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

Michael Berger, John Gao, and Thomas Hamnett

Contents

Question 1: Forecasting using a Seasonal ARIMA model **2**
Introduction . . . 2
Start Up Code . . . 2
Data Loading and Inspection . . . 2
Exploratory Time Series Data Analysis . . . 3
Model Building . . . 8
Model Selection . . . 9
Model Evaluation and Assumption Testing . . . 9
Forecasting . . . 12
Conclusion . . . 13

Question 2: Learning how to use the xts library **14**
Materials covered in Question 2 of this lab . . . 14

Task 1: **14**

A quick introduction to xts and zoo objects **14**
xts . . . 14

Task 2: **14**

Creating an xts object and converting to an xts object from an imported dataset **15**
Deconstructing xts . . . 16
Conversion to xts from other time-series objects . . . 17

Task 3: **19**

Merging and modifying time series **19**

Missing value imputation **23**
Calculate difference in time series . . . 31

Task 4: **40**

Apply various functions to time series **40**

Calculate basic rolling statistics of series by month **41**

Task 5: **42**

Question 1: Forecasting using a Seasonal ARIMA model

Introduction

In this third lab, first question, we will look into the quarterly data of e-commerce retail sales as a percent of total retail sales. The data can be found at: https://fred.stlouisfed.org/series/ECOMPCTNSA. We will build a **Seasonal Autoregressive Integrated Moving Average** (SARIMA) model. We will first load the necessary libraries as well as the data and explore the data. We will transform the data to a time series and withhold data of 2015 and 2016 as test set. Then we will conduct an **Exploratory Time Series Data Analysis** (ETSDA). Based on our insights from ETSDA as well as using an automatic search based on the Akaike information criterion with correction (AICc), we will then specify several candidate models. Next we will evaluate the models based on AIC and the Bayesian Information Criterion (BIC) and select our baseline model. After that we will explore in-sample-fit measures as well as the behavior of the residuals in order to explore whether the modelling assumptions are met. In addition, we will compare the forecast of the model on the test set using the **Root Mean Squared Error** (RMSE) as measure. We will discuss the performance. In case the residuals are well behaved and the out-of-sample test appears reasonable, we will finally produce the forecast for 2017.

Start Up Code

```
library(knitr); opts_chunk$set(tidy.opts=list(width.cutoff=60),
                               tidy=T,warning=FALSE,message=FALSE)
```

```
# cleaning workspace; Loading required libraries
rm(list = ls())
libs <- c('ggplot2', 'ggfortify', 'plotly', 'dplyr', 'astsa','fpp2','tidyr', 'tseries', 'foreca
for (lib in libs) {require(lib, character.only = TRUE)}
```

Data Loading and Inspection

```
# Loading the data
df <- read.csv("ECOMPCTNSA.csv", header = TRUE, sep = ",")
# inspecting it
str(df); head(df,4); tail(df,4); any(is.na(df))
```

```
## 'data.frame':    69 obs. of  2 variables:
##  $ DATE      : Factor w/ 69 levels "1999-10-01","2000-01-01",..: 1 2 3 4 5 6 7 8 9 10 ...
##  $ ECOMPCTNSA: num  0.7 0.8 0.8 0.9 1.1 1.1 1 1 1.3 1.3 ...
```

```
##         DATE ECOMPCTNSA
## 1 1999-10-01        0.7
## 2 2000-01-01        0.8
## 3 2000-04-01        0.8
## 4 2000-07-01        0.9
```

```
##          DATE ECOMPCTNSA
## 66 2016-01-01        7.7
## 67 2016-04-01        7.5
## 68 2016-07-01        7.7
## 69 2016-10-01        9.5
```

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

The data is stored as a `dataframe`. We note that the first observation is 10/01/1999 and no missing values are present in the data. The data ranges from 0.7 to 9.5 with a mean of 3.835 and a median of 3.6. The data ends in the fourth quarter 2016. The frequency of the data is quarterly.

Next we convert the data into a time series object and check that the transformation was successful.

```
# converting data to quarterly time series starting 1st Oct 1999,
# hence 4th Quarter 1999
df.ts <- ts(df$ECOMPCTNSA, start = c(1999,4), frequency = 4)
# inspecting transformed data
str(df.ts)
```

```
##  Time-Series [1:69] from 2000 to 2017: 0.7 0.8 0.8 0.9 1.1 1.1 1 1 1.3 1.3 ...
```

The transformation yielded a time series object. Next, we subset the data to only include the data up to end 2014. We will keep 2015 and 2016 for out-of-sample testing and train the model on the remaining data.

```
# Subsetting to training data
df.ts.train <- window(df.ts, end=c(2014,4))
```

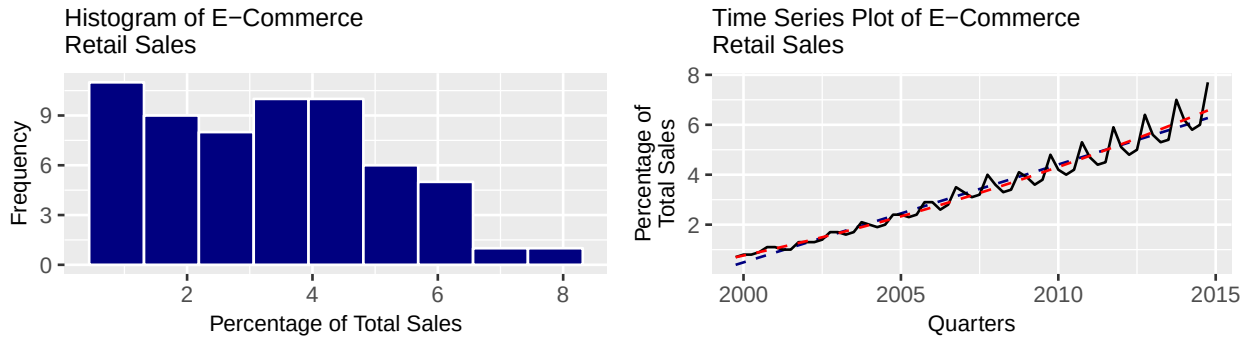
Next we conduct the ETSDA.

Exploratory Time Series Data Analysis

```
# Helper functions for visualizations
gHist <- function(input, title, nbin) {
  input %>% as.data.frame() %>% ggplot(aes(x = input)) +
  ggtitle(title) + ylab("Frequency") + xlab("Percentage of Total Sales") +
  geom_histogram(col = 'white', fill = 'navy', bins=nbin) +
  theme(plot.title = element_text(size=10),
        axis.title = element_text(size = rel(.8)))
}
gTs <- function(input, title) {
  input %>% as.data.frame() %>% ggplot(aes(x = time(input), y = input)) +
  ylab("Percentage of\nTotal Sales") + xlab("Quarters") + ggtitle(title) +
  geom_smooth(method="lm", se = FALSE, col = 'navy', lty = 2, size = 0.5) +
  geom_line() + theme(plot.title = element_text(size=10),
                      axis.title = element_text(size = rel(.8)))
}

g1 <- gHist(df.ts.train, 'Histogram of E-Commerce \nRetail Sales', 9)
```

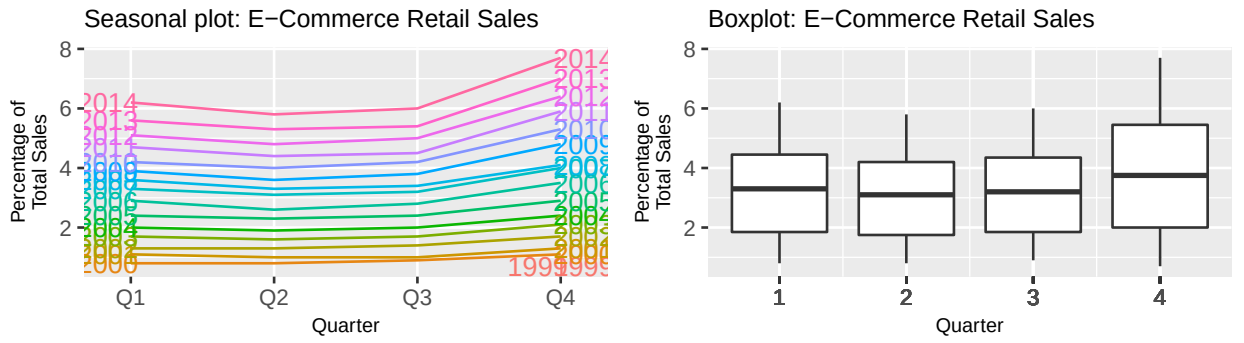
```
g2 <- gTs(df.ts.train, 'Time Series Plot of E-Commerce \nRetail Sales') +
  geom_smooth(method="lm", formula = y ~ x + I(x^2),
              se = FALSE, col = 'red', lty = 2, size = 0.5)
grid.arrange(g1, g2, ncol=2)
```



The histogram of the data looks skewed to the left with right tail. The time series plot shows a trend, which appears to be quadratic rather than linear. In addition, a seasonal pattern seems to be in place and the variance seems to be increasing with time. Hence, we might need to transform the data to deal with the increasing variance as well as explore whether first-differencing might remove the trend in order to yield a stationary series.

