

# Vignette for accessing Australian Weather Data

*Ebony Jones, Andrew Robinson, Kate Saunders and Adam Sparks*

*22 April 2016*

This vignette is designed to outline methods for accessing Bureau of Meteorology station data and the Australian Water Availability Project (AWAP) gridded datasets. The `rnoaa` package has Australian station data from the Global Historical Climatology Network (GHNC), for details see (<https://www.ncdc.noaa.gov/data-access/land-based-station-data/land-based-datasets/global-historical-climatology-network-ghcn>) .

## Outline:

### Installation

As some of the needed functionality is still in active development and the `rnoaa` package needs to be downloaded directly from github instead of Cran.

```
# devtools::install_github("rOpenSciLabs/rnoaa")
library(devtools)
devtools::install_github("hrbrmstr/rnoaa")
library(rnoaa)
```

### What station data is available?

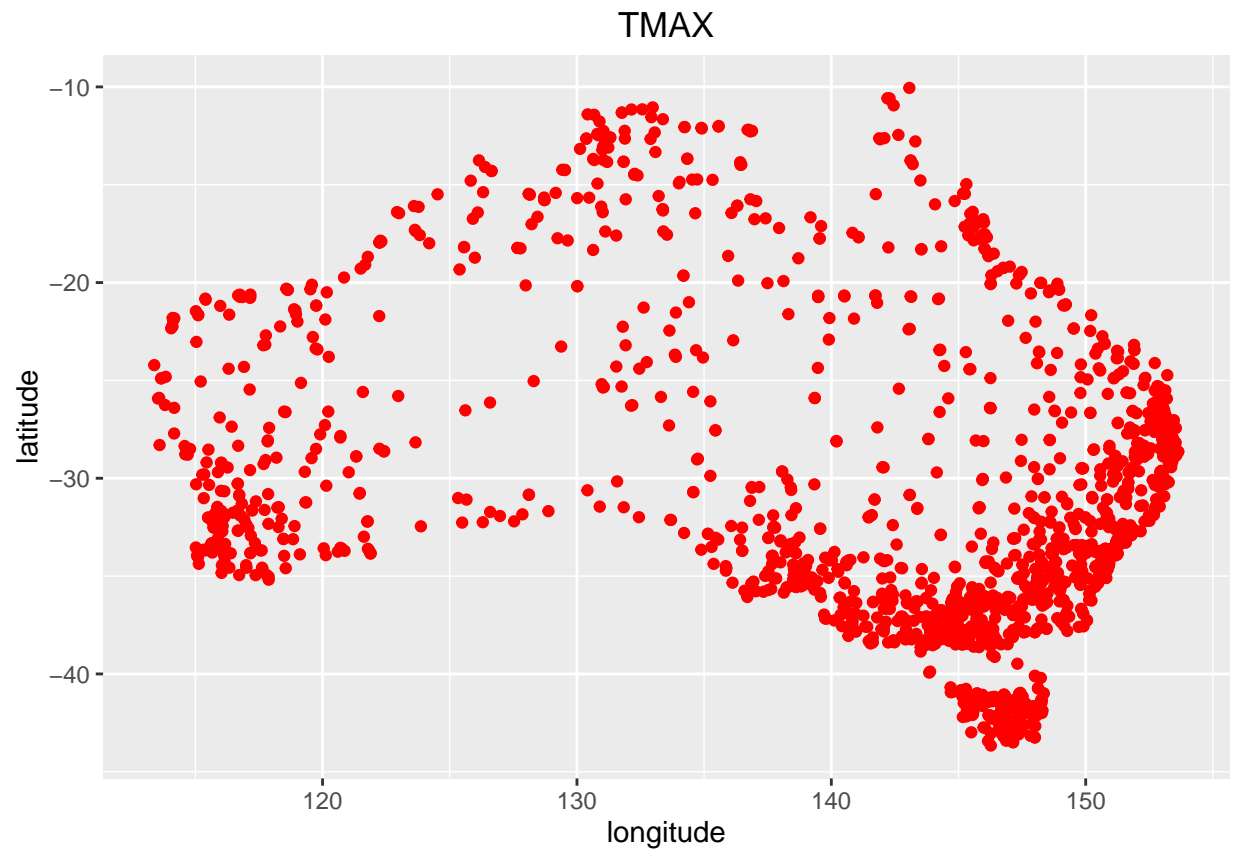
Currently available through the GHDN dataset are 16420 unique station names. The data types available for these stations are:

