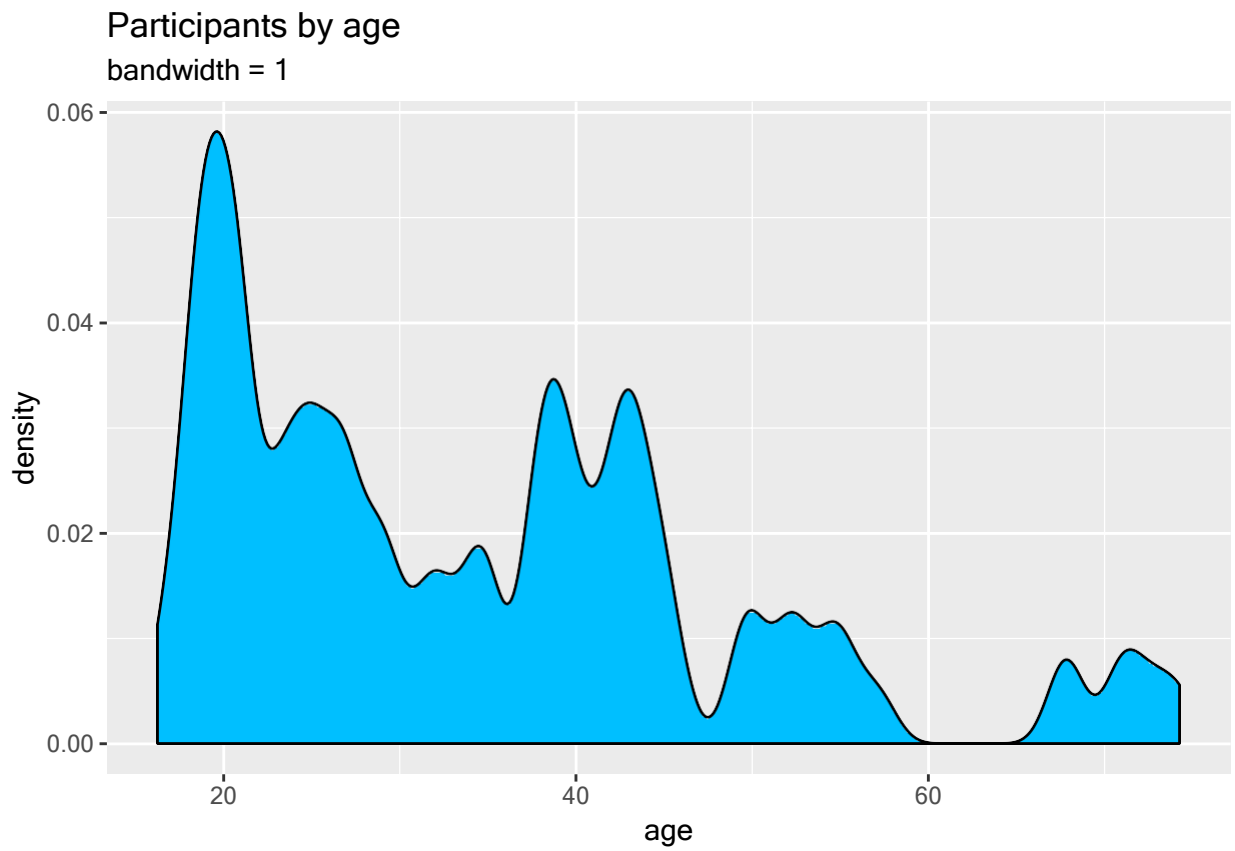


Figure 3.20: Kernel density plot with a specified bandwidth

```
# default bandwidth for the age variable
bw.nrd0(Marriage$age)
```

```
## [1] 5.181946
```

```
# Create a kernel density plot of age
ggplot(Marriage, aes(x = age)) +
  geom_density(fill = "deepskyblue",
               bw = 1) +
  labs(title = "Participants by age",
       subtitle = "bandwidth = 1")
```

In this example, the default bandwidth for age is 5.18. Choosing a value of 1 resulted in less smoothing and more detail.

Kernel density plots allow you to easily see which scores are most frequent and which are relatively rare. However it can be difficult to explain the meaning of the y-axis to a non-statistician. (But it will make you look really smart at parties!)

3.2.2 Dot Chart

Another alternative to the histogram is the dot chart. Again, the quantitative variable is divided into bins, but rather than summary bars, each observation is represented by a dot. By default, the width of a dot

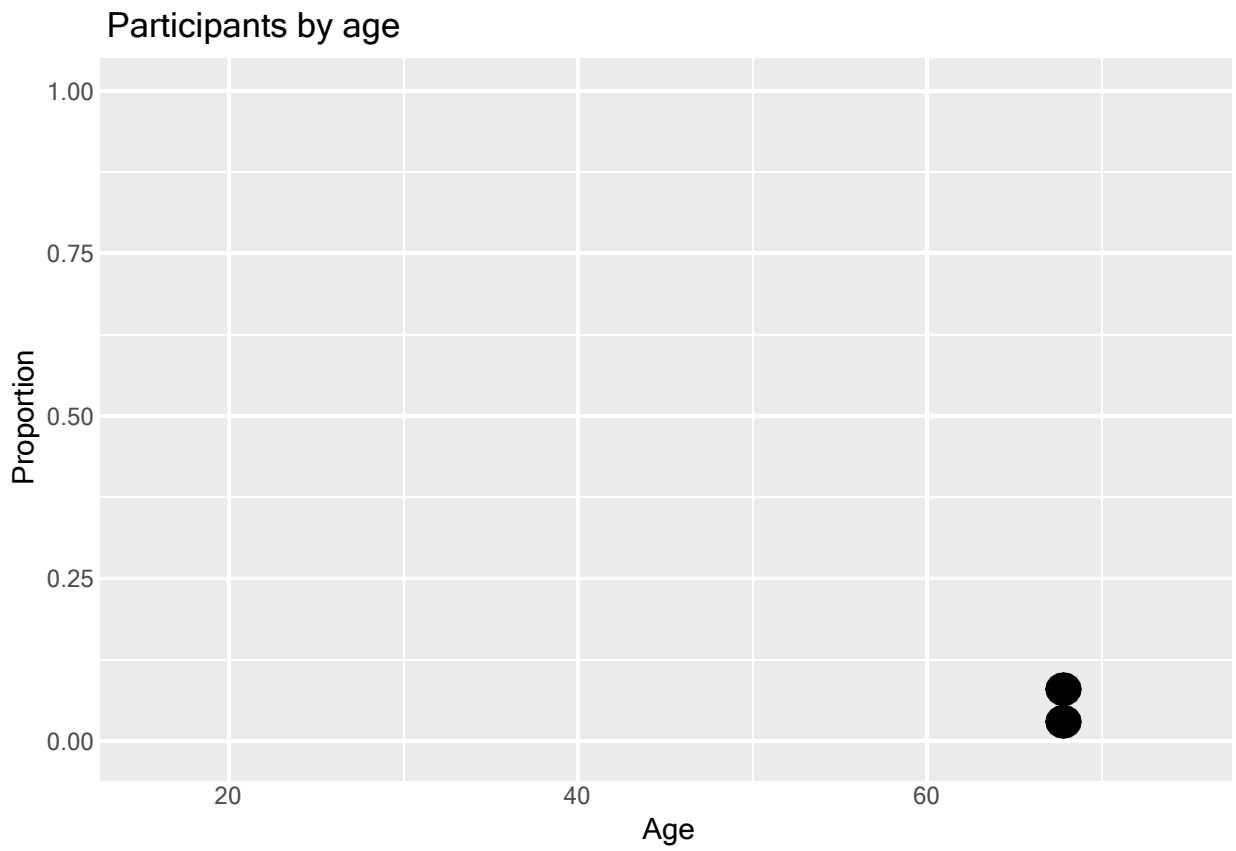


Figure 3.21: Basic dotplot

corresponds to the bin width, and dots are stacked, with each dot representing one observation. This works best when the number of observations is small (say, less than 150).

```
# plot the age distribution using a dotplot
ggplot(Marriage, aes(x = age)) +
  geom_dotplot() +
  labs(title = "Participants by age",
       y = "Proportion",
       x = "Age")
```

The `fill` and `color` options can be used to specify the fill and border color of each dot respectively.

```
# Plot ages as a dot plot using
# gold dots with black borders
ggplot(Marriage, aes(x = age)) +
  geom_dotplot(fill = "gold",
               color = "black") +
  labs(title = "Participants by age",
       y = "Proportion",
       x = "Age")
```

There are many more options available. See the help for details and examples.

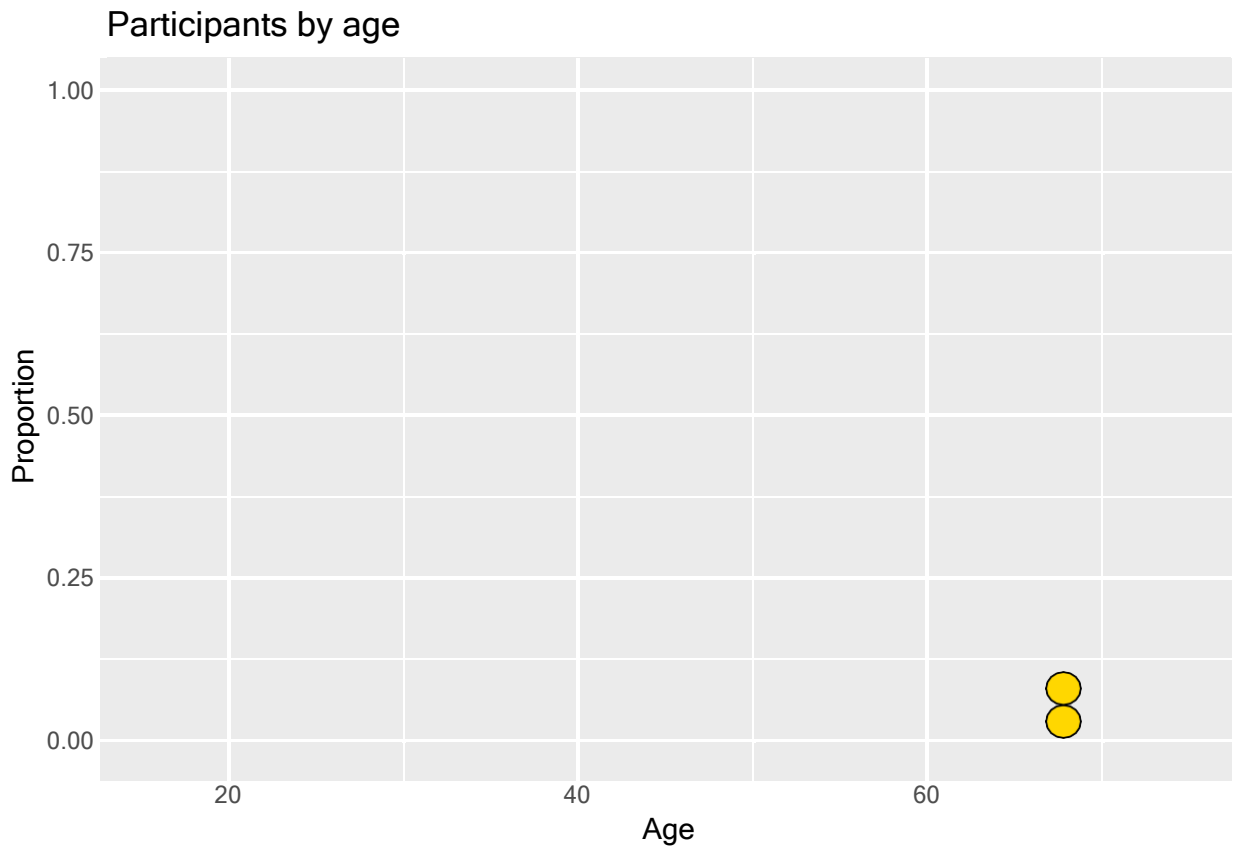


Figure 3.22: Dotplot with a specified color scheme

Chapter 4

Bivariate Graphs

Bivariate graphs display the relationship between two variables. The type of graph will depend on the measurement level of the variables (categorical or quantitative).

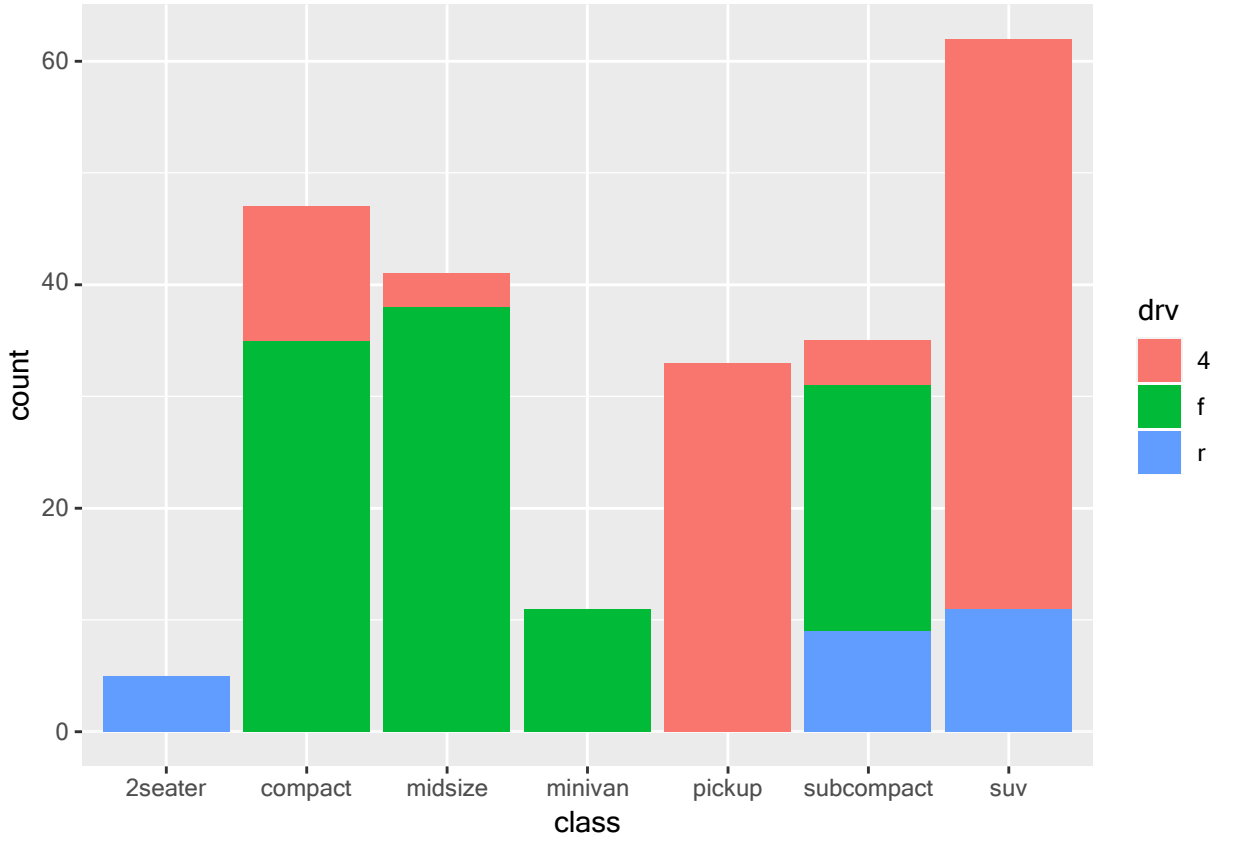
4.1 Categorical vs. Categorical

When plotting the relationship between two categorical variables, stacked, grouped, or segmented bar charts are typically used. A less common approach is the mosaic chart.

4.1.1 Stacked bar chart

Let's plot the relationship between automobile class and drive type (front-wheel, rear-wheel, or 4-wheel drive) for the automobiles in the Fuel economy dataset.

```
library(ggplot2)

# stacked bar chart
ggplot(mpg,
       aes(x = class,
           fill = drv)) +
  geom_bar(position = "stack")
```



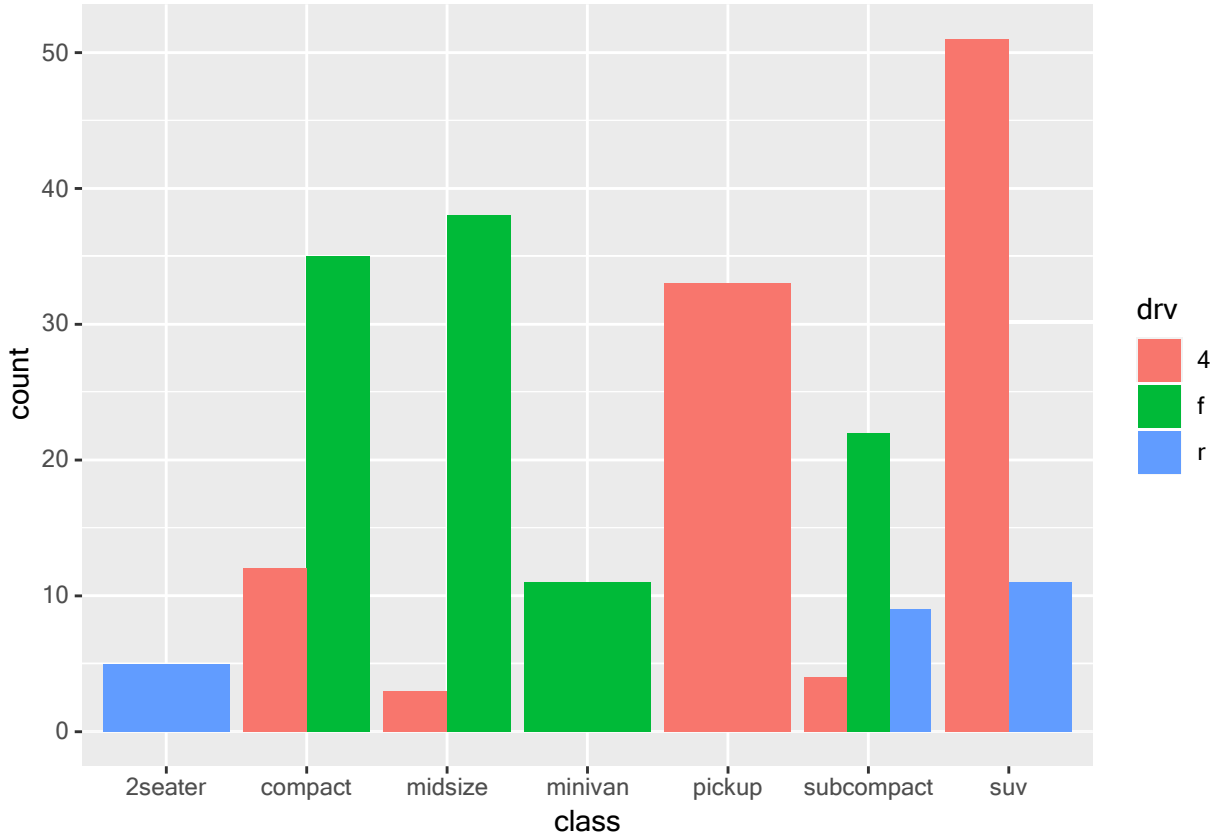
From the chart, we can see for example, that the most common vehicle is the SUV. All 2seater cars are rear wheel drive, while most, but not all SUVs are 4-wheel drive.

