Data Science with R
Data Preparation—A Template

Graham.Williams@togaware.com

14th February 2014

Visit http://onepager.togaware.com/ for more OnePageR's.

In this module we introduce a generic template for preparing data for building models using R. The intention is that this document and the included scripts serve as a template for loading a dataset, observing the dataset, and transforming the dataset in preparation for building a model. This module leads into the Models OnePageR module which provides a template for building models.

All of the steps are then collected together into a single sequence in the final two sections so that we can easily copy and paste the whole code block as the starting point for any project.

The **weather** dataset from rattle (Williams, 2014) is used in this module.

The required packages for this module include:

```
library(rattle)       # The weather dataset and normVarNames().
library(randomForest) # Use na.roughfix() to deal with missing data.
library(ggplot2)      # Plots.
```

As we work through this module, new R commands will be introduced. Be sure to review the command's documentation and understand what the command does. You can ask for help using the ? command as in:

```
?read.csv
```

We can obtain documentation on a particular package using the *help=* option of `library()`:

```
library(help=rattle)
```

This present module is intended to be hands on. To learn effectively, you are encouraged to have R running (e.g., RStudio) and to run all the commands as they appear here. Check that you get the same output, and you understand the output. Try some variations. Explore.

1 Step 1: Load—Dataset

We use the **weather** dataset from rattle (Williams, 2014) to illustrate our data preparation. Often though we will be loading the dataset from a CSV file and so we illustrate that step first. Note the path to the file—in this case we load it from the Internet.

```
dspath <- "http://rattle.togaware.com/weather.csv"
```

If we are not connected to the Internet, then we can read the data from the rattle package.

```
dspath <- system.file("csv", "weather.csv", package="rattle")
```

Then it is a simple matter of reading it in.

```
weather <- read.csv(dspath)
```

We will store the dataset as the generic variable *ds* (short for dataset). This will make the following steps more generic, and often we can just load a different dataset into *ds* and the rest of the template can be used without change.

```
dsname <- "weather"
ds     <- get(dsname)
dim(ds)

## [1] 366  24

names(ds)

##  [1] "Date"          "Location"      "MinTemp"       "MaxTemp"
##  [5] "Rainfall"      "Evaporation"   "Sunshine"      "WindGustDir"
##  [9] "WindGustSpeed" "WindDir9am"    "WindDir3pm"    "WindSpeed9am"
## [13] "WindSpeed3pm"  "Humidity9am"   "Humidity3pm"   "Pressure9am"
## [17] "Pressure3pm"   "Cloud9am"      "Cloud3pm"      "Temp9am"
## [21] "Temp3pm"       "RainToday"     "RISK_MM"       "RainTomorrow"
```

We are being a little tricky here in recording the dataset name as *dsname* and then using `get()` to load the data into the variable *ds*. We could simply assign the data to *ds*:

```
ds <- weather
```

However the use of the generic variables allows much of the following code to be run on different datasets with little, if any, change. Thus the following scripts can truely act as templates for building models.

This has its advantages, though a disadvantage is that we may be building several models and accidentally overwrite previously built models stored in a generic variable name (like *model* and even *ds*) that may have taken some time to build. This requires some care.

There are also R packages that support template type programming with data, but to keep things simple, we stay with a simple approach here.