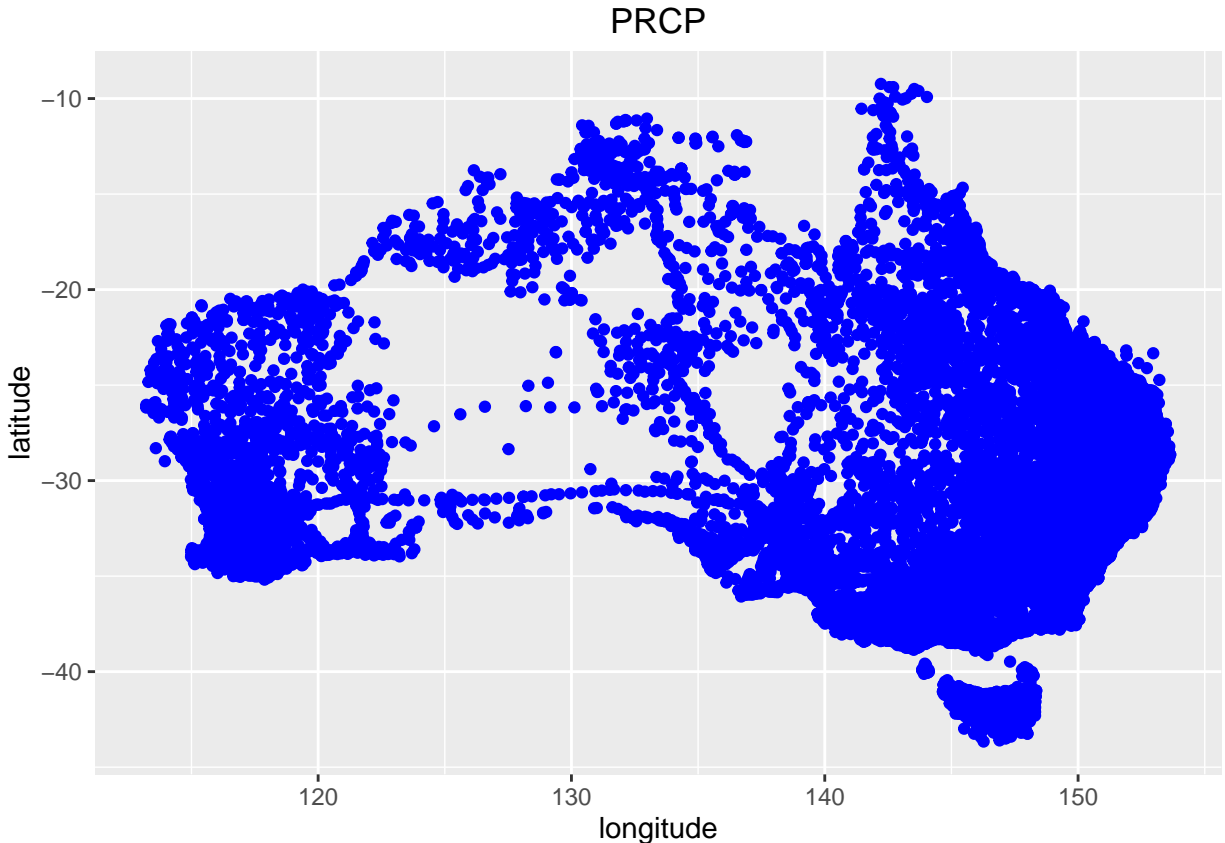
EBONY TO INSERT TABLE

The code to extract the meta data for Australian stations from the GHDN database is shown below.

```
stations <- ghcnd_stations()
list_oz_stations <- grep('^ASN', stations$data$id)
oz_stations <- stations$data[list_oz_stations,]
```

For two of the data types, TMAX and PRCP, the station locations are displayed below:





Note that it is suspected there is some discrepancy within the GHDN station network as compared with raw BoM station data available through the Weather Station Directory (<http://www.bom.gov.au/climate/data/stations/>). These differences include homogenisation processes, different quality controls and in some cases differences in the station availability, with the Bureau of Meteorology holding more complete records.

