

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 | 1 |
| Basics of R and RStudio | 2 |
| What to do when you start a new session | 2 |
| What to do when you close a session | 3 |
| Installing packages | 3 |
| Keyboard shortcuts | 3 |
| Getting help | 4 |
| Using the SWMP_r package | 4 |
| A quick workflow for retrieving and organizing | 4 |
| Viewing the attributes and subsets of the data | 5 |
| Simple numerical summaries | 6 |
| Aggregating or reducing data volume | 7 |
| Dealing with missing data | 8 |
| Graphics | 8 |
| Base graphics | 8 |
| Plotting with ggplot2 | 14 |
| Maps | 16 |
| Saving graphics | 17 |

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 SWMP_r package, as well as many others available in the base or contributed R packages. This cookbook is meant to provide participants with a ‘go-to’ document that builds on content covered in the workshop. Much of the content can be considered a basic approach to time series analysis 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 SWMP_r package installation, excluding the first few scripts that are provided only as templates for your own use. 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 available unless the package is loaded. The package must also be previously installed (see below for package installation). 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 (.RData) that contains any and all objects that were in a previous 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 installed packages
library(SWMPPr)
library(ggplot2)

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

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

# check what you loaded
ls()
```

Other data formats can also be imported or loaded into R. Flat text files or comma-separated files are commonly used, although R can import many other types. Use the following functions to import text or csv 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 File 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` as a `.csv` file).

```
# save the whole workspace as a .RData file
# will be saved in the working directory
save(list = ls(), file = 'my_workspace.RData')

# save one object (e.g., dat) 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 package to see if it is installed. Most packages will be downloaded from CRAN, although you can also download packages from 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 previously installed 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 re-install to get the latest version. Here are some basics for 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/SWMPPr')
library(SWMPPr)
```

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 SWMPPr package

A quick workflow for retrieving and organizing

Each of the functions in the SWMPPr 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 [SWMPPr 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. Data included with the SWMPPr installation are used for the remainder of this document, starting with the second example under ‘multiple files’. Also note that you must assign the data to an object to save results returned from functions (i.e., using `<-` or `=`).

Single file

This shows how to import and organize data for a single station that you downloaded.

```
# import data for apaebmet that you downloaded

# this is an example path with hypothetical csv files
# change as needed for actual data
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 SWMPPr distribution. These data 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 after running the example.

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

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

# 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 stations that you downloaded.

```
# import data for multiple stations

# this is an example path with hypothetical csv files
# change as needed for actual data
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 cookbook will use data from multiple files that were imported and organized in the following script. These are the example data included with the SWMPPr package.

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

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

# 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 SWMPPr 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.

```
# 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]

# dimensions
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 values are in the data. See the examples for how to change this behavior.

```
# 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 SWMP_r package provides several functions for reducing the volume of data. This is useful for not only making the data more manageable, but also providing summary statistics that potentially 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 function 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 DO by quarters using the mean (default)
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 DO using a window of 50 observations
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. The following example shows how to replace missing data with the mean value of the whole time series, omit all missing values, or interpolate using `na.approx`. This is not an exhaustive list of options.

```
# 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

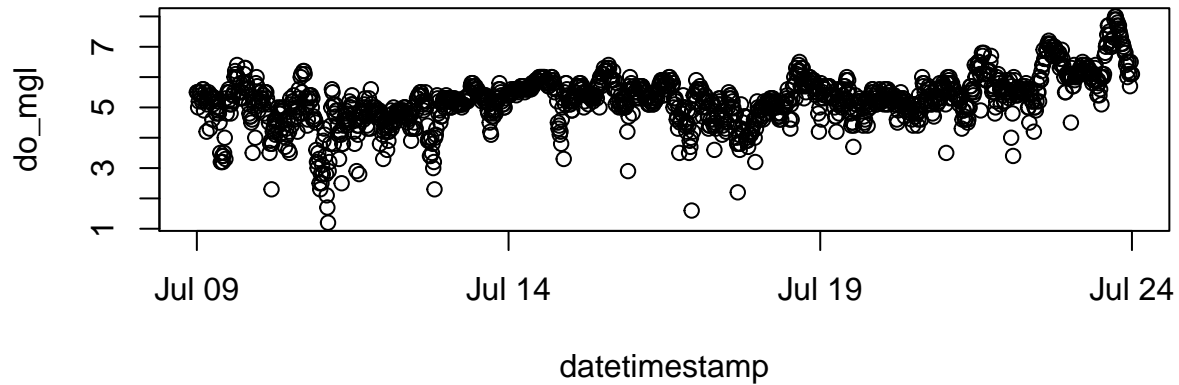
Graphics are virtually limitless in R. The base installation provides several functions that will suit most of your needs. The `ggplot2` package is also very useful for plotting multiple variables, among other things. The following are simple scripts showing how various plots can be made using base graphics 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

