SWMP time series analysis cookbook

Marcus W. Beck, beck.marcus@epa.gov, Todd D. O'Brien, todd.obrien@noaa.gov

Nov. 17, 2014

Contents

Introduction

This document provides a collection of instructions and scripts that perform specific tasks for working with SWMP data in R. It provides scripts that use functions from the SWMPr package, as well as many others available in the base or contributed packages available in R. This cookbook is meant to provide workshop participants with a 'go-to' document that builds on content covered in the workshop. Much of the content can be a considered a basic approach to time series and we encourage the exploration of alternative, more specific methods that are available in R. Please see the CRAN task view on time series for an idea of what is possible. The data used in these scripts have been included with the SWMPr package installation or can be downloaded from the workshop website. Please feel free to contact the instructors with questions regarding this cookbook or general questions about time series analysis in R. We encourage any feedback regarding these documents or the content of the workshop.

Workshop website: http://copepod.org/nerrs-swmp-workshop/

Basics of R and RStudio

What to do when you start a new session

You will need to follow a few simple steps to use data and functions in R if you are starting a new session. Here is a basic workflow of these steps.

- Open a new or saved script that you will use to type code (.r file extension). This is under the File menu on the top for both the basic R install and RStudio.
- Load R packages that you will use. The functions in a package will not be avialable unless the package is loaded. The package must also be previously installed. You can put the packages you are using at the top of your script.
- Set the working directory. This is where R will load and save files.
- Load any workspace that you were using previously. This an R specific file format that saves any and all objects that are in a workspace. Ideally, you will have saved a workspace from your previous session.

Here is a sample script of this workflow.

```
# my startup script

# load any packages
library(SWMPr)
library(ggplot2)

# set the working directory
setwd('C:/my_files')

# load your previous workspace if you saved one
# this must be in the working directory
load(file = 'my_workspace.RData')

# check what you loaded
ls()
```

Data of specific formats can also be loaded in R. Flat text files or comma-separated files are commonly used, although R can import many other types. Use these functions to import data. Don't forget to assign your data to an object.

```
# import data and assign
# data are in the working directory, or use a full path

# import a csv file
dat <- read.csv('my_data.csv', header = T)

# importa a text file, separated by commas
dat <- read.table('my_data.txt', header = T, sep = ',')
```

What to do when you close a session

Make sure you save your work when you close a session! Save the script you're working with using the menu at the top. You will also want to save your data. Just as data of different types can be imported, data can

also be saved in different formats. You will either want to save the whole workspace or individual parts (e.g., a data frame).

```
# save the whole workspace as a .RData file
save(list = ls(), file = 'my_workspace.RData')

# save one object as a .RData file
save(dat, file = 'my_data.RData')

# save as .csv
write.csv(dat, 'my_data.csv')

# save as text file
write.table(dat, 'my_data.txt', sep = ',', row.names = F, quote = F)
```

Installing packages

R installs and loads packages from its library on your computer. You can see where your library is with `.libPaths()`. Packages that you install from CRAN or elsewhere will go in this library. R will also look here when you load a library. The standard location for downloading packages are on CRAN, although you can also download packages on Github or BioConductor. In the latter case, you will have to first download and load the devtools package off CRAN. Again, you will have to load a package every time you start R if you want to use its functions. You should only have to download a package once, unless you want to install the latest version. Here are some basics to installing and loading a package.

```
# install a package from CRAN
install.packages('ggplot2')

# install packages from Github
install.packages('devtools')
library(devtools)
install_github('fawda123/SWMPr')
library(SWMPr)
```

Keyboard shortcuts

You can view all the keyboard shortcuts in RStudio by clicking on Help, then keyboard shortcuts on the top menu. Here are some common shortcuts for Windows/Linux (* denotes RSTudio only).

CTRL+ENTER	run selection *
CTRL+R	run selection
CTRL+L	clear console
CTRL+A	select all
CTRL+C	copy
CTRL+X	cut
CTRL+V	paste
CTRL+O	open document
CTRL+S	save document
CTRL+1	switch to source *
CTRL+2	switch to console *
ESC	stop current execution
HOME	skip to beginning of line
END	skip to end of line

Getting help

If all else fails, a Google search will usually point you in the right direction. All of the documentation that comes with R and its packages are available online. A quick search for a function or package will lead you to the documentation. You can also access the help files on your computer in R.

```
# access a help file for a function
help(mean)

# or do this
?mean

# run the examples in the help file
example(mean)
```

Using the SWMPr package

A quick workflow for retrieving and organizing

Each of the functions in the SWMPr package usually have default values for the arguments to make their use easier. However, be cautious since the default values may not be applicable for your specific dataset. Always consult the help documentation to determine the best options for importing and organizing your data. Also see the SWMPr tutorial for more details on each function. The following examples provide generic workflows for importing and organizing data that you have downloaded from CDMO. The `import_local` function is meant to work with data from the Zip Downloads service from the CDMO. For the purpose of this document, data included with the SWMPr installation are used. Also note that you must assign the data to an existing or new object to save results returned from functions.