2 Step 2: Review—Observations

Once we have loaded the dataset, the next step is to understand the shape of the dataset. We review the data using `head()` and `tail()` to get our first feel for the observations contained in the dataset. We also have a look at some random observations from the dataset to provide further insight.

```
head(ds)

##         Date Location MinTemp MaxTemp Rainfall Evaporation Sunshine
## 1 2007-11-01 Canberra     8.0    24.3      0.0         3.4      6.3
## 2 2007-11-02 Canberra    14.0    26.9      3.6         4.4      9.7
## 3 2007-11-03 Canberra    13.7    23.4      3.6         5.8      3.3
## 4 2007-11-04 Canberra    13.3    15.5     39.8         7.2      9.1
## 5 2007-11-05 Canberra     7.6    16.1      2.8         5.6     10.6
## 6 2007-11-06 Canberra     6.2    16.9      0.0         5.8      8.2
##   WindGustDir WindGustSpeed WindDir9am WindDir3pm WindSpeed9am
## 1          NW            30         SW         NW            6
## 2         ENE            39          E          W            4
....
```

```
tail(ds)

##           Date Location MinTemp MaxTemp Rainfall Evaporation Sunshine
## 361 2008-10-26 Canberra     7.9    26.1        0         6.8      3.5
## 362 2008-10-27 Canberra     9.0    30.7        0         7.6     12.1
## 363 2008-10-28 Canberra     7.1    28.4        0        11.6     12.7
## 364 2008-10-29 Canberra    12.5    19.9        0         8.4      5.3
## 365 2008-10-30 Canberra    12.5    26.9        0         5.0      7.1
## 366 2008-10-31 Canberra    12.3    30.2        0         6.0     12.6
##     WindGustDir WindGustSpeed WindDir9am WindDir3pm WindSpeed9am
## 361         NNW            43       <NA>        WNW            0
## 362         NNW            76        SSE         NW            7
....
```

```
ds[sample(nrow(ds), 6),]

##           Date Location MinTemp MaxTemp Rainfall Evaporation Sunshine
## 358 2008-10-23 Canberra     3.2    18.0      0.0         7.4     12.2
## 12  2007-11-12 Canberra     8.5    27.3      0.2         7.2     12.5
## 30  2007-11-30 Canberra    13.6    24.1      0.4         2.6      0.5
## 142 2008-03-21 Canberra    13.0    14.8      0.0         8.2      0.0
## 67  2008-01-06 Canberra    14.3    34.1      0.0         6.6     10.5
## 127 2008-03-06 Canberra    12.9    31.8      0.0         6.0     11.3
##     WindGustDir WindGustSpeed WindDir9am WindDir3pm WindSpeed9am
## 358         SSE            48        SSE          S           26
## 12            E            41          E         NW            2
....
```

3 Step 2: Review—Structure

Next we use `str()` to report on the structure of the dataset. Once again we get an overview of what the data looks like, and also now, how it is stored.

```
str(ds)

## 'data.frame': 366 obs. of  24 variables:
##  $ Date         : Factor w/ 366 levels "2007-11-01","2007-11-02",..: 1 2 3...
##  $ Location     : Factor w/ 1 level "Canberra": 1 1 1 1 1 1 1 1 1 1 ...
##  $ MinTemp      : num  8 14 13.7 13.3 7.6 6.2 6.1 8.3 8.8 8.4 ...
##  $ MaxTemp      : num  24.3 26.9 23.4 15.5 16.1 16.9 18.2 17 19.5 22.8 ...
##  $ Rainfall     : num  0 3.6 3.6 39.8 2.8 0 0.2 0 0 16.2 ...
##  $ Evaporation  : num  3.4 4.4 5.8 7.2 5.6 5.8 4.2 5.6 4 5.4 ...
##  $ Sunshine     : num  6.3 9.7 3.3 9.1 10.6 8.2 8.4 4.6 4.1 7.7 ...
##  $ WindGustDir  : Factor w/ 16 levels "E","ENE","ESE",..: 8 2 8 8 11 10 10...
##  $ WindGustSpeed: int  30 39 85 54 50 44 43 41 48 31 ...
##  $ WindDir9am   : Factor w/ 16 levels "E","ENE","ESE",..: 13 1 4 15 11 10 ...
##  $ WindDir3pm   : Factor w/ 16 levels "E","ENE","ESE",..: 8 14 6 14 3 1 3 ...
##  $ WindSpeed9am : int  6 4 6 30 20 20 19 11 19 7 ...
##  $ WindSpeed3pm : int  20 17 6 24 28 24 26 24 17 6 ...
##  $ Humidity9am  : int  68 80 82 62 68 70 63 65 70 82 ...
##  $ Humidity3pm  : int  29 36 69 56 49 57 47 57 48 32 ...
##  $ Pressure9am  : num  1020 1012 1010 1006 1018 ...
##  $ Pressure3pm  : num  1015 1008 1007 1007 1018 ...
##  $ Cloud9am     : int  7 5 8 2 7 7 4 6 7 7 ...
##  $ Cloud3pm     : int  7 3 7 7 7 5 6 7 7 1 ...
##  $ Temp9am      : num  14.4 17.5 15.4 13.5 11.1 10.9 12.4 12.1 14.1 13.3 ...
##  $ Temp3pm      : num  23.6 25.7 20.2 14.1 15.4 14.8 17.3 15.5 18.9 21.7 ...
##  $ RainToday    : Factor w/ 2 levels "No","Yes": 1 2 2 2 2 1 1 1 1 2 ...
##  $ RISK_MM      : num  3.6 3.6 39.8 2.8 0 0.2 0 0 16.2 0 ...
##  $ RainTomorrow : Factor w/ 2 levels "No","Yes": 2 2 2 2 1 1 1 1 2 1 ...
```