### To extract data directly from BoM

For users who prefer to directly download station meta data from the BoM; it is possible to pull the meta data for station directly through the BoM ftp server.

```
### Just need the RCurl package

library(RCurl)

### Declare the BOM ftp URL for the directory

url <- "ftp://ftp.bom.gov.au/anon2/home/ncc/metadata/lists_by_element/numeric/"

### No need for a userid or a password

# userpwd <- "yourUser:yourPass"

### Read the filenames that correspond to files stored in the directory

filenames <- getURL(url,
                    # userpwd = userpwd,
```

```

ftp.use.epsv = FALSE,
dirlistonly = TRUE)

### They come as a single string - split this string into a list using
### "\n", then name that list into a vector.

filenames <- unlist(strsplit(filenames, "\n"))

### Check the number of files by state - extract the State id from the
### file names and tabulate.

by.state <- gsub("_", "", substr(filenames, 4, 6))

table(by.state)

### Narrow the files down to to just AUS

(filenames <- grep("AUS", filenames, value = TRUE))

### Draft a function to download the files by name

get.one <- function(fileName) {
  address <- paste(url, fileName, sep = "")
  cat(address, "\n")
  out <- getURL(address, ftp.use.epsv = FALSE)
  return(out)
}

### Try it out

(test <- get.one(filenames[1]))

### It works! Ok grab them all and save them as a list using lapply

metaData <- lapply(filenames, get.one)

### Processing begins - name the list items using the file names

names(metaData) <- filenames

### Back up in case of user error. Don't want to download again.

backup <- metaData

metaData <- backup

### Each downloaded file is a long string. We want to split it into
### lines. Need to split on "\r\n" for some files

metaData <- lapply(metaData,
  function (x) {
    unlist(strsplit(x, "\r\n"))
  })

```

```

### Also need to split on "\n" for some files

metaData <- lapply(metaData,
  function (x) {
    unlist(strsplit(x, "\n"))
  })

### Grab the second lines to see what each file describes

(second.lines <- lapply(metaData, function (x) return (x[2])))

### Grab the third lines to get the column headings

(column.names <- sapply(metaData, function (x) return (x[3])))

### Oh dear! They are not all the same. How many are there?

table(nchar(column.names))

### There are three.

### Now write a suite of functions to scrape different data items Each
### is a variant on the fundamental theme: cut off the top and tail of
### the file, use substring to grab the columns, and then manipulate
### and coerce them as needed.

scrape.site <- function(x)
  return(gsub(" ", "",
    substr(x[-c(1:4, (length(x) - 2):length(x))],
      1, 8)))

scrape.site(metaData[[1]])

scrape.name <- function(x)
  return(gsub(" +$", "",
    substr(x[-c(1:4, (length(x) - 2):length(x))],
      9, 49)))

scrape.name(metaData[[1]])

scrape.latitude <- function(x)
  return(as.numeric(gsub(" ", "",
    substr(x[-c(1:4, (length(x) - 2):length(x))],
      50, 58))))

scrape.latitude(metaData[[1]])

scrape.longitude <- function(x)
  return(as.numeric(gsub(" ", "",
    substr(x[-c(1:4, (length(x) - 2):length(x))],
      60, 68))))

scrape.longitude(metaData[[1]])

```

```

scrape.start <- function(x)
  return(as.POSIXct(strptime(paste("15", substr(x[-c(1:4, (length(x) - 2):length(x)]),
                                     69, 76)), "%d %b %Y")))

scrape.start(metaData[[1]])

scrape.end <- function(x)
  return(as.POSIXct(strptime(paste("15", substr(x[-c(1:4, (length(x) - 2):length(x)]),
                                     78, 85)), "%d %b %Y")))

scrape.end(metaData[[1]])

scrape.years <- function(x)
  return(as.numeric(gsub(" ", "",
                        substr(x[-c(1:4, (length(x) - 2):length(x)]),
                              87, 93))))

scrape.years(metaData[[1]])

scrape.percent <- function(x)
  return(as.numeric(gsub(" ", "",
                        substr(x[-c(1:4, (length(x) - 2):length(x)]),
                              95, 98))))

scrape.percent(metaData[[1]])

scrape.obs <- function(x)
  return(as.numeric(gsub(" ", "",
                        substr(x[-c(1:4, (length(x) - 2):length(x)]),
                              101, 105))))

scrape.obs(metaData[[1]])

### Now! Recall that there are two types of files that we care about;
### one has the Obs column and the other does not. Happily, that's
### close to the end of the columns. But it means that the AWS flag
### is in a different position.

scrape.AWSL <- function(x)
  return(substr(x[-c(1:4, (length(x) - 2):length(x)]),
               109, 109))

scrape.AWSL(metaData[[1]])

scrape.AWSS <- function(x)
  return(substr(x[-c(1:4, (length(x) - 2):length(x)]),
               101, 101))

scrape.AWSS(metaData[["AUS_38.txt"]])

### Hack: it might always be at the end of the string. So we could
### just use one function in that case. But I don't care.