Single file

This shows how to import and organize a single file that you downloaded.

```
# import data for apaebmet that you downloaded

# this is an example path with the csv files, change as needed
path <- 'C:/my_path/'

# import, qaqc, subset to remove empty columns
dat <- import_local(path, 'apaebmet'))
dat <- subset(qaqc(dat), rem_cols = T)
```

For the next example, the `import_local` function is used to load data included with the SWMPr distribution. These date are only useful for demonstrations. To use, set the path variable using the `system.file` command shown below (you can see this full path by executing `path` at the command line). Execute both lines to import the data.

```
# import data for apaebmet that comes with SWMPr

# this is the path for csv example files
path <- system.file('zip_ex', package = 'SWMPr')
```

```
# import, qaqc, subset to remove empty columns
dat <- import_local(path, 'apaebmet'))
dat <- subset(qaqc(dat), rem_cols = T)
```

Multiple files

This shows how to import and organize multiple files that you downloaded.

```
# import data for multiple stations

# this is an example path with the csv files, change as needed
path <- 'C:/my_path/'

# import, combine, qaqc, subset to remove empty columns
wq_dat <- import_local(path, 'apacpwq')
nut_dat <- import_local(path, 'apacpnut')
met_dat <- import_local(path, 'apaebmet')
dat <- comb(wq_dat, nut_dat, met_dat)
dat <- subset(qaqc(dat), rem_cols = T)
```

The remainder of this cookcook will use data from muliple files that were imported and organized in the following script. These are the example data included with the SWMPr package.

```
# import data for multiple stations that comes with swmpr

# this is the path for csv example files
path <- system.file('zip_ex', package = 'SWMPr')

# import, combine, qaqc, subset to remove empty columns
wq_dat <- import_local(path, 'apacpwq')
nut_dat <- import_local(path, 'apacpnut')
met_dat <- import_local(path, 'apaebmet')
dat <- comb(wq_dat, nut_dat, met_dat)
dat <- subset(qaqc(dat), rem_cols = T)
```

Viewing the attributes and subsets of the data

The imported SWMPr data have descriptive attributes.

```
# names of all the attributes
names(attributes(dat))

# retrieving attributes
attr(dat, 'station')
attr(dat, 'parameters')
attr(dat, 'qaqc_cols')
attr(dat, 'date_rng')
attr(dat, 'timezone')
```

Viewing the whole dataset is often impractical. Here are some functions for viewing subsets.

```r
# View the first 1000 rows
View(dat)

# first six rows, last six rows
head(dat)
tail(dat)

# first n rows, last n rows
head(dat, 30)
tail(dat, 30)

# single variables
dat$do_mgl
dat[, 'do_mgl']

# column 1
dat[, 1]

# row 1
dat[1, ]

# row 1, column 1
dat[1, 1]

# dimenions
dim(dat)
nrow(dat)
ncol(dat)
length(dat)
```

Simple numerical summaries

Numerical summaries of the data can be obtained for the entire dataset or single variables. In some cases, you will have to explicitly specify how missing data are handled. For example, the default behavior of many functions is to return `NA` if missing value are in the data.

```r
# whole dataset
summary(dat)

# individual variables
summary(dat$do_mgl)

# mean, range, var, etc.
# note use of na.rm
mean(dat$do_mgl, na.rm = T)
range(dat$do_mgl, na.rm = T)
var(dat$do_mgl, na.rm = T)
sd(dat$do_mgl, na.rm = T)
min(dat$do_mgl, na.rm = T)
max(dat$do_mgl, na.rm = T)

# how many missing values?
sum(is.na(dat$do_mgl))
```

Aggregating or reducing data volume

The SWMPr package provides several function for reducing the volume of data. This is useful for not only making the data more manageable, but also providing summary statistics that describe trends.

The subset function lets you select columns or rows of interest.

```
# select two parameters from dat
subset(dat, select = c('rh', 'bp'))

# subset records greater than or equal to a date
subset(dat, subset = '2013-01-01 0:00', operator = '>=')

# subset records within a date range
subset(dat, subset = c('2012-07-01 6:00', '2012-08-01 18:15'))

# subset records within a date range, select two parameters
subset(dat, subset = c('2012-07-01 6:00', '2012-08-01 18:15'),
  select = c('atemp', 'temp'))

# remove rows/columns that do not contain data
subset(dat, rem_rows = T, rem_cols = T)
```

The setstep lets you change the time step of your data.

```
# change to two hour steps
setstep(dat, timestep = 120)
```

Aggregate is used to summarize data by set periods of observations.

```
# aggregate the data by mean
aggregate(dat, by = 'quarters', params = c('do_mgl'))

# change the default function, aggregate by months
fun_in <- function(x) var(x, na.rm = T)
aggregate(dat, by = 'months', FUN = fun_in, params = c('do_mgl'))
```

