One PageR Data Science
Dates and Time

Graham.Williams@togaware.com

16th May 2018

Visit https://essentials.togaware.com/onepagers for more Essentials.

Date and time data is common in many disciplines, particularly where our observations are of some event over time. R has well developed support for dealing with such data, in wranlging the data, summarising the data and analysing the data. In this chapter we review the common tasks when dealing with date and time data. *20180513*

Packages used in this chapter include tidyverse (Wickham, 2017), lubridate (Spinu *et al.*, 2018), magrittr (Bache and Wickham, 2014), rattle (Williams, 2017b), WDI (Arel-Bundock, 2018b), countrycode (Arel-Bundock, 2018a),

```
# Load required packages from local library into the R session.

library(tidyverse)     # ggplot2, tibble, tidyr, readr, purr, dplyr
library(lubridate)     # Dates and time.
library(magrittr)      # Pipe operator %>% %<>% %T>% equals().
library(rattle)        # comcat().
library(WDI)           # World Bank Data
library(countrycode)
library(gridExtra)
```

Through this guide new R commands will be introduced. The reader is encouraged to review the command's documentation and understand what the command does. Help is obtained using the `?` command as in:

```
?read.csv
```

Documentation on a particular package can be obtained using the `help=` option of `library()`:

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

This chapter is intended to be hands on. To learn effectively the reader is encouraged to run R locally (e.g., RStudio or Emacs with ESS mode) and to replicate all commands as they appear here. Check that output is the same and it is clear how it is generated. Try some variations. Explore.

1 Dataset with Dates

Often when ingesting data into R we will have a dataset in a CSV file that contains dates. An example is the **stroke** dataset. Here we set up template variables (`dsname`, `dsloc`, and `dspath`), and whilst constructing the value of the path variable we print the constructed path and display the first few lines from the CSV file for our observation. *20180513*

```
# Name of the dataset.

dsname <- "stroke"

# Identify the source location of the dataset.

dsloc <- "data"

# Construct the path to the dataset and display some if it.

dsname %>%
  paste0(".csv") %>%
  file.path(dsloc, .) %T>%
  cat("Dataset:", ., "\n\n") %T>%
  {
     paste("head", .) %>%
     system(intern=TRUE) %>%
     sub("\r", "\n", .) %>%
     print()
  } ->
dspath

## Dataset: data/stroke.csv
##
##  [1] "SEX;DIED;DSTR;AGE;DGN;COMA;DIAB;MINF;HAN\n"
##  [2] "1;7.01.1991;2.01.1991;76;INF;0;0;1;0\n"
##  [3] "1;.;3.01.1991;58;INF;0;0;0;0\n"
##  [4] "1;2.06.1991;8.01.1991;74;INF;0;0;1;1\n"
##  [5] "0;13.01.1991;11.01.1991;77;ICH;0;1;0;1\n"
##  [6] "0;23.01.1996;13.01.1991;76;INF;0;1;0;1\n"
##  [7] "1;13.01.1991;13.01.1991;48;ICH;1;0;0;1\n"
##  [8] "0;1.12.1993;14.01.1991;81;INF;0;0;0;1\n"
##  [9] "1;12.12.1991;14.01.1991;53;INF;0;0;1;1\n"
## [10] "0;.;15.01.1991;73;ID;0;0;0;1\n"
```

Observe that this CSV file actually uses semicolons rather than commas to separate the fields. This is common in countries where the comma is used to separate the decimal digits from the whole digits. We can use `readr::read_csv2()` to ingest such a CSV file.

Also observe that the two date columns `DIED` and `DSTR` are in a particular (even peculiar) format which we might determine to be day then month then year, separated by a period.

Finally, observe that missing values appear to be represented as a single period.

2 Ingest Semicolon Separated Dataset

We can now ingest the data into R using `readr::read_csv2()` with an appropriate `na="."`. *20180514*

```
dspath %>%
  read_csv2(na=".") %T>%
  glimpse() %>%
  assign(dsname, ., .GlobalEnv)

## Using ',' as decimal and '.'  as grouping mark.
## Use read_delim() for more control.

## Parsed with column specification:
## cols(
##   SEX = col_integer(),
##   DIED = col_number(),
##   DSTR = col_number(),
##   AGE = col_integer(),
##   DGN = col_character(),
##   COMA = col_integer(),
##   DIAB = col_integer(),
##   MINF = col_integer(),
##   HAN = col_integer()
## )

## Observations: 829
## Variables: 9
## $ SEX  <int> 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, ...
## $ DIED <dbl> 7011991, NA, 2061991, 13011991, 23011996, 130...
## $ DSTR <dbl> 2011991, 3011991, 8011991, 11011991, 13011991...
## $ AGE  <int> 76, 58, 74, 77, 76, 48, 81, 53, 73, 69, 86, 7...
## $ DGN  <chr> "INF", "INF", "INF", "ICH", "INF", "ICH", "IN...
## $ COMA <int> 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ DIAB <int> 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ MINF <int> 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, ...
## $ HAN  <int> 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, ...
```