```

```

### One function to bind them

scrape.all <- function(x) {
  if (nchar(x[3]) == 109) {
    return(data.frame(
      site = scrape.site(x),
      name = scrape.name(x),
      latitude = scrape.latitude(x),
      longitude = scrape.longitude(x),
      start = scrape.start(x),
      end = scrape.end(x),
      years = scrape.years(x),
      percent = scrape.percent(x),
      obs = scrape.obs(x),
      AWS = scrape.AWSL(x),
      product = x[2]
    ))
  } else {
    if (nchar(x[3]) == 101) {
      return(data.frame(
        site = scrape.site(x),
        name = scrape.name(x),
        latitude = scrape.latitude(x),
        longitude = scrape.longitude(x),
        start = scrape.start(x),
        end = scrape.end(x),
        years = scrape.years(x),
        percent = scrape.percent(x),
        obs = NA,
        AWS = scrape.AWSS(x),
        product = x[2]
      ))
    } else {
      return(NULL)
    }
  }
}

### Test the one function

head(scrape.all(metaData[[1]]))

### Run the one function across all the little bits.

meta.list <- do.call(rbind, lapply(metaData, scrape.all))

### No warnings. Encouraging! What does it look like?

str(meta.list)

### Some missing values in AWS coded as " "

is.na(meta.list$AWS) <- meta.list$AWS == c(" ")

```

```

meta.list$AWS <- factor(meta.list$AWS)

### How many missing values now?

sapply(meta.list, function(x) sum(is.na(x)))

### I can live with that.

# write.csv(meta.list, file = "~/Desktop/bom-meta.csv", row.names = FALSE)

# save(meta.list, file = "~/Desktop/bom-meta.RData")

```

MIGHT BE NICE add to convert some of my code that does scraping for rainfall directly from BoM to be compatible with the meta data output?

```

#' This function will go to the Bureau of Meteorology Weather Data Portal
#' and scrape the daily rainfall observational record for the input station number.
#' The output is saved in a zip folder under the default destination name that is used
#' by the Bureau of Meteorology.
#'
#' @param siteNum A string input for the station number. This is of width 6 padded with zeroes.
#' @return The sum of \code{0} if there is an error, \code{1} if scrape was successful
#' @examples
#' scrapeBoMData("001000") #succeed in downloading a zip folder
#' scrapeBoMData("000000") #fail in downloading a zip folder

scrapeBoMData <- function(siteNum){
  library(stringr)

  searchstr <- "dailyZippedDataFile"
  bom.site <- "http://www.bom.gov.au"
  url.part1 <- "http://www.bom.gov.au/jsp/ncc/cdio/weatherData/av?p_nccObsCode=136&p_display_type=daily"
  destFile <- paste("IDCJAC0009_", siteNum, "_1800.zip", sep = "")

  #scrape the bureau website for hyperlinks
  url <- paste(url.part1, siteNum, sep = "")
  html <- try(paste(readLines(url), collapse="\n"))
  if (typeof(html) != "character"){
    #Error site not found, no data scraped
    return(0)
  }

  #search for all hyperlinks
  #(this step could be simplified if I knew about regexp)
  hyper.links <- str_match_all(html, "<a href=\"(.*)\"")[[1]][,2]
  if (length(hyper.links) == 0){
    #Error for no hyperlinks found
    return(0)
  }

  #search for download link
  matched = lapply(X = hyper.links, FUN = str_match, pattern = searchstr)
  index = which(!is.na(matched) == 1)
  if(length(index) == 0){