The smoother function is a simple moving window average that can be used to reduce the variance in a parameter. This may help better characterize a trend.

```
# subset the data to smooth
sub_dat <- subset(dat, subset = c('2012-07-09 00:00', '2012-07-24 00:00'))

# smooth
smooth_dat <- smoother(sub_dat, window = 50, params = 'do_mgl')
```

Dealing with missing data

Missing data can be handled several ways depending on the needs of an analysis.

```r
# subset for the example
sub_dat <- subset(dat, subset = c('2013-01-22 00:00', '2013-01-26 00:00'))

## replace with mean

# a temporary object so we don't overwrite wq_dat
wq_tmp <- sub_dat

# replace with the mean
wq_tmp <- wq_dat
wq_tmp[is.na(wq_tmp$do_mgl), 'do_mgl'] <- mean(wq_tmp$do_mgl, na.rm = T)

## remove missing values

# using subset function
wq_tmp <- subset(wq_tmp, rem_rows = T)

# use na.omit
wq_tmp <- na.omit(wq_tmp)

## linearly interpolate using na.approx

# interpolate, maxgap of 10 records
wq_tmp <- na.approx(sub_dat, params = 'do_mgl', maxgap = 10)

# interpolate maxgap of 30 records
wq_tmp <- na.approx(sub_dat, params = 'do_mgl', maxgap = 30)
```

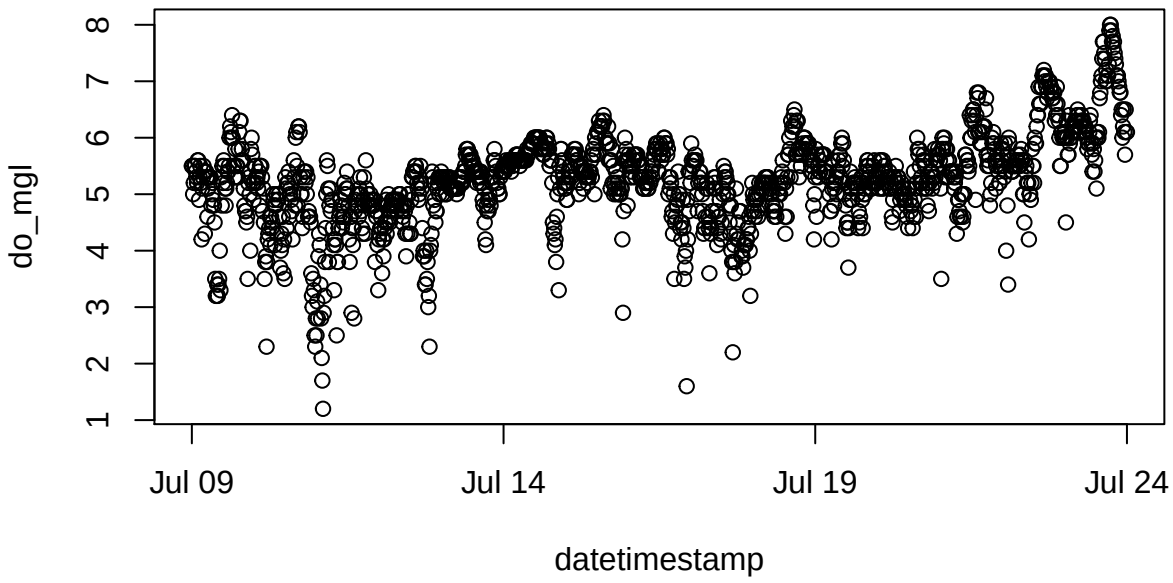
Graphics

Graphics are virtually limitless in R. The base installation provides several functions that will suit most of your needs. The ggplot2 package is also very useful for plotting multiple variables. The following are simple scripts showing how various plots can be made using base graphics and ggplot2. We have also uploaded some graphics scripts and instructions to the website provided by Kimberly Cressman, SWMP coordinator at Grand Bay.

