R403: Probabilistic and Statistical Computations with R

Topic 12: Time Series Manipulation in R

Kaloyan Ganev

2022/2023

Lecture Contents

Introduction

What is a time series?

- An informal definition:

Definition 1

A time series is a time-ordered series of data observations on one or more variables.

- Although it is brief, it suggests a lot of insight
- First, each observation belongs to a specified point in time and to no other; time values index the observations
- The number of observations cannot be indefinitely large, therefore we have discrete data
- Usually measurements are taken at equal intervals but there are exceptions to this rule

R's capabilities of working with time series

- Quite large and extensive
- Some of them shipped with the base version, others available through contributed packages
- We will start with the first type and then we will discuss some of the most frequently use ones of the second type
- Do not forget to pay a visit to the CRAN Task View of packages for a more comprehensive list

Base R time series capabilities

The **ts** class

- Represents regularly spaced time series
- Uses numeric time stamps
- Obtained through the conversion of numeric vectors
- Conversion is achieved by means of the `ts()` command available is the automatically loaded **stats** package
- General construct:

```
ts(x, start=, end=, frequency=)
```

where `x` is the input vector; the others are more or less self explanatory

An example of creating time series with `ts()`

- We will use the **eurostat** package to download some data
- Data are on GDP and its components for all EU Member States
- We choose to work with quarterly frequencies
- Load the package, alongside with **tidyverse** which will be used for data processing:

```
library(eurostat)
library(tidyverse)
```

- First, we look for all datasets containing the string “GDP”:

```
search1 <- search_eurostat("GDP", type = "dataset")
```

- Then we download data on quarterly GDP and its components

```
data_gdpq <- get_eurostat("namq_10_gdp", time_format = "date")
```

An example of creating time series with `ts()` (2)

- Using the functions of the **dplyr** package, we filter data on Bulgarian GDP and consumption only
- The values we choose are at 2010 prices, not seasonally adjusted

```
data_gdpq_bg <- data_gdpq %>%
  filter(geo == "BG",
         unit == "CLV10_MEUR",
         s_adj == "NSA",
         na_item %in% c("B1GQ", "P3")) %>%
  select(time, values, na_item) %>%
  spread(na_item, values)
```

An example of creating time series with `ts()` (3)

- Create names for the time series from the names of the variables in the data frame

```
names_ts <- character()

for(i in names(data_gdpq_bg)[2:3]){
    names_ts[i] <- paste(i, ".ts", sep = "")
  }

names_ts
```

- Create the time series by assigning variables from the data frame to each name in the character vector:

```
for(i in 1:length(names_ts)){
    assign(names_ts[i],ts(data_gdpq_bg[[i]], start = c(1995,1),
        frequency = 4))
}
```

Properties of **ts** objects

- You can view any **ts** object by just printing it in the console, e.g.:+

```
B1GQ.ts
```

- Check the class:

```
class(B1GQ.ts)
```

- The start date, the end date, and the frequency of a time series can be viewed by means of:

```
start(B1GQ.ts)
end(B1GQ.ts)
frequency(B1GQ.ts)
```

- You can also check the time interval between two consecutive observations:

```
deltat(B1GQ.ts)
```

Subsetting **ts** objects

- The usual approaches used for vectors and lists are not appropriate
- They will not produce objects of the **ts** type
- Instead, the `window()` command is used, e.g.:

```
B1GQ_smpl.ts <- window(B1GQ.ts, start = c(2010, 1), end = c
    (2019,2))

B1GQ_smpl.ts
```

- Check the sample class:

```
class(B1GQ_smpl.ts)
```

Manipulating **ts** objects

- **ts** objects can be generated directly from a data frame:

```
mts1 <- ts(data_gdpq_bg[,2:3], start = c(1995, 1), frequency = 4)
class(mts1)
```

- Individual **ts** objects can be combined using `cbind()` or `ts.union()`
- Examples:

```
ts_comb1 <- cbind(B1GQ.ts, P3.ts)
ts_comb2 <- ts.union(B1GQ.ts, P3.ts)
```

