Chapter 3

Working with Data

Data is the starting point for all data mining—without it there is nothing to mine. In today's world, there is certainly no shortage of data, but turning that data into information, knowledge, and, eventually, wisdom is not a simple matter. We often think of data as being numbers or categories. But data can also be text, images, videos, and sounds. Data mining generally only deals with numbers and categories. Often, the other forms of data can be mapped into numbers and categories if we wish to analyse them using the approaches we present here.

Whilst data abounds in our modern era, we still need to scout around to obtain the data we need. Many of today's organisations maintain massive warehouses of data. This provides a fertile ground for sourcing data but also an extensive headache for us in navigating through a massive landscape.

An early step in a data mining project is to gather all the required data together. This seemingly simple task can be a significant burden on the budgeted resources for data mining, perhaps consuming up to 70–90% of the elapsed time of a project. It should not be underestimated.

When bringing data together, a number of issues need to be considered. These include the provenance (source and purpose) and quality (accuracy and reliability) of the data. Data collected for different purposes may well store different information in confusingly similar ways. Also, some data requires appropriate permission for its use, and the privacy of anyone the data relates to needs to be considered. Time spent at this stage getting to know your data will be time well spent.

In this chapter, we introduce data, starting with the language we use to describe and talk about data.

G. Williams, *Data Mining with Rattle and R: The Art of Excavating Data for Knowledge Discovery*, Use R, DOI 10.1007/978-1-4419-9890-3_3, © Springer Science+Business Media, LLC 2011

3.1 Data Nomenclature

Data miners have a plethora of terminology, often using many different terms to describe the same concept. A lot of this confusion of terminology is due to the history of data mining, with its roots in many different disciplines, including databases, machine learning, and statistics. Throughout this book, we will use a consistent and generally accepted nomenclature, which we introduce here.

We refer to a collection of data as a **dataset**. This might be called in mathematical terms a *matrix* or in database terms a *table*. Figure 3.1 illustrates a dataset annotated with our chosen nomenclature.

We often view a dataset as consisting of rows, which we refer to as **observations**, and those observations are recorded in terms of **variables**, which form the columns of the dataset. Observations are also known as *entities*, *rows*, *records*, and *objects*. Variables are also known as *fields*, *columns*, *attributes*, *characteristics*, and *features*. The **dimension** of a dataset refers to the number of observations (rows) and the number of variables (columns).

Variables can serve different roles: as **input variables** or **output variables**. Input variables are *measured* or *preset* data items. They might also be known as *predictors*, *covariates*, *independent variables*, *observed variables*, and *descriptive variables*. An output variable may be identified in the data. These are variables that are often "influenced" by the input variables. They might also be known as *target*, *response*, or *dependent variables*. In data mining, we often build models to predict the output variables in terms of the input variables. Early on in a data mining project, we may not know for sure which variables, if any, are output variables. For some data mining tasks (e.g., clustering), we might not have any output variables.

Some variables may only serve to uniquely identify the observations. Common examples include social security and other such government identity numbers. Even the date may be a unique identifier for particular observations. We refer to such variables as **identifiers**. Identifiers are not normally used in modelling, particularly those that are essentially randomly generated.

Variables can store different types of data. The values might be the names or the qualities of objects, represented as character strings. Or the values may be quantitative and thereby represented numerically. At a high level we often only need to distinguish these two broad types of data, as we do here.

