Post 02 Time Series Data in R

Bryant Luong

2017-12-01

Motivation

In Stats 133, we've focused on categorical data, such as the basketball statistics. So I've been wondering how to handle times series data in R. It is useful to understand the DAC process for categorical **and** times series data.

Time series data is everywhere. From hospitals to Wall Street to factories and now our wrists, humans constantly track processes to watch for changes and consistencies. In the hospital, teams monitor their patients' vitals. On Wall Street, analysts and investors try to predict the trend of prices. In factories, engineers optimize the yield and quality of production lines.

In this post, we will learn how to represent and manipulate time series data in R using the `xts` package.

`xts` History

`xts` was created by Jeffrey A. Ryan, Joshua M. Ulrich, and Ross Bennett around 2008. `xts` is short for extensible time series. They wanted a fast and consistent way to represent time series data in R. Although the `xts` package is only in version 0.10 after existing for almost 10 years, it is popular. DataCamp's classes currently use the `xts` package to represent time series data, making it even more popular.

The `xts` Object

In R, there are many ways to represent time. You can use a `data.frame` and store the day or hour as a `double`. Enter the following in your R console to follow along:

```
ex1.df <- data.frame("Day" = 1:30, "Date" = Sys.Date()+1:30)
ex1.df
```

```
##    Day       Date
## 1    1 2017-12-04
## 2    2 2017-12-05
## 3    3 2017-12-06
## 4    4 2017-12-07
## 5    5 2017-12-08
## 6    6 2017-12-09
## 7    7 2017-12-10
## 8    8 2017-12-11
## 9    9 2017-12-12
## 10  10 2017-12-13
## 11  11 2017-12-14
## 12  12 2017-12-15
## 13  13 2017-12-16
## 14  14 2017-12-17
## 15  15 2017-12-18
## 16  16 2017-12-19
## 17  17 2017-12-20
## 18  18 2017-12-21
## 19  19 2017-12-22
## 20  20 2017-12-23
## 21  21 2017-12-24
## 22  22 2017-12-25
## 23  23 2017-12-26
## 24  24 2017-12-27
## 25  25 2017-12-28
## 26  26 2017-12-29
## 27  27 2017-12-30
## 28  28 2017-12-31
## 29  29 2018-01-01
## 30  30 2018-01-02
```

```
typeof(ex1.df$Date)
```

```
## [1] "double"
```

Notice the type of the values in `ex1.df$Date` is just `double`.

A better representation is an extensible time series object. You can create a extensible time series object by using the `xts` function. To use the `xts` function , you need to first install the `xts` package by running `install.packages('xts')`. Then remember to add it to your current R session:

```
# add xts library to current R sessions
library(xts)
```

```
## Loading required package: zoo
```

```
##
## Attaching package: 'zoo'
```

```
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

```
# create your first xts object
ex1.xts <- xts(1:30, Sys.Date()+1:30)

# look at xts object
ex1.xts
```

```
##            [,1]
## 2017-12-04    1
## 2017-12-05    2
## 2017-12-06    3
## 2017-12-07    4
## 2017-12-08    5
## 2017-12-09    6
## 2017-12-10    7
## 2017-12-11    8
## 2017-12-12    9
## 2017-12-13   10
## 2017-12-14   11
## 2017-12-15   12
## 2017-12-16   13
## 2017-12-17   14
## 2017-12-18   15
## 2017-12-19   16
## 2017-12-20   17
## 2017-12-21   18
## 2017-12-22   19
## 2017-12-23   20
## 2017-12-24   21
## 2017-12-25   22
## 2017-12-26   23
## 2017-12-27   24
## 2017-12-28   25
## 2017-12-29   26
## 2017-12-30   27
## 2017-12-31   28
## 2018-01-01   29
## 2018-01-02   30
```

Notice the `xts()` function is similar to the `data.frame()` function—when we call them with our parameters, R creates something that stores our data. However, you can see that the outputs are very different even though we used the same input parameters. The `xts` representation of time is useful because it allows us to subset, in the same way we subset vectors, with time:

Subset by Year

```
# subset by year
ex1.xts['2017']
```

```
##            [,1]
## 2017-12-04    1
## 2017-12-05    2
## 2017-12-06    3
## 2017-12-07    4
## 2017-12-08    5
## 2017-12-09    6
## 2017-12-10    7
## 2017-12-11    8
## 2017-12-12    9
## 2017-12-13   10
## 2017-12-14   11
## 2017-12-15   12
## 2017-12-16   13
## 2017-12-17   14
## 2017-12-18   15
## 2017-12-19   16
## 2017-12-20   17
## 2017-12-21   18
## 2017-12-22   19
## 2017-12-23   20
## 2017-12-24   21
## 2017-12-25   22
## 2017-12-26   23
## 2017-12-27   24
## 2017-12-28   25
## 2017-12-29   26
## 2017-12-30   27
## 2017-12-31   28
```

Subset by Month

```
# subset by month
ex1.xts['201801']
```

```
##            [,1]
## 2018-01-01   29
## 2018-01-02   30
```

Subset by a Window

```
# subset a window of time
ex1.xts['2017-12-15/2017-12-18']
```

```
##            [,1]
## 2017-12-15   12
## 2017-12-16   13
## 2017-12-17   14
## 2017-12-18   15
```

```
# check the type of subsetted results
typeof(ex1.xts['201801'])
```

```
## [1] "integer"
```

Subset Until Christmas

```
# subset until Christmas
ex1.xts['/2017-12-25']
```

```
##            [,1]
## 2017-12-04    1
## 2017-12-05    2
## 2017-12-06    3
## 2017-12-07    4
## 2017-12-08    5
## 2017-12-09    6
## 2017-12-10    7
## 2017-12-11    8
## 2017-12-12    9
## 2017-12-13   10
## 2017-12-14   11
## 2017-12-15   12
## 2017-12-16   13
## 2017-12-17   14
## 2017-12-18   15
## 2017-12-19   16
## 2017-12-20   17
## 2017-12-21   18
## 2017-12-22   19
## 2017-12-23   20
## 2017-12-24   21
## 2017-12-25   22
```

We can subset with time because the data in an `xts object` is indexed by time. If we subset the same set of data in the data frame, it would take more code and feel very unnatural. The data frame sees time as another column. The `xts object` sees time as the organizing attribute of the dataset. Time is so central to the `xts object` that time is represented by an international standard called ISO-8601.

Benefits of Using the `xts` object

There are other benefits from representing time series data as an `xts` object.

