3

Exploring Data

This chapter will cover

- Exploring your data using summary statistics.
- Exploring your data using visualization.
- Typical problems and issues to look for during data exploration.

In the last two chapters, you learned how to set the scope and goal of a data science project, and to load your data into R. In this chapter, we will start to get our hands into the data.

Suppose your goal is to build a model to predict which of your customers don't have health insurance; perhaps you want to market inexpensive health insurance packages to them. You have collected a data set of customers whose health insurance status you know. You've also identified some customer properties that you believe help predict insurance coverage: age, employment status, income, information about residence and vehicles, and so on. You've put all your data into a single data frame called *custdata* that you've input into R[8]. Now you are ready to start building the model to identify the customers you are interested in.

It's tempting to dive right into the modeling step without looking very hard at the dataset first, especially when you have a lot of data. Resist the temptation. No data set is perfect: you will be missing information about some of your customers, and you will have incorrect data about others. Some data fields will be dirty and inconsistent. If you don't take the time to examine the data before you start to model, then you may find yourself re-doing your work repeatedly, as you discover
bad data fields, or variables that need to be transformed before modeling. In the worst case, you will build a model that returns incorrect predictions — and you won't be sure why. By

[8] We have a copy of this synthetic data set available for download from https://github.com/WinVector/zmPDSwR/tree/master/Custdata and once saved you can load it into R with the command: custdata = read.table('custdata.tsv',header=T,sep='\t').

addressing data issues early, you can save yourself some unnecessary work, and a lot of headaches!

You'd also like to get a sense of who your customers are: are they young, middle-aged, or seniors? How affluent are they? Where do they live? Knowing the answers to these questions can help you build a better model, because you will have a more specific idea of what information predicts insurance coverage more accurately.

In this chapter we will demonstrate some ways to get to know your data, and discuss some of the potential issues that you are looking for as you explore. Data exploration uses a combination of *summary statistics* — means and medians, variances and counts — and *visualization*, or graphs of the data. You can spot some problems just using summary statistics; other problems are easier to find visually.

Organizing Data for Analysis

For most of this book, we will assume that the data you are analyzing is in a single data frame. This is not usually how that data is stored. In a database, for example, data is usually stored in *normalized form* to reduce redundancy: information about a single customer is spread across many small tables. In log data, data about a single customer can be spread across many log entries, or sessions. These formats make it easy to add (or in the case of a database, modify) data, but are not optimal for analysis. You can often join all the data you need into a single table in the database using SQL, but in Appendix A we will discuss commands like `merge` that you can use within R to further consolidate data.

3.1 Using Summary Statistics to Spot Problems

In R, you will typically use the `summary`command to take your first look at the data.

Listing 3.1 The summary() command

```
> summary(custdata)
custid          sex
Min.    :  2068    F:440
1st Qu.: 345667    M:560
Median : 693403
Mean   : 698500
3rd Qu.:1044606
Max.   :1414286

is.employed    income                                #A
Mode :logical  Min.    : -8700
FALSE:73       1st Qu.: 14600
TRUE :599      Median : 35000
NA's :328      Mean    : 53505
               3rd Qu.: 67000
               Max.    :615000

marital.stat
Divorced/Separated: 155
Married           : 516
```

```
Never Married     : 233
Widowed           : 96

health.ins                                          #B
Mode :logical
FALSE:159
TRUE :841
NA's :0

housing.type                                        #C
Homeowner free and clear      :157
Homeowner with mortgage/loan  :412
Occupied with no rent         : 11
Rented                        :364
NA's                          : 56

recent.move    num.vehicles
Mode :logical  Min.    :0.000
FALSE:820      1st Qu.:1.000
TRUE :124      Median :2.000
NA's :56       Mean   :1.916
               3rd Qu.:2.000
               Max.   :6.000
               NA's   :56

age            state.of.res                         #D
Min.   :       0.0    California :100
1st Qu.: 38.0  New York           : 71
Median : 50.0  Pennsylvania       : 70
Mean   : 51.7  Texas              : 56
3rd Qu.: 64.0  Michigan           : 52
Max.   :146.7  Ohio               : 51
               (Other)            : 600
```

#A The variable is.employed is missing for about a third of the data. The variable income has negative values, which are potentially invalid.
#B About 84% of the customers have health insurance.
#C The variables housing.type, recent.move, and num.vehicles are each missing 56 values.
#D The average value of the variable age seems plausible, but the minimum and maximum values seem unlikely. The variable state.of.res is a categorical variable; summary() reports how many customers are in each state (for the first few states).

The summarycommand on a data frame reports back a variety of summary statistics on the numerical columns of the data frame, and count statistics on any categorical columns (if the categorical columns have already been read in as factors[9]). You can also ask for summary statistics on specific numerical columns by using the commands mean, variance, median, min, max, and quantile (which will return the quartiles of the data by default).

As you see from Listing 3.1, the summary of the data helps you quickly spot potential problems, like missing data or unlikely values. You also get a rough idea of how categorical data is distributed. Let's go into more detail about the typical problems that you can spot using the summary.

[9] Categorical variables are of class factorin R. They can be represented as strings (class character), and some analytical functions will automatically convert strings variables to factor variables. To get a summary of the variable, it needs to be a factor.

3.1.1 Typical Problems that can be spotted from data summaries

At this stage, you are looking for several common issues:

- Missing Values
- Invalid Values and Outliers
- Data Range that is too wide or too narrow
- Units

MISSING VALUES

A few missing values may not really be a problem, but if a particular data field is largely unpopulated, it shouldn't be used as an input without some repair (as we will discuss in Chapter 4 section 4.14.1). In R, for example, many modeling algorithms will quietly drop rows with missing values by default. As you see in Listing 3.2, all the missing values in the *is.employed* variable could cause R to quietly ignore nearly a third of the data.

Listing 3.2 Will the variable is.employed be useful for modeling?

```
is.employed                              #A
Mode :logical
FALSE:73
TRUE :599
NA's :328

                    housing.type         #B
Homeowner free and clear     :157
Homeowner with mortgage/loan :412
Occupied with no rent        : 11
Rented                       :364
NA's                         : 56

recent.move         num.vehicles
Mode :logical       Min.    :0.000
FALSE:820           1st Qu.:1.000
TRUE :124           Median :2.000
NA's :56            Mean    :1.916
                    3rd Qu.:2.000
                    Max.    :6.000
                    NA's    :56
```

#A The variable is.employed is missing for about a third of the data. Why? Is employment status unknown? Did the company start collecting employment data only recently? Does NA mean "not in the active workforce" (for example, students or stay-at-home parents)?
#B The variables housing.type, recent.move, and num.vehicles are only missing a few values. It's probably safe to just drop the rows that are missing values -- especially if the missing values are all the same 56 rows.

If a particular data field is largely unpopulated, it is worth trying to determine why; sometimes the fact that a value is missing is informative in and of itself. For example, why is the *is.employed* variable missing so many values? There are many possible reasons, as we discussed in Listing 3.2.

Whatever the reason for missing data, you must decide on the most appropriate action. Do you include a variable with missing values in your model, or not? If you decide to include it,

do you drop all the rows where this field is missing, or do you convert the missing values to zero or to an additional category? We will discuss ways to treat missing data in Chapter 4. In this example, you might decide to drop the data rows where you are missing data about housing or vehicles, since there aren't many of them. You probably do not want to throw out the data where you are missing employment information, but instead treat the *NA* as a third employment category. You will likely encounter missing values when model scoring, so you should deal with them during model training.

INVALID VALUES AND OUTLIERS

Even when a column or variable is not missing any values, you still want to check that the values that you do have make sense. Do you have any invalid values or outliers? Examples of invalid values include negative values in what should be a non-negative numeric data field (like age or income), or text where you expect numbers. Outliers are data points that fall well out of the range where you expect the data to be. Can you spot the outliers and invalid values in Listing 3.3?

Listing 3.3 Examples of invalid values and outliers

```
> summary(custdata$income)
Min. 1st Qu.       Median  Mean 3rd Qu.

-8700       14600   35000   53500   67000                #A
Max.
615000

> summary(custdata$age)
Min. 1st Qu.       Median  Mean 3rd Qu.

0.0  38.0    50.0    51.7    64.0                        #B
Max.
146.7
```

#A Negative values for income could indicate bad data. They might also have a special meaning, like "amount of debt." Either way, you should check how prevalent the issue is, and decide what to do: Do you drop the data with negative income? Do you convert negative values to zero?
#B Customers of age zero, or customers with age greater than about 110 are outliers. They fall out of the range of expected customer values. Outliers could be data input errors. They could be special sentinel values: zero might mean "age unknown" or refuse to state." And some of your customers might be especially long-lived.

Often, invalid values are simply bad data input. Negative numbers in a field like *age* however, could be a *sentinel value* to designate "unknown". Outliers might also be data errors, or sentinel values. Or they might be valid but unusual data points — people do occasionally live past 100.

As with missing values, you must decide the most appropriate action: drop the data field, drop the data points where this field is bad, or convert the bad data to a useful value. Even if you feel certain outliers are valid data, you might still want to omit them from model construction (and also collar allowed prediction range), since the usual achievable goal of modeling is to predict the typical case correctly.

DATA RANGE

You also want to pay attention to how much the values in the data vary. If you believe that age or income help to predict health insurance coverage, then you should make sure there is enough variation in the age and income of your customers for you to see the relationships. Let's look at income again, in Listing 3.4. Is the data range wide? Is it narrow?

Listing 3.4 Looking at the data range of a variable.

```
> summary(custdata$income)
Min.        1st Qu.        Median         Mean    3rd Qu.

-8700       14600          35000          53500   67000          #A
Max.
615000
```

#A Income ranges from zero to over half a million dollars; a very wide range.