Notice the message indicating that comma is interpreted as a decimal separator and period as a grouping mark. This is based on the fact of the use of semicolon as the field separator and the usual motivation for doing so (comma used for decimal). For this dataset those assumptions don't actually hold. Observe that the two date columns have been treated as numbers under this assumption where the string of numbers with multiple periods is interpreted as numeric.

We decide to follow the suggestion to use `readr::read_delim()` which provides more control over the ingestion.

Nonetheless we observe for the first time that the dataset consists of 829 observations of 9 variables. There appear to be a number of binary encoded variables, that may indicate they represent `TRUE` and `FALSE`, whilst `SEX` would appear to encode male/female as 0/1 or 1/0, though without further information we do not know which of these encoding it is.

3 Basic Dataset Ingestion

The function `readr::read_delim()` makes fewer assumptions about the data and in this case will be a better option. We can specify the field delimiter using `delim=";"` whilst also retaining the missing value argument. *20180514*

```
dspath %>%
  read_delim(delim=";", na=".") %T>%
  glimpse() %>%
  assign(dsname, ., .GlobalEnv)

## Parsed with column specification:
## cols(
##   SEX = col_integer(),
##   DIED = col_character(),
##   DSTR = col_character(),
##   AGE = col_integer(),
##   DGN = col_character(),
##   COMA = col_integer(),
##   DIAB = col_integer(),
##   MINF = col_integer(),
##   HAN = col_integer()
## )

## Observations: 829
## Variables: 9
## $ SEX  <int> 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, ...
## $ DIED <chr> "7.01.1991", NA, "2.06.1991", "13.01.1991", "...
## $ DSTR <chr> "2.01.1991", "3.01.1991", "8.01.1991", "11.01...
## $ AGE  <int> 76, 58, 74, 77, 76, 48, 81, 53, 73, 69, 86, 7...
....
```

That is now a good start to ingesting this data into R. The dates will be wrangled shortly but ingesting them as character strings retains their format.

Following our template approach we copy the dataset to our template variable (`ds`) so that we can work on the data using the generic template name.

```
# Prepare the dataset for usage with our template.

ds <- get(dsname)
```

4 Normalise the Dataset

```
# Review the variables to optionally normalise their names.

names(ds)

## [1] "SEX"  "DIED" "DSTR" "AGE"  "DGN"  "COMA" "DIAB" "MINF"
## [9] "HAN"

# Normalise the variable names.

names(ds) %<>% normVarNames() %T>% print()

## [1] "sex"  "died" "dstr" "age"  "dgn"  "coma" "diab" "minf"
## [9] "han"

# Confirm the results are as expected.

glimpse(ds)

## Observations: 829
## Variables: 9
## $ sex  <int> 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, ...
## $ died <chr> "7.01.1991", NA, "2.06.1991", "13.01.1991", "...
## $ dstr <chr> "2.01.1991", "3.01.1991", "8.01.1991", "11.01...
## $ age  <int> 76, 58, 74, 77, 76, 48, 81, 53, 73, 69, 86, 7...
....
```

5 Text to Date Conversion Using Lubridate

We notice that there are two variables that look like dates: `died` and `dstr`.

They have been read in as character strings. Because the format of the dates is not an obvious date format the `readr::read_delim()` has not recognised them as dates. For more standard formats any date columns will be automatically identified.

The lubridate (Spinu *et al.*, 2018) package can perform the conversion into a date format for us here. We can convert the dates using `lubridate::dmy()` since the source format appears to be *d*ay, *m*onth, the *y*ear.

```
ds$died %<>% dmy() %T>% {class(.) %>% print()}

## [1] "Date"

ds$dstr %<>% dmy() %T>% {class(.) %>% print()}

## [1] "Date"

glimpse(ds)

## Observations: 829
## Variables: 9
## $ sex  <int> 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, ...
## $ died <date> 1991-01-07, NA, 1991-06-02, 1991-01-13, 1996...
## $ dstr <date> 1991-01-02, 1991-01-03, 1991-01-08, 1991-01-...
## $ age  <int> 76, 58, 74, 77, 76, 48, 81, 53, 73, 69, 86, 7...
....
```

The data types are now `Date`.