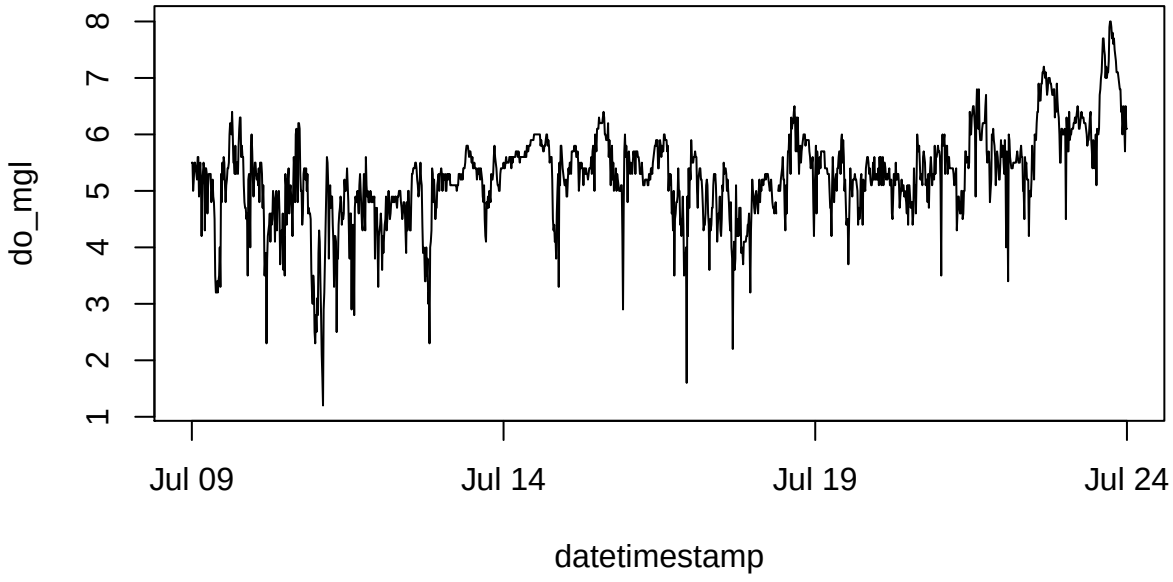
Base graphics

Simple time series plots.

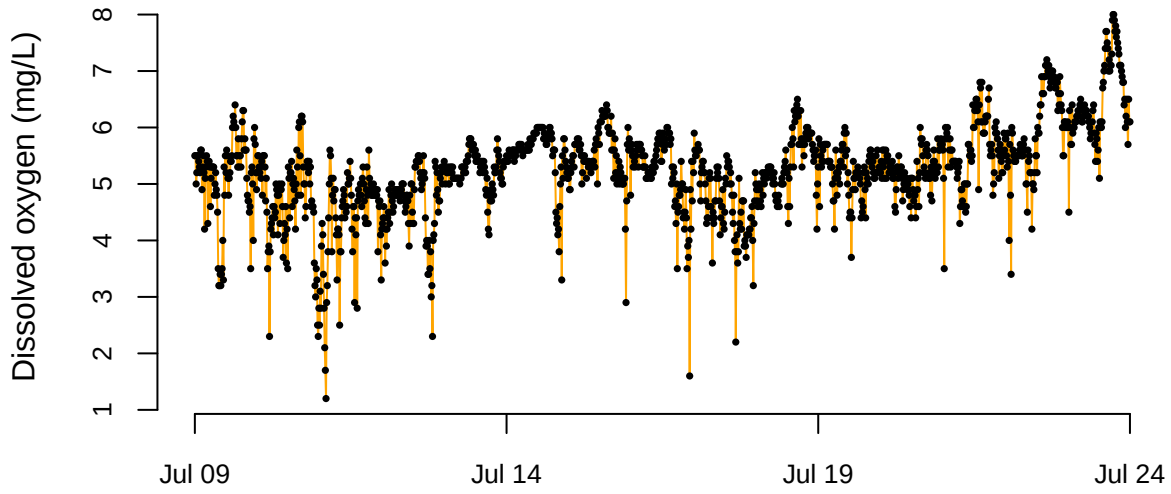
```r
# subset dat for the example
sub_dat <-  subset(dat, subset = c('2012-07-09 00:00', '2012-07-24 00:00'))

# points
plot(do_mgl ~ datetimestamp, data = sub_dat)
```




```
# lines
plot(do_mgl ~ datetimestamp, data = sub_dat, type = 'l')
```