Even ignoring negative income, the *income* variable in Listing 3.4 ranges from zero to over half a million. That's pretty wide (though typical for income). Data that ranges over several orders of magnitude like this can be a problem for some modeling methods. We will talk about mitigating data range issues when we talk about logarithmic transformations in Chapter 4.

Data can be too narrow, too. Suppose all your customers were between the ages of 50 and 55. It's a fairly good bet that age would not be a very good predictor of health insurance coverage for that population, since it doesn't vary much at all.

How narrow is "too narrow" a data range?

Of course, the term "narrow" is relative. If we were predicting the ability to read for children between ages of five and ten, then age probably is a useful variable as is. For data including adult ages you may want to transform or bin ages in some way, as you don't expect a significant change in reading ability between ages 40 and 50. You should rely on information about the problem domain to judge if the data range is narrow, but a rough rule of thumb is the ratio of the standard deviation to the mean. If that ratio is very small, then the data isn't varying much.

We will revisit data range in section 3.2, when we talk about examining data graphically.

One factor that determines apparent data range is the unit of measurement. To take a non-technical example, we measure the ages of babies and toddlers in weeks or in months, because developmental changes happen at that time scale for very young children. Suppose we measured baby ages in years. It might appear numerically that there isn't much

difference between a one-year-old and a two-year-old. In reality, there is a dramatic difference, as any parent can tell you!

Units can present potential issues in a dataset for another reason, as well.

UNITS

Does the income data in figure 3.5 represent hourly wages, or yearly wages in units of $1000? As a matter of fact, it is the latter, but what if you thought it was the former? You might not notice the error during the modeling stage, but down the line, someone will start inputting hourly wage data into the model, and get back bad predictions in return.

```
> summary(Income)                          #A
Min.        1st Qu.        Median         Mean    3rd Qu.        Max.
-8.7        14.6           35.0           53.5    67.0           615.0
```

#A The variable Income is defined as "Income = custdata$income/1000". But suppose you didn't know that. Looking only at the summary, the values could plausibly be interpreted to mean either "hourly wage" or "yearly income in units of $1000".

Are time intervals measured in days, hours, minutes or milliseconds? Are speeds in kilometers per second, miles per hour or knots? Are monetary amounts in dollars, thousands of dollars, or 1/100th of a penny (a customary practice in finance, where calculations are often done in fixed-point arithmetic)? This is actually something that you will catch by checking data definitions in data dictionaries or documentation, rather than in the summary statistics; the difference between hourly wage data and annual salary in units of $1000 may not look that obvious at a casual glance. But it is still something to keep in mind while looking over the value ranges of your variables, because often you can spot when measurements are in unexpected units. Automobile speeds in knots look a lot different than they do in miles per hour.

3.2 Spotting Problems Using Graphics and Visualization

As you've seen, you can spot plenty of problems just by looking over the data summaries. For other properties of the data, pictures are better than text.

> We cannot expect a small number of numerical values [summary statistics] to consistently convey the wealth of information that exists in data. Numerical reduction methods do not retain the information in the data.
>
> -- William Cleveland, *The Elements of Graphing Data*

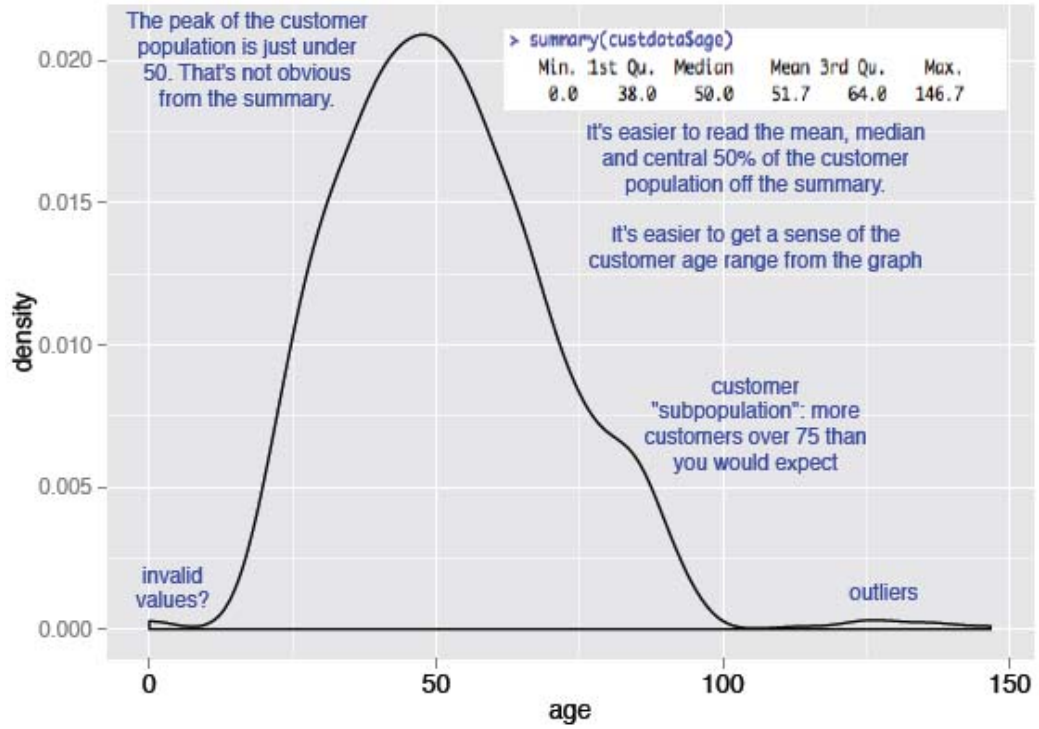


Figure 3.1 Some information is easier to read from a graph, and some from a summary.

Figure 3.1 shows a plot of how customer ages are distributed. We'll talk about what the y-axis of the graph means later; for right now, just know that the height of the graph corresponds to how many customers in the population are of that age. As you can see, information like the peak age of the distribution, the existence of subpopulations, and the presence of outliers are easier to absorb visually than they are to determine textually.

The use of graphics to examine data is called *visualization*. We try to follow William Cleveland's principles for scientific visualization. Details of specific plots aside, the key points of Cleveland's philosophy are:

- A graphic should display as much information as it can, with the lowest possible cognitive strain to the viewer.
- Strive for clarity. Make the data stand out. Specific tips for increasing clarity include:
  - Avoid too many superimposed elements, for example too many curves in the same graphing space.
  - Find the right aspect ratio and scaling to properly bring out the details of the data.
  - Avoid having the data all skewed to one side or other of your graph.

- Visualization is an iterative process. Its purpose is to answer questions about the data.

During the visualization stage, you graph the data, learn what you can, and then regraph the data to answer the questions that arise from your previous graphic. Different graphics are best suited for answering different questions. We will look at some of them in this section.

In this book, we use ggplot to demonstrate the visualizations and graphics; of course, other R visualization packages can produce similar graphics.

A note on ggplot

The theme of this section is how to use visualization to explore your data, not how to use ggplot. We chose ggplot because it excels at superimposing multiple graphical elements together, but its syntax can take some getting used to. The key points to understand when looking at our code snippets are:

- Graphs in ggplot can only be defined on data frames. Each variable in a graph -- the x variable, the y variable, the variables that define the color or the size of the points -- are called aesthetics, and are declared by using the aes function.
- The ggplot() function declares the graph object. The arguments to ggplot()can include the data frame of interest, and the aesthetics. The ggplot()function does not of itself produce a visualization; visualizations are produced by layers.
- Layers produce the plots and plot transformations, and are added to a given graph object using the + operator. Each layer can also take a data frame and aesthetics as arguments, in addition to plot-specific parameters. Examples of layers are geom_point (for a scatterplot) or geom_line (for a line plot).

This syntax will become clearer in the examples below. For more information, we recommend Hadley Wickham's reference site http://ggplot2.org, which has pointers to online documentation, as well as to Dr. Wickham's ggplot2 reference book.

In the next two sections we will see how to use pictures and graphs to identify data characteristics and issues. In section 3.12.1 we will look at visualizations for single variables. In section 3.12.2 we will look at visualizations for two variables. But let's start by looking at a single variable.

3.2.1 Visually Checking Distributional Assumptions for a Single Variable

The visualizations in this section help you answer questions like:

- What is peak value of the distribution?
- How many peaks are there in the distribution (Unimodality vs. Bimodality)?
- How normal (or lognormal) is the data? We discuss normal and lognormal distributions in Appendix B.

- How much does the data vary? Is it concentrated in a certain interval, or in a certain category?

One of the things that is easier to see visually is the shape of the data distribution. Except for the blip to the right, the graph in figure 3.1 (which we have reproduced as the gray curve in figure 3.2) is almost shaped like the normal distribution that we saw in Appendix B. As we mention in that section, many analysis methods (linear and logistic regression, in particular) assume that the input data is approximately normal in distribution (at least for continuous variables), so you want to verify whether or not this is the case.

You can also see that the gray curve in figure 3.2 has only one peak, or that it is unimodal. This is another property that you want to check in your data.

Why? Because (roughly speaking), a unimodal distribution corresponds to one population of subjects. For the gray curve in figure 3.2, the mean customer age is about 52, and 50% of the customers are between 38 and 64 (the first and third quartiles). So you can say that a "typical" customer is middle-aged, and probably possesses many of the demographic qualities of a middle-aged person — though of course you have to verify that with your actual customer information.