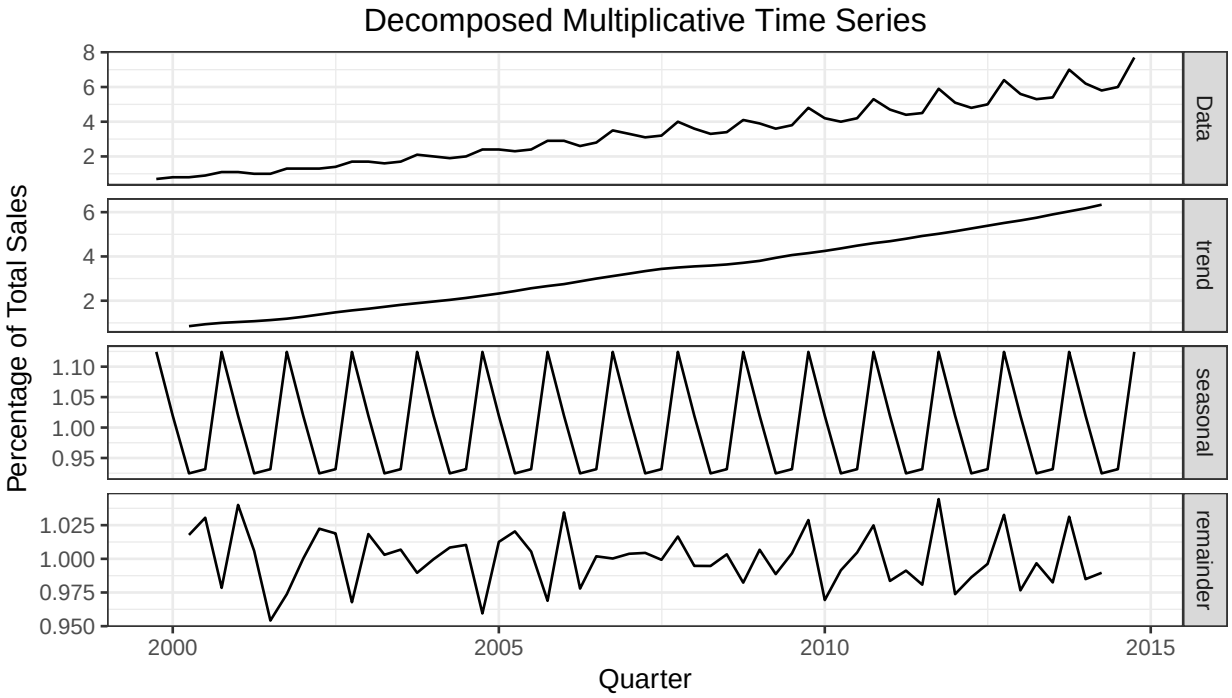
First, however, we explore the seasonality of the data.

```
# Examining seasonality
g1 <- ggseasonplot(df.ts.train, year.labels = TRUE,
                   year.labels.left = TRUE) +
  ylab("Percentage of\nTotal Sales") +
  ggtitle("Seasonal plot: E-Commerce Retail Sales") +
  theme(plot.title = element_text(size=10),
        axis.title = element_text(size=8))
cy <- cycle(df.ts.train)
g2 <- df.ts.train %>% as.data.frame() %>%
  ggplot(aes(x=cy, y=df.ts.train, group=cy)) + scale_x_continuous(breaks=cy) +
  ggtitle('Boxplot: E-Commerce Retail Sales') + geom_boxplot() +
  ylab("Percentage of\nTotal Sales") + xlab("Quarter") +
  theme(plot.title=element_text(size=10), axis.title=element_text(size=8))
grid.arrange(g1, g2, ncol=2, nrow=1)
```



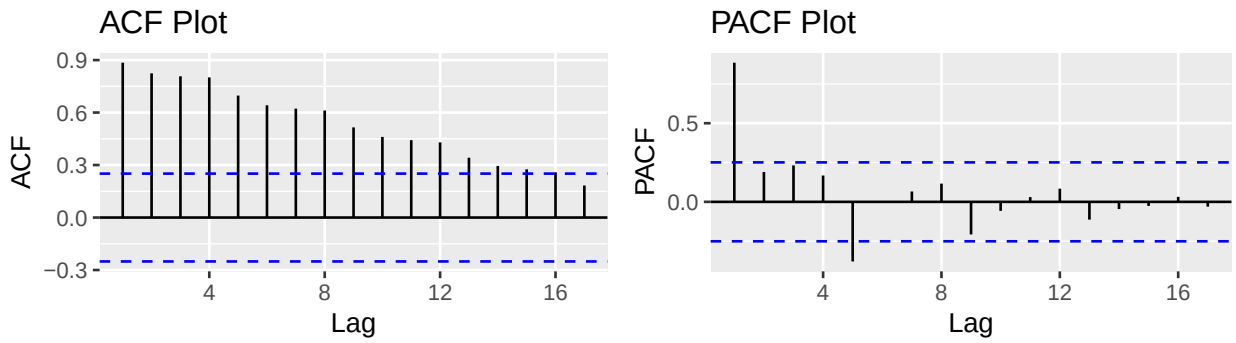
We again note the trend in the season plot. In the boxplot, we note a slight seasonal change, especially with higher median and higher inter-quartile-range for the fourth quarter.

```
decomp <- fortify(decompose(df.ts.train, type="multiplicative"))
ggplot(gather(decomp, key, value, -Index), aes(Index, value)) +
  facet_grid(factor(key, levels=c("Data","trend","seasonal",
                                  "remainder"))~ ., scales="free_y") +
  geom_line(na.rm = T) + ggtitle("Decomposed Multiplicative Time Series") +
  theme_bw() + ylab("Percentage of Total Sales") + xlab("Quarter") +
  theme(plot.title=element_text(hjust=0.5))
```



We decomposed the time series into seasonal, trend, and remainder compoenents using a multiplicative decomposition. We chose mulitplicative decomposition due to observing that the seasonality appeared to increase as the level of the data increased; that is, seasnoality appeared to be more a percetnage of level than having a constant magnitude. Of note, the seasonal component ranges from -8% to +12%, a relatively small effect given the difference in level of >10x in the actual data (min: 0.7; max: 7.7). Also, the remainder is more variable at the start and end of the series than the middle. This suggests, using this decomposition, the reamining error is not random. Remainder varies from -5% to +4%, which is realtively low compared to the seasonal component and suggests the trend and seasonal components do reasonably good job at explaining the time series data.

```
g1 <- ggAcf(df.ts.train, main = 'ACF Plot')
g2 <- ggPacf(df.ts.train, main = 'PACF Plot')
grid.arrange(g1, g2, ncol=2)
```



When looking at the ACF plot, we see it is trailing off, while the PACF plot shows significant spikes at lag 1 and lag 5, indicating seasonal effects. Besides the visual inspection, we also run the Augmented Dickey-Fuller-test to check for stationaity.

```
adf.test(df.ts.train, alternative = 'stationary')$p.value
```

```
## [1] 0.99
```

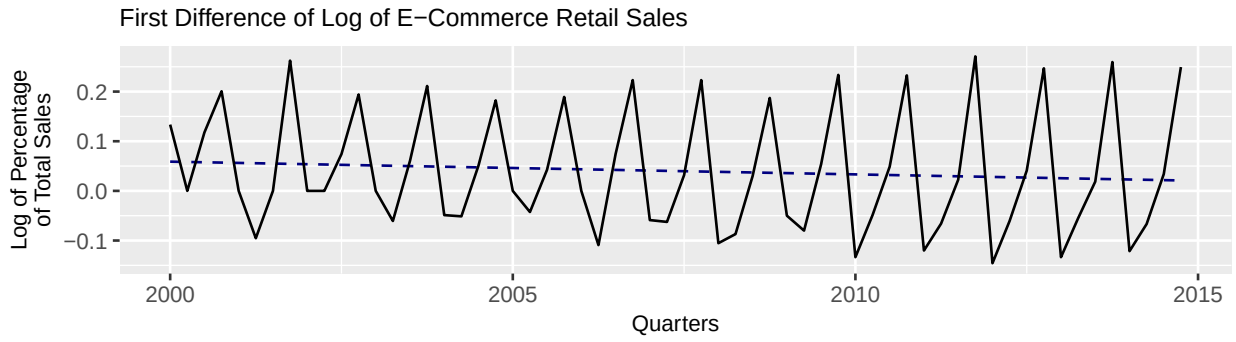
As seen in the visual inspection, the null hypothesis that time series is not stationary is not rejected given a very high p-value.

Given above analysis, we want now to check whether first-differencing would remove the trend and produce a stationary series. In addition, in order to deal with the increasing variance, we want to check whether a log-transformation can stabilize the variance. Hence, we try the first-differencing of the log-transformed series, denoted x_t, as transformation:

$$log(x_t) - log(x_{t-1}) = log(\frac{x_t}{x_{t-1}})$$

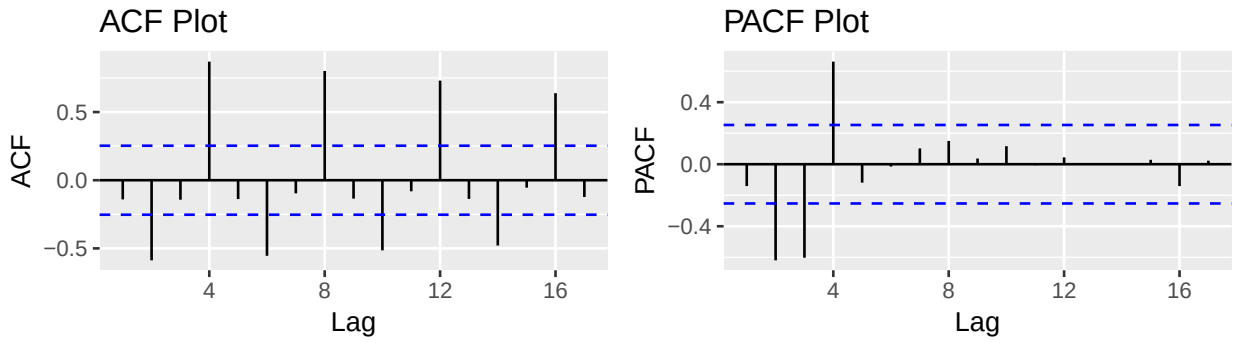
This yields the following time series plot.

```
trans.df.ts <- diff(log(df.ts.train))
print(gTs(trans.df.ts, 'First Difference of Log of E-Commerce Retail Sales') +
  ylab("Log of Percentage\nof Total Sales"))
```



First-differencing of logs of the series seems to remove a big part of the trend and indeed stabilizes the variance.

```
g1 <- ggAcf(trans.df.ts, main = 'ACF Plot')
g2 <- ggPacf(trans.df.ts, main = 'PACF Plot')
grid.arrange(g1, g2, ncol=2)
```



The first-difference of log of series still indicates the seasonal effect based on the ACF plot and shows negative partial autocrrelation with second and third lag and positive partial autocorrelation with fourth lag based on the PACF plot. Hence, the transformed series still shows a seasonal component.