Stacked is the default, so the last line could have also been written as `geom_bar()`.

4.1.2 Grouped bar chart

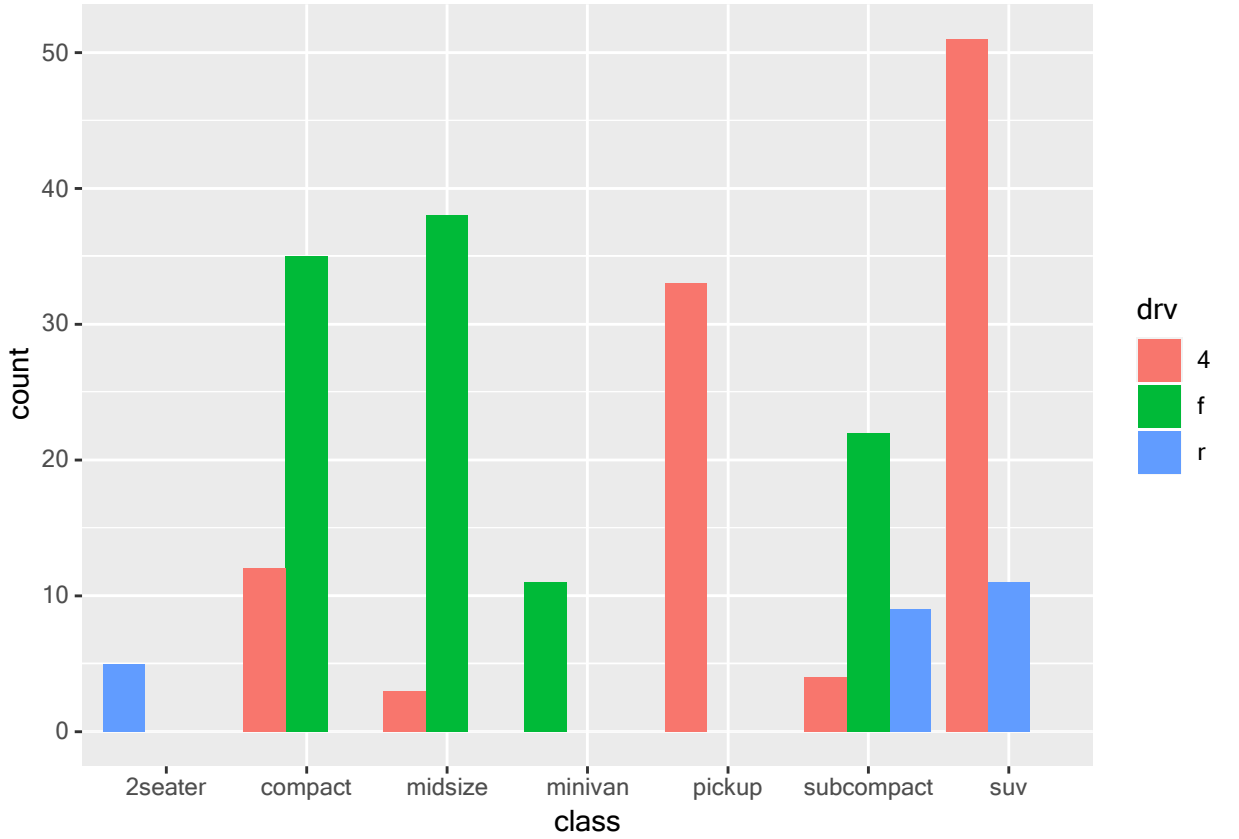
Grouped bar charts place bars for the second categorical variable side-by-side. To create a grouped bar plot use the `position = "dodge"` option.

```
library(ggplot2)

# grouped bar plot
ggplot(mpg,
       aes(x = class,
           fill = drv)) +
  geom_bar(position = "dodge")
```



Notice that all Minivans are front-wheel drive. By default, zero count bars are dropped and the remaining bars are made wider. This may not be the behavior you want. You can modify this using the `position = position_dodge(preserve = "single")` option.

```
library(ggplot2)

# grouped bar plot preserving zero count bars
ggplot(mpg,
       aes(x = class,
           fill = drv)) +
  geom_bar(position = position_dodge(preserve = "single"))
```



Note that this option is only available in the latest development version of `ggplot2`, but should should be generally available shortly.

4.1.3 Segmented bar chart

A segmented bar plot is a stacked bar plot where each bar represents 100 percent. You can create a segmented bar chart using the `position = "filled"` option.

```
library(ggplot2)

# bar plot, with each bar representing 100%
ggplot(mpg,
       aes(x = class,
           fill = drv)) +
  geom_bar(position = "fill") +
  labs(y = "Proportion")
```

This type of plot is particularly useful if the goal is to compare the percentage of a category in one variable across each level of another variable. For example, the proportion of front-wheel drive cars go up as you move from compact, to midsize, to minivan.

4.1.4 Improving the color and labeling

You can use additional options to improve color and labeling. In the graph below

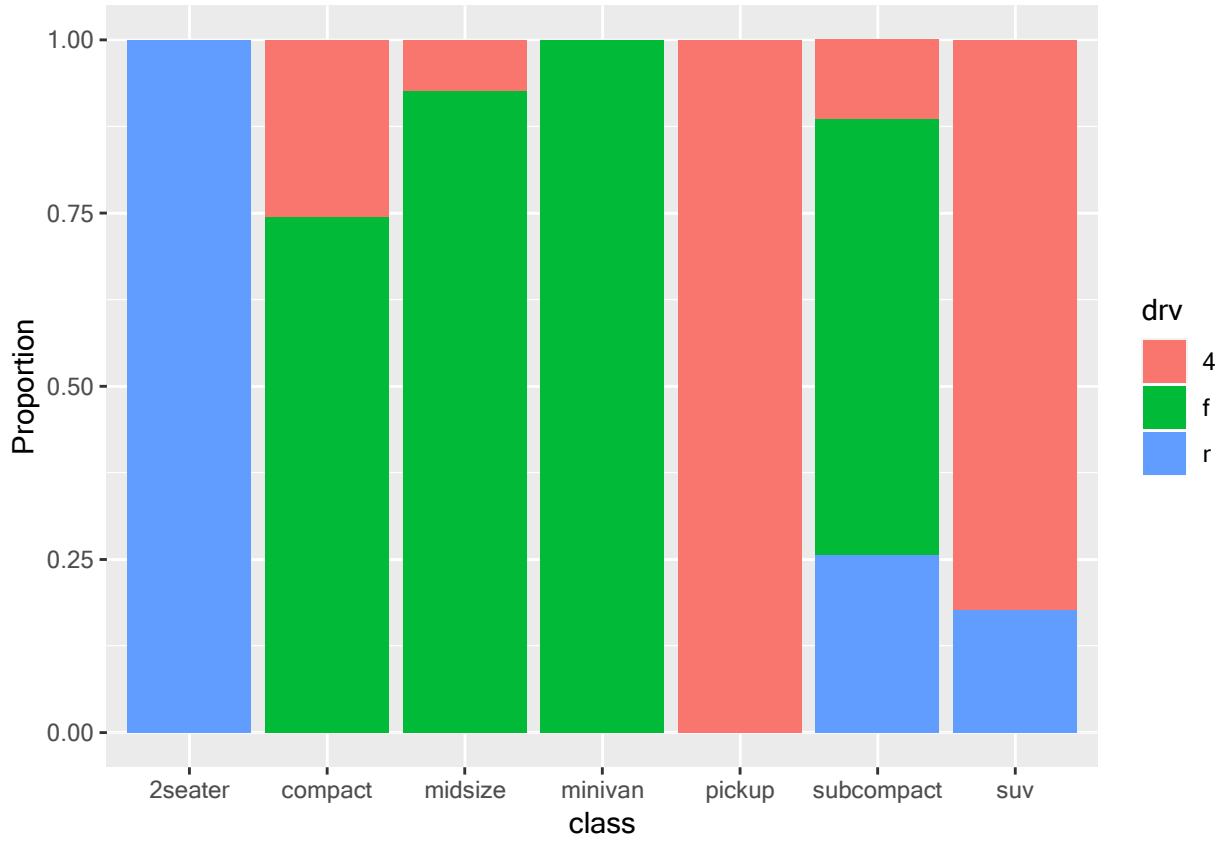
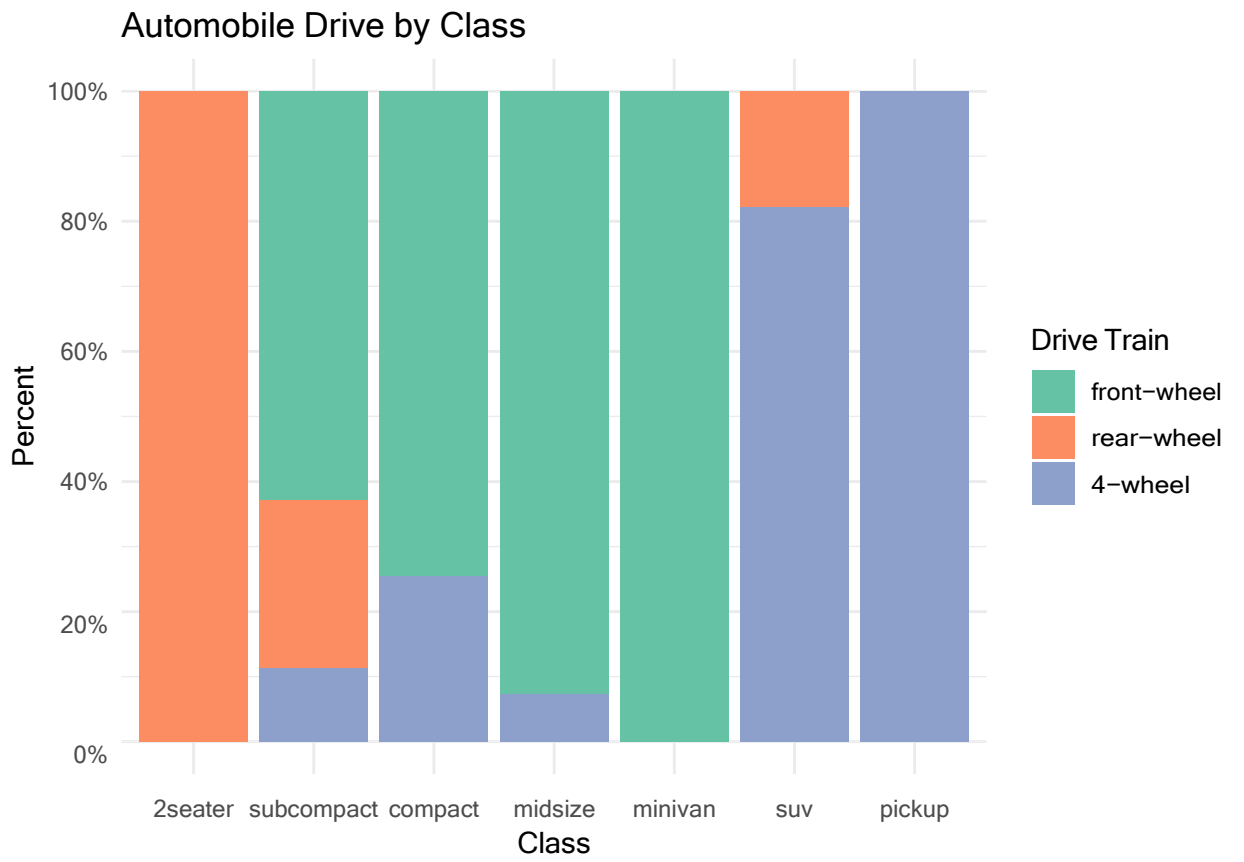


Figure 4.1: Segmented bar chart

- `factor` modifies the order of the categories for the class variable and both the order and the labels for the drive variable
- `scale_y_continuous` modifies the y-axis tick mark labels
- `labs` provides a title and changed the labels for the x and y axes and the legend
- `scale_fill_brewer` changes the fill color scheme
- `theme_minimal` removes the grey background and changed the grid color

```
library(ggplot2)

# bar plot, with each bar representing 100%,
# reordered bars, and better labels and colors
library(scales)
ggplot(mpg,
       aes(x = factor(class,
                      levels = c("2seater", "subcompact",
                                 "compact", "midsize",
                                 "minivan", "suv", "pickup")),
           fill = factor(drv,
                         levels = c("f", "r", "4"),
                         labels = c("front-wheel",
                                    "rear-wheel",
                                    "4-wheel")))) +
  geom_bar(position = "fill") +
  scale_y_continuous(breaks = seq(0, 1, .2),
                     label = percent) +
  scale_fill_brewer(palette  =  "Set2")  +
  labs(y = "Percent",
       fill = "Drive Train",
       x = "Class",
       title = "Automobile Drive by Class") +
  theme_minimal()
```



In the graph above, the `factor` function was used to reorder and/or rename the levels of a categorical variable. You could also apply this to the original dataset, making these changes permanent. It would then apply to all future graphs using that dataset. For example:

```
# change the order the levels for the categorical variable "class"
mpg$class = factor(mpg$class,
                   levels = c("2seater", "subcompact",
                              "compact", "midsize",
                              "minivan", "suv", "pickup")
```

I placed the `factor` function within the `ggplot` function to demonstrate that, if desired, you can change the order of the categories and labels for the categories for a single graph.

The other functions are discussed more fully in the section on Customizing graphs.

Next, let's add percent labels to each segment. First, we'll create a summary dataset that has the necessary labels.

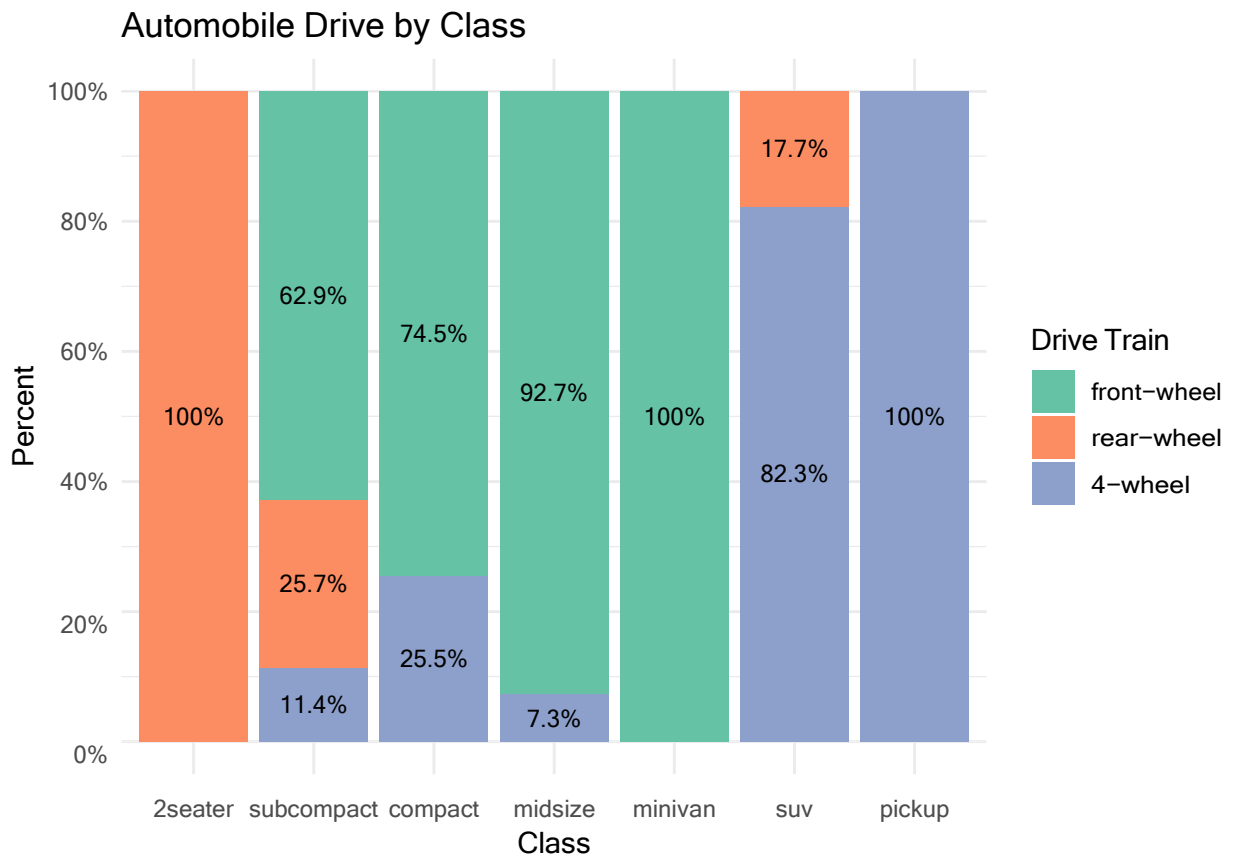
```
# create a summary dataset
library(dplyr)
plotdata <- mpg %>%
  group_by(class, drv) %>%
  summarize(n = n()) %>%
  mutate(pct = n/sum(n),
         lbl = scales::percent(pct))
plotdata
```

```
## # A tibble: 12 x 5
```

```
## # Groups:   class [7]
##    class      drv       n    pct lbl
##    <chr>      <chr> <int>  <dbl> <chr>
##  1 2seater    r         5 1.00   100%
##  2 compact    4        12 0.255  25.5%
##  3 compact    f        35 0.745  74.5%
##  4 midsize    4         3 0.0732 7.3%
##  5 midsize    f        38 0.927  92.7%
##  6 minivan    f        11 1.00   100%
##  7 pickup     4        33 1.00   100%
##  8 subcompact 4         4 0.114  11.4%
##  9 subcompact f        22 0.629  62.9%
## 10 subcompact r         9 0.257  25.7%
## 11 suv        4        51 0.823  82.3%
## 12 suv        r        11 0.177  17.7%
```

Next, we'll use this dataset and the `geom_text` function to add labels to each bar segment.

```
# create segmented bar chart
# adding labels to each segment

ggplot(plotdata,
       aes(x = factor(class,
                      levels = c("2seater", "subcompact",
                                 "compact", "midsize",
                                 "minivan", "suv", "pickup")),
           y = pct,
           fill = factor(drv,
                         levels = c("f", "r", "4"),
                         labels = c("front-wheel",
                                    "rear-wheel",
                                    "4-wheel")))) +
  geom_bar(stat = "identity",
           position = "fill") +
  scale_y_continuous(breaks = seq(0, 1, .2),
                     label = percent) +
  geom_text(aes(label = lbl),
            size = 3,
            position = position_stack(vjust = 0.5)) +
  scale_fill_brewer(palette = "Set2") +
  labs(y = "Percent",
       fill = "Drive Train",
       x = "Class",
       title = "Automobile Drive by Class") +
  theme_minimal()
```



Now we have a graph that is easy to read and interpret.