4 Step 2: Review—Summary

We use `summary()` to preview the distributions.

```
summary(ds)

##          Date          Location      MinTemp          MaxTemp
##  2007-11-01:  1   Canberra:366   Min.   :-5.30   Min.   : 7.6
##  2007-11-02:  1                  1st Qu.: 2.30   1st Qu.:15.0
##  2007-11-03:  1                  Median : 7.45   Median :19.6
##  2007-11-04:  1                  Mean   : 7.27   Mean   :20.6
##  2007-11-05:  1                  3rd Qu.:12.50   3rd Qu.:25.5
##  2007-11-06:  1                  Max.   :20.90   Max.   :35.8
##  (Other)   :360
##     Rainfall       Evaporation       Sunshine       WindGustDir
##  Min.   : 0.00   Min.   : 0.20   Min.   : 0.00   NW     : 73
##  1st Qu.: 0.00   1st Qu.: 2.20   1st Qu.: 5.95   NNW    : 44
##  Median : 0.00   Median : 4.20   Median : 8.60   E      : 37
##  Mean   : 1.43   Mean   : 4.52   Mean   : 7.91   WNW    : 35
##  3rd Qu.: 0.20   3rd Qu.: 6.40   3rd Qu.:10.50   ENE    : 30
##  Max.   :39.80   Max.   :13.80   Max.   :13.60   (Other):144
##                                  NA's   :3       NA's   :  3
##  WindGustSpeed    WindDir9am    WindDir3pm    WindSpeed9am    WindSpeed3pm
##  Min.   :13.0   SE     : 47   NW     : 61   Min.   : 0.00   Min.   : 0
##  1st Qu.:31.0   SSE    : 40   WNW    : 61   1st Qu.: 6.00   1st Qu.:11
##  Median :39.0   NNW    : 36   NNW    : 47   Median : 7.00   Median :17
##  Mean   :39.8   N      : 31   N      : 30   Mean   : 9.65   Mean   :18
##  3rd Qu.:46.0   NW     : 30   ESE    : 27   3rd Qu.:13.00   3rd Qu.:24
##  Max.   :98.0   (Other):151   (Other):139   Max.   :41.00   Max.   :52
##  NA's   :2      NA's   : 31   NA's   :  1   NA's   :7
##   Humidity9am  Humidity3pm     Pressure9am     Pressure3pm       Cloud9am
##  Min.   :36   Min.   :13.0   Min.   : 996   Min.   : 997   Min.   :0.00
##  1st Qu.:64   1st Qu.:32.2   1st Qu.:1015   1st Qu.:1013   1st Qu.:1.00
##  Median :72   Median :43.0   Median :1020   Median :1017   Median :3.50
##  Mean   :72   Mean   :44.5   Mean   :1020   Mean   :1017   Mean   :3.89
##  3rd Qu.:81   3rd Qu.:55.0   3rd Qu.:1024   3rd Qu.:1022   3rd Qu.:7.00
##  Max.   :99   Max.   :96.0   Max.   :1036   Max.   :1033   Max.   :8.00
##
##     Cloud3pm       Temp9am          Temp3pm      RainToday     RISK_MM
##  Min.   :0.00   Min.   : 0.10   Min.   : 5.1   No :300    Min.   : 0.00
##  1st Qu.:1.00   1st Qu.: 7.62   1st Qu.:14.2   Yes: 66    1st Qu.: 0.00
##  Median :4.00   Median :12.55   Median :18.6              Median : 0.00
##  Mean   :4.03   Mean   :12.36   Mean   :19.2              Mean   : 1.43
##  3rd Qu.:7.00   3rd Qu.:17.00   3rd Qu.:24.0              3rd Qu.: 0.20
##  Max.   :8.00   Max.   :24.70   Max.   :34.5              Max.   :39.80
##
##  RainTomorrow
##  No :300
##  Yes: 66
....
```


