#################################################################

# TIME SERIES FORECASTING

#################################################################

#Open data set from: https://data.gov.in/search/site?query=paddy+price&field\_search=&item=100

#See this chart first: https://otexts.org/fpp2/arima-r.html

install.packages("TTR")

library(TTR)

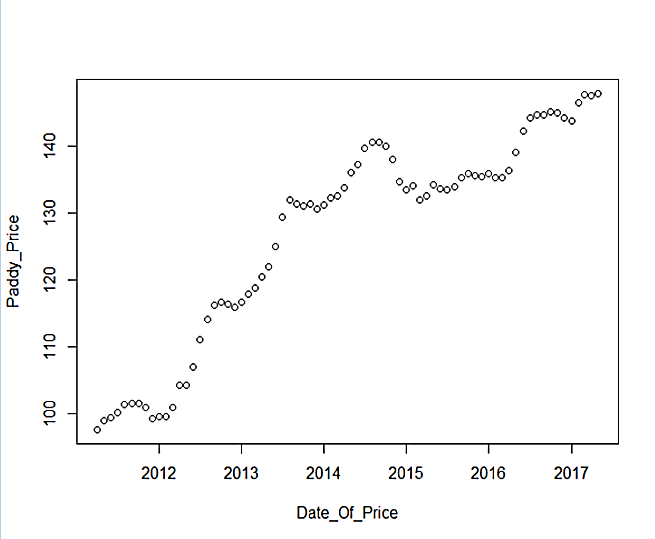
#see how time series data should look like

#data(AirPassengers)

#x=AirPassengers

#str(x)

#Time-Series [1:144] from 1949 to 1961: 112 118 132 129 121 135 148 148 136 119 ...

df=read.csv("paddy.csv")

df$Date\_Of\_Price=as.Date(df$Date\_Of\_Price, format="%d-%m-%Y")

plot(df)

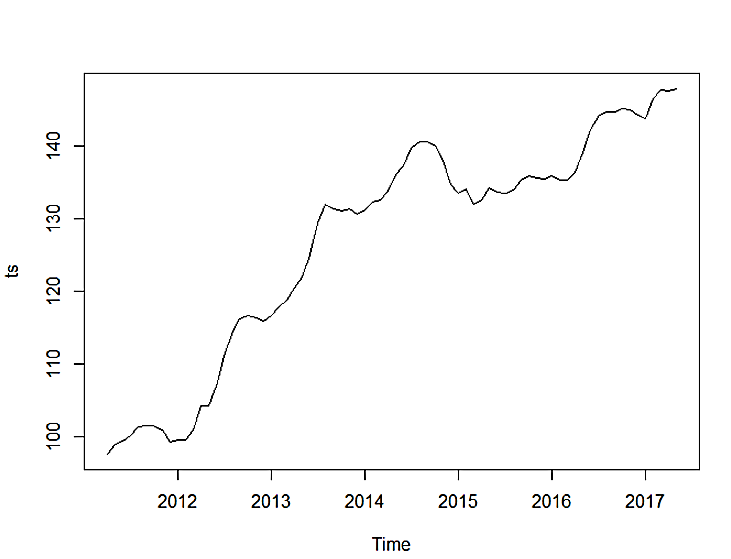
ts=ts(df$Paddy\_Price,frequency = 12,start=c(2011,4)) #year, month

class(ts)