4.1.5 Other plots

Mosaic plots provide an alternative to stacked bar charts for displaying the relationship between categorical variables. They can also provide more sophisticated statistical information.

4.2 Quantitative vs. Quantitative

The relationship between two quantitative variables is typically displayed using scatterplots and line graphs.

4.2.1 Scatterplot

The simplest display of two quantitative variables is a scatterplot, with each variable represented on an axis. For example, using the Salaries dataset, we can plot experience (*yrs.since.phd*) vs. academic salary (*salary*) for college professors.

```
library(ggplot2)
data(Salaries, package="carData")

# simple scatterplot
ggplot(Salaries,
       aes(x = yrs.since.phd,
```

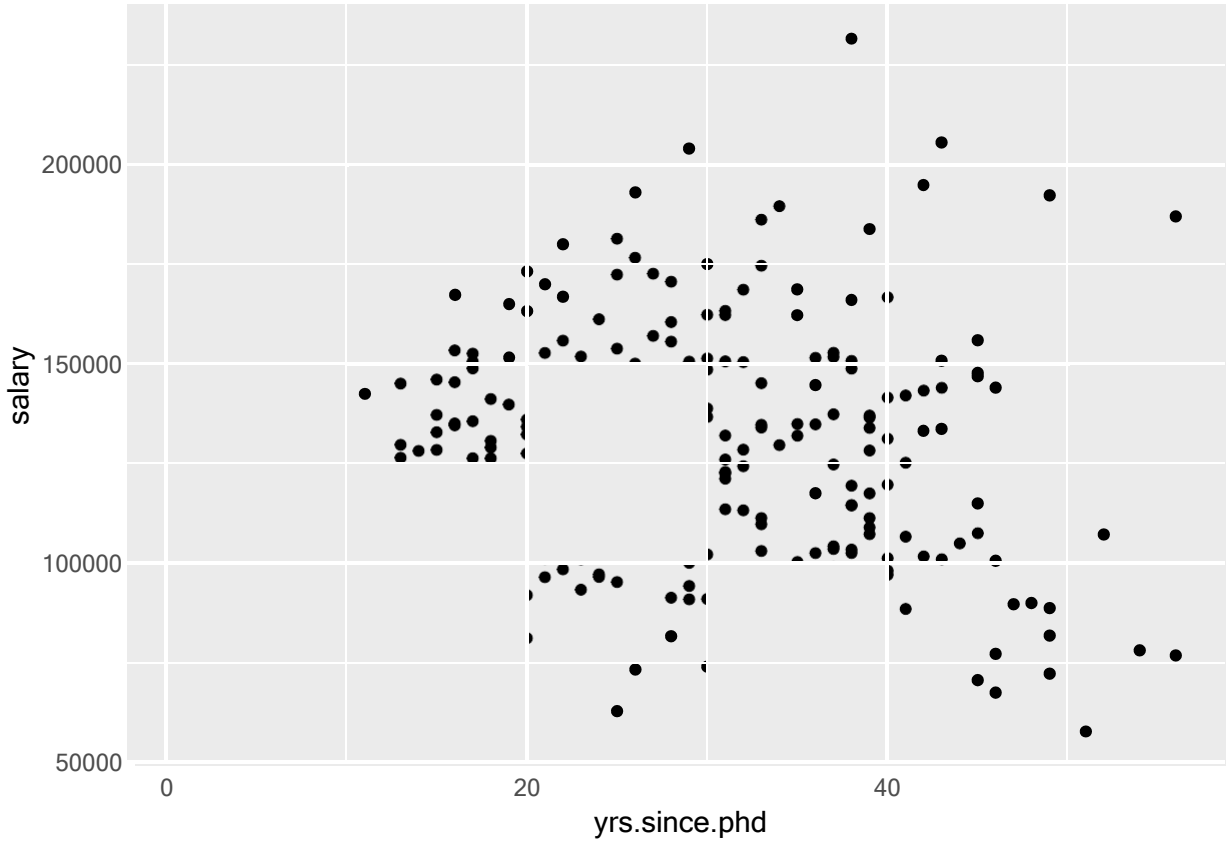


Figure 4.2: Simple scatterplot

```
          y = salary)) +
  geom_point()
```

`geom_point` options can be used to change the

- `color` - point color
- `size` - point size
- `shape` - point shape
- `alpha` - point transparency. Transparency ranges from 0 (transparent) to 1 (opaque), and is a useful parameter when points overlap.

The functions `scale_x_continuous` and `scale_y_continuous` control the scaling on x and y axes respectively.

See Customizing graphs for details.

We can use these options and functions to create a more attractive scatterplot.

```
# enhanced scatter plot
ggplot(Salaries,
       aes(x = yrs.since.phd,
           y = salary)) +
  geom_point(color="cornflowerblue",
```

Figure 4.3: Scatterplot with color, transparency, and axis scaling

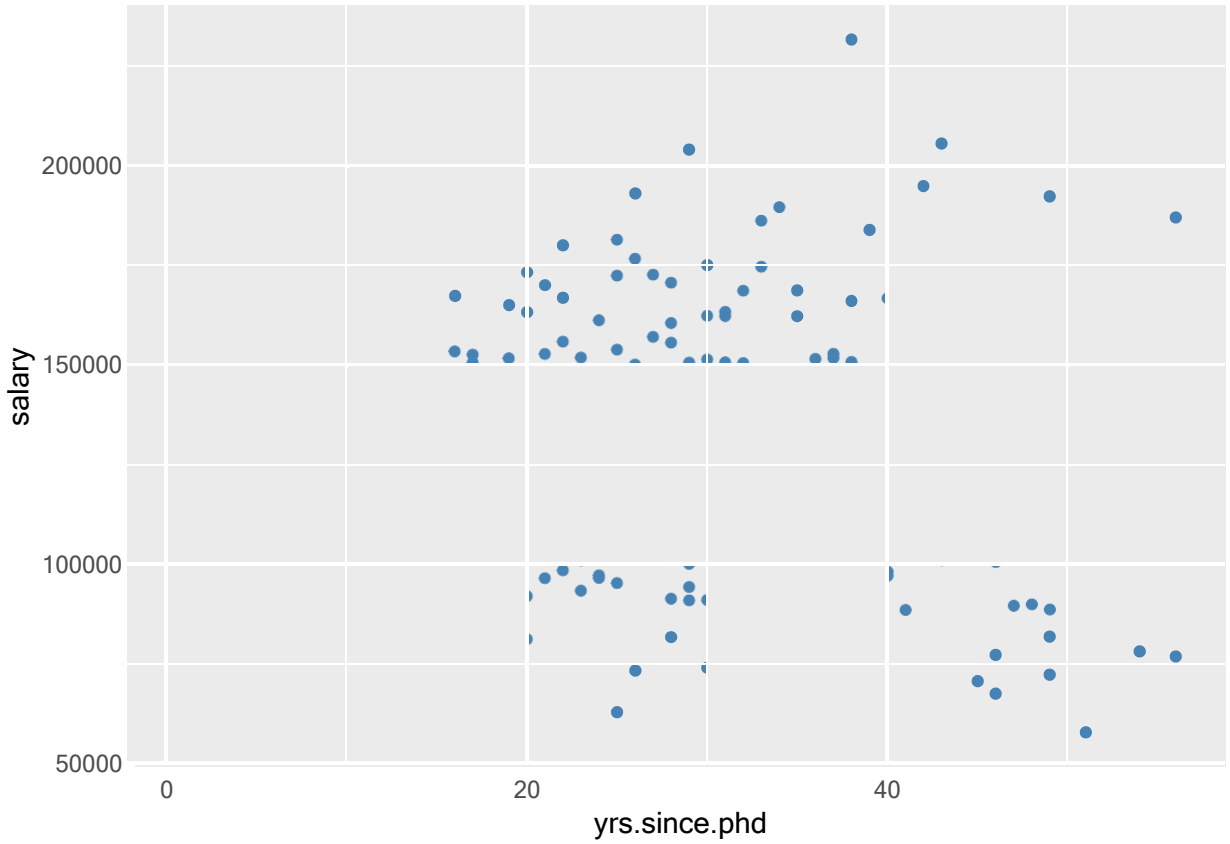
```
            size = 2,
            alpha=.8) +
  scale_y_continuous(label = scales::dollar,
                     limits = c(50000, 250000)) +
  scale_x_continuous(breaks = seq(0, 60, 10),
                     limits=c(0, 60)) +
  labs(x = "Years Since PhD",
       y = "",
       title = "Experience vs. Salary",
       subtitle = "9-month salary for 2008-2009")
```

4.2.1.1 Adding best fit lines

It is often useful to summarize the relationship displayed in the scatterplot, using a best fit line. Many types of lines are supported, including linear, polynomial, and nonparametric (loess). By default, 95% confidence limits for these lines are displayed.

```
# scatterplot with linear fit line
ggplot(Salaries,
       aes(x = yrs.since.phd,
           y = salary)) +
```

```
  geom_point(color= "steelblue") +
  geom_smooth(method = "lm")
```



Clearly, salary increases with experience. However, there seems to be a dip at the right end - professors with significant experience, earning lower salaries. A straight line does not capture this non-linear effect. A line with a bend will fit better here.

A polynomial regression line provides a fit line of the form

$$\hat{y} = \beta_0 + \beta_1 x + \beta 2 x^2 + \beta 3 x^3 + \beta 4 x^4 + \ldots$$

Typically either a quadratic (one bend), or cubic (two bends) line is used. It is rarely necessary to use a higher order(>3) polynomials. Applying a quadratic fit to the salary dataset produces the following result.

```
# scatterplot with quadratic line of best fit
ggplot(Salaries,
       aes(x = yrs.since.phd,
           y = salary)) +
  geom_point(color= "steelblue") +
  geom_smooth(method = "lm",
              formula = y ~ poly(x, 2),
              color = "indianred3")
```

Finally, a smoothed nonparametric fit line can often provide a good picture of the relationship. The default in ggplot2 is a loess line which stands for for **l**ocally **w**eighted **s**catterplot **s**moothing.

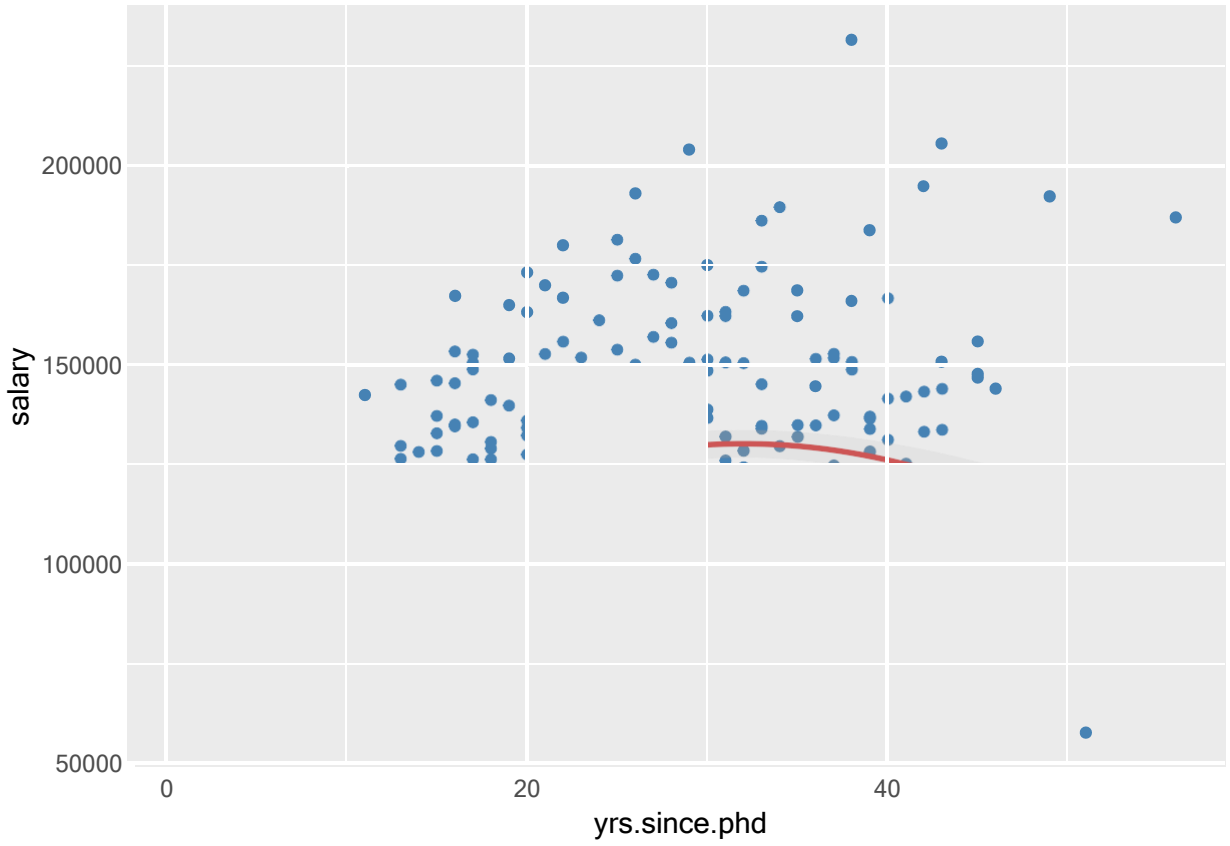


Figure 4.4: Scatterplot with quadratic fit line

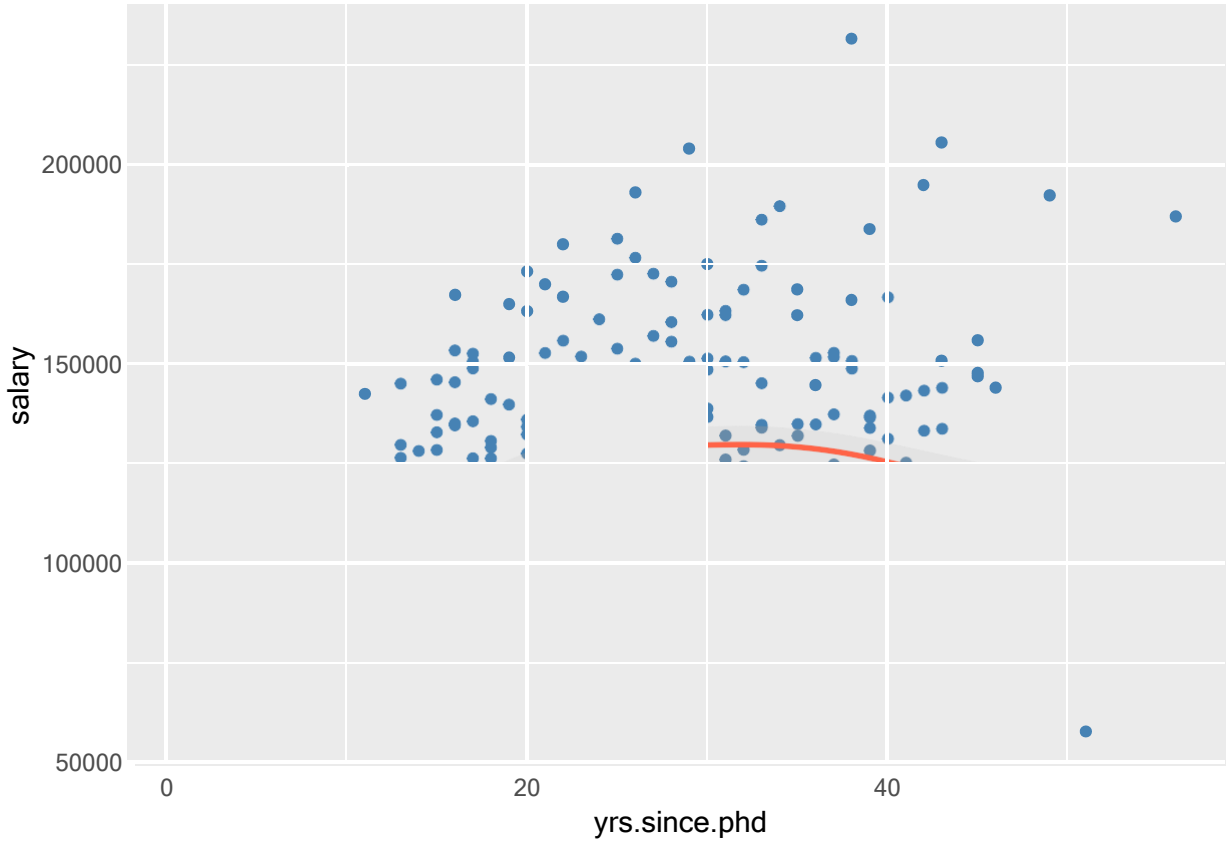


Figure 4.5: Scatterplot with nonparametric fit line

```
# scatterplot with loess smoothed line
ggplot(Salaries,
       aes(x = yrs.since.phd,
           y = salary)) +
  geom_point(color="steelblue") +
  geom_smooth(color = "tomato")
```

You can suppress the confidence bands by including the option `se = FALSE`.