5 Step 2: Review—Meta Data Cleansing

We demonstrate some meta-data changes here.

Normalise Variable Names Sometimes it is convenient to map all variable names to lowercase. R is case sensitive, so doing this does change the variable names. This can be useful when different upper/lower case conventions are intermixed in names like `Incm_tax_PyBl` and remembering how to capitalise when interactively exploring the data with 1,000 such variables is an annoyance. We often see such variable names arising when we import data from databases which are often case insensitive.

Here we use `normVarName()` from `rattle` (Williams, 2014), which attempts to do a reasonable job of converting variables from a dataset into a standard form.

```
names(ds)

##  [1] "Date"          "Location"      "MinTemp"       "MaxTemp"
##  [5] "Rainfall"      "Evaporation"   "Sunshine"      "WindGustDir"
##  [9] "WindGustSpeed" "WindDir9am"    "WindDir3pm"    "WindSpeed9am"
## [13] "WindSpeed3pm"  "Humidity9am"   "Humidity3pm"   "Pressure9am"
....

names(ds) <- normVarNames(names(ds))
names(ds)

##  [1] "date"            "location"          "min_temp"
##  [4] "max_temp"        "rainfall"          "evaporation"
##  [7] "sunshine"        "wind_gust_dir"     "wind_gust_speed"
## [10] "wind_dir_9am"    "wind_dir_3pm"      "wind_speed_9am"
....
```

6 Step 2: Review—Data Formats

We may want to correct the format of some of the variables in our dataset. We might first check the data type of each variable.

```
sapply(ds, class)

##            date        location        min_temp        max_temp
##        "factor"        "factor"       "numeric"       "numeric"
##        rainfall     evaporation        sunshine   wind_gust_dir
##       "numeric"       "numeric"       "numeric"        "factor"
....
```

We note that the `date` variable is a factor rather than a date. Thus we may like to convert it into a date using lubridate (?):

```
library(lubridate)
head(ds$date)

## [1] 2007-11-01 2007-11-02 2007-11-03 2007-11-04 2007-11-05 2007-11-06
## 366 Levels: 2007-11-01 2007-11-02 2007-11-03 2007-11-04 ... 2008-10-31

ds$date <- ymd(as.character(ds$date))
head(ds$date)

## [1] "2007-11-01 UTC" "2007-11-02 UTC" "2007-11-03 UTC" "2007-11-04 UTC"
## [5] "2007-11-05 UTC" "2007-11-06 UTC"
```

```
sapply(ds, class)

## $date
## [1] "POSIXct" "POSIXt"
##
## $location
....
```

7 Step 2: Review—Variable Roles

We are now in a position to identify the roles played by the variables within the dataset. From our observations so far we note that the first variable (*Date*) is not relevant, as is, to the modelling (we could turn it into a seasonal variable which might be useful). Also we remove the second variable (*Location*) as in the data here it is a constant. We also identify the risk variable, if it is provided—it is a measure of the amount of risk or the importance of an observation with respect to the target variable. The risk is an output variable, and thus should not be used as an input to the modelling.

```
(vars  <- names(ds))

##  [1] "date"            "location"        "min_temp"
##  [4] "max_temp"        "rainfall"        "evaporation"
##  [7] "sunshine"        "wind_gust_dir"   "wind_gust_speed"
## [10] "wind_dir_9am"    "wind_dir_3pm"    "wind_speed_9am"
## [13] "wind_speed_3pm"  "humidity_9am"    "humidity_3pm"
## [16] "pressure_9am"    "pressure_3pm"    "cloud_9am"
## [19] "cloud_3pm"       "temp_9am"        "temp_3pm"
## [22] "rain_today"      "risk_mm"         "rain_tomorrow"

target <- "rain_tomorrow"
risk   <- "risk_mm"
id     <- c("date", "location")
```