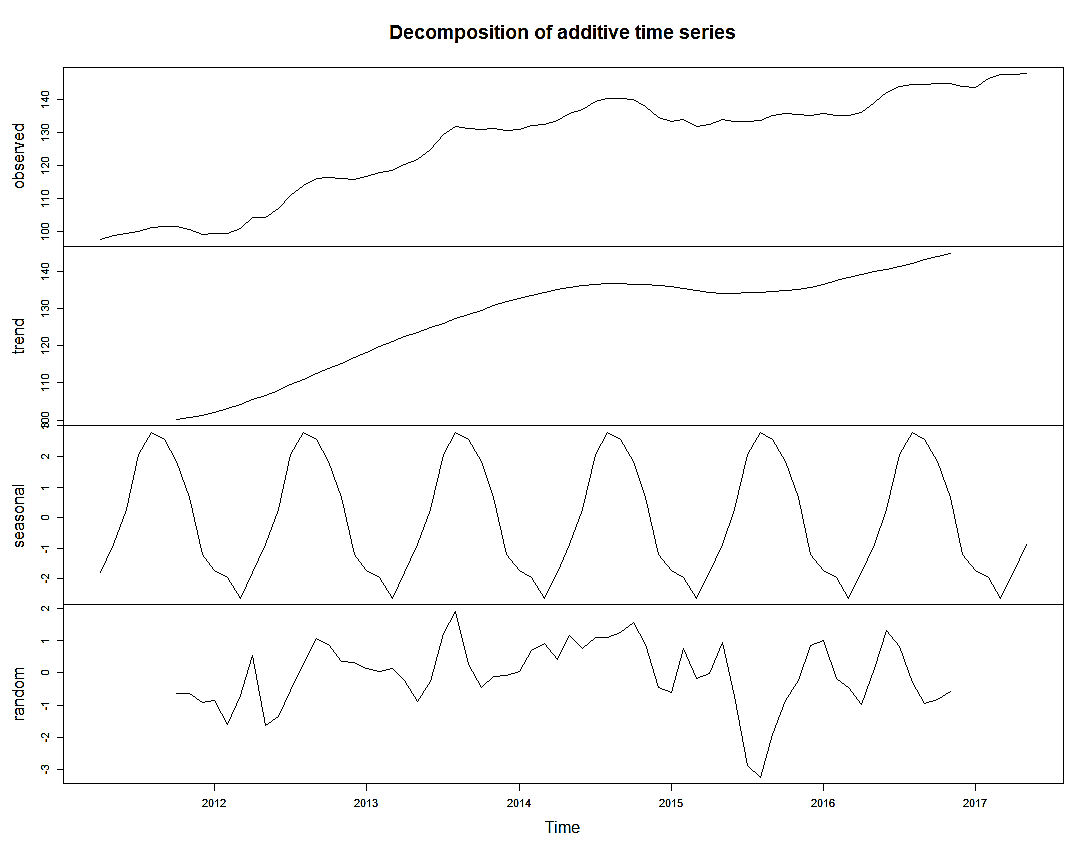
#it should be "ts" only. If we use entire df cols, we will get ts, matrix, etc.

#[1] "ts"

plot(ts)

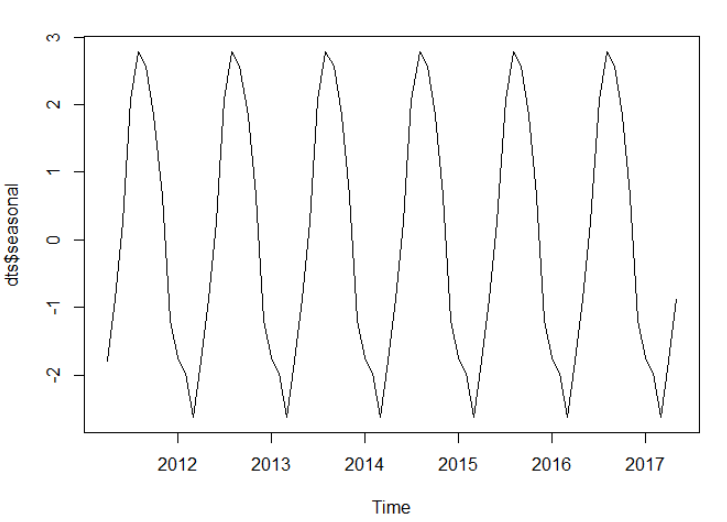


plot(decompose(ts))

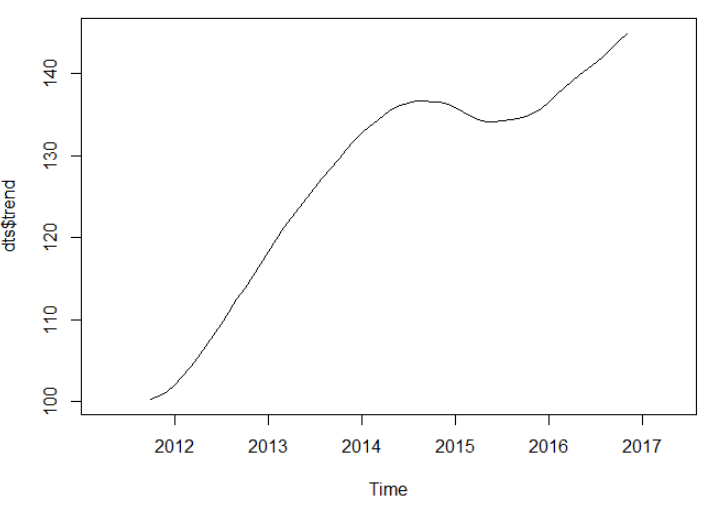


dts=decompose(ts)

plot(dts$seasonal)