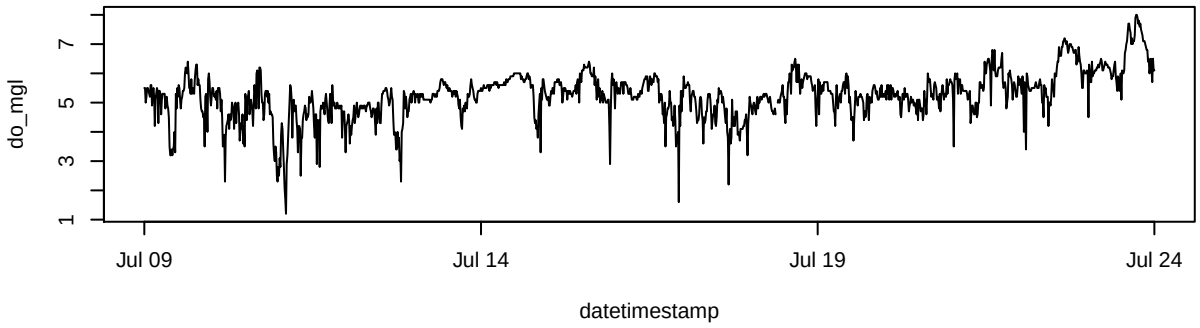


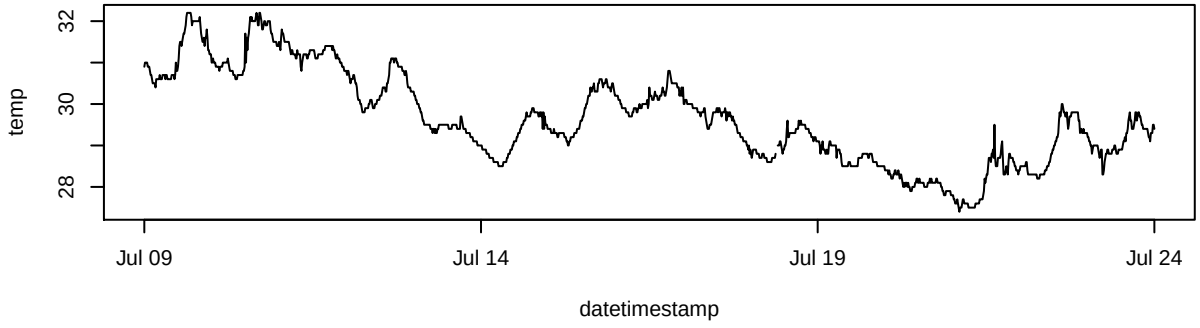
```
# changing the default arguments, lines and add points to graph
plot(do_mgl ~ datetimestamp, data = sub_dat, type = 'l', col =
    'orange', xlab = '', ylab = 'Dissolved oxygen (mg/L)',
  cex.axis = 0.8, bty = 'n')
points(sub_dat$datetimestamp, sub_dat$do_mgl, pch = 16, cex = 0.5)
```

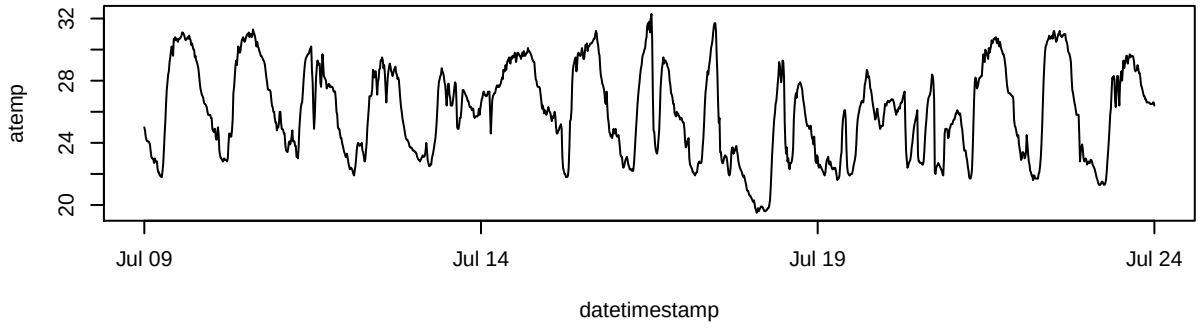


Multiple plots in the same window.

```
# it is often useful to open a new plot window, uncomment and run the next line
# windows()

# create three plots
# 3 rows one column in window, use mfrow argument with par
par(mfrow = c(3, 1))
plot(do_mgl ~ datetimestamp, data = sub_dat, type = 'l')
plot(temp ~ datetimestamp, data = sub_dat, type = 'l')
plot(atemp ~ datetimestamp, data = sub_dat, type = 'l')
```