6 Text to Date Conversion Using Base R

As an alternative we could have used `as.Date()` to convert them into a Date class. Because the original format is not automatically recognised by `as.Date()` we need to tell it the format using `format=`.

```
tmp <- stroke

tmp$DIED %<>% as.Date(format="%d.%m.%Y") %T>% {class(.) %>% print()}

## [1] "Date"

tmp$DSTR %<>% as.Date(format="%d.%m.%Y") %T>% {class(.) %>% print()}

## [1] "Date"

glimpse(tmp)

## Observations: 829
## Variables: 9
## $ SEX  <int> 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, ...
## $ DIED <date> 1991-01-07, NA, 1991-06-02, 1991-01-13, 1996...
## $ DSTR <date> 1991-01-02, 1991-01-03, 1991-01-08, 1991-01-...
## $ AGE  <int> 76, 58, 74, 77, 76, 48, 81, 53, 73, 69, 86, 7...
....

rm(tmp)
```

Notice now that the dates are printed in a standard ISO format which is `%Y-%m-%d` and is strongly suggested as the preferred format so as to remove any ambiguity .

Generated 16th May 2018 12:37pm

7 POSIXct and POSIXlt

Objects of class POSIXct (calendar time) and POSIXlt (local time) represent calendar dates and times. They both represent the same information, but in different ways, calendar time as a single number and local time as a vector of the components making up the date/time. Both POSIXct and POSIXlt objects are also POSIXt objects, thus effectively inheriting from the common class POSIXt, allowing operations on mixed class (POSIXct and POSIXlt) objects. Generally, for data frames we use POSIXct. POSIXlt is more directly accessible for us to read.

POSIXct is simply the number of seconds since 1 January 1970.

```
(ct <- Sys.time())

## [1] "2018-05-16 12:37:47 +08"

class(ct)

## [1] "POSIXct" "POSIXt"

str(ct)

##  POSIXct[1:1], format: "2018-05-16 12:37:47"

unclass(ct)

## [1] 1526445467
```

POSIXlt (local time) represents the date and time as a named list of vectors.

```
(ct <- as.POSIXlt(ct))

## [1] "2018-05-16 12:37:47 +08"

class(ct)

## [1] "POSIXlt" "POSIXt"

str(ct)

##  POSIXlt[1:1], format: "2018-05-16 12:37:47"

unclass(ct)

## $sec
## [1] 47.44999
##
## $min
## [1] 37
##
## $hour
## [1] 12
##
## $mday
## [1] 16
##
....
```


8 Formatting Dates

A wide variety of formats are supported in printing a date and time. The format string is a common standard used with many applications.

To print a date/time to a specific format we specify the format with in the call to `format()`:

```
format(Sys.time(), "%a %d %b %Y %H:%M:%S %Z")

## [1] "Wed 16 May 2018 12:37:47 +08"
```

The table below illustrates many of the available options.

%c	date and time	Wed 16 May 2018 12:37:47 +08
%x	date	16/05/18
%F	ISO 8601	2018-05-16
%d/%m/%Y	day/month/year	16/05/2018
%a %e %m %Y	day month year	Wed 16 May 2018
%A %d %B %Y	day month year	Wednesday 16 May 2018
Day %j and Week %U of %Y	day/week of the year	Day 136 and Week 19 of 2018
%A: Day %w of Week %U	day of week	Wednesday: Day 3 of Week 19
%y%m%d	two digit date stamp	180516
%X	time	12:37:47
%r	time	12:37:47 PM
%k.%M %p	24 hour time	12.37 PM
%l.%M %p	12 hour time	12.37 PM
%H%M%S	timestamp	123747
%I:%M:%S %p	time 12 hour clock	12:37:47 PM
%H:%M:%S %z	time and UTC offset	12:37:47 +0800
%H:%M:%S %Z	time and timezone	12:37:47 +08

There are more! See the help page for `strptime()` for details.

9 Computing on Dates and Times: difftime

R Dates can be used in computations quite naturally.

```
ds$lived <- ds$died - ds$dstr
head(ds$lived)

## Time differences in days
## [1]    5   NA  145    2 1836    0

class(ds$lived)

## [1] "difftime"
```

Similarly POSIXct representations can be computed on, though the results are reported in seconds rather than days, by default. A Date does not include a time, hence we might expect Date calculations to be in days.

```
ds$lived <- ds$died - ds$dstr
head(ds$lived)

## Time differences in days
## [1]    5   NA  145    2 1836    0

class(ds$lived)

## [1] "difftime"
```

```
as.integer(ds$lived[1])/60/60/24

## [1] 5.787037e-05
```