8 Step 3: Clean—Ignore IDs, Outputs, Missing

We will want to ignore some variables that are irrelevant or inappropriate for modelling.

IDs and Outputs We start with the identifiers and the risk variable (which is an output variable). These should play no role in the modelling. Always watch out for including output variables as inputs to the modelling. This is one trap I regularly see from beginners.

```
ignore <- union(id, if (exists("risk")) risk)
```

We might also identify any variable that has a unique value for every observation. These are sometimes identifiers as well and if so are candidates for ignoring.

```
(ids   <- which(sapply(ds, function(x) length(unique(x))) == nrow(ds)))

## date
##    1

ignore <- union(ignore, ids)
```

All Missing We then include in the variables to remove all any variables where all of the values are missing. There are none like this in the weather dataset, but in general across 1,000 variables, there may be some. We first count the number of missing values for each variable, and then list the names of those variables with only missing values.

```
mvc <- sapply(ds[vars], function(x) sum(is.na(x)))
mvn <- names(which(mvc == nrow(ds)))
ignore <- union(ignore, mvn)
```

Many Missing Perhaps we also want to ignore variables with more than 70% of the values missing.

```
mvn <- names(which(mvc >= 0.7*nrow(ds)))
ignore <- union(ignore, mvn)
```

9 Step 3: Clean—Ignore MultiLevel, Constants

Too Many Levels We might also want to ignore variables with too many levels. Another approach is to group the levels into a smaller number of levels, but here we simply ignore them

```
factors <- which(sapply(ds[vars], is.factor))
lvls    <- sapply(factors, function(x) length(levels(ds[[x]])))
(many   <- names(which(lvls > 20)))

## character(0)

ignore  <- union(ignore, many)
```

Constants Ignore variables with constant values.

```
(constants <- names(which(sapply(ds[vars], function(x) all(x == x[1L])))))

## [1] "location"

ignore     <- union(ignore, constants)
```

10 Step 3: Clean—Remove the Variables

Once we have identified the variables to ignore, we remove them from our list of variables to use.

```
length(vars)

## [1] 24

vars <- setdiff(vars, ignore)
length(vars)

## [1] 21
```

11 Step 3: Clean—Remove Missing Target

Sometimes there may be further operations to perform on the dataset prior to modelling. This can include dealing with missing values, converting variables to their correct type, etc. Here, we remove observations with a missing target.

```
dim(ds)

## [1] 366  24

sum(is.na(ds[target]))

## [1] 0

ds <- ds[!is.na(ds[target]),]
sum(is.na(ds[target]))

## [1] 0

dim(ds)

## [1] 366  24
```

12 Step 3: Clean—Deal with Missing Values

Missing values for the variables are an issue for some but not all model builders. For example, `randomForest()` has not been coded to handle missing values whilst `rpart()` has a particularly well developed approach to dealing with missing values.

We may want to impute missing values in the data (not always wise to do so). Here we do this using `na.roughfix()` from randomForest (Breiman *et al.*, 2012).

As previously, we will demonstrate the process here but then restore the original dataset as these operations are not required for our dataset here.

```
ods <- ds
```

```
dim(ds[vars])

## [1] 366  21

sum(is.na(ds[vars]))

## [1] 47

ds[vars] <- na.roughfix(ds[vars])
sum(is.na(ds[vars]))

## [1] 0

dim(ds[vars])

## [1] 366  21
```

```
ds <- ods
```

13 Step 3: Clean—Omitting Observations

We might want to simply remove observations that have missing values. Here `na.omit()` identifies the rows to omit based on the `vars` to be included for modelling. This list of rows to omit is stored as the `na.action` attribute of the returned object. We then remove these observations from the dataset.