- Check class:

```
class(ts_comb1)
class(ts_comb2)
```

Manipulating **ts** objects (2)

- Lagged series (note we are using the function from the **stats** package):

```
B1GQ_lag1.ts <- stats::lag(B1GQ.ts,-1)
B1GQ_lag1.ts
start(B1GQ_lag1.ts)
```

- The default value is 1, however it shifts all observations one period backwards
- Using -1 shifts observations forward, thus matching current time with one-period lag value
- Get the series and its first lag together

```
mts2 <- ts.union(B1GQ.ts, B1GQ_lag1.ts)
```

- Get the common sample data for two or more series:

```
mts3 <- ts.intersect(B1GQ.ts, B1GQ_lag1.ts)
```

Manipulating **ts** objects (3)

- Differences of series:

```
diff(B1GQ.ts)
```

- This will give you the first difference of the series
- You can specify a higher differencing order:

```
diff(B1GQ.ts, differences = 2)
```

- Differences can be used in conjunction with lags (an option provided in the function):

```
diff(B1GQ.ts, lag = 4)
```

- The latter will produce seasonal differences in the current example

Using contributed packages

The **zoo** package

- The **ts** class has some limitations, e.g. it cannot work with irregularly spaced time series
- This can cause unpleasant issues for example when you work with financial time series (especially daily or higher-frequency data[1])
- The package provides the **zoo** class of objects which handle this type of situations
- This class has similar (but much more powerful) functionality to that of **ts**
- Therefore, using **zoo** is very straightforward to use when one is familiar with **ts**
- Also, there is easy convertibility between the two classes

[1] Monday to Friday are equally spaced but spacing between Friday and Monday is different, etc.

The **zoo** package (2)

- To create a **zoo** object, one needs data and a time-ordered index
- The data can be contained in a vector or in a matrix
- Let's take an example; create a random 120×4 matrix:

```
m1 <- matrix(rnorm(480), nrow = 120)
colnames(m1) <- c("var1", "var2", "var3", "var4")
```

- Create a time index of class **Date**:

```
idx1 <- seq(from = as.Date("2001-01-01"),
            length.out = 120, by = "months")
```

- Combine the data and the index in the **zoo** object:

```
zoo1 <- zoo(m1, order.by = idx1)
```

- The index variable can be of any of the valid date and time classes

The **zoo** package (3)

- In order to extract the index of a **zoo** object, type:

```
index(zoo1)
```

- The data can be extracted via:

```
coredata(zoo1)
```

- Of course, both are assignable to new objects
- As with **ts** objects, you can find start and end dates by:

```
start(zoo1)
end(zoo1)
```

The **zoo** package (4)

- The `summary()` command works seamlessly with **zoo** objects:

```
summary(zoo1)
```

- The same goes for the `str()` function:

```
str(zoo1)
```

- For larger **zoo** objects, it is convenient to look at only some observations and not at all of them:

```
head(zoo1)
tail(zoo1)
```

(the latter two work also for other objects)

The **zoo** package (5)

- Subsetting (sampling) is (not surprisingly) done by means of the `window()` function:

```
zoo2 <- window(zoo1, start = as.Date(2005-11-15), end = as.Date
    (2007-08-30))
```

- You can check the type of the new object to see that the data type is preserved
- An individual series can be selected in two ways:

```
zoo1[,1]
zoo1$var1
```

(the former uses matrix notation, and the latter – list notation)

- Individual elements (observations) can be selected using their matrix indices or, again, list notation:

```
zoo1[2,3]
zoo1$var3[2]
```

The **zoo** package (6)

- Lags and differences is performed in the same way as with **ts** objects:

```
stats::lag(zoo1, -1)
diff(zoo1, differences = 1)
```

- It is possible to merge several **zoo** objects:

```
zoo3 <- cbind(zoo1$var2, zoo1$var1, stats::lag(zoo1$var1, -1))
# or
zoo3 <- merge(zoo1$var2, zoo1$var1, stats::lag(zoo1$var1, -1))
```

- It is preferable however to use `merge()` as it works as expected for cases when different objects have different time indices (not valid for `cbind()`)

The **zoo** package (7)

