Importing and saving data from CSV files

It is very common for public datasets to be stored in text files. Text files can be read on virtually any computer or operating system, which makes the format nearly universal. They can also be exported and imported to and from programs such as Microsoft Excel, providing a quick and easy way to work with spreadsheet data.

A **tabular** (as in "table") data file is structured in the matrix form, such that each line of text reflects one example, and each example has the same number of features. The feature values on each line are separated by a predefined symbol, known as a **delimiter**. Often, the first line of a tabular data file lists the names of the columns of data. This is called a **header** line.

Perhaps the most common tabular text file format is the **CSV** (**Comma-Separated Values**) file, which as the name suggests, uses the comma as a delimiter. The CSV files can be imported to and exported from many common applications. A CSV file representing the medical dataset constructed previously could be stored as:

```
subject_name,temperature,flu_status,gender,blood_type
John Doe,98.1,FALSE,MALE,O
Jane Doe,98.6,FALSE,FEMALE,AB
Steve Graves,101.4,TRUE,MALE,A
```

Given a patient data file named `pt_data.csv` located in the R working directory, the `read.csv()` function can be used as follows to load the file into R:

```
> pt_data <- read.csv("pt_data.csv", stringsAsFactors = FALSE)
```

This will read the CSV file into a data frame titled `pt_data`. Just as we did previously while constructing a data frame, we need to use the `stringsAsFactors = FALSE` parameter to prevent R from converting all text variables into factors. This step is better left to you, not R, to perform.

> If your dataset resides outside the R working directory, the full path to the CSV file (for example, `/path/to/mydata.csv`) can be used when calling the `read.csv()` function.

By default, R assumes that the CSV file includes a header line listing the names of the features in the dataset. If a CSV file does not have a header, specify the option `header = FALSE`, as shown in the following command, and R will assign default feature names in the `V1` and `V2` forms and so on:

```
> mydata <- read.csv("mydata.csv", stringsAsFactors = FALSE,
                                    header = FALSE)
```

The `read.csv()` function is a special case of the `read.table()` function, which can read tabular data in many different forms, including other delimited formats such as **Tab-Separated Values** (**TSV**). For more detailed information on the `read.table()` family of functions, refer to the R help page using the `?read.table` command.

To save a data frame to a CSV file, use the `write.csv()` function. If your data frame is named `pt_data`, simply enter:

```
> write.csv(pt_data, file = "pt_data.csv", row.names = FALSE)
```

This will write a CSV file with the name `pt_data.csv` to the R working folder. The `row.names` parameter overrides R's default setting, which is to output row names in the CSV file. Unless row names have been added to a data frame, this output is unnecessary and will simply inflate the size of the resulting file.

Exploring and understanding data

After collecting data and loading it into R's data structures, the next step in the machine learning process involves examining the data in detail. It is during this step that you will begin to explore the data's features and examples, and realize the peculiarities that make your data unique. The better you understand your data, the better you will be able to match a machine learning model to your learning problem.

The best way to learn the process of data exploration is with an example. In this section, we will explore the `usedcars.csv` dataset, which contains actual data about used cars recently advertised for sale on a popular U.S. website.

> The `usedcars.csv` dataset is available for download on the Packt Publishing support page for this book. If you are following along with the examples, be sure that this file has been downloaded and saved to your R working directory.

Since the dataset is stored in the CSV form, we can use the `read.csv()` function to load the data into an R data frame:

```
> usedcars <- read.csv("usedcars.csv", stringsAsFactors = FALSE)
```

Given the `usedcars` data frame, we will now assume the role of a data scientist who has the task of understanding the used car data. Although data exploration is a fluid process, the steps can be imagined as a sort of investigation in which questions about the data are answered. The exact questions may vary across projects, but the types of questions are always similar. You should be able to adapt the basic steps of this investigation to any dataset you like, whether large or small.

Exploring the structure of data

One of the first questions to ask in an investigation of a new dataset should be about how the dataset is organized. If you are fortunate, your source will provide a **data dictionary**, which is a document that describes the dataset's features. In our case, the used car data does not come with this documentation, so we'll need to create one on our own.

The `str()` function provides a method to display the structure of R data structures such as data frames, vectors, or lists. It can be used to create the basic outline for our data dictionary:

```
> str(usedcars)
'data.frame':   150 obs. of 6 variables:
 $ year        : int  2011 2011 2011 2011 ...
 $ model       : chr  "SEL" "SEL" "SEL" "SEL" ...
 $ price       : int  21992 20995 19995 17809 ...
 $ mileage     : int  7413 10926 7351 11613 ...
 $ color       : chr  "Yellow" "Gray" "Silver" "Gray" ...
 $ transmission: chr  "AUTO" "AUTO" "AUTO" "AUTO" ...
```

Using such a simple command, we learn a wealth of information about the dataset. The statement `150 obs` informs us that the data includes 150 **observations**, which is just another way of saying that the dataset contains 150 records or examples. The number of observations is often simply abbreviated as *n*. Since we know that the data describes used cars, we can now presume that we have examples of *n = 150* automobiles for sale.

The `6 variables` statement refers to the six features that were recorded in the data. These features are listed by name on separate lines. Looking at the line for the feature called `color`, we can note some additional details:

```
$ color       : chr  "Yellow" "Gray" "Silver" "Gray" ...
```

After the variable's name, the `chr` label tells us that the feature is `character` type. In this dataset, three of the variables are character while three are noted as `int`, which indicates `integer` type. Although the `usedcars` dataset includes only `character` and `integer` variables, you are also likely to encounter `num` or `numeric` type while using noninteger data. Any factors would be listed as `factor` type. Following each variable's type, R presents a sequence of the first few feature values. The values `"Yellow" "Gray" "Silver" "Gray"` are the first four values of the `color` feature.