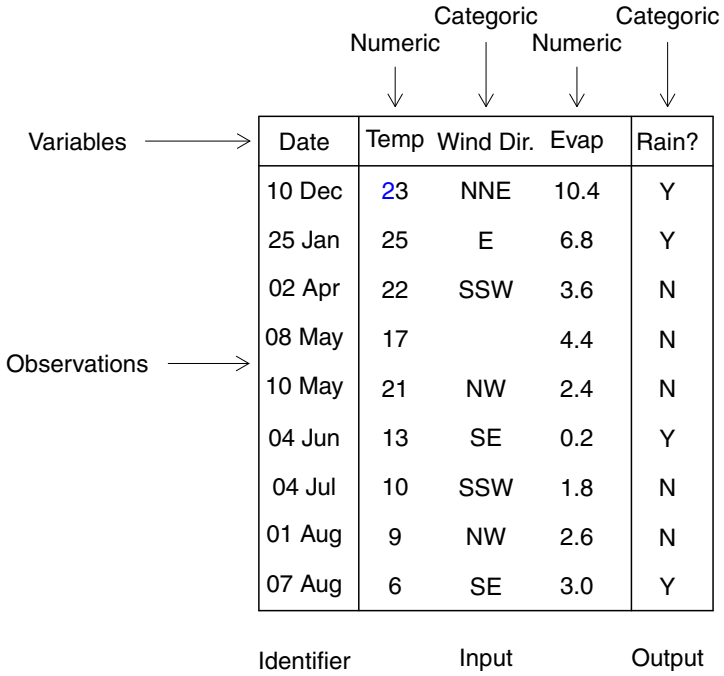

Figure 3.1: A simple dataset showing the nomenclature used. Each column is a *variable* and each row is an *observation*.

A **categoric variable**[1] is one that takes on a single value, for a particular observation, from a fixed set of possible values. Examples include `eye colour` (with possible values including `blue`, `green`, and `brown`), `age group` (with possible values `young`, `middle age`, and `old`), and `rain tomorrow` (with only two possible values, `Yes` and `No`). Categoric variables are always *discrete* (i.e., they can only take on specific values).

Categoric variables like `eye colour` are also known as *nominal variables*, *qualitative variables*, or *factors*. The possible values have no order to them. That is, `blue` is no less than or greater than `green`.

On the other hand, categoric variables like `age group` are also known as *ordinal variables*. The possible values have a natural order to them, so that `young` is in some sense less than `middle age`, which in turn is less than `old`.

[1] We use the terms *categoric* rather than *categorical* and *numeric* rather than *numerical*.

A categoric variable like `rain tomorrow`, having only two possible values, is also known as a *binary variable*.

A **numeric variable** has values that are integers or real numbers, such as a person's age or weight or their income or bank balance. Numeric variables are also known as *quantitative variables*. Numeric variables can be *discrete* (integers) or *continuous* (real).

A **dataset** (or, in particular, different randomly chosen subsets of a dataset) can have different roles. For building predictive models, for example, we often partition a dataset into three independent datasets: a **training dataset**, a **validation dataset**, and a **testing dataset**. The partitioning is done randomly to ensure each dataset is representative of the whole collection of observations. Typical splits might be 40/30/30 or 70/15/15. A validation dataset is also known as a design dataset (since it assists in the design of the model).

We build our model using the training dataset. The validation dataset is used to assess the model's performance. This will lead us to tune the model, perhaps through setting different model parameters. Once we are satisfied with the model, we assess its expected performance into the future using the testing dataset.

It is important to understand the significance of the testing dataset. This dataset must be a so-called *holdout* or *out-of-sample* dataset. It consists of randomly selected observations from the full dataset that are not used in any way in the building of the model. That is, it contains no observations in common with the training or validation datasets. This is important in relation to ensuring we obtain an unbiased estimate of the true performance of a model on new, previously unseen observations.

We can summarise our generic nomenclature, in one sentence, as:

> A **dataset** consists of **observations** recorded using **variables**, which consist of a mixture of **input variables** and **output variables**, either of which may be **categoric** or **numeric**.

Having introduced our generic nomenclature, we also need to relate the same concepts to how they are implemented in an actual system, like R. We do so, briefly, here.

R has the concept of a *data frame* to represent a dataset. A data frame is, technically, a *list* of variables. Each variable in the list represents a column of data—a variable stores a collection of data items that are all

of the same type. For example, this might be a collection of integers recording the ages of clients. Technically, R refers to what we call a variable within a dataset as a *vector*.

Each variable will record the same number of data items, and thus we can picture the dataset as a rectangular matrix, as we illustrated in Figure 3.1. A data frame is much like a table in a database or a page in a spreadsheet. It consists of rows, which we have called *observations*, and columns, which we have called *variables*.

3.2 Sourcing Data for Mining

To start a data mining project, we must first recognise and understand the problem to tackle. Whilst that might be quite obvious, there are subtleties we need to address, as discussed in Chapter 1. We also need data—again, somewhat obvious. As we suggested above, though, sourcing our data is usually not a trivial matter. We discuss the general data issue here before we delve into some technical aspects of data.