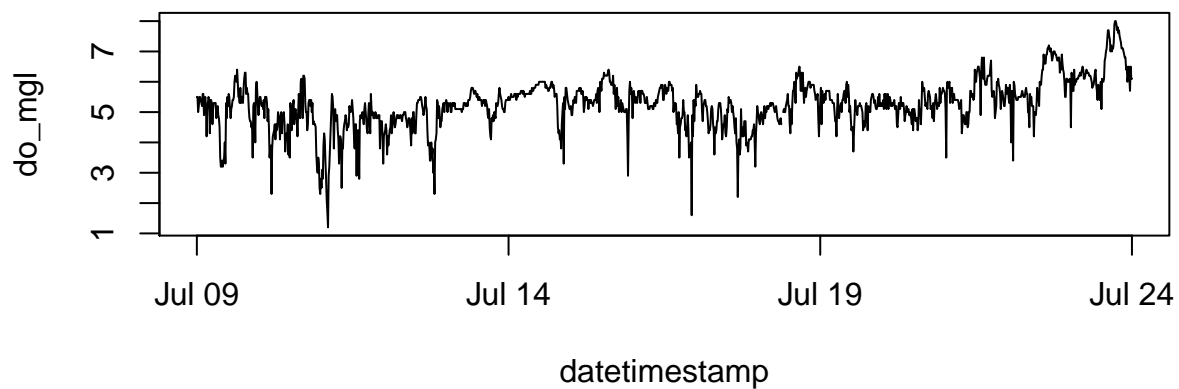
Simple time series plots.