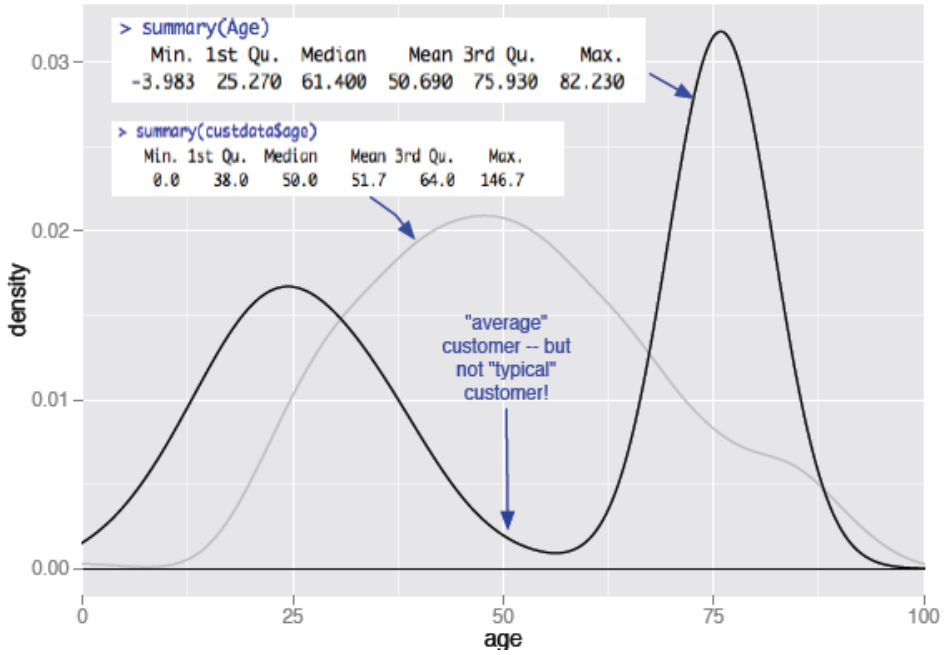


Figure 3.2 A unimodal distribution (in gray) can usually be modelled as coming from a single population of users. With a bimodal distribution (in black), your data often comes from two populations of users.

The black curve in figure 3.2 shows what can happen when you have two peaks, or a bimodal distribution. (A distribution with more than two peaks is simply called multimodal.) This set of customers has about the same mean age as the customers represented by the gray curve — but a 50-year old is hardly a "typical" customer! This (admittedly exaggerated) example corresponds to two populations of customers: a fairly young population mostly in their 20s and 30s, and an older population mostly in their 70s. These two populations probably have very different behavior patterns, and if you want to model whether a customer has health insurance or not, it wouldn't be a bad idea to model the two populations separately — especially if you are using linear or logistic regression.

The histogram and the densityplot are two visualizations that help you quickly examine the distribution of a numerical variable.

Figures 3.1 and 3.2 are densityplots. Whether you use histograms or densityplots is largely a matter of taste. We tend to prefer densityplots, but histograms are easier to explain to less quantitatively-minded audiences.

HISTOGRAMS

A basic histogram bins a variable into fixed-width buckets and returns the number of data points that falls into each bucket. For example, you could group your customers by age range, in intervals of five years: 20-25, 25-30, 30-35, and so on. Customers at a boundary age would go into the higher bucket: 25-year-olds go into the 25-30 bucket. For each bucket, you then count how many customers are in that bucket. The resulting histogram is shown in figure 3.3.

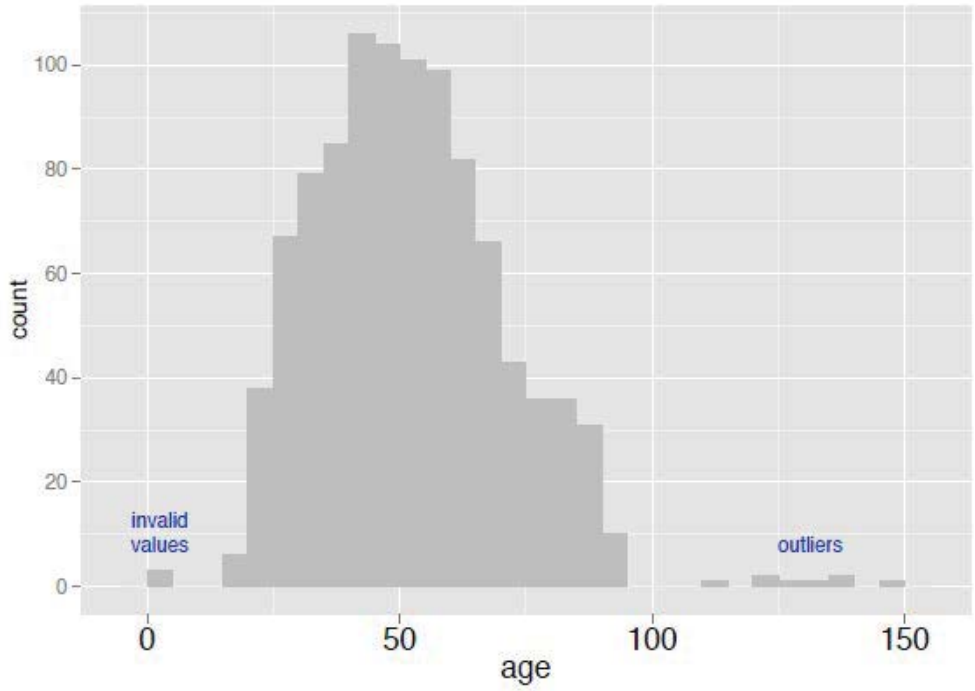


Figure 3.3 A histogram tells you where your data is concentrated. It also visually highlights outliers and anomalies.

You create the histogram in figure 3.3 in `ggplot` with the `geom_histogram` layer.

```
library(ggplot2)                                    #A

ggplot(custdata) +
    geom_histogram(aes(x=age),

    binwidth=5, fill="gray")                        #B
```

#A Load the `ggplot2` library, if you haven't already done so.
#B The binwidth parameter tells the geom_histogram call how to make bins of five year intervals (default is datarange/30). The fill parameter specifies the color of the histogram bars (default: black).

The primary disadvantage of histograms is that you must decide ahead of time how wide the buckets are. If the buckets are too wide, you can lose information about the shape of the distribution. If the buckets are too narrow, the histogram can look too noisy to read easily. An alternative visualization is the densityplot.

DENSITYPLOTS

You can think of a densityplot as a "continuous histogram" of a variable, except the area under the densityplot is equal to one. A point on a densityplot corresponds to the fraction of data (or the percentage of data, divided by 100) that takes on a particular value. This

fraction is usually very small. When you look at a densityplot, you are more interested in the overall shape of the curve, than in the actual values on the y-axis. You've seen the densityplot of age; figure 3.4 shows the densityplot of income.

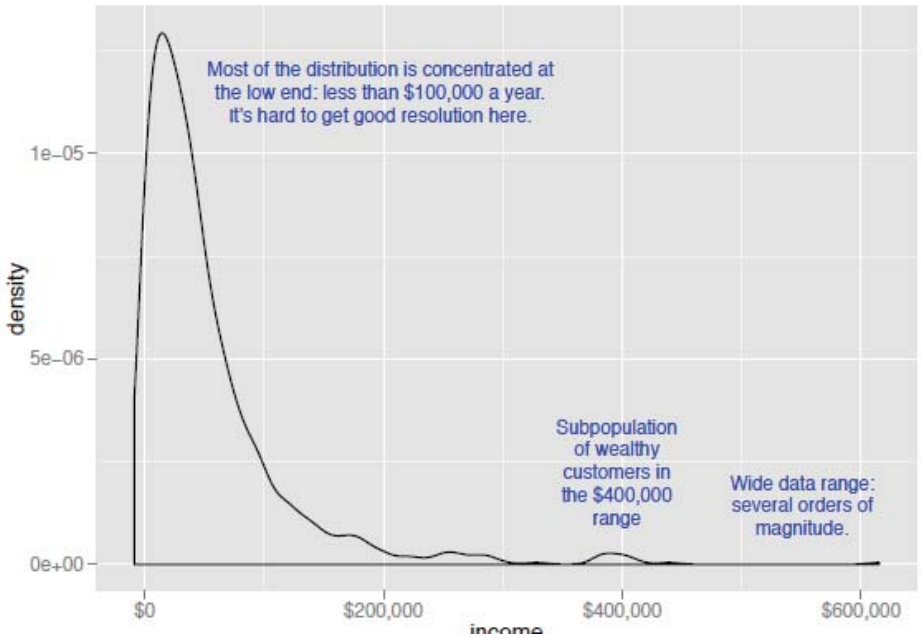


Figure 3.4 Densityplots also show where data is concentrated. This plot also highlights a population of higher-income customers.

You produce figure 3.4 with the `geom_density` layer.

```
library(scales)                                          #A

ggplot(custdata) + geom_density(aes(x=income)) +

    scale_x_continuous(labels=dollar)                    #B
```

#A The scales package brings in the dollar scale notation.
#B Set the x-axis labels to dollars.

When the data range is very wide, and the mass of the distribution is heavily concentrated to one side, like the distribution in figure 3.4, it is difficult to see the details of its shape. For instance, it's hard to tell the ex act value where the income distribution has its peak. If the data is non-negative, then one way to bring out more detail is to plot the distribution on a logarithmic scale, as shown in figure 3.5. This is equivalent to plotting the densityplot of *log10(income)*.