Applying a bit of the subject-area knowledge to the feature names and values allows us to make some assumptions about what the variables represent. The `year` variable could refer to the year the vehicle was manufactured or it could specify the year the advertisement was posted. We will have to investigate this feature more in detail later, since the four example values (`2011 2011 2011 2011`) could be used to argue for either possibility. The `model`, `price`, `mileage`, `color`, and `transmission` variables most likely refer to the characteristics of the car for sale.

Although our data seems to have been given meaningful variable names, this is not always the case. Sometimes datasets have features with nonsensical names or codes like `V1`. In these cases it may be necessary to do additional sleuthing to determine what a feature actually represents. Still, even with helpful feature names, it is always prudent to be skeptical about the labels you have been provided with. Let's investigate further.

Exploring numeric variables

To investigate the numeric variables in the used car data, we will employ a common set of measurements to describe values known as **summary statistics**. The `summary()` function displays several common summary statistics. Let's take a look at a single feature, `year`:

```
> summary(usedcars$year)
   Min.  1st Qu.  Median    Mean 3rd Qu.    Max.
  2000     2008     2009    2009   2010    2012
```

Even if you aren't already familiar with summary statistics, you may be able to guess some of them from the heading before the `summary()` output. Ignoring the meaning of the values for now, the fact that we see numbers such as `2000`, `2008`, and `2009` could lead us to believe that the `year` variable indicates the year of manufacture rather than the year the advertisement was posted, since we know the vehicles were recently listed for sale.

We can also use the `summary()` function to obtain summary statistics for several `numeric` variables at the same time:

```
> summary(usedcars[c("price", "mileage")])
     price           mileage
 Min.   : 3800   Min.   :  4867
 1st Qu.:10995   1st Qu.: 27200
 Median :13592   Median : 36385
 Mean   :12962   Mean   : 44261
 3rd Qu.:14904   3rd Qu.: 55125
 Max.   :21992   Max.   :151479
```

The six summary statistics that the `summary()` function provides are simple, yet powerful tools to investigate data. They can be divided into two types: measures of center and measures of spread.

Measuring the central tendency – mean and median

Measures of **central tendency** are a class of statistics used to identify a value that falls in the middle of a set of data. You most likely are already familiar with one common measure of center: the average. In common use, when something is deemed average, it falls somewhere between the extreme ends of the scale. An average student might have marks falling in the middle of his or her classmates; an average weight is neither unusually light nor heavy. An average item is typical and not too unlike the others in the group. You might think of it as an exemplar by which all the others are judged.

In statistics, the average is also known as the **mean**, which is a measurement defined as the sum of all values divided by the number of values. For example, to calculate the mean income in a group of three people with incomes of $36,000, $44,000, and $56,000, use the following command:

```
> (36000 + 44000 + 56000) / 3
[1] 45333.33
```

R also provides a `mean()` function, which calculates the mean for a vector of numbers:

```
> mean(c(36000, 44000, 56000))
[1] 45333.33
```

The mean income of this group of people is about $45,333. Conceptually, this can be imagined as the income each person would have, if the total amount of income were divided equally across every person.

Recall that the preceding `summary()` output listed mean values for the `price` and `mileage` variables. The means suggest that the typical used car in this dataset was listed at a price of $12,962 and had an odometer reading of 44,261. What does this tell us about our data? Since the average price is relatively low, we might expect that the dataset contains economy class cars. Of course, the data can also include late-model luxury cars with high mileage, but the relatively low mean mileage statistic doesn't provide evidence to support this hypothesis. On the other hand, it doesn't provide evidence to ignore the possibility either. We'll need to keep this in mind as we examine the data further.

Although the mean is by far the most commonly cited statistic to measure the center of a dataset, it is not always the most appropriate one. Another commonly used measure of central tendency is the **median**, which is the value that occurs halfway through an ordered list of values. As with the mean, R provides a `median()` function, which we can apply to our salary data, as shown in the following example:

```
> median(c(36000, 44000, 56000))
[1] 44000
```

Because the middle value is `44000`, the median income is $44,000.

If a dataset has an even number of values, there is no middle value. In this case, the median is commonly calculated as the average of the two values at the center of the ordered list. For example, the median of the values 1, 2, 3, and 4 is 2.5.

At the first glance, it seems like the median and mean are very similar measures. Certainly, the mean value of $45,333 and the median value of $44,000 are not very different. Why have two measures of central tendency? The reason is due to the fact that the mean and median are affected differently by the values falling at the far ends of the range. In particular, the mean is highly sensitive to **outliers**, or values that are atypically high or low in relation to the majority of data. Because the mean is sensitive to outliers, it is more likely to be shifted higher or lower by a small number of extreme values.

Recall again the reported median values in the `summary()` output for the used car dataset. Although the mean and median price are fairly similar (differing by approximately five percent), there is a much larger difference between the mean and median for mileage. For mileage, the mean of 44,261 is approximately 20 percent more than the median of 36,385. Since the mean is more sensitive to extreme values than the median, the fact that the mean is much higher than the median might lead us to suspect that there are some used cars in the dataset with extremely high mileage values. To investigate this further, we'll need to add additional summary statistics to our analysis.

Measuring spread – quartiles and the five-number summary

Measuring the mean and median provides one way to quickly summarize the values, but these measures of center tell us little about whether or not there is diversity in the measurements. To measure the diversity, we need to employ another type of summary statistics that is concerned with the **spread** of data, or how tightly or loosely the values are spaced. Knowing about the spread provides a sense of the data's highs and lows and whether most values are like or unlike the mean and median.