We start again by keeping a copy of the original dataset to restore below. We also initialise a list of row indicies that we will omit from the dataset.

```
ods <- ds
omit <- NULL
```

```
dim(ds[vars])

## [1] 366  21

sum(is.na(ds[vars]))

## [1] 47

mo <- attr(na.omit(ds[vars]), "na.action")
omit <- union(omit, mo)
if (length(omit)) ds <- ds[-omit,]
sum(is.na(ds[vars]))

## [1] 0

dim(ds[vars])

## [1] 328  21
```

Restore the dataset.

```
ds <- ods
```

14 Step 3: Clean—Normalise Factors

Some variables will have levels with spaces, and mixture of cases, etc. We may like to normalise the levels for each of the categoric variables. For very large datasets this can take some time and so we may want to be selective.

```
factors <- which(sapply(ds[vars], is.factor))
for (f in factors) levels(ds[[f]]) <- normVarNames(levels(ds[[f]]))
```

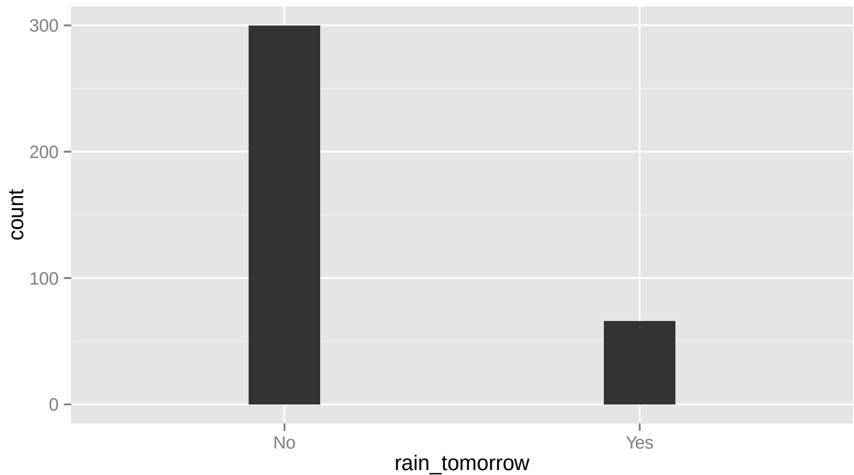
15 Step 3: Clean—Ensure Target is Categoric

For classification models we want to ensure the target is categoric. Often it is 0/1 and hence is loaded as numeric. We could tell our modeller of choice to explicitly do classification, or set the target using `as.factor()` in the formula, but it is generally cleaner to do this here, and this is a no-op if the target is already categoric.

```
ds[target] <- as.factor(ds[[target]])
table(ds[target])

##
##  No Yes
## 300  66
```

Here we visualise the distribution of the target variable using ggplot2 (Wickham and Chang, 2013).

```
p <- ggplot(ds, aes_string(x=target))
p <- p + geom_bar(width=0.2)
print(p)
```



16 Step 4: Prepare—Variables

We are now ready to identify the variables that we will use to build the model. Previously we identified the variable roles. Now we identify those that we wish to model.

```
(inputs  <- setdiff(vars, target))

##  [1] "min_temp"        "max_temp"        "rainfall"
##  [4] "evaporation"     "sunshine"        "wind_gust_dir"
##  [7] "wind_gust_speed" "wind_dir_9am"    "wind_dir_3pm"
## [10] "wind_speed_9am"  "wind_speed_3pm"  "humidity_9am"
....

(nobs    <- nrow(ds))

## [1] 366

dim(ds[vars])

## [1] 366  21
```

