Statistical Methods for Discrete Response, Time Series, and Panel Data (W271): Lab 3

Eric Yang, Samir Datta, Carlos Castro

October 25, 2017

Instructions:

- **Due Date: 11/17/2017 (by mid-night)**
- Submission:
  - Submit your own assignment via ISVC
  - Submit 2 files:
    1. A pdf file including the summary, the details of your analysis, and all the R codes used to produce the analysis. Please do not suppress the codes in your pdf file.
    2. R markdown file used to produce the pdf file
  - Each group only needs to submit one set of files
  - Use the following file naming convensation
    - SectionNumber_hw01_FirstNameLastNameFirstInitial.fileExtension
    - For example, if you are in Section 1 and have two students named John Smith and Jane Doe, you should name your file the following
      - Section1_hw01_JohnS_JaneD.Rmd
      - Section1_hw01_JohnS_JaneD.pdf
  - Although it sounds obvious, please write the name of each members of your group on page 1 of your report.
  - This lab can be completed in a group of up to 3 people. Each group only needs to make one submission. Although you can work by yourself, we encourage you to work in a group.

Introduction and Objective

This lab is should be treated as a tutorial instead of a typical lab in this class. The objective of this "lab"" is to help you practice manipulating time series data in R using the *xts* time-series class.

Instead of having you read a bunch of documents before even having you develop any codes, I design a 3-step approach for you to walk through this lab:

1. Have you read only a couple of pages to get a very quick introduction and motivation of using the time-series class xts
2. Have you developed some conduct to learn how to use xts to accomplish frequently-encountered tasks when working with time series data
3. Have you gone back and studied the details behind the methods you used in your code development.

This lab/tutorial starts with a quick introduction to xts and zoo objects, followed by the concepts of creating an xts object and converting to an xts object from an imported dataset, explaining how to construct and deconstruct an xts object.

As in any data analysis, you most likely will have to combine dataset. We cover merging and modifying time series after studynig xts object construction. We cover different kinds of joins - outer, inner, left, and right join.

I introduce the library `quantmod` and the `getSymbols` function to download the historical Twitter stock price directly from the Google website. Unlike the unemployment time series, which comes in with monthly frequency, the Twitter stock price and volume series come in as the daily frequency, making them good candidates for learning how to merge time series of different time frequencies, an activity that you may have to implement a lot in practice.

With time series of different time frequencies, inevitably one may have to fill in missing values. Remember that the specific method for missing value imputation is context-dependent. I cover a couple of methods but by no means endose them as the "to-go" methods for filling in missing values.

Finally, this lab introduces two use techniques: differencing a time series (against itself) and apply various functions to time series, which is used frequently in rolling statistics calculation and time series aggregation.

Materials Covered in this lab

- Primarily the references listed in this document
- Reference
  - "xts: Extensible Time Series" by Jeffrey A. Ryan and Joshua M. Ulrich. 2008. (xts.pdf)
  - "xts FAQ" by xts Development Team. 2013 (xts_faq.pdf)
  - xts_cheatsheet.pdf

Tasks 1:

1. Read A. the **Introduction** section (Section 1), which only has 1 page of reading of xts: Extensible Time Series" by Jeffrey A. Ryan and Joshua M. Ulrich B. the first three questions in"xts FAQ" a. What is xts? b. Why should I use xts rather than zoo or another time-series package? c. HowdoIinstallxts? C. The "A quick introduction to xts and zoo objects" section in this document
2. Read the "A quick introduction to xts and zoo objects" of this document

A quick introduction to xts and zoo objects

xts

`xts` - stands for eXtensible Time Series - is an extended zoo object - is essentially matrix + (time-based) index (aka, observation + time)

- xts is a constructor or a subclass that inherits behavior from parent (zoo); in fact, it extends the popular zoo class. As such. most zoo methods work for xts
- is a matrix objects; subsets always preserve the matrix form
- importantly, xts are indexed by a formal time object. Therefore, the data is time-stamped

- The two most important arguments are `x` for the data and `order.by` for the index. `x` must be a vector or matrix. `order.by` is a vector of the same length or number of rows of `x`; it must be a proper time or date object and be in an increasing order

Task 2:

1. Read A. Section 3.1 of "xts: Extensible Time Series" by Jeffrey A. Ryan and Joshua M. Ulrich

   B. the following questions in "xts FAQ" a. How do I create an xts index with millisecond precision? b. OK, so now I have my millisecond series but I still can't see the milliseconds displayed. What went wrong?

2. Follow the following section of this document

Creating an xts object and converting to an xts object from an imported dataset

We will create an xts object from a matrix and a time index. First, let's create a matrix and a time index. The matrix, as it creates, is not associated with the time indext yet.

```
# Create a matrix
x <- matrix(rnorm(200), ncol=2, nrow=100)
colnames(x) <- c("Series01", "Series02")
str(x)
```

```
##  num [1:100, 1:2] -0.599 -0.342 -0.146 -0.56 -0.472 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : NULL
##   ..$ : chr [1:2] "Series01" "Series02"
```

```
head(x,10)
```

```
##         Series01   Series02
##  [1,] -0.5987866 -1.7965721
##  [2,] -0.3424454 -1.3564580
##  [3,] -0.1458975  1.6013686
##  [4,] -0.5598972  0.1916030
##  [5,] -0.4721212  0.3436217
##  [6,] -0.8045174  2.0705147
##  [7,]  0.3919237 -1.0638484
##  [8,] -0.5558084  0.3627414
##  [9,]  1.5826427 -0.1371614
## [10,] -1.2642530 -0.1196200
```

```
idx <- seq(as.Date("2015/1/1"), by = "day", length.out = 100)
str(idx)
```

```
##  Date[1:100], format: "2015-01-01" "2015-01-02" "2015-01-03" "2015-01-04" "2015-01-05" ...
```

```
head(idx)
```

```
## [1] "2015-01-01" "2015-01-02" "2015-01-03" "2015-01-04" "2015-01-05"
## [6] "2015-01-06"
```

```
tail(idx)
```

```
## [1] "2015-04-05" "2015-04-06" "2015-04-07" "2015-04-08" "2015-04-09"
## [6] "2015-04-10"
```

In a nutshell, xts is a matrix indexed by a time object. To create an xts object, we "bind" the object with the index. Since we have already created a matrix and a time index (of the same length as the number of rows of the matrix), we are ready to "bind" them together. We will name it X.

```
library(xts)
```

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

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

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

```
X <- xts(x, order.by=idx)
str(X)
```

```
## An 'xts' object on 2015-01-01/2015-04-10 containing:
##   Data: num [1:100, 1:2] -0.599 -0.342 -0.146 -0.56 -0.472 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : NULL
##   ..$ : chr [1:2] "Series01" "Series02"
##   Indexed by objects of class: [Date] TZ: UTC
##   xts Attributes:
##  NULL
```

```
head(X,10)
```

```
##              Series01   Series02
## 2015-01-01 -0.5987866 -1.7965721
## 2015-01-02 -0.3424454 -1.3564580
## 2015-01-03 -0.1458975  1.6013686
## 2015-01-04 -0.5598972  0.1916030
## 2015-01-05 -0.4721212  0.3436217
## 2015-01-06 -0.8045174  2.0705147
## 2015-01-07  0.3919237 -1.0638484
## 2015-01-08 -0.5558084  0.3627414
## 2015-01-09  1.5826427 -0.1371614
## 2015-01-10 -1.2642530 -0.1196200
```

As you can see from the structure of an `xts` objevct, it contains both a data component and an index, indexed by an objevct of class `Date`.

xtx constructor

```
xts(x=Null,
    order.by=index(x),
    frequency=NULL,
    unique=NULL,
    tzone=Sys.getenv("TZ"))
```

As mentioned previous, the two most important arguments are `x` and `order.by`. In fact, we only use these two arguments to create a xts object before.