```



```

    # print(paste("No zip file link for:", siteNum))
    return(0)
  } else if(length(index) == 1){
    print(paste("Zip file link found for site number:", siteNum))
    download.link = hyper.links[index]
    #remove amp; from search regexp, and paste bom.site address to create full link
    download.link = paste(bom.site, download.link, sep = "")
    download.link = paste(strsplit(download.link, "amp;")[[1]], collapse = "")
  } else {
    print(paste("Error: More than one download link found", siteNum))
    return(0)
  }
  # desired hyperlink is of the form
  #"http://www.bom.gov.au/jsp/ncc/cdio/weatherData/av?p_display_type=dailyZippedDataFile&p_stn_num=0409

  #download the data (need to add bom site directory, and get rid of amp; from search)
  try(download.file(url = download.link, destFile, mode = "wb"))

  return(1)
}

```

Grab the data for Brunette Downs.

```
(BDs <- grep("BRUNETTE", meta.list$name, value = TRUE))
```

```
## [1] "BRUNETTE DOWNS" "BRUNETTE DOWNS" "BRUNETTE DOWNS" "BRUNETTE DOWNS"
## [5] "BRUNETTE DOWNS" "BRUNETTE DOWNS" "BRUNETTE DOWNS" "BRUNETTE DOWNS"
## [9] "BRUNETTE DOWNS" "BRUNETTE DOWNS" "BRUNETTE DOWNS" "BRUNETTE DOWNS"
## [13] "BRUNETTE DOWNS" "BRUNETTE DOWNS" "BRUNETTE DOWNS" "BRUNETTE DOWNS"
## [17] "BRUNETTE DOWNS" "BRUNETTE DOWNS" "BRUNETTE DOWNS"
```

```
meta.list[grep("BRUNETTE", meta.list$name), c("name", "site")]
```

```
##           name site
## 332  BRUNETTE DOWNS 15085
## 1861 BRUNETTE DOWNS 15085
## 3551 BRUNETTE DOWNS 15085
## 5424 BRUNETTE DOWNS 15085
## 7050 BRUNETTE DOWNS 15085
## 7553 BRUNETTE DOWNS 15085
## 8094 BRUNETTE DOWNS 15085
## 8419 BRUNETTE DOWNS 15085
## 8761 BRUNETTE DOWNS 15085
## 14112 BRUNETTE DOWNS 15085
## 32429 BRUNETTE DOWNS 15085
## 47818 BRUNETTE DOWNS 15085
## 52594 BRUNETTE DOWNS 15085
## 68438 BRUNETTE DOWNS 15085
## 76787 BRUNETTE DOWNS 15085
## 79731 BRUNETTE DOWNS 15085
## 81719 BRUNETTE DOWNS 15085
## 83646 BRUNETTE DOWNS 15085
## 85300 BRUNETTE DOWNS 15085
```

```
scrapeBoMData("015085")
```

```
## [1] "Zip file link found for site number: 015085"
```

```
## [1] 1
```

### How to get the data:

- how to search for a location (radius search)
- date range

### How to read the data

- give a plotted example (rain/temp)

### Quality control

It is important to note that although the station data is subject to some quality controls by BoM, not all data has been quality assured. This is particularly relevant for older observational records.

Some of the common sources of erroneous observations within the records are:

- Changes to station location
- Changes in observer behaviour
- Changes in instrumentation
- Rounding errors
- Shifts in record
- Heat island effect
- Multiday accumulations errors (PRCP)
- Spurious Zeroes (PRCP)
- Undercatch (PRCP)

Contributions of code to assess station record quality would be desirable. Currently there is no one size fits all approach.

## Using Australian Water Availability Project (AWAP) Data

About AWAP, from CSIRO, <http://www.csiro.au/awap/>,

"The aim of the Australian Water Availability Project (AWAP) is to monitor the state and trend of the terrestrial water balance of the Australian continent, using model-data fusion methods to combine both measurements and modelling. The project determines the past history and present state of soil moisture and all water fluxes contributing to changes in soil moisture (rainfall, transpiration, soil evaporation, surface runoff and deep drainage), across the entire Australian continent at a spatial resolution of 5 km. Using the same basic framework, the project provides soil moistures and water fluxes over the Australian continent in three forms: (1) weekly/monthly near-real-time reporting, (2) monthly/annual historical time series from 1900, and (3) monthly climatologies (on request)."