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

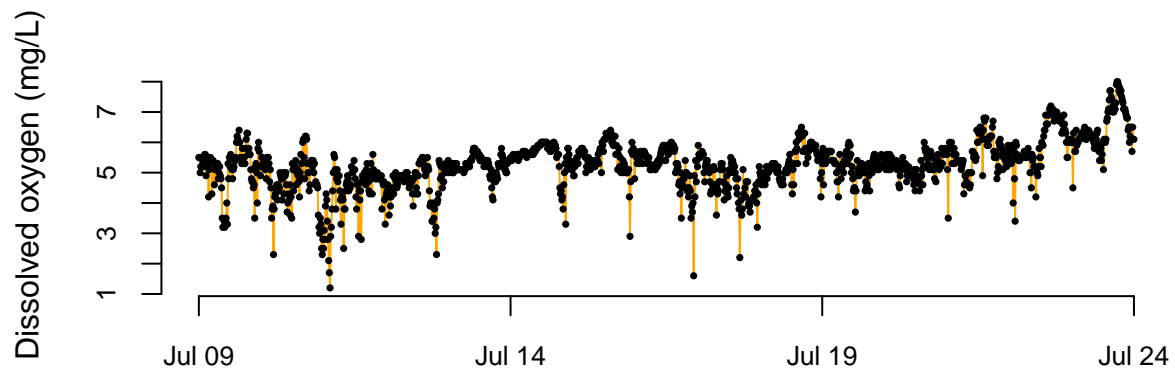
# plot with points
plot(do_mgl ~ datetimestamp, data = sub_dat)
```



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

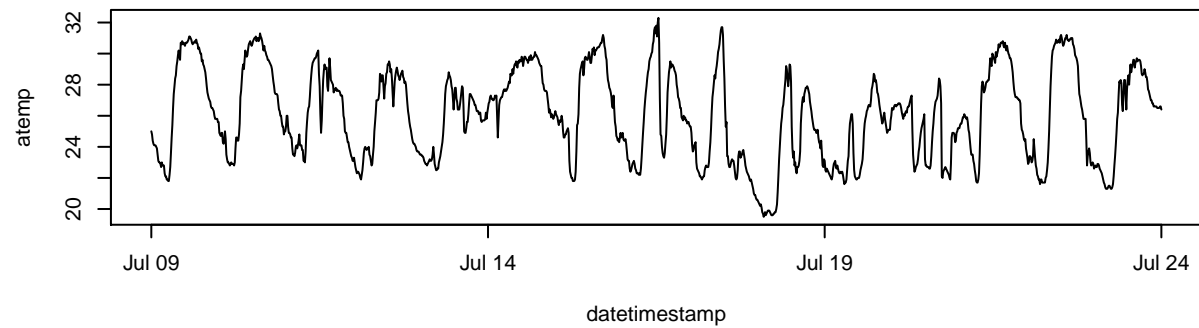
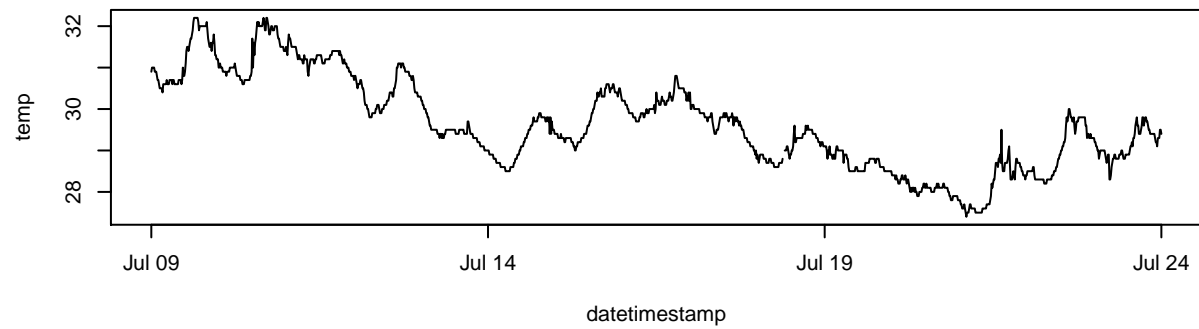
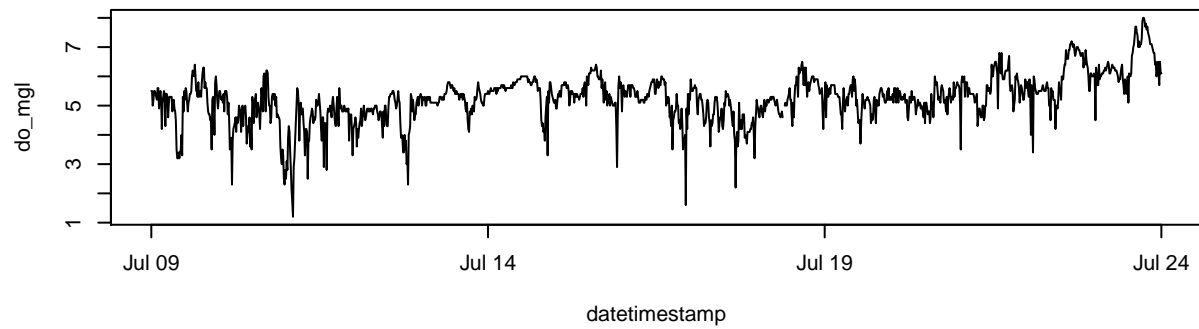