With a xts object, one can decompose it.

Deconstructing xts

`coredata()` is used to extract the data component

```
head(coredata(X),5)
```

```
##        Series01   Series02
## [1,] -0.5987866 -1.7965721
## [2,] -0.3424454 -1.3564580
## [3,] -0.1458975  1.6013686
## [4,] -0.5598972  0.1916030
## [5,] -0.4721212  0.3436217
```

`index()` is used to extract the index (aka times)

```
head(index(X),5)
```

```
## [1] "2015-01-01" "2015-01-02" "2015-01-03" "2015-01-04" "2015-01-05"
```

Conversion to xts from other time-series objects

We will use the same dataset "bls_unemployment.csv" that we used in the last live session to illustarte the functions below.

```
# Set working directory
wd <-"C:/Users/sdatta/Documents/MIDS_remote"
setwd(wd)

# Clean up the workspace before we begin
rm(list = ls())

df <- read.csv("bls_unemployment.csv", header=TRUE, stringsAsFactors = FALSE)

# Examine the data structure
  str(df)
```

```
## 'data.frame':    121 obs. of  4 variables:
##  $ Series.id: chr  "LNU04000000" "LNU04000000" "LNU04000000" "LNU04000000" ...
##  $ Year     : int  2007 2007 2007 2007 2007 2007 2007 2007 2007 2007 ...
##  $ Period   : chr  "M01" "M02" "M03" "M04" ...
##  $ Value    : num  5 4.9 4.5 4.3 4.3 4.7 4.9 4.6 4.5 4.4 ...
```

```
  names(df)
```

```
## [1] "Series.id" "Year"      "Period"    "Value"
```

```
  head(df)
```

```
##     Series.id Year Period Value
## 1 LNU04000000 2007    M01   5.0
## 2 LNU04000000 2007    M02   4.9
## 3 LNU04000000 2007    M03   4.5
## 4 LNU04000000 2007    M04   4.3
## 5 LNU04000000 2007    M05   4.3
## 6 LNU04000000 2007    M06   4.7
```

```
  tail(df)
```

```
##       Series.id Year Period Value
## 116 LNU04000000 2016    M08   5.0
## 117 LNU04000000 2016    M09   4.8
## 118 LNU04000000 2016    M10   4.7
## 119 LNU04000000 2016    M11   4.4
## 120 LNU04000000 2016    M12   4.5
## 121 LNU04000000 2017    M01   5.1
```

```
# Convert a column of the data frame into a time-series object
unemp <- ts(df$Value, start = c(2007,1), end = c(2017,1), frequency = 12)
  str(unemp)
```

```
##  Time-Series [1:121] from 2007 to 2017: 5 4.9 4.5 4.3 4.3 4.7 4.9 4.6 4.5 4.4 ...
```

```
  head(cbind(time(unemp), unemp),5)
```

```
##      time(unemp) unemp
## [1,]    2007.000   5.0
## [2,]    2007.083   4.9
## [3,]    2007.167   4.5
## [4,]    2007.250   4.3
## [5,]    2007.333   4.3
```

```
# Now, let's convert it to an xts object
df_matrix <- as.matrix(df)
  head(df_matrix)
```

```
##      Series.id     Year   Period Value
## [1,] "LNU04000000" "2007" "M01"  " 5.0"
## [2,] "LNU04000000" "2007" "M02"  " 4.9"
## [3,] "LNU04000000" "2007" "M03"  " 4.5"
## [4,] "LNU04000000" "2007" "M04"  " 4.3"
## [5,] "LNU04000000" "2007" "M05"  " 4.3"
## [6,] "LNU04000000" "2007" "M06"  " 4.7"
```

```
  str(df_matrix)
```

```
##  chr [1:121, 1:4] "LNU04000000" "LNU04000000" "LNU04000000" ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : NULL
##   ..$ : chr [1:4] "Series.id" "Year" "Period" "Value"
```

```
  rownames(df)
```

```
##   [1] "1"   "2"   "3"   "4"   "5"   "6"   "7"   "8"   "9"   "10"  "11"
##  [12] "12"  "13"  "14"  "15"  "16"  "17"  "18"  "19"  "20"  "21"  "22"
##  [23] "23"  "24"  "25"  "26"  "27"  "28"  "29"  "30"  "31"  "32"  "33"
##  [34] "34"  "35"  "36"  "37"  "38"  "39"  "40"  "41"  "42"  "43"  "44"
##  [45] "45"  "46"  "47"  "48"  "49"  "50"  "51"  "52"  "53"  "54"  "55"
##  [56] "56"  "57"  "58"  "59"  "60"  "61"  "62"  "63"  "64"  "65"  "66"
##  [67] "67"  "68"  "69"  "70"  "71"  "72"  "73"  "74"  "75"  "76"  "77"
##  [78] "78"  "79"  "80"  "81"  "82"  "83"  "84"  "85"  "86"  "87"  "88"
##  [89] "89"  "90"  "91"  "92"  "93"  "94"  "95"  "96"  "97"  "98"  "99"
## [100] "100" "101" "102" "103" "104" "105" "106" "107" "108" "109" "110"
## [111] "111" "112" "113" "114" "115" "116" "117" "118" "119" "120" "121"
```

```
unemp_idx <- seq(as.Date("2007/1/1"), by = "month", length.out =
length(df[,1]))
  head(unemp_idx)
```

```
## [1] "2007-01-01" "2007-02-01" "2007-03-01" "2007-04-01" "2007-05-01"
## [6] "2007-06-01"
```

```
unemp_xts <- xts(df$Value, order.by = unemp_idx)
  str(unemp_xts)
```

```
## An 'xts' object on 2007-01-01/2017-01-01 containing:
##   Data: num [1:121, 1] 5 4.9 4.5 4.3 4.3 4.7 4.9 4.6 4.5 4.4 ...
##   Indexed by objects of class: [Date] TZ: UTC
##   xts Attributes:
##  NULL
```

```
  head(unemp_xts)
```

```
##            [,1]
## 2007-01-01  5.0
## 2007-02-01  4.9
## 2007-03-01  4.5
## 2007-04-01  4.3
## 2007-05-01  4.3
## 2007-06-01  4.7
```

Task 3:

1. Read A. Section 3.2 of "xts: Extensible Time Series" by Jeffrey A. Ryan and Joshua M. Ulrich
2. Follow the following section of this document

Merging and modifying time series

One of the key strengths of `xts` is that it is easy to join data by column and row using a only few different functions. It makes creating time series datasets almost effortless.