We now test the transformed series in respect to stationarity.

```
adf.test(trans.df.ts, alternative = 'stationary')$p.value
```

```
## [1] 0.01
```

We see that the null hypothesis of non-stationarity is rejected. Hence, first-difference of log seems to have resulted in a stationary time series with seasonal component. We will also consider a Box-Cox transformation to see, if it suggests a different result for the transformation.

```
log.df.ts <- log(df.ts.train); lam <- BoxCox.lambda(df.ts.train)
BC.df.ts <- BoxCox(df.ts.train,lam); lam
```

```
## [1] 0.01467236
```

```
mean((BC.df.ts-log.df.ts)) #Avg Difference between log and BC
```

```
## [1] 0.01075712
```

We see that the suggestion is lambda = 0.015 for the Box-Cox transformation which is very close a log-transfromation (where lambda = 0 for Box-Cox). On average it also provides slighlty higher values after transformation of ~0.1 (although differences early in the time series are lower).

For interpretability, we will continue with the log transformation. The trade-off is that we might have a slightly better model performance with the Box-Cox transform, but the log transform is easier to interpret and explain as it models the percentage in differences.

Hence, we will use a log-transformed time series and apply a *seasonal ARIMA* model with $I(1)$.For a log-transformed time series $log(x_t)$ for which dth differencing yields an $ARMA(p, q)$ model, with seasonal component, the model is defined as:

$$\Theta_P(B^s)\Theta_p(B)(1-B^s)^D(1-B)^d log(x_t) = \Phi_Q(B^s)\Phi_q(B)w_t$$

with p, d and q relating to the non-seasonal and P, D, Q relating to the seasonal part of the model. Based on this model family and the ETSDA, we build several candidate models as well as use the `auto.arima` function to specify a candidate model based on the AICc, the AIC and the BIC.

Model Building

```
p1 = q1 = P1 = Q1 = 0:2; d1 = D1 = 0:1 #testing different parameters
fAIC <- function(x, mod,ic) {
  tmp <- try(Arima(mod, order=c(x[1],x[2],x[3]),
                   seasonal=c(x[4],x[5],x[6])),TRUE)
  if(isTRUE(class(tmp)=="try-error")) { return(NA) } #error handling
  else { ifelse(ic == "BIC", return(round(BIC(tmp),2)),
                return(round(AIC(tmp),2))) } #BIC or AIC
}
mod_df <- expand.grid(p=p1,d=d1,q=q1,P=P1,D=D1,Q=Q1)
mod_df$AIC <- apply(mod_df, MARGIN = 1, fAIC, mod=log.df.ts, ic="AIC")
mod_df$BIC <- apply(mod_df, MARGIN = 1, fAIC, mod=log.df.ts, ic="BIC")
#Top candidate models by AIC and BIC
head(mod_df %>% arrange(AIC),4); head(mod_df %>% arrange(BIC),4)
```

```
##   p d q P D Q     AIC     BIC
## 1 0 1 0 1 0 2 -210.12 -201.74
## 2 1 0 0 1 0 2 -209.76 -197.10
## 3 0 1 0 2 0 0 -209.20 -202.92
## 4 1 0 0 2 0 0 -208.84 -198.28
```

```
##   p d q P D Q     AIC     BIC
## 1 0 1 0 2 0 0 -209.20 -202.92
## 2 0 1 0 1 1 0 -206.22 -202.17
## 3 0 1 0 1 0 2 -210.12 -201.74
## 4 0 1 0 0 1 2 -207.22 -201.14
```

```
mod.auto <- auto.arima(log.df.ts)
setNames(data.frame(matrix(mod.auto$arma, nrow=1)),
         c('p','q','P','Q','S','d','D')) #auto.arima parameters
```

```
##   p q P Q S d D
## 1 0 0 2 0 4 1 1
```

```
AIC(mod.auto); BIC(mod.auto) #auto.arima AIC and BIC
```

```
## [1] -206.4998
```

```
## [1] -200.4238
```

After training the models, we will now select the most parsimonous model within the range of the best AIC and BIC performances.

Model Selection

We see that $ARIMA(0,1,0)(1,0,2)[4]$ has the lowest AIC and $ARIMA(0,1,0)(2,0,0)[4]$ has the lowest BIC (and third lowest AIC). The `auto.arima` function suggests $ARIMA(0,1,0)(2,1,0)[4]$, which yields the lowest AICc. As we want to select the most parsimonous model within the best performances, we select $ARIMA(0,1,0)(2,0,0)[4]$ as our baseline model.

```
baseline <- Arima(log.df.ts, order = c(0,1,0), seasonal = c(2,0,0))
summary(baseline)
```

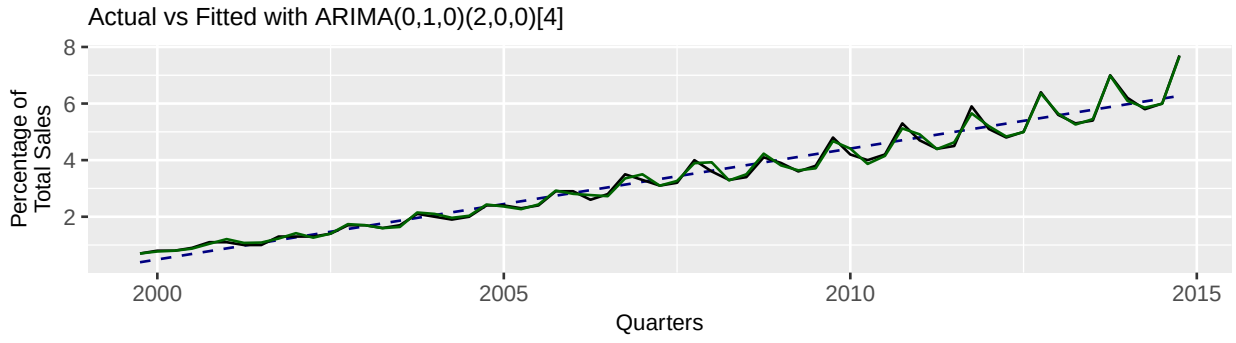
```
## Series: log.df.ts
## ARIMA(0,1,0)(2,0,0)[4]
##
## Coefficients:
##         sar1    sar2
##       0.3247  0.6535
## s.e.  0.1202  0.1221
##
## sigma^2 estimated as 0.001353:  log likelihood=107.6
## AIC=-209.2   AICc=-208.78   BIC=-202.92
##
## Training set error measures:
##                       ME       RMSE       MAE  MPE MAPE      MASE
## Training set -0.005625774 0.03587087 0.0268497 -Inf  Inf 0.1838643
##                     ACF1
## Training set -0.2186788
```

We see that both seasonal autoregressive variables are statistically significant. `sar2` appears relatively to be more significant given the computed standard error.

We next evaluate the model and look at the residuals.

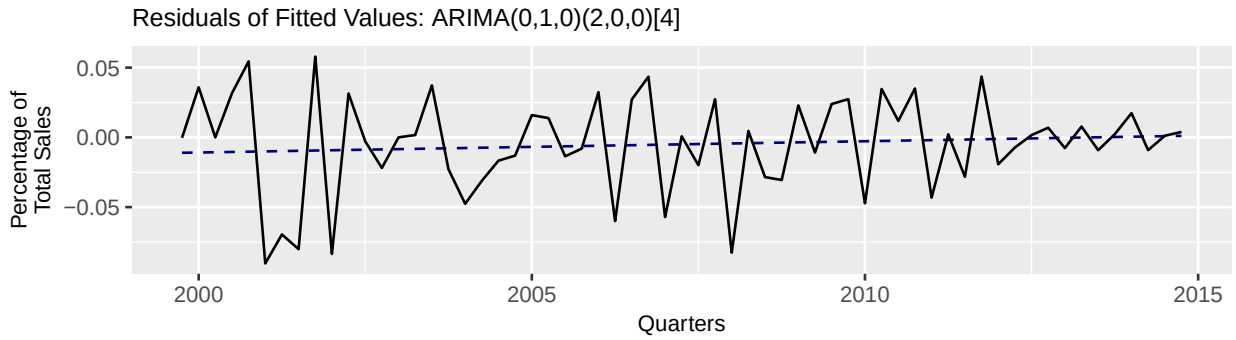
Model Evaluation and Assumption Testing

```
fitted.values <- exp(fitted(baseline))
print(gTs(df.ts.train, 'Actual vs Fitted with ARIMA(0,1,0)(2,0,0)[4]') +
  geom_line(aes(x=time(df.ts.train), y=fitted.values), col='darkgreen'))
```



We see that the model together with the log-transformation follows the time series pattern of the training data very closely and captures the increasing variance and seasonal patterns quite well.

```
res.baseline <- exp(residuals(baseline))-1
print(gTs(res.baseline, 'Residuals of Fitted Values: ARIMA(0,1,0)(2,0,0)[4]'))
```

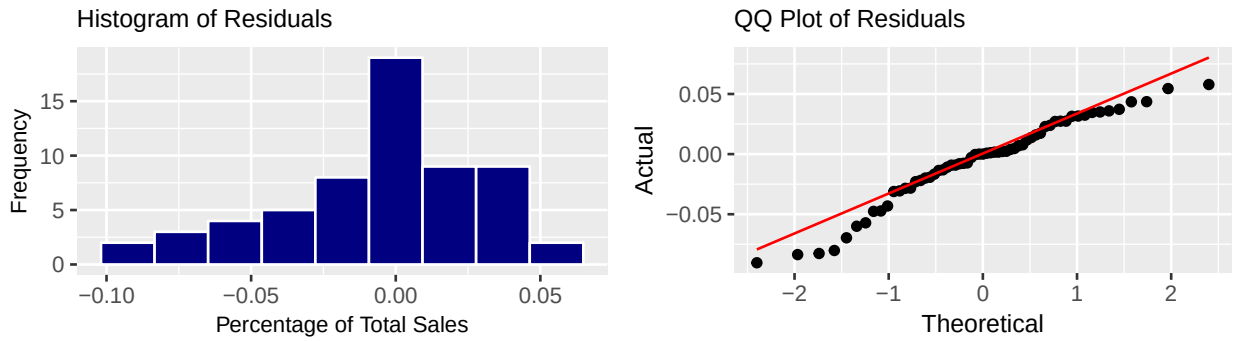



```
mean(res.baseline)
```

```
## [1] -0.004990885
```