```
# changing the default arguments, add points to graph
plot(do_mgl ~ datetimestamp, data = sub_dat, type = 'l', col =
      'orange', xlab = '', ylab = 'Dissolved oxygen (mg/L)',
      cex.axis = 0.8, bty = 'n')
points(sub_dat$datetimestamp, sub_dat$do_mgl, pch = 16, cex = 0.5)
```



Multiple plots in the same window.

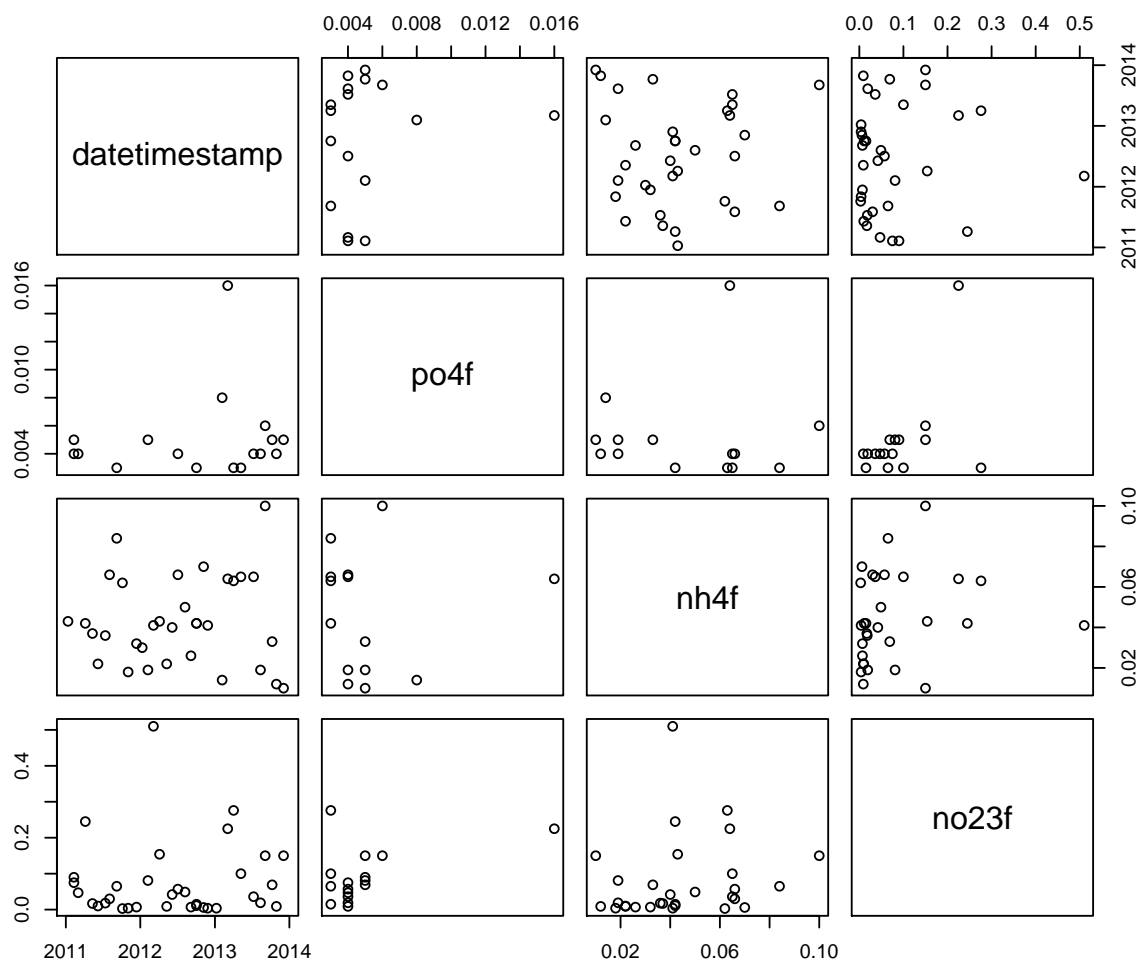
```
# it is often useful to open a new plot window, uncomment and run the next line  
# windows()  
  