The important criterion is that the xts objects must be of identical type (e.g. integer + integer), or be POSIXct dates vector, or be atomic vectors of the same type (e.g. numeric), or be a single NA. It does not work on data.frames with various column types.

The major functions is `merge`. It works like `cbind` or SQL's `join`:

Let's look at an example. It assumes that you are familiar with concepts of inner join, outer join, left join, and right join.

```
library(quantmod)
```

```
## Loading required package: TTR
```

```
## Version 0.4-0 included new data defaults. See ?getSymbols.
```

```
getSymbols("TWTR", src="google")
```

```
## 'getSymbols' currently uses auto.assign=TRUE by default, but will
## use auto.assign=FALSE in 0.5-0. You will still be able to use
```

```
## 'loadSymbols' to automatically load data. getOption("getSymbols.env")
## and getOption("getSymbols.auto.assign") will still be checked for
## alternate defaults.
##
## This message is shown once per session and may be disabled by setting
## options("getSymbols.warning4.0"=FALSE). See ?getSymbols for details.
```

```
## [1] "TWTR"
```

```
head(TWTR)
```

```
##            TWTR.Open TWTR.High TWTR.Low TWTR.Close TWTR.Volume
## 2013-11-07     45.10     50.09    44.00      44.90   117701670
## 2013-11-08     45.93     46.94    40.68      41.65    27925307
## 2013-11-11     40.50     43.00    39.40      42.90    16113941
## 2013-11-12     43.66     43.78    41.83      41.90     6316755
## 2013-11-13     41.03     42.87    40.76      42.60     8688325
## 2013-11-14     42.34     45.67    42.24      44.69    11099433
```

```
str(TWTR)
```

```
## An 'xts' object on 2013-11-07/2017-11-17 containing:
##   Data: num [1:1016, 1:5] 45.1 45.9 40.5 43.7 41 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : NULL
##   ..$ : chr [1:5] "TWTR.Open" "TWTR.High" "TWTR.Low" "TWTR.Close" ...
##   Indexed by objects of class: [Date] TZ: UTC
##   xts Attributes:
## List of 2
##  $ src    : chr "google"
##  $ updated: POSIXct[1:1], format: "2017-11-17 17:47:04"
```

Note that the date obtained from the getSymbols function of teh quantmod library is already an xts object. As such, we can merge it directly with our unemployment rate xts object constructed above. Nevertheless, it is instructive to examine the data using the View() function to ensure that you understand the number of observations resulting from the joined series.

```
# 1. Inner join
TWTR_unemp01 <- merge(unemp_xts, TWTR, join = "inner")
  str(TWTR_unemp01)
```

```
## An 'xts' object on 2014-04-01/2016-12-01 containing:
##   Data: num [1:22, 1:6] 5.9 6.1 6.5 6.3 5.5 5.4 5.1 5.3 5.5 5.6 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : NULL
##   ..$ : chr [1:6] "unemp_xts" "TWTR.Open" "TWTR.High" "TWTR.Low" ...
##   Indexed by objects of class: [Date] TZ: UTC
##   xts Attributes:
##  NULL
```

```
  head(TWTR_unemp01)
```

```
##            unemp_xts TWTR.Open TWTR.High TWTR.Low TWTR.Close TWTR.Volume
## 2014-04-01       5.9     46.71     47.59    46.18      46.98     6916147
## 2014-05-01       6.1     39.01     40.77    38.97      39.09    15759771
## 2014-07-01       6.5     42.06     42.95    41.91      42.05    36019345
## 2014-08-01       6.3     45.01     45.54    43.81      44.13    37194768
## 2014-10-01       5.5     51.08     51.29    49.15      50.06    24733453
```

```
## 2014-12-01       5.4     41.29     41.29    39.00      39.04    22213988
```

```
# 2. Outer join (filling the missing observations with 99999)
# Basic argument use
TWTR_unemp02 <- merge(unemp_xts, TWTR, join = "outer", fill = 99999)
  str(TWTR_unemp02)
```

```
## An 'xts' object on 2007-01-01/2017-11-17 containing:
##   Data: num [1:1115, 1:6] 5 4.9 4.5 4.3 4.3 4.7 4.9 4.6 4.5 4.4 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : NULL
##   ..$ : chr [1:6] "unemp_xts" "TWTR.Open" "TWTR.High" "TWTR.Low" ...
##   Indexed by objects of class: [Date] TZ: UTC
##   xts Attributes:
##  NULL
```

```
  head(TWTR_unemp02)
```

```
##            unemp_xts TWTR.Open TWTR.High TWTR.Low TWTR.Close TWTR.Volume
## 2007-01-01       5.0     99999     99999    99999      99999       99999
## 2007-02-01       4.9     99999     99999    99999      99999       99999
## 2007-03-01       4.5     99999     99999    99999      99999       99999
## 2007-04-01       4.3     99999     99999    99999      99999       99999
## 2007-05-01       4.3     99999     99999    99999      99999       99999
## 2007-06-01       4.7     99999     99999    99999      99999       99999
```

```
  #View(TWTR_unemp02)

# Left join
TWTR_unemp03 <- merge(unemp_xts, TWTR, join = "left", fill = 99999)
  str(TWTR_unemp03)
```

```
## An 'xts' object on 2007-01-01/2017-01-01 containing:
##   Data: num [1:121, 1:6] 5 4.9 4.5 4.3 4.3 4.7 4.9 4.6 4.5 4.4 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : NULL
##   ..$ : chr [1:6] "unemp_xts" "TWTR.Open" "TWTR.High" "TWTR.Low" ...
##   Indexed by objects of class: [Date] TZ: UTC
##   xts Attributes:
##  NULL
```

```
  head(TWTR_unemp03)
```

```
##            unemp_xts TWTR.Open TWTR.High TWTR.Low TWTR.Close TWTR.Volume
## 2007-01-01       5.0     99999     99999    99999      99999       99999
## 2007-02-01       4.9     99999     99999    99999      99999       99999
## 2007-03-01       4.5     99999     99999    99999      99999       99999
## 2007-04-01       4.3     99999     99999    99999      99999       99999
## 2007-05-01       4.3     99999     99999    99999      99999       99999
## 2007-06-01       4.7     99999     99999    99999      99999       99999
```

```
  #View(TWTR_unemp03)

# Right join
TWTR_unemp04 <- merge(unemp_xts, TWTR, join = "right", fill = 99999)
  str(TWTR_unemp04)
```

```
## An 'xts' object on 2013-11-07/2017-11-17 containing:
```

```
##   Data: num [1:1016, 1:6] 99999 99999 99999 99999 99999 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : NULL
##   ..$ : chr [1:6] "unemp_xts" "TWTR.Open" "TWTR.High" "TWTR.Low" ...
##   Indexed by objects of class: [Date] TZ: UTC
##   xts Attributes:
##  NULL
```

```
head(TWTR_unemp04)
```

```
##            unemp_xts TWTR.Open TWTR.High TWTR.Low TWTR.Close TWTR.Volume
## 2013-11-07     99999     45.10     50.09    44.00      44.90   117701670
## 2013-11-08     99999     45.93     46.94    40.68      41.65    27925307
## 2013-11-11     99999     40.50     43.00    39.40      42.90    16113941
## 2013-11-12     99999     43.66     43.78    41.83      41.90     6316755
## 2013-11-13     99999     41.03     42.87    40.76      42.60     8688325
## 2013-11-14     99999     42.34     45.67    42.24      44.69    11099433
```

```
#View(TWTR_unemp04)
```

Missing value imputation

xts also offers methods that allows filling missing values using last or previous observation. Note that I include this simply to point out that this is possible. I by no mean certify that this is the preferred method of imputing missing values in a time series. As I mentioned in live session, the specific method to use in missing value imputation is completely context dependent.