Here is a complete (and more attractive) plot.

```
# scatterplot with loess smoothed line
# and better labeling and color
ggplot(Salaries,
       aes(x = yrs.since.phd,
           y = salary)) +
  geom_point(color="cornflowerblue",
             size = 2,
             alpha = .6) +
  geom_smooth(size = 1.5,
              color = "darkgrey") +
  scale_y_continuous(label = scales::dollar,
                     limits = c(50000, 250000)) +
```

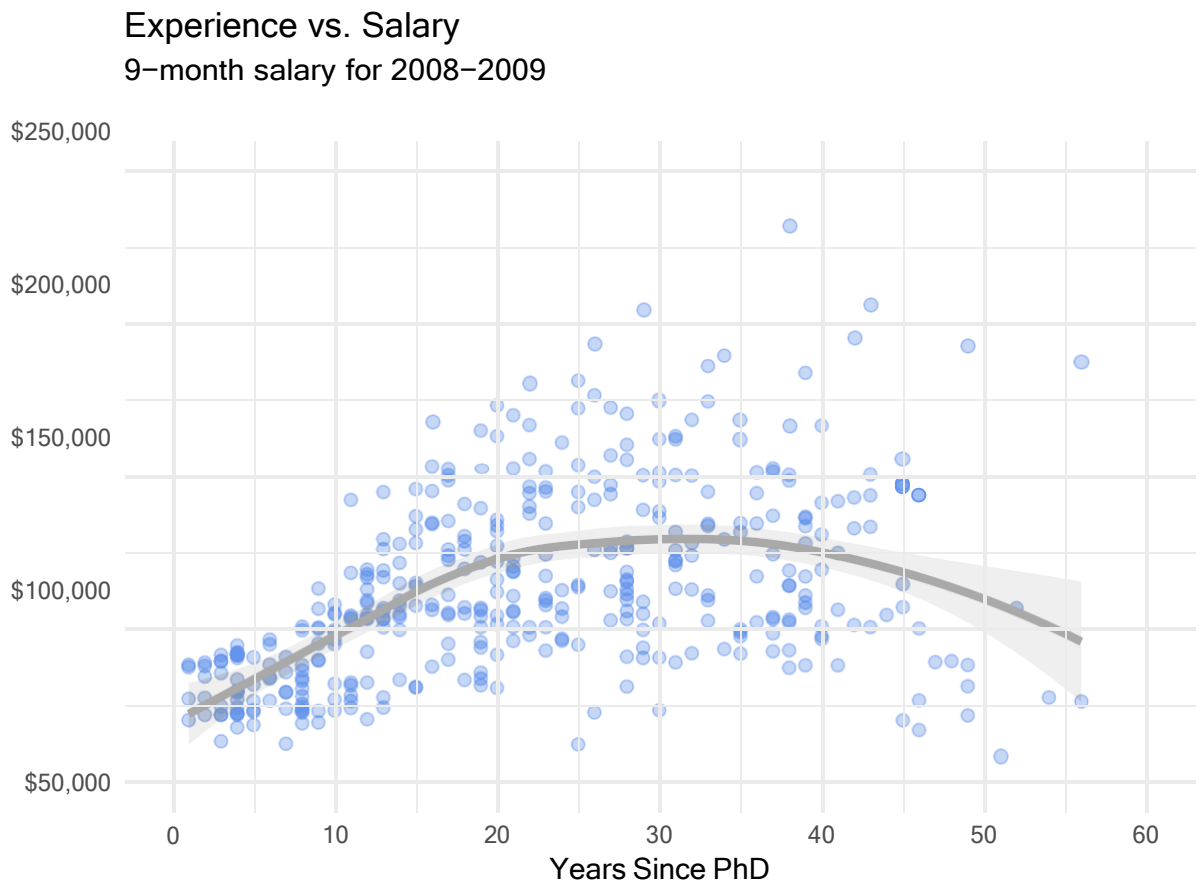


Figure 4.6: Scatterplot with nonparametric fit line

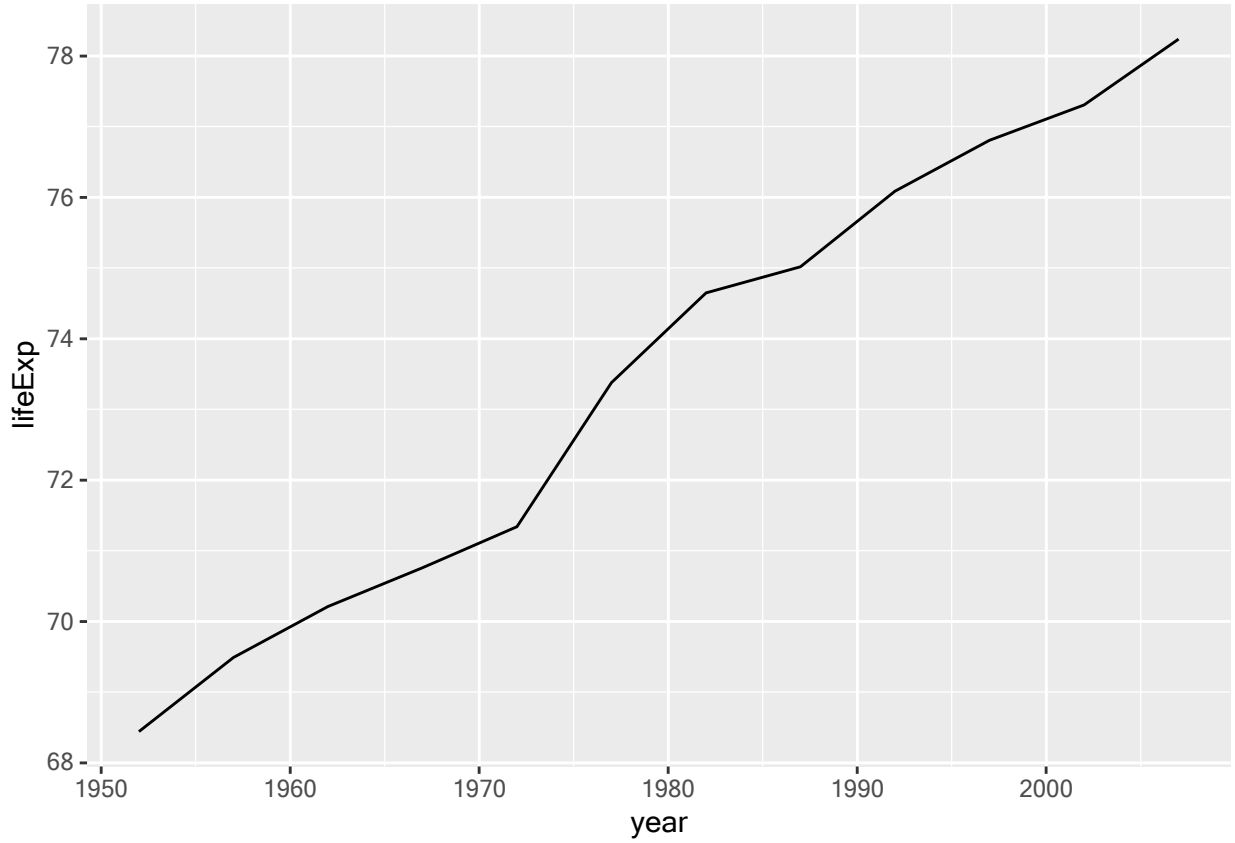
```
  scale_x_continuous(breaks = seq(0, 60, 10),
                     limits = c(0, 60)) +
  labs(x = "Years Since PhD",
       y = "",
       title = "Experience vs. Salary",
       subtitle = "9-month salary for 2008-2009") +
  theme_minimal()
```

4.2.2 Line plot

When one of the two variables represents time, a line plot can be an effective method of displaying relationship. For example, the code below displays the relationship between time (*year*) and life expectancy (*lifeExp*) in the United States between 1952 and 2007. The data comes from the gapminder dataset.

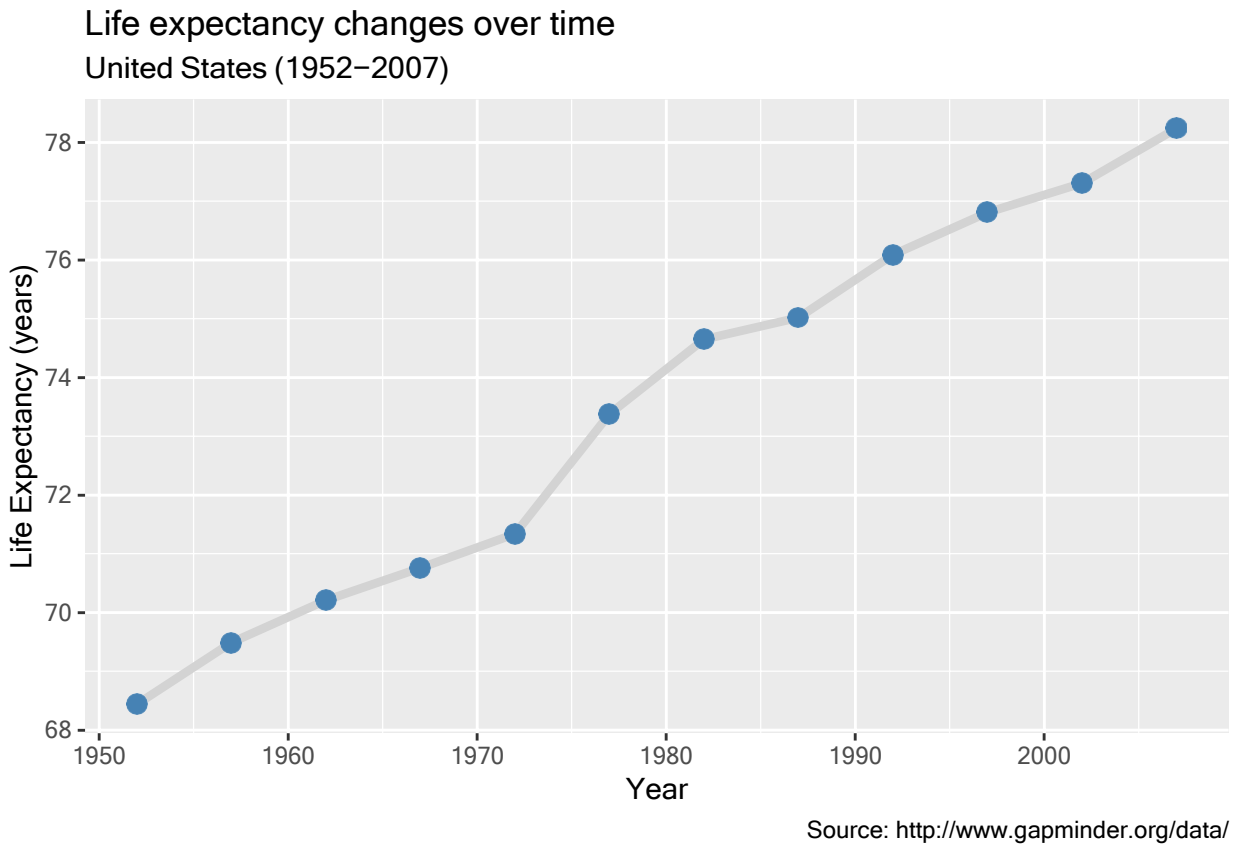
```
data(gapminder, package="gapminder")

# Select US cases
library(dplyr)
plotdata <- filter(gapminder,
                   country == "United States")
```

```
# simple line plot
ggplot(plotdata,
       aes(x = year,
           y = lifeExp)) +
  geom_line()
```



It is hard to read individual values in the graph above. In the next plot, we'll add points as well.

```
# line plot with points#
and improved labeling
ggplot(plotdata,
       aes(x = year,
           y = lifeExp)) +
  geom_line(size = 1.5,
            color = "lightgrey") +
  geom_point(size = 3,
             color = "steelblue") +
  labs(y = "Life Expectancy (years)",
       x = "Year",
       title = "Life expectancy changes over time",
       subtitle = "United States (1952-2007)",
       caption = "Source: http://www.gapminder.org/data/")
```



Time dependent data is covered in more detail under Time series. Customizing line graphs is covered in the Customizing graphs section.

4.3 Categorical vs. Quantitative

When plotting the relationship between a categorical variable and a quantitative variable, a large number of graph types are available. These include bar charts using summary statistics, grouped kernel density plots, side-by-side box plots, side-by-side violin plots, mean/sem plots, ridgeline plots, and Cleveland plots.

4.3.1 Bar chart (on summary statistics)

In previous sections, bar charts were used to display the number of cases by category for a single variable or for two variables. You can also use bar charts to display other summary statistics (e.g., means or medians) on a quantitative variable for each level of a categorical variable.

For example, the following graph displays the mean salary for a sample of university professors by their academic rank.

```
data(Salaries, package="carData")

# calculate mean salary for each rank
library(dplyr)
plotdata <- Salaries %>%
  group_by(rank) %>%
  summarize(mean_salary = mean(salary))
```

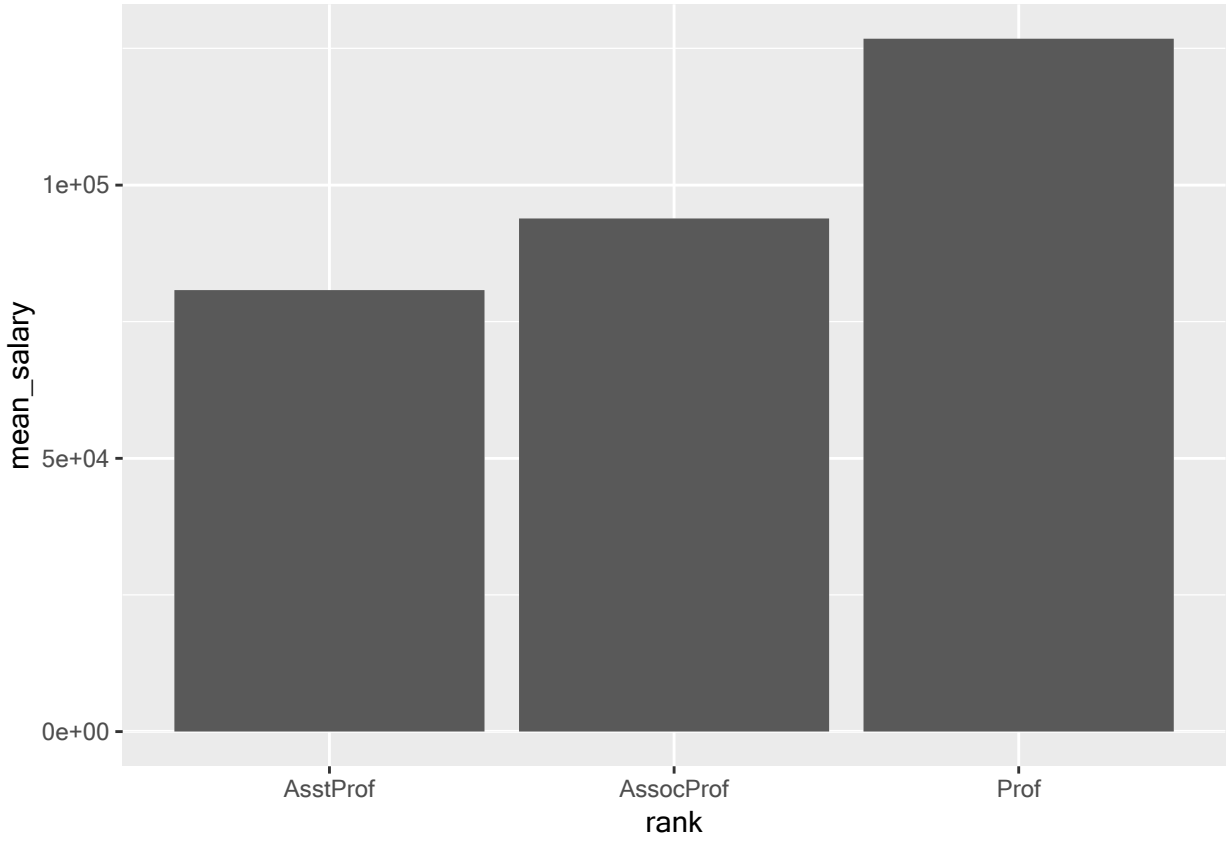


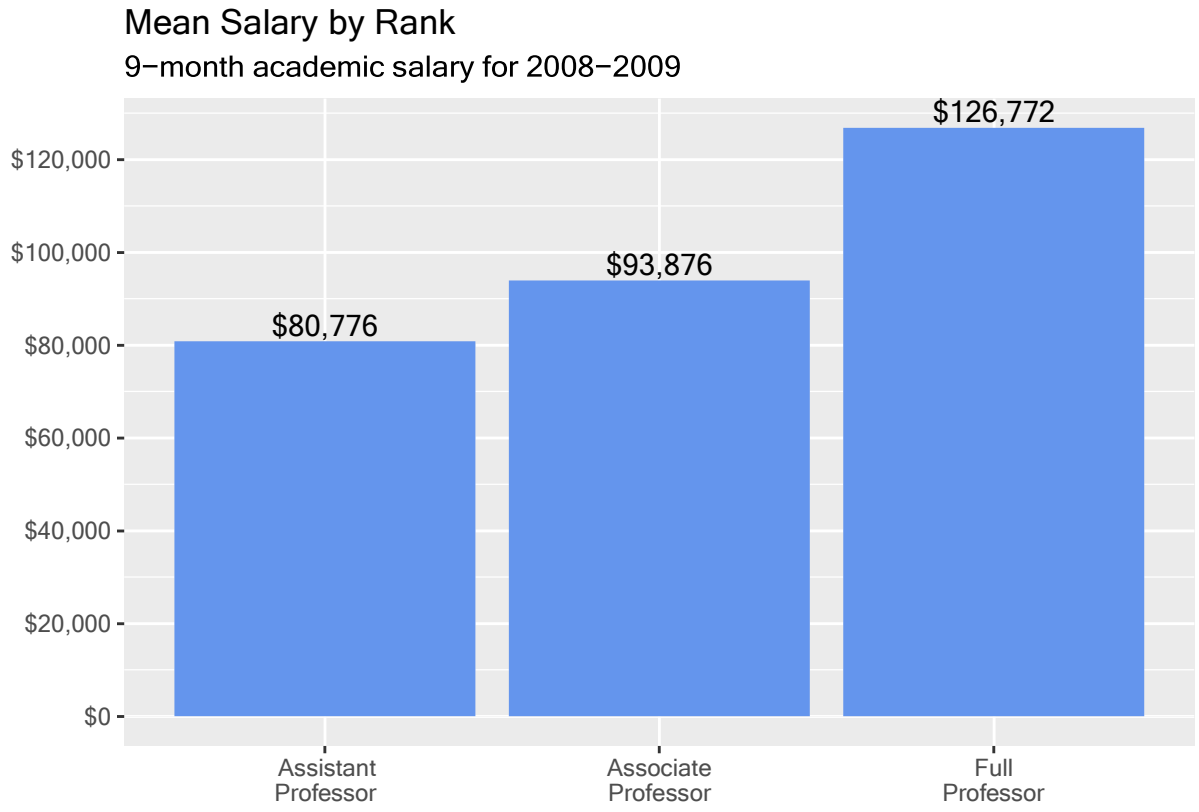
Figure 4.7: Bar chart displaying means

```
# plot mean salaries
ggplot(plotdata,
       aes(x = rank,
           y = mean_salary)) +
  geom_bar(stat = "identity")
```

We can make it more attractive with some options.

```
# plot mean salaries in a more attractive fashion
library(scales)
ggplot(plotdata,
       aes(x = factor(rank,
                      labels = c("Assistant\nProfessor",
                                 "Associate\nProfessor",
                                 "Full\nProfessor")),
                      y = mean_salary)) +
  geom_bar(stat = "identity",
           fill = "cornflowerblue") +
  geom_text(aes(label = dollar(mean_salary)),
            vjust = -0.25) +
  scale_y_continuous(breaks = seq(0, 130000, 20000),
                     label = dollar) +
```

```
  labs(title = "Mean Salary by Rank",
       subtitle = "9-month academic salary for 2008-2009",
       x = "",
       y = "")
```

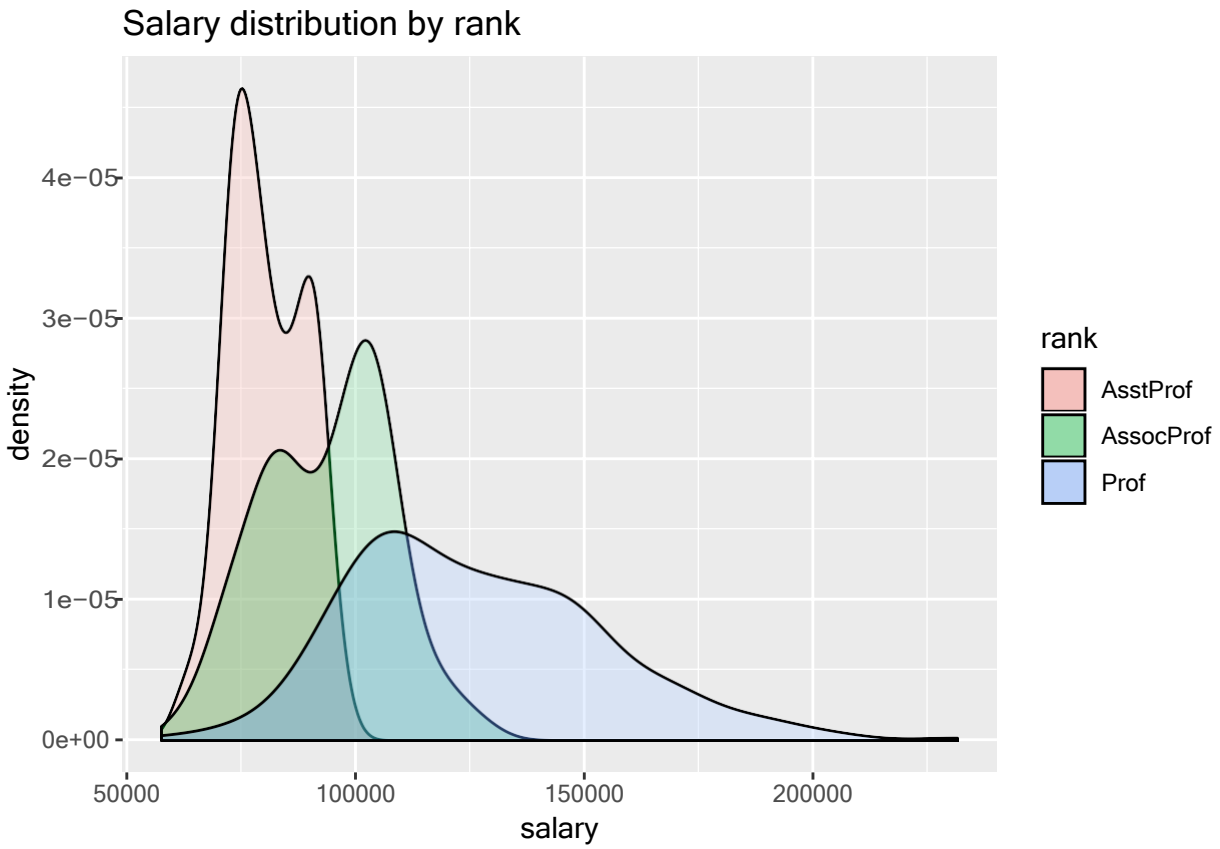


One limitation of such plots is that they do not display the distribution of the data - only the summary statistic for each group. The plots below correct this limitation to some extent.

