Applied Time Series Analysis and Forecasting with R

Koji Mizumura

2018.08.09

<https://eylearning.udemy.com/r-applied-time-series-analysis-forecasting-r-projects-r-tutorials/learn/v4/t/lecture/11010476?start=0>

# Section 1

## Lecture 3: Main Functions

1. Basic functions in R Base are:

* decompose()
* stl()
* arima()
* HolWinters()
* acf(), pacf()
* plot()
* ts()

1. Add-on package - forecast package

# package load  
library(forecast)

## Warning: package 'forecast' was built under R version 3.5.1

library(ggplot2)

## Warning: package 'ggplot2' was built under R version 3.5.1

### Forecast structure

Function forecast() + standard model = Forecasting

Standar model is (1) ARIMA, (2) Sesonal decomposition, (3) Exponential smoothing, and (4) simple models (e.g., naive mean).

Forecasting Results are always in the same structure regardless of the model(which are different from function predict()).

In the Forecast library, automatic parameter selection is available both for the ARIMA model and the exponential smoothing.

### ARIMA models in Library Forecast

auto.arima(time\_series) -> most suitable ARIMA model auto.arima sets the complexity with the arguments stepwise and approximation. WE can get a list of possible models, as well as the information criteria.

Arima(time\_series) we could set the parameters manually (manual parameter selecion). In the manual selection, we look fro autocorrelation based on the acf() and pacf() plots. we adjust the model by time\_series - lags until no more autocorrelation.

### Exponential smooothing in Library Forecast

Automatic function - ets() manual functions: ses(), hw(), holt()

### Plotting with Library Forrecast

R Base Plots: - plot() - monthplot() - seasonplot()

ggplot2 plots: - autoplot() - ggmothplot() - ggseasonplot()

### Model comparison with Library Forecast

accruacy() getting the model accuracy with a training and a test set.

tsCV() time series cross validation for small datasets.

### Other packages

we would use getSymbols(), quantmod() and xts() packages.

# install.packages("quantmod")  
# install.packages("xts")

## Lecture 4: Supporting Resources

R Time series Task View curated by Rob Hyndman (Blog on Otexts.org) <https://robjhyndman.com/>

Vignetts some packages come with vignettes (description, PDF)

Free e-Books “Forecasting: Principles and Practice” <https://otexts.org/fpp2/index.html> <https://robjhyndman.com/seminars/uwa2017/>

R Community on Stackoverflow.com <https://stackoverflow.com/questions/tagged/r>

## Lecture 5: Course Link List

Time Series Task View: <https://cran.r-project.org/web/views/TimeSeries.html>

Blog, Ebook and Forecast Documentation by Rob Hyndman: <https://otexts.org/fpp2/intro.html>

Stackoverflow Community: <https://stackoverflow.com/questions>

Singapur Data of Project I: <https://docs.google.com/spreadsheets/d/1frieoKODnD9sX3VCZy5c3QAjBXMY-vN7k_I9gR-gcU8/pub> <http://www.gapminder.org/data/>

German Inflation Data of Project II: <https://www.statbureau.org/en/germany/inflation-tables>

# Section 2: Project I Trending <Data:Singapur> Labor Force Participation Rate

## LEcture 6: Importing the data

Uneployment rate - Used for propaganda purposes - Easy to manipulate - Who is unemployed? - Who doesn’t show up in the metric?

Labor Force Participatin Rate - Harder to manipulate - Ratio of people in the work force vs available people of a particular age range - Factors for manipulation /bias <https://www.gapminder.org/data/>

We need to compara things that are the same or similar level. Unbalanced comparisons require well thought out methodology to adjust for a mismatch.

# package load  
library(forecast)  
library(ggplot2)  
library(quantmod)

## Warning: package 'quantmod' was built under R version 3.5.1

## Loading required package: xts

## Warning: package 'xts' was built under R version 3.5.1

## Loading required package: zoo

## Warning: package 'zoo' was built under R version 3.5.1

##   
## Attaching package: 'zoo'

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

## Loading required package: TTR

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

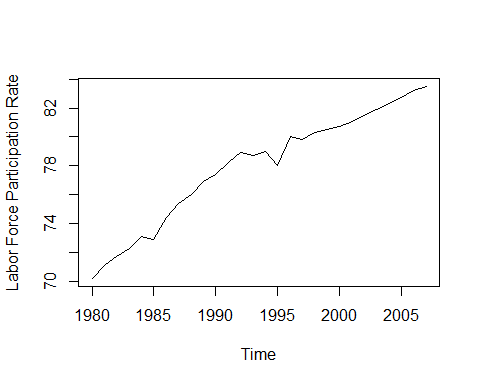
library(xts)  
  
# Import with scan  
# "70.19999695 71.09999847 71.69999695 72.30000305 73.09999847 72.90000153 74.40000153 75.40000153 76 76.90000153 77.40000153 78.19999695 78.90000153 78.69999695 79 78 80 79.80000305 80.30000305 80.5 80.69999695 81.09999847 81.5 81.90000153 82.30000305 82.69999695 83.19999695 83.5"  
# singapur=scan()  
# singapur  
singapur <-c(70.19999695, 71.09999847, 71.69999695, 72.30000305, 73.09999847, 72.90000153, 74.40000153, 75.40000153, 76, 76.90000153, 77.40000153, 78.19999695, 78.90000153, 78.69999695, 79, 78, 80, 79.80000305, 80.30000305, 80.5, 80.69999695, 81.09999847, 81.5, 81.90000153, 82.30000305, 82.69999695, 83.19999695, 83.5)  
# item 28 are shown by this code   
singapur

## [1] 70.2 71.1 71.7 72.3 73.1 72.9 74.4 75.4 76.0 76.9 77.4 78.2 78.9 78.7  
## [15] 79.0 78.0 80.0 79.8 80.3 80.5 80.7 81.1 81.5 81.9 82.3 82.7 83.2 83.5

singapur <- ts(singapur,start=1980)  
singapur

## Time Series:  
## Start = 1980   
## End = 2007   
## Frequency = 1   
## [1] 70.2 71.1 71.7 72.3 73.1 72.9 74.4 75.4 76.0 76.9 77.4 78.2 78.9 78.7  
## [15] 79.0 78.0 80.0 79.8 80.3 80.5 80.7 81.1 81.5 81.9 82.3 82.7 83.2 83.5

plot(singapur,ylab="Labor Force Participation Rate")



From the above chart, the possible models are: - ARIMA - Holt linear trend method

However, one feature of this data is that the values cannnot exceed 100% in the model.There needs to be some press hold. Holt method has a nice damping parameter (holt()).

## Lecture 7: Mission Statement

From this lecture, we would focous on the theory and practical implementation through the four questions. - How to handle time series with trend?  
- Which methods are available?  
- What are the pitfalls?  
- How to visualize time series data?

The process is 1. Get the dataset  
2. Get the mission statement  
3. Work on the project  
4. Proceed with lectures and check the work

The potential models are:  
- Linear trend model with holt()  
- With damping paramter  
- Without damping parameter  
- ARIMA model

These models can be applied for forecasts, but with pitfalls, visualizations, alternatives.

For applying the above models, we just need:  
(1) Libraries “forecast” and “ggplo2” + R base functions  
(2) Resources from the introductory section

## Lecture 9: Exponential smoothing

Forecast package - Simple exponential smooothing: ses() - Holt’s linear trend model]: holt() +damped - Holt-Winters seasonal method: hw() - Automated exponential smooothing: ets()

Holt linear trend model is:

- : estimated forecast value at time point  
- : level value at time point  
- : trend value at time point multiplied by

Should the model react only to recent data or should it consider older data as well? -> Smoothing parameters

Smoothing parameters of a Holt Linear Trend Model are:  
1. :smoothing parameter for the level 2. :smoothing parameter for the trend

The closer the smoothing parameter is to zero, the model becomes smoooth model (older date is consdered too), otherwise (close to 1) the model is the reactive model, heavily relying on recent data.

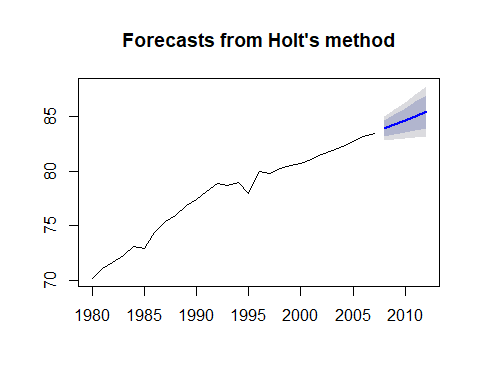
## Lecture 10: The Holt Linear Trend Model

In the Project I, we use holt linear trend model. In practice, we firstly activate the library forecast, and use the holt() function to create the model.  
- data=the time series to model  
- h=forecast length

# holt(data, h=x, damped=FALSE, level=c(80,95),fan=FALSE,initial=c("optimal","simple"),exponential=FALSE,alpha=NULL,beta=NULL,phi=NULL,lambda=NULL)  
  
library(forecast)  
holt\_trend <- holt(singapur,h=5)  
summary(holt\_trend)

##   
## Forecast method: Holt's method  
##   
## Model Information:  
## Holt's method   
##   
## Call:  
## holt(y = singapur, h = 5)   
##   
## Smoothing parameters:  
## alpha = 0.6378   
## beta = 0.1212   
##   
## Initial states:  
## l = 69.619   
## b = 0.6666   
##   
## sigma: 0.5529  
##   
## AIC AICc BIC   
## 65.79969 68.52697 72.46072   
##   
## Error measures:  
## ME RMSE MAE MPE MAPE  
## Training set -0.08247984 0.5118728 0.3084414 -0.1056525 0.3986079  
## MASE ACF1  
## Training set 0.5047223 -0.09190023  
##   
## Forecasts:  
## Point Forecast Lo 80 Hi 80 Lo 95 Hi 95  
## 2008 83.90071 83.19216 84.60926 82.81708 84.98435  
## 2009 84.28751 83.39799 85.17703 82.92711 85.64791  
## 2010 84.67431 83.58796 85.76065 83.01289 86.33573  
## 2011 85.06111 83.76361 86.35860 83.07676 87.04545  
## 2012 85.44790 83.92606 86.96975 83.12045 87.77536

plot(holt\_trend)



Most of the above arguments are the same for ses() and hw(). By summary() function, we can obtain smoothing parameters, and initial state values. In the Singapur case, the participation cannot exceed

If we encounter simiular situation, exact parameters and thresholds. They know the literature, and experience, background information.

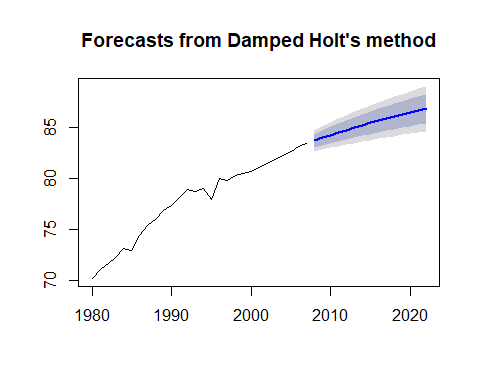
For the above case, we use the damped argument. In the Holt linear trend model, the smoothing parameters are:  
- : smoothing parameter for the level alpha= - : smoothing parameter for the trend beta= - : damping parameter 0<<1 phi=

We can easily adjust the model by the damping parameter.

If is one, its close to the original slope of the Holt trend model. If the is between 0.85 to 0.95, it is the generally recommended range of phi. If the is close to 0, its a flattened curve.

**When the damped parameter is used, the slope of the trend cannnot be constant: It changes over time**

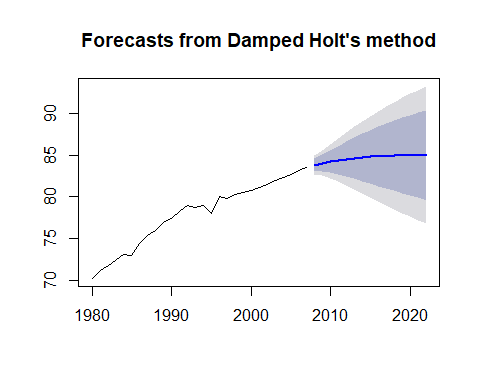
#phi auto generated  
plot(holt(singapur,h=15,damped=T))



# To see the generated value for phi: phi=0.96  
summary(holt(singapur,h=15,damped=T))

##   
## Forecast method: Damped Holt's method  
##   
## Model Information:  
## Damped Holt's method   
##   
## Call:  
## holt(y = singapur, h = 15, damped = T)   
##   
## Smoothing parameters:  
## alpha = 0.5149   
## beta = 1e-04   
## phi = 0.9666   
##   
## Initial states:  
## l = 69.5404   
## b = 0.771   
##   
## sigma: 0.5149  
##   
## AIC AICc BIC   
## 62.62649 66.62649 70.61972   
##   
## Error measures:  
## ME RMSE MAE MPE MAPE MASE  
## Training set 0.01419094 0.4667051 0.3250851 0.01746896 0.419571 0.5319574  
## ACF1  
## Training set -0.001595197  
##   
## Forecasts:  
## Point Forecast Lo 80 Hi 80 Lo 95 Hi 95  
## 2008 83.72693 83.06701 84.38685 82.71766 84.73620  
## 2009 84.00520 83.26291 84.74750 82.86996 85.14044  
## 2010 84.27418 83.45776 85.09060 83.02557 85.52279  
## 2011 84.53417 83.64979 85.41856 83.18163 85.88672  
## 2012 84.78548 83.83798 85.73298 83.33641 86.23456  
## 2013 85.02840 84.02171 86.03509 83.48880 86.56800  
## 2014 85.26320 84.20060 86.32580 83.63809 86.88831  
## 2015 85.49016 84.37443 86.60589 83.78380 87.19653  
## 2016 85.70954 84.54308 86.87600 83.92560 87.49348  
## 2017 85.92159 84.70651 87.13668 84.06328 87.77990  
## 2018 86.12656 84.86471 87.38841 84.19673 88.05639  
## 2019 86.32469 85.01773 87.63164 84.32587 88.32350  
## 2020 86.51619 85.16563 87.86675 84.45068 88.58170  
## 2021 86.70130 85.30848 88.09412 84.57117 88.83143  
## 2022 86.88023 85.44639 88.31407 84.68736 89.07310

# manual setting of phi  
plot(holt(singapur,h=15,damped=T,phi=0.8))



# ARIMA auto generated  
singapurarima=auto.arima(singapur)  
summary(singapurarima)

## Series: singapur   
## ARIMA(1,1,0) with drift   
##   
## Coefficients:  
## ar1 drift  
## -0.3690 0.4904  
## s.e. 0.1763 0.0720  
##   
## sigma^2 estimated as 0.2779: log likelihood=-20.05  
## AIC=46.1 AICc=47.14 BIC=49.99  
##   
## Training set error measures:  
## ME RMSE MAE MPE MAPE  
## Training set 0.006855948 0.4981113 0.3755194 0.01821962 0.4863707  
## MASE ACF1  
## Training set 0.6144862 0.05505323

## Lecture 11: The ARIMA model - Project I

The Box Jenkins models are standard modeling system for time series model. There are three parameters

* AR: Autoregressive such as seasonality, trend P
* |: Integreation - defferencing of the dataset D
* MA: Moving average - movement around a constant mean Q

The ARIMA model is very flexible for explaining: - Random Walk - Exponential Smooothing - Autoregressive models such as AR(1), ARIMA(1,0,0) - Moving average (MA(1), ARIMA(0,0,1))

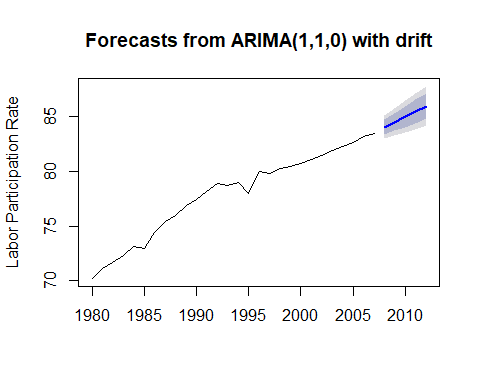
The labour participation rate can be modeled by ARIMA. For that, we need to take into consideration, trending, autocorrelation (AR). Please note that, it seems there is no no moving average, seasonality, thus no differencing is needed.

