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

Heather Feinstein, Himal Suthar, Daniel Vanlunen

Instructions:

- **Due Date: Monday of Week 11 4p.m. Pacific Time**
- **Page limit of the pdf report for Question 1: 12 (not include title and the table of content pag**
- Use the margin, linespace, and font size specification below:
  - fontsize=11pt
  - margin=1in
  - line_spacing=single
- Submission:
  - Each group makes one submission to Github; please have one of your team members made the submission
  - Submit 2 files:
    1. A pdf file including 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
  - Use the following file-naming convensation; fail to do so will receive 10% reduction in the grade:
    - FirstNameLastName1_FirstNameLastName2_FirstNameLastName3_LabNumber.fileExtension
    - For example, if you have three students in the group for Lab Z, and their names are Gerard Kelley, Steve Yang, and Jeffrey Yau, then you should name your file the following
      - GerardKelley_SteveYang_JeffreyYau_LabZ.Rmd
      - GerardKelley_SteveYang_JeffreyYau_LabZ.pdf
  - Although it sounds obvious, please write the name of each members of your group on page 1 of your pdf and Rmd files.
  - This lab can be completed in a group of up to 3 students in your session. Students are encouraged to work in a group for the lab.
- Other general guidelines:
  - For statistical methods that we cover in this course, use only the R libraries and functions that are covered in this course. If you use libraries and functions for statistical modeling that we have not covered, you have to provide (1) explanation of why such libraries and functions are used instead and (2) reference to the library documentation. Lacking the explanation and reference to the documentation will result in a score of zero for the corresponding question.
- Students are expected to act with regards to UC Berkeley Academic Integrity.

Question 1: Forecasting using a SARIMA model

ECOMPCTNSA.csv, contains quarterly data of E-Commerce Retail Sales as a Percent of Total Sales. The data can be found at: https://fred.stlouisfed.org/series/ECOMPCTNSA.

Build a Seasonal ARIMA model and generate quarterly forecast for 2017. Make sure you use all the steps of building a univariate time series model between lecture 6 and 9, such as checking the raw data, conducting a thorough EDA, justifying all modeling decisions (including transformation), testing model assumptions, and clearly articulating why you chose your given model. Measure and discuss your model's performance. Use both in-sample and out-of-sample model performance. When estimating your model, exclude the series from 2015 and 2016. For the out-of-sample forecast, measure your model's performance in forecasting the quarterly E-Commerce retail sales in 2015 and 2016. Discuss the model performance. Also forecast beyond the observed time-period of the series. Specifically, generate quarterly forecast for 2017.

EDA

First we load and examine the raw data.

```
df <- read_csv("ECOMPCTNSA.csv")
```

```
## Parsed with column specification:
## cols(
##   DATE = col_date(format = ""),
##   ECOMPCTNSA = col_double()
## )
```

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

```
## # A tibble: 6 x 2
##   DATE       ECOMPCTNSA
##   <date>          <dbl>
## 1 1999-10-01        0.7
## 2 2000-01-01        0.8
## 3 2000-04-01        0.8
## 4 2000-07-01        0.9
## 5 2000-10-01        1.1
## 6 2001-01-01        1.1
```

```
summary(df)
```

```
##       DATE              ECOMPCTNSA
##  Min.   :1999-10-01   Min.   :0.700
##  1st Qu.:2004-01-01   1st Qu.:2.000
##  Median :2008-04-01   Median :3.600
##  Mean   :2008-03-31   Mean   :3.835
##  3rd Qu.:2012-07-01   3rd Qu.:5.300
##  Max.   :2016-10-01   Max.   :9.500
```

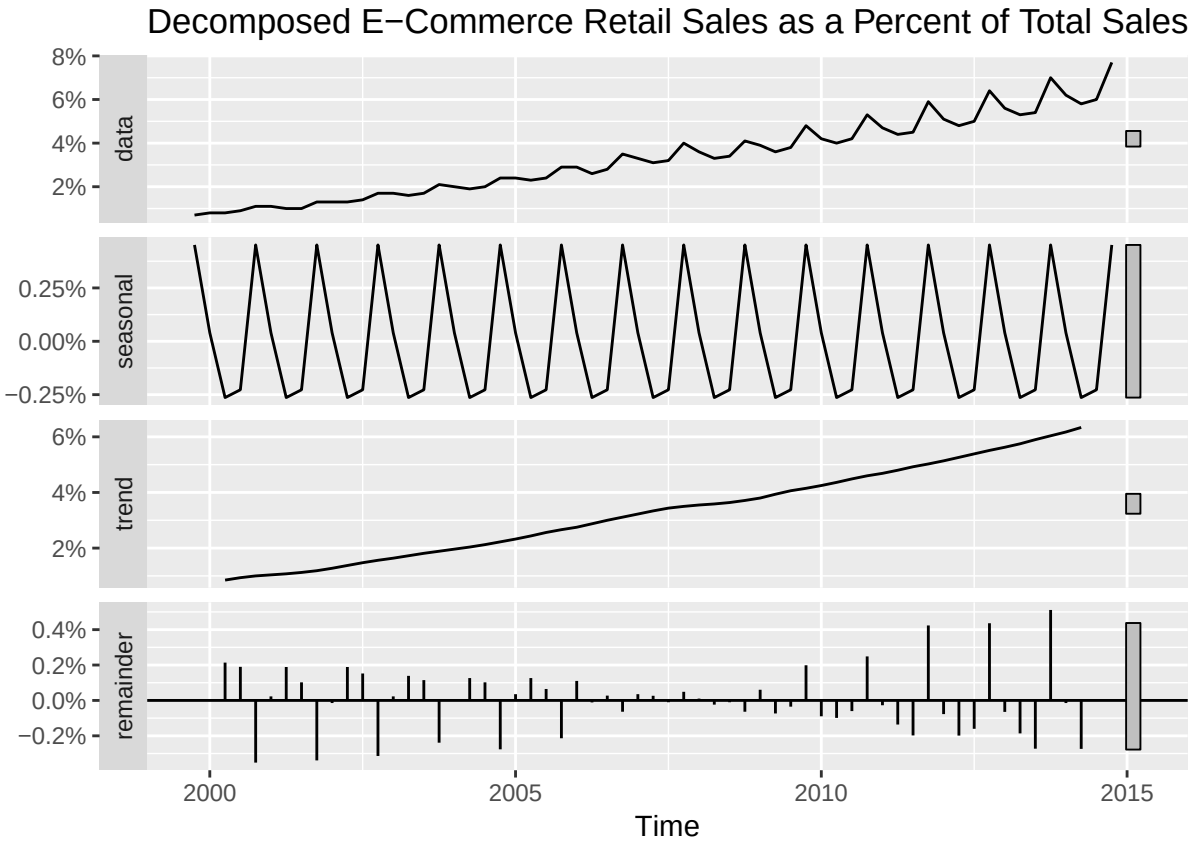
```
describe(df)
```

```
## df
##
##  2  Variables      69  Observations
## ---------------------------------------------------------------------------
## DATE
##        n  missing distinct
##       69        0       69
##
## lowest : 1999-10-01 2000-01-01 2000-04-01 2000-07-01 2000-10-01
## highest: 2015-10-01 2016-01-01 2016-04-01 2016-07-01 2016-10-01
## ---------------------------------------------------------------------------
## ECOMPCTNSA
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##       69        0       50        1    3.835    2.524     0.94     1.10
##      .25      .50      .75      .90      .95
##     2.00     3.60     5.30     6.92     7.70
##
## lowest : 0.7 0.8 0.9 1.0 1.1, highest: 7.0 7.5 7.7 8.7 9.5
## ---------------------------------------------------------------------------
```

```
# Convert it into a time seriea object
series1 <- ts(df$ECOMPCTNSA/100, start=c(1999,4), frequency = 4)
```

There are no missing values. The data is quarterly from 1999 Q4 to 2016 Q4 with no missing quarters in between. The data lie between 0.7 and 9.5%.

```
# question requests fits using pre-2015 data
series1_train <- series1 %>% window(end=c(2014,4))
series1_test <- series1 %>% window(start=2015)

autoplot(decompose(series1_train)) +
  labs(title="Decomposed E-Commerce Retail Sales as a Percent of Total Sales") +
  scale_y_continuous(labels = scales::percent)
```

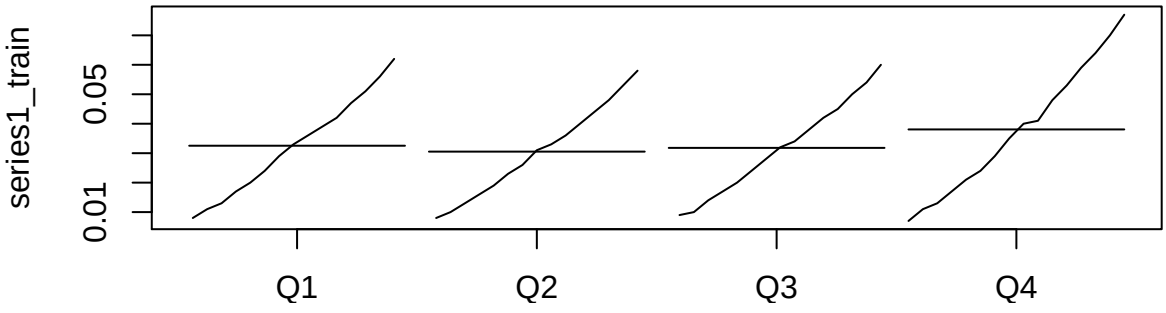



```
series1 %>% ndiffs()
```

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

There is a clear upward trend in the data with fairly strong seasonal variation. It also has an increasing variance over time. The variance could be stablized by a transform, but could also be stablized by differencing as well.