# create three plots in the same window  
# 3 rows one column, use mfrow argument with par  
par(mfrow = c(3, 1))  
plot(do_mgl ~ datetimestamp, data = sub_dat, type = 'l')  
plot(temp ~ datetimestamp, data = sub_dat, type = 'l')  
plot(atemp ~ datetimestamp, data = sub_dat, type = 'l')
```



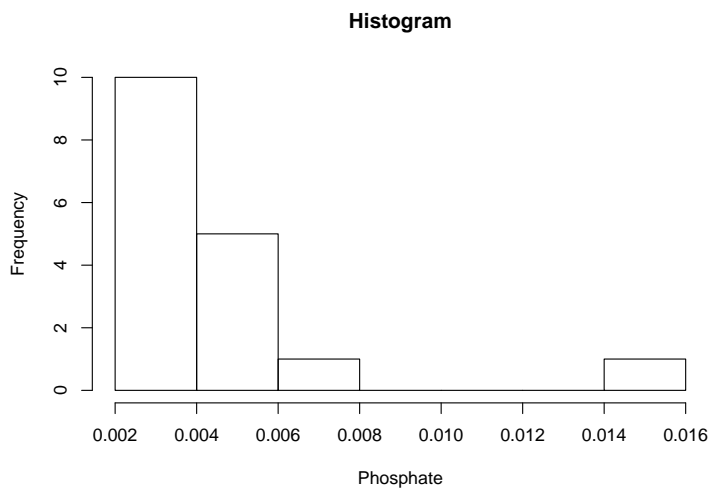
Diagnostics plots.

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

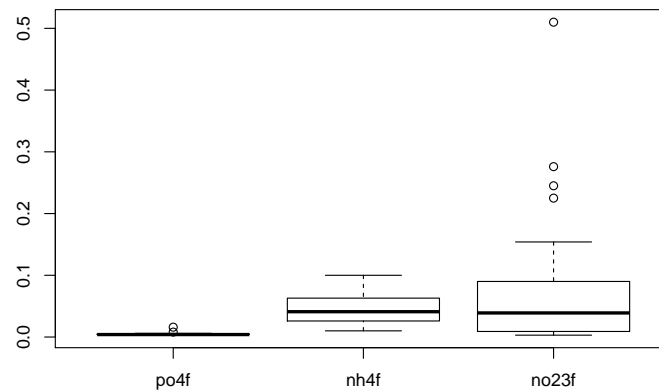
# pairs plot
# a pairs plot is a matrix of bivariate scatterplots
# caution, this plot may take a while to load for large datasets
pairs(sub_dat)
```



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



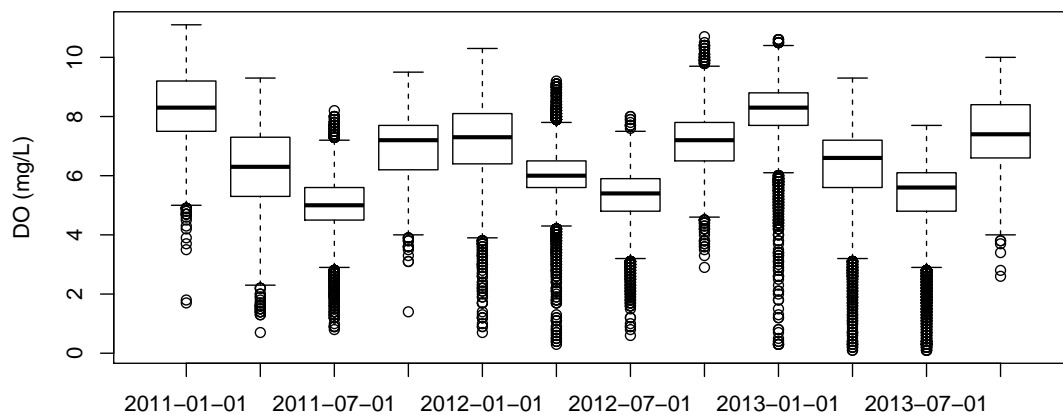
```
# boxplots
# must remove datetimestamp
to_plo <- data.frame(sub_dat)[, -1]
boxplot(to_plo)
```



A boxplot of aggregation results.

```
# aggregate by DO by seasons/quarters
to_plo <- aggregate(dat, by = 'quarters', params = 'do_mgl',
  aggs_out = T)

# plot
boxplot(do_mgl ~ datetimestamp, data = to_plo, ylab = 'DO (mg/L)')
```