Filling missing values from the last observation

```
# First, let's replace the "99999" values with NA and then exammine the series.

# Let's examine the first few dozen observations with NA
TWTR_unemp02['2013-10-01/2013-12-15'][,1]
```

```
##            unemp_xts
## 2013-10-01       7.0
## 2013-11-01       6.6
## 2013-11-07   99999.0
## 2013-11-08   99999.0
## 2013-11-11   99999.0
## 2013-11-12   99999.0
## 2013-11-13   99999.0
## 2013-11-14   99999.0
## 2013-11-15   99999.0
## 2013-11-18   99999.0
## 2013-11-19   99999.0
## 2013-11-20   99999.0
## 2013-11-21   99999.0
## 2013-11-22   99999.0
## 2013-11-25   99999.0
## 2013-11-26   99999.0
## 2013-11-27   99999.0
## 2013-11-29   99999.0
## 2013-12-01       6.5
## 2013-12-02   99999.0
```

```
## 2013-12-03   99999.0
## 2013-12-04   99999.0
## 2013-12-05   99999.0
## 2013-12-06   99999.0
## 2013-12-09   99999.0
## 2013-12-10   99999.0
## 2013-12-11   99999.0
## 2013-12-12   99999.0
## 2013-12-13   99999.0
```

```
# Replace observations with "99999" with NA and store in a new series
unemp01 <- TWTR_unemp02[, 1]
unemp01['2013-10-01/2013-12-15']
```

```
##            unemp_xts
## 2013-10-01       7.0
## 2013-11-01       6.6
## 2013-11-07   99999.0
## 2013-11-08   99999.0
## 2013-11-11   99999.0
## 2013-11-12   99999.0
## 2013-11-13   99999.0
## 2013-11-14   99999.0
## 2013-11-15   99999.0
## 2013-11-18   99999.0
## 2013-11-19   99999.0
## 2013-11-20   99999.0
## 2013-11-21   99999.0
## 2013-11-22   99999.0
## 2013-11-25   99999.0
## 2013-11-26   99999.0
## 2013-11-27   99999.0
## 2013-11-29   99999.0
## 2013-12-01       6.5
## 2013-12-02   99999.0
## 2013-12-03   99999.0
## 2013-12-04   99999.0
## 2013-12-05   99999.0
## 2013-12-06   99999.0
## 2013-12-09   99999.0
## 2013-12-10   99999.0
## 2013-12-11   99999.0
## 2013-12-12   99999.0
## 2013-12-13   99999.0
```

```
str(unemp01)
```

```
## An 'xts' object on 2007-01-01/2017-11-17 containing:
##   Data: num [1:1115, 1] 5 4.9 4.5 4.3 4.3 4.7 4.9 4.6 4.5 4.4 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : NULL
##   ..$ : chr "unemp_xts"
##   Indexed by objects of class: [Date] TZ: UTC
##   xts Attributes:
##  NULL
```

```
head(unemp01)
```

```
##            unemp_xts
## 2007-01-01       5.0
## 2007-02-01       4.9
## 2007-03-01       4.5
## 2007-04-01       4.3
## 2007-05-01       4.3
## 2007-06-01       4.7
```

```
#TWTR_unemp02[, 1][TWTR_unemp02[, 1] >= 99990] <- NA

unemp02 <- unemp01
unemp02[unemp02 >= 99990] <- NA

cbind(unemp01['2013-10-01/2013-12-15'], unemp02['2013-10-01/2013-12-15'])
```

```
##            unemp_xts unemp_xts.1
## 2013-10-01       7.0         7.0
## 2013-11-01       6.6         6.6
## 2013-11-07   99999.0          NA
## 2013-11-08   99999.0          NA
## 2013-11-11   99999.0          NA
## 2013-11-12   99999.0          NA
## 2013-11-13   99999.0          NA
## 2013-11-14   99999.0          NA
## 2013-11-15   99999.0          NA
## 2013-11-18   99999.0          NA
## 2013-11-19   99999.0          NA
## 2013-11-20   99999.0          NA
## 2013-11-21   99999.0          NA
## 2013-11-22   99999.0          NA
## 2013-11-25   99999.0          NA
## 2013-11-26   99999.0          NA
## 2013-11-27   99999.0          NA
## 2013-11-29   99999.0          NA
## 2013-12-01       6.5         6.5
## 2013-12-02   99999.0          NA
## 2013-12-03   99999.0          NA
## 2013-12-04   99999.0          NA
## 2013-12-05   99999.0          NA
## 2013-12-06   99999.0          NA
## 2013-12-09   99999.0          NA
## 2013-12-10   99999.0          NA
## 2013-12-11   99999.0          NA
## 2013-12-12   99999.0          NA
## 2013-12-13   99999.0          NA
```

```
# Impute the missing values (stored as NA) with the last observation
#TWTR_unemp02_v2a <- na.locf(TWTR_unemp02[,1],
#                            na.rm = TRUE, fromLast = TRUE)
unemp03 <- unemp02
unemp03 <- na.locf(unemp03, na.rm = TRUE, fromLast = FALSE)

# Examine the pre- and post-imputed series
```

```
#cbind(TWTR_unemp02['2013-10-01/2013-12-30'][,1], TWTR_unemp02_v2a['2013-10-01/2013-12-15'])
cbind(unemp01['2013-10-01/2013-12-15'], unemp02['2013-10-01/2013-12-15'],
unemp03['2013-10-01/2013-12-15'])
```

```
##            unemp_xts unemp_xts.1 unemp_xts.2
## 2013-10-01       7.0         7.0         7.0
## 2013-11-01       6.6         6.6         6.6
## 2013-11-07   99999.0          NA         6.6
## 2013-11-08   99999.0          NA         6.6
## 2013-11-11   99999.0          NA         6.6
## 2013-11-12   99999.0          NA         6.6
## 2013-11-13   99999.0          NA         6.6
## 2013-11-14   99999.0          NA         6.6
## 2013-11-15   99999.0          NA         6.6
## 2013-11-18   99999.0          NA         6.6
## 2013-11-19   99999.0          NA         6.6
## 2013-11-20   99999.0          NA         6.6
## 2013-11-21   99999.0          NA         6.6
## 2013-11-22   99999.0          NA         6.6
## 2013-11-25   99999.0          NA         6.6
## 2013-11-26   99999.0          NA         6.6
## 2013-11-27   99999.0          NA         6.6
## 2013-11-29   99999.0          NA         6.6
## 2013-12-01       6.5         6.5         6.5
## 2013-12-02   99999.0          NA         6.5
## 2013-12-03   99999.0          NA         6.5
## 2013-12-04   99999.0          NA         6.5
## 2013-12-05   99999.0          NA         6.5
## 2013-12-06   99999.0          NA         6.5
## 2013-12-09   99999.0          NA         6.5
## 2013-12-10   99999.0          NA         6.5
## 2013-12-11   99999.0          NA         6.5
## 2013-12-12   99999.0          NA         6.5
## 2013-12-13   99999.0          NA         6.5
```