With the forecast package, we can model the ARIMA model with manual parameter selection, and automatic parameter selection. <forecast package> - Arima() - auto.arima()

# ARIMA auto generated  
singapurarima = auto.arima(singapur)  
summary(singapurarima)

## Series: singapur   
## ARIMA(1,1,0) with drift   
##   
## Coefficients:  
## ar1 drift  
## -0.3690 0.4904  
## s.e. 0.1763 0.0720  
##   
## sigma^2 estimated as 0.2779: log likelihood=-20.05  
## AIC=46.1 AICc=47.14 BIC=49.99  
##   
## Training set error measures:  
## ME RMSE MAE MPE MAPE  
## Training set 0.006855948 0.4981113 0.3755194 0.01821962 0.4863707  
## MASE ACF1  
## Training set 0.6144862 0.05505323

## ARIMA(1,1,0) with drift  
## Summary output is same as the holt model  
## The buttom shows that training set error measures  
  
plot(forecast(singapurarima,h=5),ylab="Labor Participation Rate")



# The above setting might work for short term period  
# Flatting curve needs to be designed  
  
# Exact calculation of Arima parameters  
auto.arima(singapur,stepwise = F,approximation = F)

## Series: singapur   
## ARIMA(1,1,0) with drift   
##   
## Coefficients:  
## ar1 drift  
## -0.3690 0.4904  
## s.e. 0.1763 0.0720  
##   
## sigma^2 estimated as 0.2779: log likelihood=-20.05  
## AIC=46.1 AICc=47.14 BIC=49.99

## Lecture 12: Comparison between ggplot

The visualization should be simple.

The plotting methods with forecast package are: - plot(): quick and simple graphs with R base - autoplot() and autolayer(): Visually improved plots with libraries ggplot2 and forecast.

The latest forecast package is in collaboration with the ggplot package. IN this lecture, we would visualize the Singapur data with the ggplot package.

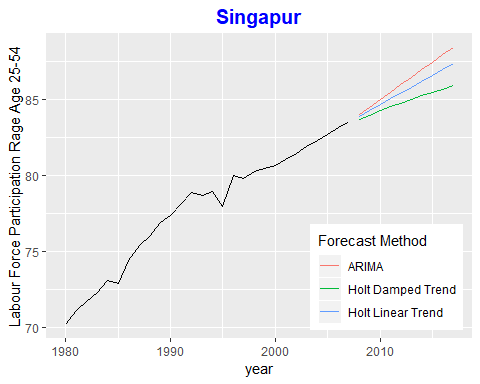
# model setting  
# 1: Holttrend model  
library(forecast)  
holt\_trend = holt(singapur,h=10)  
holtdamped=holt(singapur,h=10,damped=T)  
arimafore=forecast(auto.arima(singapur),h=10)

* Functions: autoplot() + autolayer()
* activate the libraries ggplot and forecast

1. We first return a line graph(time series) by model. In the visualization,
2. Models are added layer by later by using autolayer() from the forecast package.In addition, $mean extracts the forecast values from the model.
3. Axis labels are added with +.
4. The legend is added.

library(ggplot2)  
autoplot(singapur)+  
 forecast::autolayer(holt\_trend$mean,series="Holt Linear Trend")+  
 forecast::autolayer(holtdamped$mean,series="Holt Damped Trend")+  
 forecast::autolayer(arimafore$mean,series="ARIMA")+  
 xlab("year")+ylab("Labour Force Participation Rage Age 25-54")+  
 guides(colour=guide\_legend(title="Forecast Method"))+  
 theme(legend.position = c(0.8,0.2))+  
 ggtitle("Singapur")+  
 theme(plot.title = element\_text(family="Times",hjust=0.5,color="blue",face="bold",size=15))

## Warning in grid.Call(C\_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :  
## Windows のフォントデータベースにフォントファミリが見付かりません  
  
## Warning in grid.Call(C\_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :  
## Windows のフォントデータベースにフォントファミリが見付かりません  
  
## Warning in grid.Call(C\_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :  
## Windows のフォントデータベースにフォントファミリが見付かりません

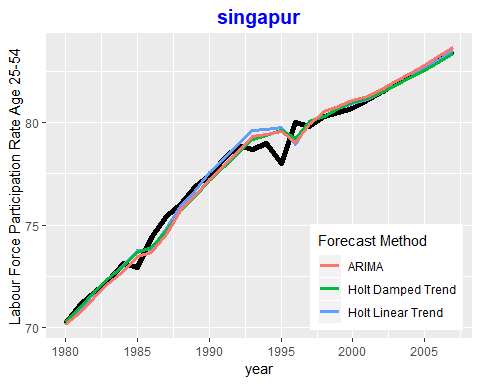


## Lecture 13: In-Sample forecast vs actual data

The idea is to reconstruct the plot, and we compare the original data with in-sample values of the models.In comparision, we will color code the lines. Lastly, we adjust the tiles and labels.

# package load  
library(forecast)  
library(ggplot2)  
  
# models   
holttrend=holt(singapur,h=10)  
holtdamped=holt(singapur,h=10,damped=T)  
airmafore=forecast(auto.arima(singapur),h=10)  
  
autoplot(singapur)+geom\_line(size=2)+  
 forecast::autolayer(holttrend$fitted,series="Holt Linear Trend",size=1.1)+  
 forecast::autolayer(holtdamped$fitted,series="Holt Damped Trend",size=1.1)+  
 forecast::autolayer(arimafore$fitted,series="ARIMA",size=1.1)+  
 xlab("year")+ylab("Labour Force Participation Rate Age 25-54")+  
 guides(colour=guide\_legend(title="Forecast Method"))+  
 theme(legend.position=c(0.8,0.2))+  
 ggtitle("singapur")+theme(plot.title=element\_text(family="Times",hjust=0.5,color="blue",face="bold",size=15))

## Warning in grid.Call(C\_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :  
## Windows のフォントデータベースにフォントファミリが見付かりません  
  
## Warning in grid.Call(C\_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :  
## Windows のフォントデータベースにフォントファミリが見付かりません  
  
## Warning in grid.Call(C\_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :  
## Windows のフォントデータベースにフォントファミリが見付かりません



# Section 3: Project II Seasonal data: monthly inflation rates of Germany

## Lecture 14: Getting familar with data

Inflation rate is a measure of change in purchasing power per unit of money. It is also important to know how inflation moves along with other indicators.

We use a monthly inflation rate of Germany (Jany 2008 - Oct 2017) from ststbureau.org. <https://www.statbureau.org/> <https://www.statbureau.org/en/germany/inflation-tables>

The monthly rate should be more intuitive in the following comparison methods. - Oct 2017 vs Oct 2016: Year on year rate (standard method) - OCt 2017 vs Sept 2017: Month on month rate (changes are easier to detect)

The inspecting the dataset comes from 1. No **trend** is present (constant mean) 2. **Seasonality** is present (seasonal decomposition, seasonal ARIMA, exponential smoothing) 3. Presence of **negative values** (multiplicative exponential smoothing models are excluded) 4. **Stable amplitude** (stable variance)

## Lecture 15: Importing the data

The general strucure of the dataset should be common.

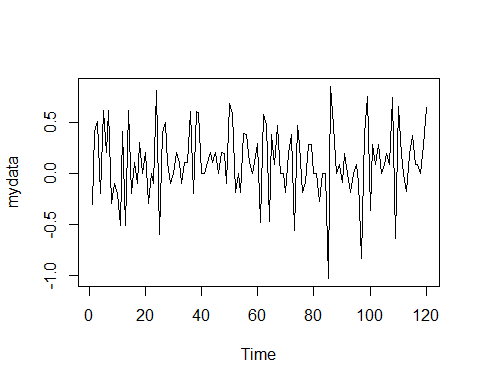
There are two things to keep in mind while importing - No headrs or row IDs - data needs to be arranged/sorted (preformating process)

Let’s use a scan() function to import data.Only with this function, no timestamp is set yet.

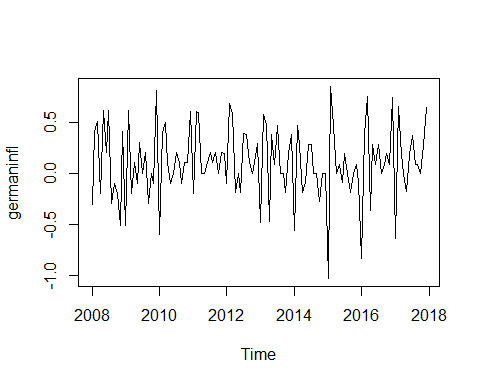
# data load  
# mydata = scan()  
  
mydata <- c(-0.31, 0.41, 0.51, -0.2, 0.61, 0.2, 0.61, -0.3, -0.1, -0.2, -0.51, 0.41, -0.51, 0.61, -0.2, 0.1, -0.1, 0.3, 0, 0.2, -0.3, 0, -0.1, 0.81, -0.6, 0.4, 0.5, 0.1, -0.1, 0, 0.2, 0.1, -0.1, 0.1, 0.1, 0.6, -0.2, 0.6, 0.59, 0, 0, 0.1, 0.2, 0.1, 0.2, 0, 0.2, 0.19, -0.1, 0.68, 0.58, -0.19, 0, -0.19, 0.39, 0.38, 0.1, 0, 0.1, 0.29, -0.48, 0.57, 0.48, -0.47, 0.38, 0.09, 0.47, 0, 0, -0.19, 0.19, 0.38, -0.56, 0.47, 0.28, -0.19, -0.09, 0.28, 0.28, 0, 0, -0.28, 0, 0, -1.03, 0.85, 0.47, 0, 0.09, -0.09, 0.19, 0, -0.19, 0, 0.09, -0.09, -0.84, 0.38, 0.75, -0.37, 0.28, 0.09, 0.28, 0, 0.09, 0.19, 0.09, 0.74, -0.64, 0.65, 0.18, 0, -0.18, 0.18, 0.37, 0.09, 0.09, 0, 0.27, 0.64)  
mydata

## [1] -0.31 0.41 0.51 -0.20 0.61 0.20 0.61 -0.30 -0.10 -0.20 -0.51  
## [12] 0.41 -0.51 0.61 -0.20 0.10 -0.10 0.30 0.00 0.20 -0.30 0.00  
## [23] -0.10 0.81 -0.60 0.40 0.50 0.10 -0.10 0.00 0.20 0.10 -0.10  
## [34] 0.10 0.10 0.60 -0.20 0.60 0.59 0.00 0.00 0.10 0.20 0.10  
## [45] 0.20 0.00 0.20 0.19 -0.10 0.68 0.58 -0.19 0.00 -0.19 0.39  
## [56] 0.38 0.10 0.00 0.10 0.29 -0.48 0.57 0.48 -0.47 0.38 0.09  
## [67] 0.47 0.00 0.00 -0.19 0.19 0.38 -0.56 0.47 0.28 -0.19 -0.09  
## [78] 0.28 0.28 0.00 0.00 -0.28 0.00 0.00 -1.03 0.85 0.47 0.00  
## [89] 0.09 -0.09 0.19 0.00 -0.19 0.00 0.09 -0.09 -0.84 0.38 0.75  
## [100] -0.37 0.28 0.09 0.28 0.00 0.09 0.19 0.09 0.74 -0.64 0.65  
## [111] 0.18 0.00 -0.18 0.18 0.37 0.09 0.09 0.00 0.27 0.64

# raw data  
# -0.31 0.41 0.51 -0.2 0.61 0.2 0.61 -0.3 -0.1 -0.2 -0.51 0.41 -0.51 0.61 -0.2 0.1 -0.1 0.3 0 0.2 -0.3 0 -0.1 0.81 -0.6 0.4 0.5 0.1 -0.1 0 0.2 0.1 -0.1 0.1 0.1 0.6 -0.2 0.6 0.59 0 0 0.1 0.2 0.1 0.2 0 0.2 0.19 -0.1 0.68 0.58 -0.19 0 -0.19 0.39 0.38 0.1 0 0.1 0.29 -0.48 0.57 0.48 -0.47 0.38 0.09 0.47 0 0 -0.19 0.19 0.38 -0.56 0.47 0.28 -0.19 -0.09 0.28 0.28 0 0 -0.28 0 0 -1.03 0.85 0.47 0 0.09 -0.09 0.19 0 -0.19 0 0.09 -0.09 -0.84 0.38 0.75 -0.37 0.28 0.09 0.28 0 0.09 0.19 0.09 0.74 -0.64 0.65 0.18 0 -0.18 0.18 0.37 0.09 0.09 0 0.27 0.64  
  