4.3.2 Grouped kernel density plots

One can compare groups on a numeric variable by superimposing kernel density plots in a single graph.

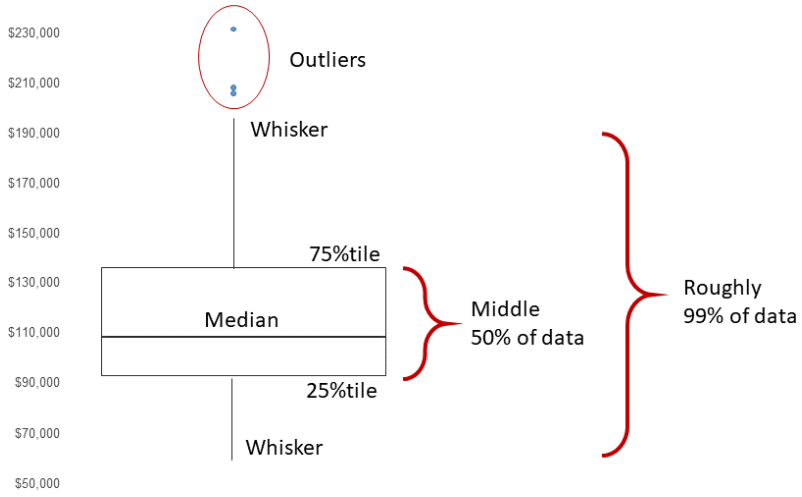
```
# plot the distribution of salaries
# by rank using kernel density plots
ggplot(Salaries,
       aes(x = salary,
           fill = rank)) +
  geom_density(alpha = 0.4) +
  labs(title = "Salary distribution by rank")
```



The `alpha` option makes the density plots partially transparent, so that we can see what is happening under the overlaps. Alpha values range from 0 (transparent) to 1 (opaque). The graph makes clear that, in general, salary goes up with rank. However, the salary range for full professors is *very* wide.

4.3.3 Box plots

A boxplot displays the 25th percentile, median, and 75th percentile of a distribution. The whiskers (vertical lines) capture roughly 99% of a normal distribution, and observations outside this range are plotted as points representing outliers (see the figure below).



Side-by-side box plots are very useful for comparing groups (i.e., the levels of a categorical variable) on a numerical variable.

```
# plot the distribution of salaries by rank using boxplots
ggplot(Salaries,
       aes(x = rank,
           y = salary)) +
  geom_boxplot() +
  labs(title = "Salary distribution by rank")
```

Notched boxplots provide an approximate method for visualizing whether groups differ. Although not a formal test, if the notches of two boxplots do not overlap, there is strong evidence (95% confidence) that the medians of the two groups differ.

```
# plot the distribution of salaries by rank using boxplots
ggplot(Salaries, aes(x = rank,
                     y = salary)) +
  geom_boxplot(notch = TRUE,
               fill = "cornflowerblue",
               alpha = .7) +
  labs(title = "Salary distribution by rank")
```

In the example above, all three groups appear to differ.

One of the advantages of boxplots is that their widths are not usually meaningful. This allows you to compare the distribution of many groups in a single graph.

4.3.4 Violin plots

Violin plots are similar to kernel density plots, but are mirrored and rotated 90°.

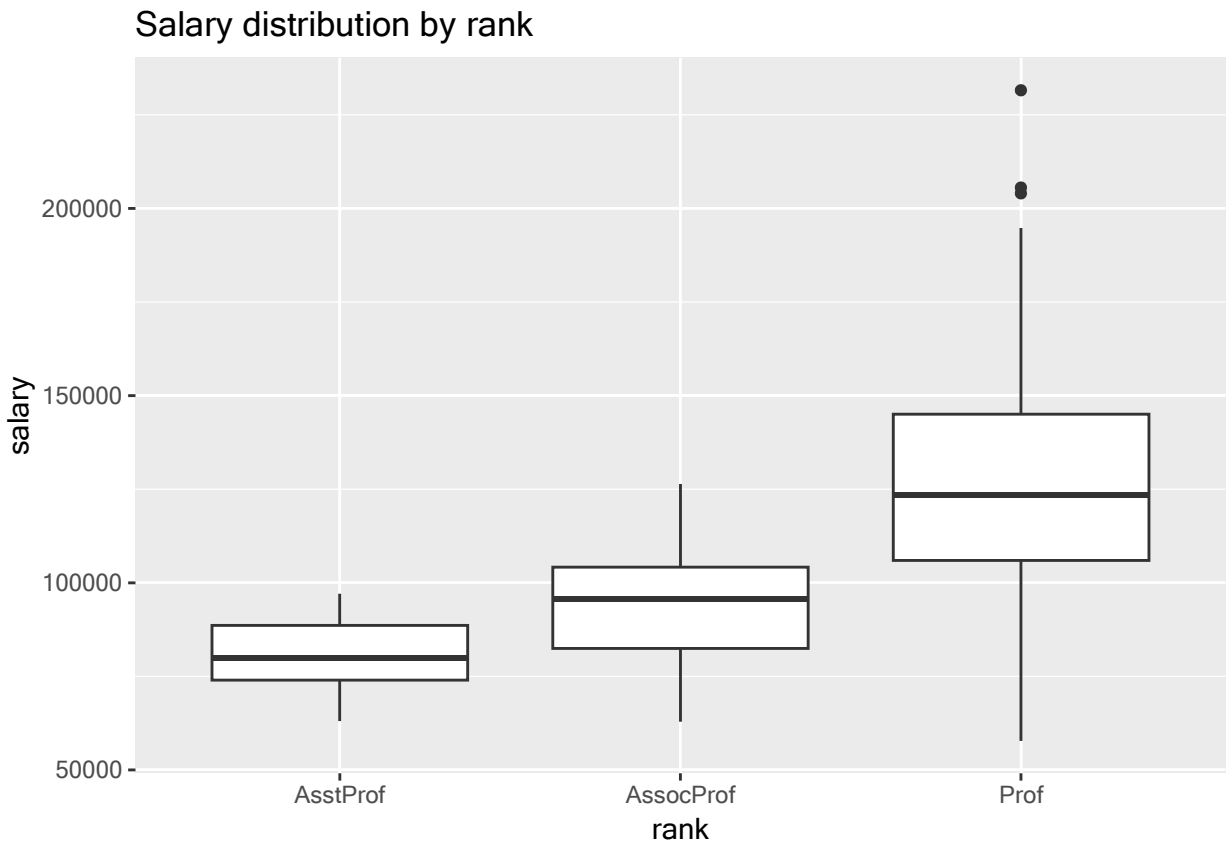


Figure 4.8: Side-by-side boxplots

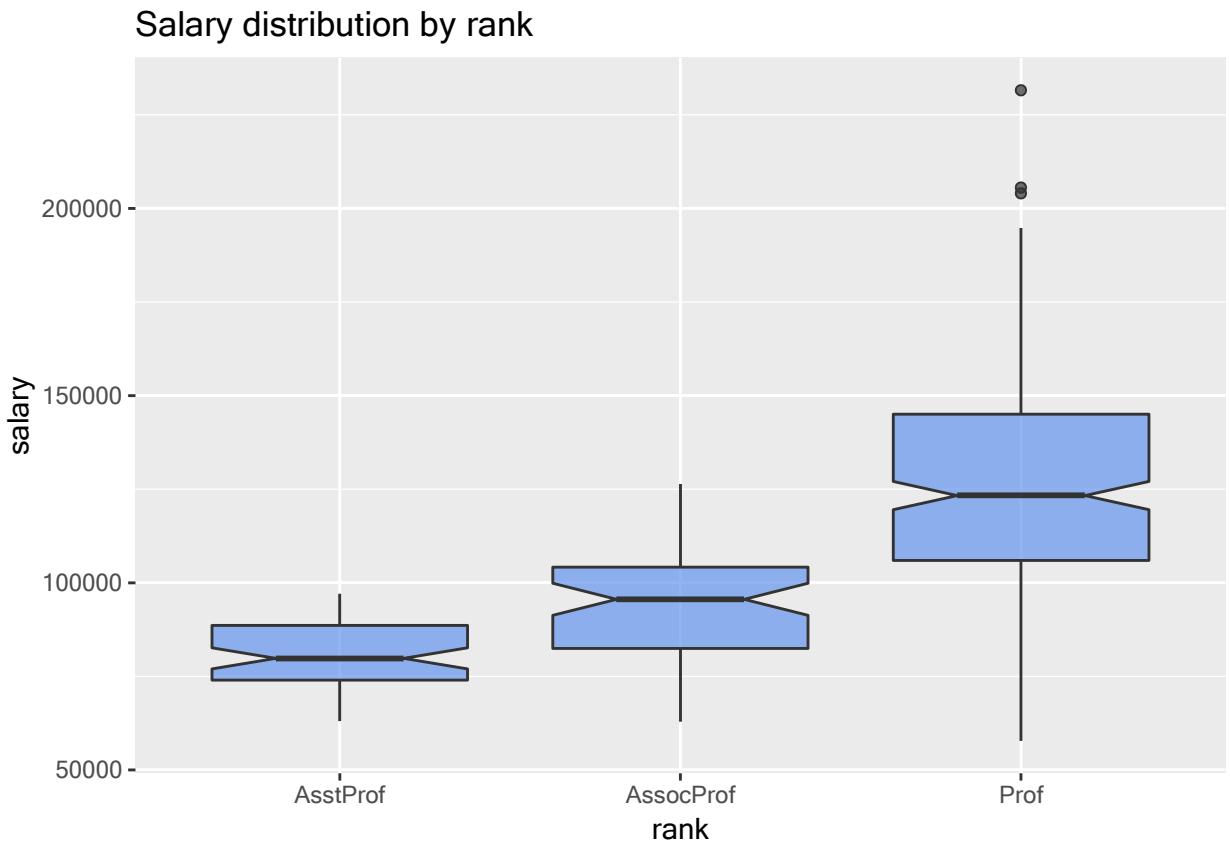
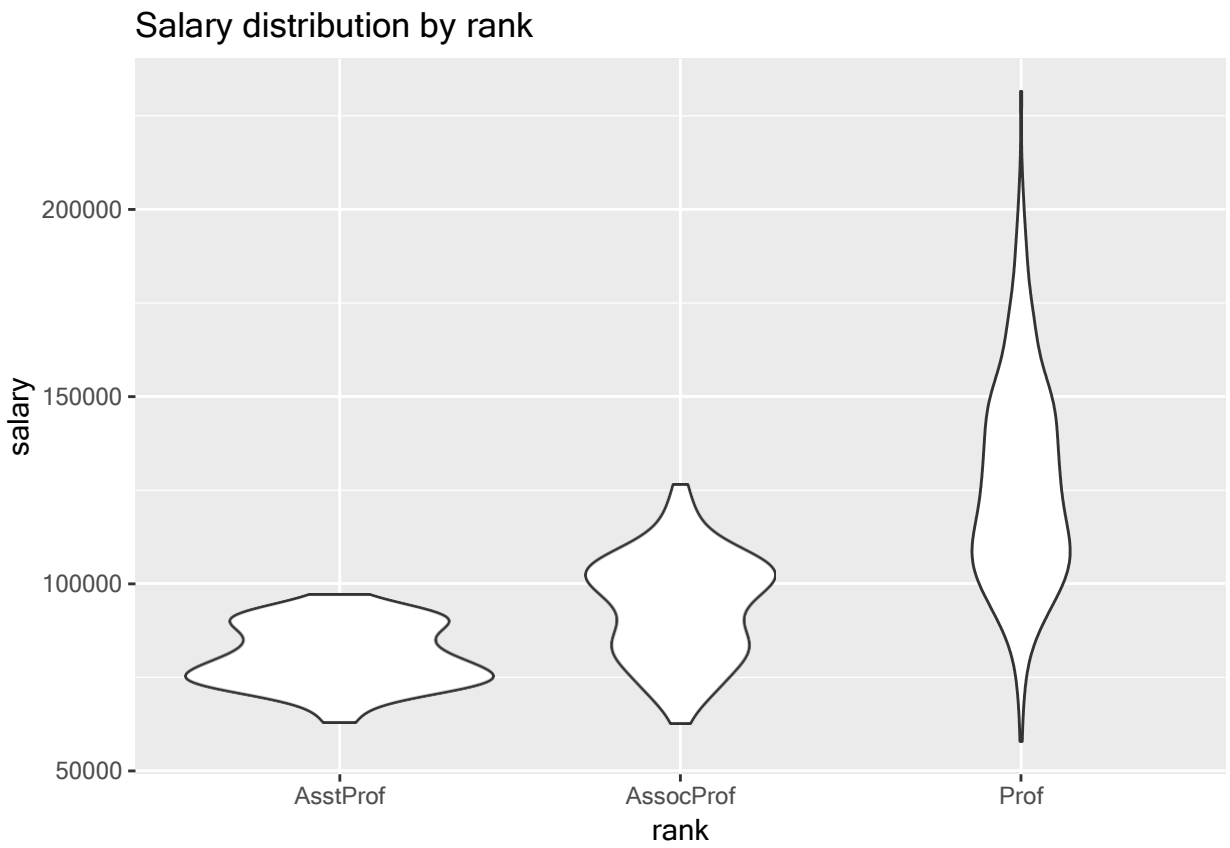


Figure 4.9: Side-by-side notched boxplots

```
# plot the distribution of salaries
# by rank using violin plots
ggplot(Salaries,
       aes(x = rank,
           y = salary)) +
  geom_violin() +
  labs(title = "Salary distribution by rank")
```



A useful variation is to superimpose boxplots on violin plots.

```
# plot the distribution using violin and boxplots
ggplot(Salaries,
       aes(x = rank,
           y = salary)) +
  geom_violin(fill = "cornflowerblue") +
  geom_boxplot(width = .2,
               fill = "orange",
               outlier.color = "orange",
               outlier.size = 2) +
  labs(title = "Salary distribution by rank")
```

4.3.5 Ridgeline plots

A ridgeline plot (also called a joyplot) displays the distribution of a quantitative variable for several groups. They're similar to kernel density plots with vertical faceting, but take up less room. Ridgeline plots are