Another missing value imputation method is linear interpolation, which can also be easily done in xts objects. In the following example, we use linear interpolation to fill in the NA in between months. The result is stored in unemp04. Note in the following the different ways of imputing missing values.

```
unemp04 <- unemp02
#unemp04['2013-10-01/2014-02-01']
unemp04 <- na.approx(unemp04, maxgap=31)
#unemp04['2013-10-01/2014-02-01']

round(cbind(unemp01['2013-10-01/2013-12-15'], unemp02['2013-10-01/2013-12-15'],
unemp03['2013-10-01/2013-12-15'],
unemp04['2013-10-01/2013-12-15']),2)
```

```
##            unemp_xts unemp_xts.1 unemp_xts.2 unemp_xts.3
## 2013-10-01       7.0         7.0         7.0        7.00
## 2013-11-01       6.6         6.6         6.6        6.60
## 2013-11-07   99999.0          NA         6.6        6.58
## 2013-11-08   99999.0          NA         6.6        6.58
## 2013-11-11   99999.0          NA         6.6        6.57
## 2013-11-12   99999.0          NA         6.6        6.56
```

```
## 2013-11-13   99999.0          NA         6.6        6.56
## 2013-11-14   99999.0          NA         6.6        6.56
## 2013-11-15   99999.0          NA         6.6        6.55
## 2013-11-18   99999.0          NA         6.6        6.54
## 2013-11-19   99999.0          NA         6.6        6.54
## 2013-11-20   99999.0          NA         6.6        6.54
## 2013-11-21   99999.0          NA         6.6        6.53
## 2013-11-22   99999.0          NA         6.6        6.53
## 2013-11-25   99999.0          NA         6.6        6.52
## 2013-11-26   99999.0          NA         6.6        6.52
## 2013-11-27   99999.0          NA         6.6        6.51
## 2013-11-29   99999.0          NA         6.6        6.51
## 2013-12-01       6.5         6.5         6.5        6.50
## 2013-12-02   99999.0          NA         6.5        6.52
## 2013-12-03   99999.0          NA         6.5        6.53
## 2013-12-04   99999.0          NA         6.5        6.55
## 2013-12-05   99999.0          NA         6.5        6.56
## 2013-12-06   99999.0          NA         6.5        6.58
## 2013-12-09   99999.0          NA         6.5        6.63
## 2013-12-10   99999.0          NA         6.5        6.65
## 2013-12-11   99999.0          NA         6.5        6.66
## 2013-12-12   99999.0          NA         6.5        6.68
## 2013-12-13   99999.0          NA         6.5        6.69
```

Calculate difference in time series

A very common operation on time series is to take a difference of the series to transform a non-stationary serier to a stationary series. First order differencing takes the form $x(t) - x(t-k)$ where k denotes the number of time lags. Higher order differences are simply the reapplication of a difference to each prior result (like a second derivative or a difference of the difference).

Let's use the `unemp_xts` series as examples:

```
str(unemp_xts)
```

```
## An 'xts' object on 2007-01-01/2017-01-01 containing:
##   Data: num [1:121, 1] 5 4.9 4.5 4.3 4.3 4.7 4.9 4.6 4.5 4.4 ...
##   Indexed by objects of class: [Date] TZ: UTC
##   xts Attributes:
##  NULL
```

```
head(unemp_xts)
```

```
##            [,1]
## 2007-01-01  5.0
## 2007-02-01  4.9
## 2007-03-01  4.5
## 2007-04-01  4.3
## 2007-05-01  4.3
## 2007-06-01  4.7
```

```
head(diff(unemp_xts, lag = 1, difference = 1, log = FALSE, na.pad = TRUE))
```

```
##            [,1]
## 2007-01-01   NA
## 2007-02-01 -0.1
```

```
## 2007-03-01 -0.4
## 2007-04-01 -0.2
## 2007-05-01  0.0
## 2007-06-01  0.4
```

```
# calculate the first difference of AirPass using lag and subtraction
#AirPass - lag(AirPass, k = 1)

# calculate the first order 12-month difference if AirPass
head(diff(unemp_xts, lag = 12, differences = 1))
```

```
##            [,1]
## 2007-01-01   NA
## 2007-02-01   NA
## 2007-03-01   NA
## 2007-04-01   NA
## 2007-05-01   NA
## 2007-06-01   NA
```

Task 4:

1. Read A. Section 3.4 of "xts: Extensible Time Series" by Jeffrey A. Ryan and Joshua M. Ulrich

   B. the following questions in "xts FAQ" a. I am using apply() to run a custom function on my xts series. Why the returned matrix has different dimensions than the original one?

2. Follow the following two sections of this document

Apply various functions to time series

The family of `apply` functions perhaps is one of the most powerful R function families. In time series, `xts` provides `period.apply`, which takes (1) a time series, (2) an index of endpoints, and (3) a function to apply. It takes the following general form:

```
period.apply(x, INDEX, FUN, ...)
```

As an example, we use the Twitter stock price series (to be precise, the daily closing price), create an index storing the points corresopnding to the weeks of the daily series, and apply functions to calculate the weekly mean.

```
# Step 1: Identify the endpoints; in this case, we use weekly time interval. That is, we extract the en

#View(TWTR)
head(TWTR)
```

```
##            TWTR.Open TWTR.High TWTR.Low TWTR.Close TWTR.Volume
## 2013-11-07     45.10     50.09    44.00      44.90   117701670
## 2013-11-08     45.93     46.94    40.68      41.65    27925307
## 2013-11-11     40.50     43.00    39.40      42.90    16113941
## 2013-11-12     43.66     43.78    41.83      41.90     6316755
## 2013-11-13     41.03     42.87    40.76      42.60     8688325
## 2013-11-14     42.34     45.67    42.24      44.69    11099433
```

```
TWTR_ep <- endpoints(TWTR[,4], on = "weeks")
#TWTR_ep
```

```
# Step 2: Calculate the weekly mean
TWTR.Close_weeklyMean <- period.apply(TWTR[, 4], INDEX = TWTR_ep, FUN = mean)
head(round(TWTR.Close_weeklyMean,2),8)
```

```
##            TWTR.Close
## 2013-11-08      43.27
## 2013-11-15      43.21
## 2013-11-22      41.40
## 2013-11-29      40.43
## 2013-12-06      43.28
## 2013-12-13      53.56
## 2013-12-20      57.21
## 2013-12-27      67.89
```