# timestamp  
# mydata  
plot.ts(mydata)



# ts() function - ts data load  
germaninfl=ts(mydata,start=2008,frequency = 12)  
plot(germaninfl)



# basic structure  
ts(mydata, start=c(2008,3),frequency=12)

## Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov  
## 2008 -0.31 0.41 0.51 -0.20 0.61 0.20 0.61 -0.30 -0.10  
## 2009 -0.51 0.41 -0.51 0.61 -0.20 0.10 -0.10 0.30 0.00 0.20 -0.30  
## 2010 -0.10 0.81 -0.60 0.40 0.50 0.10 -0.10 0.00 0.20 0.10 -0.10  
## 2011 0.10 0.60 -0.20 0.60 0.59 0.00 0.00 0.10 0.20 0.10 0.20  
## 2012 0.20 0.19 -0.10 0.68 0.58 -0.19 0.00 -0.19 0.39 0.38 0.10  
## 2013 0.10 0.29 -0.48 0.57 0.48 -0.47 0.38 0.09 0.47 0.00 0.00  
## 2014 0.19 0.38 -0.56 0.47 0.28 -0.19 -0.09 0.28 0.28 0.00 0.00  
## 2015 0.00 0.00 -1.03 0.85 0.47 0.00 0.09 -0.09 0.19 0.00 -0.19  
## 2016 0.09 -0.09 -0.84 0.38 0.75 -0.37 0.28 0.09 0.28 0.00 0.09  
## 2017 0.09 0.74 -0.64 0.65 0.18 0.00 -0.18 0.18 0.37 0.09 0.09  
## 2018 0.27 0.64   
## Dec  
## 2008 -0.20  
## 2009 0.00  
## 2010 0.10  
## 2011 0.00  
## 2012 0.00  
## 2013 -0.19  
## 2014 -0.28  
## 2015 0.00  
## 2016 0.19  
## 2017 0.00  
## 2018

## Lecture 16: Mission statement

Seaonal time series data, we are focusing on (1) seasonality, and (2) cross-validation.Primarily, we can inport data by:ts(data, start=, frequency=)

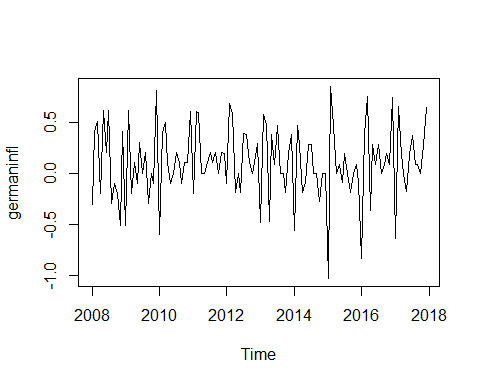
This dataset is a monthly inflation rate of Germany, class=ts, frequency=!2, no trend with seasonal pattern. For dealing with seasonality, we should consider (1) seasonal decomposition, (2) exponential smooothing and (3) seasonal ARIMA.

1. **Seasonal decomposition** decompose() function is not allowed for the forecast. STL + ETL enables us to decompose for the forecast, which can be resembled by stlf().
2. **Sesonal ARIMA** ARIMA(1,0,2)(0,1,1) or auto.arima() function
3. **Exponential smoothing** get an automated exponential smoothing model by est() - automated or hw().

Do these models have high accuracy? To answer this question, we should splot test/training set, and corss validate by tsCV().

## Lecture 17: Project II Script

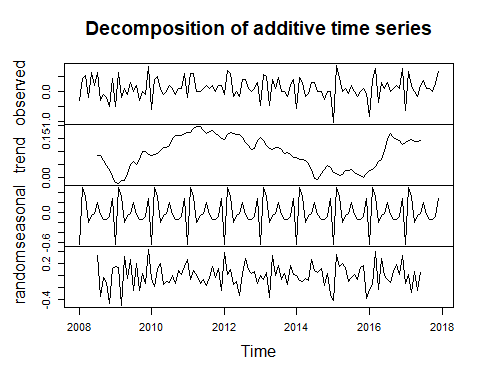
# Data load  
# mydata=scan()  
# plot.ts(mydata)  
  
# germaninfl=ts(mydata,start=2008,frequency = 12)  
# plot(germaninfl)  
  
# Seasonal decomposition  
plot(germaninfl)



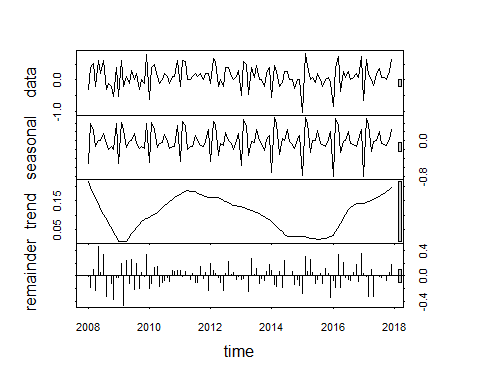
decompose(germaninfl)

## $x  
## Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov  
## 2008 -0.31 0.41 0.51 -0.20 0.61 0.20 0.61 -0.30 -0.10 -0.20 -0.51  
## 2009 -0.51 0.61 -0.20 0.10 -0.10 0.30 0.00 0.20 -0.30 0.00 -0.10  
## 2010 -0.60 0.40 0.50 0.10 -0.10 0.00 0.20 0.10 -0.10 0.10 0.10  
## 2011 -0.20 0.60 0.59 0.00 0.00 0.10 0.20 0.10 0.20 0.00 0.20  
## 2012 -0.10 0.68 0.58 -0.19 0.00 -0.19 0.39 0.38 0.10 0.00 0.10  
## 2013 -0.48 0.57 0.48 -0.47 0.38 0.09 0.47 0.00 0.00 -0.19 0.19  
## 2014 -0.56 0.47 0.28 -0.19 -0.09 0.28 0.28 0.00 0.00 -0.28 0.00  
## 2015 -1.03 0.85 0.47 0.00 0.09 -0.09 0.19 0.00 -0.19 0.00 0.09  
## 2016 -0.84 0.38 0.75 -0.37 0.28 0.09 0.28 0.00 0.09 0.19 0.09  
## 2017 -0.64 0.65 0.18 0.00 -0.18 0.18 0.37 0.09 0.09 0.00 0.27  
## Dec  
## 2008 0.41  
## 2009 0.81  
## 2010 0.60  
## 2011 0.19  
## 2012 0.29  
## 2013 0.38  
## 2014 0.00  
## 2015 -0.09  
## 2016 0.74  
## 2017 0.64  
##   
## $seasonal  
## Jan Feb Mar Apr May  
## 2008 -0.63583333 0.49347222 0.31523148 -0.20324074 -0.06333333  
## 2009 -0.63583333 0.49347222 0.31523148 -0.20324074 -0.06333333  
## 2010 -0.63583333 0.49347222 0.31523148 -0.20324074 -0.06333333  
## 2011 -0.63583333 0.49347222 0.31523148 -0.20324074 -0.06333333  
## 2012 -0.63583333 0.49347222 0.31523148 -0.20324074 -0.06333333  
## 2013 -0.63583333 0.49347222 0.31523148 -0.20324074 -0.06333333  
## 2014 -0.63583333 0.49347222 0.31523148 -0.20324074 -0.06333333  
## 2015 -0.63583333 0.49347222 0.31523148 -0.20324074 -0.06333333  
## 2016 -0.63583333 0.49347222 0.31523148 -0.20324074 -0.06333333  
## 2017 -0.63583333 0.49347222 0.31523148 -0.20324074 -0.06333333  
## Jun Jul Aug Sep Oct  
## 2008 -0.01467593 0.19726852 -0.04009259 -0.12634259 -0.13462963  
## 2009 -0.01467593 0.19726852 -0.04009259 -0.12634259 -0.13462963  
## 2010 -0.01467593 0.19726852 -0.04009259 -0.12634259 -0.13462963  
## 2011 -0.01467593 0.19726852 -0.04009259 -0.12634259 -0.13462963  
## 2012 -0.01467593 0.19726852 -0.04009259 -0.12634259 -0.13462963  
## 2013 -0.01467593 0.19726852 -0.04009259 -0.12634259 -0.13462963  
## 2014 -0.01467593 0.19726852 -0.04009259 -0.12634259 -0.13462963  
## 2015 -0.01467593 0.19726852 -0.04009259 -0.12634259 -0.13462963  
## 2016 -0.01467593 0.19726852 -0.04009259 -0.12634259 -0.13462963  
## 2017 -0.01467593 0.19726852 -0.04009259 -0.12634259 -0.13462963  
## Nov Dec  
## 2008 -0.07189815 0.28407407  
## 2009 -0.07189815 0.28407407  
## 2010 -0.07189815 0.28407407  
## 2011 -0.07189815 0.28407407  
## 2012 -0.07189815 0.28407407  
## 2013 -0.07189815 0.28407407  
## 2014 -0.07189815 0.28407407  
## 2015 -0.07189815 0.28407407  
## 2016 -0.07189815 0.28407407  
## 2017 -0.07189815 0.28407407  
##   
## $trend  
## Jan Feb Mar Apr May  
## 2008 NA NA NA NA NA  
## 2009 -0.016250000 -0.020833333 -0.008333333 -0.008333333 0.017083333  
## 2010 0.084166667 0.088333333 0.092500000 0.105000000 0.117500000  
## 2011 0.174166667 0.174166667 0.186666667 0.195000000 0.195000000  
## 2012 0.147083333 0.166666667 0.174166667 0.170000000 0.165833333  
## 2013 0.155833333 0.143333333 0.123333333 0.111250000 0.107083333  
## 2014 0.078750000 0.070833333 0.070833333 0.067083333 0.055416667  
## 2015 0.020416667 0.016666667 0.008750000 0.012500000 0.027916667  
## 2016 0.027916667 0.031666667 0.043333333 0.062916667 0.070833333  
## 2017 0.135416667 0.142916667 0.146666667 0.138750000 0.138333333  
## Jun Jul Aug Sep Oct  
## 2008 NA 0.085833333 0.085833333 0.064583333 0.047500000  
## 2009 0.050833333 0.063750000 0.051250000 0.071666667 0.100833333  
## 2010 0.117083333 0.125000000 0.150000000 0.162083333 0.161666667  
## 2011 0.182083333 0.169166667 0.176666667 0.179583333 0.171250000  
## 2012 0.165833333 0.154166667 0.133750000 0.125000000 0.109166667  
## 2013 0.114583333 0.115000000 0.107500000 0.095000000 0.098333333  
## 2014 0.031666667 -0.003750000 -0.007500000 0.016250000 0.032083333  
## 2015 0.027916667 0.032083333 0.020416667 0.012500000 0.008750000  
## 2016 0.105416667 0.148333333 0.167916667 0.155416667 0.147083333  
## 2017 0.141666667 NA NA NA NA  
## Nov Dec  
## 2008 0.030416667 0.005000000  
## 2009 0.100833333 0.088333333  
## 2010 0.161666667 0.170000000  
## 2011 0.163333333 0.151250000  
## 2012 0.113333333 0.140833333  
## 2013 0.090416667 0.078750000  
## 2014 0.047500000 0.039583333  
## 2015 0.001250000 0.016666667  
## 2016 0.143333333 0.127916667  
## 2017 NA NA  
##   
## $random  
## Jan Feb Mar Apr May  
## 2008 NA NA NA NA NA  
## 2009 1.420833e-01 1.373611e-01 -5.068981e-01 3.115741e-01 -5.375000e-02  
## 2010 -4.833333e-02 -1.818056e-01 9.226852e-02 1.982407e-01 -1.541667e-01  
## 2011 2.616667e-01 -6.763889e-02 8.810185e-02 8.240741e-03 -1.316667e-01  
## 2012 3.887500e-01 1.986111e-02 9.060185e-02 -1.567593e-01 -1.025000e-01  
## 2013 5.551115e-17 -6.680556e-02 4.143519e-02 -3.780093e-01 3.362500e-01  
## 2014 -2.916667e-03 -9.430556e-02 -1.060648e-01 -5.384259e-02 -8.208333e-02  
## 2015 -4.145833e-01 3.398611e-01 1.460185e-01 1.907407e-01 1.254167e-01  
## 2016 -2.320833e-01 -1.451389e-01 3.914352e-01 -2.296759e-01 2.725000e-01  
## 2017 -1.395833e-01 1.361111e-02 -2.818981e-01 6.449074e-02 -2.550000e-01  
## Jun Jul Aug Sep Oct  
## 2008 NA 3.268981e-01 -3.457407e-01 -3.824074e-02 -1.128704e-01  
## 2009 2.638426e-01 -2.610185e-01 1.888426e-01 -2.453241e-01 3.379630e-02  
## 2010 -1.024074e-01 -1.222685e-01 -9.907407e-03 -1.357407e-01 7.296296e-02  
## 2011 -6.740741e-02 -1.664352e-01 -3.657407e-02 1.467593e-01 -3.662037e-02  
## 2012 -3.411574e-01 3.856481e-02 2.863426e-01 1.013426e-01 2.546296e-02  
## 2013 -9.907407e-03 1.577315e-01 -6.740741e-02 3.134259e-02 -1.537037e-01  
## 2014 2.630093e-01 8.648148e-02 4.759259e-02 1.100926e-01 -1.774537e-01  
## 2015 -1.032407e-01 -3.935185e-02 1.967593e-02 -7.615741e-02 1.258796e-01  
## 2016 -7.407407e-04 -6.560185e-02 -1.278241e-01 6.092593e-02 1.775463e-01  
## 2017 5.300926e-02 NA NA NA NA  
## Nov Dec  
## 2008 -4.685185e-01 1.209259e-01  
## 2009 -1.289352e-01 4.375926e-01  
## 2010 1.023148e-02 1.459259e-01  
## 2011 1.085648e-01 -2.453241e-01  
## 2012 5.856481e-02 -1.349074e-01  
## 2013 1.714815e-01 1.717593e-02  
## 2014 2.439815e-02 -3.236574e-01  
## 2015 1.606481e-01 -3.907407e-01  
## 2016 1.856481e-02 3.280093e-01  
## 2017 NA NA  
##   
## $figure  
## [1] -0.63583333 0.49347222 0.31523148 -0.20324074 -0.06333333  
## [6] -0.01467593 0.19726852 -0.04009259 -0.12634259 -0.13462963  
## [11] -0.07189815 0.28407407  
##   
## $type  
## [1] "additive"  
##   
## attr(,"class")  
## [1] "decomposed.ts"