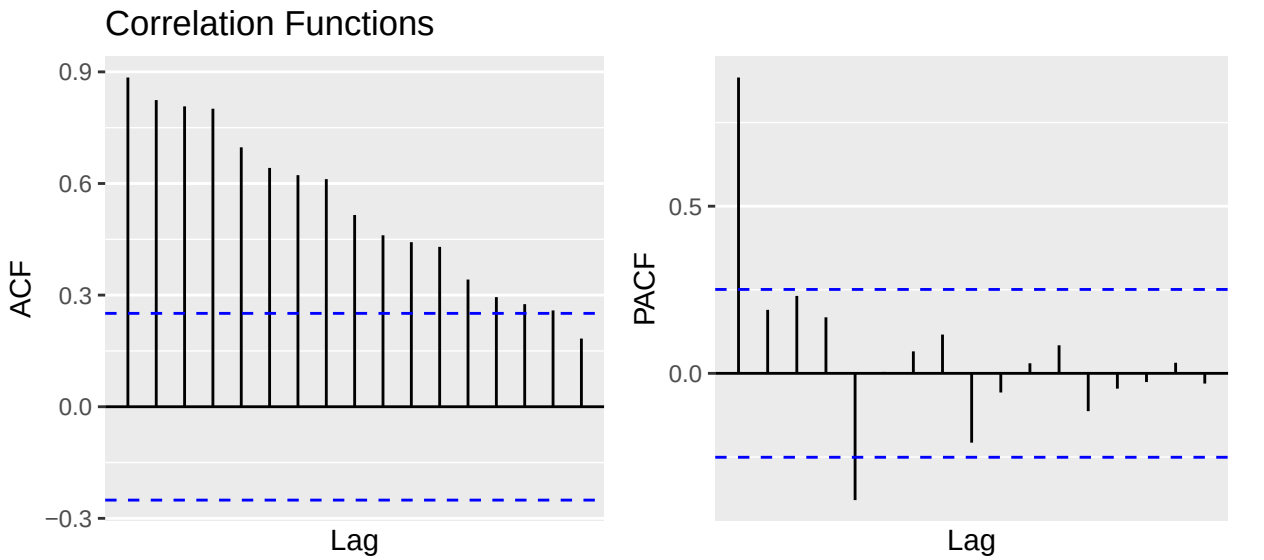
```
monthplot(series1_train)
```



The seasonal trend peaks in Q4.

```
p1 = autoplot(acf(series1_train, plot = FALSE)) +
  ggtitle("Correlation Functions")

p2 = autoplot(pacf(series1_train, plot = FALSE)) +
  ggtitle("")
```

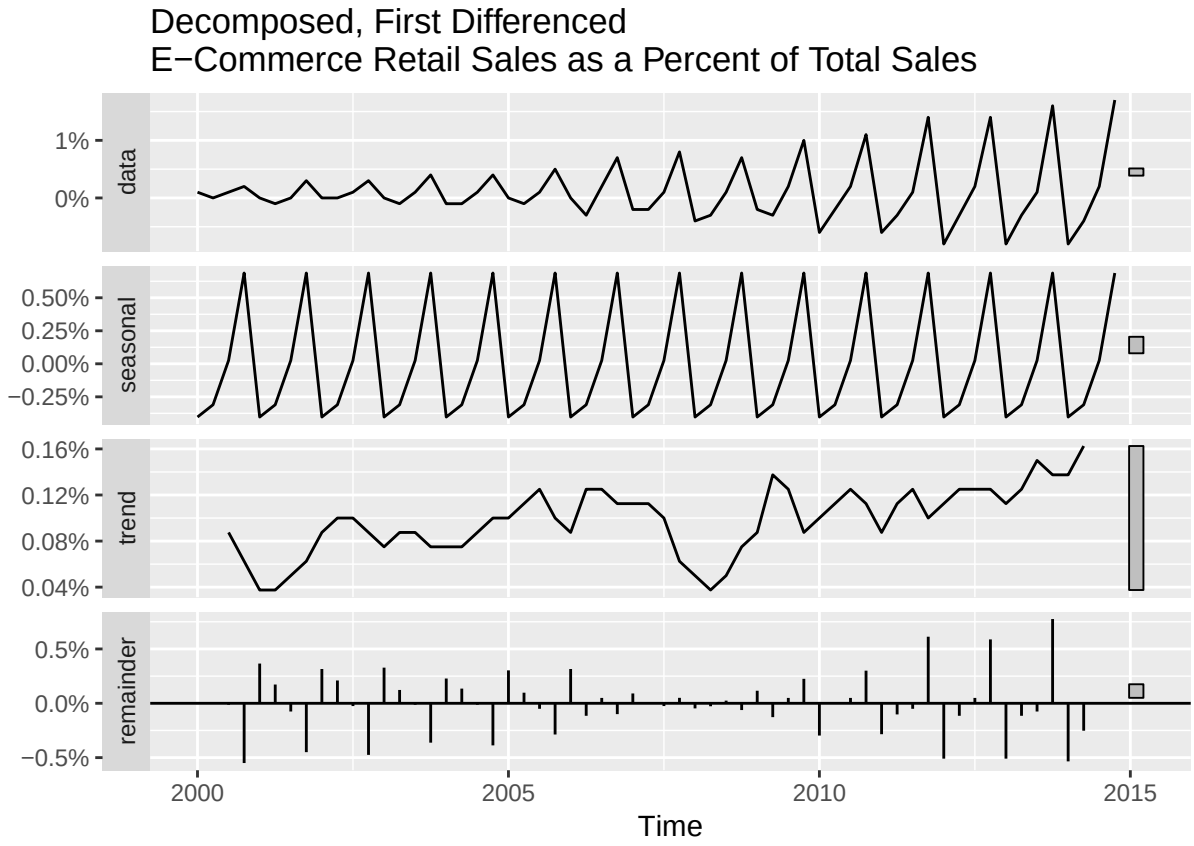
```
grid.arrange(p1, p2, ncol=2)
```



The ACF also points to a strong trend given it remains highly significant for many lags. The PACF also indicates seasonality.

First we attempted a log transform to stabalize the variance which we followed with differencing. However we found that differencing alone produced more stationary residuals which you will find below.`vndiffs` indicated a first difference would be a good start according to the KPSS test.

```
series_diff1 <- diff(series1_train)
autoplot(decompose(series_diff1)) +
  labs(title="Decomposed, First Differenced\nE-Commerce Retail Sales as a Percent of Total Sale
  scale_y_continuous(labels = scales::percent)
```

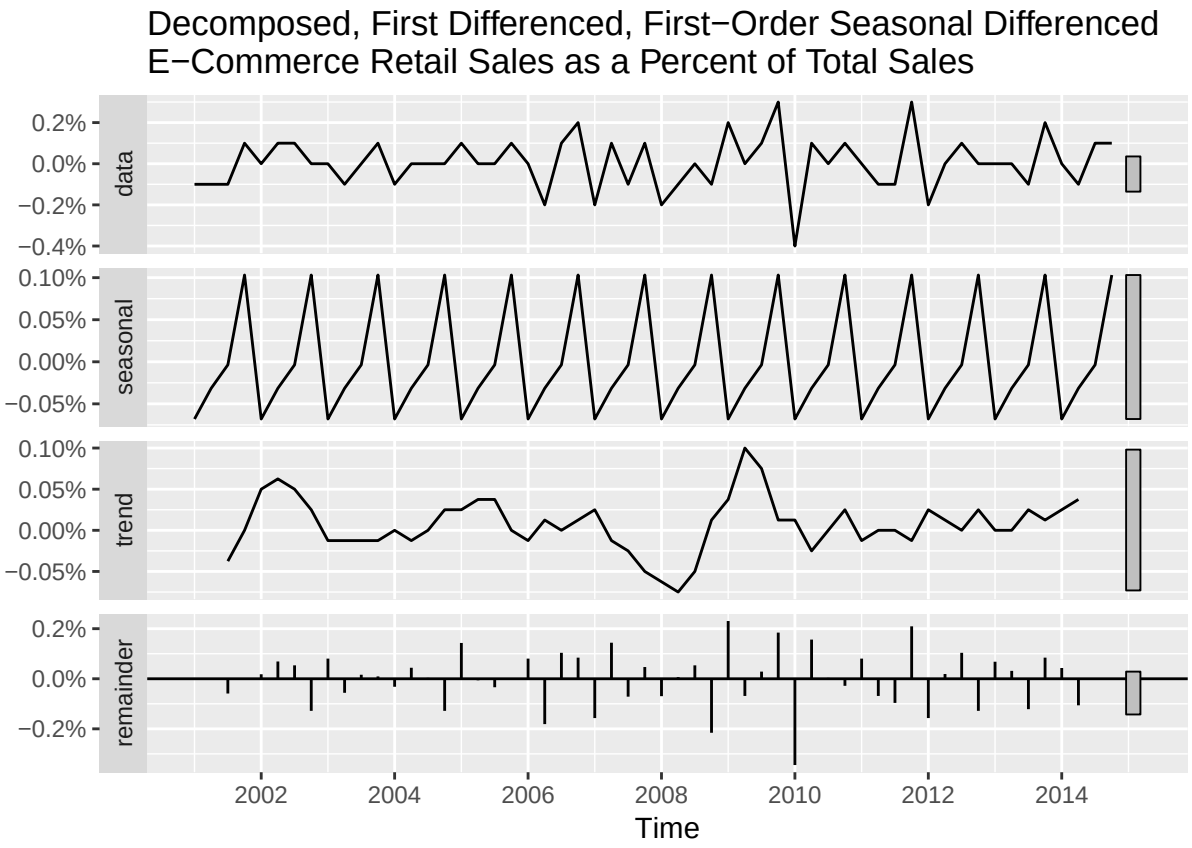



```
series_diff1 %>% nsdiffs()
```

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

A lot of the trend is removed by the first difference, but there is still a strong seasonal component we can likely remove with a first order seasonal differencing. `nsdiffs` also indicates a single seasonal difference could help achieve stationarity.

```
series_diff1_diff4 <- diff(series_diff1,4)
autoplot(decompose(series_diff1_diff4)) +
  labs(title="Decomposed, First Differenced, First-Order Seasonal Differenced\nE-Commerce Reta
  scale_y_continuous(labels = scales::percent)
```