The power of the apply function really comes with the use of custom-defined function. For instance, we can easily

```
f <- function(x) {
  mean <- mean(x)
  quantile <- quantile(x,c(0.05,0.25,0.50,0.75,0.95))
  sd <- sd(x)

  result <- c(mean, sd, quantile)
  return(result)
}
head(round(period.apply(TWTR[, 4], INDEX = TWTR_ep, FUN = f),2),10)
```

```
##                          5%   25%   50%   75%   95%
## 2013-11-08 43.27 2.30 41.81 42.46 43.27 44.09 44.74
## 2013-11-15 43.21 1.11 42.04 42.60 42.90 43.98 44.55
## 2013-11-22 41.40 0.48 41.01 41.05 41.14 41.75 42.00
## 2013-11-29 40.43 1.07 39.23 39.90 40.54 41.07 41.47
## 2013-12-06 43.28 2.14 40.90 41.37 43.69 44.95 45.49
## 2013-12-13 53.56 3.75 49.71 51.99 52.34 55.33 58.27
## 2013-12-20 57.21 1.71 55.70 56.45 56.61 57.49 59.51
## 2013-12-27 67.89 4.55 63.87 64.34 67.25 70.80 72.81
## 2014-01-03 65.16 3.84 60.98 62.86 65.58 67.88 68.78
## 2014-01-10 60.22 3.86 57.01 57.05 59.29 61.46 65.32
```

Calculate basic rolling statistics of series by month

Using `rollapply`, one can calculate rolling statistics of a series:

```
# Calculate rolling mean over a 10-day period and print it with the original series
head(cbind(TWTR[,4], rollapply(TWTR[, 4], 10, FUN = mean, na.rm = TRUE)),15)
```

```
##            TWTR.Close TWTR.Close.1
## 2013-11-07      44.90           NA
## 2013-11-08      41.65           NA
## 2013-11-11      42.90           NA
## 2013-11-12      41.90           NA
## 2013-11-13      42.60           NA
## 2013-11-14      44.69           NA
## 2013-11-15      43.98           NA
```

```
## 2013-11-18      41.14           NA
## 2013-11-19      41.75           NA
## 2013-11-20      41.05       42.656
## 2013-11-21      42.06       42.372
## 2013-11-22      41.00       42.307
## 2013-11-25      39.06       41.923
## 2013-11-26      40.18       41.751
## 2013-11-27      40.90       41.581
```

Task 5:

1. Read AMAZ.csv and UMCSENT.csv into R as R DataFrames

5.1 Answer

```
library(xts)
amazon.data = data.frame(read.csv('AMAZ.csv'))
umcsent.data = data.frame(read.csv('UMCSENT.csv'))
```

2. Convert them to xts objects

5.2 Answer

```
amazon.xts = xts(amazon.data[,2:6], order.by = as.Date(amazon.data$Index))
umcsent.xts = xts(umcsent.data[,2], order.by = as.Date(umcsent.data$Index))

head(amazon.xts)
```

```
##            AMAZ.Open AMAZ.High AMAZ.Low AMAZ.Close AMAZ.Volume
## 2007-01-03      20.0      20.0     16.0       16.0         650
## 2007-01-04      20.0      20.0     20.0       20.0          67
## 2007-01-08      19.2      22.0     19.2       22.0        1801
## 2007-01-09      22.0      22.0     20.8       20.8         356
## 2007-01-10      20.8      20.8     20.8       20.8         438
## 2007-01-11      20.8      21.6     20.8       21.6        2318
```

```
head(umcsent.xts)
```

```
##            [,1]
## 1978-01-01 83.7
## 1978-02-01 84.3
## 1978-03-01 78.8
## 1978-04-01 81.6
## 1978-05-01 82.9
## 1978-06-01 80.0
```

3. Merge the two set of series together, perserving all of the obserbvations in both set of series
    a. fill all of the missing values of the UMCSENT series with -9999

5.3 Answer

5.3.a Answer

```
# First merge the series
UMCSENT01 = merge(amazon.xts, umcsent.xts, join = "outer")

# We are asked to fill the missing values of UMCSENT
UMCSENT01$umcsent.xts[is.na(UMCSENT01$umcsent.xts)] <- -9999

# Display a summary
summary(UMCSENT01)
```

```
##       Index              AMAZ.Open       AMAZ.High         AMAZ.Low
##  Min.   :1978-01-01   Min.   : 0.16   Min.   : 0.200   Min.   : 0.080
##  1st Qu.:2007-04-19   1st Qu.: 0.80   1st Qu.: 0.800   1st Qu.: 0.720
##  Median :2009-01-06   Median : 1.08   Median : 1.120   Median : 1.000
##  Mean   :2006-03-11   Mean   : 4.83   Mean   : 4.954   Mean   : 4.696
##  3rd Qu.:2010-08-04   3rd Qu.: 6.00   3rd Qu.: 6.400   3rd Qu.: 5.650
##  Max.   :2017-09-01   Max.   :24.40   Max.   :26.000   Max.   :24.400
##                       NA's   :699     NA's   :699      NA's   :699
##    AMAZ.Close      AMAZ.Volume     umcsent.xts
##  Min.   : 0.080   Min.   :    0   Min.   :-9999.0
##  1st Qu.: 0.620   1st Qu.:   25   1st Qu.:-9999.0
##  Median : 1.000   Median :  312   Median :-9999.0
##  Mean   : 4.129   Mean   : 1499   Mean   :-7027.8
##  3rd Qu.: 4.000   3rd Qu.: 1250   3rd Qu.:   70.4
##  Max.   :25.600   Max.   :68900   Max.   :  112.0
##  NA's   :440      NA's   :440
```

b. then create a new series, named UMCSENT02, from the original UMCSENT series replace all of the -999

5.3.b Answer

```
# Transform -9999 records in umcsent.xts to NA
UMCSENT02 = UMCSENT01
UMCSENT02$umcsent.xts[UMCSENT02$umcsent.xts <= -9999] <- NA

# Display a summary
summary(UMCSENT02)
```

```
##       Index              AMAZ.Open       AMAZ.High         AMAZ.Low
##  Min.   :1978-01-01   Min.   : 0.16   Min.   : 0.200   Min.   : 0.080
##  1st Qu.:2007-04-19   1st Qu.: 0.80   1st Qu.: 0.800   1st Qu.: 0.720
##  Median :2009-01-06   Median : 1.08   Median : 1.120   Median : 1.000
##  Mean   :2006-03-11   Mean   : 4.83   Mean   : 4.954   Mean   : 4.696
##  3rd Qu.:2010-08-04   3rd Qu.: 6.00   3rd Qu.: 6.400   3rd Qu.: 5.650
##  Max.   :2017-09-01   Max.   :24.40   Max.   :26.000   Max.   :24.400
##                       NA's   :699     NA's   :699      NA's   :699
##    AMAZ.Close      AMAZ.Volume     umcsent.xts
##  Min.   : 0.080   Min.   :    0   Min.   : 51.70
##  1st Qu.: 0.620   1st Qu.:   25   1st Qu.: 76.10
##  Median : 1.000   Median :  312   Median : 89.30
```

```
##  Mean   : 4.129   Mean   : 1499   Mean   : 85.69
##  3rd Qu.: 4.000   3rd Qu.: 1250   3rd Qu.: 94.30
##  Max.   :25.600   Max.   :68900   Max.   :112.00
##  NA's   :440      NA's   :440     NA's   :1142
```

c. then create a new series, named UMCSENT03, and replace the NAs with the last observation