In an ideal world, the data we require for data mining will be nicely stored in a data warehouse or a database, or perhaps a spreadsheet. However, we live in a less than ideal world. Data is stored in many different forms and on many different systems, with many different meanings. Data is everywhere, for sure, but we need to find it, understand it, and bring it together.

Over the years, organisations have implemented well-managed data warehouse systems. They serve as the organisation-wide repository of data. It is true, though that, despite this, data will always spring up outside of the data warehouse, and will have none of the careful controls that surround the data warehouse with regard to data provenance and data quality. Eventually the organisation's data custodians will recapture the useful new "cottage industry" repositories into the data warehouse and the cycle of new "cottage industries" will begin once again. We will always face the challenge of finding data from many sources within an organisation.

An organisation's data is often not the only data we access within a data mining project. Data can be sourced from outside the organisation. This could include data publicly available, commercially collected, or legislatively obtained. The data will be in a variety of formats and of varying quality. An early task for us is to assess whether the data will

be useful for the business problem and how we will bring the new data together with our other data. We delve further into understanding the data in Chapter 5. We consider data quality now.

3.3 Data Quality

No real-world data is perfectly collected. Despite the amount of effort organisations put into ensuring the quality of the data they collect, errors will always occur. We need to understand issues relating to, for example, consistency, accuracy, completeness, interpretability, accessibility, and timeliness.

It is important that we recognise and understand that our data will be of varying quality. We need to treat (i.e., transform) our data appropriately and be aware of the limitations (uncertainties) of any analysis we perform on it. Chapter 7 covers many aspects of data quality and how we can work towards improving the quality of our available data. Below we summarise some of the issues.

In the past, much data was entered by data entry staff working from forms or directly in conversation with clients. Different data entry staff often interpret different data fields (variables) differently. Such inconsistencies might include using different formats for dates or recording expenses in different currencies in the same field, with no information to identify the currency.

Often in the collection of data some data is more carefully (or accurately) collected than other data. For bank transactions, for example, the dollar amounts must be very accurate. The precise spelling of a person's name or address might not need to be quite so accurate. Where the data must be accurate, extra resources will be made available to ensure data quality. Where accuracy is less critical, resources might be saved. In analysing data, it is important to understand these aspects of accuracy.

Related to accuracy is the issue of completeness. Some less important data might only be optionally collected, and thus we end up with much missing data in our datasets. Alternatively, some data might be hard to collect, and so for some observations it will be missing. When analysing data, we need to understand the reasons for missing data and deal with the data appropriately. We cover this in detail in Chapter 7.

Another major issue faced by the data miner is the interpretation of the data. Having a thorough understanding of the meaning of the data is

critical. Knowing that height is measured in feet or in meters will make a difference to the analysis. We might find that some data was entered as feet and other data as meters (the consistency problem). We might have dollar amounts over many years, and our analysis might need to interpret the amounts in terms of their relative present-day value. Codes are also often used, and we need to understand what each code means and how different codes relate to each other. As the data ages, the meaning of the different variables will often change or be altogether lost. We need to understand and deal with this.

The accessibility of the right data for analysis will often also be an issue. A typical process in data collection involves checking for obvious data errors in the data supplied and correcting those errors. In collecting tax return data from taxpayers, for example, basic checks will be performed to ensure the data appears correct (e.g., checking for mistakes that enter data as 3450 to mean $3450, whereas it was meant to be $34.50). Sometimes the checks might involve discussing the data with its supplier and modifying it appropriately. Often it is this "cleaner" data that is stored on the system rather than the original data supplied. The original data is often archived, but often it is such data that we actually need for the analysis—we want to analyse the data as supplied originally. Accessing archived data is often problematic.

Accessing the most recent data can sometimes be a challenge. In an online data processing environment, where the key measure of performance is the turnaround time of the transaction, providing other systems with access to the data in a timely manner can be a problem. In many environments, the data can only be accessed after a sometimes complex extract/transform/load (ETL) process. This can mean that the data may only be available after a day or so, which may present challenges for its timely analysis. Often, business processes need to be changed so that more timely access is possible.