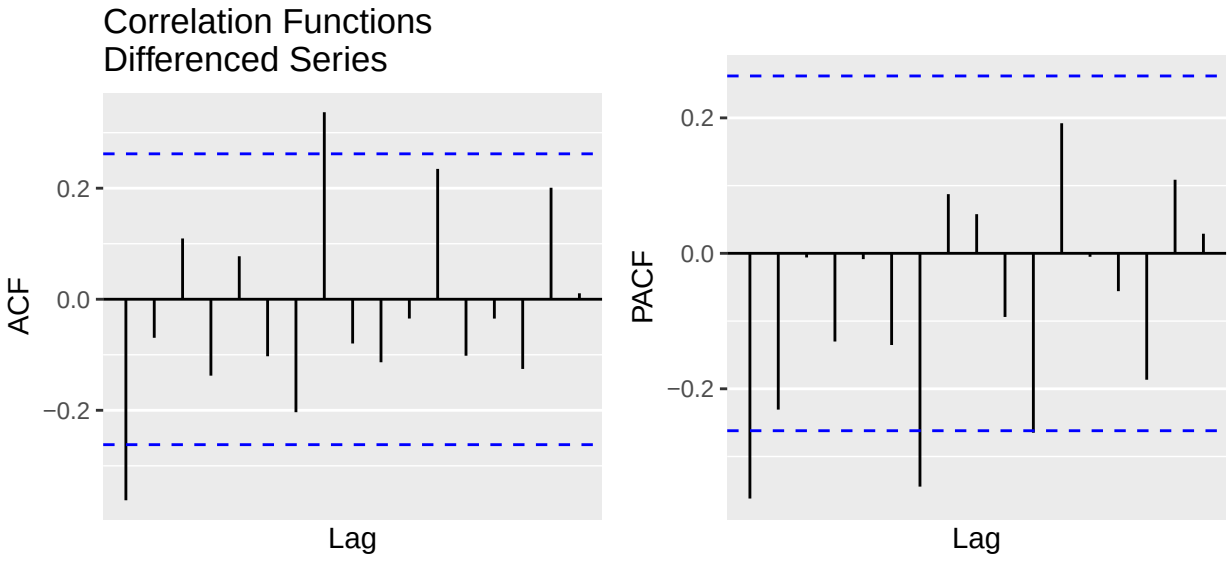



```
series_diff1_diff4 %>% ur.kpss()
```

```
##
## ########################################
## # KPSS Unit Root / Cointegration Test #
## ########################################
##
## The value of the test statistic is: 0.094
```

After these differences, the time series looks roughly stationary with no strong trend and roughly stable variance (so no need to transform the series). The KPSS test also indicates that there is not sufficient evidence to reject the null hypothesis that the data are stationary.

```
plot_cfs <- function(ts,title){
  p1 = autoplot(acf(ts, plot = FALSE)) +
  ggtitle(title)

p2 = autoplot(pacf(ts, plot = FALSE)) +
  ggtitle("")

grid.arrange(p1, p2, ncol=2)
}

plot_cfs(series_diff1_diff4,"Correlation Functions\nDifferenced Series")
```



The pacf and acf of the differenced series are both significant at the first lag and another lag roughly 2 years back. This is inidicative of a low order p and maybe q to take care of the first lag and low order P and Q (perhaps 2 because the lags are about 2 years back) to take care of the later lag in our SARIMA model.

Model Construction

Our exploratory data analysis points to a SARIMA model with $d = 1$ and $D = 1$. It also points to maximum values of the other parameters as $p = 1$, $q = 1$, $P = 2$, $Q = 2$. Let's fit that model, a few around it with fewer parameters, and one with `auto.arima` to compare.

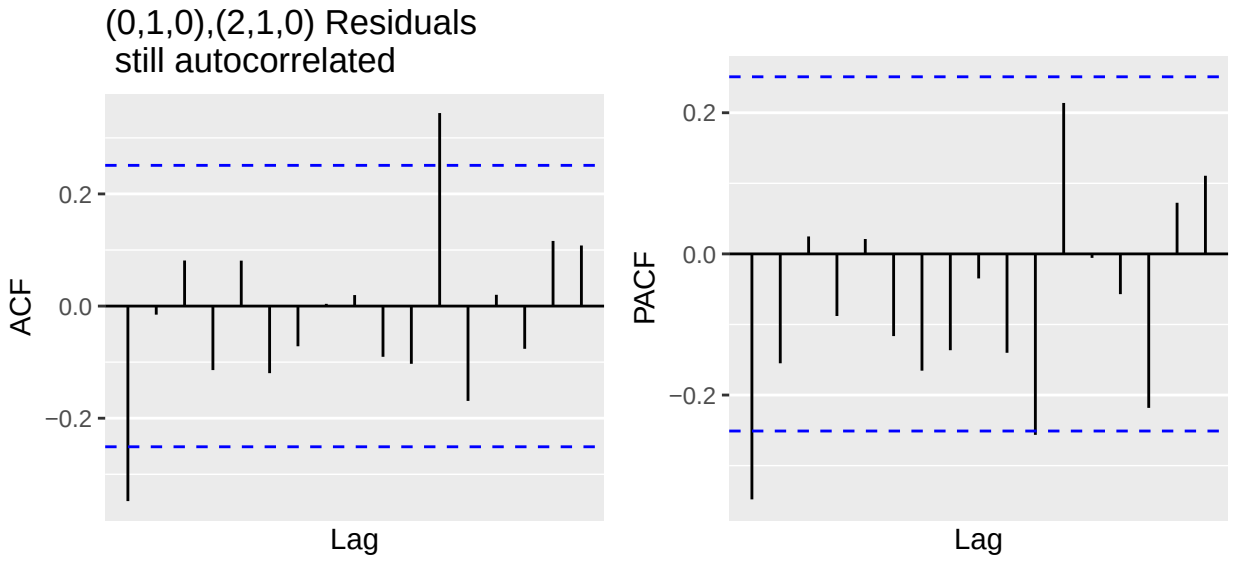
```
# train models
  # all models up to the max according to EDA
models <- list()
i <- 1
model_pdqPDQ <- c()
num_params <- c()
for (p in 0:1){
  for (q in 0:1){
    for (P in 0:2){
      for (Q in 0:2){
        models[[i]] <- series1_train %>% Arima(order=c(p,1,q),
                                               seasonal = c(P,1,Q),
                                               method='ML')
        model_pdqPDQ <- c(model_pdqPDQ,
                          glue("({p},1,{q})({P},1,{Q})"))
        num_params <- c(num_params,p+1+q+P+1+Q)
        i <- i+1
      }
    }
  }
}
```

```
  # add the auto arima model
auto.arima.model <- series1_train %>% auto.arima(stepwise=FALSE, approximation=FALSE)
models[[i]] <- auto.arima.model
model_pdqPDQ <- c(model_pdqPDQ, "auto.arima (0,1,1),(1,1,2)")
num_params <- c(num_params,5)


# model evaluation
# function to evaluate models
evaluate_model <- function(sarima_model, model_pdqPDQ, num_params){
  m_forecast <- sarima_model %>% forecast(8)
  rmse <- sqrt(mean((m_forecast$mean - series1_test)^2))
  return(c(model_pdqPDQ,num_params,rmse,
           sarima_model$aic,
           sarima_model$aicc,
           sarima_model$bic
           ))
}
model_info <- tibble(
  sarima_model=models,
  model_pdqPDQ=model_pdqPDQ,
  num_params=num_params
)

model_info <-
  model_info %>%
  pmap(evaluate_model) %>%
  bind_cols() %>%
  t() %>% as.tibble()
names(model_info) <- c("model","num_params","rmse for 2015-2016 prediction","aic","aicc","bic")

model_info %>% arrange(num_params,model)
```