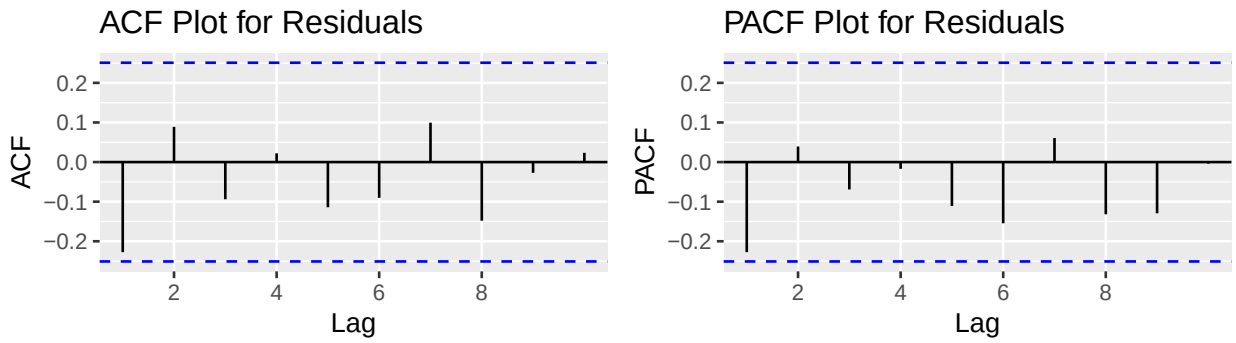
We see from the residual time series plot that the residuals still have a skew towards negative values and the volatility is decreasing. This indicates that the residuals are not necessarily white noise.

```
g1 <- gHist(res.baseline, 'Histogram of Residuals', 9)
g2 <- ggplot(fortify(res.baseline), aes(sample=Data)) + ylab("Actual") +
  stat_qq() + stat_qq_line(color='red') + ggtitle("QQ Plot of Residuals") +
  xlab("Theoretical") + theme(plot.title = element_text(size = 10),
                              axis.title = element_text(size = 10))
grid.arrange(g1, g2, ncol=2)
```



The histogram of residuals highlights the negative skew of residuals, there are tails on the right and left. This is also supported when looking at the normal Q-Q plot.

```
g1 <- ggAcf(res.baseline, main = 'ACF Plot for Residuals', lag.max = 10)
g2 <- ggPacf(res.baseline, main = 'PACF Plot for Residuals', lag.max = 10)
grid.arrange(g1, g2, ncol=2)
```




```
Box.test(res.baseline, type="Ljung-Box")$p.value
```

```
## [1] 0.06864694
```

When looking at the ACF and PACF plots, however, we see that no lag is significant. The Box-Ljung test rejects the null hypothesis that the residuals are non-stationary at the 10% level, but not at the 5% level.

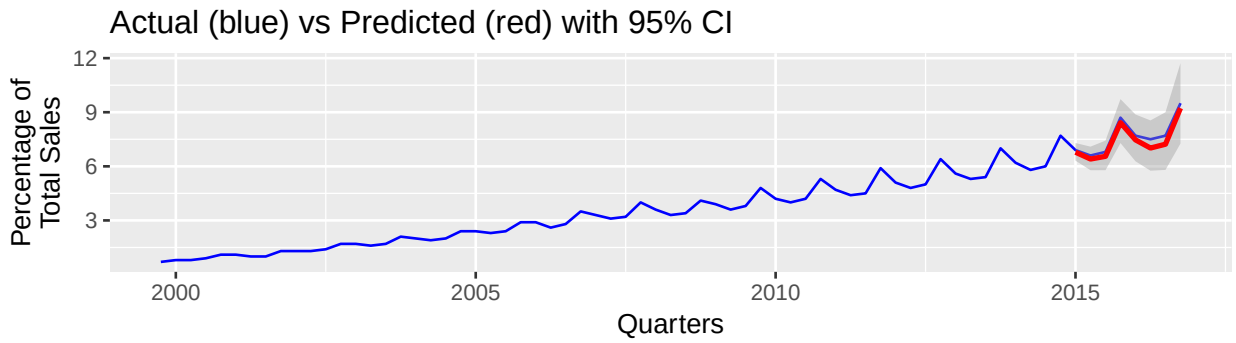
```
res.auto <- exp(residuals(mod.auto))-1
Box.test(res.auto, type="Ljung-Box")$p.value
```

```
## [1] 0.07828305
```

When re-evaluating the models we see that the autoselected $ARIMA(0,1,0)(2,1,0)[4]$ model would also reject the null-hypothesis of non-stationary residuals at the 10% level, but not the 5% level (with a higher p-value than our baseline model). The residual plots would not look much different from above. As both models indicate a similar structure of the residuals, we decided to go with the more parsimonous $ARIMA(0,1,0)(2,0,0)[4]$ model for the out-of-sample evaluation and the forecasting. We think that for the forecast a more parsimonous model might generalize better and hence do better in forecasting due to less parameters to be fitted.

```
test.forecast <- forecast(baseline, h=8)
fdf <- function(input, fcst) {
  out1 <- data.frame(time=time(input), values =input)
  out2 <- data.frame(time_mean=time(fcst$mean), values_mean=exp(fcst$mean),
                     time_upper=time(fcst$mean),time_lower=time(fcst$mean),
                     values_upper=exp(fcst$upper[,'95%']),
                     values_lower=exp(fcst$lower[,'95%']))
  return(list(out1, out2))
}
df1 <- fdf(df.ts, test.forecast)
gFcst <- function(input, pred) {
  ggplot(input, aes(x = time, y = values)) + geom_line(colour='blue') +
  geom_smooth(aes(x=time_mean, y=values_mean,
                  ymax=values_upper, ymin=values_lower),
              colour='red', data=pred, stat='identity') +
  ggtitle('Actual (blue) vs Predicted (red) with 95% CI') +
  ylab("Percentage of\nTotal Sales") + xlab("Quarters")
}
```

```
print(gFcst(df1[[1]], df1[[2]]))
```



For the out-of-sample testing we see that the mean of the forecast follows the actual values quite closely and the actual values are always within the 95% quantile. However, the downswings are less profound in the actual data compared to the model. This may indicate there is a larger underlying trend increase than captured in our log transformed model. The Box-Cox transformation may be necessary to better capture the true trend, since we saw that it yielded higher values than the log transform. We also note that each estimate is below the actual data, which will need to be considered when applying the model to the respective business problem (the model is then more conservative in terms of the forecast of the e-commerce percentage of total retail sales).

We calculate the RMSE as a metric of how well our model performs on the test data. The RMSE is defined as follows:

$$RMSE = \sqrt{\frac{1}{T}\sum_{t=1}^{T}(\hat{x_t} - x_t)^2}$$

with T being the number of time observations, $\hat{x_t}$ the fitted values and x_t the observed values. RMSE yields the average deviation of our model's forecasted percentage of e-commerce of total retail sales relative to the actual percentage of total sales.

```
y <- window(df.ts, start=c(2015,1))
y_hat <- tail(exp(test.forecast$mean),8)
sqrt(mean((y - y_hat)^2)) #RMSE
```

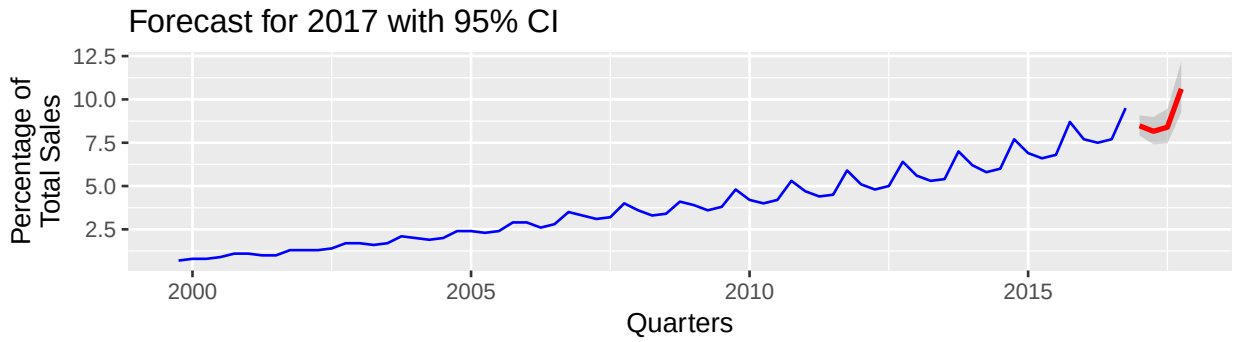
```
## [1] 0.3122734
```

The RMSE is 0.31. This means in average the model's forecast on the test data is 0.31 percentage points off in respect of the percentage of e-commerce of the total retail sales. We consider this to be a good performance in general, but it depends on the business problem to be solved whether the performance would be judged to be sufficient.

We will use this model for forecasting the 2017 values.

Forecasting

```
model.complete <- Arima(log(df.ts), order = c(0,1,0),
                        seasonal = c(1,1,0))
forecast.2017 = forecast(model.complete, h=4) # forecast 2017
df1 <- fdf(df.ts, forecast.2017)
print(gFcst(df1[[1]], df1[[2]]) +
  ggtitle('Forecast for 2017 with 95% CI'))
```



For the forecast we see that the model assumes the seasonal pattern to continue with a downswing in the first quarters and a strong upswing in the second half of the year.

Conclusion

We examined the quarterly data of E-Commerce Retail Sales, first splitting the data into test (last two years) and training sets. We found that the training data shows a trend and an increase in variance over time. By first-differencing the log of the data we could stabilize the variance and remove the trend resulting in a stationary model. In addition, we found seasonal patterns while conducting the ETSDA. Hence, we decided to fit a seasonal ARIMA model to the log-transformed data.

We found two candidate models, $ARIMA(0,1,0)(2,0,0)[4]$ and $ARIMA(0,1,0)(2,1,0)[4]$, which are quite close in terms of AIC and BIC. We decided to use $ARIMA(0,1,0)(2,0,0)[4]$ as baseline model as it is more parsimonous.

We then evaluated the model and found that it fits the training data very well, while the residuals look somewhat normal, albeit with longer tails on both sides. In addition, the Box-Ljung test fails to reject non-stationarity at the 5% level (but does so at the 10% level). Nevertheless, the out-of-sample performance based on RMSE is good in our mind. Thus, we used this model to conduct the 2017 forecast. The forecast predicts the trend, increase in variance and the seasonal pattern to hold with a downswing in the first half and a strong upswing in the second half of 2017.