Plotting with ggplot2

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

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

# install the reshape2 package for formatting data before plotting
install.packages('reshape2')
```

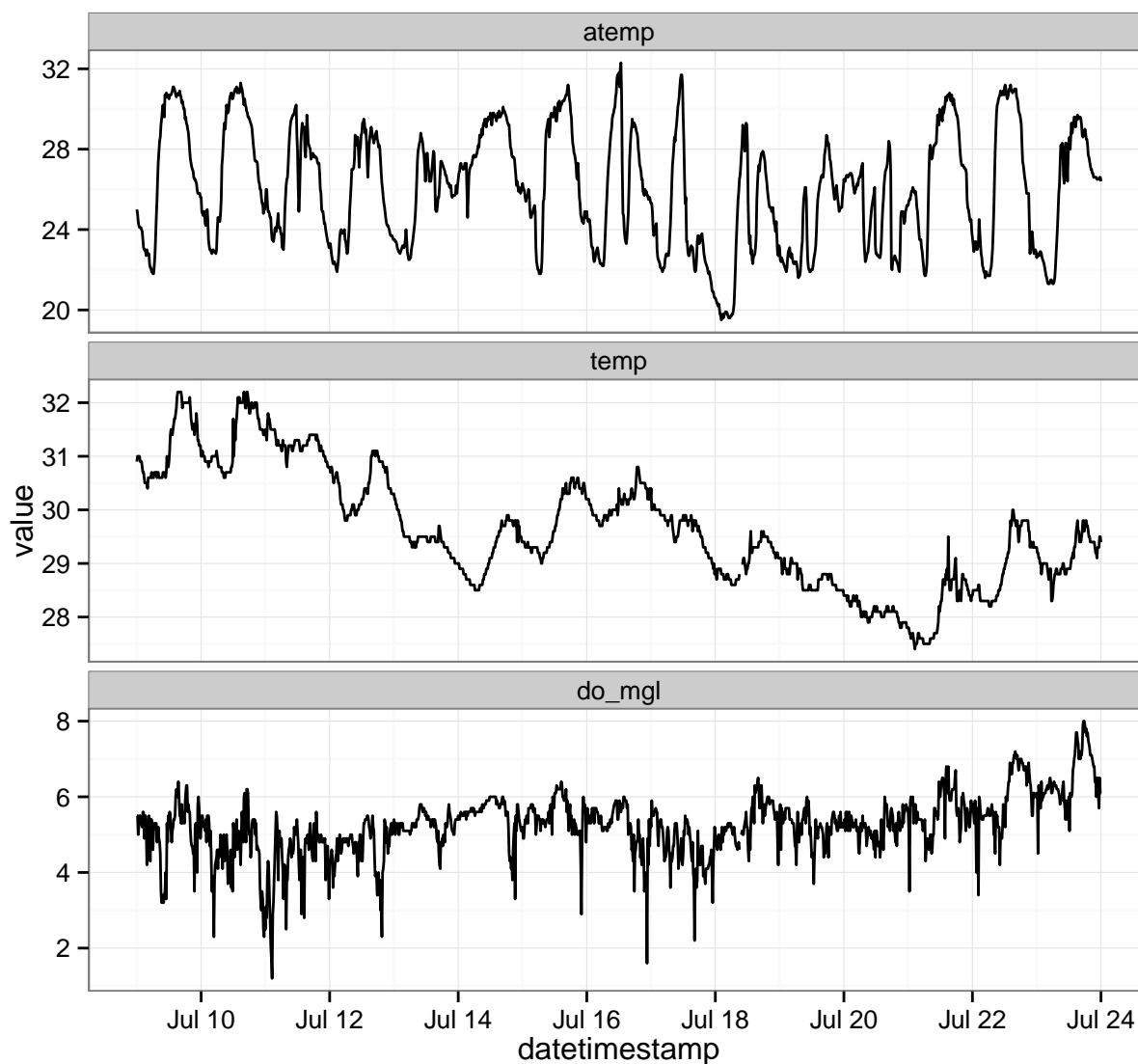
A multivariate time series plot with ggplot2.