The **five-number summary** is a set of five statistics that roughly depict the spread of a feature's values. All five of the statistics are included in the output of the `summary()` function. Written in order, they are:

1. Minimum (`Min.`)
2. First quartile, or Q1 (`1st Qu.`)
3. Median, or Q2 (`Median`)
4. Third quartile, or Q3 (`3rd Qu.`)
5. Maximum (`Max.`)

As you would expect, minimum and maximum are the most extreme feature values, indicating the smallest and largest values, respectively. R provides the `min()` and `max()` functions to calculate these values on a vector of data.

The span between the minimum and maximum value is known as the **range**. In R, the `range()` function returns both the minimum and maximum value. Combining `range()` with the `diff()` difference function allows you to examine the range of data with a single line of code:

```
> range(usedcars$price)
[1]  3800 21992
> diff(range(usedcars$price))
[1] 18192
```

The first and third quartiles—Q1 and Q3—refer to the value below or above which one quarter of the values are found. Along with the (Q2) median, the **quartiles** divide a dataset into four portions, each with the same number of values.

Quartiles are a special case of a type of statistics called **quantiles**, which are numbers that divide data into equally sized quantities. In addition to quartiles, commonly used quantiles include **tertiles** (three parts), **quintiles** (five parts), **deciles** (10 parts), and **percentiles** (100 parts).

Percentiles are often used to describe the ranking of a value; for instance, a student whose test score was ranked at the 99th percentile performed better than, or equal to, 99 percent of the other test takers.

The middle 50 percent of data between the first and third quartiles is of particular interest because it in itself is a simple measure of spread. The difference between Q1 and Q3 is known as the **Interquartile Range** (**IQR**), and it can be calculated with the `IQR()` function:

```
> IQR(usedcars$price)
[1] 3909.5
```

We could have also calculated this value by hand from the `summary()` output for the `usedcars$price` variable by computing *14904 – 10995 = 3909*. The small difference between our calculation and the `IQR()` output is due to the fact that R automatically rounds the `summary()` output.

The `quantile()` function provides a robust tool to identify quantiles for a set of values. By default, the `quantile()` function returns the five-number summary. Applying the function to the used car data results in the same statistics as done earlier:

```
> quantile(usedcars$price)
     0%     25%     50%     75%    100%
 3800.0 10995.0 13591.5 14904.5 21992.0
```

While computing quantiles, there are many methods to handle ties among values and datasets with no middle value. The `quantile()` function allows you to specify among nine different algorithms by specifying the `type` parameter. If your project requires a precisely defined quantile, it is important to read the function documentation using the `?quantile` command.

If we specify an additional `probs` parameter using a vector denoting cut points, we can obtain arbitrary quantiles, such as the 1st and 99th percentiles:

```
> quantile(usedcars$price, probs = c(0.01, 0.99))
      1%      99%
 5428.69 20505.00
```

The `seq()` function is used to generate vectors of evenly-spaced values. This makes it easy to obtain other slices of data, such as the quintiles (five groups), as shown in the following command:

```
> quantile(usedcars$price, seq(from = 0, to = 1, by = 0.20))
     0%     20%     40%     60%     80%    100%
 3800.0 10759.4 12993.8 13992.0 14999.0 21992.0
```

Equipped with an understanding of the five-number summary, we can re-examine the used car `summary()` output. On the `price` variable, the minimum was $3,800 and the maximum was $21,992. Interestingly, the difference between the minimum and Q1 is about $7,000, as is the difference between Q3 and the maximum; yet, the difference from Q1 to the median to Q3 is roughly $2,000. This suggests that the lower and upper 25 percent of values are more widely dispersed than the middle 50 percent of values, which seem to be more tightly grouped around the center. We see a similar trend with the `mileage` variable, which is not unsurprising. As you will learn later in this chapter, this pattern of spread is common enough that it has been called a "normal" distribution of data.

The spread of the `mileage` variable also exhibits another interesting property: the difference between Q3 and the maximum value is far greater than that between the minimum value and Q1. In other words, the larger values are far more spread out than the smaller values.

This finding explains why the mean value is much greater than the median. Because the mean is sensitive to extreme values, it is pulled higher, while the median stays relatively in the same place. This is an important property, which becomes more apparent when the data is presented visually.

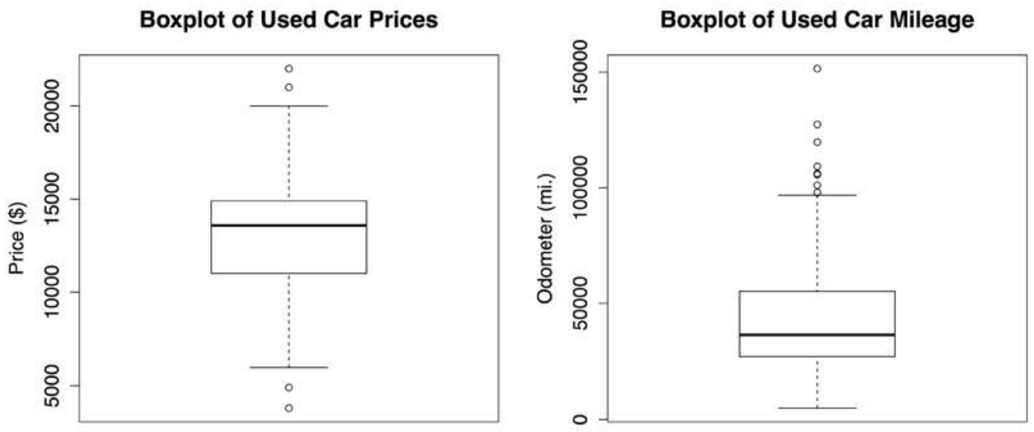
Visualizing numeric variables – boxplots

Visualizing numeric variables can be helpful in diagnosing data problems. A common visualization of the five-number summary is **boxplot**, also known as a **box-and-whiskers** plot. The boxplot displays the center and spread of a numeric variable in a format that allows you to quickly obtain a sense of the range and skew of a variable or compare it to other variables.