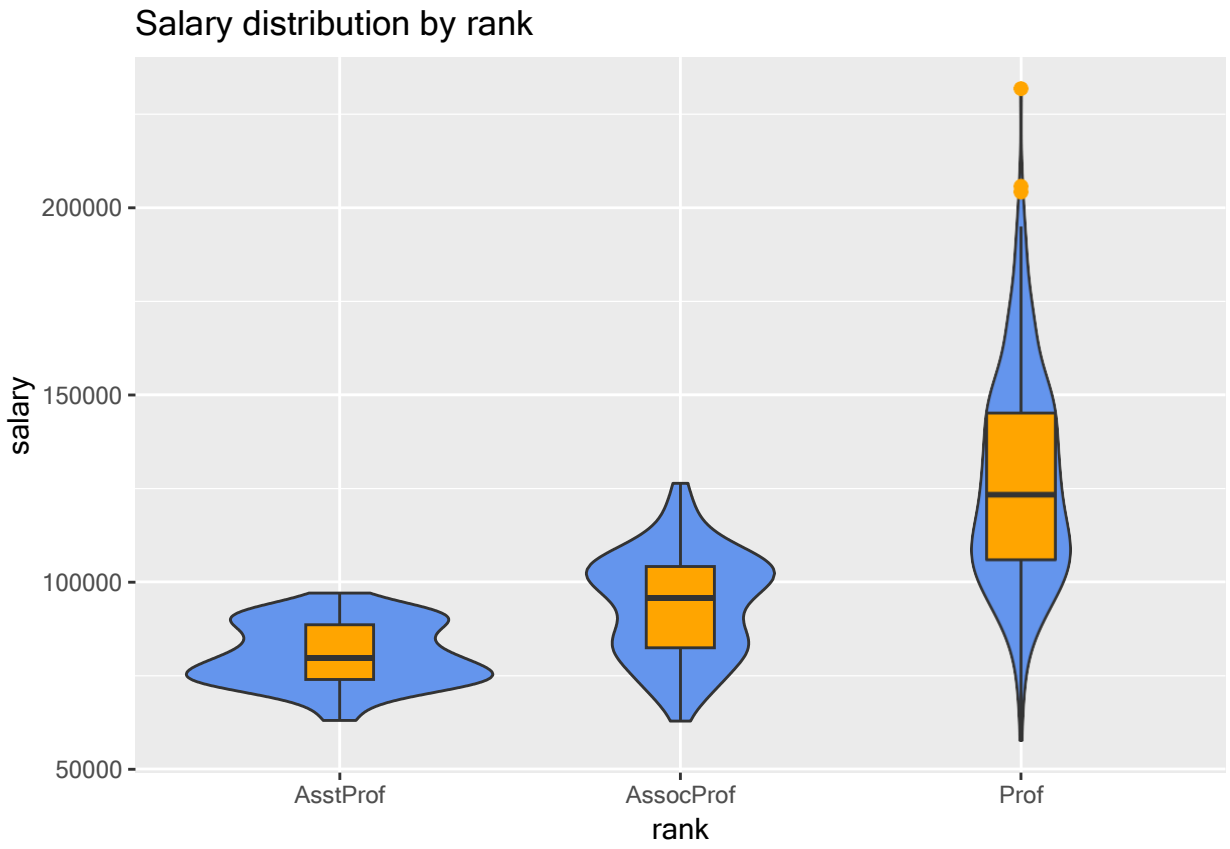


Figure 4.10: Side-by-side violin/box plots

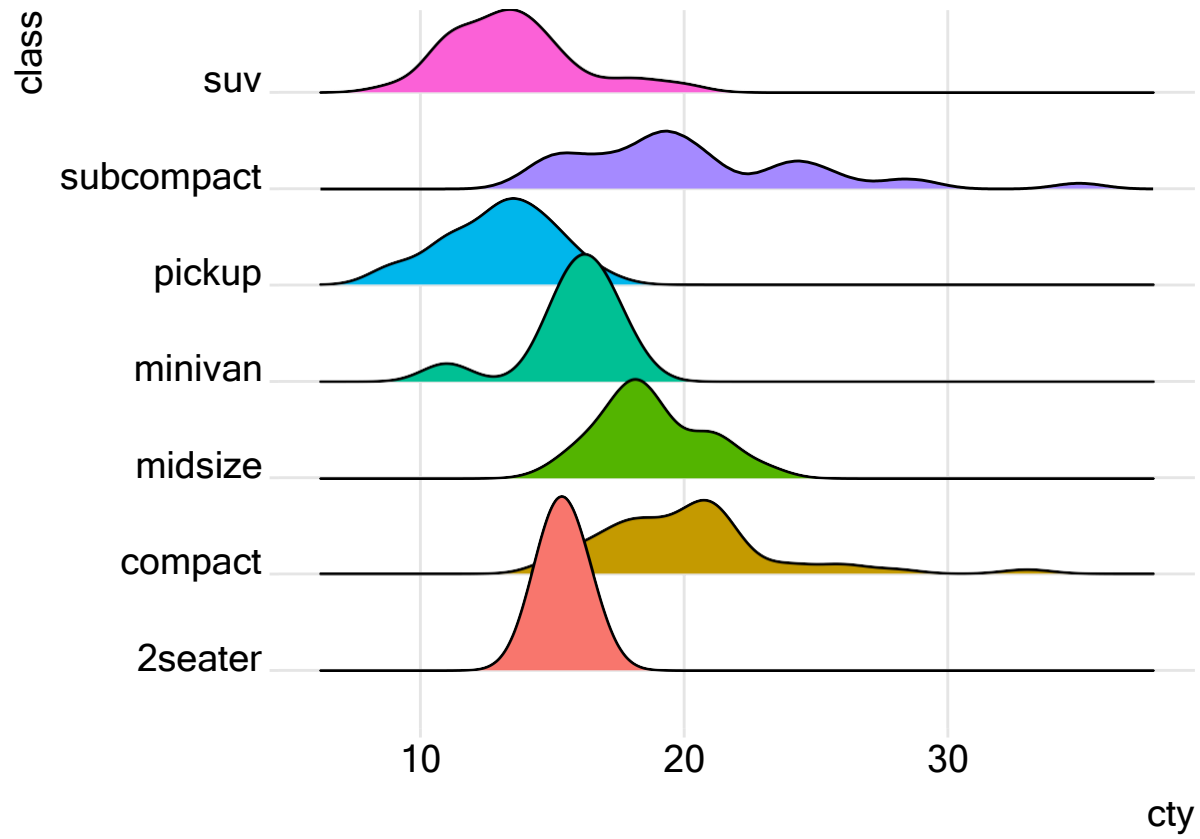


Figure 4.11: Ridgeline graph with color fill

created with the `ggridges` package.

Using the Fuel economy dataset, let's plot the distribution of city driving miles per gallon by car class.

```
# create ridgeline graph
library(ggplot2)
library(ggridges)

ggplot(mpg,
       aes(x = cty,
           y = class,
           fill = class)) +
  geom_density_ridges() +
  theme_ridges() +
  labs("Highway mileage by auto class") +
  theme(legend.position = "none")
```

I've suppressed the legend here because it's redundant (the distributions are already labeled on the y-axis). Unsurprisingly, pickup trucks have the poorest mileage, while subcompacts and compact cars tend to achieve ratings. However, there is a very wide range of gas mileage scores for these smaller cars.

Note the the possible overlap of distributions is the trade-off for a more compact graph. You can add transparency if the the overlap is severe using `geom_density_ridges(alpha = n)`, where n ranges from 0 (transparent) to 1 (opaque). See the package vingnette for more details.

Table 4.1: Plot data

rank	n	mean	sd	se	ci
AsstProf	67	80775.99	8174.113	998.6268	1993.823
AssocProf	64	93876.44	13831.700	1728.9625	3455.056
Prof	266	126772.11	27718.675	1699.5410	3346.322

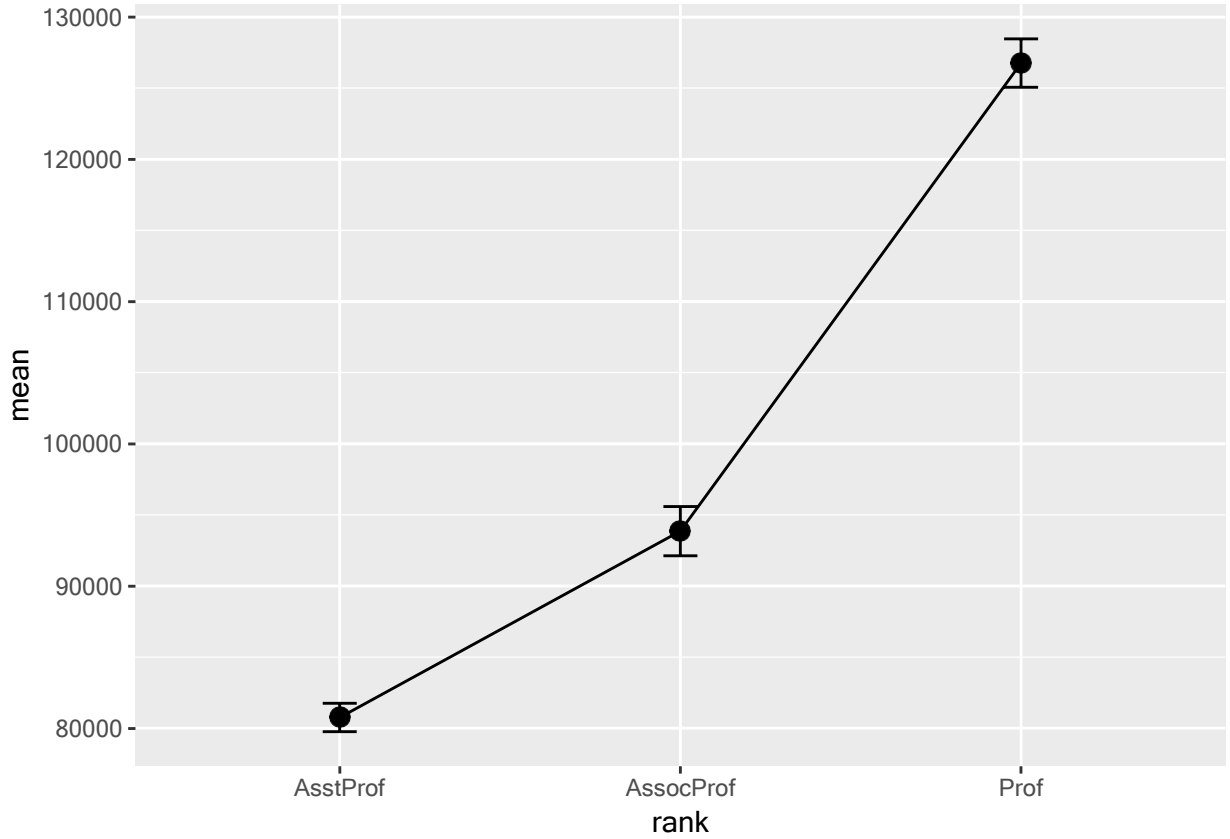
4.3.6 Mean/SEM plots

A popular method for comparing groups on a numeric variable is the mean plot with error bars. Error bars can represent standard deviations, standard error of the mean, or confidence intervals. In this section, we'll plot means and standard errors.

```
# calculate means, standard deviations,
# standard errors, and 95% confidence
# intervals by rank
library(dplyr)
plotdata <- Salaries %>%
  group_by(rank) %>%
  summarize(n = n(),
            mean = mean(salary),
            sd = sd(salary),
            se = sd / sqrt(n),
            ci = qt(0.975, df = n - 1) * sd / sqrt(n))
```

The resulting dataset is given below.

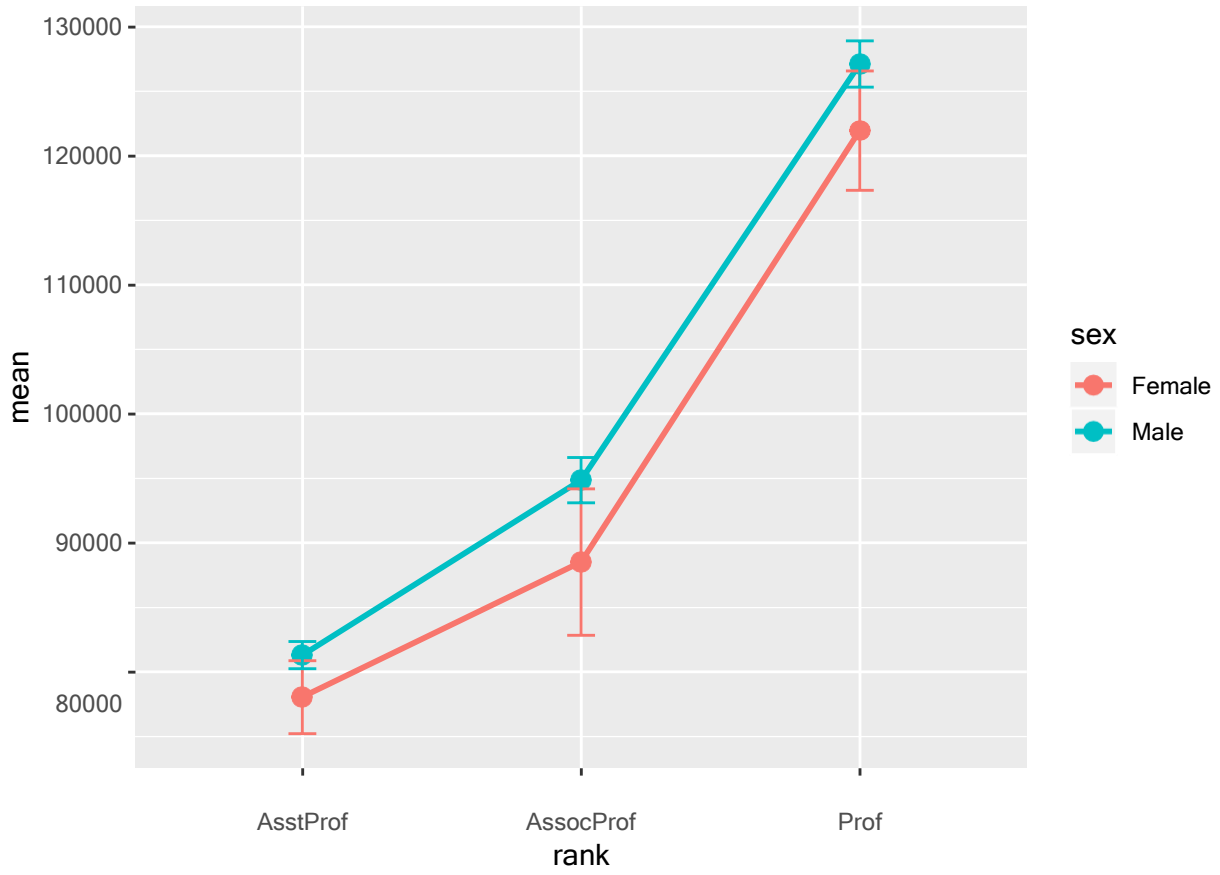
```
# plot the means and standard errors
ggplot(plotdata,
       aes(x = rank,
           y = mean,
           group = 1)) +
  geom_point(size = 3) +
  geom_line() +
  geom_errorbar(aes(ymin = mean - se,
                    ymax = mean + se),
                width = .1)
```



Although we plotted error bars representing the standard error, we could have plotted standard deviations or 95% confidence intervals. Simply replace `se` with `sd` or `error` in the `aes` option.

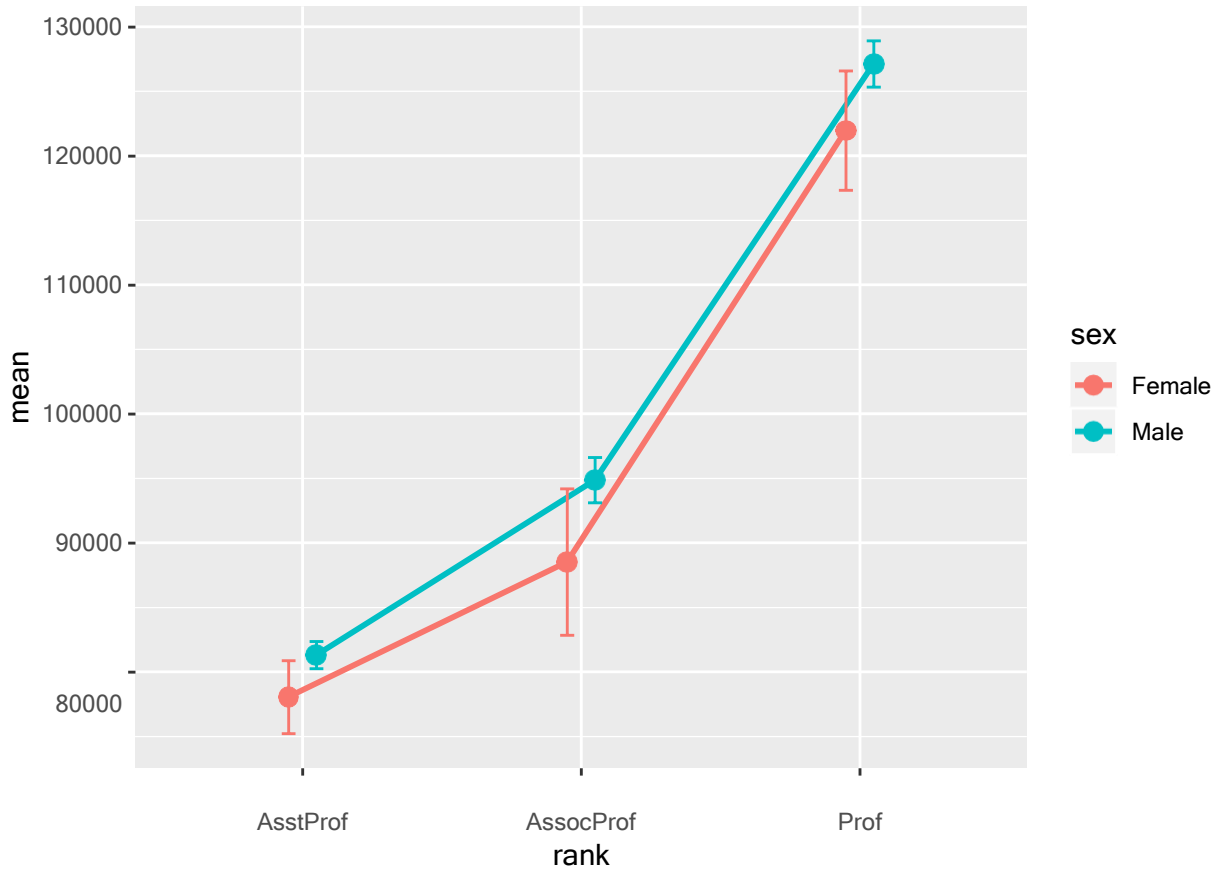
We can use the same technique to compare salary across rank and sex. (Technically, this is not bivariate since we're plotting rank, sex, and salary, but it seems to fit here)

```
# calculate means and standard errors by rank and sex
plotdata <- Salaries %>%
  group_by(rank, sex) %>%
  summarize(n = n(),
            mean = mean(salary),
            sd = sd(salary),
            se = sd/sqrt(n))

# plot the means and standard errors by sex
ggplot(plotdata, aes(x = rank,
                     y = mean,
                     group=sex,
                     color=sex)) +
  geom_point(size = 3) +
  geom_line(size = 1) +
  geom_errorbar(aes(ymin =mean - se,
                    ymax = mean+se),
                width = .1)
```



Unfortunately, the error bars overlap. We can dodge the horizontal positions a bit to overcome this.

```
# plot the means and standard errors by sex (dodged)
pd <- position_dodge(0.2)
ggplot(plotdata,
       aes(x = rank,
           y = mean,
           group=sex,
           color=sex)) +
  geom_point(position = pd,
             size = 3) +
  geom_line(position = pd,
            size = 1) +
  geom_errorbar(aes(ymin = mean - se,
                    ymax = mean + se),
                width = .1,
                position= pd)
```



Finally, lets add some options to make the graph more attractive.

```
# improved means/standard error plot
pd <- position_dodge(0.2)
ggplot(plotdata,
       aes(x = factor(rank,
                      labels = c("Assistant\nProfessor",
                                 "Associate\nProfessor",
                                 "Full\nProfessor")),
           y = mean,
           group=sex,
           color=sex)) +
  geom_point(position=pd,
             size = 3) +
  geom_line(position = pd,
            size = 1) +
  geom_errorbar(aes(ymin = mean - se,
                    ymax = mean + se),
                width = .1,
                position = pd,
                size = 1) +
  scale_y_continuous(label = scales::dollar) +
  scale_color_brewer(palette="Set1") +
  theme_minimal() +
  labs(title = "Mean salary by rank and sex",
       subtitle = "(mean +/- standard error)",
       x = "",
```