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

- 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
- The first three questions in "xts FAQ"
  - What is xts?
  - Why should I use xts rather than zoo or another time-series package?
  - How do I install xts?
- 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

- Section 3.1 of "xts: Extensible Time Series" by Jeffrey A. Ryan and Joshua M. Ulrich
- The following questions in "xts FAQ"
  - How do I create an xts index with millisecond precision?

- 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 index yet.

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

```
##  num [1:100, 1:2] -0.7321 -1.5325 -0.1203 -0.0334 0.1342 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : NULL
##   ..$ : chr [1:2] "Series01" "Series02"
```

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

```
##          Series01   Series02
##  [1,] -0.73207883  0.5020909
##  [2,] -1.53245105 -0.2676996
##  [3,] -0.12031252 -0.5833745
##  [4,] -0.03338244  0.5609014
##  [5,]  0.13423335 -0.2372371
##  [6,] -1.27844197 -0.7227326
##  [7,] -0.79398186  0.2166008
##  [8,] -0.23688455  0.6529659
##  [9,]  0.06764128  1.6282735
## [10,]  0.71257844 -0.3948252
```

```
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)
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] -0.7321 -1.5325 -0.1203 -0.0334 0.1342 ...
##  - 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 -0.73207883  0.5020909
## 2015-01-02 -1.53245105 -0.2676996
## 2015-01-03 -0.12031252 -0.5833745
## 2015-01-04 -0.03338244  0.5609014
## 2015-01-05  0.13423335 -0.2372371
## 2015-01-06 -1.27844197 -0.7227326
## 2015-01-07 -0.79398186  0.2166008
## 2015-01-08 -0.23688455  0.6529659
## 2015-01-09  0.06764128  1.6282735
## 2015-01-10  0.71257844 -0.3948252
```

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,] -0.73207883  0.5020909
## [2,] -1.53245105 -0.2676996
## [3,] -0.12031252 -0.5833745
## [4,] -0.03338244  0.5609014
## [5,]  0.13423335 -0.2372371
```

`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
```

```r
#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 ...
```

```r
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
```

```r
# 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"
```

```r
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"
```

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

```
## [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 2013-11-07/2019-03-29 containing:
##   Data: num [1:1356, 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: "2019-04-01 15:19:27"
```

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.

```
# 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
```

```
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
```

```
#View(TWTR_unemp01)

# 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 2007-01-01/2019-03-29 containing:
##   Data: num [1:1455, 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_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
```

```r
#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
```

```r
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
```

```r
#View(TWTR_unemp03)

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

```
## An 'xts' object on 2013-11-07/2019-03-29 containing:
##   Data: num [1:1356, 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
```

```r
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
```

```r
str(unemp01)
```

```
## An 'xts' object on 2007-01-01/2019-03-29 containing:
##   Data: num [1:1455, 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
```

```r
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
```

```r
#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)
```

```
## 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
```

```
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
## 2007-07-01  4.9
## 2007-08-01  4.6
## 2007-09-01  4.5
## 2007-10-01  4.4
## 2007-11-01  4.5
## 2007-12-01  4.8
## 2008-01-01  5.4
## 2008-02-01  5.2
## 2008-03-01  5.2
## 2008-04-01  4.8
## 2008-05-01  5.2
## 2008-06-01  5.7
## 2008-07-01  6.0
## 2008-08-01  6.1
## 2008-09-01  6.0
## 2008-10-01  6.1
## 2008-11-01  6.5
## 2008-12-01  7.1
## 2009-01-01  8.5
## 2009-02-01  8.9
## 2009-03-01  9.0
## 2009-04-01  8.6
## 2009-05-01  9.1
## 2009-06-01  9.7
## 2009-07-01  9.7
## 2009-08-01  9.6
## 2009-09-01  9.5
## 2009-10-01  9.5
## 2009-11-01  9.4
## 2009-12-01  9.7
## 2010-01-01 10.6
## 2010-02-01 10.4
## 2010-03-01 10.2
## 2010-04-01  9.5
## 2010-05-01  9.3
## 2010-06-01  9.6
## 2010-07-01  9.7
## 2010-08-01  9.5
```

```
## 2010-09-01  9.2
## 2010-10-01  9.0
## 2010-11-01  9.3
## 2010-12-01  9.1
## 2011-01-01  9.8
## 2011-02-01  9.5
## 2011-03-01  9.2
## 2011-04-01  8.7
## 2011-05-01  8.7
## 2011-06-01  9.3
## 2011-07-01  9.3
## 2011-08-01  9.1
## 2011-09-01  8.8
## 2011-10-01  8.5
## 2011-11-01  8.2
## 2011-12-01  8.3
## 2012-01-01  8.8
## 2012-02-01  8.7
## 2012-03-01  8.4
## 2012-04-01  7.7
## 2012-05-01  7.9
## 2012-06-01  8.4
## 2012-07-01  8.6
## 2012-08-01  8.2
## 2012-09-01  7.6
## 2012-10-01  7.5
## 2012-11-01  7.4
## 2012-12-01  7.6
## 2013-01-01  8.5
## 2013-02-01  8.1
## 2013-03-01  7.6
## 2013-04-01  7.1
## 2013-05-01  7.3
## 2013-06-01  7.8
## 2013-07-01  7.7
## 2013-08-01  7.3
## 2013-09-01  7.0
## 2013-10-01  7.0
## 2013-11-01  6.6
## 2013-12-01  6.5
## 2014-01-01  7.0
## 2014-02-01  7.0
## 2014-03-01  6.8
## 2014-04-01  5.9
## 2014-05-01  6.1
## 2014-06-01  6.3
## 2014-07-01  6.5
## 2014-08-01  6.3
```

```
## 2014-09-01  5.7
## 2014-10-01  5.5
## 2014-11-01  5.5
## 2014-12-01  5.4
## 2015-01-01  6.1
## 2015-02-01  5.8
## 2015-03-01  5.6
## 2015-04-01  5.1
## 2015-05-01  5.3
## 2015-06-01  5.5
## 2015-07-01  5.6
## 2015-08-01  5.2
## 2015-09-01  4.9
## 2015-10-01  4.8
## 2015-11-01  4.8
## 2015-12-01  4.8
## 2016-01-01  5.3
## 2016-02-01  5.2
## 2016-03-01  5.1
## 2016-04-01  4.7
## 2016-05-01  4.5
## 2016-06-01  5.1
## 2016-07-01  5.1
## 2016-08-01  5.0
## 2016-09-01  4.8
## 2016-10-01  4.7
## 2016-11-01  4.4
## 2016-12-01  4.5
## 2017-01-01  5.1
```

```
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
## 2007-07-01  0.2
## 2007-08-01 -0.3
## 2007-09-01 -0.1
## 2007-10-01 -0.1
## 2007-11-01  0.1
## 2007-12-01  0.3
## 2008-01-01  0.6
## 2008-02-01 -0.2
## 2008-03-01  0.0
```

```
## 2008-04-01 -0.4
## 2008-05-01  0.4
## 2008-06-01  0.5
## 2008-07-01  0.3
## 2008-08-01  0.1
## 2008-09-01 -0.1
## 2008-10-01  0.1
## 2008-11-01  0.4
## 2008-12-01  0.6
## 2009-01-01  1.4
## 2009-02-01  0.4
## 2009-03-01  0.1
## 2009-04-01 -0.4
## 2009-05-01  0.5
## 2009-06-01  0.6
## 2009-07-01  0.0
## 2009-08-01 -0.1
## 2009-09-01 -0.1
## 2009-10-01  0.0
## 2009-11-01 -0.1
## 2009-12-01  0.3
## 2010-01-01  0.9
## 2010-02-01 -0.2
## 2010-03-01 -0.2
## 2010-04-01 -0.7
## 2010-05-01 -0.2
## 2010-06-01  0.3
## 2010-07-01  0.1
## 2010-08-01 -0.2
## 2010-09-01 -0.3
## 2010-10-01 -0.2
## 2010-11-01  0.3
## 2010-12-01 -0.2
## 2011-01-01  0.7
## 2011-02-01 -0.3
## 2011-03-01 -0.3
## 2011-04-01 -0.5
## 2011-05-01  0.0
## 2011-06-01  0.6
## 2011-07-01  0.0
## 2011-08-01 -0.2
## 2011-09-01 -0.3
## 2011-10-01 -0.3
## 2011-11-01 -0.3
## 2011-12-01  0.1
## 2012-01-01  0.5
## 2012-02-01 -0.1
## 2012-03-01 -0.3
```

```
## 2012-04-01 -0.7
## 2012-05-01  0.2
## 2012-06-01  0.5
## 2012-07-01  0.2
## 2012-08-01 -0.4
## 2012-09-01 -0.6
## 2012-10-01 -0.1
## 2012-11-01 -0.1
## 2012-12-01  0.2
## 2013-01-01  0.9
## 2013-02-01 -0.4
## 2013-03-01 -0.5
## 2013-04-01 -0.5
## 2013-05-01  0.2
## 2013-06-01  0.5
## 2013-07-01 -0.1
## 2013-08-01 -0.4
## 2013-09-01 -0.3
## 2013-10-01  0.0
## 2013-11-01 -0.4
## 2013-12-01 -0.1
## 2014-01-01  0.5
## 2014-02-01  0.0
## 2014-03-01 -0.2
## 2014-04-01 -0.9
## 2014-05-01  0.2
## 2014-06-01  0.2
## 2014-07-01  0.2
## 2014-08-01 -0.2
## 2014-09-01 -0.6
## 2014-10-01 -0.2
## 2014-11-01  0.0
## 2014-12-01 -0.1
## 2015-01-01  0.7
## 2015-02-01 -0.3
## 2015-03-01 -0.2
## 2015-04-01 -0.5
## 2015-05-01  0.2
## 2015-06-01  0.2
## 2015-07-01  0.1
## 2015-08-01 -0.4
## 2015-09-01 -0.3
## 2015-10-01 -0.1
## 2015-11-01  0.0
## 2015-12-01  0.0
## 2016-01-01  0.5
## 2016-02-01 -0.1
## 2016-03-01 -0.1
```