Let's take a look at a boxplot for the used car price and mileage data. To obtain a boxplot for a variable, we will use the `boxplot()` function. We will also specify a pair of extra parameters, `main` and `ylab`, to add a title to the figure and label the *y* axis (the vertical axis), respectively. The commands to create the `price` and `mileage` boxplots are:

```
> boxplot(usedcars$price, main="Boxplot of Used Car Prices",
                  ylab="Price ($)")
> boxplot(usedcars$mileage, main="Boxplot of Used Car Mileage",
                  ylab="Odometer (mi.)")
```

R will produce figures as follows:



The box-and-whiskers plot depicts the five-number summary values using the horizontal lines and dots. The horizontal lines forming the box in the middle of each figure represent Q1, Q2 (the median), and Q3 while reading the plot from the bottom to the top. The median is denoted by the dark line, which lines up with $13,592 on the vertical axis for `price` and 36,385 mi. on the vertical axis for `mileage`.

In simple boxplots such as those in the preceding diagram, the width of the box-and-whiskers plot is arbitrary and does not illustrate any characteristic of the data. For more sophisticated analyses, it is possible to use the shape and size of the boxes to facilitate comparisons of the data across several groups. To learn more about such features, begin by examining the `notch` and `varwidth` options in the R `boxplot()` documentation by typing the `?boxplot` command.

The minimum and maximum values can be illustrated using the whiskers that extend below and above the box; however, a widely used convention only allows the whiskers to extend to a minimum or maximum of 1.5 times the IQR below Q1 or above Q3. Any values that fall beyond this threshold are considered outliers and are denoted as circles or dots. For example, recall that the IQR for the `price` variable was 3,909 with a Q1 of 10,995 and a Q3 of 14,904. An outlier is therefore any value that is less than *10995 - 1.5 * 3909= 5131.5* or greater than *14904 + 1.5 * 3909 = 20767.5*.

The plot shows two such outliers on both the high and low ends. On the `mileage` boxplot, there are no outliers on the low end and thus, the bottom whisker extends to the minimum value, 4,867. On the high end, we see several outliers beyond the 100,000 mile mark. These outliers are responsible for our earlier finding, which noted that the mean value was much greater than the median.

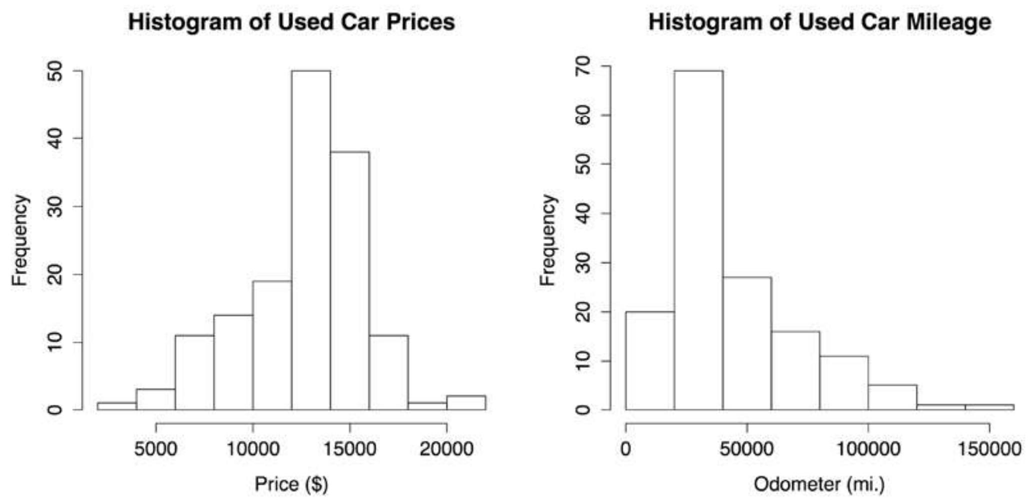
Visualizing numeric variables – histograms

A **histogram** is another way to graphically depict the spread of a numeric variable. It is similar to a boxplot in a way that it divides the variable's values into a predefined number of portions or **bins** that act as containers for values. Their similarities end there, however. On one hand, a boxplot requires that each of the four portions of data must contain the same number of values, and widens or narrows the bins as needed. On the other hand, a histogram uses any number of bins of an identical width, but allows the bins to contain different number of values.

We can create a histogram for the used car price and mileage data using the `hist()` function. As we did with the boxplot, we will specify a title for the figure using the `main` parameter, and label the *x* axis with the `xlab` parameter. The commands to create the histograms are:

```
> hist(usedcars$price, main = "Histogram of Used Car Prices",
              xlab = "Price ($)")
> hist(usedcars$mileage, main = "Histogram of Used Car Mileage",
              xlab = "Odometer (mi.)")
```

This produces the following diagram:



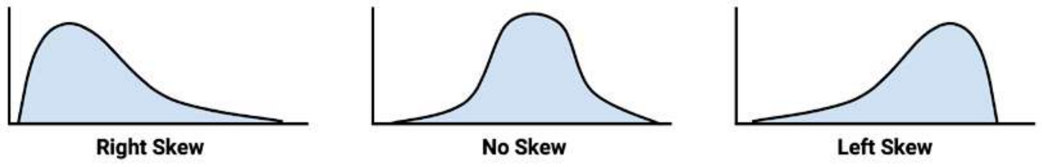
The histogram is composed of a series of bars with heights indicating the count, or **frequency** of values falling within each of the equal width bins partitioning the values. The vertical lines that separate the bars, as labeled on the horizontal axis, indicate the start and end points of the range of values for the bin.