We can change the default displayed units if desired.

```
units(ds$lived)

## [1] "days"

units(ds$lived) <- "days"
units(ds$lived)

## [1] "days"

head(ds$lived)

## Time differences in days
## [1]    5   NA  145    2 1836    0
```


Generated 16th May 2018 12:37pm

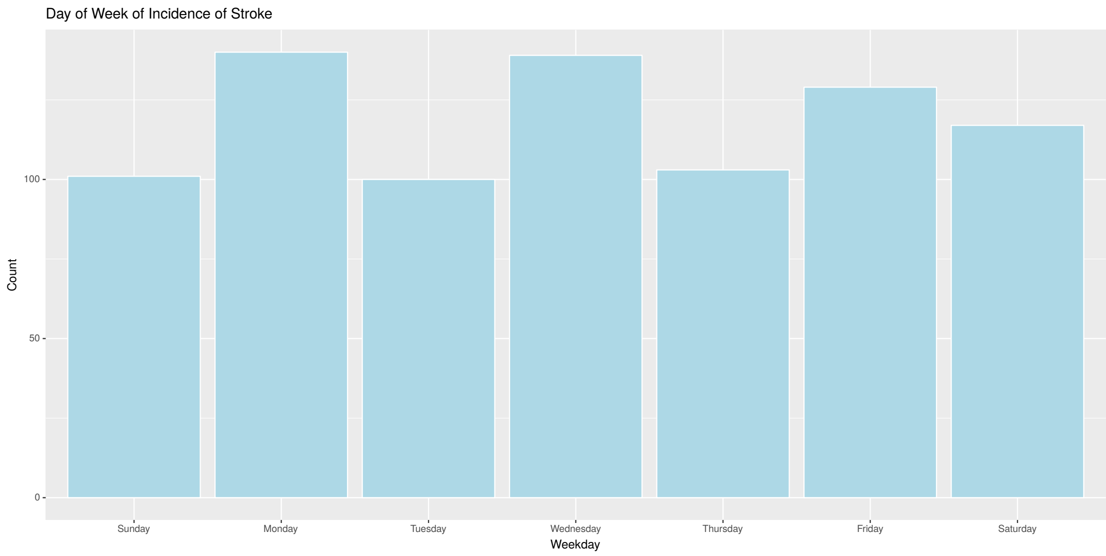
10 Lubridate Intervals

```
ds$interval <- with(ds, interval(dstr, died))
head(ds$interval)

## [1] 1991-01-02 UTC--1991-01-07 UTC
## [2] 1991-01-03 UTC--NA
## [3] 1991-01-08 UTC--1991-06-02 UTC
## [4] 1991-01-11 UTC--1991-01-13 UTC
## [5] 1991-01-13 UTC--1996-01-23 UTC
## [6] 1991-01-13 UTC--1991-01-13 UTC
....

class(ds$interval)

## [1] "Interval"
## attr(,"package")
## [1] "lubridate"

max(ds$interval, na.rm=TRUE)

## [1] 158630400

min(ds$interval, na.rm=TRUE)

## [1] 0
```

```
head(as.duration(ds$interval))

## [1] "432000s (~5 days)"        NA
## [3] "12528000s (~20.71 weeks)" "172800s (~2 days)"
## [5] "158630400s (~5.03 years)" "0s"
```