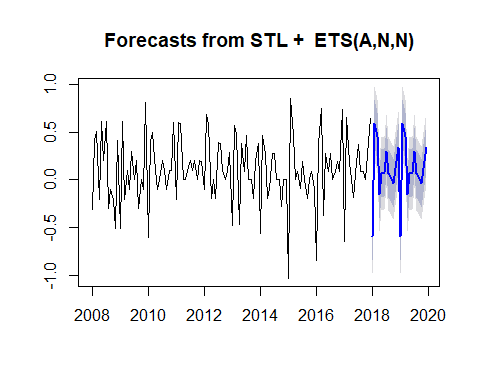
plot(decompose(germaninfl))



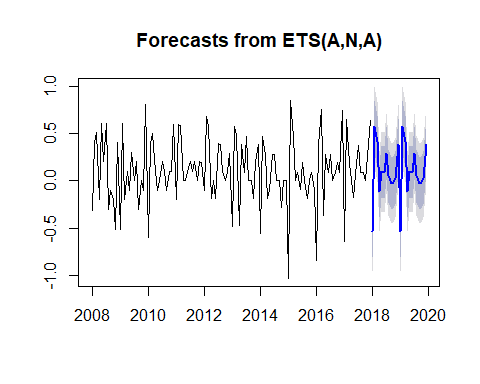
# using the stl method  
plot(stl(germaninfl,s.window = 7))



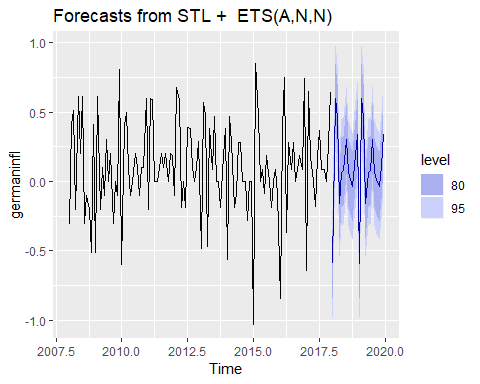
# stl forecasting  
library(forecast)  
plot(stlf(germaninfl,method="ets"))



# comparison with a standard est forecast  
plot(forecast(ets(germaninfl),h=24))



# using autoplot  
library(ggplot2)  
autoplot(stlf(germaninfl,method="ets"))



# Seasonal Arima(package forecast)  
auto.arima(germaninfl,stepwise=T,  
 approximation=F,trace=T)

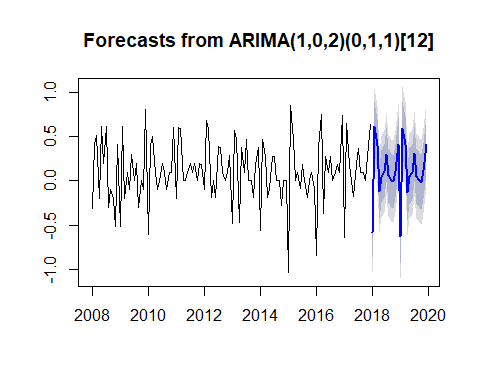
##   
## ARIMA(2,0,2)(1,1,1)[12] with drift : Inf  
## ARIMA(0,0,0)(0,1,0)[12] with drift : 31.04128  
## ARIMA(1,0,0)(1,1,0)[12] with drift : 21.57402  
## ARIMA(0,0,1)(0,1,1)[12] with drift : 7.519111  
## ARIMA(0,0,0)(0,1,0)[12] : 28.99805  
## ARIMA(0,0,1)(1,1,1)[12] with drift : Inf  
## ARIMA(0,0,1)(0,1,0)[12] with drift : 28.07929  
## ARIMA(0,0,1)(0,1,2)[12] with drift : Inf  
## ARIMA(0,0,1)(1,1,2)[12] with drift : Inf  
## ARIMA(1,0,1)(0,1,1)[12] with drift : 2.573287  
## ARIMA(1,0,0)(0,1,1)[12] with drift : 7.188973  
## ARIMA(1,0,2)(0,1,1)[12] with drift : 0.5156938  
## ARIMA(2,0,3)(0,1,1)[12] with drift : 3.491814  
## ARIMA(1,0,2)(0,1,1)[12] : -1.710145  
## ARIMA(1,0,2)(1,1,1)[12] : Inf  
## ARIMA(1,0,2)(0,1,0)[12] : 18.96391  
## ARIMA(1,0,2)(0,1,2)[12] : Inf  
## ARIMA(1,0,2)(1,1,2)[12] : Inf  
## ARIMA(0,0,2)(0,1,1)[12] : 2.553462  
## ARIMA(2,0,2)(0,1,1)[12] : -0.001234885  
## ARIMA(1,0,1)(0,1,1)[12] : 0.3814517  
## ARIMA(1,0,3)(0,1,1)[12] : 0.2762055  
## ARIMA(0,0,1)(0,1,1)[12] : 5.362176  
## ARIMA(2,0,3)(0,1,1)[12] : Inf  
##   
## Best model: ARIMA(1,0,2)(0,1,1)[12]

## Series: germaninfl   
## ARIMA(1,0,2)(0,1,1)[12]   
##   
## Coefficients:  
## ar1 ma1 ma2 sma1  
## -0.7883 0.7832 0.2201 -0.7581  
## s.e. 0.1238 0.1494 0.1005 0.1369  
##   
## sigma^2 estimated as 0.04927: log likelihood=6.15  
## AIC=-2.3 AICc=-1.71 BIC=11.11

# Getting an object  
germaninflarima = auto.arima(germaninfl,stepwise=T,  
 approximation = F,trace=T)

##   
## ARIMA(2,0,2)(1,1,1)[12] with drift : Inf  
## ARIMA(0,0,0)(0,1,0)[12] with drift : 31.04128  
## ARIMA(1,0,0)(1,1,0)[12] with drift : 21.57402  
## ARIMA(0,0,1)(0,1,1)[12] with drift : 7.519111  
## ARIMA(0,0,0)(0,1,0)[12] : 28.99805  
## ARIMA(0,0,1)(1,1,1)[12] with drift : Inf  
## ARIMA(0,0,1)(0,1,0)[12] with drift : 28.07929  
## ARIMA(0,0,1)(0,1,2)[12] with drift : Inf  
## ARIMA(0,0,1)(1,1,2)[12] with drift : Inf  
## ARIMA(1,0,1)(0,1,1)[12] with drift : 2.573287  
## ARIMA(1,0,0)(0,1,1)[12] with drift : 7.188973  
## ARIMA(1,0,2)(0,1,1)[12] with drift : 0.5156938  
## ARIMA(2,0,3)(0,1,1)[12] with drift : 3.491814  
## ARIMA(1,0,2)(0,1,1)[12] : -1.710145  
## ARIMA(1,0,2)(1,1,1)[12] : Inf  
## ARIMA(1,0,2)(0,1,0)[12] : 18.96391  
## ARIMA(1,0,2)(0,1,2)[12] : Inf  
## ARIMA(1,0,2)(1,1,2)[12] : Inf  
## ARIMA(0,0,2)(0,1,1)[12] : 2.553462  
## ARIMA(2,0,2)(0,1,1)[12] : -0.001234885  
## ARIMA(1,0,1)(0,1,1)[12] : 0.3814517  
## ARIMA(1,0,3)(0,1,1)[12] : 0.2762055  
## ARIMA(0,0,1)(0,1,1)[12] : 5.362176  
## ARIMA(2,0,3)(0,1,1)[12] : Inf  
##   
## Best model: ARIMA(1,0,2)(0,1,1)[12]

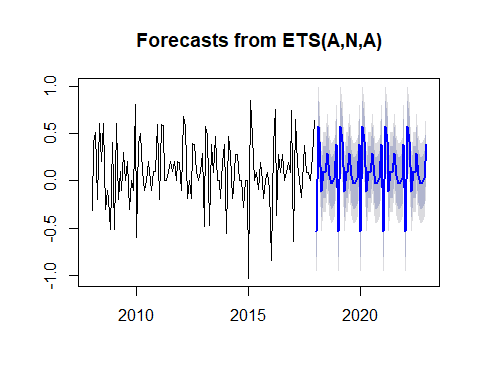
# forecast  
forec=forecast(germaninflarima)  
plot(forec)



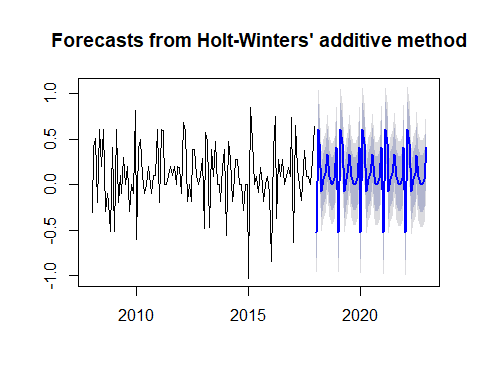
# exponential smoothing with ets  
# auto generated  
ets(germaninfl)

## ETS(A,N,A)   
##   
## Call:  
## ets(y = germaninfl)   
##   
## Smoothing parameters:  
## alpha = 1e-04   
## gamma = 1e-04   
##   
## Initial states:  
## l = 0.104   
## s = 0.2832 -0.0736 -0.1241 -0.1295 -0.0511 0.19  
## -0.0164 -0.0093 -0.2138 0.3141 0.4727 -0.6423  
##   
## sigma: 0.2153  
##   
## AIC AICc BIC   
## 220.9850 225.6004 262.7974

# forecast plot  
germaninflets=ets(germaninfl)  
plot(forecast(germaninflets,h=60))



# comparison with seasonal Holt Winters model  
plot(hw(germaninfl,h=60))



# cross validation of 2 models  
germnainflets=ets(germaninfl)  
germaninflarima=auto.arima(germaninfl,  
 stepwise=T,  
 approximation=F,  
 trace=T)

##   
## ARIMA(2,0,2)(1,1,1)[12] with drift : Inf  
## ARIMA(0,0,0)(0,1,0)[12] with drift : 31.04128  
## ARIMA(1,0,0)(1,1,0)[12] with drift : 21.57402  
## ARIMA(0,0,1)(0,1,1)[12] with drift : 7.519111  
## ARIMA(0,0,0)(0,1,0)[12] : 28.99805  
## ARIMA(0,0,1)(1,1,1)[12] with drift : Inf  
## ARIMA(0,0,1)(0,1,0)[12] with drift : 28.07929  
## ARIMA(0,0,1)(0,1,2)[12] with drift : Inf  
## ARIMA(0,0,1)(1,1,2)[12] with drift : Inf  
## ARIMA(1,0,1)(0,1,1)[12] with drift : 2.573287  
## ARIMA(1,0,0)(0,1,1)[12] with drift : 7.188973  
## ARIMA(1,0,2)(0,1,1)[12] with drift : 0.5156938  
## ARIMA(2,0,3)(0,1,1)[12] with drift : 3.491814  
## ARIMA(1,0,2)(0,1,1)[12] : -1.710145  
## ARIMA(1,0,2)(1,1,1)[12] : Inf  
## ARIMA(1,0,2)(0,1,0)[12] : 18.96391  
## ARIMA(1,0,2)(0,1,2)[12] : Inf  
## ARIMA(1,0,2)(1,1,2)[12] : Inf  
## ARIMA(0,0,2)(0,1,1)[12] : 2.553462  
## ARIMA(2,0,2)(0,1,1)[12] : -0.001234885  
## ARIMA(1,0,1)(0,1,1)[12] : 0.3814517  
## ARIMA(1,0,3)(0,1,1)[12] : 0.2762055  
## ARIMA(0,0,1)(0,1,1)[12] : 5.362176  
## ARIMA(2,0,3)(0,1,1)[12] : Inf  
##   
## Best model: ARIMA(1,0,2)(0,1,1)[12]

## Function definition  
forecastets = function(x,h){  
 forecast(ets(x),h=h)  
}  
  
forecastarima = function(x,h){  
 forecast(auto.arima(x),stepwise=T,approximation=F,h=h)  
}  
  
# esterror=tsCV(germaninfl,forecastets,h=1)  
# arimaerror=tsCV(germaninfl,forecastarima,h=1)  
  
# mean(esterror^2,na.rm=T)  
# mean(arimaerror^2,na.rm=T)

## Lesson 18: Seasonal Decomposition

Date:7/17 Seasonal decomposition is an old analytical method, but useful for dividing the data into its components: - trend - seasonality - remainder/white noise

The seasonal decomposition is an additive/multiplicative model, and we can get q quick idea about the data.