> You may have noticed that the preceding histograms have a different number of bins. This is because the `hist()` function attempts to identify a reasonable number of bins for the variable's range. If you'd like to override this default, use the `breaks` parameter. Supplying an integer like `breaks = 10` would create exactly 10 bins of equal width; supplying a vector like `c(5000, 10000, 15000, 20000)` would create bins that break at the specified values.

On the `price` histogram, each of the 10 bars spans an interval of $2,000, beginning at $2,000 and ending at $22,000. The tallest bar at the center of the figure covers the $12,000 to $14,000 range and has a frequency of 50. Since we know that our data includes 150 cars, we know that one-third of all the cars are priced from $12,000 to $14,000. Nearly 90 cars—more than half—are priced from $12,000 to $16,000.

The `mileage` histogram includes eight bars indicating bins of 20,000 miles each, beginning at 0 and ending at 160,000 miles. Unlike the price histogram, the tallest bar is not at the center of the data, but on the left-hand side of the diagram. The 70 cars contained in this bin have odometer readings from 20,000 to 40,000 miles.

You might also notice that the shape of the two histograms is somewhat different. It seems that the used car prices tend to be evenly divided on both sides of the middle, while the car mileages stretch further to the right. This characteristic is known as **skew**, or more specifically right skew, because the values on the high end (right side) are far more spread out than the values on the low end (left side). As shown in the following diagram, histograms of skewed data look stretched on one of the sides:

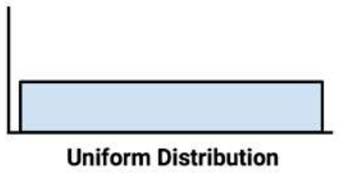


The ability to quickly diagnose such patterns in our data is one of the strengths of the histogram as a data exploration tool. This will become even more important as we start examining other patterns of spread in numeric data.

Understanding numeric data – uniform and normal distributions

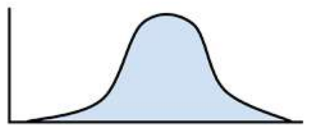
Histograms, boxplots, and statistics describing the center and spread provide ways to examine the distribution of a variable's values. A variable's **distribution** describes how likely a value is to fall within various ranges.

If all the values are equally likely to occur—say, for instance, in a dataset recording the values rolled on a fair six-sided die—the distribution is said to be **uniform**. A uniform distribution is easy to detect with a histogram, because the bars are approximately the same height. When visualized with a histogram, it may look something like the following diagram:



It's important to note that not all random events are uniform. For instance, rolling a weighted six-sided trick die would result in some numbers coming up more often than others. While each roll of the die results in a randomly selected number, they are not equally likely.

Take, for instance, the used car data. This is clearly not uniform, since some values are seemingly far more likely to occur than others. In fact, on the price histogram, it seems that values grow less likely to occur as they are further away from both sides of the center bar, resulting in a bell-shaped distribution of data. This characteristic is so common in real-world data that it is the hallmark of the so-called **normal distribution**. The stereotypical bell-shaped curve of normal distribution is shown in the following diagram:


Normal Distribution

Although there are numerous types of non-normal distributions, many real-world phenomena generate data that can be described by the normal distribution. Therefore, the normal distribution's properties have been studied in great detail.

Measuring spread – variance and standard deviation

Distributions allow us to characterize a large number of values using a smaller number of parameters. The normal distribution, which describes many types of real-world data, can be defined with just two: center and spread. The center of normal distribution is defined by its mean value, which we have used earlier. The spread is measured by a statistic called the **standard deviation**.

In order to calculate the standard deviation, we must first obtain the **variance**, which is defined as the average of the squared differences between each value and the mean value. In mathematical notation, the variance of a set of *n* values of *x* is defined by the following formula. The Greek letter *mu* (similar in appearance to an *m* or *u*) denotes the mean of the values, and the variance itself is denoted by the Greek letter *sigma* squared (similar to a *b* turned sideways):

$$\mathrm{Var(X)} = \sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2$$

The standard deviation is the square root of the variance, and is denoted by *sigma*, as shown in the following formula:

$$\mathrm{StdDev(X)} = \sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2}$$

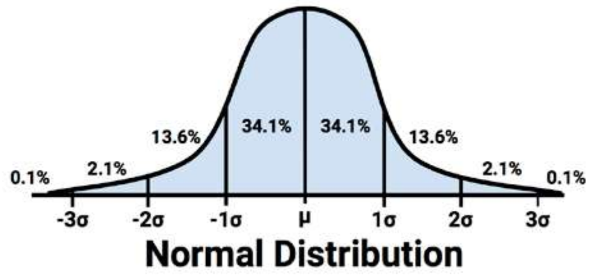
The `var()` and `sd()` functions can be used to obtain the variance and standard deviation in R. For example, computing the variance and standard deviation on our `price` and `mileage` variables, we find:

```
> var(usedcars$price)
[1] 9749892
> sd(usedcars$price)
[1] 3122.482
> var(usedcars$mileage)
[1] 728033954
> sd(usedcars$mileage)
[1] 26982.1
```

While interpreting the variance, larger numbers indicate that the data are spread more widely around the mean. The standard deviation indicates, on average, how much each value differs from the mean.

[If you compute these statistics by hand using the formulas in the preceding diagrams, you will obtain a slightly different result than the built-in R functions. This is because the preceding formulae use the population variance (which divides by *n*), while R uses the sample variance (which divides by *n - 1*). Except for very small datasets, the distinction is minor.]