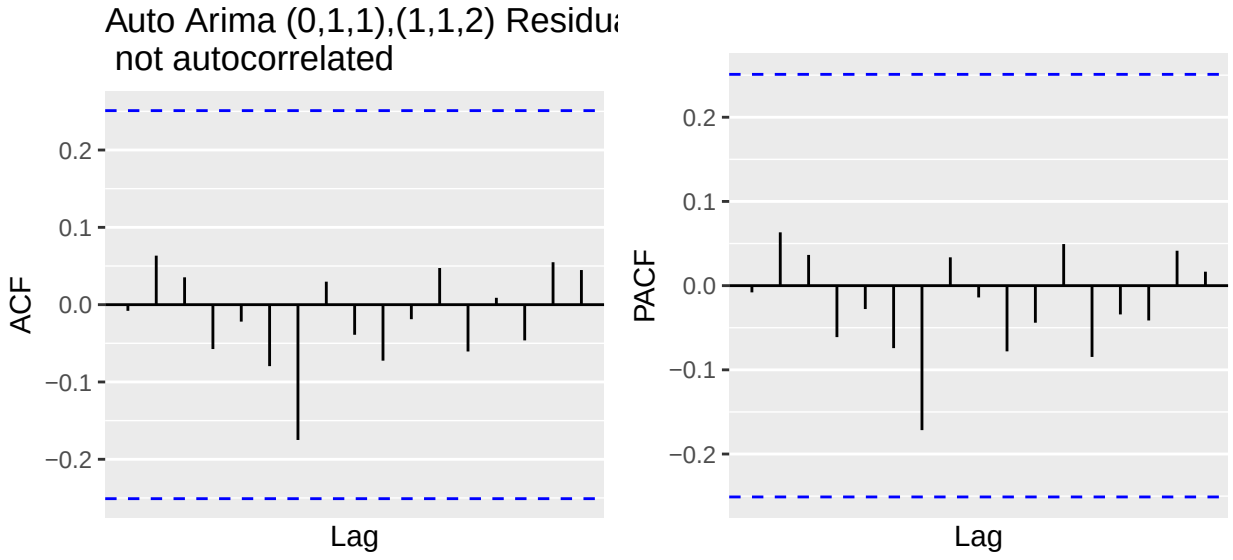
```
## # A tibble: 37 x 6
##    model          num_params `rmse for 2015-2016~ aic      aicc    bic
##    <chr>          <chr>      <chr>                <chr>    <chr>   <chr>
##  1 (0,1,0)(0,1,0) 2          0.00239791576165635  -586.80~ -586.7~ -584.7~
##  2 (0,1,0)(0,1,1) 3          0.00245966282249882  -585.40~ -585.1~ -581.3~
##  3 (0,1,0)(1,1,0) 3          0.00249761164798811  -585.83~ -585.6~ -581.7~
##  4 (0,1,1)(0,1,0) 3          0.00356276096229345  -596.57~ -596.3~ -592.5~
##  5 (1,1,0)(0,1,0) 3          0.00277991025461094  -592.60~ -592.3~ -588.5~
##  6 (0,1,0)(0,1,2) 4          0.00252918271466643  -592.00~ -591.5~ -585.9~
##  7 (0,1,0)(1,1,1) 4          0.00232410813518602  -585.29~ -584.8~ -579.2~
##  8 (0,1,0)(2,1,0) 4          0.00222101445388669  -590.26~ -589.8~ -584.1~
##  9 (0,1,1)(0,1,1) 4          0.00372441347880992  -595.34~ -594.8~ -589.2~
## 10 (0,1,1)(1,1,0) 4          0.00377881978794254  -595.76~ -595.2~ -589.6~
## # ... with 27 more rows
```

Auto arima finds the model with the best aic. The models with fewer parameters seem to do better on the out of sample forecast though. Let's take a look at the residuals.

```
plot_cfs(series1_train %>%
           Arima(order=c(0,1,0), seasonal = c(2,1,0), method='ML')
         %>% residuals(),
         "(0,1,0),(2,1,0) Residuals\n still autocorrelated")
```

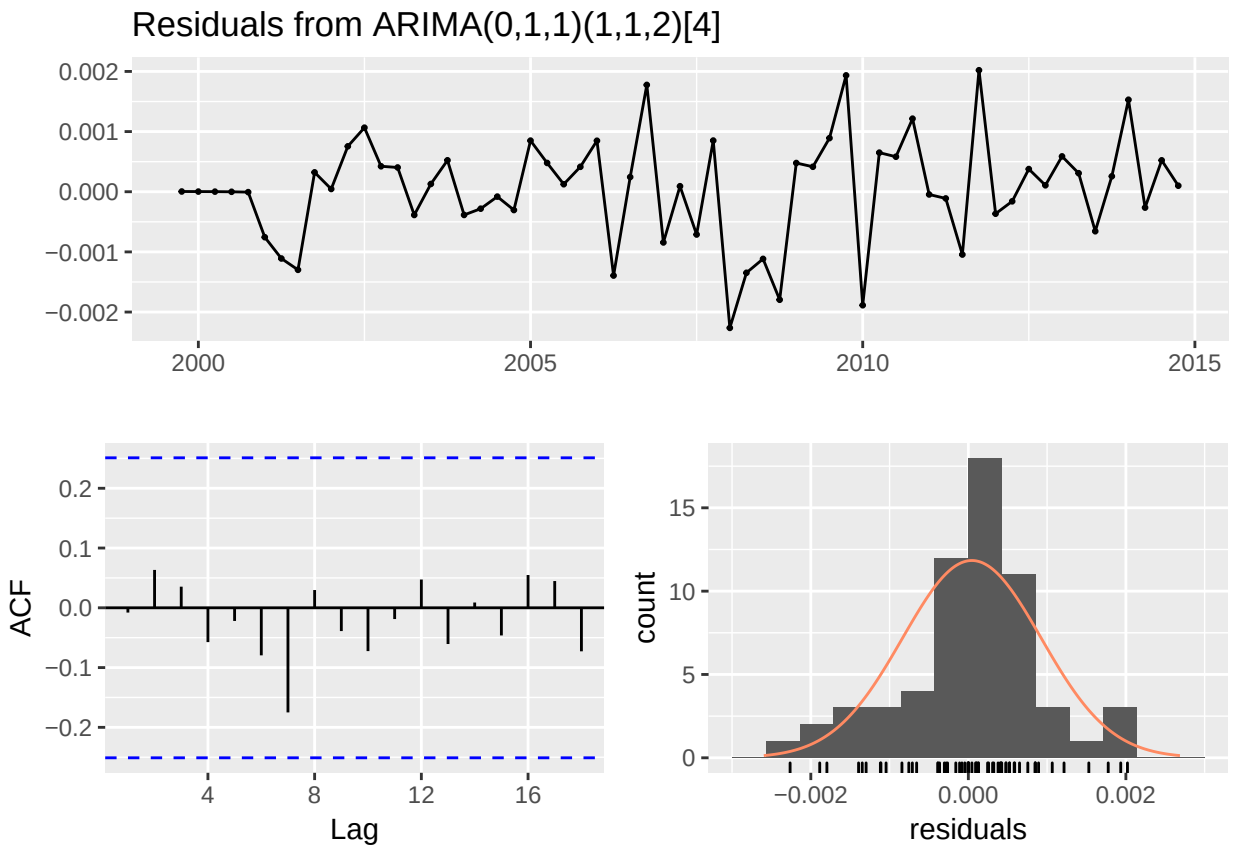



```
plot_cfs(auto.arima.model %>% residuals(),
         "Auto Arima (0,1,1),(1,1,2) Residuals\n not autocorrelated")
```



Given the remaining autocorrelation on simpler models (only one shown above, both others checked appeared similar), we believe the auto.arima SARIMA(0,1,1),(1,1,1) model is best.

```
checkresiduals(auto.arima.model)
```




```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(0,1,1)(1,1,2)[4]
## Q* = 3.2899, df = 4, p-value = 0.5105
##
## Model df: 4.   Total lags used: 8
```

The auto arima residuals look like white noise. The residual time series plot looks approximately stationary and centered around 0. This is confirmed by the histogram, which is centered around 0 and has an approximately normal distribution. Finally, the ACF plot shows no significant correlations, indiciative of white noise. Based on LJung test, we cannot reject H_0 that there is no correlation bewteen the lag values and the current values. We'll also run a Shapiro-Wilk test to test the normality of the residuals.

```
shapiro.test(auto.arima.model$residuals)
```

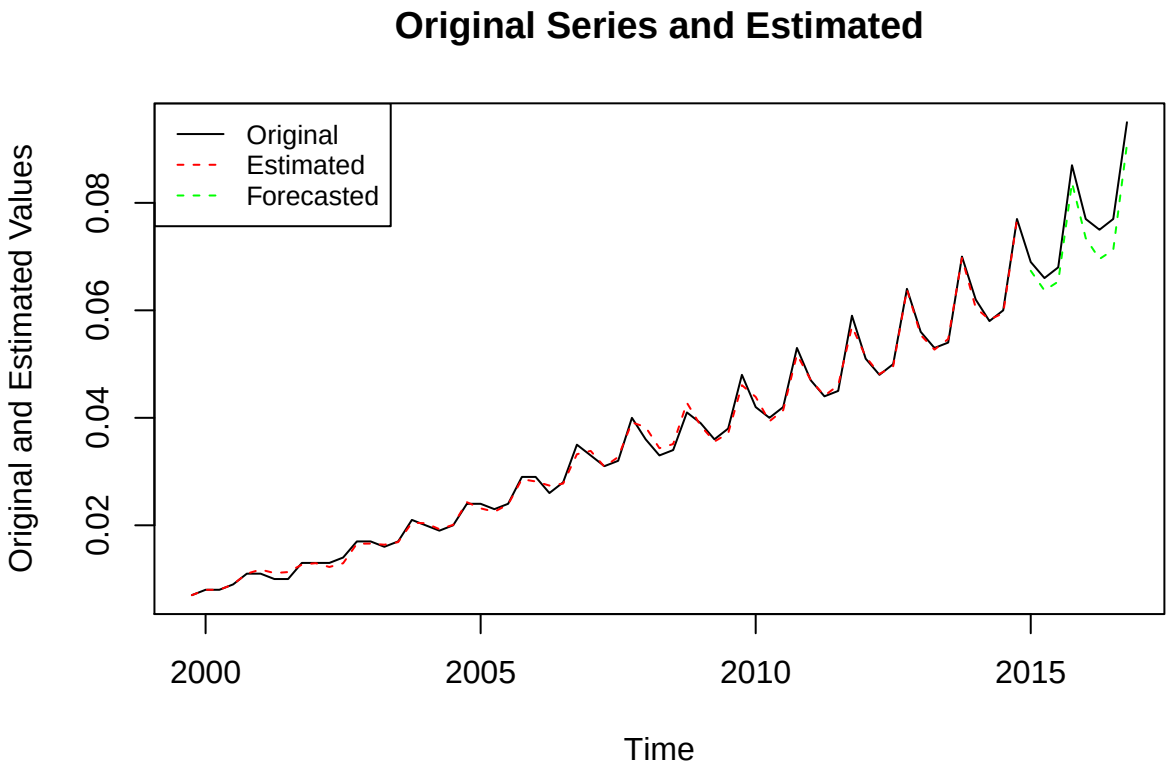
```
##
##  Shapiro-Wilk normality test
##
## data:  auto.arima.model$residuals
## W = 0.97398, p-value = 0.2186
```

Based on this, we cannot reject H_0 that the distribution of the residuals is not statistically different from a normal distribution.