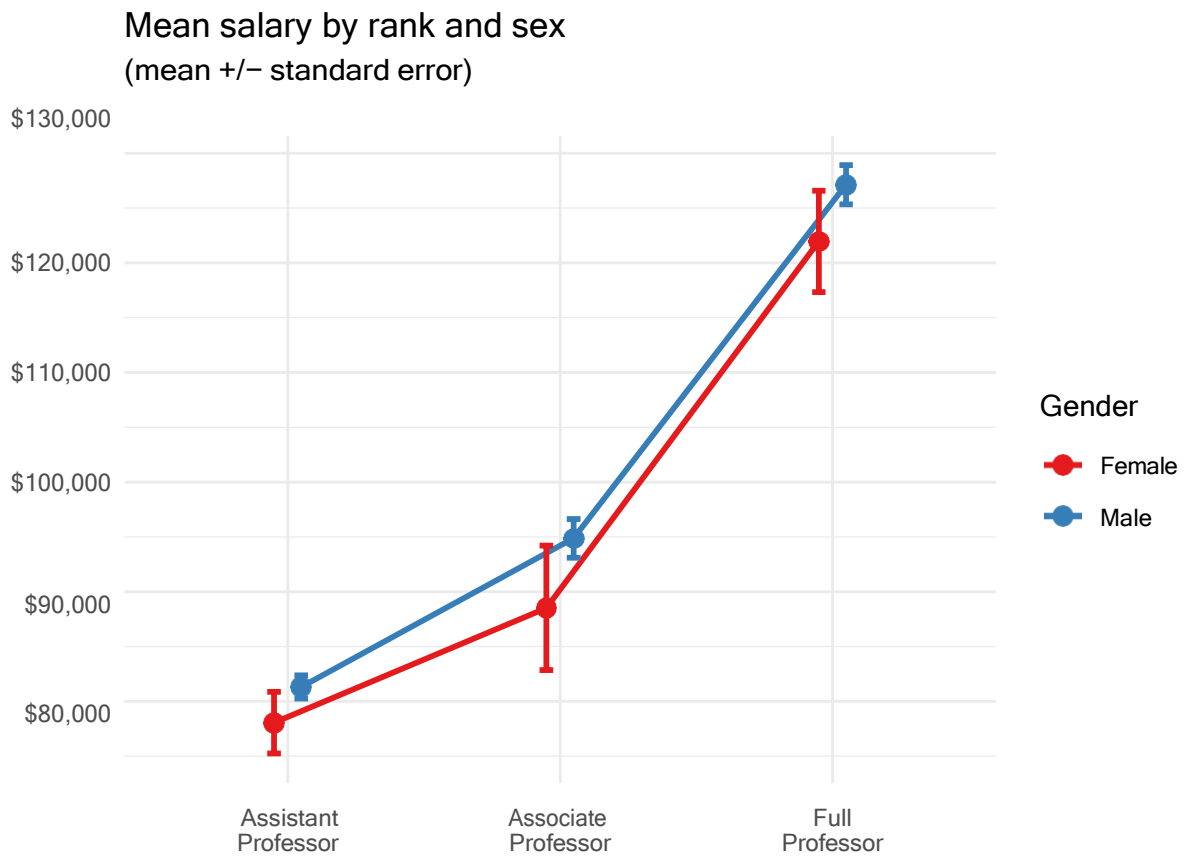


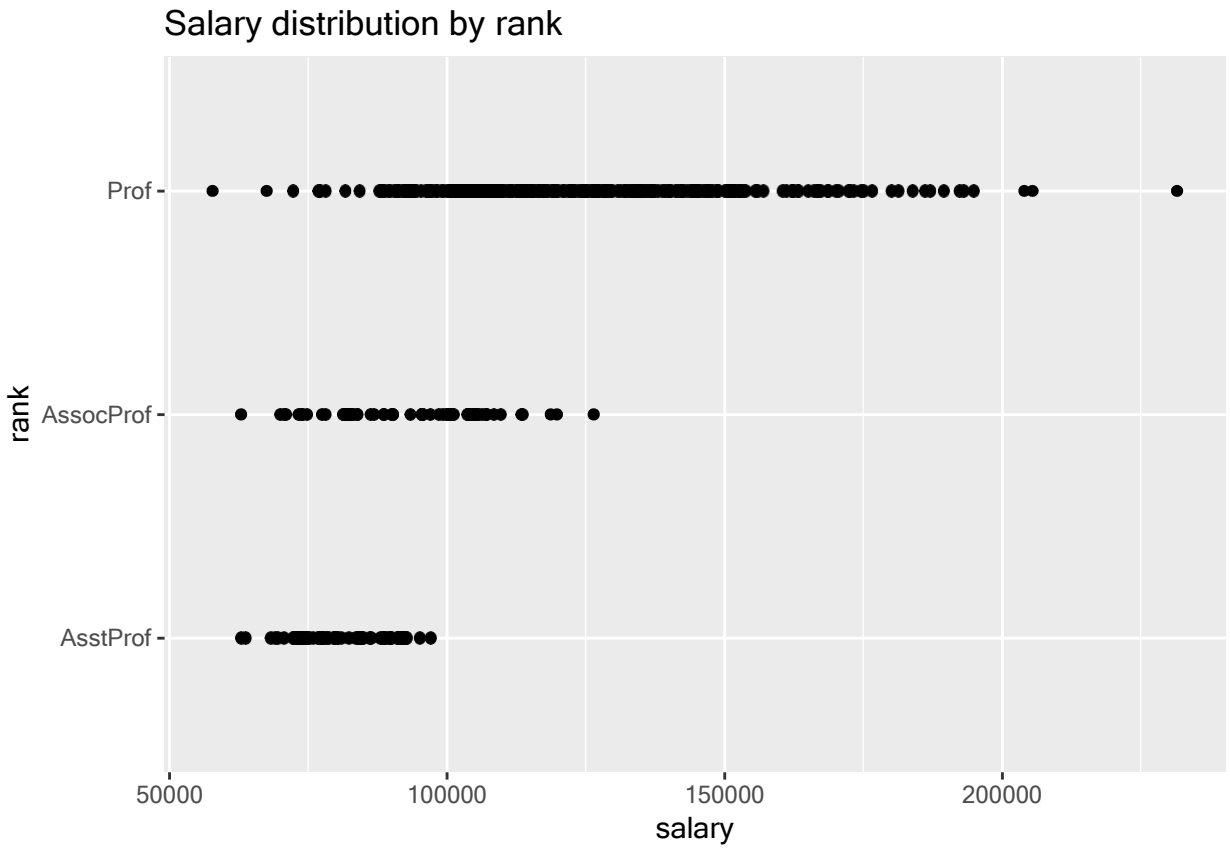
Figure 4.12: Mean/se plot with better labels and colors

```
       y = "",
       color = "Gender")
```

4.3.7 Strip plots

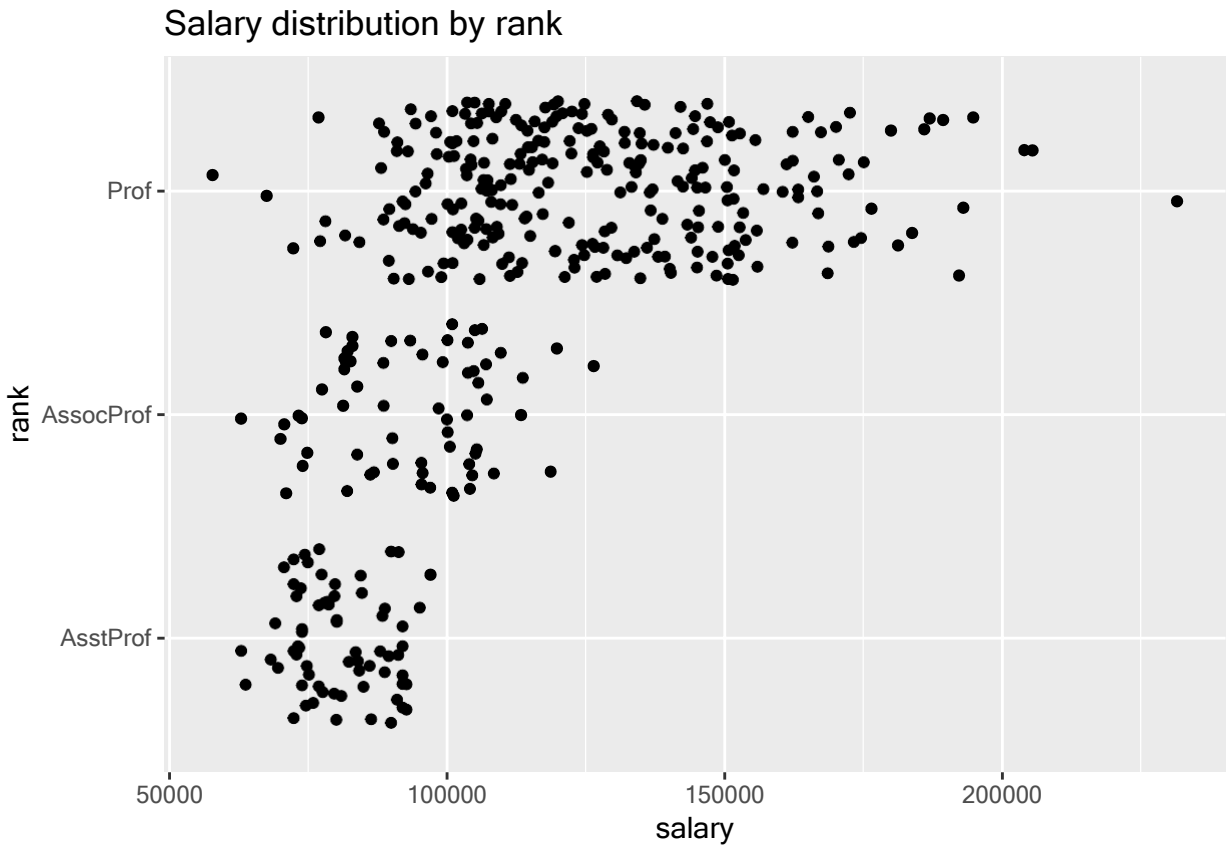
The relationship between a grouping variable and a numeric variable can be displayed with a scatter plot. For example

```
# plot the distribution of salaries
# by rank using strip plots
ggplot(Salaries,
       aes(y = rank,
           x = salary)) +
  geom_point() +
  labs(title = "Salary distribution by rank")
```



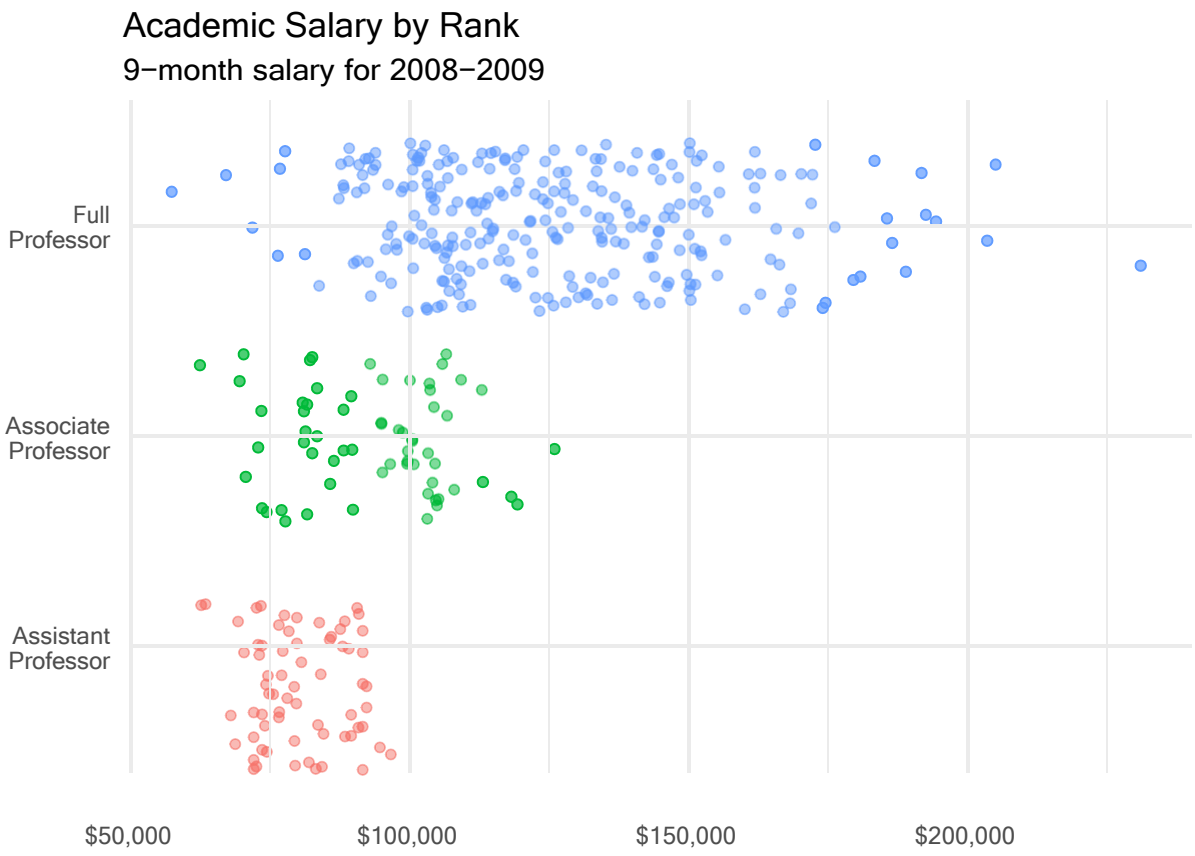
These one-dimensional scatterplots are called strip plots. Unfortunately, overprinting of points makes interpretation difficult. The relationship is easier to see if the the points are jittered. Basically a small random number is added to each y-coordinate.

```
# plot the distribution of salaries
# by rank using jittering
ggplot(Salaries,
       aes(y = rank,
           x = salary)) +
  geom_jitter() +
  labs(title = "Salary distribution by rank")
```



It is easier to compare groups if we use color.

```
# plot the distribution of salaries
# by rank using jittering
library(scales)
ggplot(Salaries,
       aes(y = factor(rank,
                      labels = c("Assistant\nProfessor",
                                 "Associate\nProfessor",
                                 "Full\nProfessor")),
           x = salary,
           color = rank)) +
  geom_jitter(alpha = 0.7,
              size = 1.5) +
  scale_x_continuous(label = dollar) +
  labs(title = "Academic Salary by Rank",
       subtitle = "9-month salary for 2008-2009",
       x = "",
       y = "") +
  theme_minimal() +
  theme(legend.position = "none")
```



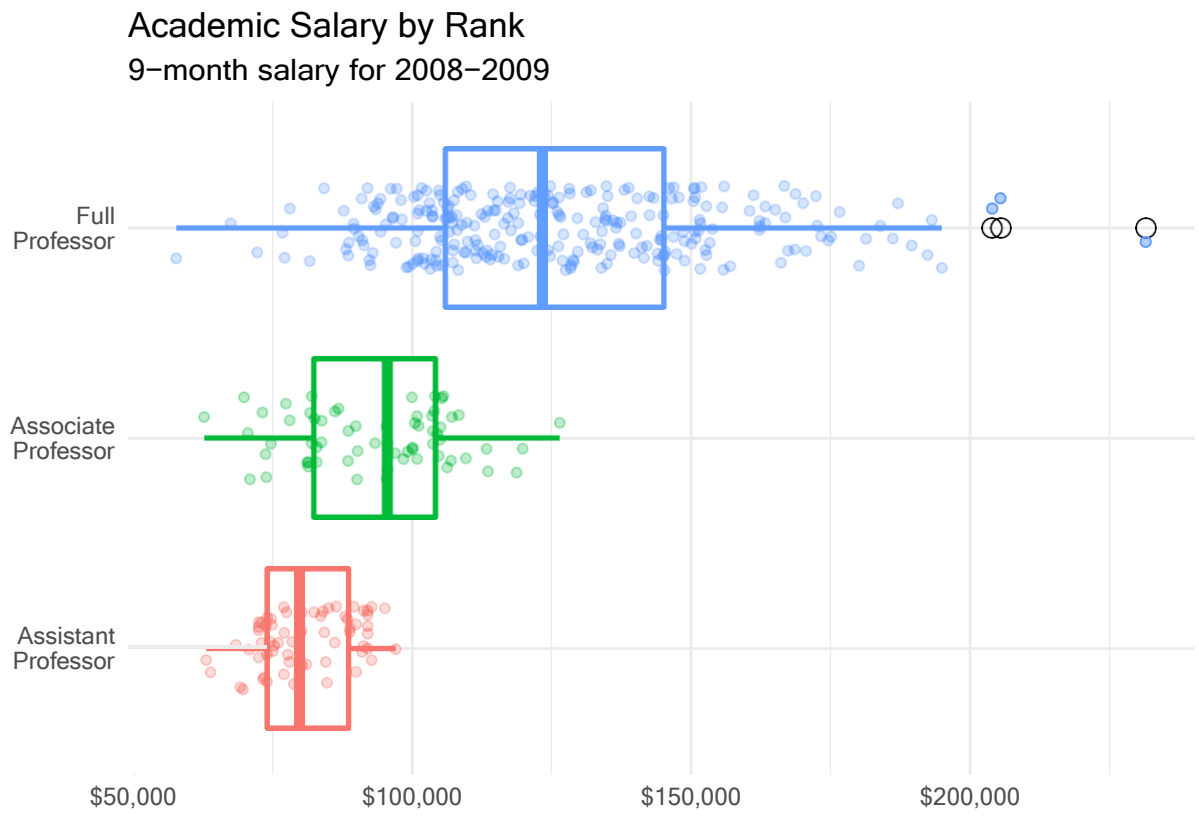
The option `legend.position = "none"` is used to suppress the legend (which is not needed here). Jittered plots work well when the number of points in not overly large.

4.3.7.1 Combining jitter and boxplots

It may be easier to visualize distributions if we add boxplots to the jitter plots.

```
# plot the distribution of salaries
# by rank using jittering
library(scales)
ggplot(Salaries,
       aes(x = factor(rank,
                      labels = c("Assistant\nProfessor",
                                 "Associate\nProfessor",
                                 "Full\nProfessor")),
           y = salary,
           color = rank)) +
  geom_boxplot(size=1,
               outlier.shape = 1,
               outlier.color = "black",
               outlier.size  = 3) +
  geom_jitter(alpha = 0.5,
              width=.2) +
  scale_y_continuous(label = dollar) +
  labs(title = "Academic Salary by Rank",
       subtitle = "9-month salary for 2008-2009",
```

```
       x = "",
       y = "") +
  theme_minimal() +
  theme(legend.position = "none") +
  coord_flip()
```



Several options were added to create this plot.