### awaptools

An R package, `awaptools`, created by Ivan Hanigan, [ivan.hanigan@gmail.com](mailto:ivan.hanigan@gmail.com), J. Guillaume, F. Markham, G. Williamson and I. Szarka is used here to retrieve the AWAP grid files. This package allows for automated downloading of AWAP data. Required are start date, end date and the weather variable(s) of interest.

```
if(!require("awaptools"))
{
  library(devtools)
  install_github("swish-climate-impact-assessment/awaptools")
}
```

```
## Loading required package: awaptools
```

```
## Loading required package: raster
```

```
## Loading required package: sp
```

```
library(awaptools)
```

```
vars <- c("maxave", "minave", "totals")
```

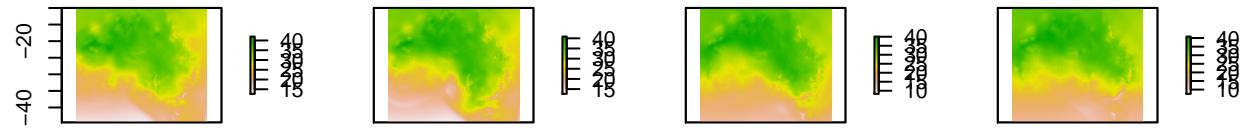
```
for (measure in vars) {
  get_away_data(start = "2016-04-21", end = "2016-05-09", measure = measure)
}
```

```
# load some, plot to check, and demonstrate conversion to GeoTIF
```

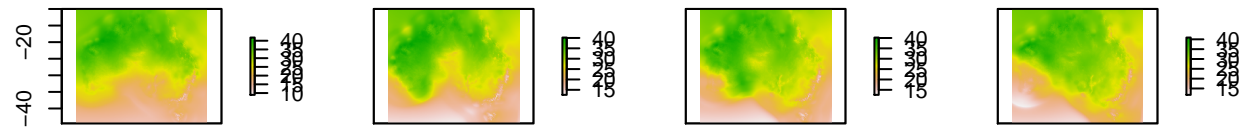
```
cfiles <- stack(dir(pattern="grid$"))
```

```
plot(cfiles) # plot to look at
```

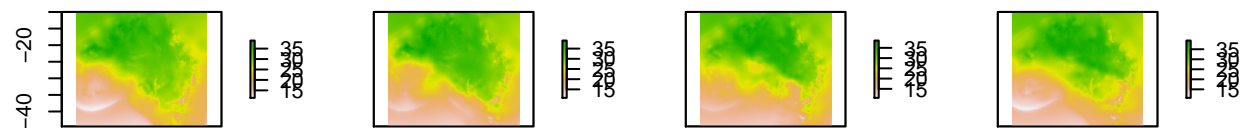
**kave\_2016041920160419 kave\_2016042020160420 kave\_2016042120160421 kave\_2016042220160422**



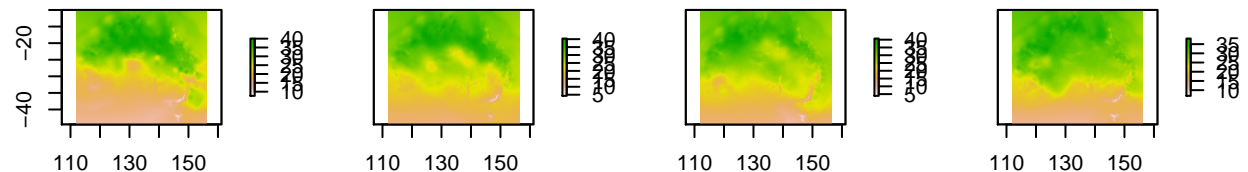
**kave\_2016042320160423 kave\_2016042420160424 kave\_2016042520160425 kave\_2016042620160426**



**kave\_2016042720160427 kave\_2016042820160428 kave\_2016042920160429 kave\_2016043020160430**



**kave\_2016050120160501 kave\_2016050220160502 kave\_2016050320160503 kave\_2016050420160504**



```
writeRaster(cfiles, filename = "AWAP", format = "GTiff",
  bylayer = TRUE, overwrite = TRUE, suffix = "names")
```

For additional processing codes see the short demo in <https://github.com/swish-climate-impact-assessment/awaptools/blob/master/README.md>

To start cleaning up the grid files for mapping or looking at specific areas, download Australia, level 2 data from the GADM database of Global Administrative Areas.