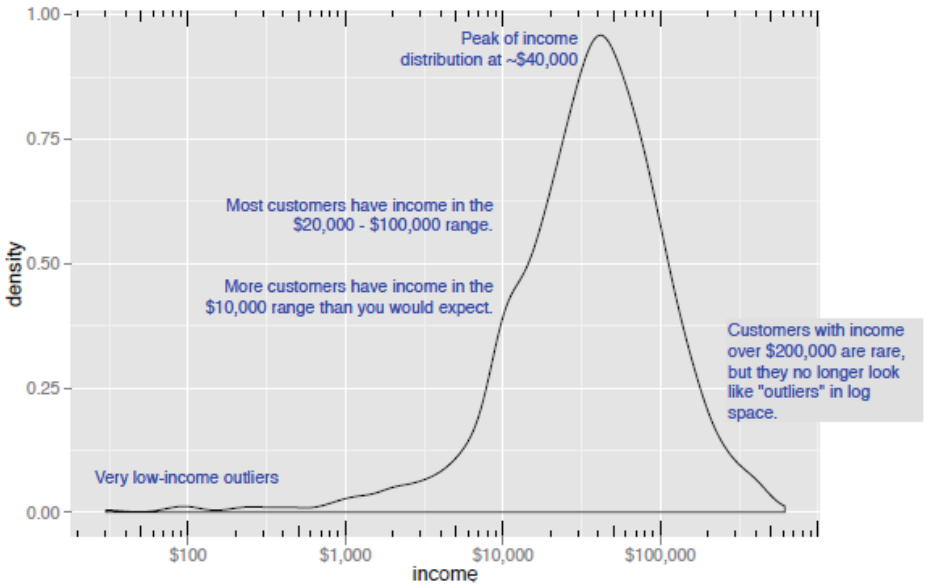


Figure 3.5 The densityplot of income on a log10 scale highlights details of the income distribution that are harder to see in a regular densityplot.

In `ggplot`, you can plot figure 3.5 with the `geom_density` and `scale_x_log10` layers.

```
ggplot(custdata) + geom_density(aes(x=income)) +
    scale_x_log10(breaks=c(100,1000,10000,100000), labels=dollar) +   #A
    annotation_logticks(sides="bt")                                   #B
```

#A Set the x-axis to be in log10 scale, with manually set tick points and labels as dollars.
#B Add log-scaled tick marks to the top and bottom of the graph.

When you issued the command above, you also got back a warning message:

```
Warning messages:
1: In scale$trans$trans(x) : NaNs produced
2: Removed 79 rows containing non-finite values (stat_density).
```

This tells you that ggplot ignored the zero and negative valued rows (since *log(0)* is Infinity), and that there were 79 such rows. Keep that in mind when evaluating the graph.

TIP When should you use a logarithmic scale?

You should use a logarithmic scale when percent change, or change in orders of magnitude is more important than changes in absolute units. You should also use a log scale to better visualize data that is heavily skewed.

For example, in income data a difference in income of five thousand dollars means something very different in a population where the incomes tend to fall in the tens of thousands of dollars than it does in populations where income falls in the hundreds of thousands or millions of dollars. In other word, what constitutes a "significant difference" depends on the order of magnitude of the incomes you are looking at. Similarly, in a population like that in figure 3.5, a few people with very high income will cause the majority of the data to be compressed into a relatively small area of the graph. For both those reasons, plotting the income distribution on a logarithmic scale is a good idea.

In log space, income is distributed as something that looks like a "normalish" distribution, as you would expect from our discussion in Appendix B. It's not exactly a normal (in fact, it appears to be at least two normal distributions mixed together), but it's close enough to normal for analysis methods like linear or logistic regression.

BARCHARTS

A barchart is a histogram for discrete data: it records the frequency of every value of a categorical variable. Figure 3.6 shows the distribution of marital status in your customer data set. If you believe that marital status helps predict health insurance coverage, then you want to check that you have enough customers with different marital statuses to help you discover the relationship between being married (or not) and having health insurance.

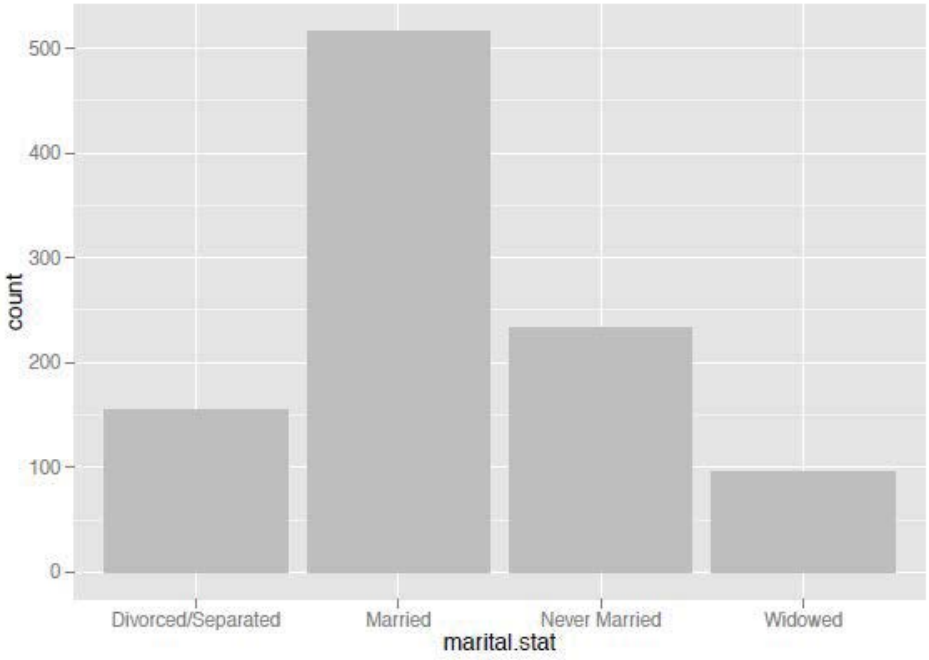


Figure 3.6 Barcharts show the distribution of categorical variables.

The `ggplot` command to produce figure 3.6 uses `geom_bar`.

```
ggplot(custdata) + geom_bar(aes(x=marital.stat), fill="gray")
```

This graph doesn't really show any more information than `summary(custdata$marital.stat)` would show, although some people find the graph easier to absorb than the text. Barcharts are most useful when the number of possible values is fairly large, like state of residence. In this situation, we often find that a sideways graph is more legible than an upright graph.

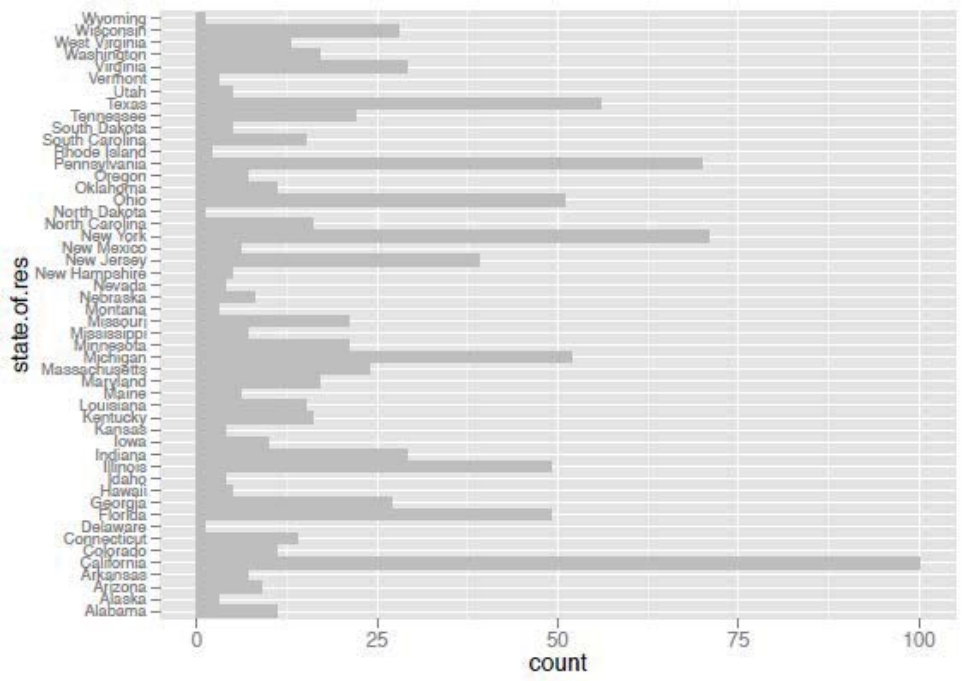


Figure 3.7 A sideways barchart can be easier to read when there are several categories with long names.

The `ggplot` command to produce figure 3.7 is

```
ggplot(custdata) +
    geom_bar(aes(x=state.of.res), fill="gray") +        #A
    coord_flip() +                                      #B
    theme(axis.text.y=element_text(size=rel(0.8)))      #C
```

#A Plot barchart as before: state.of.res is on x axis, count is on y-axis.
#B Flip the x and y axes: state.of.res is now on the y-axis.
#C Reduce the size of the y-axis tick labels to 80% of default size for legibility.

Cleveland[10] recommends that the data in a barchart (or in a dotplot, Cleveland's preferred visualization in this instance) be sorted, to more efficiently extract insight from the data. This is shown in figure 3.8.