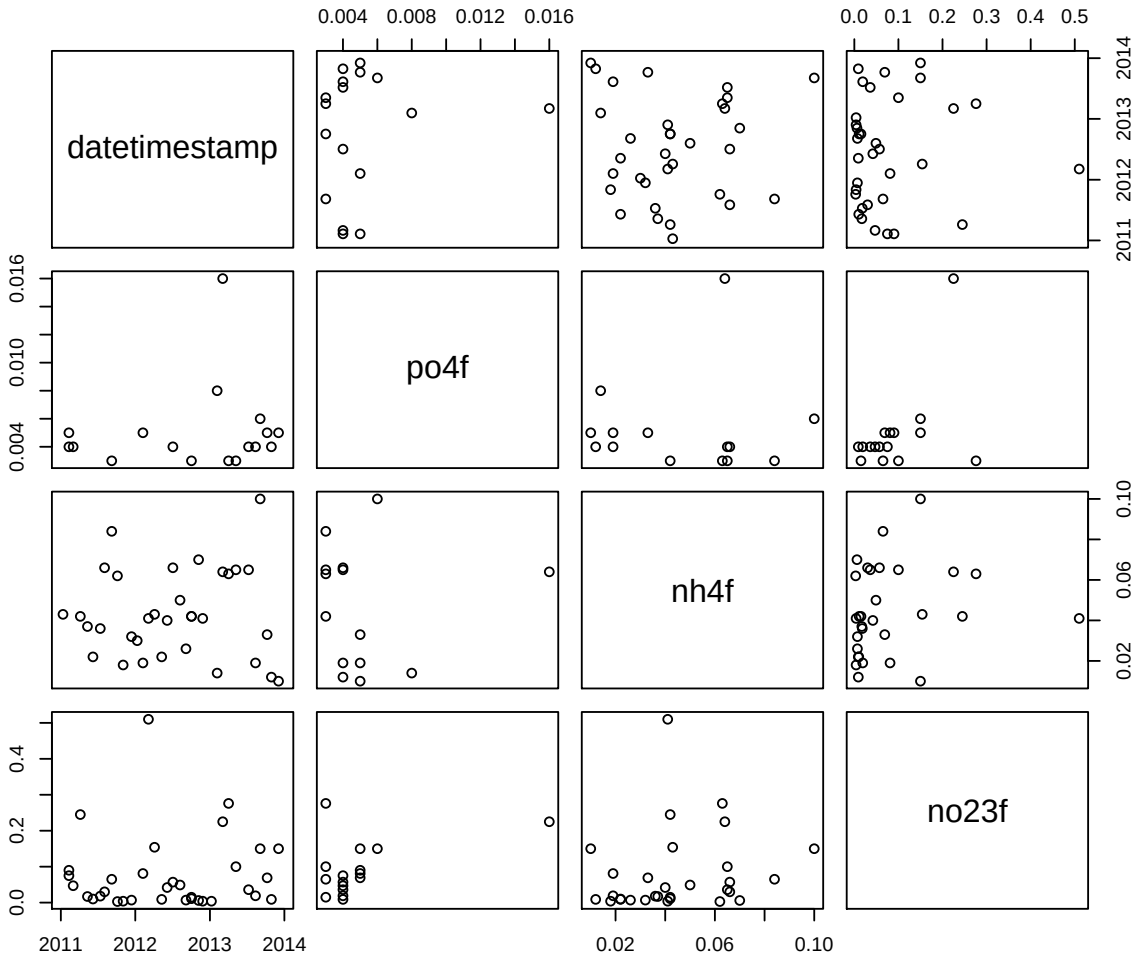




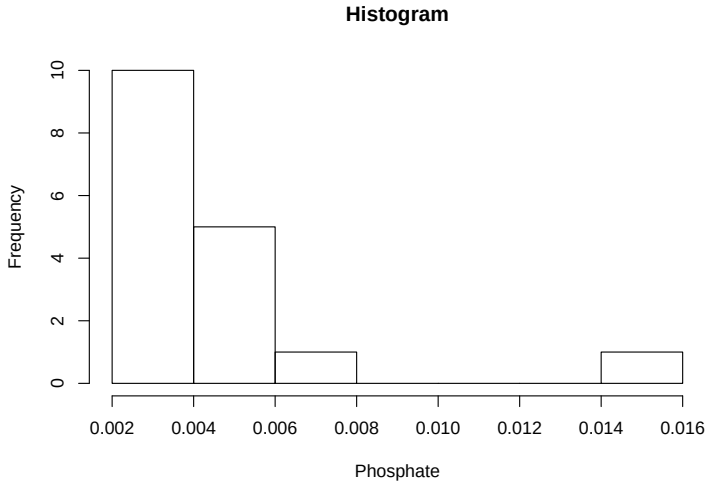


Diagnostics plots.

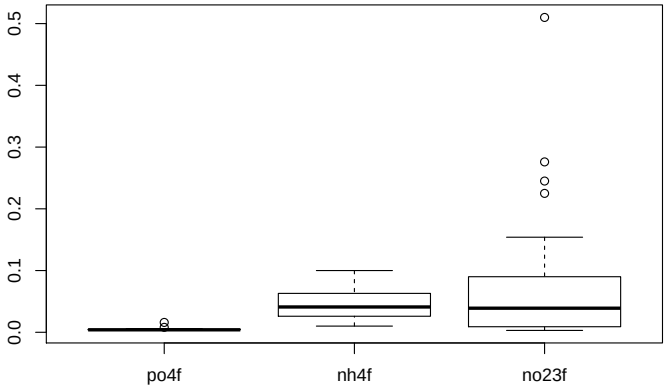
```
# subset for the example
sub_dat <- subset(dat, select = c('po4f', 'nh4f', 'no23f'))

## pairs plot
# a pairs plot is a matrix of bivariate scatterplots
# caution, this plot may take a while to load if using the whole time series
pairs(sub_dat)
```




```
## histograms
hist(sub_dat$po4f, xlab = 'Phosphate', main = 'Histogram')
```



```
## boxplots
# remove datetimestamp
to_plo <- data.frame(sub_dat)[, -1]
boxplot(to_plo)
```