```
# quickly find the value at the end of the weeks
endpoints(ex1.xts, on = "weeks")
```

```
## [1]  0  7 14 21 28 30
```

```
# quickly find the value at the end of months
endpoints(ex1.xts, on = "months")
```

```
## [1]  0 28 30
```

Notice the last value `30` is not the value at the end of January. It is the last January data point in the data set. Check this is so by entering `ex1.xts` into the console. These functions are useful to quickly track milestones.

For example, if the data is collected cumulatively, the endpoints we looked at above would represent the current progress. A real example is to find out the total number of cars that Tesla has created by the time you're ready to buy one of their cars. Currently, the federal tax credit applies to the first 250,000 cars that Tesla sells. If you are in line for a Model 3, you might need to know if your car is within the first 250,000.

You can also look at the first and last parts of the data:

```
# look at first day of data
first(ex1.xts, 'day')
```

```
##            [,1]
## 2017-12-04    1
```

```
# look at last day of data
last(ex1.xts, 'day')
```

```
##            [,1]
## 2018-01-02   30
```

```
# look at first week of data
first(ex1.xts, 'week')
```

```
##            [,1]
## 2017-12-04    1
## 2017-12-05    2
## 2017-12-06    3
## 2017-12-07    4
## 2017-12-08    5
## 2017-12-09    6
## 2017-12-10    7
```

```
# look at first 2 weeks of data
first(ex1.xts, '2 weeks')
```

```
##            [,1]
## 2017-12-04    1
## 2017-12-05    2
## 2017-12-06    3
## 2017-12-07    4
## 2017-12-08    5
## 2017-12-09    6
## 2017-12-10    7
## 2017-12-11    8
## 2017-12-12    9
## 2017-12-13   10
## 2017-12-14   11
## 2017-12-15   12
## 2017-12-16   13
## 2017-12-17   14
```

You can also use the `first` and `last` functions together:

```
# look at last 9 days of the year
last(first(ex1.xts, 'month'), '9 days')
```

```
##            [,1]
## 2017-12-23   20
## 2017-12-24   21
## 2017-12-25   22
## 2017-12-26   23
## 2017-12-27   24
## 2017-12-28   25
## 2017-12-29   26
## 2017-12-30   27
## 2017-12-31   28
```

'Tis The Season

You can also inspect, change, and use the periodicity of the data.

If you do not know how often the data is recorded, you can just ask:

```
periodicity(ex1.xts)
```

```
## Daily periodicity from 2017-12-04 to 2018-01-02
```

If daily data is too much, you can change the periodicity too:

```
to.period(ex1.xts, "weeks")
```

```
##            ex1.xts.Open ex1.xts.High ex1.xts.Low ex1.xts.Close
## 2017-12-10            1            7           1             7
## 2017-12-17            8           14           8            14
## 2017-12-24           15           21          15            21
## 2017-12-31           22           28          22            28
## 2018-01-02           29           30          29            30
```

`xts` will give you a summary of the head, tail, max, and min values of the new periods! The column names of `Open`, `High`, `Low`, and `Close` were chosen because `xtr` was created to analyze stock market securities data.

Now imagine you're a manufacturing engineer and your manager needs to know how the line is performing. If you need to find the maximum number of units produced in a period, you can use the `period.apply` function in `xts`:

```
period.apply(ex1.xts, INDEX = endpoints(ex1.xts, "weeks"), FUN = max)
```

```
##            [,1]
## 2017-12-10    7
## 2017-12-17   14
## 2017-12-24   21
## 2017-12-31   28
## 2018-01-02   30
```

The first argument to `period.apply` is the data. The second argument is how you want to divide your data by time. The third argument is the function you want to apply to each window of time.

`xts` also has built-in `apply` functions for daily, weekly, monthly, quarterly, and yearly. The structure of these built-in functions is `apply.X`. If you need to apply a function quarterly, the function is `apply.quarterly()`.

Conclusion

Time series data is very powerful. In this post, we only used dummy data that we created but the methods work on very large data sets. You now can inspect data about the stock market, the sales performance of a business, or the production numbers of a factory. You can even parse log files after some manipulation. `xts` can handle time-stamped strings but I recommend you shorten the messages of log files or create mappings from log file messages to integers or characters.

In this post, you learned about the `xts` object and package. You learned why the `xts` object is powerful and how to manipulate it.

References

1. http://joshuaulrich.github.io/xts/
2. https://cran.r-project.org/web/packages/xts/vignettes/xts.pdf
3. https://cran.r-project.org/web/packages/xts/vignettes/xts-faq.pdf
4. http://past.rinfinance.com/agenda/2009/xts_quantmod_workshop.pdf
5. https://www.r-bloggers.com/xts-cheat-sheet-time-series-in-r/
6. https://cran.r-project.org/web/packages/xts/index.html
7. https://www.datacamp.com/tracks/quantitative-analyst-with-r