Sometimes we need to identify the numeric and categoric variables. Many cluster analysis algorithms only deal with numeric variables, for example. Here we identify them both by name and by index. Note that when using the index we have to assume the variables always remain in the same order within the dataset and all variables are present. Otherwise the indicies will get out of sync.

```
numi       <- which(sapply(ds[inputs], is.numeric))
numi

##        min_temp        max_temp        rainfall     evaporation
##               1               2               3               4
##        sunshine wind_gust_speed  wind_speed_9am  wind_speed_3pm
##               5               7              10              11
....

numerics   <- names(numi)
numerics

##  [1] "min_temp"        "max_temp"        "rainfall"
##  [4] "evaporation"     "sunshine"        "wind_gust_speed"
##  [7] "wind_speed_9am"  "wind_speed_3pm"  "humidity_9am"
## [10] "humidity_3pm"    "pressure_9am"    "pressure_3pm"
....

cati       <- which(sapply(ds[inputs], is.factor))
cati

## wind_gust_dir  wind_dir_9am  wind_dir_3pm    rain_today
##             6             8             9            20

categorics <- names(cati)
categorics

## [1] "wind_gust_dir" "wind_dir_9am"  "wind_dir_3pm"  "rain_today"
```

17 Step 4: Prepare—Save Dataset

For large datasets we may want to save it to a bianry RData file once we have it in the right shape. It will also save reading from CSV again—a CSV file with 2 million observations and 800 variables might take 30 minutes to read.csv(), 5 minutes to save(), and 30 seconds to load().

```
dsdate <- paste0("_", format(Sys.Date(), "%y%m%d"))
dsrdata <- paste0(dsname, dsdate, ".RData")
save(ds, dsname, dspath, dsdate, target, risk, id, ignore, vars,
     nobs, omit, inputs, numi, numerics, cati, categorics, file=dsrdata)
```

We would only do the above steps once, and then each time we wish to use the dataset, we would `load()` it into R.

```
(load(dsrdata))

##  [1] "ds"         "dsname"     "dspath"     "dsdate"     "target"
##  [6] "risk"       "id"         "ignore"     "vars"       "nobs"
## [11] "omit"       "inputs"     "numerics"   "categorics"

dsname

## [1] "weather"

dspath

## [1] "/home/gjw/R/x86_64-pc-linux-gnu-library/3.0/rattle/csv/weather.csv"

dsdate

## [1] "_130704"

dim(ds)

## [1] 366  24

id

## [1] "date"     "location"

target

## [1] "rain_tomorrow"

risk

## [1] "risk_mm"

ignore

## [1] "date"     "location" "risk_mm"  "1"

vars

##  [1] "min_temp"         "max_temp"         "rainfall"
##  [4] "evaporation"      "sunshine"         "wind_gust_dir"
##  [7] "wind_gust_speed" "wind_dir_9am"     "wind_dir_3pm"
## [10] "wind_speed_9am"  "wind_speed_3pm"   "humidity_9am"
....
```

18 Review—Data Preparation Process—Load and Clean

Here in one sequence (across this and the following section) is the code to perform all of the data preparation. Notice that we would not necessarily do all of these, such as lower casing the variable names, imputing missing values, omitting observations with missing values, etc, so pick and choose as is appropriate to your situation.