Forecast

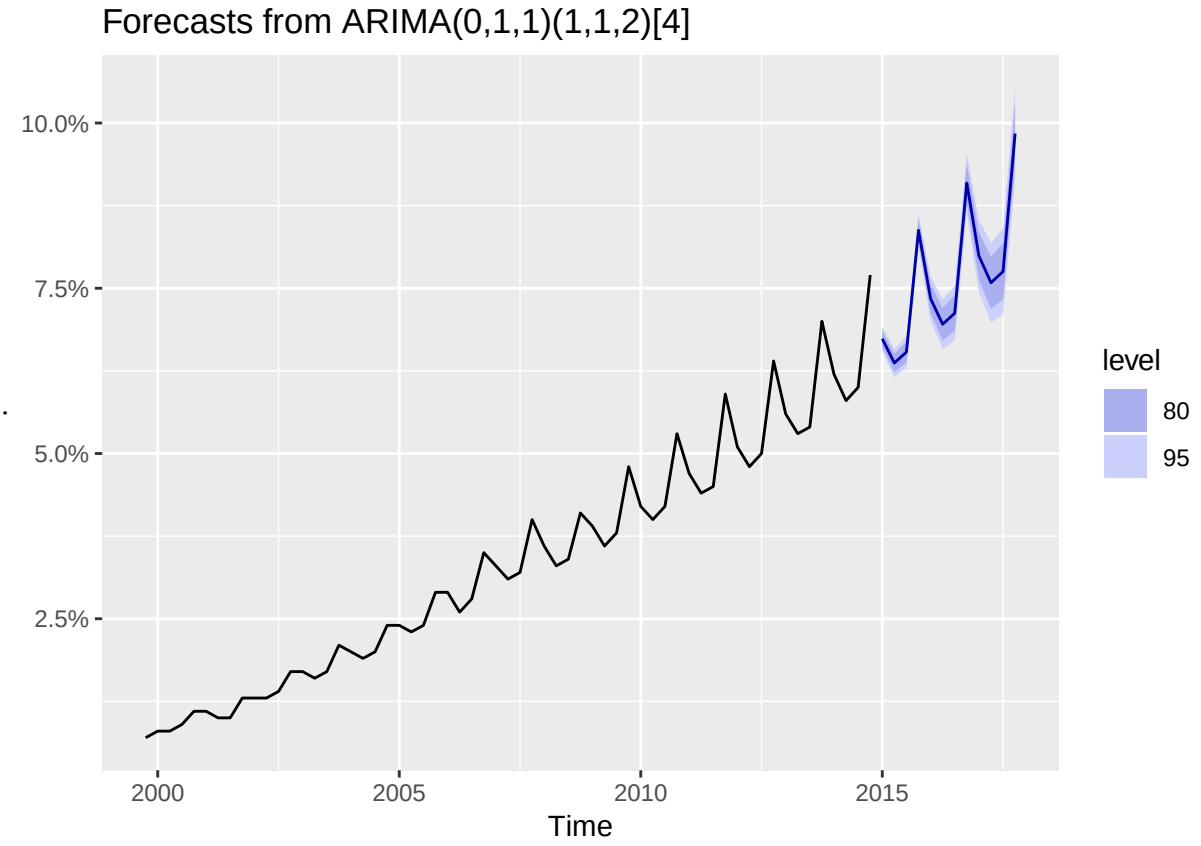
Let's first check the in-sample performance of our model:

```
# create time series of predicted in-samples values through 2016 Q4
pred <- predict(auto.arima.model, n.ahead = 8)
pred <-  ts(pred$pred, start = c(2015, 1), frequency = 4)

plot(series1, ylab = 'Original and Estimated Values', main = 'Original Series and Estimated')
lines(fitted.values(auto.arima.model), col = 'red', lty = 2)
lines(pred, col = 'green', lty = 2 )
legend('topleft', legend = c('Original', 'Estimated', 'Forecasted'), col = c('black', 'red', '
```



Based on the chart above, we see that our model's estimated values fit the training series (up to Q1, 2015) pretty well, with low deviation from the original values. The forecasted values for the test set (2015 - 2016) also fit fairly well, following the overall trend and seasonality, though the forecast appears to slightly underestimate the true series especially in the seasonal low points.Let's now forecast through 2017 using our model.

```
auto.arima.model %>% forecast(h=12) %>% autoplot() +
  scale_y_continuous(labels = scales::percent)
```



The predictions seem to follow the trend very well, with the forecasted values as well as the confidence bands following the trend and seasonality very closely.

Question 2: Learning how to use the xts library

Materials covered in Question 2 of this lab

- Primarily the references listed in this document:
  - "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

Task 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] -0.207 -0.849 -0.979 -0.176 0.419 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : NULL
##   ..$ : chr [1:2] "Series01" "Series02"
```

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

```
##         Series01    Series02
##  [1,] -0.2067903 -2.57432001
##  [2,] -0.8494274  0.03869383
##  [3,] -0.9791297  0.39014700
##  [4,] -0.1755092  0.69912158
```

```
##  [5,]  0.4186232  0.25366738
##  [6,] -0.5242737 -0.96714468
##  [7,]  0.8142566 -0.31002885
##  [8,] -2.1859139  1.05499234
##  [9,] -1.0006592  0.33356424
## [10,]  0.9378056 -0.02988635
```

```
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
```

```
##
## Attaching package: 'xts'
```

```
## The following objects are masked from 'package:dplyr':
##
##     first, last
```

```
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] -0.207 -0.849 -0.979 -0.176 0.419 ...
##  - 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 -0.2067903 -2.57432001
## 2015-01-02 -0.8494274  0.03869383
## 2015-01-03 -0.9791297  0.39014700
## 2015-01-04 -0.1755092  0.69912158
## 2015-01-05  0.4186232  0.25366738
## 2015-01-06 -0.5242737 -0.96714468
## 2015-01-07  0.8142566 -0.31002885
## 2015-01-08 -2.1859139  1.05499234
## 2015-01-09 -1.0006592  0.33356424
## 2015-01-10  0.9378056 -0.02988635
```

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,] -0.2067903 -2.57432001
## [2,] -0.8494274  0.03869383
## [3,] -0.9791297  0.39014700
## [4,] -0.1755092  0.69912158
## [5,]  0.4186232  0.25366738
```

`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.

```
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
```

```
# table(df$Series.id, useNA = 'always') table(df$Period,
# useNA = 'always')

# 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
```

```
## Jan 2007    2007.000   5.0
## Feb 2007    2007.083   4.9
## Mar 2007    2007.167   4.5
## Apr 2007    2007.250   4.3
## May 2007    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.
```

```
##
## Attaching package: 'quantmod'
```

```
## The following object is masked from 'package:Hmisc':
##
##     Lag
```

```
getSymbols("TWTR")
```

```
## '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.

##
## WARNING: There have been significant changes to Yahoo Finance data.
## Please see the Warning section of '?getSymbols.yahoo' for details.
##
## This message is shown once per session and may be disabled by setting
## options("getSymbols.yahoo.warning"=FALSE).

## [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   117701600
## 2013-11-08     45.93     46.94    40.69      41.65    27925300
## 2013-11-11     40.50     43.00    39.40      42.90    16113900
## 2013-11-12     43.66     43.78    41.83      41.90     6316700
## 2013-11-13     41.03     42.87    40.76      42.60     8688300
## 2013-11-14     42.34     45.67    42.24      44.69    11099400
##            TWTR.Adjusted
## 2013-11-07         44.90
## 2013-11-08         41.65
## 2013-11-11         42.90
## 2013-11-12         41.90
## 2013-11-13         42.60
## 2013-11-14         44.69
```

```
str(TWTR)
```

```
## An 'xts' object on 2013-11-07/2018-11-23 containing:
##   Data: num [1:1271, 1:6] 45.1 45.9 40.5 43.7 41 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : NULL
##   ..$ : chr [1:6] "TWTR.Open" "TWTR.High" "TWTR.Low" "TWTR.Close" ...
##   Indexed by objects of class: [Date] TZ: UTC
##   xts Attributes:
## List of 2
##  $ src    : chr "yahoo"
##  $ updated: POSIXct[1:1], format: "2018-11-25 20:10:22"
```