```
# 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
```

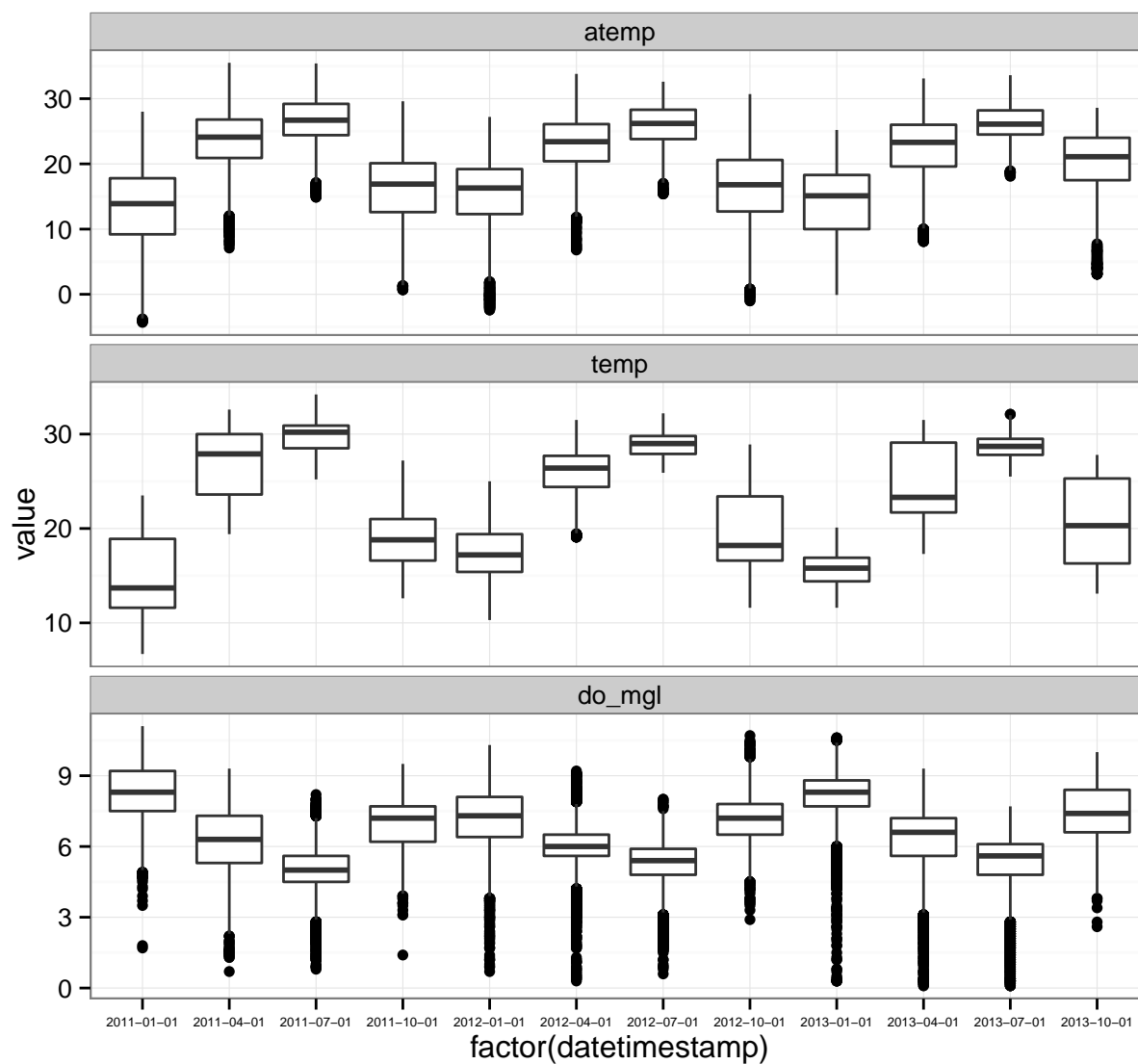


Using aggregation and ggplot2 to plot multiple variables.