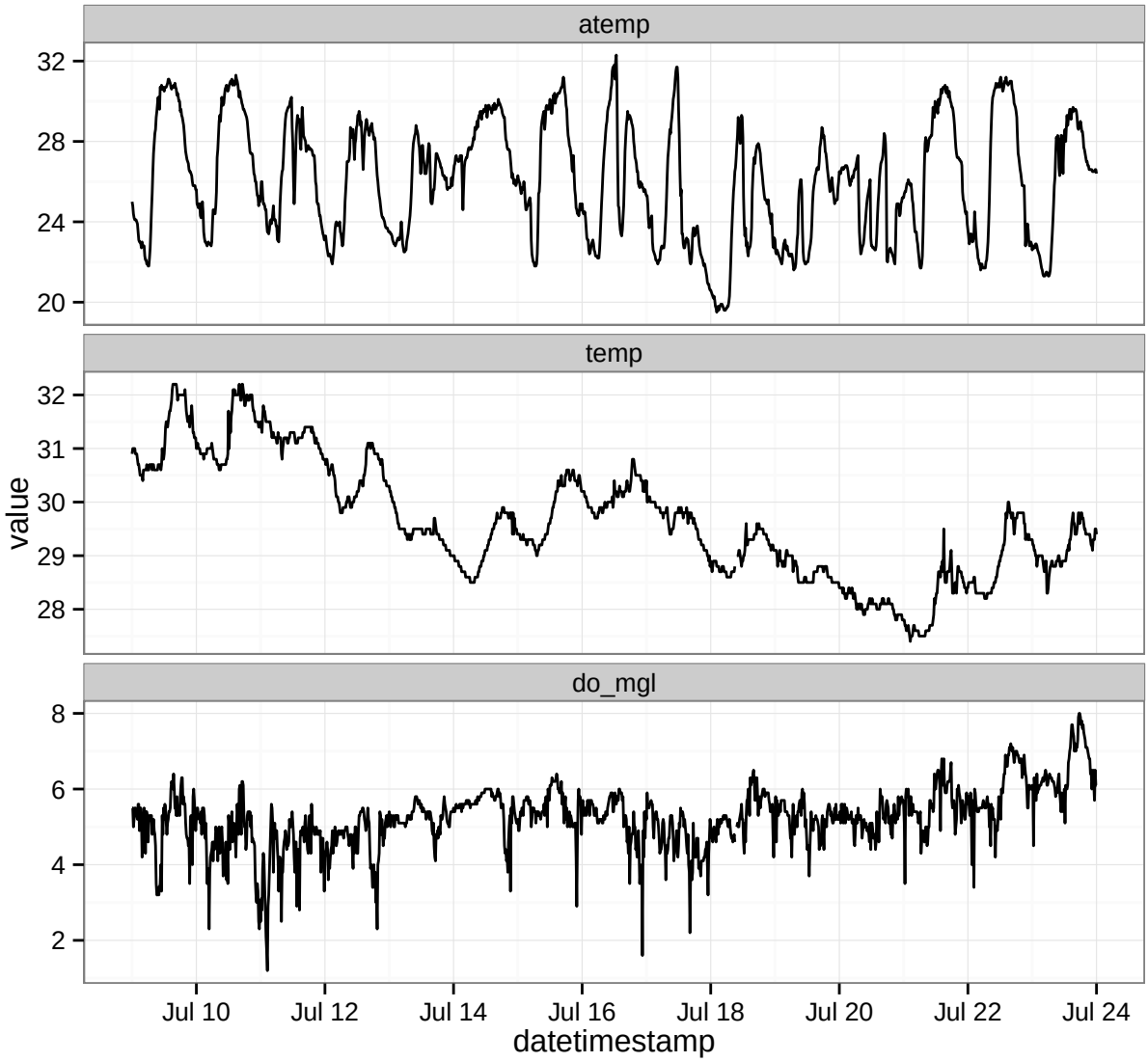
Plotting with ggplot2

The ggplot2 package offers many plotting features that are incredibly useful . We presents some examples here to illustrate plotting multiple variables on the same scale.

```
# install ggplot2, should be installed with SWMPr
install.packages('ggplot2')

# install the reshape2 package for formatting data to plot
install.packages('reshape2')
```