The standard deviation can be used to quickly estimate how extreme a given value is under the assumption that it came from a normal distribution. The **68-95-99.7 rule** states that 68 percent of the values in a normal distribution fall within one standard deviation of the mean, while 95 percent and 99.7 percent of the values fall within two and three standard deviations, respectively. This is illustrated in the following diagram:



Applying this information to the used car data, we know that since the mean and standard deviation of `price` were $12,962 and $3,122, respectively, assuming that the prices are normally distributed, approximately 68 percent of cars in our data were advertised at prices between *$12,962 - $3,122 = $9,840* and *$12,962 + $3,122 = $16,804*.

Although, strictly speaking, the 68-95-99.7 rule only applies to normal distributions, the basic principle applies to any data; values more than three standard deviations away from the mean are exceedingly rare events.

Exploring categorical variables

If you recall, the used car dataset had three categorical variables: `model`, `color`, and `transmission`. Because we used the `stringsAsFactors = FALSE` parameter while loading the data, R has left them as the `character` (`chr`) type vectors rather than automatically converting them into `factor` type. Additionally, we might consider treating the year variable as categorical; although it has been loaded as a `numeric` (`int`) type vector, each year is a category that could apply to multiple cars.

In contrast to `numeric` data, categorical data is typically examined using tables rather than summary statistics. A table that presents a single categorical variable is known as a **one-way table**. The `table()` function can be used to generate one-way tables for our used car data:

```
> table(usedcars$year)
2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012
   3    1    1    1    3    2    6   11   14   42   49   16    1
> table(usedcars$model)
 SE SEL SES
78  23  49
> table(usedcars$color)
 Black    Blue    Gold    Gray  Green     Red Silver  White Yellow
    35      17       1      16      5      25     32     16      3
```

The `table()` output lists the categories of the nominal variable and a count of the number of values falling into this category. Since we know that there are 150 used cars in the dataset, we can determine that roughly one-third of all the cars were manufactured in the year `2010`, given that *49/150 = 0.327*.

R can also perform the calculation of table proportions directly, by using the `prop.table()` command on a table produced by the `table()` function:

```
> model_table <- table(usedcars$model)
> prop.table(model_table)
       SE       SEL       SES
0.5200000 0.1533333 0.3266667
```

The results of `prop.table()` can be combined with other R functions to transform the output. Suppose that we would like to display the results in percentages with a single decimal place. We can do this by multiplying the proportions by 100, then using the `round()` function while specifying `digits = 1`, as shown in the following example:

```
> color_pct <- table(usedcars$color)
> color_pct <- prop.table(color_table) * 100
> round(color_pct, digits = 1)
Black   Blue    Gold   Gray  Green    Red Silver  White Yellow
 23.3   11.3     0.7   10.7    3.3   16.7   21.3   10.7    2.0
```

Although this includes the same information as the default `prop.table()` output, this is easier to read. The results show that black is the most common color, since nearly a quarter (23.3 percent) of all the advertised cars are `Black`. `Silver` is a close second with 21.3 percent and `Red` is third with 16.7 percent.

Measuring the central tendency – the mode

In statistics terms, the **mode** of a feature is the value occurring most often. Like the mean and median, the mode is another measure of central tendency. It is often used for categorical data, since the mean and median are not defined for nominal variables.

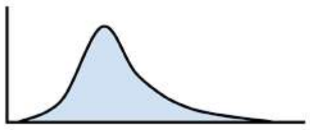
For example, in the used car data, the mode of the `year` variable is 2010, while the modes for the `model` and `color` variables are `SE` and `Black`, respectively. A variable may have more than one mode; a variable with a single mode is **unimodal**, while a variable with two modes is **bimodal**. Data having multiple modes is more generally called **multimodal**.

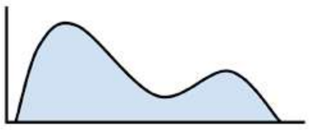
Although you might suspect that you could use the `mode()` function, R uses this to obtain the type of variable (as in `numeric`, `list`, and so on) rather than the statistical mode. Instead, to find the statistical mode, simply look at the table output of the category with the greatest number of values.

The mode or modes are used in a qualitative sense to gain an understanding of important values. Yet, it would be dangerous to place too much emphasis on the mode, since the most common value is not necessarily a majority. For instance, although `Black` was the most common value for the `color` variable, black cars were only about a quarter of all advertised cars.

It is best to think about modes in relation to the other categories. Is there one category that dominates all the others or are there several? From here, we may ask what the most common values tell us about the variable being measured. If black and silver are commonly used car colors, we might assume that the data are for luxury cars, which tend to be sold in more conservative colors. These colors could also indicate economy cars, which are sold with fewer color options. We will keep this question in mind as we continue to examine this data.

Thinking about modes as common values allows us to apply the concept of statistical mode to the numeric data. Strictly speaking, it would be unlikely to have a mode for a continuous variable, since no two values are likely to repeat. Yet, if we think about modes as the highest bars on a histogram, we can discuss the modes of variables such as `price` and `mileage`. It can be helpful to consider mode while exploring the numeric data, particularly to examine whether or not the data is multimodal.


Unimodal Distribution


Bimodal Distribution

Exploring relationships between variables

So far, we have examined variables one at a time, calculating only **univariate** statistics. During our investigation, we raised questions that we were unable to answer at that time:

- Does the `price` data imply that we are examining only economy-class cars or are there also luxury cars with high mileage?
- Do relationships between the `model` and `color` data provide insight into the types of cars we are examining?

These type of questions can be addressed by looking at **bivariate** relationships, which consider the relationship between two variables. Relationships of more than two variables are called **multivariate** relationships. Let's begin with the bivariate case.