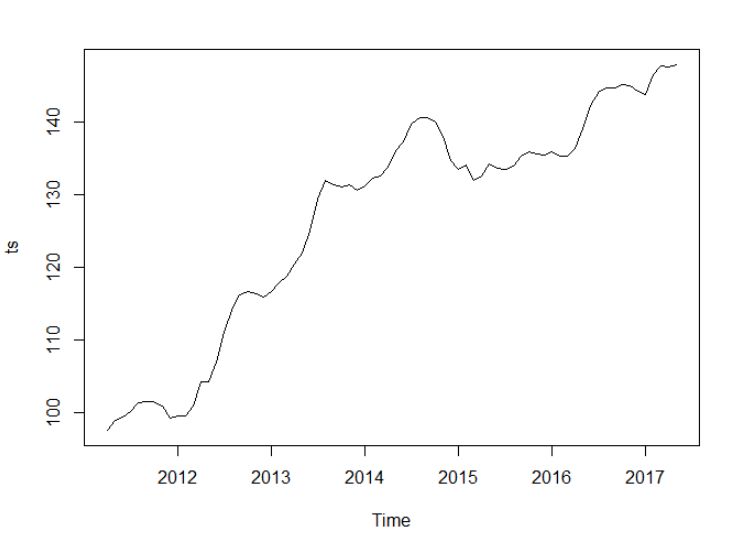
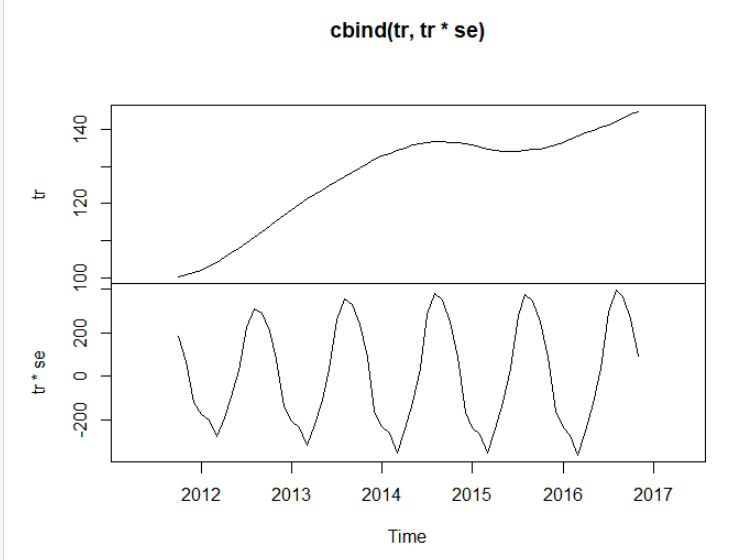
plot(dts$trend)



tr=dts$trend

se=dts$seasonal

plot(cbind(tr,tr\*se)) plot.ts(ts)



#conclusion : this ts has seasonality and trend

##------------------------------- Smoothening techniques ---------------------------