Note that the date obtained from the getSymbols function of the 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.

```r
# 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, 1:7] 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 [1:7] "unemp_xts" "TWTR.Open" "TWTR.High" "TWTR.Low" ...
##   Indexed by objects of class: [Date] TZ: UTC
##   xts Attributes:
##  NULL
```

```r
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     6916100
## 2014-05-01       6.1     39.01     40.77    38.97      39.09    15759800
## 2014-07-01       6.5     42.06     42.95    41.91      42.05    36019300
## 2014-08-01       6.3     45.01     45.54    43.81      44.13    37194800
## 2014-10-01       5.5     51.08     51.29    49.15      50.06    24733500
## 2014-12-01       5.4     41.29     41.29    39.00      39.04    22214000
##            TWTR.Adjusted
## 2014-04-01         46.98
## 2014-05-01         39.09
## 2014-07-01         42.05
## 2014-08-01         44.13
## 2014-10-01         50.06
## 2014-12-01         39.04
```

```r
# 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 2007-01-01/2018-11-23 containing:
##   Data: num [1:1370, 1:7] 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 [1:7] "unemp_xts" "TWTR.Open" "TWTR.High" "TWTR.Low" ...
##   Indexed by objects of class: [Date] TZ: UTC
##   xts Attributes:
##  NULL
```

```r
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
##            TWTR.Adjusted
## 2007-01-01         99999
## 2007-02-01         99999
## 2007-03-01         99999
## 2007-04-01         99999
## 2007-05-01         99999
## 2007-06-01         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, 1:7] 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 [1:7] "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
##            TWTR.Adjusted
## 2007-01-01         99999
## 2007-02-01         99999
## 2007-03-01         99999
## 2007-04-01         99999
## 2007-05-01         99999
## 2007-06-01         99999
```

```
# View(TWTR_unemp03)

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

```
## An 'xts' object on 2013-11-07/2018-11-23 containing:
```

```
##   Data: num [1:1271, 1:7] 99999 99999 99999 99999 99999 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : NULL
##   ..$ : chr [1:7] "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   117701600
## 2013-11-08     99999     45.93     46.94    40.69      41.65    27925300
## 2013-11-11     99999     40.50     43.00    39.40      42.90    16113900
## 2013-11-12     99999     43.66     43.78    41.83      41.90     6316700
## 2013-11-13     99999     41.03     42.87    40.76      42.60     8688300
## 2013-11-14     99999     42.34     45.67    42.24      44.69    11099400
##            TWTR.Adjusted
## 2013-11-07         44.90
## 2013-11-08         41.65
## 2013-11-11         42.90
## 2013-11-12         41.90
## 2013-11-13         42.60
## 2013-11-14         44.69
```

```
# 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 2007-01-01/2018-11-23 containing:
##   Data: num [1:1370, 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"])
```

```
##            unemp_xts unemp_xts.1
## 2013-10-01       7.0         7.0
## 2013-11-01       6.6         6.6
## 2013-11-07   99999.0     99999.0
## 2013-11-08   99999.0     99999.0
## 2013-11-11   99999.0     99999.0
## 2013-11-12   99999.0     99999.0
## 2013-11-13   99999.0     99999.0
## 2013-11-14   99999.0     99999.0
## 2013-11-15   99999.0     99999.0
## 2013-11-18   99999.0     99999.0
## 2013-11-19   99999.0     99999.0
## 2013-11-20   99999.0     99999.0
## 2013-11-21   99999.0     99999.0
## 2013-11-22   99999.0     99999.0
## 2013-11-25   99999.0     99999.0
## 2013-11-26   99999.0     99999.0
```

```
## 2013-11-27   99999.0     99999.0
## 2013-11-29   99999.0     99999.0
## 2013-12-01       6.5         6.5
## 2013-12-02   99999.0     99999.0
## 2013-12-03   99999.0     99999.0
## 2013-12-04   99999.0     99999.0
## 2013-12-05   99999.0     99999.0
## 2013-12-06   99999.0     99999.0
## 2013-12-09   99999.0     99999.0
## 2013-12-10   99999.0     99999.0
## 2013-12-11   99999.0     99999.0
## 2013-12-12   99999.0     99999.0
## 2013-12-13   99999.0     99999.0
## 2013-12-16   99999.0          NA
## 2013-12-17   99999.0          NA
## 2013-12-18   99999.0          NA
## 2013-12-19   99999.0          NA
## 2013-12-20   99999.0          NA
## 2013-12-23   99999.0          NA
## 2013-12-24   99999.0          NA
## 2013-12-26   99999.0          NA
## 2013-12-27   99999.0          NA
## 2013-12-30   99999.0          NA
```

```
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"]
```

```
##            unemp_xts
## 2013-10-01       7.0
## 2013-11-01       6.6
## 2013-11-07        NA
## 2013-11-08        NA
## 2013-11-11        NA
## 2013-11-12        NA
## 2013-11-13        NA
## 2013-11-14        NA
## 2013-11-15        NA
## 2013-11-18        NA
## 2013-11-19        NA
## 2013-11-20        NA
## 2013-11-21        NA
## 2013-11-22        NA
## 2013-11-25        NA
## 2013-11-26        NA
## 2013-11-27        NA
## 2013-11-29        NA
## 2013-12-01       6.5
## 2013-12-02        NA
## 2013-12-03        NA
## 2013-12-04        NA
## 2013-12-05        NA
## 2013-12-06        NA
## 2013-12-09        NA
## 2013-12-10        NA
## 2013-12-11        NA
## 2013-12-12        NA
## 2013-12-13        NA
## 2013-12-16        NA
```

```
## 2013-12-17        NA
## 2013-12-18        NA
## 2013-12-19        NA
## 2013-12-20        NA
## 2013-12-23        NA
## 2013-12-24        NA
## 2013-12-26        NA
## 2013-12-27        NA
## 2013-12-30        NA
## 2013-12-31        NA
## 2014-01-01       7.0
## 2014-01-02        NA
## 2014-01-03        NA
## 2014-01-06        NA
## 2014-01-07        NA
## 2014-01-08        NA
## 2014-01-09        NA
## 2014-01-10        NA
## 2014-01-13        NA
## 2014-01-14        NA
## 2014-01-15        NA
## 2014-01-16        NA
## 2014-01-17        NA
## 2014-01-21        NA
## 2014-01-22        NA
## 2014-01-23        NA
## 2014-01-24        NA
## 2014-01-27        NA
## 2014-01-28        NA
## 2014-01-29        NA
## 2014-01-30        NA
## 2014-01-31        NA
## 2014-02-01       7.0
```

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

```
##            unemp_xts
## 2013-10-01  7.000000
## 2013-11-01  6.600000
## 2013-11-07  6.580000
## 2013-11-08  6.576667
## 2013-11-11  6.566667
## 2013-11-12  6.563333
## 2013-11-13  6.560000
## 2013-11-14  6.556667
## 2013-11-15  6.553333
## 2013-11-18  6.543333
```

```
## 2013-11-19  6.540000
## 2013-11-20  6.536667
## 2013-11-21  6.533333
## 2013-11-22  6.530000
## 2013-11-25  6.520000
## 2013-11-26  6.516667
## 2013-11-27  6.513333
## 2013-11-29  6.506667
## 2013-12-01  6.500000
## 2013-12-02  6.516129
## 2013-12-03  6.532258
## 2013-12-04  6.548387
## 2013-12-05  6.564516
## 2013-12-06  6.580645
## 2013-12-09  6.629032
## 2013-12-10  6.645161
## 2013-12-11  6.661290
## 2013-12-12  6.677419
## 2013-12-13  6.693548
## 2013-12-16  6.741935
## 2013-12-17  6.758065
## 2013-12-18  6.774194
## 2013-12-19  6.790323
## 2013-12-20  6.806452
## 2013-12-23  6.854839
## 2013-12-24  6.870968
## 2013-12-26  6.903226
## 2013-12-27  6.919355
## 2013-12-30  6.967742
## 2013-12-31  6.983871
## 2014-01-01  7.000000
## 2014-01-02  7.000000
## 2014-01-03  7.000000
## 2014-01-06  7.000000
## 2014-01-07  7.000000
## 2014-01-08  7.000000
## 2014-01-09  7.000000
## 2014-01-10  7.000000
## 2014-01-13  7.000000
## 2014-01-14  7.000000
## 2014-01-15  7.000000
## 2014-01-16  7.000000
## 2014-01-17  7.000000
## 2014-01-21  7.000000
## 2014-01-22  7.000000
## 2014-01-23  7.000000
## 2014-01-24  7.000000
## 2014-01-27  7.000000
```

```
## 2014-01-28  7.000000
## 2014-01-29  7.000000
## 2014-01-30  7.000000
## 2014-01-31  7.000000
## 2014-02-01  7.000000
```

```
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) unemp_xts

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

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

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

cbind(unemp_xts, diff(unemp_xts, lag = 1, difference = 1, log = FALSE,
    na.pad = TRUE), diff(unemp_xts, lag = 12, differences = 1)) %>%
    head(20)
```

```
##            ..1  ..2 ..3
## 2007-01-01 5.0   NA  NA
## 2007-02-01 4.9 -0.1  NA
## 2007-03-01 4.5 -0.4  NA
## 2007-04-01 4.3 -0.2  NA
## 2007-05-01 4.3  0.0  NA
## 2007-06-01 4.7  0.4  NA
## 2007-07-01 4.9  0.2  NA
## 2007-08-01 4.6 -0.3  NA
## 2007-09-01 4.5 -0.1  NA
## 2007-10-01 4.4 -0.1  NA
## 2007-11-01 4.5  0.1  NA
## 2007-12-01 4.8  0.3  NA
## 2008-01-01 5.4  0.6 0.4
## 2008-02-01 5.2 -0.2 0.3
## 2008-03-01 5.2  0.0 0.7
## 2008-04-01 4.8 -0.4 0.5
## 2008-05-01 5.2  0.4 0.9
## 2008-06-01 5.7  0.5 1.0
## 2008-07-01 6.0  0.3 1.1
## 2008-08-01 6.1  0.1 1.5
```

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 end index on each
# week of the series

# 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   117701600
## 2013-11-08     45.93     46.94    40.69      41.65    27925300
## 2013-11-11     40.50     43.00    39.40      42.90    16113900
## 2013-11-12     43.66     43.78    41.83      41.90     6316700
## 2013-11-13     41.03     42.87    40.76      42.60     8688300
## 2013-11-14     42.34     45.67    42.24      44.69    11099400
##            TWTR.Adjusted
## 2013-11-07         44.90
## 2013-11-08         41.65
## 2013-11-11         42.90
## 2013-11-12         41.90
## 2013-11-13         42.60
## 2013-11-14         44.69
```