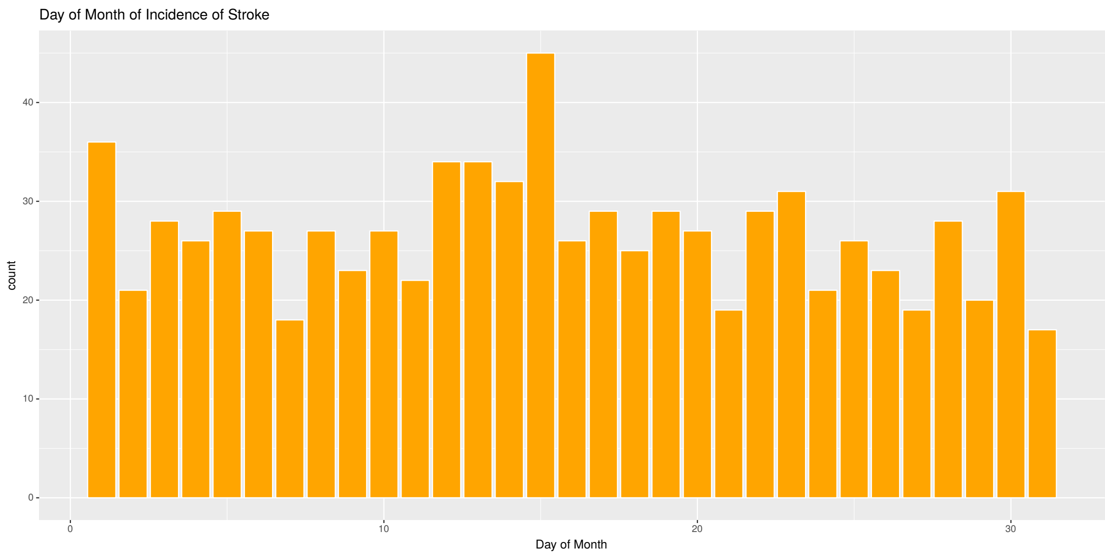

11 Plot Day of Week Frequencies

```
ds %>%
  mutate(weekday=wday(dstr, label=TRUE, abbr=FALSE)) %>%
  ggplot(aes(weekday)) +
  geom_bar(colour="white", fill="lightblue") +
  labs(title="Day of Week of Incidence of Stroke", x="Weekday", y="Count")
```



12 Plot Day of Month Frequencies

```
ds %>%
  mutate(mday=mday(dstr)) %>%
  ggplot(aes(mday)) +
  geom_bar(colour="white", fill="orange") +
  labs(title="Day of Month of Incidence of Stroke", x="Day of Month")
```



13 Plot Daily Observations

```
data.frame(L1=100+c(0, cumsum(runif(99, -20, 20))),
           L2=150+c(0, cumsum(runif(99, -10, 10))),
           Date=seq.Date(as.Date("2000-01-01"),
                         by="1 month", length.out=100)) %>%
  melt(id="Date") %>%
  ggplot(aes(x=Date, y=value, colour=variable)) +
  geom_line() +
  ylab("Observation") +
  labs(colour="Location")

## Error in melt(., id = "Date"):  could not find function "melt"
```


14 Plot World Bank Data: Obtain Data

This example was inspired by the ProgrammingR blog post of 14 May 2013.

The World Bank provide economic indicators on the Internet available via an API. We can access the data using WDI (Arel-Bundock, 2018b). We also use countrycode (Arel-Bundock, 2018a) to map the country codes.

We search the World Bank data for the fertility rate data using `WDIsearch()`. We identify the countries we are interested in, convert them to their two character country codes and then extract the country data from the World Bank for a ten year period.

```
(meta.data <- WDIsearch("Fertility rate", field="name", short=FALSE))

##       indicator
##  [1,] "SP.FER.TOTL.ZR"
##  [2,] "SP.DYN.WFRT.Q5"
##  [3,] "SP.DYN.WFRT.Q4"
##  [4,] "SP.DYN.WFRT.Q3"
##  [5,] "SP.DYN.WFRT.Q2"
....

(indicators <- meta.data[1:2, 1])

## [1] "SP.FER.TOTL.ZR" "SP.DYN.WFRT.Q5"

countries <- c("United States", "Britain", "India", "China", "Australia")
(iso2char <- countrycode(countries, "country.name", "iso2c"))

## [1] "US" "GB" "IN" "CN" "AU"

(wdids <- WDI(iso2char, meta.data[1:2,1], start=2001, end=2011))

## Warning in WDI(iso2char, meta.data[1:2, 1], start = 2001, end = 2011):  Unable
to download indicators  SP.FER.TOTL.ZR

##    iso2c        country SP.DYN.WFRT.Q5 year
## 1     AU      Australia             NA 2011
## 2     AU      Australia             NA 2010
## 3     AU      Australia             NA 2009
## 4     AU      Australia             NA 2008
## 5     AU      Australia             NA 2007
....
```


15 Plot World Bank Data: Multiple Plots

Generate the plots. We generate a list of plots, by applying a function to each indicator. Notice inside the function the call to `ggplot()` uses `environment=environment()` to ensure the variable `nm` is available to the `aes()`.

```
plots <- lapply(indicators, function(nm)
{
  p <- ggplot(wdids, aes(x=year, y=wdids[,nm], group=country, color=country),
              environment=environment())
  p <- p + geom_line(size=1)
  p <- p + scale_x_continuous(name="Year", breaks=c(unique(wdids[,"year"])))
  p <- p + scale_y_continuous(name=nm)
  p <- p + scale_linetype_discrete(name="Country")
  p <- p + theme(legend.title=element_blank())
  p <- p + labs(title=paste(meta.data[meta.data[,1]==nm, "name"], "\n"))
})
```

Once we have our list of plots, we can call `grid.arrange()` to arrange the plots to be displayed.

```
do.call(grid.arrange, plots)

## Error in `[.data.frame`(wdids, , nm):  undefined columns selected
```