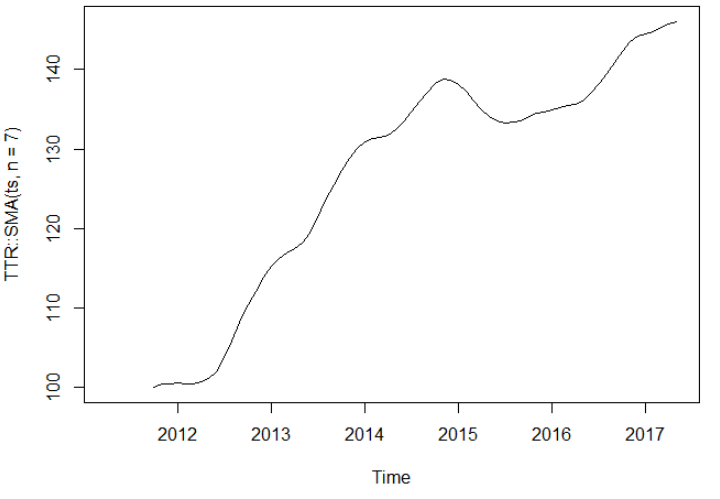
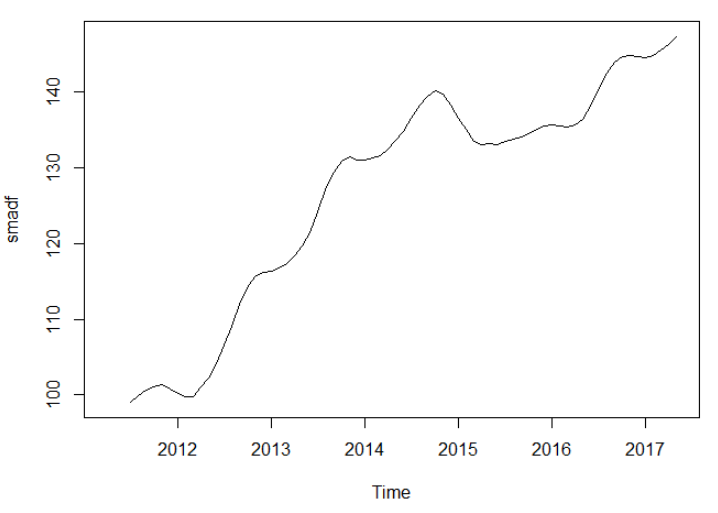
# see table :https://otexts.org/fpp2/taxonomy.html

##--------SMA from TTR package - Simple Moving Average calculations

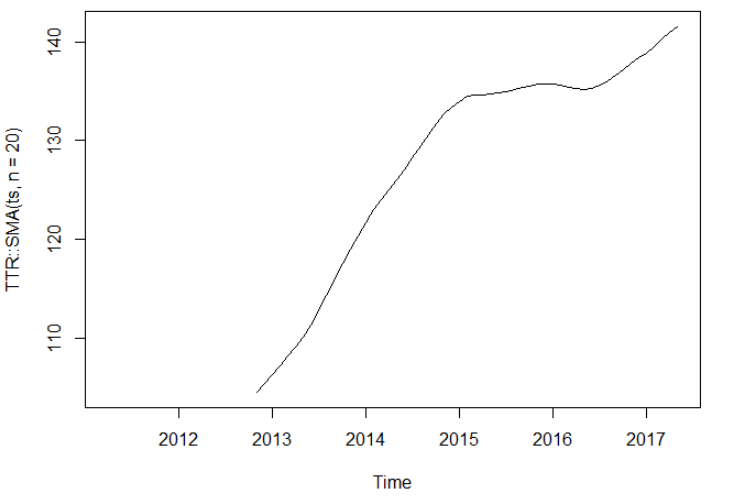
#in Moving Averages the past observations are weighted equally unlike ES methods.

smadf<- TTR::SMA(ts ,n=4) #bigger the n value more the points will get combined and averaged

plot(smadf) plot(TTR::SMA(ts ,n=7))



plot(TTR::SMA(ts ,n=20))