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

```
##   [1]   0   2   7  12  16  21  26  31  35  39  44  49  53  58
##  [15]  63  68  72  77  82  87  92  97 102 107 111 116 121 126
##  [29] 131 136 140 145 150 155 160 164 169 174 179 184 189 194
##  [43] 199 204 208 213 218 223 228 233 238 243 248 253 258 263
##  [57] 267 272 277 282 286 290 295 300 304 309 314 319 323 328
##  [71] 333 338 343 348 352 357 362 367 372 377 382 387 391 396
##  [85] 401 406 411 415 420 425 430 435 440 445 450 455 460 464
##  [99] 469 474 479 484 489 494 499 504 509 514 518 523 528 533
## [113] 537 541 546 551 555 560 565 570 574 579 584 589 594 598
## [127] 603 608 613 618 623 628 633 638 643 647 652 657 662 667
## [141] 671 676 681 686 691 696 701 706 711 715 720 725 730 735
## [155] 740 745 750 755 760 765 769 774 779 784 789 793 797 802
## [169] 806 811 816 821 826 830 835 840 845 850 855 860 864 869
```

```
## [183]  874  879  884  889  894  898  903  908  913  918  922  927  932  937
## [197]  942  947  952  957  962  966  971  976  981  986  991  996 1001 1006
## [211] 1011 1016 1020 1025 1030 1035 1040 1044 1048 1053 1057 1062 1067 1072
## [225] 1077 1081 1086 1091 1096 1101 1105 1110 1115 1120 1125 1130 1135 1140
## [239] 1145 1149 1154 1159 1164 1169 1173 1178 1183 1188 1193 1198 1203 1208
## [253] 1213 1217 1222 1227 1232 1237 1242 1247 1252 1257 1262 1267 1271
```

```
# 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      43.28
## 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, 0.5, 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 43.28 2.30 41.81 42.46 43.28 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 65.17 3.84 60.98 62.87 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

```
AMAZ_df <- read_csv(file = "AMAZ.csv")
```

```
## Parsed with column specification:
## cols(
##   Index = col_date(format = ""),
##   AMAZ.Open = col_double(),
##   AMAZ.High = col_double(),
##   AMAZ.Low = col_double(),
##   AMAZ.Close = col_double(),
##   AMAZ.Volume = col_integer()
## )
```

```
UMCSENT_df <- read_csv(file = "UMCSENT.csv")
```

```
## Parsed with column specification:
## cols(
##   Index = col_date(format = ""),
```

```
##   UMCSENT = col_double()
## )
```

```
head(AMAZ_df)
```

```
## # A tibble: 6 x 6
##   Index      AMAZ.Open AMAZ.High AMAZ.Low AMAZ.Close AMAZ.Volume
##   <date>         <dbl>     <dbl>    <dbl>      <dbl>       <int>
## 1 2007-01-03      20        20       16         16           650
## 2 2007-01-04      20        20       20         20            67
## 3 2007-01-08      19.2      22       19.2       22          1801
## 4 2007-01-09      22        22       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
```

```
head(UMCSENT_df)
```

```
## # A tibble: 6 x 2
##   Index      UMCSENT
##   <date>       <dbl>
## 1 1978-01-01    83.7
## 2 1978-02-01    84.3
## 3 1978-03-01    78.8
## 4 1978-04-01    81.6
## 5 1978-05-01    82.9
## 6 1978-06-01    80
```

2. Convert them to xts objects

```
AMAZ <- xts(x = AMAZ_df %>% select(-Index), order.by = AMAZ_df$Index)
UMCSENT <- xts(x = UMCSENT_df$UMCSENT, order.by = UMCSENT_df$Index)
```

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

```
stocks <- merge.xts(AMAZ, UMCSENT, join = "outer", fill = -9999)
```

b. then create a new series, named UMCSENT02, from the original UMCSENT series replace all of the -9999 with NAs

```
UMCSENT02 <- ifelse(stocks[, "UMCSENT"] == -9999, NA, stocks[,
    "UMCSENT"])
```

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

My interpretation of "last" is the observation the latest in time that occurs before the target row that does not have an NA value.

```
UMCSENT03 <- na.locf(UMCSENT02)
```

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

```
UMCSENT04 <- na.approx(UMCSENT02)
```

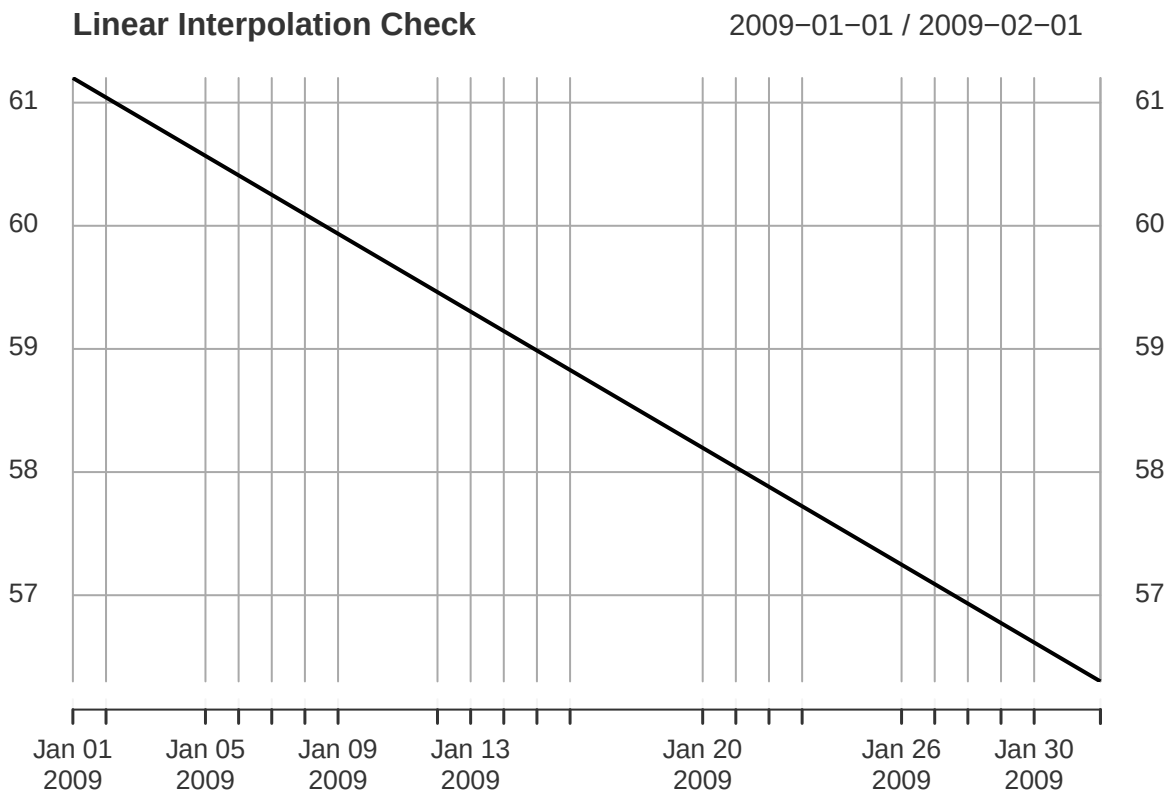
e. Print out some observations to ensure that your merge as well as the missing value imputation are done correctly. I leave it up to you to decide exactly how many observations to print; do something that makes sense. (Hint: Do not print out the entire dataset!)

```
stocks <- merge.xts(AMAZ, UMCSENT, join = "outer", fill = -9999)
stocks <- merge.xts(stocks, UMCSENT02, UMCSENT03, UMCSENT04,
    fill = -9999)
stocks["2009-01-01/2009-02-01"]
```

```
##            AMAZ.Open AMAZ.High AMAZ.Low AMAZ.Close AMAZ.Volume UMCSENT
## 2009-01-01  -9999.00  -9999.00 -9999.00   -9999.00       -9999    61.2
## 2009-01-02        NA        NA       NA       0.40           0 -9999.0
## 2009-01-05      0.48      0.48     0.48       0.48         375 -9999.0
## 2009-01-06        NA        NA       NA       0.48           0 -9999.0
## 2009-01-07      0.60      0.60     0.60       0.60         875 -9999.0
## 2009-01-08      0.60      0.60     0.40       0.40        2510 -9999.0
## 2009-01-09      0.36      0.36     0.36       0.36         250 -9999.0
## 2009-01-12      0.36      0.40     0.28       0.28        9090 -9999.0
## 2009-01-13        NA        NA       NA       0.28           0 -9999.0
## 2009-01-14      0.40      0.40     0.40       0.40          25 -9999.0
## 2009-01-15        NA        NA       NA       0.40           0 -9999.0
## 2009-01-16        NA        NA       NA       0.40           0 -9999.0
## 2009-01-20      0.40      0.40     0.34       0.34        1000 -9999.0
## 2009-01-21      0.34      0.34     0.34       0.34        1250 -9999.0
## 2009-01-22      0.40      0.40     0.40       0.40         250 -9999.0
## 2009-01-23      0.40      0.40     0.33       0.33         182 -9999.0
## 2009-01-26        NA        NA       NA       0.33           0 -9999.0
## 2009-01-27        NA        NA       NA       0.33           0 -9999.0
## 2009-01-28        NA        NA       NA       0.33           0 -9999.0
## 2009-01-29        NA        NA       NA       0.33           0 -9999.0
## 2009-01-30      0.40      0.40     0.32       0.32        8250 -9999.0
## 2009-02-01  -9999.00  -9999.00 -9999.00   -9999.00       -9999    56.3
```