```
# 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')

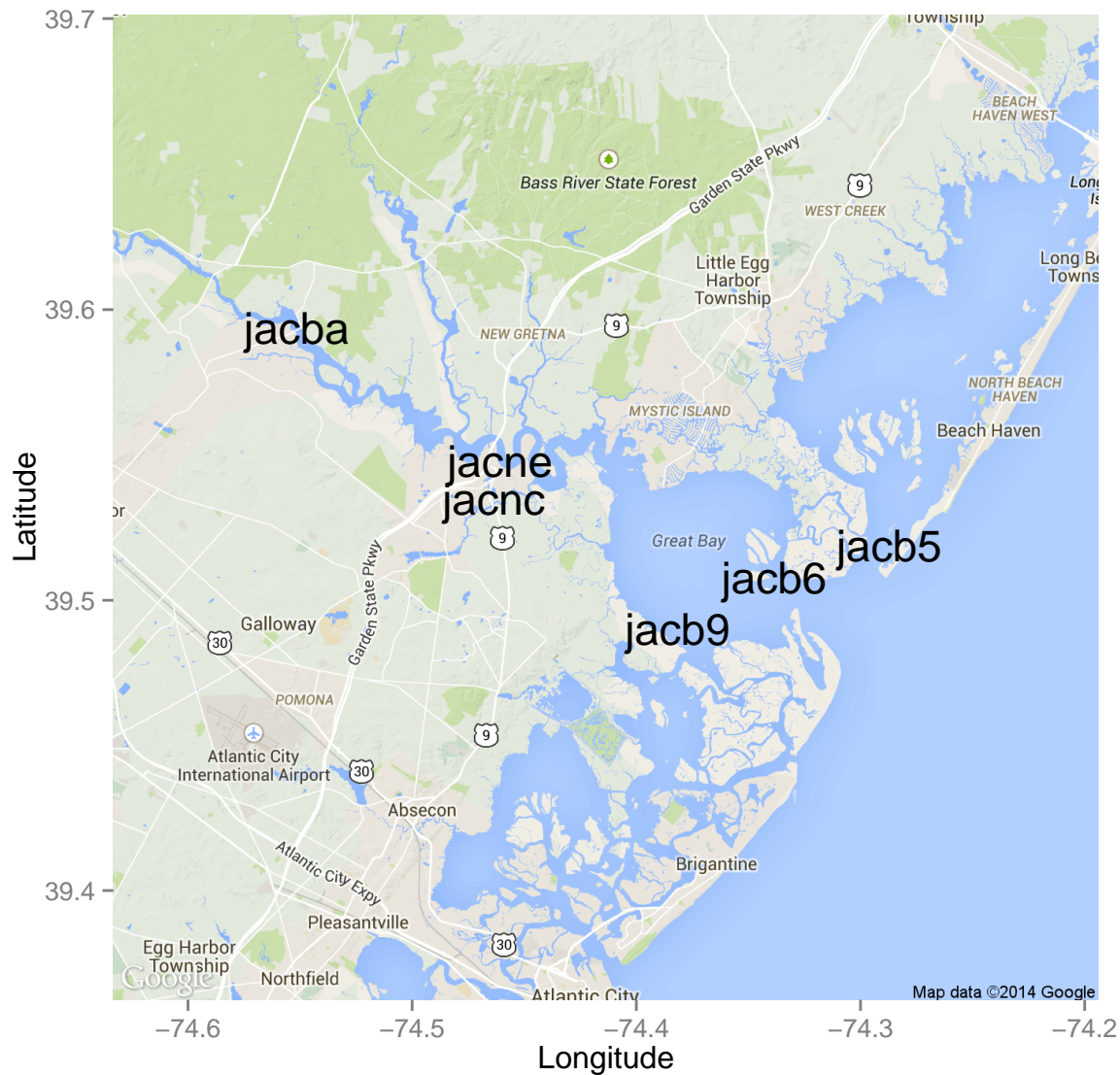
# plot
p <- ggplot(to_plo, aes(x = factor(datetimestamp), y = value)) +
  geom_boxplot() +
  facet_wrap(~ variable, ncol = 1, scales = 'free_y') +
  theme_bw() +
  theme(axis.text.x = element_text(size = 5))
p
```



Maps

The `map_reserve` function uses the `ggmap` package to create a plot of the stations at a reserve. The `ggmap` package should install and load automatically when you use `SWMPPr`. 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, or using the Export option in RStudio. You can also save graphics using specific commands that are run in the console. The following shows how to save a graphic as a pdf or png file.

```
# 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  
# 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()
```