3.4 Data Matching

In collecting data from multiple sources, we end up with a major problem in that we need to match observations from one dataset with those from another dataset. That is, we need to identify the same entities (e.g., people or companies) from different data sources. These different sources could be, for example, patient medical data from a doctor and

from a hospital. The doctor's data might contain information about the patients' general visits, basic test results, diagnoses, and prescriptions. The doctor might have a unique number to identify his or her own patients, as well as their names, dates of birth, and addresses. A hospital will also record data about patients that are admitted, including their reason for admission, treatment plan, and medications. The hospital will probably have its own unique number to identify each patient, as well as the patient's name, date and place of birth, and address.

The process of *data matching* might be as simple as joining two datasets together based on shared identifiers that are used in each of the two databases. If the doctor and the hospital share the same unique numbers to identify the patients, then the data matching process is simplified.

However, the data matching task is usually much more complex. Data matching often involves, for example, matching of names, addresses, and dates and places of birth, all of which will have inaccuracies and alternatives for the same thing. The data entered at a doctor's consulting rooms will in general be entered by a different receptionist on a different day from the data entered on admission at a hospital where surgery might be performed.

It is not uncommon to find, even within a single database, one person's name recorded differently, let alone when dealing with data from very different sources. One data source might identify "John L. Smith," and another might identify the person as "J.L. Smith," and a third might have an error or two but identify the person as "Jon Leslie Smyth."

The task of data matching is to bring different data sources together in a reliable and supportable manner so that we have the right data about the right person. An idea that can improve data matching quality is that of a trusted data matching bureau. Many data matching bureaus within organisations almost start each new data matching effort from scratch. However, over time there is the opportunity to build up a data matching database that records relevant information about all previous data matches.

Under this scenario, each time a new data matching effort is undertaken, the identities within this database, and their associated information, are used to improve the new data matching. Importantly, the results of the new data matching feed back into the data matching database to improve the quality of the matched entities and thus even improve previously matched data.

Data matching is quite an extensive topic in itself and worth a separate book. A number of commercially available tools assist with the basic task. The open source Febrl[2] system also provides data matching capabilities. They all aim to identify the same entity in all of the data sources.

3.5 Data Warehousing

The process of bringing data together into one unified and carefully managed repository is referred to as *data warehousing*—the analogy being with a large building used for the storage of goods. What we store in our warehouse is data. Data warehouses were topical in the 1990s and primarily vendor driven, servicing a real opportunity to get on top of managing data. Inmon (1996) provides a detailed introduction.

We can view a data warehouse as a large database management system. It is designed to integrate data from many different sources and to support analysis for different objectives. In any organisation, the data warehouse can be the foundation for business intelligence, providing a single, integrated source of data for the whole organisation.

Typically, a data warehouse will contain data collected together from multiple sources but focussed around the function of an organisation. The data sources will often be *operational systems* (such as transaction processing systems) that run the day-to-day functions of the organisation. In banking, for example, the transaction processing systems include ATMs and EFTPOS machines, which are today most pervasive. Transaction processing systems collect data that gets uploaded to the data warehouse on a regular basis (e.g., daily, but perhaps even more frequently).

Well-organised data warehouses, at least from the point of view of data mining, will also be nonvolatile. The data stored in our data warehouses will capture data regularly, and older data is not removed. Even when an update to data is to correct existing data items, such data must be maintained, creating a massive repository of historic data that can be used to carefully track changes over time.

Consider the case of tax returns held by our various revenue authorities. Many corrections are made to individual tax returns over time. When a tax return is filed, a number of checks for accuracy may result in

[2] http://sourceforge.net/projects/febrl/.

simple changes (e.g., correcting a misspelled address). Further changes might be made at a later time as a taxpayer corrects data originally supplied. Changes might also be the result of audits leading to corrections made by the revenue authority, or a taxpayer may notify the authority of a change in address.