Visualizing relationships – scatterplots

A **scatterplot** is a diagram that visualizes a bivariate relationship. It is a two-dimensional figure in which dots are drawn on a coordinate plane using the values of one feature to provide the horizontal *x* coordinates and the values of another feature to provide the vertical *y* coordinates. Patterns in the placement of dots reveal the underlying associations between the two features.

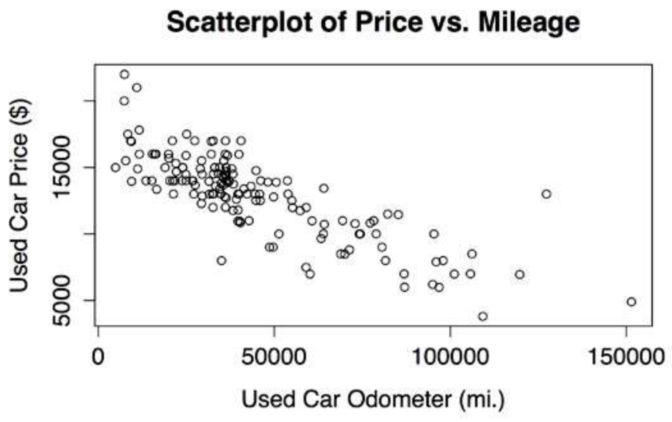
To answer our question about the relationship between `price` and `mileage`, we will examine a scatterplot. We'll use the `plot()` function along with the `main`, `xlab` and `ylab` parameters used previously to label the diagram.

To use `plot()`, we need to specify `x` and `y` vectors containing the values used to position the dots on the figure. Although the conclusions would be the same regardless of the variable used to supply the *x* and *y* coordinates, convention dictates that the *y* variable is the one that is presumed to depend on the other (and is therefore known as the dependent variable). Since a seller cannot modify the odometer reading, mileage is unlikely to be dependent on the car's price. Instead, our hypothesis is that the price depends on the odometer mileage. Therefore, we will use `price` as the *y*, or dependent, variable.

The full command to create our scatterplot is:

```
> plot(x = usedcars$mileage, y = usedcars$price,
          main = "Scatterplot of Price vs. Mileage",
          xlab = "Used Car Odometer (mi.)",
          ylab = "Used Car Price ($)")
```

This results in the following scatterplot:



Using the scatterplot, we notice a clear relationship between the price of a used car and the odometer reading. To read the plot, examine how values of the *y* axis variable change as the values on the *x* axis increase. In this case, car prices tend to be lower as the mileage increases. If you have ever sold or shopped for a used car, this is not a profound insight.

Perhaps a more interesting finding is the fact that there are very few cars that have both high price and high mileage, aside from a lone outlier at about 125,000 miles and $14,000. The absence of more points like this provides evidence to support a conclusion that our data is unlikely to include any high mileage luxury cars. All of the most expensive cars in the data, particularly those above $17,500, seem to have extraordinarily low mileage, implying that we could be looking at a brand new type of car retailing for about $20,000.

The relationship we've found between car prices and mileage is known as a negative association, because it forms a pattern of dots in a line sloping downward. A positive association would appear to form a line sloping upward. A flat line, or a seemingly random scattering of dots, is evidence that the two variables are not associated at all. The strength of a linear association between two variables is measured by a statistic known as **correlation**. Correlations are discussed in detail in *Chapter 6*, *Forecasting Numeric Data – Regression Methods*, which covers the methods for modeling linear relationships.

[Keep in mind that not all associations form straight lines. Sometimes the dots form a *U* shape, or a *V* shape; sometimes the pattern seems to be weaker or stronger for increasing values of the `x` or `y` variable. Such patterns imply that the relationship between the two variables is not linear.]

Examining relationships – two-way cross-tabulations

To examine a relationship between two nominal variables, a **two-way cross-tabulation** is used (also known as a **crosstab** or **contingency table**). A cross-tabulation is similar to a scatterplot in that it allows you to examine how the values of one variable vary by the values of another. The format is a table in which the rows are the levels of one variable, while the columns are the levels of another. Counts in each of the table's cells indicate the number of values falling into the particular row and column combination.

To answer our earlier question about whether there is a relationship between car `model` and `color`, we will examine a crosstab. There are several functions to produce two-way tables in R, including `table()`, which we used for one-way tables. The `CrossTable()` option in the `gmodels` package by Gregory R. Warnes is perhaps the most user-friendly function, as it presents the row, column, and margin percentages in a single table, saving us the trouble of combining this data ourselves. To install the `gmodels` package, type:

```
> install.packages("gmodels")
```

After the package installs, type `library(gmodels)` to load the package. You will need to do this during each R session in which you plan on using the `CrossTable()` function.

Before proceeding with our analysis, let's simplify our project by reducing the number of levels in the `color` variable. This variable has nine levels, but we don't really need this much detail. What we are really interested in is whether or not the car's color is conservative. Toward this end, we'll divide the nine colors into two groups: the first group will include the conservative colors `Black`, `Gray`, `Silver`, and `White`; and the second group will include `Blue`, `Gold`, `Green`, `Red`, and `Yellow`. We will create a binary indicator variable (often called a **dummy variable**), indicating whether or not the car's color is conservative by our definition. Its value will be `1` if true, `0` otherwise:

```
> usedcars$conservative <-
       usedcars$color %in% c("Black", "Gray", "Silver", "White")
```

You may have noticed a new command here: the `%in%` operator returns `TRUE` or `FALSE` for each value in the vector on the left-hand side of the operator depending on whether the value is found in the vector on the right-hand side. In simple terms, you can translate this line as "Is the used car color in the set of `Black`, `Gray`, `Silver`, and `White`?"