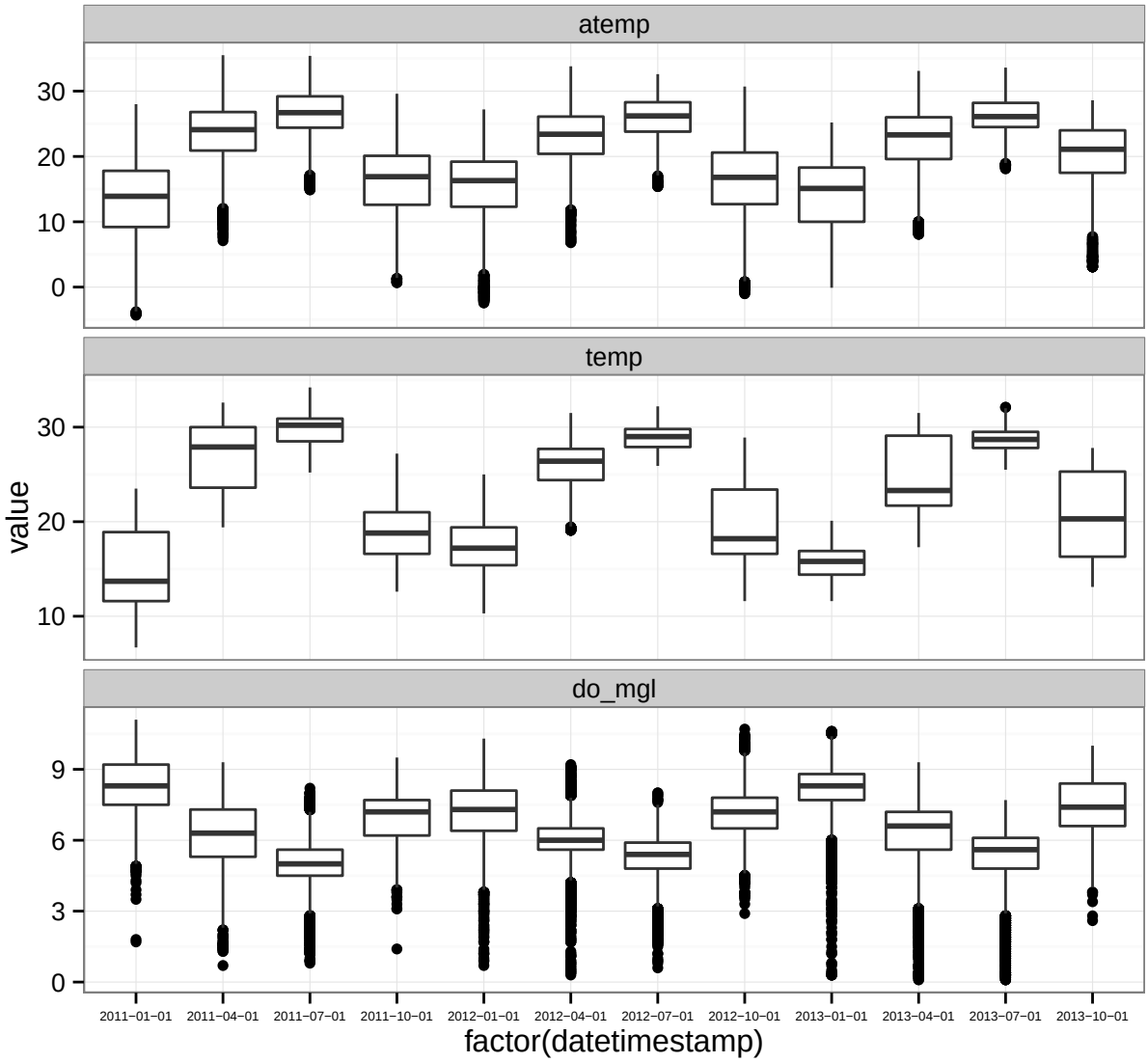
A multivariate time series plot with ggplot2.

```
# subset for the example
sub_dat <-  subset(dat, subset = c('2012-07-09 00:00', '2012-07-24 00:00'),
  select = c('do_mgl', 'atemp', 'temp'))

# load relevant packages
library(ggplot2)
library(reshape2)

# prepare data for plotting
to_plo <- melt(sub_dat, id.var = 'datetimestamp')

# plot
p <- ggplot(to_plo, aes(x = datetimestamp, y = value)) +
  geom_line() +
  facet_wrap(~ variable, ncol = 1, scales = 'free_y') +
  theme_bw()
p
```



Using aggregation and ggplot2 to plot multiple variables.

```
# aggregate by season
agg_dat <- aggregate(dat, by = 'quarters',
  params = c('do_mgl', 'temp', 'atemp'), aggs_out = T)

# prepare data for plotting
to_plo <- melt(agg_dat, id.var = 'datetimestamp')

# plot
p <- ggplot(to_plo, aes(x = factor(datetimestamp), y = value)) +
  geom_boxplot() +
  facet_wrap(~ variable, ncol = 1, scales = 'free_y') +
  theme_bw() +
  theme(axis.text.x = element_text(size = 5))
p
```



Maps

The `map_reserve` function uses the ggmap package to create a plot of the stations at a reserve. The ggmap package should load automatically when you install SWMPr. You will have to play with the `zoom` argument as the spatial extent varies at each reserve.

```
map_reserve('jac', zoom = 11)
```

Saving graphics

A graphic can be saved in different formats using the file menu in the plot window. You can also save graphics using specific commands that are run in the console.

```
# save a pdf graphic, will go to the working directory
# height, width in inches
pdf('my_plot.pdf', height = 6, width = 6)

# ggplot graphic from earlier
p

# turn off the graphics device
dev.off()

# save a png
```

```r
# height, width in pixels, res in ppi
png('my_plot.png', height = 2400, width = 2400, res = 400)

# ggplot graphic from earlier
p

# turn off the graphics device
dev.off()
```