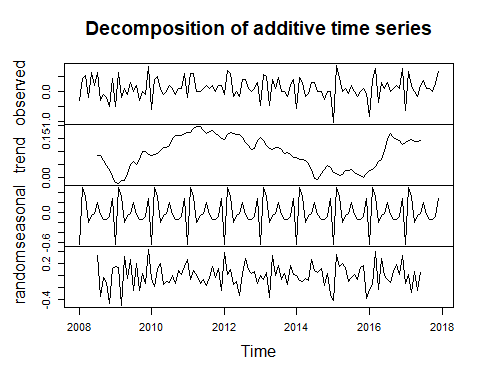
Note that there are several drawbacks: (1) First few observations are NA (2) Slow to catch fast rises (3) Adopts a constant seasonal component

The basic functions are: decompose() and stl().

decompose(germaninfl)

## $x  
## Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov  
## 2008 -0.31 0.41 0.51 -0.20 0.61 0.20 0.61 -0.30 -0.10 -0.20 -0.51  
## 2009 -0.51 0.61 -0.20 0.10 -0.10 0.30 0.00 0.20 -0.30 0.00 -0.10  
## 2010 -0.60 0.40 0.50 0.10 -0.10 0.00 0.20 0.10 -0.10 0.10 0.10  
## 2011 -0.20 0.60 0.59 0.00 0.00 0.10 0.20 0.10 0.20 0.00 0.20  
## 2012 -0.10 0.68 0.58 -0.19 0.00 -0.19 0.39 0.38 0.10 0.00 0.10  
## 2013 -0.48 0.57 0.48 -0.47 0.38 0.09 0.47 0.00 0.00 -0.19 0.19  
## 2014 -0.56 0.47 0.28 -0.19 -0.09 0.28 0.28 0.00 0.00 -0.28 0.00  
## 2015 -1.03 0.85 0.47 0.00 0.09 -0.09 0.19 0.00 -0.19 0.00 0.09  
## 2016 -0.84 0.38 0.75 -0.37 0.28 0.09 0.28 0.00 0.09 0.19 0.09  
## 2017 -0.64 0.65 0.18 0.00 -0.18 0.18 0.37 0.09 0.09 0.00 0.27  
## Dec  
## 2008 0.41  
## 2009 0.81  
## 2010 0.60  
## 2011 0.19  
## 2012 0.29  
## 2013 0.38  
## 2014 0.00  
## 2015 -0.09  
## 2016 0.74  
## 2017 0.64  
##   
## $seasonal  
## Jan Feb Mar Apr May  
## 2008 -0.63583333 0.49347222 0.31523148 -0.20324074 -0.06333333  
## 2009 -0.63583333 0.49347222 0.31523148 -0.20324074 -0.06333333  
## 2010 -0.63583333 0.49347222 0.31523148 -0.20324074 -0.06333333  
## 2011 -0.63583333 0.49347222 0.31523148 -0.20324074 -0.06333333  
## 2012 -0.63583333 0.49347222 0.31523148 -0.20324074 -0.06333333  
## 2013 -0.63583333 0.49347222 0.31523148 -0.20324074 -0.06333333  
## 2014 -0.63583333 0.49347222 0.31523148 -0.20324074 -0.06333333  
## 2015 -0.63583333 0.49347222 0.31523148 -0.20324074 -0.06333333  
## 2016 -0.63583333 0.49347222 0.31523148 -0.20324074 -0.06333333  
## 2017 -0.63583333 0.49347222 0.31523148 -0.20324074 -0.06333333  
## Jun Jul Aug Sep Oct  
## 2008 -0.01467593 0.19726852 -0.04009259 -0.12634259 -0.13462963  
## 2009 -0.01467593 0.19726852 -0.04009259 -0.12634259 -0.13462963  
## 2010 -0.01467593 0.19726852 -0.04009259 -0.12634259 -0.13462963  
## 2011 -0.01467593 0.19726852 -0.04009259 -0.12634259 -0.13462963  
## 2012 -0.01467593 0.19726852 -0.04009259 -0.12634259 -0.13462963  
## 2013 -0.01467593 0.19726852 -0.04009259 -0.12634259 -0.13462963  
## 2014 -0.01467593 0.19726852 -0.04009259 -0.12634259 -0.13462963  
## 2015 -0.01467593 0.19726852 -0.04009259 -0.12634259 -0.13462963  
## 2016 -0.01467593 0.19726852 -0.04009259 -0.12634259 -0.13462963  
## 2017 -0.01467593 0.19726852 -0.04009259 -0.12634259 -0.13462963  
## Nov Dec  
## 2008 -0.07189815 0.28407407  
## 2009 -0.07189815 0.28407407  
## 2010 -0.07189815 0.28407407  
## 2011 -0.07189815 0.28407407  
## 2012 -0.07189815 0.28407407  
## 2013 -0.07189815 0.28407407  
## 2014 -0.07189815 0.28407407  
## 2015 -0.07189815 0.28407407  
## 2016 -0.07189815 0.28407407  
## 2017 -0.07189815 0.28407407  
##   
## $trend  
## Jan Feb Mar Apr May  
## 2008 NA NA NA NA NA  
## 2009 -0.016250000 -0.020833333 -0.008333333 -0.008333333 0.017083333  
## 2010 0.084166667 0.088333333 0.092500000 0.105000000 0.117500000  
## 2011 0.174166667 0.174166667 0.186666667 0.195000000 0.195000000  
## 2012 0.147083333 0.166666667 0.174166667 0.170000000 0.165833333  
## 2013 0.155833333 0.143333333 0.123333333 0.111250000 0.107083333  
## 2014 0.078750000 0.070833333 0.070833333 0.067083333 0.055416667  
## 2015 0.020416667 0.016666667 0.008750000 0.012500000 0.027916667  
## 2016 0.027916667 0.031666667 0.043333333 0.062916667 0.070833333  
## 2017 0.135416667 0.142916667 0.146666667 0.138750000 0.138333333  
## Jun Jul Aug Sep Oct  
## 2008 NA 0.085833333 0.085833333 0.064583333 0.047500000  
## 2009 0.050833333 0.063750000 0.051250000 0.071666667 0.100833333  
## 2010 0.117083333 0.125000000 0.150000000 0.162083333 0.161666667  
## 2011 0.182083333 0.169166667 0.176666667 0.179583333 0.171250000  
## 2012 0.165833333 0.154166667 0.133750000 0.125000000 0.109166667  
## 2013 0.114583333 0.115000000 0.107500000 0.095000000 0.098333333  
## 2014 0.031666667 -0.003750000 -0.007500000 0.016250000 0.032083333  
## 2015 0.027916667 0.032083333 0.020416667 0.012500000 0.008750000  
## 2016 0.105416667 0.148333333 0.167916667 0.155416667 0.147083333  
## 2017 0.141666667 NA NA NA NA  
## Nov Dec  
## 2008 0.030416667 0.005000000  
## 2009 0.100833333 0.088333333  
## 2010 0.161666667 0.170000000  
## 2011 0.163333333 0.151250000  
## 2012 0.113333333 0.140833333  
## 2013 0.090416667 0.078750000  
## 2014 0.047500000 0.039583333  
## 2015 0.001250000 0.016666667  
## 2016 0.143333333 0.127916667  
## 2017 NA NA  
##   
## $random  
## Jan Feb Mar Apr May  
## 2008 NA NA NA NA NA  
## 2009 1.420833e-01 1.373611e-01 -5.068981e-01 3.115741e-01 -5.375000e-02  
## 2010 -4.833333e-02 -1.818056e-01 9.226852e-02 1.982407e-01 -1.541667e-01  
## 2011 2.616667e-01 -6.763889e-02 8.810185e-02 8.240741e-03 -1.316667e-01  
## 2012 3.887500e-01 1.986111e-02 9.060185e-02 -1.567593e-01 -1.025000e-01  
## 2013 5.551115e-17 -6.680556e-02 4.143519e-02 -3.780093e-01 3.362500e-01  
## 2014 -2.916667e-03 -9.430556e-02 -1.060648e-01 -5.384259e-02 -8.208333e-02  
## 2015 -4.145833e-01 3.398611e-01 1.460185e-01 1.907407e-01 1.254167e-01  
## 2016 -2.320833e-01 -1.451389e-01 3.914352e-01 -2.296759e-01 2.725000e-01  
## 2017 -1.395833e-01 1.361111e-02 -2.818981e-01 6.449074e-02 -2.550000e-01  
## Jun Jul Aug Sep Oct  
## 2008 NA 3.268981e-01 -3.457407e-01 -3.824074e-02 -1.128704e-01  
## 2009 2.638426e-01 -2.610185e-01 1.888426e-01 -2.453241e-01 3.379630e-02  
## 2010 -1.024074e-01 -1.222685e-01 -9.907407e-03 -1.357407e-01 7.296296e-02  
## 2011 -6.740741e-02 -1.664352e-01 -3.657407e-02 1.467593e-01 -3.662037e-02  
## 2012 -3.411574e-01 3.856481e-02 2.863426e-01 1.013426e-01 2.546296e-02  
## 2013 -9.907407e-03 1.577315e-01 -6.740741e-02 3.134259e-02 -1.537037e-01  
## 2014 2.630093e-01 8.648148e-02 4.759259e-02 1.100926e-01 -1.774537e-01  
## 2015 -1.032407e-01 -3.935185e-02 1.967593e-02 -7.615741e-02 1.258796e-01  
## 2016 -7.407407e-04 -6.560185e-02 -1.278241e-01 6.092593e-02 1.775463e-01  
## 2017 5.300926e-02 NA NA NA NA  
## Nov Dec  
## 2008 -4.685185e-01 1.209259e-01  
## 2009 -1.289352e-01 4.375926e-01  
## 2010 1.023148e-02 1.459259e-01  
## 2011 1.085648e-01 -2.453241e-01  
## 2012 5.856481e-02 -1.349074e-01  
## 2013 1.714815e-01 1.717593e-02  
## 2014 2.439815e-02 -3.236574e-01  
## 2015 1.606481e-01 -3.907407e-01  
## 2016 1.856481e-02 3.280093e-01  
## 2017 NA NA  
##   
## $figure  
## [1] -0.63583333 0.49347222 0.31523148 -0.20324074 -0.06333333  
## [6] -0.01467593 0.19726852 -0.04009259 -0.12634259 -0.13462963  
## [11] -0.07189815 0.28407407  
##   
## $type  
## [1] "additive"  
##   
## attr(,"class")  
## [1] "decomposed.ts"

plot(decompose(germaninfl))



Overall there is no clear trend of German inflation rate, however, from the seasonality (second from the below), we can see some pattern (Christmas period - high inflation due to mass parchasing behavior).

The simple decomposition method is helpful, but the new methods for seasonal decomposition are: - x11() - Seats() - STL()

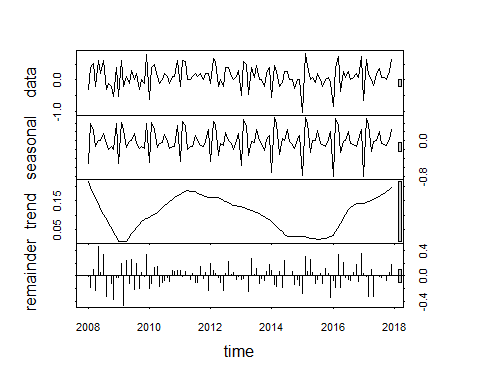
As the benefits of the above new methods, they don’t require omitting values (i.e., No NA values), and the seasonal part can be adjusted over time.

1. STL function

* Seaonsla and trend decomposition with LOESS
* Function of R base/data=time series
* Robust towards outlier
* Suitable toward an additive model (For multiplicative model, data transaformation is needed)
* Seaonanl and trend cycle may change overtime

Example: stl(data,s.window=7) - s.window=: number of required seasonal cycles to calculate changes for the seasonality (e.g., x>=7) - t.window=: number of required seasonal cycles to calculate changes for the trend

# plot visualization  
plot(stl(germaninfl,s.window = 7))



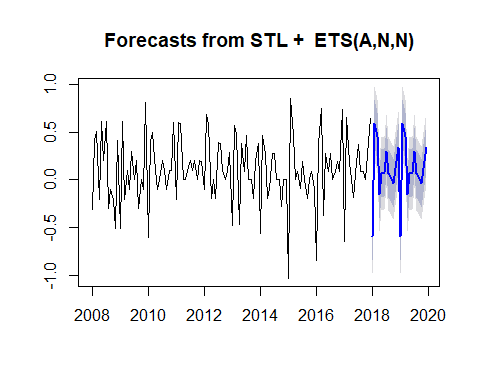
# second to top: seasonality

### Create a forecast with STL decomposition

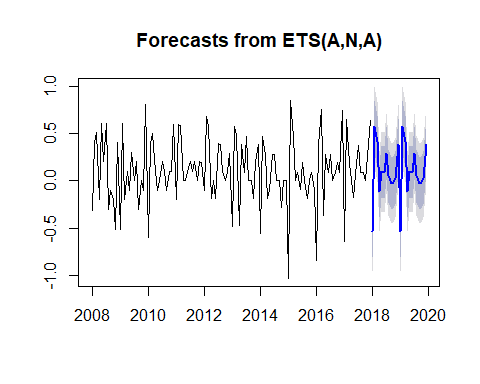
There are two approaches 1. Use stl() on the data 2. Put the results into an object 3. Feed the objection into the forecast.stl() function

1. Use the stlf() function on the data 2. USe the method argument

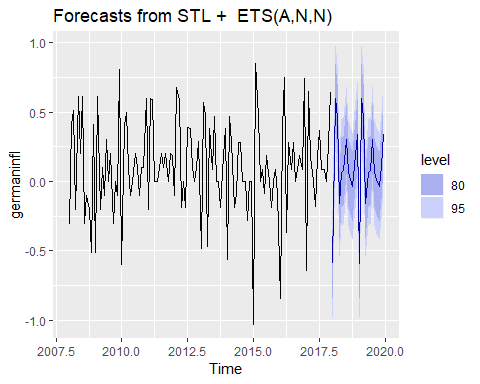
library(forecast)  
  
# stl forecasting + ETS  
plot(stlf(germaninfl,method="ets"))



# comparison with a standard ets forecast  
# peaks and slppes are different  
plot(forecast(ets(germaninfl),h=24))



# using autoplot  
library(ggplot2)  
autoplot(stlf(germaninfl,method="ets"))



## Lecture 19: Seasonal ARIMA

ARIMA model can be applied to seasonal data. For setting, there are two methods. 1. Manual parameter selection with acf() and pacf() 2. Differencing the dataset 3. Feed the data and the parameters into an ARIMA function

1. Feed the time series into the auto.arima() function (“forecast” package) 2. Fine tune the model for best results.

Seasonal ARIMA models have two sets of parameters: A regular set and a second set for the seasonal part

ARIMA (p,d,q) (P,D,Q) [m] - first(p,d,q):non-seasonal part - second(P,D,Q):seasonal part - third(m): frequency

# Seasonal ARIMA (package forecast)  
library(forecast)  
auto.arima(germaninfl,stepwise=T,  
 approximation=F,trace=T)

##   
## ARIMA(2,0,2)(1,1,1)[12] with drift : Inf  
## ARIMA(0,0,0)(0,1,0)[12] with drift : 31.04128  
## ARIMA(1,0,0)(1,1,0)[12] with drift : 21.57402  
## ARIMA(0,0,1)(0,1,1)[12] with drift : 7.519111  
## ARIMA(0,0,0)(0,1,0)[12] : 28.99805  
## ARIMA(0,0,1)(1,1,1)[12] with drift : Inf  
## ARIMA(0,0,1)(0,1,0)[12] with drift : 28.07929  
## ARIMA(0,0,1)(0,1,2)[12] with drift : Inf  
## ARIMA(0,0,1)(1,1,2)[12] with drift : Inf  
## ARIMA(1,0,1)(0,1,1)[12] with drift : 2.573287  
## ARIMA(1,0,0)(0,1,1)[12] with drift : 7.188973  
## ARIMA(1,0,2)(0,1,1)[12] with drift : 0.5156938  
## ARIMA(2,0,3)(0,1,1)[12] with drift : 3.491814  
## ARIMA(1,0,2)(0,1,1)[12] : -1.710145  
## ARIMA(1,0,2)(1,1,1)[12] : Inf  
## ARIMA(1,0,2)(0,1,0)[12] : 18.96391  
## ARIMA(1,0,2)(0,1,2)[12] : Inf  
## ARIMA(1,0,2)(1,1,2)[12] : Inf  
## ARIMA(0,0,2)(0,1,1)[12] : 2.553462  
## ARIMA(2,0,2)(0,1,1)[12] : -0.001234885  
## ARIMA(1,0,1)(0,1,1)[12] : 0.3814517  
## ARIMA(1,0,3)(0,1,1)[12] : 0.2762055  
## ARIMA(0,0,1)(0,1,1)[12] : 5.362176  
## ARIMA(2,0,3)(0,1,1)[12] : Inf  
##   
## Best model: ARIMA(1,0,2)(0,1,1)[12]