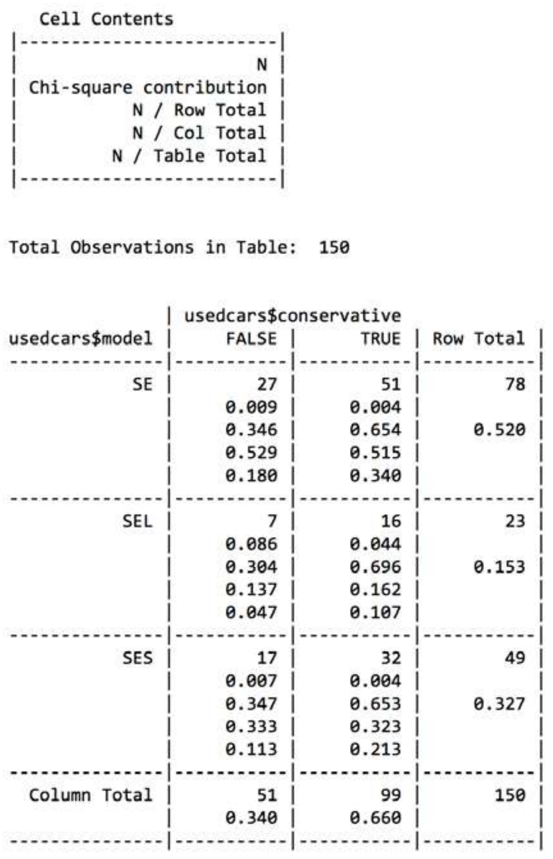
Examining the `table()` output for our newly created variable, we see that about two-thirds of the cars have conservative colors, while one-third do not have conservative colors:

```
> table(usedcars$conservative)
FALSE  TRUE
   51    99
```

Now, let's look at a cross-tabulation to see how the proportion of conservatively colored cars varies by the model. Since we're assuming that the model of the car dictates the choice of color, we'll treat the conservative color indicator as the dependent (`y`) variable. The `CrossTable()` command is therefore:

```
> CrossTable(x = usedcars$model, y = usedcars$conservative)
```

The preceding command results in the following table:



```
   Cell Contents
|-------------------------|
|                       N |
| Chi-square contribution |
|           N / Row Total |
|           N / Col Total |
|         N / Table Total |
|-------------------------|


Total Observations in Table:  150


               | usedcars$conservative
usedcars$model |     FALSE |      TRUE | Row Total |
---------------|-----------|-----------|-----------|
            SE |        27 |        51 |        78 |
               |     0.009 |     0.004 |           |
               |     0.346 |     0.654 |     0.520 |
               |     0.529 |     0.515 |           |
               |     0.180 |     0.340 |           |
---------------|-----------|-----------|-----------|
           SEL |         7 |        16 |        23 |
               |     0.086 |     0.044 |           |
               |     0.304 |     0.696 |     0.153 |
               |     0.137 |     0.162 |           |
               |     0.047 |     0.107 |           |
---------------|-----------|-----------|-----------|
           SES |        17 |        32 |        49 |
               |     0.007 |     0.004 |           |
               |     0.347 |     0.653 |     0.327 |
               |     0.333 |     0.323 |           |
               |     0.113 |     0.213 |           |
---------------|-----------|-----------|-----------|
  Column Total |        51 |        99 |       150 |
               |     0.340 |     0.660 |           |
---------------|-----------|-----------|-----------|
```

There is a wealth of data in the `CrossTable()` output. The legend at the top (labeled `Cell Contents`) indicates how to interpret each value. The rows in the table indicate the three models of used cars: `SE`, `SEL`, and `SES` (plus an additional row for the total across all models). The columns indicate whether or not the car's color is conservative (plus a column totaling across both types of color). The first value in each cell indicates the number of cars with that combination of model and color. The proportions indicate that the cell's proportion is relative to the chi-square statistic, row's total, column's total, and table's total.

What we are most interested in is the row proportion for conservative cars for each model. The row proportions tell us that 0.654 (65 percent) of `SE` cars are colored conservatively in comparison to 0.696 (70 percent) of `SEL` cars and 0.653 (65 percent) of `SES`. These differences are relatively small, suggesting that there are no substantial differences in the types of colors chosen by the model of the car.

The chi-square values refer to the cell's contribution in the **Pearson's Chi-squared test for independence** between two variables. This test measures how likely it is that the difference in the cell counts in the table is due to chance alone. If the probability is very low, it provides strong evidence that the two variables are associated.

You can obtain the chi-squared test results by adding an additional parameter specifying `chisq = TRUE` while calling the `CrossTable()` function. In this case, the probability is about 93 percent, suggesting that it is very likely that the variations in cell count are due to chance alone and not due to a true association between the model and the color.

Summary

In this chapter, we learned about the basics of managing data in R. We started by taking an in-depth look at the structures used for storing various types of data. The foundational R data structure is the vector, which is extended and combined into more complex data types such as lists and data frames. The data frame is an R data structure that corresponds to the notion of a dataset, having both features and examples. R provides functions for reading and writing data frames to spreadsheet-like tabular data files.

We then explored a real-world dataset containing data on used car prices. We examined numeric variables using common summary statistics of center and spread, and visualized relationships between prices and odometer readings with a scatterplot. We examined nominal variables using tables. In examining the used car data, we followed an exploratory process that can be used to understand any dataset. These skills will be required for the other projects throughout this book.

Now that we have spent some time understanding the basics of data management with R, you are ready to begin using machine learning to solve real-world problems. In the next chapter, we will tackle our first classification task using nearest neighbor methods.