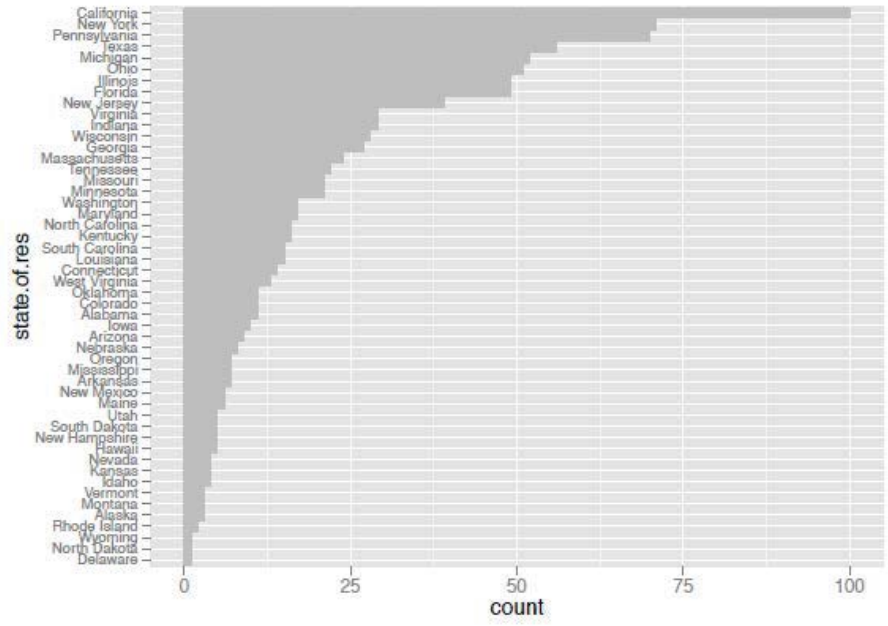


Figure 3.8 Sorting the barchart by count makes it even easier to read

This visualization requires a bit more manipulation, at least in `ggplot` because by default, `ggplot` will plot the categories of a factor variable in alphabetical order. To change this, we have to manually specify the order of the categories -- in the factor variable, not in `ggplot`.

```
> statesums <- table(custdata$state.of.res)            #A

> statef <- as.data.frame(statesums)                   #B

> colnames(statef)<-c("state.of.res",  "count")        #C

> summary(statef)                                      #D
state.of.rescount
Alabama     : 1     Min.    :      1.00
```

[10] Cleveland, William S. *The Elements of Graphing Data*, Hobart Press, 1994.

```
Alaska       : 1    1st Qu.:       5.00
Arizona      : 1    Median : 12.00
Arkansas     : 1    Mean   : 20.00
California: 1       3rd Qu.: 26.25
Colorado     : 1    Max.   :100.00 (Other)       :44
> statef <- transform(statef,

     state.of.res=reorder(state.of.res,  count))          #E

> summary(statef)                                        #F
state.of.res       count
Delaware     : 1    Min.   : 1.00
North Dakota: 1    1st Qu.: 5.00
Wyoming      : 1    Median : 12.00
Rhode Island: 1    Mean   : 20.00
Alaska       : 1    3rd Qu.: 26.25
Montana      : 1    Max.   :100.00
(Other)      :44
> ggplot(statef)+ geom_bar(aes(x=state.of.res,y=count),

     stat="identity",                                     #G
     fill="gray") +

     coord_flip() +                                       #H
     theme(axis.text.y=element_text(size=rel(0.8)))
```

#A The table() command aggregates the data by state of residence -- exactly the information that barchart plots.
#B Convert the table object to a data frame using as.data.frame(). The default column names are "Var1" and "Freq".
#C Rename the columns for readability.
#D Notice that the default ordering for the state.of.res variable is alphabetical
#E Use the reorder() function to set the state.of.res variable to be count-ordered. Use the transform() function to apply the transformation to the statef data frame.
#F The state.of.res variable is now count ordered.
#G Since the data is being passed to geom_bar pre-aggregated, specify both the x and y variables, and use stat="identity" to plot the data exactly as given.
#H Flip the axes and reduce the size of the label text as before.

Before we move on to visualizations for two variables, let's summarize the visualizations that we've reviewed in this section.

Table 3.1 Visualizations for One Variable

Graph Type	Uses
Histogram	▪ Examine data range ▪ Check number of modes ▪ Check if distrubution is normal/lognormal ▪ Check for anomalies and outliers

Densityplot	▪ Examine data range ▪ Check number of modes ▪ Check if distrubution is normal/lognormal ▪ Check for anomalies and outliers
Barcharts	Compare relative or absolute frequencies of the values of a categorical variable.

3.2.2 Visually Checking Relationships Between Two Variables

In addition to examining variables in isolation, you will often want to look at the relationship between two variables. For instance you want to answer questions like:

- Is there a relationship between the two inputs *age* and *income* in my data?
- What kind of relationship, and how strong?
- Is there a relationship between the input *marital status* and the output *health insurance*? How strong?

You will precisely quantify these relationships during the modeling phase, but exploring them now gives you a feel for the data, and helps you determine which variables are the best candidates to include in a model.

First, let's consider the relationship between two continuous variables. The most obvious way (though not always the best) is the line plot.

LINE PLOTS

Line plots work best when the relationship between two variables is relatively clean: that is each x-value has a unique (or nearly unique) y-value.

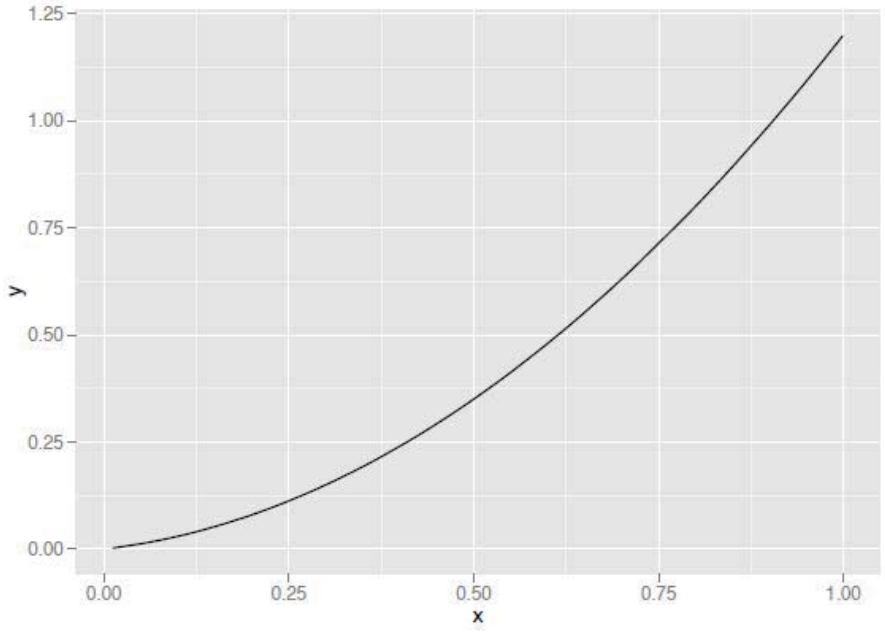


Figure 3.9 Example of a line plot.

You plot Figure 3.9 with `geom_line`.

```
x = runif(100)                                          #A

y = x^2 + 0.2*x                                         #B

ggplot(data.frame(x=x,y=y),  aes(x=x,y=y)) + geom_line()        #C
```

#A First, generate the data for this example. The x variable is uniformly randomly distributed between zero and one.
#B The y variable is a quadratic function of x
#C Plot the line plot

When the data is not so cleanly related, line plots are not as useful; you will want to use the scatterplot instead, as we will see in the next section.

SCATTERPLOTS AND SMOOTHING CURVES

You would expect that there is a relationship between age and health insurance, and also a relationship between income and health insurance. But what is the relationship between age and income? If they track each other perfectly, then you might not want to use both variables in a model for health insurance. The appropriate summary statistic is the correlation, which we compute on a safe subset of our data:

```
custdata2 <- subset(custdata,
```

```
    (custdata$age > 0 & custdata$age < 100
    & custdata$income > 0))                    #A

cor(custdata2$age, custdata2$income)           #B

[1] -0.02240845                                #C
```

#A Only consider a subset of the data with reasonable age and income.
#B Only values. Get the correlation of age and income.
#C The resulting correlation.

The negative correlation is surprising, since you would expect that income should increase as people get older. A visualization gives you more insight into what is going on than a single number can. We can try a scatter plot first (Figure 3.10).

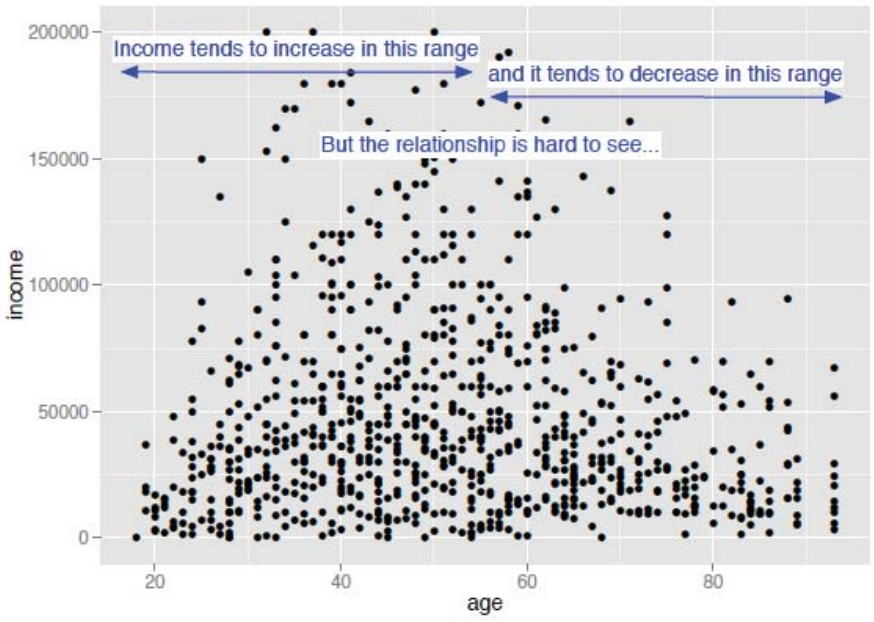


Figure 3.10 A scatterplot of income versus age.