For the boxplot

- `size = 1` makes the lines thicker
- `outlier.color = "black"` makes outliers black
- `outlier.shape = 1` specifies circles for outliers
- `outlier.size = 3` increases the size of the outlier symbol

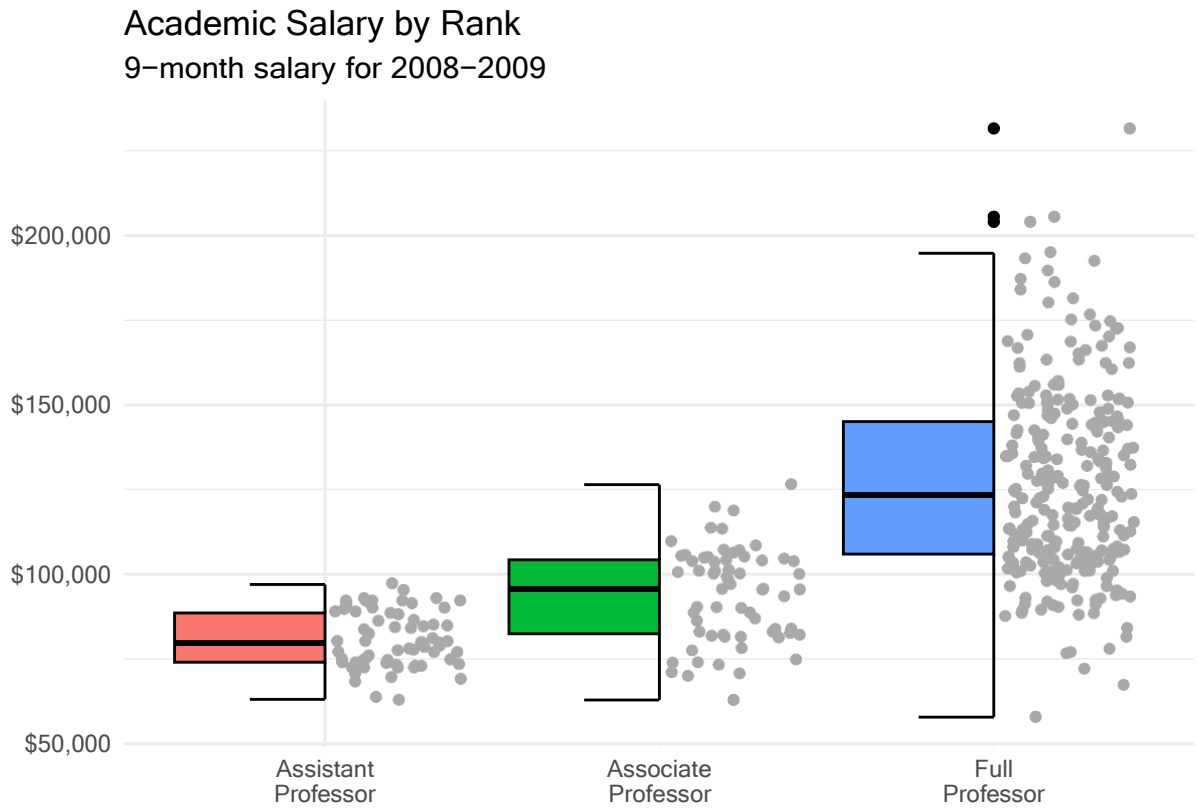
For the jitter

- `alpha = 0.5` makes the points more transparent
- `width = .2` decreases the amount of jitter (.4 is the default)

Finally, the x and y axes are revered using the `coord_flip` function (i.e., the graph is turned on its side).

Before moving on, it is worth mentioning the `geom_boxjitter` function provided in the `ggpol` package. It creates a hybrid boxplot - half boxplot, half scatterplot.

```
# plot the distribution of salaries
# by rank using jittering
library(ggpol)
library(scales)
ggplot(Salaries,
       aes(x = factor(rank,
                      labels = c("Assistant\nProfessor",
                                 "Associate\nProfessor",
                                 "Full\nProfessor")),
           y = salary,
           fill=rank)) +
  geom_boxjitter(color="black",
                 jitter.color = "darkgrey",
                 errorbar.draw = TRUE) +
  scale_y_continuous(label = dollar) +
  labs(title = "Academic Salary by Rank",
       subtitle = "9-month salary for 2008-2009",
       x = "",
       y = "") +
  theme_minimal() +
  theme(legend.position = "none")
```

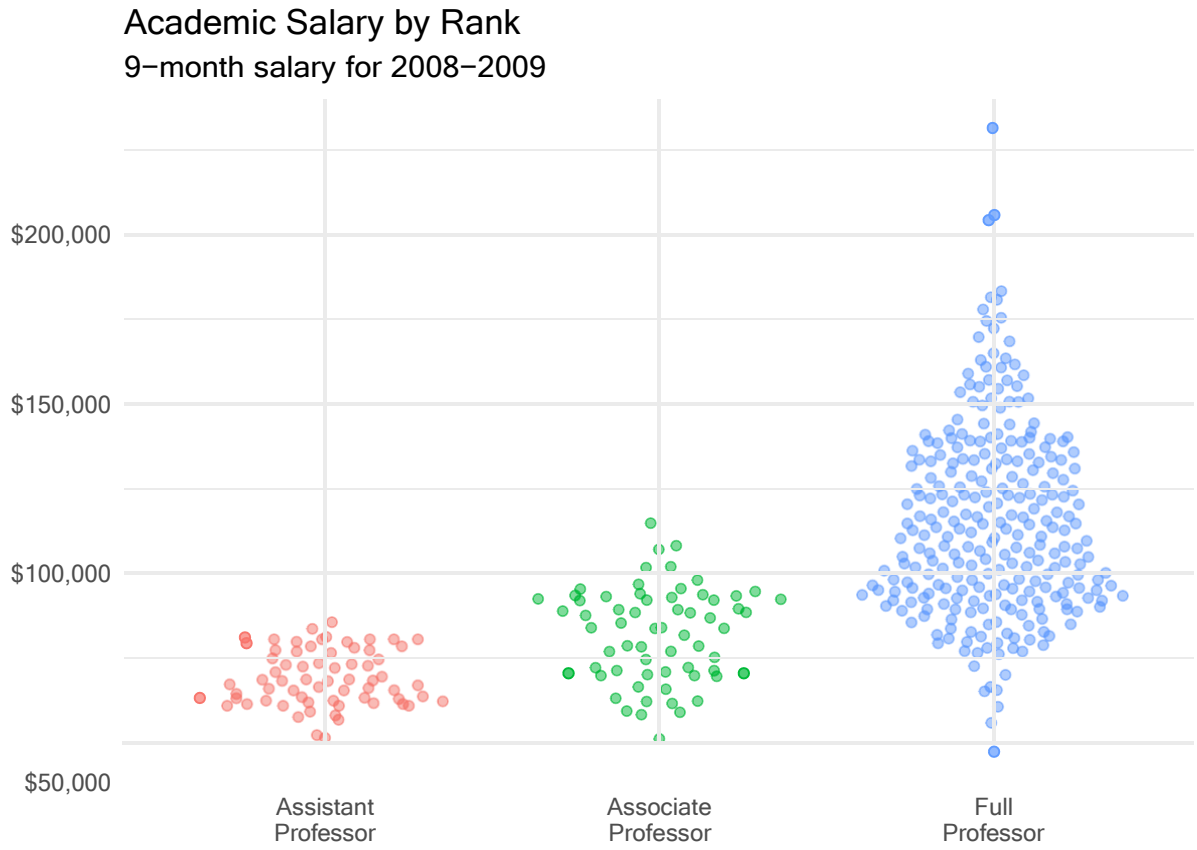


Beeswarm Plots

Beeswarm plots (also called violin scatter plots) are similar to jittered scatterplots, in that they display the distribution of a quantitative variable by plotting points in way that reduces overlap. In addition, they also help display the density of the data at each point (in a manner that is similar to a violin plot). Continuing

the previous example

```
# plot the distribution of salaries
# by rank using beewarm-syle plots
library(ggbeeswarm)
library(scales)
ggplot(Salaries,
       aes(x = factor(rank,
                      labels = c("Assistant\nProfessor",
                                 "Associate\nProfessor",
                                 "Full\nProfessor")),
           y = salary,
           color = rank)) +
  geom_quasirandom(alpha = 0.7,
                   size = 1.5) +
  scale_y_continuous(label = dollar) +
  labs(title = "Academic Salary by Rank",
       subtitle = "9-month salary for 2008-2009",
       x = "",
       y = "") +
  theme_minimal() +
  theme(legend.position = "none")
```



The plots are create using the `geom_quasirandom` function. These plots can be easier to read than simple jittered strip plots. To learn more about these plots, see Beeswarm-style plots with ggplot2.

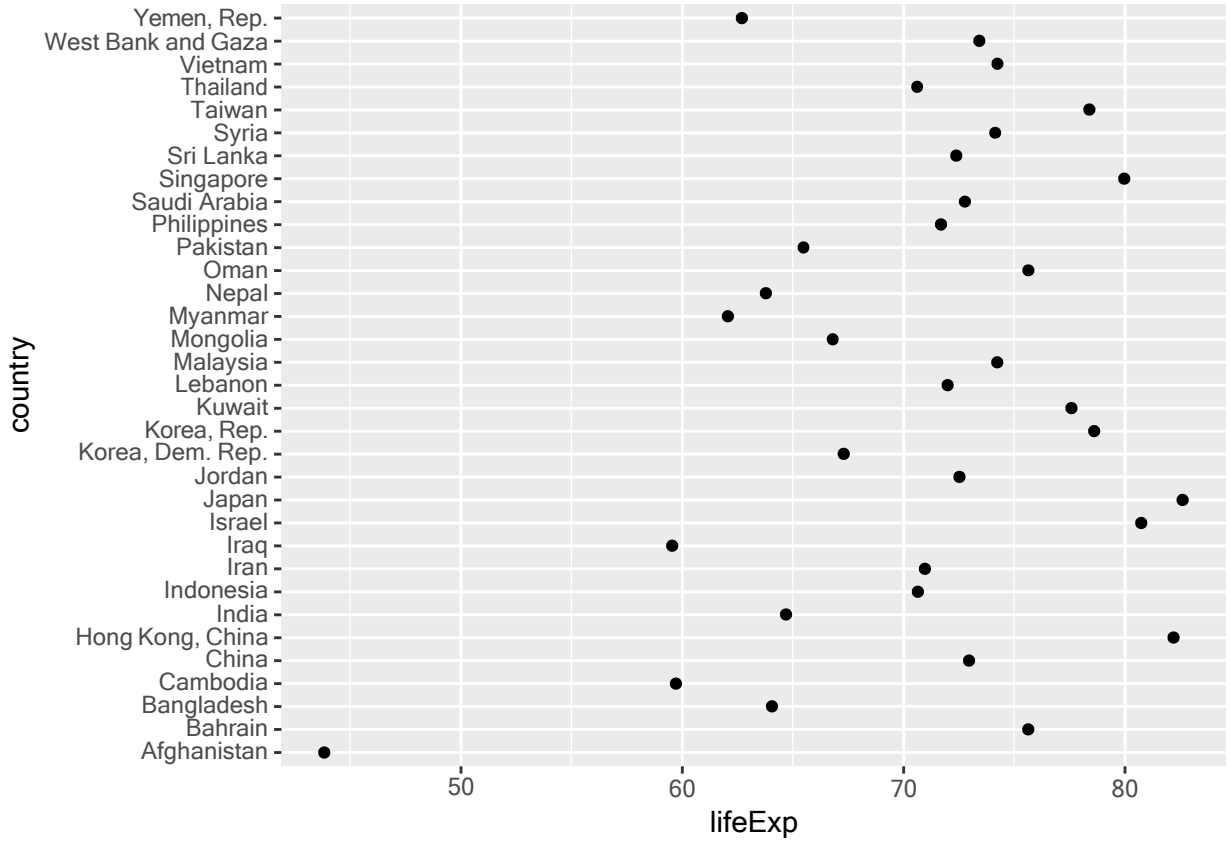


Figure 4.13: Basic Cleveland dot plot

4.3.8 Cleveland Dot Charts

Cleveland plots are useful when you want to compare a numeric statistic for a large number of groups. For example, say that you want to compare the 2007 life expectancy for Asian country using the gapminder dataset.

```
data(gapminder, package="gapminder")

# subset Asian countries in 2007
library(dplyr)
plotdata <- gapminder %>%
  filter(continent == "Asia" &
         year == 2007)

# basic Cleveland plot of life expectancy by country
ggplot(plotdata,
       aes(x= lifeExp, y = country)) +
  geom_point()
```

Comparisons are usually easier if the y-axis is sorted.

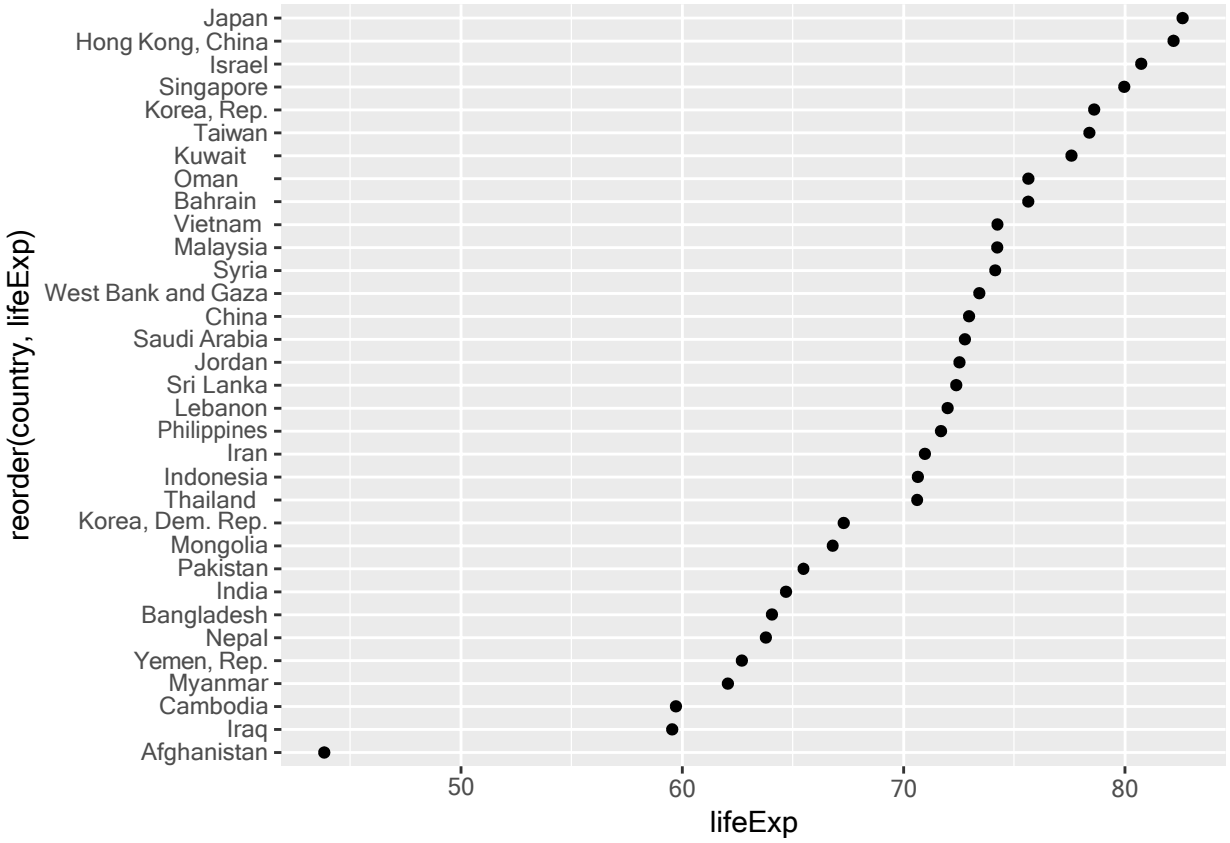


Figure 4.14: Sorted Cleveland dot plot

```
# Sorted Cleveland plot
ggplot(plotdata,
       aes(x=lifeExp,
           y=reorder(country, lifeExp))) +
  geom_point()
```

Finally, we can use options to make the graph more attractive.

```
# Fancy Cleveland plot
ggplot(plotdata,
       aes(x=lifeExp,
           y=reorder(country, lifeExp))) +
  geom_point(color="blue",
             size = 2) +
  geom_segment(aes(x = 40,
                   xend = lifeExp,
                   y = reorder(country, lifeExp),
                   yend = reorder(country, lifeExp)),
                   color = "lightgrey") +
  labs (x = "Life Expectancy (years)",
        y = "",
        title = "Life Expectancy by Country",
        subtitle = "GapMinder data for Asia - 2007") +
```

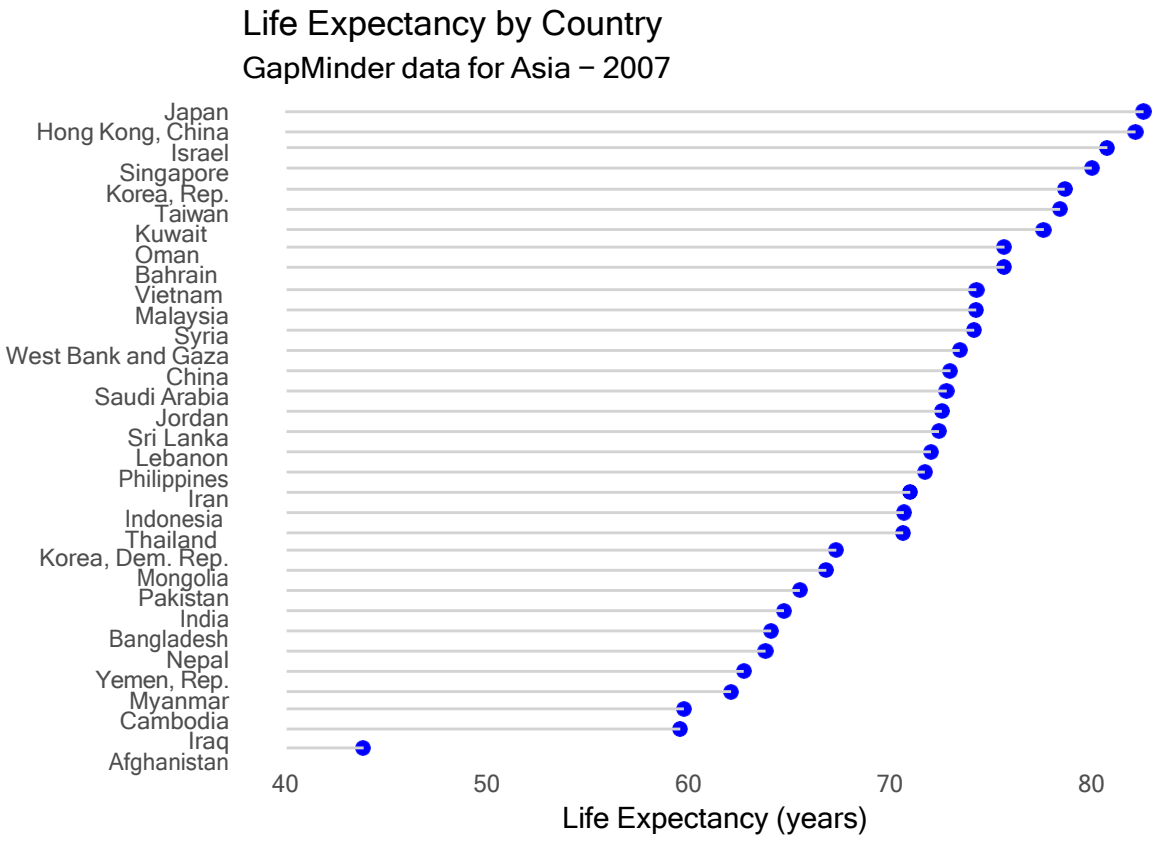


Figure 4.15: Fancy Cleveland plot

```
  theme_minimal() +
  theme(panel.grid.major = element_blank(),
        panel.grid.minor = element_blank())
```

Japan clearly has the highest life expectancy, while Afghanistan has the lowest by far. This last plot is also called a lollipop graph (you can see why)