16 Time Series Plot

We will prepare a dataset to illustrate a number of options for plotting. We first pick a few variables to plot.

```
vars <- c("Date", "MinTemp", "MaxTemp", "Sunshine", "Rainfall", "Evaporation")
ds <- weather[vars]

## Error in eval(expr, envir, enclos):  object 'weather' not found
```

We want to illustrate a common issue with different scales on the one plot, so we convert the hours of sunshine into seconds.

```
ds$Sunshine <- ds$Sunshine * 60

## Warning:  Unknown or uninitialised column:  'Sunshine'.

## Error in `$<-.data.frame`(`*tmp*`, Sunshine, value = numeric(0)):  replacement
has 0 rows, data has 829
```

We will also accumulate the amount of rainfall and the amount of evaporation over the period:

```
ds$CumRainfall <- cumsum(ds$Rainfall)

## Warning:  Unknown or uninitialised column:  'Rainfall'.

## Error in `$<-.data.frame`(`*tmp*`, CumRainfall, value = numeric(0)):  replacement
has 0 rows, data has 829

ds$CumEvaporation <- cumsum(ds$Evaporation)

## Warning:  Unknown or uninitialised column:  'Evaporation'.

## Error in `$<-.data.frame`(`*tmp*`, CumEvaporation, value = numeric(0)):  replacement
has 0 rows, data has 829
```

We now also melt the dataset into a form that will facilitate plotting all of the variables.

```
dsm <- melt(ds, id="Date")

## Error in melt(ds, id = "Date"):  could not find function "melt"
```

```
g <- ggplot(dsm, aes(x=Date, y=value, colour=variable))

## Error in ggplot(dsm, aes(x = Date, y = value, colour = variable)):  object 'dsm'
not found

g <- g + geom_point()

## Error in eval(expr, envir, enclos):  object 'g' not found

print(g)

## Error in print(g):  object 'g' not found
```

That's a start, but not real good. The very large numbers swamp the rest. Notice also the warning regarding observations with missing values. We'll ignore that (and turn the warning off for the following plots).

17 Rescale with a Log10 Transform

We can perform a log (base 10) transform to ensure the low valued variables get some resolution in the plot.

```
g <- ggplot(dsm, aes(x=Date, y=value, colour=variable))

## Error in ggplot(dsm, aes(x = Date, y = value, colour = variable)):  object 'dsm'
not found

g <- g + geom_point()

## Error in eval(expr, envir, enclos):  object 'g' not found

g <- g + scale_y_log10()

## Error in eval(expr, envir, enclos):  object 'g' not found

print(g)

## Error in print(g):  object 'g' not found
```

So that is a little better but note the warnings. We can not take the log of numbers less than or equal to zero. These data are ignored in plotting. That is not really what we wanted to do.

Generated 16th May 2018 12:37pm

18 Rescale with an asinh Transform for Negatives

We can use alternative transformations and one good transformation for rescaling positive and negative data is based on asinh (the inverse hyperbolic sine of the data). This handles negatives and zero and serves a similar purpose to the log transforms.

```
asinh_trans <- function() trans_new(name="asinh",
                                    transform=asinh,
                                    inverse=sinh)
g <- ggplot(dsm, aes(x=Date, y=value, colour=variable))

## Error in ggplot(dsm, aes(x = Date, y = value, colour = variable)):  object 'dsm'
not found

g <- g + geom_point()

## Error in eval(expr, envir, enclos):  object 'g' not found

g <- g + scale_y_continuous(trans="asinh")

## Error in eval(expr, envir, enclos):  object 'g' not found

print(g)

## Error in print(g):  object 'g' not found
```

We now get the negatives and zeros into the picture.

Generated 16th May 2018 12:37pm

19 Scale Options: Setting Limits on the Y Axis

The y axis is unbalanced above and below zero. That is usually just fine, but we can also balance it up if desired.

```
g <- ggplot(dsm, aes(x=Date, y=value, colour=variable))
## Error in ggplot(dsm, aes(x = Date, y = value, colour = variable)): object 'dsm'
not found
g <- g + geom_point()
## Error in eval(expr, envir, enclos): object 'g' not found
g <- g + scale_y_continuous(trans="asinh",
                            limits=c(-1e4, 1e4))
## Error in eval(expr, envir, enclos): object 'g' not found
print(g)
## Error in print(g): object 'g' not found
```

Actually though, there's quite a bit of wasted space now, so we'll drop the limits for the following plots. There is no point really in taking up precious real estate for no particular purpose.