You plot figure 3.10 with `geom_point`.

```
ggplot(custdata2, aes(x=age, y=income)) +
geom_point() + ylim(0, 200000)
```

The relationship between age and income isn't easy to see. You can try to make the relationship clearer by also plotting a linear fit through the data, as shown in Figure 3.11.

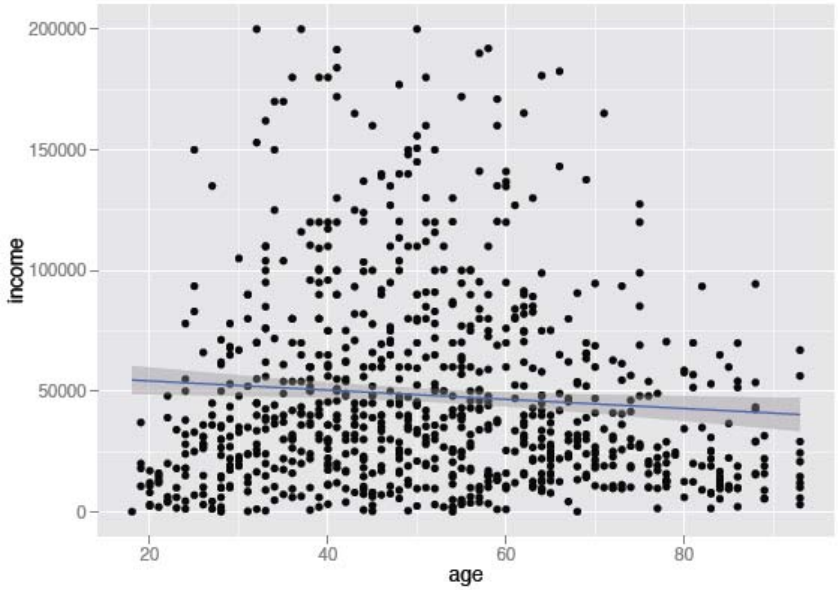


Figure 3.11 A scatterplot of income versus age, with a linear fit.

You plot Figure 3.11 using the `stat_smooth` layer. [11]

```
ggplot(custdata2, aes(x=age, y=income)) + geom_point() +
  stat_smooth(method="lm") +
  ylim(0, 200000)
```

In this case, the linear fit doesn't really capture the shape of the data. You can better capture the shape by instead plotting a smoothing curve through the data, as shown in Figure 3.12.

[11] The *stat* layers in ggplot2 are the layers that perform transformations on the data. They are usually called under the covers by the *geom* layers. Sometimes you have to call them directly, to access parameters that are not accessible from the *geom* layers. In this case, the default smoothing curve used geom_smooth (which we will see shortly) is a loess curve. To plot a linear fit we must call stat_smooth directly.

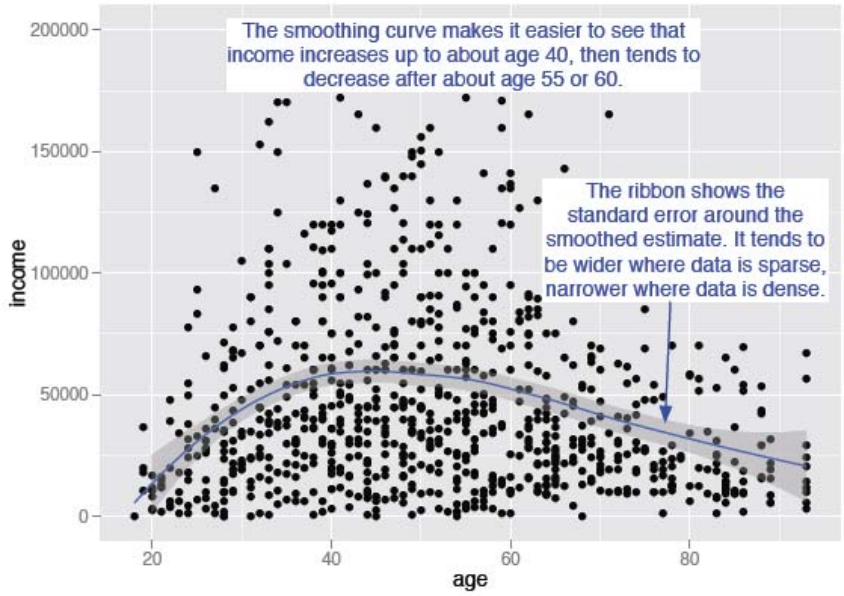


Figure 3.12 A scatterplot of income versus age, with a smoothing curve.

In R, smoothing curves are fit using the loess (or lowess) functions, which calculate smoothed local linear fits of the data. In ggplot, you can plot a smoothing curve to the data by using geom_smooth.

```
ggplot(custdata2, aes(x=age, y=income)) +
  geom_point() + geom_smooth() +
  ylim(0, 200000)
```

A scatterplot with smoothing curve also makes a good visualization of the relationship between a continuous variable and a boolean. Suppose you are considering using age as an input to your health insurance model. You might want to plot health insurance coverage as a function of age, as shown in figure 3.13. This will show you that the probability of having health insurance increases as customer age increases.

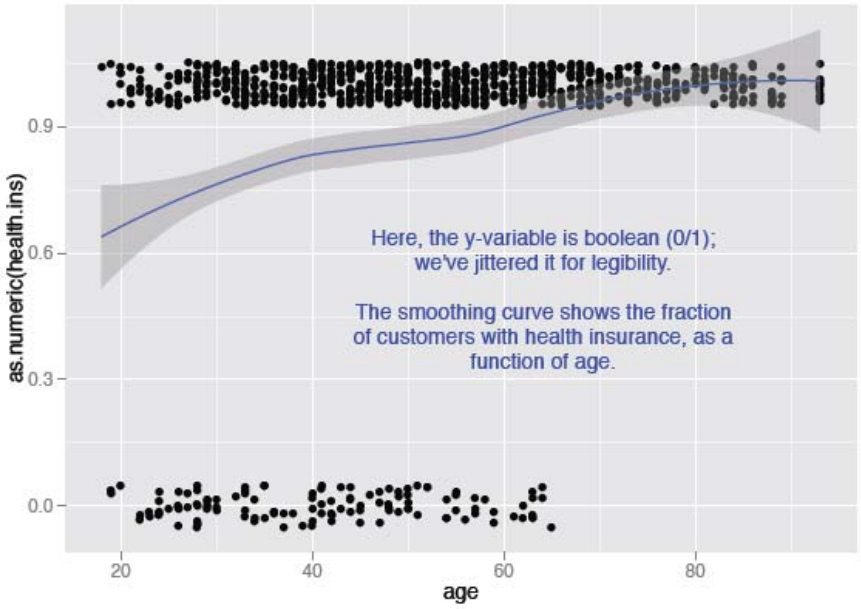


Figure 3.13 Fraction of customers with health insurance, as a function of age

You plot figure 3.13 with the command:

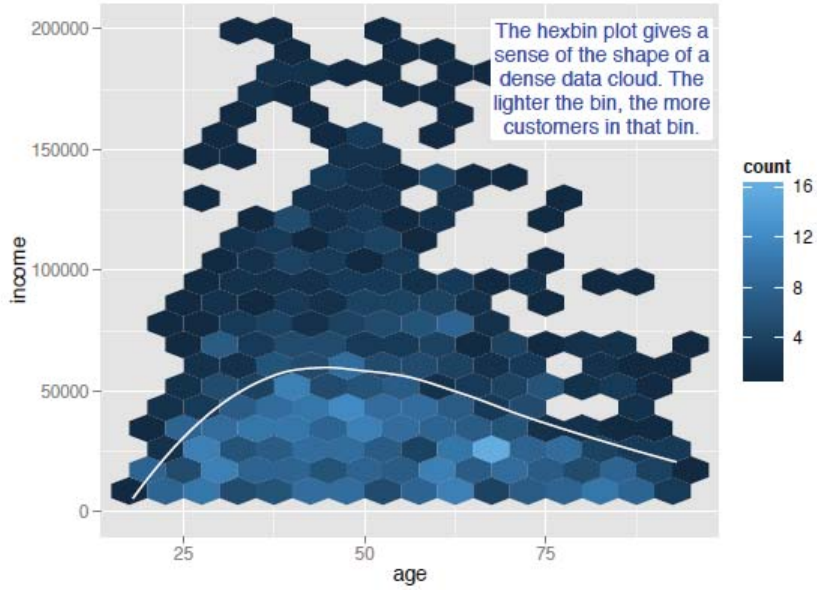
```
ggplot(custdata2, aes(x=age, y=as.numeric(health.ins))) +        #A

  geom_point(position=position_jitter(w=0.05, h=0.05)) +         #B

  geom_smooth()                                                  #C
```

#A The boolean variable health.ins must be converted to a 0/1 variable using as.numeric.
#B Since the y values can only be zero or one, add a small jitter to get a sense of data density.
#C Add the smoothing curve.

In our health insurance examples, the dataset is small enough that the scatterplots that you have created are still legible. If the dataset were a hundred times bigger, there would be so many points that they would begin to plot on top of each other; the scatterplot would turn into an illegible smear. In high-volume situations like this, try an aggregated plot, like a hexbin plot.