5.3.c Answer

```
# na.locf is designed exactly for this: https://www.rdocumentation.org/packages/zoo/versions/1.8-0/topi
# "Generic function for replacing each NA with the most recent non-NA prior to it."
UMCSENT03 = UMCSENT02
UMCSENT03$umcsent.xts <- na.locf(UMCSENT03$umcsent.xts, fromLast = FALSE, na.rm = TRUE)

# Display a summary
summary(UMCSENT03)
```

```
##       Index              AMAZ.Open       AMAZ.High         AMAZ.Low
##  Min.   :1978-01-01   Min.   : 0.16   Min.   : 0.200   Min.   : 0.080
##  1st Qu.:2007-04-19   1st Qu.: 0.80   1st Qu.: 0.800   1st Qu.: 0.720
##  Median :2009-01-06   Median : 1.08   Median : 1.120   Median : 1.000
##  Mean   :2006-03-11   Mean   : 4.83   Mean   : 4.954   Mean   : 4.696
##  3rd Qu.:2010-08-04   3rd Qu.: 6.00   3rd Qu.: 6.400   3rd Qu.: 5.650
##  Max.   :2017-09-01   Max.   :24.40   Max.   :26.000   Max.   :24.400
##                       NA's   :699     NA's   :699      NA's   :699
##    AMAZ.Close      AMAZ.Volume     umcsent.xts
##  Min.   : 0.080   Min.   :    0   Min.   : 51.70
##  1st Qu.: 0.620   1st Qu.:   25   1st Qu.: 67.40
##  Median : 1.000   Median :  312   Median : 73.60
##  Mean   : 4.129   Mean   : 1499   Mean   : 75.29
##  3rd Qu.: 4.000   3rd Qu.: 1250   3rd Qu.: 83.40
##  Max.   :25.600   Max.   :68900   Max.   :112.00
##  NA's   :440      NA's   :440
```

d. then create a new series, named UMCSENT04, and replace the NAs using linear interpolation.

5.3.d Answer

```
# na.approx does exactly what is asked: https://www.rdocumentation.org/packages/zoo/versions/1.8-0/topi
UMCSENT04 = UMCSENT02
UMCSENT04$umcsent.xts <- na.approx(UMCSENT04$umcsent.xts)

# Display a summary
summary(UMCSENT04)
```

```
##       Index              AMAZ.Open       AMAZ.High         AMAZ.Low
##  Min.   :1978-01-01   Min.   : 0.16   Min.   : 0.200   Min.   : 0.080
##  1st Qu.:2007-04-19   1st Qu.: 0.80   1st Qu.: 0.800   1st Qu.: 0.720
##  Median :2009-01-06   Median : 1.08   Median : 1.120   Median : 1.000
##  Mean   :2006-03-11   Mean   : 4.83   Mean   : 4.954   Mean   : 4.696
##  3rd Qu.:2010-08-04   3rd Qu.: 6.00   3rd Qu.: 6.400   3rd Qu.: 5.650
##  Max.   :2017-09-01   Max.   :24.40   Max.   :26.000   Max.   :24.400
##                       NA's   :699     NA's   :699      NA's   :699
```

```
##    AMAZ.Close      AMAZ.Volume     umcsent.xts
##  Min.   : 0.080   Min.   :    0   Min.   : 51.70
##  1st Qu.: 0.620   1st Qu.:   25   1st Qu.: 67.50
##  Median : 1.000   Median :  312   Median : 73.40
##  Mean   : 4.129   Mean   : 1499   Mean   : 75.21
##  3rd Qu.: 4.000   3rd Qu.: 1250   3rd Qu.: 83.40
##  Max.   :25.600   Max.   :68900   Max.   :112.00
##  NA's   :440      NA's   :440
```

e. Print out some observations to ensure that your merge as well as the missing value imputation are do

5.3.e Answer

Lets first get the number of observations:

```
length(UMCSENT01[,1])
```

```
## [1] 1619
```

There are 1619 observations. However, printing the first results does not make sense, since the umcsent series starts before than the amazon series.

We choose to display the latest records where amazon observations are available.

5.3.a display

```
tail(UMCSENT01[!is.na(UMCSENT01$AMAZ.Open)])
```

```
##            AMAZ.Open AMAZ.High AMAZ.Low AMAZ.Close AMAZ.Volume umcsent.xts
## 2013-01-04      0.88      0.88     0.80       0.80        3850       -9999
## 2013-01-07      0.80      1.00     0.80       1.00        2715       -9999
## 2013-01-08      0.80      0.80     0.68       0.68        4668       -9999
## 2013-01-09      0.88      0.88     0.80       0.80        2750       -9999
## 2013-01-11      0.80      0.80     0.80       0.80        3000       -9999
## 2013-01-15      0.68      0.68     0.68       0.68        1000       -9999
```

5.3.b display

```
tail(UMCSENT02[!is.na(UMCSENT02$AMAZ.Open)])
```

```
##            AMAZ.Open AMAZ.High AMAZ.Low AMAZ.Close AMAZ.Volume umcsent.xts
## 2013-01-04      0.88      0.88     0.80       0.80        3850          NA
## 2013-01-07      0.80      1.00     0.80       1.00        2715          NA
## 2013-01-08      0.80      0.80     0.68       0.68        4668          NA
## 2013-01-09      0.88      0.88     0.80       0.80        2750          NA
## 2013-01-11      0.80      0.80     0.80       0.80        3000          NA
## 2013-01-15      0.68      0.68     0.68       0.68        1000          NA
```

5.3.c display

```
tail(UMCSENT03[!is.na(UMCSENT03$AMAZ.Open)])
```

```
##            AMAZ.Open AMAZ.High AMAZ.Low AMAZ.Close AMAZ.Volume umcsent.xts
## 2013-01-04      0.88      0.88     0.80       0.80        3850        73.8
## 2013-01-07      0.80      1.00     0.80       1.00        2715        73.8
## 2013-01-08      0.80      0.80     0.68       0.68        4668        73.8
```

```
## 2013-01-09      0.88      0.88     0.80       0.80        2750        73.8
## 2013-01-11      0.80      0.80     0.80       0.80        3000        73.8
## 2013-01-15      0.68      0.68     0.68       0.68        1000        73.8
```

5.3.d display

```
tail(UMCSENT04[!is.na(UMCSENT04$AMAZ.Open)])
```

```
##            AMAZ.Open AMAZ.High AMAZ.Low AMAZ.Close AMAZ.Volume umcsent.xts
## 2013-01-04      0.88      0.88     0.80       0.80        3850    74.16774
## 2013-01-07      0.80      1.00     0.80       1.00        2715    74.53548
## 2013-01-08      0.80      0.80     0.68       0.68        4668    74.65806
## 2013-01-09      0.88      0.88     0.80       0.80        2750    74.78065
## 2013-01-11      0.80      0.80     0.80       0.80        3000    75.02581
## 2013-01-15      0.68      0.68     0.68       0.68        1000    75.51613
```