20 Scale Options: Specify Breaks Along the Y Axis

The y axis labels are somewhat sparse, with no indications between 0 and 1,000. We can spice that up a little by specifying where the breaks along the axis should be labelled.

```
g <- ggplot(dsm, aes(x=Date, y=value, colour=variable))

## Error in ggplot(dsm, aes(x = Date, y = value, colour = variable)):  object 'dsm'
not found

g <- g + geom_point()

## Error in eval(expr, envir, enclos):  object 'g' not found

g <- g + scale_y_continuous(trans="asinh",
                            breaks=c(-10, 0, 10, 1e2, 1e3))

## Error in eval(expr, envir, enclos):  object 'g' not found

print(g)

## Error in print(g):  object 'g' not found
```

This does add value to the plot. The actual gradation of points along the y axis is now much easier to perceive.

21 Scale Options: Label the Breaks

As well as specifying the breaks we can also specify how they are to be labelled. This could be useful when we want to abbreviate the labels in some standard way, if that improves the readability.

```
g <- ggplot(dsm, aes(x=Date, y=value, colour=variable))

## Error in ggplot(dsm, aes(x = Date, y = value, colour = variable)):  object 'dsm'
not found

g <- g + geom_point()

## Error in eval(expr, envir, enclos):  object 'g' not found

g <- g + scale_y_continuous(trans="asinh",
                            breaks=c(-10, 0, 10, 1e2, 1e3),
                            labels=c("-10", "0", "10", "100", "1K"))

## Error in eval(expr, envir, enclos):  object 'g' not found

print(g)

## Error in print(g):  object 'g' not found
```

22 Plot Lines instead of Points

```
g <- ggplot(dsm, aes(x=Date, y=value, colour=variable))

## Error in ggplot(dsm, aes(x = Date, y = value, colour = variable)):  object 'dsm'
not found

g <- g + geom_line()

## Error in eval(expr, envir, enclos):  object 'g' not found

g <- g + scale_y_continuous(trans="asinh",
                            breaks=c(-10, 0, 10, 1e2, 1e3),
                            labels=c("-10", "0", "10", "100", "1K"))

## Error in eval(expr, envir, enclos):  object 'g' not found

print(g)

## Error in print(g):  object 'g' not found
```

That is pretty messy looking and the story is hard to tell.

23 Plot Points and Lines

The two cumulative plots might be better as lines and the others as points. Thus we will have a mixture of point and line geometries.

```
draw.lines <- c("CumRainfall", "CumEvaporation")

g <- ggplot(dsm, aes(x=Date, y=value, colour=variable))

## Error in ggplot(dsm, aes(x = Date, y = value, colour = variable)):  object 'dsm'
not found

g <- g + geom_point(data=subset(dsm, !variable %in% draw.lines))

## Error in eval(expr, envir, enclos):  object 'g' not found

g <- g + geom_line(data=subset(dsm, variable %in% draw.lines))

## Error in eval(expr, envir, enclos):  object 'g' not found

g <- g + scale_y_continuous(trans="asinh",
                            breaks=c(-10, 0, 10, 1e2, 1e3),
                            labels=c("-10", "0", "10", "100", "1K"))

## Error in eval(expr, envir, enclos):  object 'g' not found

print(g)

## Error in print(g):  object 'g' not found
```


24 Vertical Lines and Text

There may be significant dates we wish to note on the plot. Here we add two vertical lines that may be of some relevance. We use `geom_vline()` to do so but note that the intercept must be numeric. We'll use a dotted line (`linetype=3`) so the vertical lines are dominating the plot.

```
events <- as.Date(c("2007-12-25", "2008-03-22"))

g <- ggplot(dsm, aes(x=Date, y=value, colour=variable))

## Error in ggplot(dsm, aes(x = Date, y = value, colour = variable)):  object 'dsm'
not found

g <- g + geom_point(data=subset(dsm, !variable %in% draw.lines))

## Error in eval(expr, envir, enclos):  object 'g' not found

g <- g + geom_line(data=subset(dsm, variable %in% draw.lines))

## Error in eval(expr, envir, enclos):  object 'g' not found

g <- g + scale_y_continuous(trans="asinh",
                            breaks=c(-10, 0, 10, 1e2, 1e3),
                            labels=c("-10", "0", "10", "100", "1K"))

## Error in eval(expr, envir, enclos):  object 'g' not found

g <- g + geom_vline(xintercept=as.numeric(events), linetype=3)

## Error in eval(expr, envir, enclos):  object 'g' not found

g <- g + annotate("text", events[1], -9, label="Christmas", size=3, colour="blue")

## Error in eval(expr, envir, enclos):  object 'g' not found

g <- g + annotate("text", events[2], -9, label="Easter", size=3, colour="purple")

## Error in eval(expr, envir, enclos):  object 'g' not found

print(g)

## Error in print(g):  object 'g' not found
```