```
##            UMCSENT.1 UMCSENT.2 UMCSENT.3
## 2009-01-01      61.2      61.2  61.20000
## 2009-01-02        NA      61.2  61.04194
## 2009-01-05        NA      61.2  60.56774
## 2009-01-06        NA      61.2  60.40968
## 2009-01-07        NA      61.2  60.25161
## 2009-01-08        NA      61.2  60.09355
## 2009-01-09        NA      61.2  59.93548
## 2009-01-12        NA      61.2  59.46129
## 2009-01-13        NA      61.2  59.30323
## 2009-01-14        NA      61.2  59.14516
## 2009-01-15        NA      61.2  58.98710
## 2009-01-16        NA      61.2  58.82903
## 2009-01-20        NA      61.2  58.19677
## 2009-01-21        NA      61.2  58.03871
## 2009-01-22        NA      61.2  57.88065
## 2009-01-23        NA      61.2  57.72258
## 2009-01-26        NA      61.2  57.24839
## 2009-01-27        NA      61.2  57.09032
## 2009-01-28        NA      61.2  56.93226
## 2009-01-29        NA      61.2  56.77419
## 2009-01-30        NA      61.2  56.61613
## 2009-02-01      56.3      56.3  56.30000
```

```
plot(stocks["2009-01-01/2009-02-01"][, "UMCSENT.3"], main = "Linear Interpolation Check")
```



Printing the observations from january 2009 to the first day of Feb we see the values were fed forward.

The plot shows the linear interpolation also occured correctly.

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.

In order to get the daily return we need data at the daily level:

```
# make another series with daily frequency that has a
# meaningless value purpose is to get stocks index to be
# daily
date_range <- seq(as.Date(min(index(AMAZ))), as.Date(max(index(AMAZ))),
    by = "day")
daily <- xts(rep(1, length(date_range)), order.by = date_range)
stocks <- merge.xts(stocks, daily, fill = -9999)
stocks %>% head()
```

```
##            AMAZ.Open AMAZ.High AMAZ.Low AMAZ.Close AMAZ.Volume UMCSENT
## 1978-01-01     -9999     -9999    -9999      -9999       -9999    83.7
## 1978-02-01     -9999     -9999    -9999      -9999       -9999    84.3
## 1978-03-01     -9999     -9999    -9999      -9999       -9999    78.8
## 1978-04-01     -9999     -9999    -9999      -9999       -9999    81.6
## 1978-05-01     -9999     -9999    -9999      -9999       -9999    82.9
## 1978-06-01     -9999     -9999    -9999      -9999       -9999    80.0
##            UMCSENT.1 UMCSENT.2 UMCSENT.3 daily
## 1978-01-01      83.7      83.7      83.7 -9999
## 1978-02-01      84.3      84.3      84.3 -9999
## 1978-03-01      78.8      78.8      78.8 -9999
## 1978-04-01      81.6      81.6      81.6 -9999
## 1978-05-01      82.9      82.9      82.9 -9999
## 1978-06-01      80.0      80.0      80.0 -9999
```

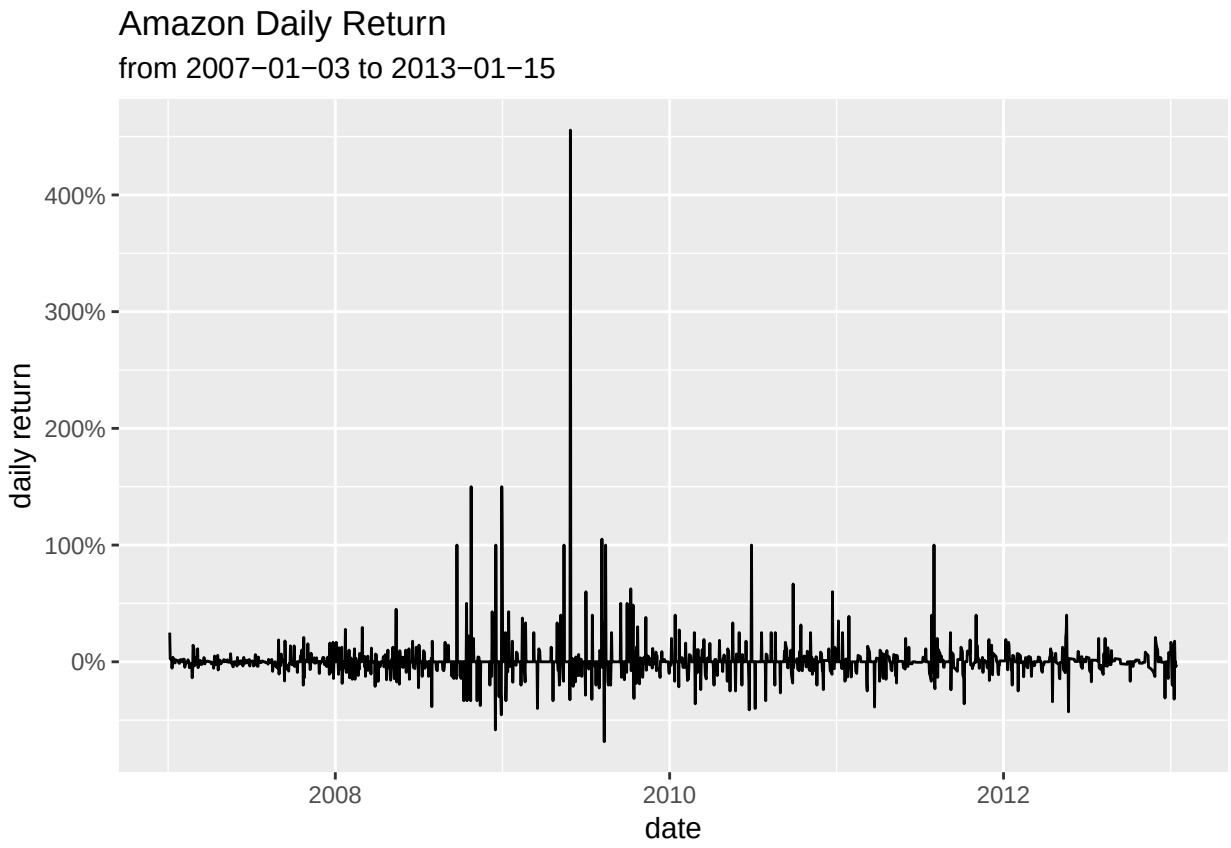
```
plot(AMAZ[, "AMAZ.Close"])
```

Next we need to handle missing values for the Amazon close values. From the plot above it seems like the time series frequency is high enough that a linear interpolation would be a good approximation of the NA values:

```
# pull close data for all days replacing -9999 with NA
AMAZ.Close <- ifelse(stocks[, "AMAZ.Close"] == -9999, NA, stocks[,
    "AMAZ.Close"])
# cut to range amazon had data for
AMAZ.Close <- AMAZ.Close[paste0(min(index(AMAZ)), "/", max(index(AMAZ)))]
# interpolate missing values
AMAZ_fixed <- na.approx(AMAZ.Close)
```

Now we can calculate and plot the daily return.

```
AMAZ_daily_return <- (AMAZ_fixed - lag(AMAZ_fixed))/lag(AMAZ_fixed)

AMAZ_daily_return %>% ggplot(aes(y = AMAZ.Close, x = index(AMAZ_daily_return))) +
    geom_line() + scale_y_continuous(labels = scales::percent) +
    labs(title = "Amazon Daily Return", subtitle = glue("from {min(index(AMAZ))} to {max(index
        x = "date", y = "daily return")
```



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

We choose to average over X-day intervals and just remove missing observations. That way we can interpret every point as the average of a range the same width even though each average may have a different number of points within it.

```
# 20 day rolling average
AMAZ_20_day_rolling_mean <- rollapply(AMAZ.Close, 20, FUN = mean,
    na.rm = TRUE)
names(AMAZ_20_day_rolling_mean) <- "AMAZ.Close.20.Day.Rolling.Average"
# 50 day
AMAZ_50_day_rolling_mean <- rollapply(AMAZ.Close, 50, FUN = mean,
    na.rm = TRUE)
names(AMAZ_50_day_rolling_mean) <- "AMAZ.Close.50.Day.Rolling.Average"
# plot together
AMAZ_close_w_means <- merge.xts(AMAZ.Close, AMAZ_20_day_rolling_mean,
    AMAZ_50_day_rolling_mean)
plot(AMAZ_close_w_means, main = "AMAZ Close Price with Rolling Averages",
    legend.loc = "topright")
legend("topright", c("Actual Close Price", "20 Day Rolling Average",
    "50 Day Rolling Average"), col = c(1, 2, 3))
```


AMAZ Close Price with Rolling Averages
2007−01−03 / 2013−01−15
AMAZ.Close
AMAZ.Close.20.Day.Rolling.Average
AMAZ.Close.50.Day.Rolling.Average
25
20
15
10
5
Jan 03
2007
Jan 01
2008
Jan 01
2009
Jan 01
2010
Jan 01
2011
Jan 01
2012
Dec 31
2012