```
## 2016-04-01 -0.4
## 2016-05-01 -0.2
## 2016-06-01  0.6
## 2016-07-01  0.0
## 2016-08-01 -0.1
## 2016-09-01 -0.2
## 2016-10-01 -0.1
## 2016-11-01 -0.3
## 2016-12-01  0.1
## 2017-01-01  0.6
```

```
# 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)
```

```
##            [,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
## 2007-07-01   NA
## 2007-08-01   NA
## 2007-09-01   NA
## 2007-10-01   NA
## 2007-11-01   NA
## 2007-12-01   NA
## 2008-01-01  0.4
## 2008-02-01  0.3
## 2008-03-01  0.7
## 2008-04-01  0.5
## 2008-05-01  0.9
## 2008-06-01  1.0
## 2008-07-01  1.1
## 2008-08-01  1.5
## 2008-09-01  1.5
## 2008-10-01  1.7
## 2008-11-01  2.0
## 2008-12-01  2.3
## 2009-01-01  3.1
## 2009-02-01  3.7
## 2009-03-01  3.8
## 2009-04-01  3.8
## 2009-05-01  3.9
## 2009-06-01  4.0
```

```
## 2009-07-01  3.7
## 2009-08-01  3.5
## 2009-09-01  3.5
## 2009-10-01  3.4
## 2009-11-01  2.9
## 2009-12-01  2.6
## 2010-01-01  2.1
## 2010-02-01  1.5
## 2010-03-01  1.2
## 2010-04-01  0.9
## 2010-05-01  0.2
## 2010-06-01 -0.1
## 2010-07-01  0.0
## 2010-08-01 -0.1
## 2010-09-01 -0.3
## 2010-10-01 -0.5
## 2010-11-01 -0.1
## 2010-12-01 -0.6
## 2011-01-01 -0.8
## 2011-02-01 -0.9
## 2011-03-01 -1.0
## 2011-04-01 -0.8
## 2011-05-01 -0.6
## 2011-06-01 -0.3
## 2011-07-01 -0.4
## 2011-08-01 -0.4
## 2011-09-01 -0.4
## 2011-10-01 -0.5
## 2011-11-01 -1.1
## 2011-12-01 -0.8
## 2012-01-01 -1.0
## 2012-02-01 -0.8
## 2012-03-01 -0.8
## 2012-04-01 -1.0
## 2012-05-01 -0.8
## 2012-06-01 -0.9
## 2012-07-01 -0.7
## 2012-08-01 -0.9
## 2012-09-01 -1.2
## 2012-10-01 -1.0
## 2012-11-01 -0.8
## 2012-12-01 -0.7
## 2013-01-01 -0.3
## 2013-02-01 -0.6
## 2013-03-01 -0.8
## 2013-04-01 -0.6
## 2013-05-01 -0.6
## 2013-06-01 -0.6
```

```
## 2013-07-01 -0.9
## 2013-08-01 -0.9
## 2013-09-01 -0.6
## 2013-10-01 -0.5
## 2013-11-01 -0.8
## 2013-12-01 -1.1
## 2014-01-01 -1.5
## 2014-02-01 -1.1
## 2014-03-01 -0.8
## 2014-04-01 -1.2
## 2014-05-01 -1.2
## 2014-06-01 -1.5
## 2014-07-01 -1.2
## 2014-08-01 -1.0
## 2014-09-01 -1.3
## 2014-10-01 -1.5
## 2014-11-01 -1.1
## 2014-12-01 -1.1
## 2015-01-01 -0.9
## 2015-02-01 -1.2
## 2015-03-01 -1.2
## 2015-04-01 -0.8
## 2015-05-01 -0.8
## 2015-06-01 -0.8
## 2015-07-01 -0.9
## 2015-08-01 -1.1
## 2015-09-01 -0.8
## 2015-10-01 -0.7
## 2015-11-01 -0.7
## 2015-12-01 -0.6
## 2016-01-01 -0.8
## 2016-02-01 -0.6
## 2016-03-01 -0.5
## 2016-04-01 -0.4
## 2016-05-01 -0.8
## 2016-06-01 -0.4
## 2016-07-01 -0.5
## 2016-08-01 -0.2
## 2016-09-01 -0.1
## 2016-10-01 -0.1
## 2016-11-01 -0.4
## 2016-12-01 -0.3
## 2017-01-01 -0.2
```

Task 4:

1. Read

- Section 3.4 of "xts: Extensible Time Series" by Jeffrey A. Ryan and Joshua M. Ulrich
- The following questions in "xts FAQ"
  - 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

# 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 [note: Lab3.pdf setnence cuts off here]

```
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 43.28 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

```
az_df <- read.csv("AMAZ.csv", header=TRUE, stringsAsFactors = FALSE)
umc_df <- read.csv("UMCSENT.csv", header=TRUE, stringsAsFactors = FALSE)

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

```
## 'data.frame':    1179 obs. of  6 variables:
##  $ Index      : chr  "2007-01-03" "2007-01-04" "2007-01-08" "2007-01-09" ...
##  $ AMAZ.Open  : num  20 20 19.2 22 20.8 20.8 22 21.6 22 23.2 ...
##  $ AMAZ.High  : num  20 20 22 22 20.8 21.6 22 21.6 22 23.2 ...
##  $ AMAZ.Low   : num  16 20 19.2 20.8 20.8 20.8 22 21.2 21.6 22.8 ...
##  $ AMAZ.Close : num  16 20 22 20.8 20.8 21.6 22 21.2 21.6 22.8 ...
##  $ AMAZ.Volume: int  650 67 1801 356 438 2318 306 925 2138 527 ...
```

```
names(az_df)
```

```
## [1] "Index"       "AMAZ.Open"   "AMAZ.High"   "AMAZ.Low"    "AMAZ.Close"
## [6] "AMAZ.Volume"
```

```
head(az_df)
```

```
##        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
```

```
tail(az_df)
```

```
##           Index AMAZ.Open AMAZ.High AMAZ.Low AMAZ.Close AMAZ.Volume
## 1174 2013-01-04      0.88      0.88     0.80       0.80        3850
## 1175 2013-01-07      0.80      1.00     0.80       1.00        2715
## 1176 2013-01-08      0.80      0.80     0.68       0.68        4668
## 1177 2013-01-09      0.88      0.88     0.80       0.80        2750
## 1178 2013-01-11      0.80      0.80     0.80       0.80        3000
## 1179 2013-01-15      0.68      0.68     0.68       0.68        1000
```

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

```
## 'data.frame':    477 obs. of  2 variables:
##  $ Index  : chr  "1978-01-01" "1978-02-01" "1978-03-01" "1978-04-01" ...
##  $ UMCSENT: num  83.7 84.3 78.8 81.6 82.9 80 82.4 78.4 80.4 79.3 ...
```

```
names(umc_df)
```

```
## [1] "Index"   "UMCSENT"
```

```
head(umc_df)
```

```
##        Index UMCSENT
## 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    80.0
```

```
tail(umc_df)
```

```
##          Index UMCSENT
## 472 2017-04-01    97.0
## 473 2017-05-01    97.1
## 474 2017-06-01    95.1
## 475 2017-07-01    93.4
## 476 2017-08-01    96.8
## 477 2017-09-01    95.1
```

2. Convert them to xts objects

```
az_idx <- as.Date(az_df$Index)
az_xts <- xts(az_df[,2:6], order.by = az_idx)
str(az_xts)
```

```
## An 'xts' object on 2007-01-03/2013-01-15 containing:
##   Data: num [1:1179, 1:5] 20 20 19.2 22 20.8 20.8 22 21.6 22 23.2 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : NULL
##   ..$ : chr [1:5] "AMAZ.Open" "AMAZ.High" "AMAZ.Low" "AMAZ.Close" ...
```

```
##   Indexed by objects of class: [Date] TZ: UTC
##   xts Attributes:
##  NULL
```

```
head(coredata(az_xts))
```

```
##      AMAZ.Open AMAZ.High AMAZ.Low AMAZ.Close AMAZ.Volume
## [1,]      20.0      20.0     16.0       16.0         650
## [2,]      20.0      20.0     20.0       20.0          67
## [3,]      19.2      22.0     19.2       22.0        1801
## [4,]      22.0      22.0     20.8       20.8         356
## [5,]      20.8      20.8     20.8       20.8         438
## [6,]      20.8      21.6     20.8       21.6        2318
```

```
tail(coredata(az_xts))
```

```
##         AMAZ.Open AMAZ.High AMAZ.Low AMAZ.Close AMAZ.Volume
## [1174,]      0.88      0.88     0.80       0.80        3850
## [1175,]      0.80      1.00     0.80       1.00        2715
## [1176,]      0.80      0.80     0.68       0.68        4668
## [1177,]      0.88      0.88     0.80       0.80        2750
## [1178,]      0.80      0.80     0.80       0.80        3000
## [1179,]      0.68      0.68     0.68       0.68        1000
```

```
head(index(az_xts))
```

```
## [1] "2007-01-03" "2007-01-04" "2007-01-08" "2007-01-09" "2007-01-10"
## [6] "2007-01-11"
```

```
tail(index(az_xts))
```

```
## [1] "2013-01-04" "2013-01-07" "2013-01-08" "2013-01-09" "2013-01-11"
## [6] "2013-01-15"
```

```
umc_xts <- xts(umc_df$UMCSENT, order.by = as.Date(umc_df$Index))
str(umc_xts)
```

```
## An 'xts' object on 1978-01-01/2017-09-01 containing:
##   Data: num [1:477, 1] 83.7 84.3 78.8 81.6 82.9 80 82.4 78.4 80.4 79.3 ...
##   Indexed by objects of class: [Date] TZ: UTC
##   xts Attributes:
##  NULL
```

```
head(coredata(umc_xts))
```

```
##      [,1]
## [1,] 83.7
## [2,] 84.3
## [3,] 78.8
## [4,] 81.6
## [5,] 82.9
## [6,] 80.0
```

```
tail(coredata(umc_xts))
```

```
##        [,1]
## [472,] 97.0
## [473,] 97.1
## [474,] 95.1
## [475,] 93.4
## [476,] 96.8
## [477,] 95.1
```

```
head(index(umc_xts))
```

```
## [1] "1978-01-01" "1978-02-01" "1978-03-01" "1978-04-01" "1978-05-01"
## [6] "1978-06-01"
```

```
tail(index(umc_xts))
```

```
## [1] "2017-04-01" "2017-05-01" "2017-06-01" "2017-07-01" "2017-08-01"
## [6] "2017-09-01"
```

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

```
mrgix01 <- merge(umc_xts,index(az_xts),join='outer',fill = -9999)
mrg01 <- merge(mrgix01,az_xts,join='left')
```

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

```
UMCSENT02 <- mrg01$umc_xts
UMCSENT02[UMCSENT02 <= -9999] <- NA
```

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

```
UMCSENT03 <- UMCSENT02
UMCSENT03 <- na.locf(UMCSENT02, na.rm = TRUE, fromLast = FALSE)
```

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

```
UMCSENT04 <- UMCSENT02
UMCSENT04 <-na.approx(UMCSENT04, maxgap=31)
```

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!)