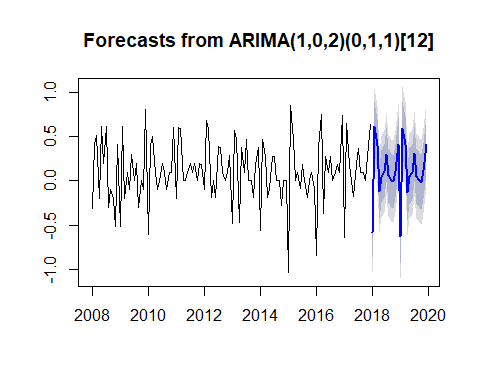
## Series: germaninfl   
## ARIMA(1,0,2)(0,1,1)[12]   
##   
## Coefficients:  
## ar1 ma1 ma2 sma1  
## -0.7883 0.7832 0.2201 -0.7581  
## s.e. 0.1238 0.1494 0.1005 0.1369  
##   
## sigma^2 estimated as 0.04927: log likelihood=6.15  
## AIC=-2.3 AICc=-1.71 BIC=11.11

ARIMA(*1*,0,*2*) (0,1,1)[12] is selected as the best model. 1. *auto correlation 1* coefficient -0.7947 2. Moving average *1* coefficient 0.7786 3. Moving average *2* coefficient 0.215

# Getting an object <forecast>  
germaninflarima <- auto.arima(germaninfl,  
 stepwise=T,  
 approximation=F,  
 trace=T)

##   
## ARIMA(2,0,2)(1,1,1)[12] with drift : Inf  
## ARIMA(0,0,0)(0,1,0)[12] with drift : 31.04128  
## ARIMA(1,0,0)(1,1,0)[12] with drift : 21.57402  
## ARIMA(0,0,1)(0,1,1)[12] with drift : 7.519111  
## ARIMA(0,0,0)(0,1,0)[12] : 28.99805  
## ARIMA(0,0,1)(1,1,1)[12] with drift : Inf  
## ARIMA(0,0,1)(0,1,0)[12] with drift : 28.07929  
## ARIMA(0,0,1)(0,1,2)[12] with drift : Inf  
## ARIMA(0,0,1)(1,1,2)[12] with drift : Inf  
## ARIMA(1,0,1)(0,1,1)[12] with drift : 2.573287  
## ARIMA(1,0,0)(0,1,1)[12] with drift : 7.188973  
## ARIMA(1,0,2)(0,1,1)[12] with drift : 0.5156938  
## ARIMA(2,0,3)(0,1,1)[12] with drift : 3.491814  
## ARIMA(1,0,2)(0,1,1)[12] : -1.710145  
## ARIMA(1,0,2)(1,1,1)[12] : Inf  
## ARIMA(1,0,2)(0,1,0)[12] : 18.96391  
## ARIMA(1,0,2)(0,1,2)[12] : Inf  
## ARIMA(1,0,2)(1,1,2)[12] : Inf  
## ARIMA(0,0,2)(0,1,1)[12] : 2.553462  
## ARIMA(2,0,2)(0,1,1)[12] : -0.001234885  
## ARIMA(1,0,1)(0,1,1)[12] : 0.3814517  
## ARIMA(1,0,3)(0,1,1)[12] : 0.2762055  
## ARIMA(0,0,1)(0,1,1)[12] : 5.362176  
## ARIMA(2,0,3)(0,1,1)[12] : Inf  
##   
## Best model: ARIMA(1,0,2)(0,1,1)[12]

forec <- forecast(germaninflarima)  
plot(forec)



ARIMA is one option for seasonal data set. When we have such a data, ETS model, and Holt-Winters model should be considered together.

## **Lecture 20: Exponential Smooothing with ETS()**

Project II

Modeling with exponential smooothing is: - Holt-winters seasonal method hw() - ETS method ets()

Parameters are (error),(trend),(seasonality).

Depending on the dataset, we might not need to consider seasonality. Generally, smoothing condition is categorized based on the scale of x: - Smooth model (older data is considered, x close to 0) - Reactive model (Heavily relies on recent data, x close to 1)

Activate the forecast package Exponential smooothing function iwth automated parameter seection, data= time series, model=“ZZZ” automated parameter selection.

# library(forecast)  
# est(data)  
# # Function arguments  
?ets()

## starting httpd help server ... done

The value of the model arguments are: - Z: Auto generated - A: Additive - M: Multiplicative - N: Non-existent

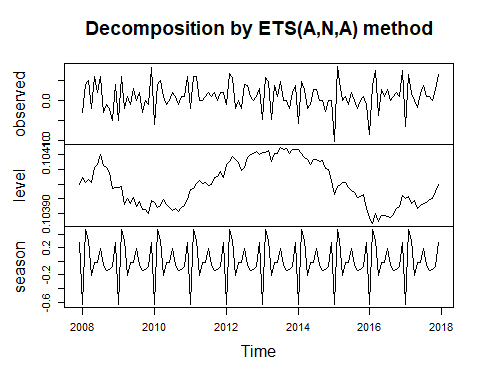
Each parameter of the model can be set to one of above values. The error must be existent.

For example, model==“MZM” is going to be: multiplicative error, auto selected trend, multiplicative seasonality.

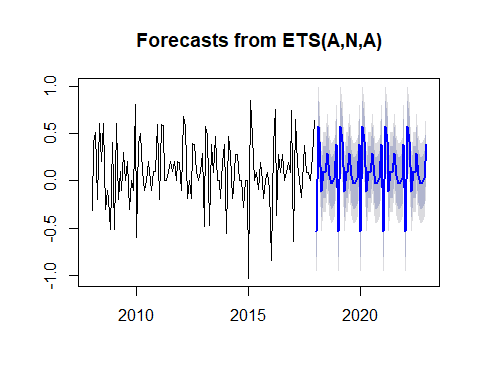
# Exponentila smoothing with ets  
# Auto generated  
library(forecast)  
ets(germaninfl)

## ETS(A,N,A)   
##   
## Call:  
## ets(y = germaninfl)   
##   
## Smoothing parameters:  
## alpha = 1e-04   
## gamma = 1e-04   
##   
## Initial states:  
## l = 0.104   
## s = 0.2832 -0.0736 -0.1241 -0.1295 -0.0511 0.19  
## -0.0164 -0.0093 -0.2138 0.3141 0.4727 -0.6423  
##   
## sigma: 0.2153  
##   
## AIC AICc BIC   
## 220.9850 225.6004 262.7974

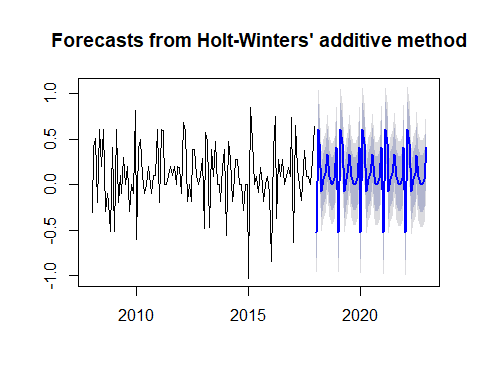
# forecast plot  
germaninflets=ets(germaninfl)  
plot(germaninflets)



plot(forecast(germnainflets,h=60))



# comparison with seasonal Holt Winter model  
plot(hw(germaninfl,h=60))



## **Lecture 21: Time series cross validation**

### **How to compare time series model**

Infomration criteria is not a sufficinet base to compare different systems of models. Forecast accuracy is measured by mean square error (MSE). We use function tsCV() from the forecastpackage.

However, the computation of errors might take long. Alternative function could be cvar() for autoregressive neural nets.

In calculating the error rate, producing an error rate for each time point is the series - actual value vs calculated value

Training and test sets - Test = 1 observation - Training = all observations before the test obs

The error rates are comuputed along timeline (rolling forecast origin). The number of step-ahead forecasts can be specified (window argument)

The forecast accuracy is average value of the all the errors of the whole time series.

Mdels and forecasts we prodocued are: 1. Seasonal Decomposition stlf() 2. Seasonal ARIMA auto.arima() 3. Exponential Smoothing ets()

Steps to proceed 1. Create the function for the model 2. Feeding the time series and the model function into the tsCV() function 3. Generate the mean of the error rates

**Function for the ETS model** - x=dataset - h=forecast length - est() function from the forecast package

forecastes=  
 function(x,h)  
 {forecast(est(x),h=h)}

**Function for the ARIMA model** - x=dataset - h=forecast length - auto.arima() function from the forecast package - Getting the most accurate model possible with stepwise=T and approximation=F

library(forecast)  
forecastarima=  
 function(x,h)  
 {forecast(auto.arima(x),  
 stepwise=T,  
 approximation=F,  
 h=h)  
 }

We also need to defice cross validation.

# etserror=tsCV(germaninfl,forecastets,h=1)  
# arimaerror=tsCV(germaninfl,forecastarima,h=1)

The tsCV() results in a vector containing errors between forecasts and actual values.

We choose a metric of forecast accuracy - RMSE: Root mean squared error - MSE: Mean squared error

The difference between two errors, root of the results or not. The better model with lower error is the ARIMA model.

# error calculation  
# mean(etserror^2,na.rm=T)  
# mean(arimaerror^2,na.rm = T)

Alternatively, we can go with the following options. 1) Training (80) + Test Set (20) + accuracy() 2) Time series cross validation with tsCV()

# Section 4: Project III Irregularly Spaced Data: Analyze A Biotech Stock

## Lecture 22: Mission Statement

In the project III, we focus on 1) Scraping data from Yahoo Finance, and 2) Processing of real world data.

Step 1) Getting the dataset By quantmod lirary, we use the function getSymbols() and its variations for scraiping data from various sources for the Novartis stock (NVS)

Novartis Stock (NVS) 01.Jan 2015 - 01.Jan 2016

### Getting familar with the dataset

What is the general dataset structure? Which class is the dataset? What are the potential problems? Get some initial models - e.g., ARIMA, ETS? Do forecastng on the original dataset Compare the models - How do they perform against each other?

**Key Question** Are there any patterns between the traiding theys which could be exploited?

Workplan - data.frame format - data is recognized as date value - Identifying missing days - Delete the weekends: frequency=5 - Impute the NAs - na.x() family of functions (e.g., na.locf()) - Answering the main analytical question with plots (monthplot() and seasonplot() with the highest and lowest prices).

## LEcture 23：scraping the Data from Yahoo Finance

1. Scraping data through API

* Bloomberg
* Google finance
* Yahoo Finance
* Oanda
* Federal Reserve

1. Outsourcing Tasks through API

* Tableau

Quantiative Financial Modeling Framework - install and activate the package quantmod

# install.packages("quantmod")  
library(quantmod)

Function family getSymbols() no API connection is required, in the getSymbols() - argument src ="": “google”,“yahoo”,“oanda”,“MySQL” - variation of getSymbols(): getSymbols.google(), getSymbols.yahoo(), getSymbols.MySQL()

# activate the package "quantmod"  
library(quantmod)  
  
# name of the new object  
# novartis = getSymbols(  
# # function for stock data scraping  
# symbol="NVS",  
# #stock symbol - ticker for Novartis stock  
# # scraping more than one stock, symbol=c("","")  
# auto.assign = F,  
# # assign an object name automatically - FALSE for manual assignemnt of single stack, while scraping multiple stocks this must be TRUE  
# from="2015-01-01",  
# to="2016-01-01")  
  
novartis = getSymbols("NVS", auto.assign=F,   
 from = "2015-01-01", to = "2016-01-01")  
  
sessionInfo()  
getSymbols("AAPL")  
getSymbols("OIH", src = "yahoo", auto.assign=FALSE)[,1:5]  
w <- curl::curl\_fetch\_memory("https://finance.yahoo.com",curl::new\_handle(verbose=TRUE))  
  
# View(novartis)

xts refers to extensible time series data. The feature of this data type is: - multivariate format - rowID = Date - Data is available only for tranding days - Weekend and Holidays are not present - Irregular spaced dataset - Frequency cannot be asigned - Seasonal patterns cannot be identified

We plot the data

# data visualization  
plot(as.ts(novartis$NVS.Open))

## Lecture 24: Exploring the data

Getting started with irregular spaced time series data, most time series functions and models don’t work effectively on irregularly spaed data. Some teqniques help to get a basic ide about the data even if irregulaly spaced.

The first option is plot the data

# activate quantmod library  
library(quantmod)  
  
# Function chartSeries() for plotting quantmod derived financial time series data   
 # data = time series data  
 # type = chart type ("auto","line","bars","candlesticks","matchsticks")  
chartSeries(data=novartis,type="line")  
  
# ac and pacf to get an idea about auto-  
library(forecast)  
ggtsdisplay(novartis$NVS.Open)

# ARIMA model  
novartisarima = auto.arima(novartis$NVS.Open,  
 stepwise=T,  
 approximation=F,  
 trace=T);novartisarima  
# ARIMA (0,1,1) is obtained - data seems randomly distributed  
# AIC=895, several models were tested and AIC of the optimal model is minimal  
  
# Alternative ARIMA with autogregressive part  
novartisarima2 = Arima(novartis$NVS.Open,order=c(1,1,1))  
novartisarima2

# Forecast arima  
plot(forecast(novartisarima,h=20))  
plot(forecast(novartisarima2,h=20))

The feature of the above optimal model’s plot is; - MA (moving average) with short window - Almost like an LOCF (last observation carried foward) model - No clear trend - No seasonality

When we see the alternative **ARIMA** model, it is not that different. The ARIMA models assume that the last observation is indicative of the next one. Prior obsevations don’t matter much.