- External data can be read directly to **zoo** objects
- This is done by means of the `read.zoo()` command which essentially is a wrapper around the `read.table()` command
- Example (use again the GDP data):

```
data_gdpq_bg.zoo <- read.zoo("data_gdpq_bg.csv",
                             index.column = 1,
                             FUN = as.Date,
                             sep = ",",
                             header = T)
```

The **zoo** package (8)

- In this example, `index.column = 1` refers to the column where date and time information lies
- `FUN = as.Date` specifies the function to apply to this column so that it is converted to the **Date** class (**yearmon** and **yearqtr** are also provided by **zoo**)
- See the documentation for a complete list of options
- Also, note that the options are extensible by the list of the options of `read.table()`
- Similarly to `write.table()`, there is a `write.zoo()` command:

```
write.zoo(data_gdpq_bg.zoo, "data_gdpq_bg.zoo.csv", index.name =
    "Time period", sep = ",")
```

The **zoo** package (9)

- Finally, let's take examples of conversion between **zoo** and **ts**
- Direct conversion from **zoo** to **ts** is no longer possible, so you have to use a data frame as a medium

```
ts_conv <- as.data.frame(data_gdpq_bg.zoo)
class(ts_conv)
ts_conv <- ts(ts_conv, start = c(1995,1), frequency = 4)
```

- To convert from **ts** to **zoo**:

```
zoo_conv <- as.zoo(ts_comb1)
zoo_conv
index(zoo_conv)
class(zoo_conv)
```

The **xts** package

- The **xts** class is an extension of the **zoo** class
- Decrypted as “extensible time series”
- Possesses a wider set of functions allowing more versatile data processing, manipulation, and conversion
- This is achieved alongside with greater user-friendliness, simplicity and usability
- Each **xts** objects has three components:
    - A vector of times and/or dates
    - The core data which is again a matrix
    - Attributes which include an index of times and dates and time zone format

The **xts** package (2)

- To create an **xts** object, the `xts()` command is used:

```
xts1 <- xts(data_gdpq_bg[,2:3], order.by = data_gdpq_bg$time,
    descr = "first xts object")
```

- Obviously, it can contain a `descr` attribute for easier reference
- Check its class to see that it is created on top of a **zoo** structure:

```
class(xts1)
```

- Data can be queried/sampled by character matching, e.g.:

```
xts1["2010"] # select all quarters of 2010
xts1["2010-01"] # select Q1 2010
xts1["2010-01/2010-06"] # select Q1-Q2 2010
```

The **xts** package (3)

- Let's create a date-time object in the POSIXct format using the **lubridate** package:

```
library(lubridate)
nowtime <- now()
idx2 <- nowtime + days(0:364)
last(idx2)
```

- Create a data vector of the same length:

```
rnorm_data <- rnorm(length(idx2))
```

- Create the **xts** object

```
xts2 <- xts(rnorm_data, order.by = idx2)
colnames(xts2)[1] <- "data"
head(xts2)
```

The **xts** package (4)

- There are specialized loop functions for **xts** objects, e.g.:

```
apply.monthly(xts2, mean)
apply.quarterly(xts2, sd)
apply.yearly(xts2, sum)
```

- You can also use an anonymous function in place of the above
- Merging **xts** objects is again done with `merge()`, with some more additional options (not discussed here)
- Start time and ending time are obtained by means of:

```
start(xts2)
end(xts2)
```

The **xts** package (5)

- You can also calculate the number of days, weeks, months, etc. in an **xts** object:

```
ndays(xts2)
nmonths(xts2)
nyears(xts2)
```

- Statistical calculations by time period are performed in the following manner:

```
endp1 <- endpoints(xts2,'weeks')
period.apply(xts2, INDEX=endp1, mean)
endp2 <- endpoints(xts2,'months')
period.max(xts2, INDEX=endp2)
```

- etc.

References

- Packages documentation
- Kabacoff, R. (2015): *R in Action: Data analysis and graphics with R*, Manning, 2nd ed., Ch. 15
- Zhang, D. (2016): *R for Programmers*, CRC Press, Ch. 2