```
#Before AMAZ series and then first couple months
mrg01['2006-11-01/2007-03-01']
```

```
##            umc_xts AMAZ.Open AMAZ.High AMAZ.Low AMAZ.Close AMAZ.Volume
## 2006-11-01    92.1        NA        NA       NA          NA          NA
## 2006-12-01    91.7        NA        NA       NA          NA          NA
## 2007-01-01    96.9        NA        NA       NA          NA          NA
```

```
## 2007-01-03 -9999.0      20.0      20.0      16.0      16.0         650
## 2007-01-04 -9999.0      20.0      20.0      20.0      20.0          67
## 2007-01-08 -9999.0      19.2      22.0      19.2      22.0        1801
## 2007-01-09 -9999.0      22.0      22.0      20.8      20.8         356
## 2007-01-10 -9999.0      20.8      20.8      20.8      20.8         438
## 2007-01-11 -9999.0      20.8      21.6      20.8      21.6        2318
## 2007-01-12 -9999.0      22.0      22.0      22.0      22.0         306
## 2007-01-16 -9999.0      21.6      21.6      21.2      21.2         925
## 2007-01-17 -9999.0      22.0      22.0      21.6      21.6        2138
## 2007-01-22 -9999.0      23.2      23.2      22.8      22.8         527
## 2007-01-23 -9999.0      22.8      22.8      22.8      22.8          38
## 2007-01-26 -9999.0      22.0      22.0      22.0      22.0         250
## 2007-01-29 -9999.0      22.8      23.2      22.0      23.2         986
## 2007-01-31 -9999.0      23.6      24.0      23.6      24.0         125
## 2007-02-01    91.3      24.0      24.0      24.0      24.0         270
## 2007-02-02 -9999.0      23.6      24.0      23.6      24.0         729
## 2007-02-05 -9999.0      24.0      25.6      24.0      25.6         375
## 2007-02-06 -9999.0      24.4      24.4      24.4      24.4         142
## 2007-02-09 -9999.0      24.0      24.0      23.2      23.6        2690
## 2007-02-12 -9999.0      23.6      23.6      23.2      23.2         251
## 2007-02-13 -9999.0      22.8      23.6      22.8      23.6         427
## 2007-02-14 -9999.0      23.6      23.6      23.6      23.6         500
## 2007-02-15 -9999.0      23.6      23.6      23.6      23.6          40
## 2007-02-16 -9999.0      23.2      23.6      22.4      22.4         850
## 2007-02-20 -9999.0      22.4      22.4      20.8      20.8         312
## 2007-02-21 -9999.0      20.8      20.8      20.4      20.4         238
## 2007-02-22 -9999.0      18.8      18.8      16.8      17.6         625
## 2007-02-23 -9999.0      18.8      18.8      16.0      16.0         375
## 2007-02-26 -9999.0      16.0      22.8      16.0      22.8        2302
## 2007-02-27 -9999.0      22.8      22.8      20.8      22.0         700
## 2007-02-28 -9999.0      22.0      22.0      22.0      22.0         540
## 2007-03-01    88.4      22.0      22.0      22.0      22.0          62
```

```
UMCSENT02['2006-11-01/2007-03-01']
```

```
##            umc_xts
## 2006-11-01    92.1
## 2006-12-01    91.7
## 2007-01-01    96.9
## 2007-01-03      NA
## 2007-01-04      NA
## 2007-01-08      NA
## 2007-01-09      NA
## 2007-01-10      NA
## 2007-01-11      NA
## 2007-01-12      NA
## 2007-01-16      NA
## 2007-01-17      NA
```

```
## 2007-01-22      NA
## 2007-01-23      NA
## 2007-01-26      NA
## 2007-01-29      NA
## 2007-01-31      NA
## 2007-02-01    91.3
## 2007-02-02      NA
## 2007-02-05      NA
## 2007-02-06      NA
## 2007-02-09      NA
## 2007-02-12      NA
## 2007-02-13      NA
## 2007-02-14      NA
## 2007-02-15      NA
## 2007-02-16      NA
## 2007-02-20      NA
## 2007-02-21      NA
## 2007-02-22      NA
## 2007-02-23      NA
## 2007-02-26      NA
## 2007-02-27      NA
## 2007-02-28      NA
## 2007-03-01    88.4
```

```
UMCSENT03['2006-11-01/2007-03-01']
```

```
##            umc_xts
## 2006-11-01    92.1
## 2006-12-01    91.7
## 2007-01-01    96.9
## 2007-01-03    96.9
## 2007-01-04    96.9
## 2007-01-08    96.9
## 2007-01-09    96.9
## 2007-01-10    96.9
## 2007-01-11    96.9
## 2007-01-12    96.9
## 2007-01-16    96.9
## 2007-01-17    96.9
## 2007-01-22    96.9
## 2007-01-23    96.9
## 2007-01-26    96.9
## 2007-01-29    96.9
## 2007-01-31    96.9
## 2007-02-01    91.3
## 2007-02-02    91.3
## 2007-02-05    91.3
## 2007-02-06    91.3
```

```
## 2007-02-09    91.3
## 2007-02-12    91.3
## 2007-02-13    91.3
## 2007-02-14    91.3
## 2007-02-15    91.3
## 2007-02-16    91.3
## 2007-02-20    91.3
## 2007-02-21    91.3
## 2007-02-22    91.3
## 2007-02-23    91.3
## 2007-02-26    91.3
## 2007-02-27    91.3
## 2007-02-28    91.3
## 2007-03-01    88.4
```

```
UMCSENT04['2006-11-01/2007-03-01']
```

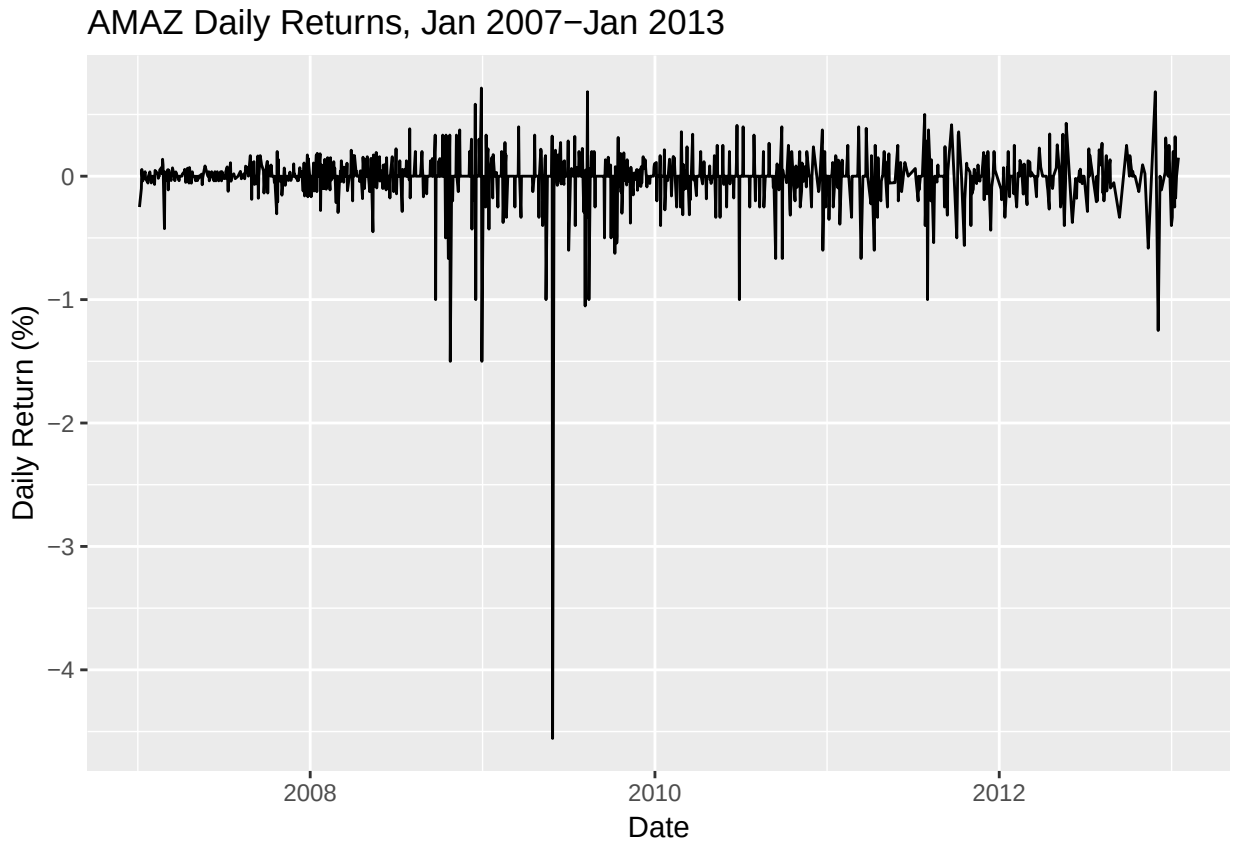
```
##             umc_xts
## 2006-11-01 92.10000
## 2006-12-01 91.70000
## 2007-01-01 96.90000
## 2007-01-03 96.53871
## 2007-01-04 96.35806
## 2007-01-08 95.63548
## 2007-01-09 95.45484
## 2007-01-10 95.27419
## 2007-01-11 95.09355
## 2007-01-12 94.91290
## 2007-01-16 94.19032
## 2007-01-17 94.00968
## 2007-01-22 93.10645
## 2007-01-23 92.92581
## 2007-01-26 92.38387
## 2007-01-29 91.84194
## 2007-01-31 91.48065
## 2007-02-01 91.30000
## 2007-02-02 91.19643
## 2007-02-05 90.88571
## 2007-02-06 90.78214
## 2007-02-09 90.47143
## 2007-02-12 90.16071
## 2007-02-13 90.05714
## 2007-02-14 89.95357
## 2007-02-15 89.85000
## 2007-02-16 89.74643
## 2007-02-20 89.33214
## 2007-02-21 89.22857
## 2007-02-22 89.12500
```