4. Calculate the daily return of the Amazon closing price (AMAZ.close), where daily return is defined as $(x(t) - x(t-1))/x(t-1)$. Plot the daily return series.

```
head(amazon.data)
```

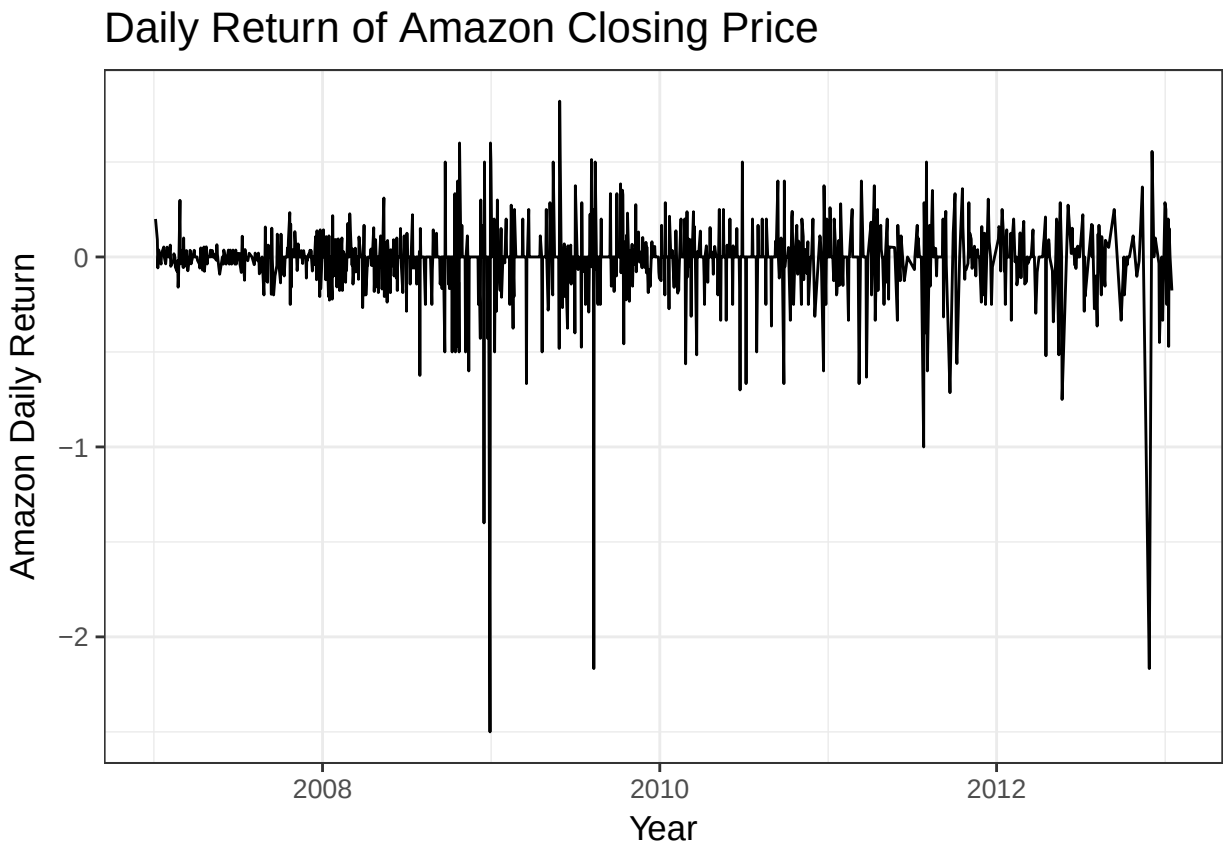
```
##        Index AMAZ.Open AMAZ.High AMAZ.Low AMAZ.Close AMAZ.Volume
## 1 2007-01-03      20.0      20.0     16.0       16.0         650
## 2 2007-01-04      20.0      20.0     20.0       20.0          67
## 3 2007-01-08      19.2      22.0     19.2       22.0        1801
## 4 2007-01-09      22.0      22.0     20.8       20.8         356
## 5 2007-01-10      20.8      20.8     20.8       20.8         438
## 6 2007-01-11      20.8      21.6     20.8       21.6        2318
```

```
AMAZ.Returns = c(NA, diff(amazon.data$AMAZ.Close, lag = 1, difference = 1, log = FALSE, na.pad = TRUE))

amazon.data = cbind(amazon.data, AMAZ.Returns)
amazon.xts = xts(amazon.data[,2:7], order.by = as.Date(amazon.data$Index))


library(ggplot2)
theme_set(theme_bw(base_size = 13))
ggp <- ggplot(amazon.xts, aes(x=Index, y=AMAZ.Returns))
ggp + geom_line() + ylab("Amazon Daily Return")+xlab("Year")+ggtitle("Daily Return of Amazon Closing Pr
```

```
## Warning: Removed 1 rows containing missing values (geom_path).
```



5. Create a 20-day and a 50-day rolling mean series from the AMAZ.close series.

```
amaz.close = cbind(amazon.xts[,4], rollapply(amazon.xts[, 4], 20, FUN = mean, na.rm = TRUE), rollapply(a
colnames(amaz.close) = c('ClosingPrice', 'TwentyDayRolling', 'FiftyDayRolling')
tail(amaz.close)
```

```
##            ClosingPrice TwentyDayRolling FiftyDayRolling
## 2013-01-04         0.80            0.969          1.1512
## 2013-01-07         1.00            0.969          1.1432
## 2013-01-08         0.68            0.955          1.1304
## 2013-01-09         0.80            0.947          1.1200
## 2013-01-11         0.80            0.939          1.1080
## 2013-01-15         0.68            0.919          1.0936
```

```
df1 <- data.frame(time=index(amaz.close), n=amaz.close$ClosingPrice, Value="Original Values", size=1)
colnames(df1) <- c("time", "n", "Value", "size")
df2 <- data.frame(time=index(amaz.close), n=amaz.close$TwentyDayRolling, Value="20-day Rolling Mean", si
colnames(df2) <- c("time", "n", "Value", "size")
df3 <- data.frame(time=index(amaz.close), n=amaz.close$FiftyDayRolling, Value="50-day Rolling Mean", siz
colnames(df3) <- c("time", "n", "Value", "size")
df.combined <- rbind(df1, df2, df3)



ggp <- ggplot(df.combined, aes(x=time, y=n, group=Value, color=Value, size=size))
ggp + geom_line() +
  scale_color_manual(values=c("black", "blue", "red"))+
```

```
  ylab("Closing Price")+
  xlab("Year")+ggtitle("Amazon Closing Price")+
  scale_size_continuous(range=c(.5,1), guide='none')+
  theme(legend.position="top")
```

```
## Warning: Removed 68 rows containing missing values (geom_path).
```