Keeping the history of data changes is essential for data mining. It may be quite significant, from a fraud point of view, that a number of clients in a short period of time change their details in a common way. Similarly, it might be significant, from the point of view of understanding client behaviour, that a client has had ten different addresses in the past 12 months. It might be of interest that a taxpayer always files his or her tax return on time each year, and then makes the same two adjustments subsequently, each year. All of this historic data is important in building a picture of the entities we are interested in. Whilst the operational systems may only store data for one or two months before it is archived, having this data accessible for many years within a data warehouse for data mining is important.

In building a data warehouse, much effort goes into how the data warehouse is structured. It must be designed to facilitate the queries that operate on a large proportion of data. A careful design that exposes all of the data to those who require it will aid in the data mining process.

Data warehouses quickly become unwieldly as more data is collected. This often leads to the development of specific *data marts*, which can be thought of as creating a tuned subset of the data warehouse for specific purposes. An organisation, for example, may have a finance data mart, a marketing data mart, and a sales data mart. Each data mart will draw its information from various other data collected in the warehouse. Different data sources within the warehouse will be shared by different data marts and present the data in different ways.

A crucial aspect of a data warehouse (and any data storage, in fact) is the maintenance of information about the data—so-called **metadata**. Metadata helps make the data understandable and thereby useful. We might talk about two types of metadata: **technical metadata** and **business metadata**.

Technical metadata captures data about the operational systems from which the data was obtained, how it was extracted from the source systems, how it was transformed, how it was loaded into the warehouse, where it is stored within the warehouse, and its structure as stored in the warehouse.

The actual process of extracting, transforming, and then loading data is often referred to as ETL (extract, transform, load). Many vendors provide ETL tools, and there is also extensive capability for automating ETL using open source software, including R.

The business metadata, on the other hand, provides the information that is useful in understanding the data. It will include descriptions of the variables contained in the data and measures of their data quality. It can also include who "owns" the data, who has access to it, the cost of accessing it, when it was last updated, how frequently it is updated, and how it is used operationally.

Before data mining became a widely adopted technology, the data warehouse supported analyses through business intelligence (BI) technology. The simplest analyses build reports that aggregate the data within a warehouse in many different ways. Through this technology, an organisation is able to ensure its executives are aware of its activities. On-line, analytic processing (OLAP) within the BI technology supports user-driven and multidimensional analyses of the data contained within the warehouse. Extending the concept of a human-driven and generally manual analysis of data, as in business intelligence, data mining provides a data-driven approach to the analysis of the data.

Ideally, the data warehouse is the primary data source for data mining. Integrating data from multiple sources, the data warehouse should contain an extensive resource that captures all of the activity of an organisation. Also, ideally, the data will be consistent, of high quality, and documented with very good metadata. If all that is true, the data mining will be quite straightforward. Rarely is this true. Nonetheless, mining data from the data warehouse can significantly reduce the time for preparing it and sharing the data across many data mining and reporting projects.

Data warehouses will often be accessed through the common *structured query language* (SQL). Our data will usually be spread across multiple locations within the warehouse, and SQL queries will be used to bring them together. Some basic familiarity with SQL will be useful as we extract our data. Otherwise we will need to ensure we have ready access to the skills of a data analyst to extract the data for us.

3.6 Interacting with Data Using R

Once we have scouted for data, matched common entities, and brought the data together, we need to structure the data into a form suitable for data mining. More specifically, we need to structure the data to suit the data mining tool we are intending to use. In our case, this involves putting the data into a form that allows it to be easily loaded into R, using Rattle, where we will then explore, test, and transform it in various ways in preparation for mining.

Once we have loaded a dataset into Rattle, through one of the mechanisms we introduce in Chapter 4 (or directly through R itself), we may want to modify the data, clean it, and transform it into the structures we require. We may already be familiar with a variety of tools for dealing with data (like SQL or a spreadsheet). These tools may be quite adequate for the manipulations we need to undertake. We can easily prepare the data with them and then load it into Rattle when ready. But R itself is also a very powerful data manipulation language.

Much of R's capabilities for data management are covered in other books, including those of Spector (2008), Muenchen (2008), and Chambers (2008). Rattle provides access to some data cleaning operations under the Transform tab, as covered in Chapter 7. We provide here elementary instruction in using R itself for a limited set of manipulations that are typical in preparing data for data mining. We do not necessarily cover the details nor provide the systematic coverage of R available through other means.