Then, we construct a **ETS(exponential smooothing)** model for Novatis stock data. It does not look different, the model is based on M (multipled initial level), no trend nor seasonality, several of the last observations are considered.

# ETS model  
novartisets = ets(novartis$NVS.Open)  
# Forecast model  
plot(novartisets, h=20)

As a conclusion, clear and actionable patterns are hard to extract.Tiny advantages=tiny gains, going deeper with the analysis 1) Is there a difference in the day of the week? 2) Are rises expected on a specific week day? 3) Which months are the most promising ones for certain strategies?

For the above things, pre-processing is required to answer those type of question, and converted to regular time series data.

## Lecture 25: Project III Script

# Fetching data from yahoo with quantmod  
library(quantmod)  
  
novartis = getSymbols("NVS", auto.assign=F,   
 from = "2015-01-01", to = "2016-01-01")  
# use argument return.class to modify output class to ts or mts  
  
## using a column like a standard ts  
plot(as.ts(novartis$NVS.Open))  
  
# functions to explore unprocessed xts from quantmod  
chartSeries(novartis, type = "line")  
  
# acf and pacf to get an idea about autocorrelation  
library(forecast)  
ggtsdisplay(novartis$NVS.Open)  
  
# Arima model  
novartisarima = auto.arima(novartis$NVS.Open,   
 stepwise = T,   
 approximation = F,   
 trace = T); novartisarima  
  
# Alternative arima with autoregressive part   
novartisarima2 = Arima(novartis$NVS.Open, order = c(1,1,1))  
novartisarima2  
  
# Forecast arima  
plot(forecast(novartisarima, h = 20))  
plot(forecast(novartisarima2, h = 20))  
  
# Ets model  
novartisets = ets(novartis$NVS.Open)  
  
# Forecast ets  
plot(forecast(novartisets, h = 20))  
  
## Getting a regular time series  
  
# Conversion to dataframe  
novartis = as.data.frame(novartis)  
  
### Adding the rownames as date  
novartis$Date = rownames(novartis)  
novartis$Date = as.Date(novartis$Date)  
head(novartis)  
  
# Creating the date column, 'by' can be either nr days or integer  
# 'from' and 'to' with as.Date to make sure this is a date format  
mydates = seq.Date(from = as.Date("2015-01-01"),   
 to = as.Date("2016-01-01"),   
 by = 1)  
  
# Converting to a df (required for the merge)  
mydates = data.frame(Date = mydates)  
  
# Padding with 'mydates'  
mydata = merge(novartis, mydates, by = "Date", all.y = T)  
  
# Removing initial days to start on monday  
mydata = mydata[5:366,]  
  
# Removing sundays, watch the from as the first one to remove  
mydata = mydata[-(seq(from = 7, to = nrow(mydata), by = 7)),]  
# Removing saturdays  
mydata = mydata[-(seq(from = 6, to = nrow(mydata), by = 6)),]  
  
# Using last observatoin carried forward imputation  
mydata = na.locf(mydata)  
  
## Which days are the ones best to buy or sell?  
  
# Putting the closeprice into a weekly time series  
highestprice = ts(as.numeric(mydata$NVS.High),   
 frequency = 5)  
  
# Various plots  
seasonplot(highestprice, season.labels = c("Mon", "Tue", "Wed", "Thu", "Fri"))  
monthplot(highestprice)  
monthplot(highestprice, base = median, col.base = "red")  
plot(stl(highestprice, s.window = "periodic"))  
  
# Comparison with the low prices  
par(mfrow = c(1,2))  
lowestprice = ts(as.numeric(mydata$NVS.Low),   
 frequency = 5)  
monthplot(lowestprice, base = median, col.base = "red")  
monthplot(highestprice, base = median, col.base = "red")  
par(mfrow = c(1,1))

## Lecture 26: Getting a Regular Time Series

1. Data vs Analytical Goals

* Irregularly spaced data e.g., Holidays and weekends are missing, US traiding days only
* Analytical questions e.g., Is there any pattern in the trading days?

1. Conveting the data into the Regularly spaced time series
2. Convert the original data into a data frame, and (B) Create a second data.frame with a calender days result in merging both data frames to the *regular spaced time series with NAs*

### Conversion to a data frame

novartis=as.data.frame(novartis)

### Add the row names as date

novartis %>%   
 mutate(Date=rownames(novartis)) %>%   
 as.Date(Date)  
head(novartis)

* Name of the object to be created
* Sequence of data values
* Parameters “from” and “to” to specify the time window
* data class=as.Date(“”)
* Parameter “by” to specify the step size

# creating the date column, "by" can be either nr days or   
# "from" and "to" with as.Date to make sure this is a date   
  
mydates=  
seq.Date(  
 from=as.Date("2015-01-01"),  
 to=as.Date("2016-01-01"),  
 by=1)  
  
# converting to a df(tibble)  
library(tidyverse)  
mydates=tibble(Date=mydates)

### Padding with “mydates” by merge()

* merting the two data frames
* identifying missing dates and marking them as NA
* Name of the new data frame
* Merging function of R Base (alternatively, join functions from dplyr package)
* First object(X)
* Second object(Y)
* Name of the column to merge by
* All rows of object Y get included
* Results in a full join of the two data frames

mydata = merge(novartis, mydates,by="Date",all.y=T)

### Removing initial days to start on Monday

* removing Saturdays and Sundays
* Impute holiday prices with na.locf()

First of all subset the data

mydata=mydata[5:366,]  
  
# Removing Sundays, watch the from   
mydata <-   
# Removing Saturday  
mydata <-

For the above Holiday extraction, we use seq() - substracte the sequence from mydata - sequence generation function of R base: seq() - Form = the first observation to be removed - To= the last observation of the dataset

mydata =  
 mydata[-(  
 seq(  
 from=7,  
 to=nrow(mydata),  
 by=7)),]

To confirm the process: 1) Converting the original dataset into a data frame 2) Create a second data frame with all the calender days 3) Merging the two data frames 4) Remove all the Saturdays and Sundays 5) Impute Holiday prices with a last observation carried foward method

## Lecture 27: Visually analyzing the data

We convert the time-series data with frequency five.

* Extracting the highest prices
* Name of the ts object to be created
* Time series, convert the data to numeric
* frequency to five

# Putting the close price into a weekly time series  
library(tidyverse)

## Warning: package 'tidyverse' was built under R version 3.5.1

## -- Attaching packages --------------------------- tidyverse 1.2.1 --

## √ tibble 1.4.2 √ purrr 0.2.5  
## √ tidyr 0.8.1 √ dplyr 0.7.6  
## √ readr 1.1.1 √ stringr 1.3.1  
## √ tibble 1.4.2 √ forcats 0.3.0

## Warning: package 'purrr' was built under R version 3.5.1

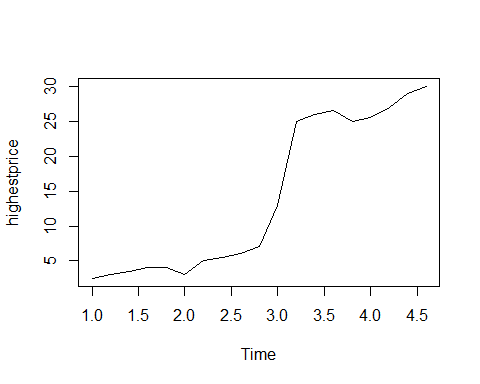
## Warning: package 'dplyr' was built under R version 3.5.1

## -- Conflicts ------------------------------ tidyverse\_conflicts() --  
## x dplyr::filter() masks stats::filter()  
## x dplyr::first() masks xts::first()  
## x dplyr::lag() masks stats::lag()  
## x dplyr::last() masks xts::last()

# supplement sample data  
mydata <- tibble(  
 High=c(2.5,3,3.5,4,4,3,5,5.5,  
 6,7,13,25,26,26.5,25,25.5,27,29,30)  
)  
mydata

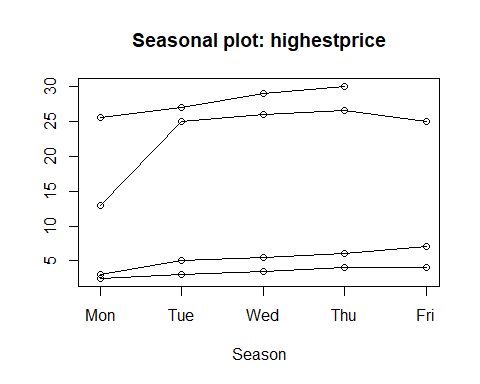
## # A tibble: 19 x 1  
## High  
## <dbl>  
## 1 2.5  
## 2 3   
## 3 3.5  
## 4 4   
## 5 4   
## 6 3   
## 7 5   
## 8 5.5  
## 9 6   
## 10 7   
## 11 13   
## 12 25   
## 13 26   
## 14 26.5  
## 15 25   
## 16 25.5  
## 17 27   
## 18 29   
## 19 30

highestprice =  
 ts(  
 as.numeric(mydata$High),  
 frequency=5  
 )  
plot(highestprice)

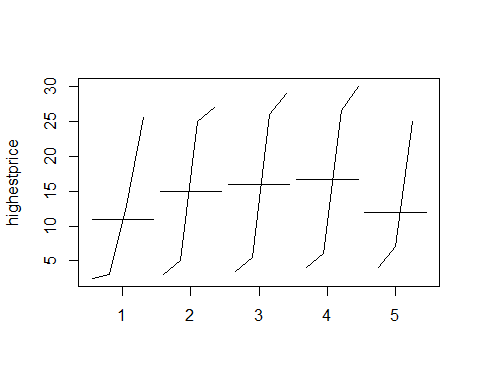


Plotting the weeks against each other, two functions are useful: seasonaplot() and ggseasonaplot().

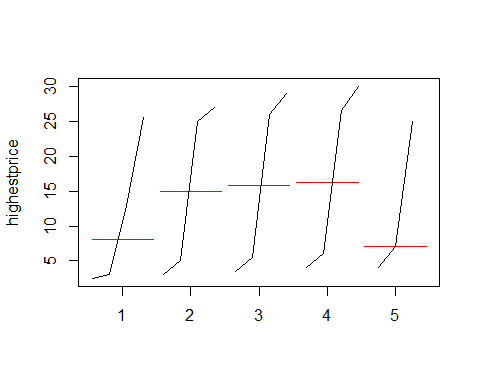
library(forecast)  
  
seasonplot(highestprice,season.label=c("Mon","Tue","Wed","Thu","Fri"))



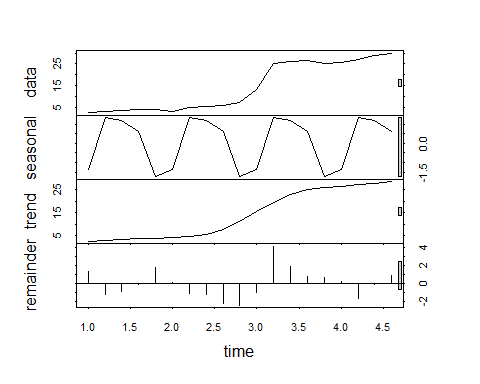
monthplot(highestprice)



monthplot(highestprice,base=median,col.base="red")



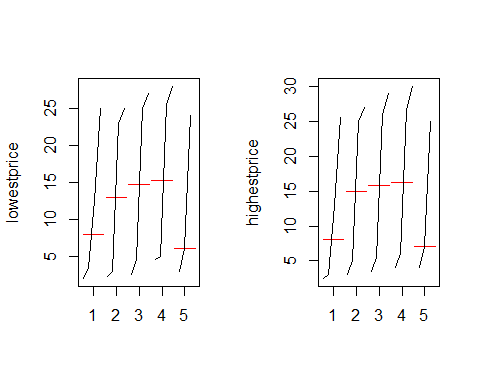
plot(stl(highestprice,s.window="periodic"))



# comparison with the low prices  
mydata1 <- tibble(  
 Low=c(2,2.3,2.5,4.5,3,3.5,3,4.5,  
 5,6,12.5,23,25,25.5,24,25,25,27,28)  
)  
mydata2 <- bind\_cols(mydata1,mydata)  
mydata2

## # A tibble: 19 x 2  
## Low High  
## <dbl> <dbl>  
## 1 2 2.5  
## 2 2.3 3   
## 3 2.5 3.5  
## 4 4.5 4   
## 5 3 4   
## 6 3.5 3   
## 7 3 5   
## 8 4.5 5.5  
## 9 5 6   
## 10 6 7   
## 11 12.5 13   
## 12 23 25   
## 13 25 26   
## 14 25.5 26.5  
## 15 24 25   
## 16 25 25.5  
## 17 25 27   
## 18 27 29   
## 19 28 30

lowestprice =ts(as.numeric(mydata2$Low),  
 frequency=5)  
  
# data plot check  
par(mfrow=c(1,2))  
monthplot(lowestprice,base=median,col.base="red")  
monthplot(highestprice,base=median,col.base="red")



par(mfrow=c(1,1))

# Section 5: Project IV Neural Networks: Neural Nets and Interactive Graphs

## Lecture 39: Mission Statement

**Step 1: Data pre-processing** 1) Clean the data, as real world date often needs to be cleaned prior to analysis. People collecting data are not always trained in data science. 2) A time consuming process The goal is to prepare a numeric time series format, missing data imputed, and outlier detection and replacement. 3) Tools tidyr package - separate() forecast package - tsclean()

**Step2: Modeling the data** 1) Neural network model forecast 36 months using forecast package - nnetar(), Seasonal changes

**Step3: Data visualization using the Interactive chart** 1) Interative data visualization Plot the model and the forecast, using interactive chart (zoom-in tool, reveal more detail upon hovering) - dygraphs package 2) Further option R Shiny

## Lecture 29: Getting familar with ras dataset

Several cleaning steps are required for the dataset. PRoblems to be fixed are 1) missing data, 2) outliers, 3) poor formating (quotations) and 4) data type.

The data is about the restaurant at a campsite: - open the whole year - peak season in summer - seasonal data

Once a data is properly cleanied, - trend might be present - revenue by month in USD - 1997-2016

The data set is donwloadable from the Udemy website.