Using the GADM data, crop the grid files and mask using the administrative boundaries to map the area of interest.

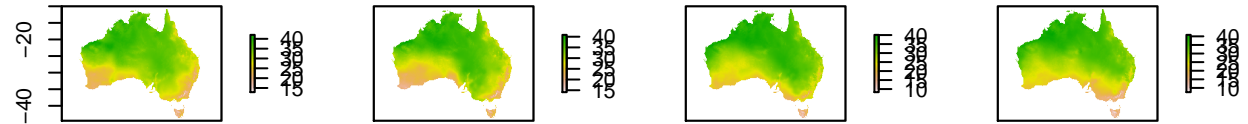
```
library(raster)
AUS <- getData(name = "GADM", download = TRUE, country = "AUS", level = 2)

# Subset Australia data to QLD
QLD <- AUS[AUS@data$NAME_1 == "Queensland", ]
```

Crop, mask and plot the raster stack using Australia

```
AUS_data <- mask(cfiles, AUS)
plot(AUS_data)
```

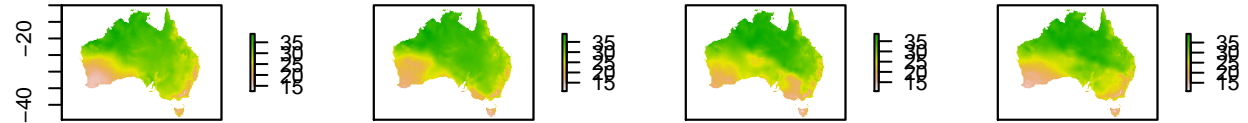
**kave\_2016041920160419 kave\_2016042020160420 kave\_2016042120160421 kave\_2016042220160422**



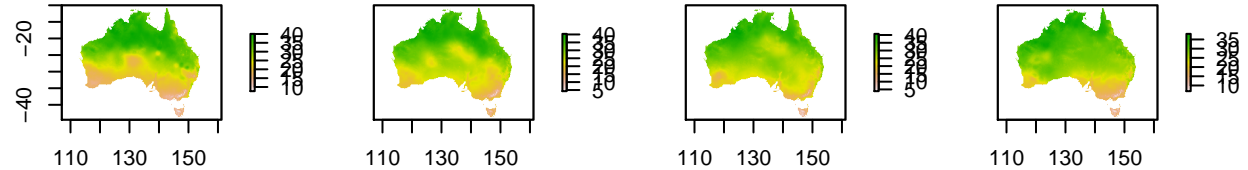
**kave\_2016042320160423 kave\_2016042420160424 kave\_2016042520160425 kave\_2016042620160426**



**kave\_2016042720160427 kave\_2016042820160428 kave\_2016042920160429 kave\_2016043020160430**



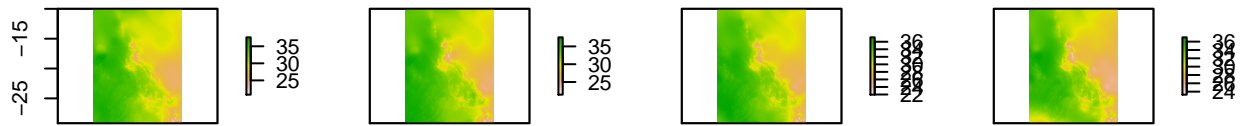
**kave\_2016050120160501 kave\_2016050220160502 kave\_2016050320160503 kave\_2016050420160504**



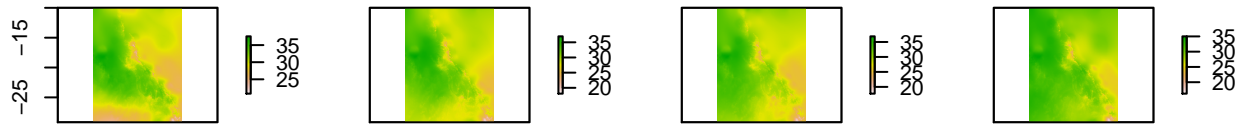
Crop, plot and then mask and plot the raster stack using QLD to see the difference between cropping and masking.

```
QLD_data <- crop(cfiles, QLD)
plot(QLD_data)
```