```
## 2007-02-23 89.02143
## 2007-02-26 88.71071
## 2007-02-27 88.60714
## 2007-02-28 88.50357
## 2007-03-01 88.40000
```

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.

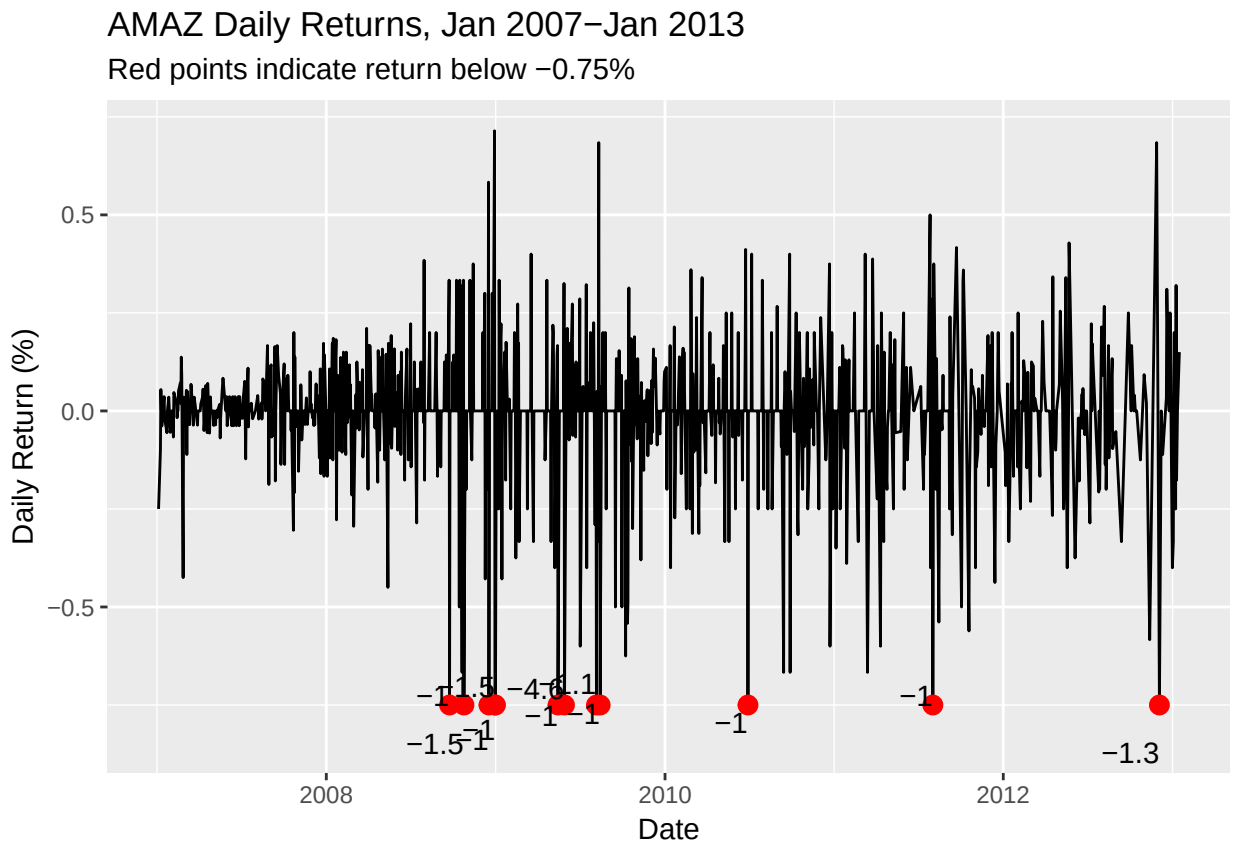
```
rtrn <- 1 - diff.xts(az_xts$AMAZ.Close, lag = 1, difference = 1,
                arithmetic = F, na.pad=F)
rtrn2 <- rtrn
rtrn2 <- fortify(rtrn2)

autoplot.zoo(rtrn, main = "AMAZ Daily Returns, Jan 2007-Jan 2013") +
  labs(y = "Daily Return (%)", x = "Date")
```




```
rtrn2 %>% mutate(pt=ifelse(abs(AMAZ.Close)>=.75,
                           .75*sign(AMAZ.Close),NA)) %>%
  mutate(ln=ifelse(abs(AMAZ.Close)>=.75,
                           .75*sign(AMAZ.Close),AMAZ.Close)) %>%
  ggplot() +
  geom_line(aes(x = Index, y = ln)) +
  geom_point(aes(x = Index, y = pt), na.rm = T,
```

```
            color='red', size=3, label=pt) +
  labs(title = "AMAZ Daily Returns, Jan 2007-Jan 2013",
       y ="Daily Return (%)", x = "Date",
       subtitle="Red points indicate return below -0.75%") +
  geom_text(aes(x = Index, y=pt, label=round(AMAZ.Close,1)),
            position=position_jitter(width=.02,height=.1),
            hjust=1, vjust=1, size=4)
```



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

```
rm20 <- rollmean(az_xts$AMAZ.Close, k=20)
head(rm20)
```

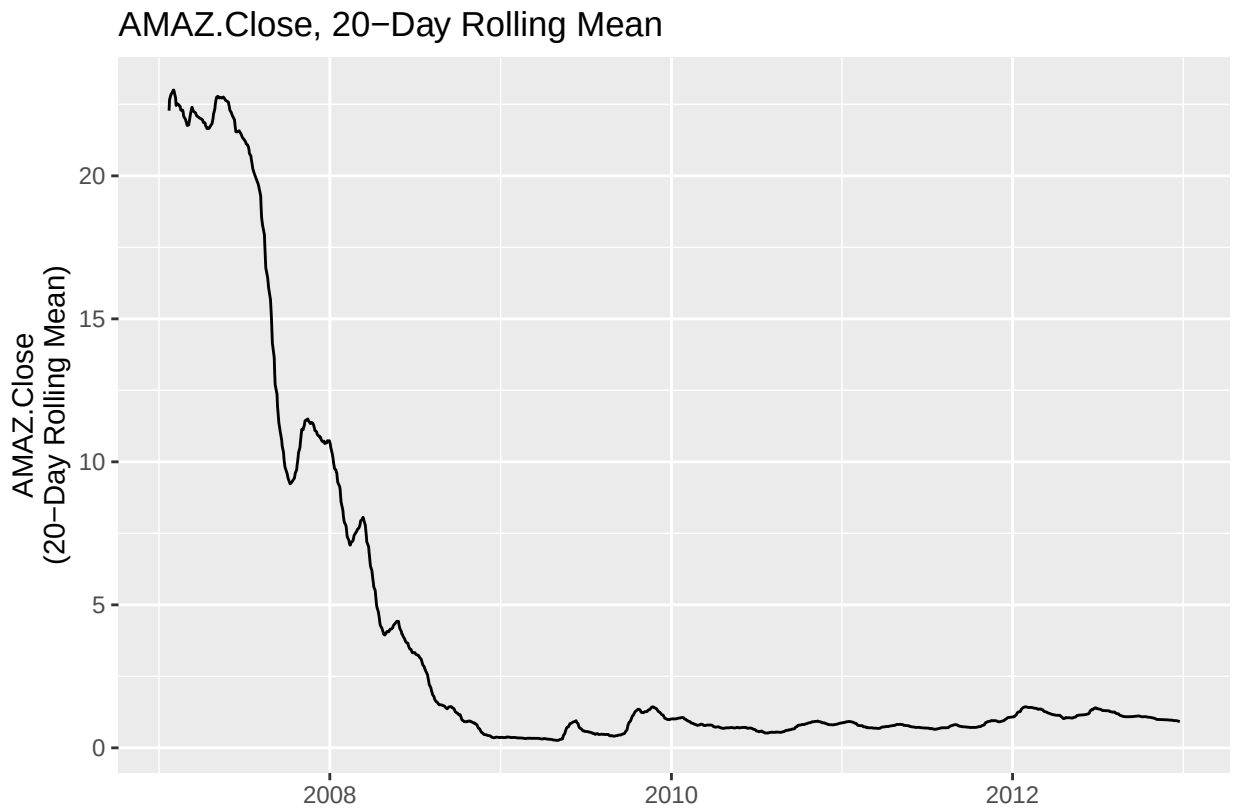
```
##            AMAZ.Close
## 2007-01-22      22.28
## 2007-01-23      22.66
## 2007-01-26      22.84
## 2007-01-29      22.92
## 2007-01-31      23.00
## 2007-02-01      23.00
```

```
tail(rm20)
```

```
##            AMAZ.Close
## 2012-12-03      0.969
```

```
## 2012-12-07      0.969
## 2012-12-10      0.955
## 2012-12-18      0.947
## 2012-12-19      0.939
## 2012-12-24      0.919
```

```
autoplot(rm20, main="AMAZ.Close, 20-Day Rolling Mean",
         ylab="AMAZ.Close\n(20-Day Rolling Mean)")
```




```
rm50 <- rollmean(az_xts$AMAZ.Close, k=50)
head(rm50)
```

```
##            AMAZ.Close
## 2007-02-20    22.0520
## 2007-02-21    22.1640
## 2007-02-22    22.1680
## 2007-02-23    22.1488
## 2007-02-26    22.1648
## 2007-02-27    22.2048
```

```
tail(rm50)
```

```
##            AMAZ.Close
## 2012-08-24     1.1512
## 2012-08-31     1.1432
```

```
## 2012-09-12     1.1304
## 2012-09-27     1.1200
## 2012-10-03     1.1080
## 2012-10-04     1.0936
```

```
autoplot(rm50, main="AMAZ.Close, 50-Day Rolling Mean",
         ylab="AMAZ.Close\n(50-Day Rolling Mean)")
```