## Project IV script

## Import Data Camping\_Revenue as revenue  
  
# check the quotes while importing to get 2 columns  
class(revenue$V2)  
  
# chopping off the useless quotes at 2 positions  
library(tidyr)  
revenue <- separate(revenue, col = V2,   
 sep = c(2, -3),   
 into = c("rest", "data", "rest2"))  
  
# all the relevant data is in column "data"  
head(revenue)  
  
# class is still a character (with some missing data)  
class(revenue$data)  
  
# conversion to time series  
myts <- ts(as.numeric(revenue$data),  
 start = 1997, frequency = 12)  
  
# data is still not clean (outliers and NAs)  
summary(myts)  
  
# all in one cleaning tool  
library(forecast)  
myts <- tsclean(myts)  
  
# check the data  
summary(myts)  
  
plot(myts)  
  
# set up a Neural Network model  
mynnetar <- nnetar(myts)  
  
# forecasting 3 years with the model  
nnetforecast <- forecast(mynnetar, h = 36,  
 PI = T)  
library(ggplot2)  
autoplot(nnetforecast)  
  
## interactive dygraph  
  
# data we need for the graph  
data <- nnetforecast$x  
lower <- nnetforecast$lower[,2]  
upper <- nnetforecast$upper[,2]  
pforecast <- nnetforecast$mean  
  
mydata <- cbind(data, lower, upper,  
 pforecast)  
  
library(dygraphs)  
  
dygraph(mydata, main = "Oregon Campsite Restaurant") %>%   
 dyRangeSelector() %>%   
 dySeries(name = "data", label = "Revenue Data") %>%  
 dySeries(c("lower","pforecast","upper"), label = "Revenue Forecast") %>%  
 dyLegend(show = "always", hideOnMouseOut = FALSE) %>%  
 dyAxis("y", label = "Monthly Revenue USD") %>%  
 dyHighlight(highlightCircleSize = 5,  
 highlightSeriesOpts = list(strokeWidth = 2)) %>%  
 dyOptions(axisLineColor = "navy", gridLineColor = "grey") %>%  
 dyAnnotation("2010-8-1", text = "CF", tooltip = "Camp Festival", attachAtBottom = T)

## Lecture 31: Cleaning with tidyr

1. Import the data into RStudio
2. Remove the quotes (part of the dataset)
3. Convert the data into a time series
4. Detecting and replacing missing data and outliers

# package  
library(tidyverse)  
library(lubridate)

##   
## Attaching package: 'lubridate'

## The following object is masked from 'package:base':  
##   
## date

# tidyverse  
# revenue <- read.csv("C:/Users/kojikm.mizumura/Desktop/Data Science/1. Udemy/Applied Time Series and Forecasting with R/camping-revenue-97-17.csv",sep=',',header=F,quote="\"")  
  
# revenue <- read\_csv("U:/2. Data Science/1. Udemy/Applied Time Series and Forecasting with R/camping-revenue-97-17.csv",col\_names=F)  
  
camping.revenue.97.17 <- read.csv("U:/2. Data Science/1. Udemy/Applied Time Series and Forecasting with R/camping-revenue-97-17.csv", header=FALSE, quote="")  
  
revenue <- camping.revenue.97.17  
head(revenue)

## V1 V2  
## 1 "1 ""16857"""  
## 2 "2 ""14209"""  
## 3 "3 ""15513"""  
## 4 "4 ""16415"""  
## 5 "5 ""19047"""  
## 6 "6 ""20655"""

We chop off the useless quote in the data by using the tidyr package. - Name of the object to be updated - Function separate() to separate strings - Data to be used - column to work with - Index position of characters where to separate - ‘’’|16857|‘’’ (2 characters from the left, 3 characters from the right) - columns to be created (“rest”,“data”,“rest2”)

library(tidyr)  
revenue <-   
 separate(revenue,  
 col=V2,  
 sep=c(2,-3),  
 into=c("rest","data","rest2"))  
head(revenue)

## V1 rest data rest2  
## 1 "1 "" 16857 """  
## 2 "2 "" 14209 """  
## 3 "3 "" 15513 """  
## 4 "4 "" 16415 """  
## 5 "5 "" 19047 """  
## 6 "6 "" 20655 """

Upon the above conversion, we would change the data type of the variable data into time series.

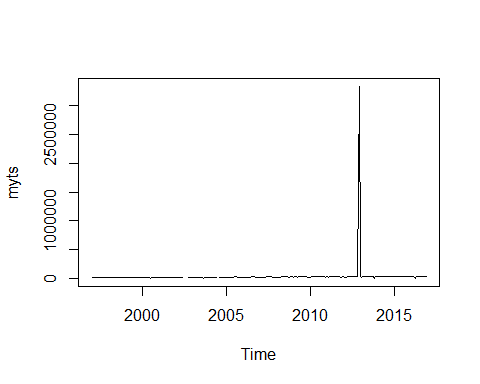
# conversion to time series  
myts <- ts(as.numeric(revenue$data),  
 start=1997,frequency=12)

## Warning in is.data.frame(data): 強制変換により NA が生成されました

summary(myts)

## Min. 1st Qu. Median Mean 3rd Qu. Max. NA's   
## 3 18980 23218 36912 26816 3334333 4

plot(myts)

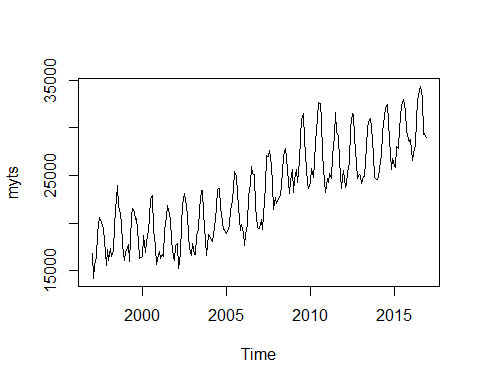


The max and min are obviously outliers, thus, we would apply tsclean function for linear interpolation.

library(forecast)  
myts <- tsclean(myts)  
  
# outlier was adjusted by tsclean()  
summary(myts)

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 14209 19280 23267 23282 26658 34366

plot(myts)



In this lecture, we learned 1) import the data into RStudio, 2) Remove the quotes by separate() from tidyverse 3) Conversion of the data into time series by as.numeric(), ts(), and 4) Detecting and replacing missing data and outliers by tsclean().

## Lecture 32: Fitting the neural net model

In this lecture,we will learn 1) Data pre-processing by tidyr forecast 2) Modeling based on neraul nets by forecast 3) interactive data visualization by dygraphs

The neural net is simplied by one output layer and multiple inputs/layers with hidden layers, called multilayer feed foward network.

NNAR-Neural Network Autoregression Model

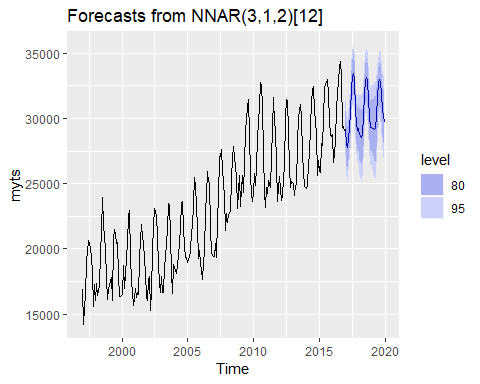
* output - NNAR(p,k) output - NNAR(14,8)
* p lagged values - input 8 nodes(hidden layers)
* k nodes are present - Last 14 lags as input

In the forecast package, we have a function nnetar(). In such function we have a input to set the number of inputs.

## set up a Neural Network Model  
mynnetar <- forecast::nnetar(myts)  
  
# forecast 3 years with the model  
## PI - prediction interval = TRUE (default is FALSE)  
nnetforecast <- forecast(mynnetar,h=36,PI=T)  
  
library(ggplot2)  
nnetforecast

## Point Forecast Lo 80 Hi 80 Lo 95 Hi 95  
## Jan 2017 29172.36 27737.28 30588.86 27138.64 31380.08  
## Feb 2017 27964.60 26399.10 29723.68 25651.16 30515.74  
## Mar 2017 27797.51 26207.38 29415.35 25242.10 30168.35  
## Apr 2017 28363.08 26758.21 29928.29 25994.22 30680.65  
## May 2017 30354.49 28604.21 31843.55 27730.77 32792.14  
## Jun 2017 32329.19 30700.00 33631.78 29854.24 34526.39  
## Jul 2017 33221.89 31697.42 34511.01 30921.60 35268.26  
## Aug 2017 33476.06 31898.08 34879.60 31151.28 35578.14  
## Sep 2017 33031.35 31421.22 34402.76 30700.26 35226.47  
## Oct 2017 30616.36 29116.11 32198.10 28421.26 32871.62  
## Nov 2017 29625.80 28092.64 31166.37 27378.32 32016.46  
## Dec 2017 29053.29 27633.78 30728.49 26766.81 31572.91  
## Jan 2018 29252.92 27571.58 31046.83 26556.19 31812.33  
## Feb 2018 28769.74 26838.71 30840.89 25733.20 31792.97  
## Mar 2018 28514.24 26467.23 30684.09 25176.03 31744.26  
## Apr 2018 28726.81 26694.41 30674.93 25541.73 31907.31  
## May 2018 30123.28 27809.69 31955.87 26960.18 32923.86  
## Jun 2018 31926.90 29702.36 33324.06 28915.20 34336.56  
## Jul 2018 32883.65 31133.40 34209.12 30254.84 35084.54  
## Aug 2018 33125.54 31403.88 34649.49 30445.34 35345.49  
## Sep 2018 32897.74 30905.48 34296.30 29941.16 35200.25  
## Oct 2018 31408.88 29627.47 33112.78 28428.31 33946.38  
## Nov 2018 30153.20 28337.89 31900.82 27190.56 32884.10  
## Dec 2018 29320.92 27536.22 31404.34 26382.54 32329.51  
## Jan 2019 29295.39 27529.95 31412.72 26455.93 32551.96  
## Feb 2019 29192.91 27138.61 31468.56 25704.46 32545.34  
## Mar 2019 29129.02 26787.35 31640.03 25410.81 32753.53  
## Apr 2019 29237.78 26721.70 31667.28 25414.12 32734.82  
## May 2019 30158.96 27689.87 32377.24 26542.14 33376.81  
## Jun 2019 31650.88 29207.54 33205.55 28149.14 34272.20  
## Jul 2019 32643.20 30421.79 33886.55 29325.91 34637.43  
## Aug 2019 32945.50 30940.26 34297.32 30056.73 35193.14  
## Sep 2019 32814.42 30764.80 34178.40 29755.98 35151.37  
## Oct 2019 31881.46 29832.60 33421.68 28739.09 34423.88  
## Nov 2019 30683.27 28642.04 32556.10 27609.90 33629.81  
## Dec 2019 29711.08 27778.88 31907.20 26450.89 33030.49

autoplot(nnetforecast)



The model recognizes that there is a seasonal pattern the summer usually yields a much higher revenue, and the winter does. Interstingly, the yearly peaks of the forecast new stay at an even level in this three year forecast, and the below are ascending. The hear shows the NNAR(3,1,2)[12], which are 3 obs, 1 obs in the last seasonal cycle (total input = 4) conpressed to two hidden layers.

## Lecture 33: Interactive graphs with dygraphs

Web-based application - Shiny - Local - viewed tab - RShiny web server - Markdown document

Library dygraphs - implementation of dygraphs JS library - good quality - good documentation - pipe operator %>%

# interactive dygraph  
# data we need for the graph  
data <- nnetforecast$x  
lower <- nnetforecast$lower[,2]  
upper <- nnetforecast$upper[,2]  
pforecast <- nnetforecast$mean  
  
# library(dplyr)  
# bind\_cols(list(data,lower,upper,pforecast))  
  
mydata <- cbind(data,lower,upper,pforecast)  
head(mydata)

## data lower upper pforecast  
## Jan 1997 16857 NA NA NA  
## Feb 1997 14209 NA NA NA  
## Mar 1997 15513 NA NA NA  
## Apr 1997 16415 NA NA NA  
## May 1997 19047 NA NA NA  
## Jun 1997 20655 NA NA NA

The script structure is

library(dygraphs)  
  
# Fetch the data frame and add a title  
dygraph(mydata,main="Oregon Campsite Restaurant") %>%  
 dyRangeSelector() %>%   
 # zoom in tool  
 dySeries(name="data",label="Revenue Data") %>%   
 # add the time series with a label  
 dySeries(c("lower","pforecast","upper","Revenue Forecast")) %>%  
 # add the forecast, the 95% lower and uppoer CI boundaries and a label  
 dyLegend(show="always",hideOnMouseOut = FALSE) %>%   
 # add a legend (time series + forecast); always show the legend  
 dyAxis("y",label="Monthly Revenue USD") %>%   
 # add axis labels  
 dyHighlight(highlightCircleSize = 5,  
 highlightSeriesOpts = list(strokeWidth=2)) %>%   
 # add highlighting effect - enlarged circle for selected data point + bold line for selected series  
 dyOptions(axisLineColor = "navy",gridLineColor = "grey") %>% # set axia and gridline colors  
 dyAnnotation("2010-8-1",text="CF",tooltip = "Camp Festival",attachAtBottom = T)  
 # add an annotation with tooltip to the timeline, where the `Camp Festival` took place   
library(magrittr)

data <- nnetforecast$x  
lower <- nnetforecast$lower[,2]  
upper <- nnetforecast$upper[,2]  
pforecast <- nnetforecast$mean  
  
mydata <- cbind(data, lower, upper,  
 pforecast)  
  
dygraph(mydata, main = "Oregon Campsite Restaurant") %>%   
 dyRangeSelector() %>%   
 dySeries(name = "data", label = "Revenue Data") %>%  
 dySeries(c("lower","pforecast","upper"), label = "Revenue Forecast") %>%  
 dyLegend(show = "always", hideOnMouseOut = FALSE) %>%  
 dyAxis("y", label = "Monthly Revenue USD") %>%  
 dyHighlight(highlightCircleSize = 5,  
 highlightSeriesOpts = list(strokeWidth = 2)) %>%  
 dyOptions(axisLineColor = "navy", gridLineColor = "grey") %>%  
 dyAnnotation("2010-8-1", text = "CF", tooltip = "Camp Festival", attachAtBottom = T)

## Lecture 34: Course summary

Recommended books 1) Forecasting with Exponential Smoothing by Rob Hyndman 2) Time series analysis ,forecasting and control by Box, Jenkins,Reinsel, Ljung 3) Forecasting with Dynamic Regression Models by Alan Pankratz 4) Multivariate Time Series Analysis by Ruey Tsay 5) Forecasting: Principles and Practice by Rob Hyndman