HEXBIN PLOTS

A hexbin plot is like a two-dimensional histogram. The data is divided into bins, and the number of data points in each bin is represented by color or shading. Let's go back to the income versus age example. Figure 3.14 shows a hexbin plot of the data. Note how the smoothing curve traces out the shape formed by the densest region of data.



3.14 Hexbin plot of income versus age, with a smoothing curve superimposed in white.

To make a hexbin plot in R, you must have the hexbinpackage installed. We discuss how to install R packages in Appendix A. Once hexbin is installed and the library loaded, you create the plots using the geom_hexlayer.

```
library(hexbin)                                    #A

ggplot(custdata2, aes(x=age, y=income)) +

geom_hex(binwidth=c(5, 10000)) +                   #B
geom_smooth(color="white", se=F) +        "
ylim(0,200000)                                     #C
```

#A Load the hexbin library.
#B Create the hexbin with age binned into 5-year increments, income in increments of$10,000
#C Add the smoothing curve in white, suppress the standard error ribbon (se=F)

In this section and the previous section we've looked at plots where at least one of the variables is numerical. But in our health insurance example, the output is categorical, and so are many of the input variables. Next we will look at ways to visualize the relationship between two categorical variables.

BARCHARTS FOR TWO CATEGORICAL VARIABLES

Let's examine the relationship between marital status and health insurance. The most straightforward way to visualize this is with a stacked barchart, as shown in figure 3.15.

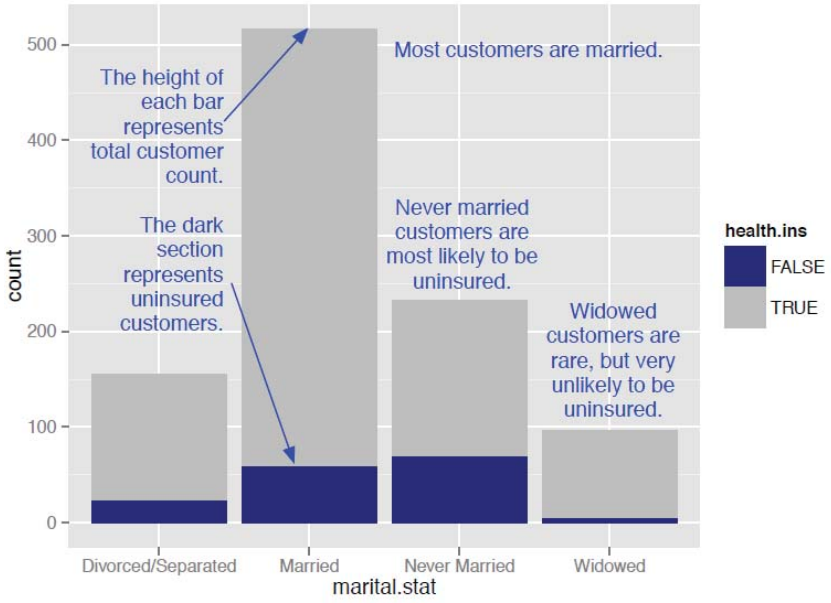


Figure 3.15 Health insurance versus marital status: Stacked barchart

Some people prefer the side-by-side barchart, shown in figure 3.16, which makes it easier to compare the number of both insured and uninsured across categories.

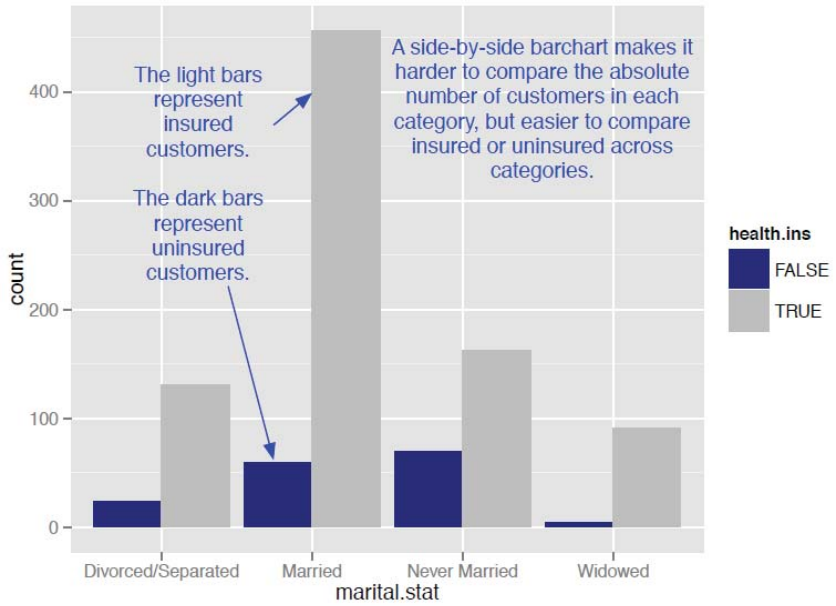


Figure 3.16 Health insurance versus marital status: Side-by-side barchart

The main shortcoming of both the stacked and side-by-side barcharts is that you can't easily compare the ratios of insured to uninsured across categories, especially for rare categories like Widowed. You can use what ggplot calls a filled barchart to plot a visualization of the ratios directly, as in figure 3.17.

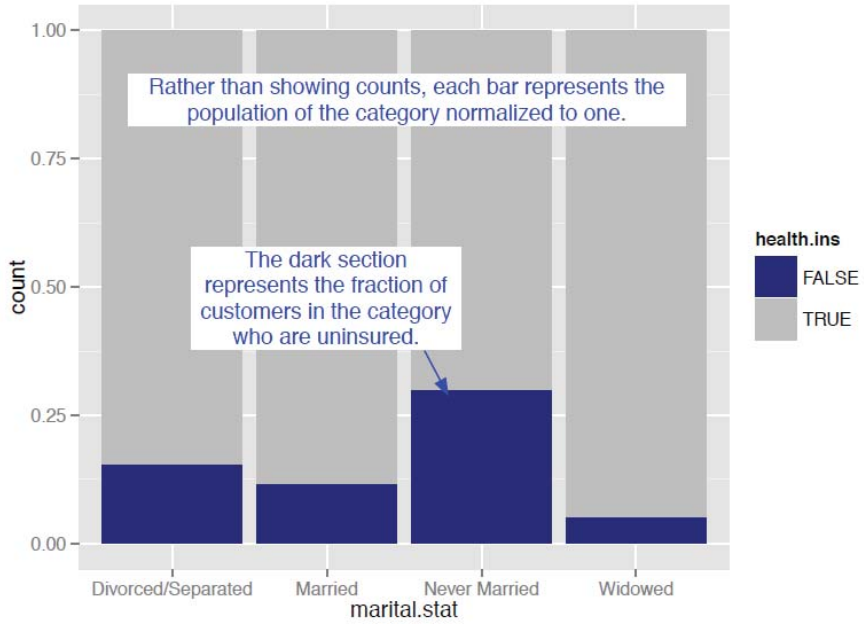


Figure 3.17 Health insurance versus marital status: Filled barchart

The filled barchart makes it obvious that divorced customers are slightly more likely to be uninsured than married ones. However, you've lost the information that being widowed, while highly predictive of insurance coverage, is a rare category.

Which barchart you use depends on what information is most important for you to convey. The ggplot commands for each of these plots are given below. Note the use of the fill aesthetic; this tells ggplot to color (fill) the bars according to the value of the variable specifies *health.ins*. The `position` argument to `geom._bar` specifies the barchart style.

```
ggplot(custdata) + geom_bar(aes(x=marital.stat,          #A
  fill=health.ins))

ggplot(custdata) + geom_bar(aes(x=marital.stat,          #B
  fill=health.ins),
  position="dodge")

ggplot(custdata) + geom_bar(aes(x=marital.stat,          #C
  fill=health.ins),
  position="fill")
```

#A Stacked barchart, the default.
#B Side-by-side barchart
#C Filled barchart

To get a simultaneous sense of both the population in each category and the ratio of insured to uninsured, you can add what's called a "rug" to the filled barchart. A rug is a

series of ticks or points on the x-axis, one tick per datum. The rug is dense where you have a lot of data, and sparse where you have little data. This is shown in figure 3.18.

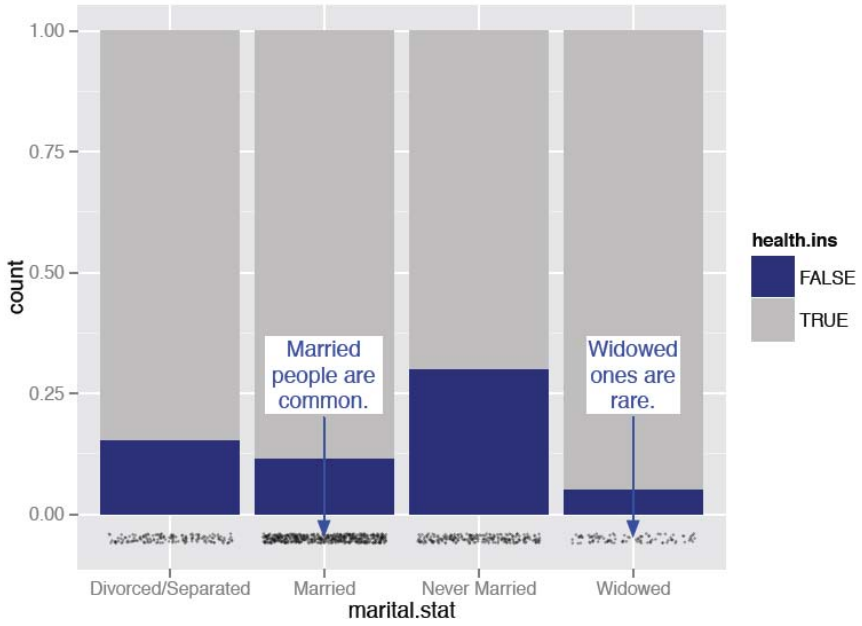


Figure 3.18 Health insurance versus marital status: Filled barchart with rug

You generate this graph by adding a geom_pointlayer to the graph.

```
ggplot(custdata, aes(x=marital.stat)) +
  geom_bar(aes(fill=health.ins), position="fill") +
  geom_point(aes(y=-0.05), size=0.75, alpha=0.3,       #A
  position=position_jitter(h=0.01))                    #B
```

#A Set the points just under the y-axis, three-quarters of default size, and make them slightly transparent with the alpha parameter.
#B Jitter the points slightly for legibility.