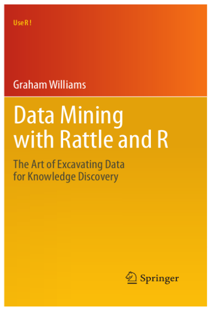

Generated 16th May 2018 12:37pm

25 Finishing Touches

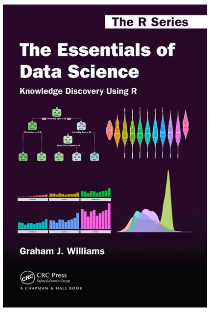
Add a title. Place the legend at the bottom.

```
g <- ggplot(dsm, aes(x=Date, y=value, colour=variable))

## Error in ggplot(dsm, aes(x = Date, y = value, colour = variable)):  object 'dsm'
not found

g <- g + geom_point(data=subset(dsm, !variable %in% draw.lines))

## Error in eval(expr, envir, enclos):  object 'g' not found

g <- g + geom_line(data=subset(dsm, variable %in% draw.lines))

## Error in eval(expr, envir, enclos):  object 'g' not found

g <- g + scale_y_continuous(trans="asinh",
                            breaks=c(-10, 0, 10, 1e2, 1e3),
                            labels=c("-10", "0", "10", "100", "1K"))

## Error in eval(expr, envir, enclos):  object 'g' not found

g <- g + geom_vline(xintercept=as.numeric(events), linetype=3)

## Error in eval(expr, envir, enclos):  object 'g' not found

g <- g + annotate("text", events[1], -9, label="Christmas", size=3, colour="blue")

## Error in eval(expr, envir, enclos):  object 'g' not found

g <- g + annotate("text", events[2], -9, label="Easter", size=3, colour="purple")

## Error in eval(expr, envir, enclos):  object 'g' not found

g <- g + labs(title=sprintf("Weather pattern for %s", weather$Location[1]))

## Error in eval(expr, envir, enclos):  object 'g' not found

g <- g + theme(legend.direction="horizontal", legend.position="bottom")

## Error in eval(expr, envir, enclos):  object 'g' not found

print(g)

## Error in print(g):  object 'g' not found
```


26 Further Reading and Acknowledgements



The Rattle Book (Williams, 2011), published by Springer, provides a comprehensive introduction to data mining and analytics using Rattle and R. It is available from Amazon. Rattle provides a graphical user interface through which the user is able to load, explore, visualise, and transform data, and to build, evaluate, and export models. Through its Log tab it specifically aims to provide an R template which can be exported and serve as the starting point for further programming with data in R.



The Essentials of Data Science book (Williams, 2017a), published by CRC Press, provides a comprehensive introduction to data science through programming with data using R. It is available from Amazon. The book provides a template based approach to doing data science and knowledge discovery. Templates are provided for data wrangling and model building. These serve as generic starting points for programming with data, and are designed to require minimal effort to get started. Visit `https://essentials.togaware.com` for further guides and templates.

Other resources include:

- Garrett Grolemund and Hadley Wickham's paper, *Dates and Time Made Easy with lubridate*, published in the Journal of Statistical Software, April 2011, Volume 40, Issue 3, provides a great introduction to effectively using lubridate. It is freely available at `http://www.jstatsoft.org/v40/i03/paper`.

27 References

Arel-Bundock V (2018a). *countrycode: Convert Country Names and Country Codes*. R package version 1.00.0, URL https://CRAN.R-project.org/package=countrycode.

Arel-Bundock V (2018b). *WDI: World Development Indicators (World Bank)*. R package version 2.5, URL https://CRAN.R-project.org/package=WDI.

Bache SM, Wickham H (2014). *magrittr: A Forward-Pipe Operator for R*. R package version 1.5, URL https://CRAN.R-project.org/package=magrittr.

R Core Team (2018). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

Spinu V, Grolemund G, Wickham H (2018). *lubridate: Make Dealing with Dates a Little Easier*. R package version 1.7.4, URL https://CRAN.R-project.org/package=lubridate.

Wickham H (2017). *tidyverse: Easily Install and Load the 'Tidyverse'*. R package version 1.2.1, URL https://CRAN.R-project.org/package=tidyverse.

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.

Williams GJ (2017a). *The Essentials of Data Science: Knowledge discovery using R*. The R Series. CRC Press.

Williams GJ (2017b). *rattle: Graphical User Interface for Data Science in R*. R package version 5.1.0, URL https://CRAN.R-project.org/package=rattle.

This document, sourced from DateTimeO.Rnw bitbucket revision 234, was processed by KnitR version 1.20 of 2018-02-20 10:11:46 UTC and took 9.9 seconds to process. It was generated by gjw on Ubuntu 18.04 LTS.