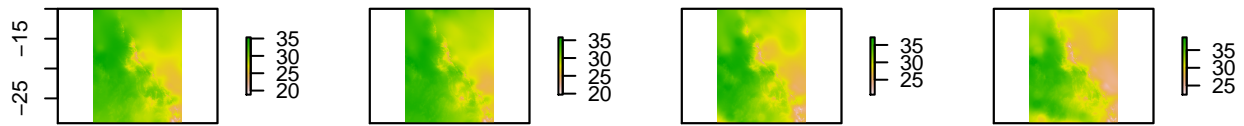
**kave\_2016041920160419 kave\_2016042020160420 kave\_2016042120160421 kave\_2016042220160422**



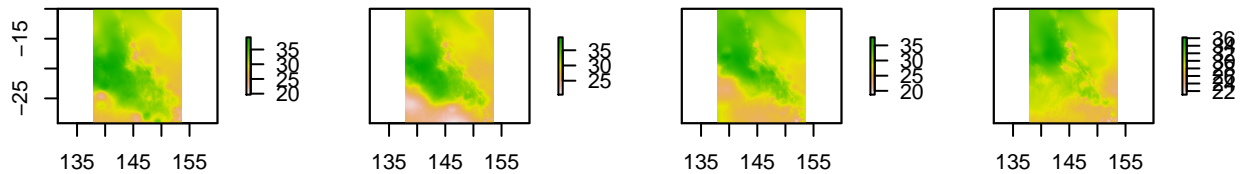
**kave\_2016042320160423 kave\_2016042420160424 kave\_2016042520160425 kave\_2016042620160426**



**kave\_2016042720160427 kave\_2016042820160428 kave\_2016042920160429 kave\_2016043020160430**

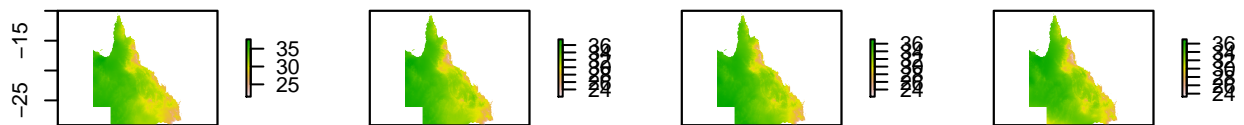


**kave\_2016050120160501 kave\_2016050220160502 kave\_2016050320160503 kave\_2016050420160504**

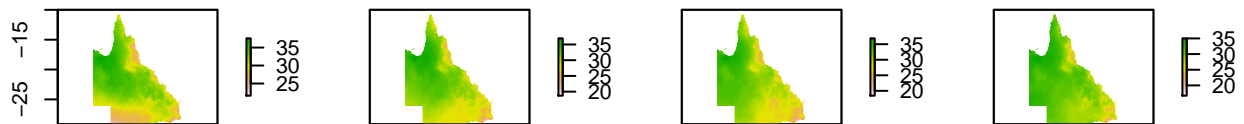


```
QLD_data <- mask(QLD_data, QLD)
plot(QLD_data)
```

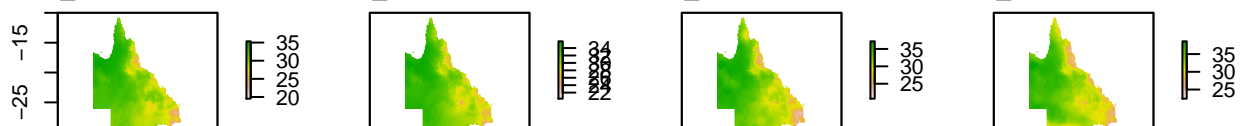
**κave\_2016041920160419 κave\_2016042020160420 κave\_2016042120160421 κave\_2016042220160422**



**κave\_2016042320160423 κave\_2016042420160424 κave\_2016042520160425 κave\_2016042620160426**



**κave\_2016042720160427 κave\_2016042820160428 κave\_2016042920160429 κave\_2016043020160430**



**κave\_2016050120160501 κave\_2016050220160502 κave\_2016050320160503 κave\_2016050420160504**