In the above examples, one of the variables was binary; the same plots can be applied to two variables that each have several categories, but the results are harder to read. Suppose you are interested in the distribution of marriage status across housing types. I find the side-by-side barchart easiest to read in this situation, but it's not perfect, as you see in figure 3.19.

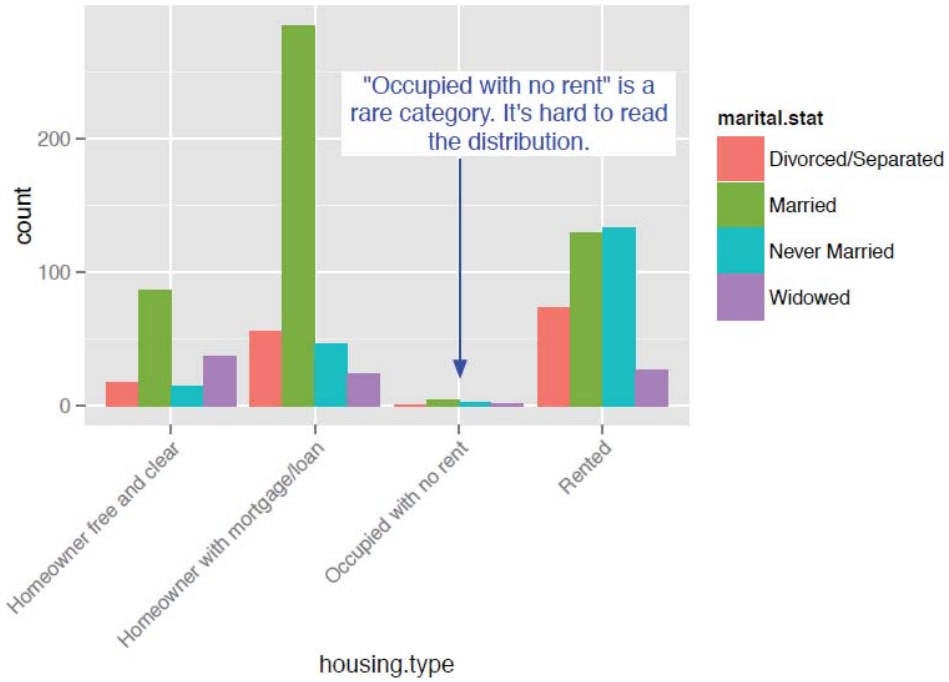


Figure 3.19 Distribution of marital status by housing type: side-by-side barchart

A graph like figure 3.19 above gets cluttered if either of the variables has a large number of categories. A better alternative is to break the distributions into different graphs, one for each housing type. In ggplot this is called *faceting* the graph, and you use the facet_wraplayer. The result is in figure 3.20.

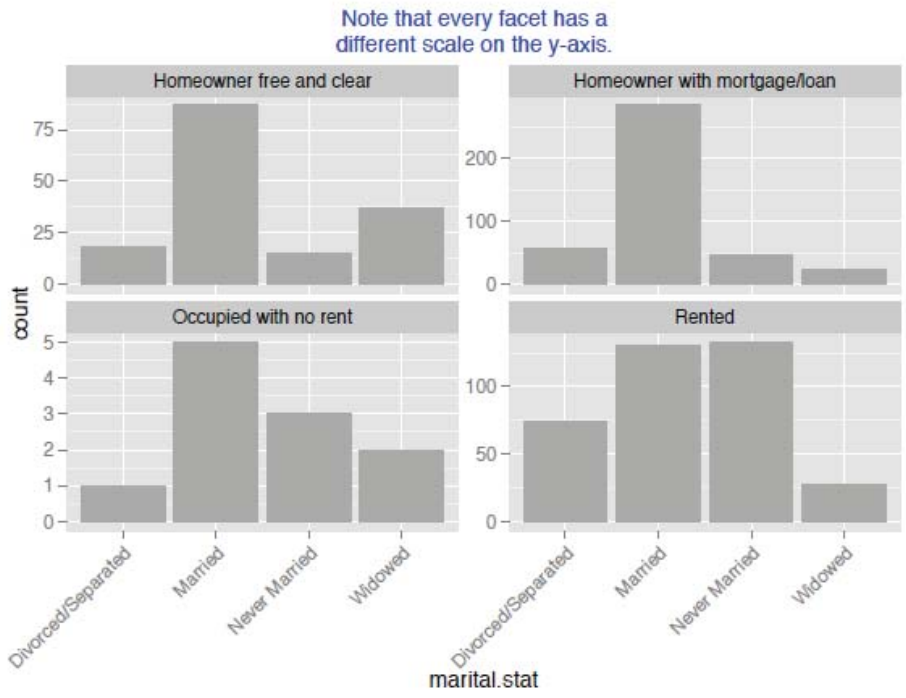


Figure 3.20 Distribution of marital status by housing type: faceted side-by-side barchart

The code for figure 3.19 and figure 3.20:

```
ggplot(custdata2) +                                                    #A
   geom_bar(aes(x=housing.type,  fill=marital.stat ),
       position="dodge") +

   theme(axis.text.x  = element_text(angle  = 45, hjust = 1))     #B


ggplot(custdata2) +                                                    #C
   geom_bar(aes(x=marital.stat),  position="dodge",
       fill="darkgray") +

facet_wrap(~housing.type,  scales="free_y") +                      #D

theme(axis.text.x  = element_text(angle  = 45, hjust = 1))        #E
```

#A Side-by-side barchart
#B Tilt the x-axis labels so they don't overlap. You can also use coord_flip() to rotate the graph, as we saw previously. I prefer coord_flip() because the theme() layer is complicated to use.
#C The faceted barchart.
#D Facet the graph by housing.type. The scales="free_y" argument specifies that each facet has an independently scaled y-axis (the default is that all facets have the same scales on both axes). The argument "free_x" would free the x-axis scaling, and "free" frees both axes.
#E As of this writing, facet_wrap is incompatible with coord_flip, so we have to tilt the x-axis labels.

Let's summarize the visualizations for two variables that we've covered.

Table 3.2 Visualizations for Two Variables

Graph Type	Uses
Line Plot	Shows the relationship between two continuous variables. Best when that relationship is functional, or nearly so.
Scatterplot	Shows the relationship between two continuous variables. Best when the relationship is looser or more "cloudlike" than can be easily seen on a line plot.
Smoothing Curves	Shows underlying "average" relationship, or trend, between two continuous variables. Can also be used to show the relationship between a continuous and a binary or boolean variable: the fraction of "true" values of the discrete variable as a function of the continuous variable.
Hexbin Plot	Shows the relationship between two continuous variables when the data is very dense.
Stacked Barchart	Shows the relationship between two categorical variables (*var1* and *var2*). Highlights the frequencies of each value of *var1*.
Side-by-side Barchart	Shows the relationship between two categorical variables (*var1* and *var2*). Good for comparing the frequencies of each value of *var2* across the values of *var1*. Works best when *var2* is binary.
Filled Barchart	Shows the relationship between two categorical variables (*var1* and *var2*). Good for comparing the relative frequencies of each value of *var2* within each value of *var1*. Works best when *var2* is binary.
Barchart with faceting	Shows the relationship between two categorical variables (*var1* and *var2*). Best for comparing the relative frequencies of each value of *var2* within each value of *var1* when *var2* takes on more than two values.

There are many other variations and visualizations you could use to explore the data; the above set covers some of the most useful and basic graphs. You should try different kinds of graphs to get different insights from the data. It's an interactive process. One graph will raise questions that you can try to answer by replotting the data again, with a different visualization.

Eventually, you have explored your data enough to get a sense of it, and to have spotted most major problems and issues. In the next chapter, we will discuss some ways to address common problems that you may have discovered in the data.

3.3 Summary

At this point, you've gotten a feel for your data. You've explored it through summaries and visualizations; you now have a sense of the quality of your data, and of the relationships among your variables. You've caught and are ready to correct several kinds of data issues -- although you will likely run into more issues as you progress.

Maybe some of the things you've discovered have led you to re-evaluate the question you are trying to answer, or to modify your goals. Maybe you've decided that you need more or different types of data to achieve your goals. This is all good. As we mentioned in the previous chapter, the data science process is made of loops within loops. The data exploration and data cleaning stages (we will discuss cleaning in the next chapter) are two of the more time-consuming -- and also the most important -- stages of the process. Without good data, you can't build good models. Time you spend here is time you don't waste, elsewhere.

Key Takeaways

Take the time to examine your data before diving into the modeling. The summarycommand helps you spot issues with data range, units, data type, and missing or invalid values.

Visualization additionally gives you a sense of data distribution and relationships among variables.

Visualization is an iterative process and helps answer questions about the data. Time spent here is time not wasted during the modeling process.

In the next chapter, we will talk about fixing the issues that you've discovered in the data.