One of the most basic operations is accessing the data within a dataset. We *index* a dataset using the notation of square brackets, and within the square brackets we identify the index of the observations and the variables we are interested in, separating them with a comma. We briefly saw this previously in Section 2.9.

Using the same *weather* dataset as in Chapter 2 (available from **rattle**, which we can load into R's `library()`), we can access observations 100 to 105 and variables 5 to 6 by *indexing* the dataset. If either index (observations or variables) is left empty, then the result will be all observations or all variables, respectively, rather than just a subset of them. Using `dim()` to report on the resulting size (dimensions) of the dataset, we can see the effect of the indexing:

```
> library(rattle)
> weather[100:105, 5:6]

    Rainfall Evaporation
100     16.2         5.4
101      0.0         4.0
102      0.0         5.8
103      0.0         5.0
104      4.4         6.6
105     11.0         3.2

> dim(weather)

[1] 366  24

> dim(weather[100:105, 5:6])

[1] 6 2

> dim(weather[100:105,])

[1]  6 24

> dim(weather[,5:6])

[1] 366   2

> dim(weather[5:6])

[1] 366   2

> dim(weather[,])

[1] 366  24
```

Note that the notation `100:105` is actually shorthand for a call to `seq()`, which generates a list of numbers. Another way to generate a list of numbers is to use `c()` (for combine) and list each of the numbers explicitly. These expressions can replace the "`100:105`" in the example above to have the same effect. We can see this in the following code block.

```
> 100:105
[1] 100 101 102 103 104 105
> seq(100, 105)
[1] 100 101 102 103 104 105
> c(100, 101, 102, 103, 104, 105)
[1] 100 101 102 103 104 105
```

Variables can be referred to by their position number, as above, or by the variable name. In the following example, we extract six observations of just two variables. Note the use of the `vars` object to list the variables of interest and then from that index the dataset.

```
> vars <- c("Evaporation", "Sunshine")
> weather[100:105, vars]
    Evaporation Sunshine
100         5.4      5.6
101         4.0      8.9
102         5.8      9.6
103         5.0     10.7
104         6.6      5.9
105         3.2      0.4
```

We can list the variable names contained within a dataset using `names()`:

```
> head(names(weather))
[1] "Date"        "Location"    "MinTemp"
[4] "MaxTemp"     "Rainfall"    "Evaporation"
```

In this example we list only the first six names, making use of `head()`. This example also illustrates the "functional" nature of R. Notice how we directly feed the output of one function (`names()`) into another function (`head()`).

We could also use indexing to achieve the same result:

```
> names(weather)[1:6]
[1] "Date"        "Location"    "MinTemp"
[4] "MaxTemp"     "Rainfall"    "Evaporation"
```

When we index a dataset with single brackets, as in `weather[2]` or `weather[4:7]`, we retrieve a "subset" of the dataset—specifically, we retrieve a subset of the variables. The result itself is another dataset, even if it contains just a single variable. Compare this with `weather[[2]]`, which returns the actual values of the variable. The differences may appear subtle, but as we gain experience with R, they become important. We do not dwell on this here, though.

```
> head(weather[2])

  Location
1 Canberra
2 Canberra
3 Canberra
4 Canberra
5 Canberra
6 Canberra

> head(weather[[2]])

[1] Canberra Canberra Canberra Canberra Canberra Canberra
46 Levels: Adelaide Albany Albury ... Woomera
```

We can use the `$` notation to access specific variables within a dataset. The expression `weather$MinTemp` refers to the `MinTemp` variable of the *weather* dataset:

```
> head(weather$MinTemp)

[1]  8.0 14.0 13.7 13.3  7.6  6.2
```

3.7 Documenting the Data

The *weather* dataset, for example, though very small in the number of observations, is somewhat typical of data mining. We have obtained the dataset from a known source and have processed it to build a dataset ready for our data mining. To do this, we've had to research the meaning of the variables and read about any idiosyncrasies associated with the collection of the data. Such information needs to be captured in a data mining report. The report should record where our data has come from, our understanding of its integrity, and the meaning of the variables. This

information will come from a variety of sources and usually from multiple domain experts. We need to understand and document the provenance of the data: how it was collected, who collected it, and how they understood what they were collecting.