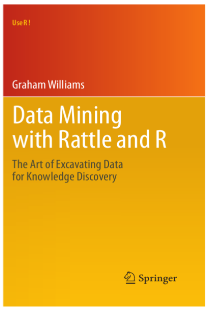
```
# Required packages
library(rattle)          # normVarNames()
library(randomForest)    # Impute missing using na.roughfix()

# Data setup
dspath      <- system.file("csv", "weather.csv", package="rattle")
weather     <- read.csv(dspath)
dsname      <- "weather"
ds          <- get(dsname)
names(ds)   <- normVarNames(names(ds)) # Optional lower case variable names.
vars        <- names(ds)
target      <- "rain_tomorrow"
risk        <- "risk_mm"
id          <- c("date", "location")

# Summarise
dim(ds)
names(ds)
head(ds)
tail(ds)
ds[sample(nrow(ds), 6),]
str(ds)
summary(ds)

# Variables to ignore
ignore      <- c(id, if (exists("risk")) risk)
mvc         <- sapply(ds[vars], function(x) sum(is.na(x))) # Missing value count.
mvn         <- names(ds)[(which(mvc == nrow(ds)))]         # Missing var names.
ignore      <- union(ignore, mvn)

factors     <- which(sapply(ds[vars], is.factor))
lvls        <- sapply(factors, function(x) length(levels(ds[[x]])))
many        <- names(which(lvls > 20))    # Factors with too many levels.
ignore      <- union(ignore, many)

vars        <- setdiff(vars, ignore)

# Normalise factors
factors     <- which(sapply(ds[vars], is.factor))
for (f in factors) levels(ds[[f]]) <- normVarNames(levels(ds[[f]]))

# Remove all observations with a missing target.
```

```
ds          <- ds[!is.na(ds[target]),]

# Optionally impute missing values, but do this wisely - understand why missing.
if (sum(is.na(ds[vars]))) ds[vars] <- na.roughfix(ds[vars])

# Observations to omit
omit        <- NULL
mo          <- attr(na.omit(ds[vars]), "na.action")
omit        <- union(omit, mo)
if (length(omit)) ds <- ds[-omit,]      # Optional remove ommited observations.
```

19 Review—Data Preparation Process—Finalise

```
# Finalise
ds[target] <- as.factor(ds[[target]])
inputs     <- setdiff(vars, target)
nobs       <- nrow(ds)
numi       <- which(sapply(ds[inputs], is.numeric))
numerics   <- names(numi)
cati       <- which(sapply(ds[inputs], is.factor))
categorics <- names(cati)

# Save the data
dsdate     <- paste0("_", format(Sys.Date(), "%y%m%d"))
dsrdata    <- paste0(dsname, dsdate, ".RData")
save(ds, dsname, dspath, dsdate, target, risk, id, ignore, vars,
     nobs, omit, inputs, numerics, categorics, file=dsrdata)
```

20 Further Reading



The Rattle Book, published by Springer, provides a comprehensive introduction data mining and analytics using Rattle and R. It is available from Amazon. Other documentation on a broader selection of R topics of relevance to the data scientist is freely available from `http://datamining.togaware.com`, including the Datamining Desktop Survival Guide.

This module is one of many OnePageR modules available from `http://onepager.togaware.com`. In particular follow the links on the website with a * which indicates the generally more developed OnePageR modules.

The process we have resented here for preparing the data for modelling has been tuned over many years of delivering analytics and data mining projects. An early influence was CRISP-DM the CRoss Industry Standard Process for Data Mining.

21 References

Breiman L, Cutler A, Liaw A, Wiener M (2012). *randomForest: Breiman and Cutler's random forests for classification and regression.* R package version 4.6-7, URL http://CRAN.R-project.org/package=randomForest.

R Core Team (2013). *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria. URL http://www.R-project.org/.

Wickham H, Chang W (2013). *ggplot2: An implementation of the Grammar of Graphics.* R package version 0.9.3.1, URL http://CRAN.R-project.org/package=ggplot2.

Williams GJ (2009). "Rattle: A Data Mining GUI for R." *The R Journal*, **1**(2), 45–55. URL http://journal.r-project.org/archive/2009-2/RJournal_2009-2_Williams.pdf.

Williams GJ (2011). *Data Mining with Rattle and R: The art of excavating data for knowledge discovery.* Use R! Springer, New York. URL http://www.amazon.com/gp/product/1441998896/ref=as_li_qf_sp_asin_tl?ie=UTF8&tag=togaware-20&linkCode=as2&camp=217145&creative=399373&creativeASIN=1441998896.

Williams GJ (2014). *rattle: Graphical user interface for data mining in R.* R package version 3.0.2, URL http://rattle.togaware.com/.

This document, sourced from DataO.Rnw revision 282, was processed by KnitR version 1.5 of 2013-09-28 and took 3.5 seconds to process. It was generated by gjw on nyx running Ubuntu 13.10 with Intel(R) Xeon(R) CPU W3520 @ 2.67GHz having 4 cores and 12.3GB of RAM. It completed the processing 2014-02-14 06:13:25.