The following summary will be useful. It is obtained from processing the output from the `str()`. That output, which is normally only displayed in the console, is first captured into a variable using `capture.output()`:

```
> sw <- capture.output(str(weather, vec.len=1))
> cat(sw[1])

'data.frame':        366 obs. of  24 variables:
```

The output is then processed to add a variable number and appropriately fit the page. The processing first uses `sprintf()` to generate a list of variable numbers, each number stored as a string of width 2 ("`%2d`"):

```
> swa <- sprintf("%2d", 1:length(sw[-1]))
```

Each number is then `pasted` to each line of the output, collapsing the separate lines to form one long string with a new line ("`\n`") separating each line:

```
> swa <- paste(swa, sw[-1], sep="", collapse="\n")
```

The `gsub()` function is then used to truncate lines that are too long by substituting a particular pattern of dots and digits with just "`..`".

```
> swa <- gsub("\\.\\.: [0-9]+ [0-9]+ \\.\\.\\.", "..", swa)
```

The final substitution removes some unnecessary characters, again to save on space. That is a little complex at this stage but illustrates the power of R for string processing (as well as statistics).

```
> swa <- gsub("( \\$|:|)", "", swa)
```

We use `cat()` to then display the results of this processing.

```
> cat(swa)

 1 Date          Date, format "2007-11-01" ...
 2 Location      Factor w/ 46 levels "Adelaide","Albany",..
 3 MinTemp       num  8 14 ...
 4 MaxTemp       num  24.3 26.9 ...
 5 Rainfall      num  0 3.6 ...
 6 Evaporation   num  3.4 4.4 ...
 7 Sunshine      num  6.3 9.7 ...
 8 WindGustDir   Ord.factor w/ 16 levels "N"<"NNE"<"NE"<..
 9 WindGustSpeed num  30 39 ...
10 WindDir9am    Ord.factor w/ 16 levels "N"<"NNE"<"NE"<..
11 WindDir3pm    Ord.factor w/ 16 levels "N"<"NNE"<"NE"<..
12 WindSpeed9am  num  6 4 ...
13 WindSpeed3pm  num  20 17 ...
14 Humidity9am   int  68 80 ...
15 Humidity3pm   int  29 36 ...
16 Pressure9am   num  1020 ...
17 Pressure3pm   num  1015 ...
18 Cloud9am      int  7 5 ...
19 Cloud3pm      int  7 3 ...
20 Temp9am       num  14.4 17.5 ...
21 Temp3pm       num  23.6 25.7 ...
22 RainToday     Factor w/ 2 levels "No","Yes" 1 2 ...
23 RISK_MM       num  3.6 3.6 ...
24 RainTomorrow  Factor w/ 2 levels "No","Yes" 2 2 ...
```

3.8 Summary

In this chapter, we have introduced the concepts of data and dataset. We have described how we obtain data and issues related to the data we use for data mining. We have also introduced some basic data manipulation using R. We will revisit the *weather*, *weatherAUS*, and *audit* datasets throughout the book. Appendix B describes in detail how these datasets are obtained and processed into a form for use in data mining. The amount of detail there and the R code provided may be useful in learning more about manipulating data in R.

3.9 Command Summary

This chapter has referenced the following R packages, commands, functions, and datasets:

`<-`	function	Assign a value into a named reference.
`$`	function	Extract a variable from a dataset.
audit	dataset	Sample dataset from **rattle**.
`c()`	function	Combine items to form a collection.
`cat()`	function	Display the arguments to the screen.
`dim()`	function	Report the rows and columns of a dataset.
`gsub()`	function	Globally substitute one string for another.
`head()`	function	Show top observations of a dataset.
`library()`	command	Load a package into the R library.
`names()`	function	Show variables contained in a dataset.
`paste()`	function	Combine strings into one string.
rattle	package	Provides the *weather* and *audit* datasets.
`seq()`	function	Generate a sequence of numbers.
`sprintf`	function	Format a string with substitution.
`str()`	function	Show the structure of an object.
weather	dataset	Sample dataset from **rattle**.
weatherAUS	dataset	A larger dataset from **rattle**.