#---------------- ES(Exponential smoothing) methods ---------------------

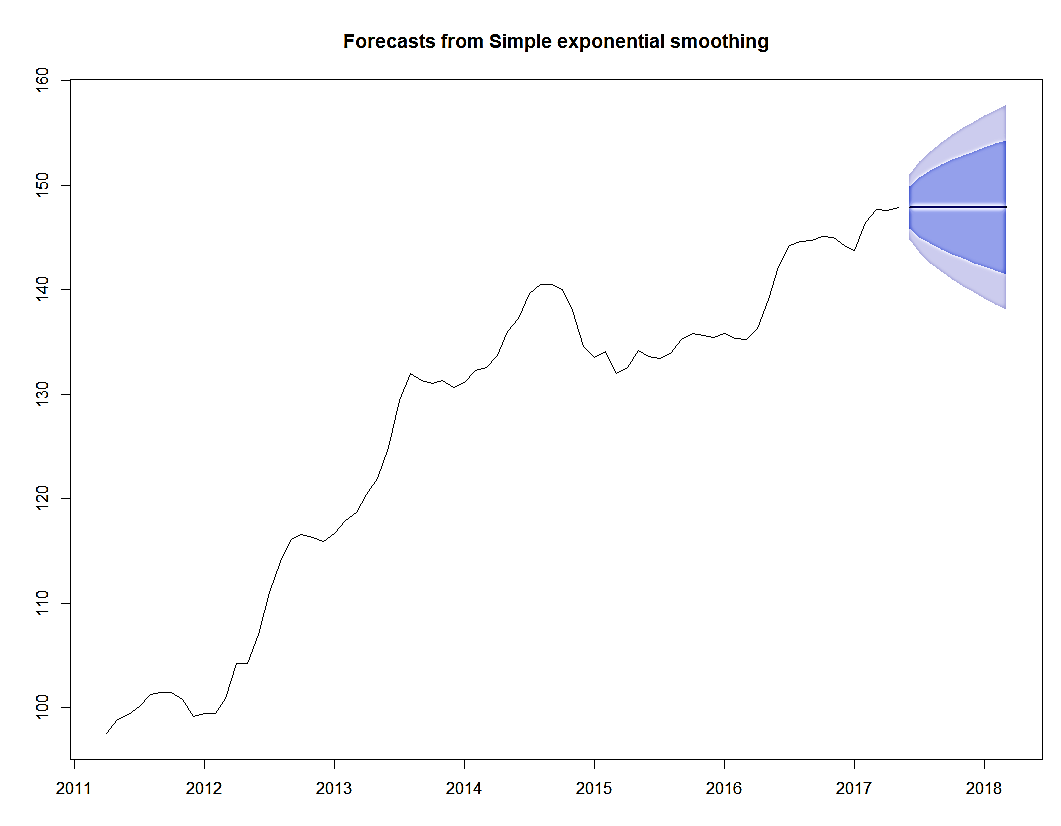
#Exponential Smoothing assigns exponentially decreasing weights as the observation get older. In other words, recent observations are given relatively more weight in forecasting than the older observations.

#ses from forecast package

#This method is suitable for forecasting data with no clear trend or seasonal pattern. Not useful here but just mentioning

library(forecast) ; sests=forecast::ses(ts, h=10) #forecasted value for the next 10 months

plot(sests)



sests #to see the forecasted value for the next 10 months

Point Forecast Lo 80 Hi 80 Lo 95 Hi 95

Jun 2017 147.9 145.8763 149.9236 144.8051 150.9948

Jul 2017 147.9 145.0383 150.7617 143.5234 152.2765

Aug 2017 147.9 144.3952 151.4048 142.5399 153.2601

Sep 2017 147.9 143.8530 151.9469 141.7107 154.0892

Oct 2017 147.9 143.3754 152.4246 140.9802 154.8197

Nov 2017 147.9 142.9435 152.8564 140.3198 155.4802

Dec 2017 147.9 142.5464 153.2535 139.7124 156.0875

Jan 2018 147.9 142.1768 153.6231 139.1471 156.6528

Feb 2018 147.9 141.8296 153.9703 138.6162 157.1837

Mar 2018 147.9 141.5013 154.2986 138.1140 157.6859

summary(sests)

# Forecast method: Simple exponential smoothing

#

# Model Information:

# Simple exponential smoothing

#

# Call:

# ses(y = ts, h = 10)

#

# Smoothing parameters:

# alpha = 0.9999

#

# Initial states:

# l = 97.4998

#

# sigma: 1.579

#

# AIC AICc BIC

# 390.0823 390.4252 396.9945

#

# Error measures:

# ME RMSE MAE MPE MAPE MASE ACF1

# Training set 0.6811515 1.557556 1.140615 0.5550998 0.9135698 0.1165235 0.5419242

#

# Forecasts:

# Point Forecast Lo 80 Hi 80 Lo 95 Hi 95

# Jun 2017 147.9 145.8763 149.9236 144.8051 150.9948

# Jul 2017 147.9 145.0383 150.7617 143.5234 152.2765

# Aug 2017 147.9 144.3952 151.4048 142.5399 153.2601

# Sep 2017 147.9 143.8530 151.9469 141.7107 154.0892

# Oct 2017 147.9 143.3754 152.4246 140.9802 154.8197

# Nov 2017 147.9 142.9435 152.8564 140.3198 155.4802

# Dec 2017 147.9 142.5464 153.2535 139.7124 156.0875

# Jan 2018 147.9 142.1768 153.6231 139.1471 156.6528

# Feb 2018 147.9 141.8296 153.9703 138.6162 157.1837

# Mar 2018 147.9 141.5013 154.2986 138.1140 157.6859

round(forecast::accuracy(sests),2)

# ME RMSE MAE MPE MAPE MASE ACF1

# Training set 0.68 1.56 1.14 0.56 0.91 0.12 0.54

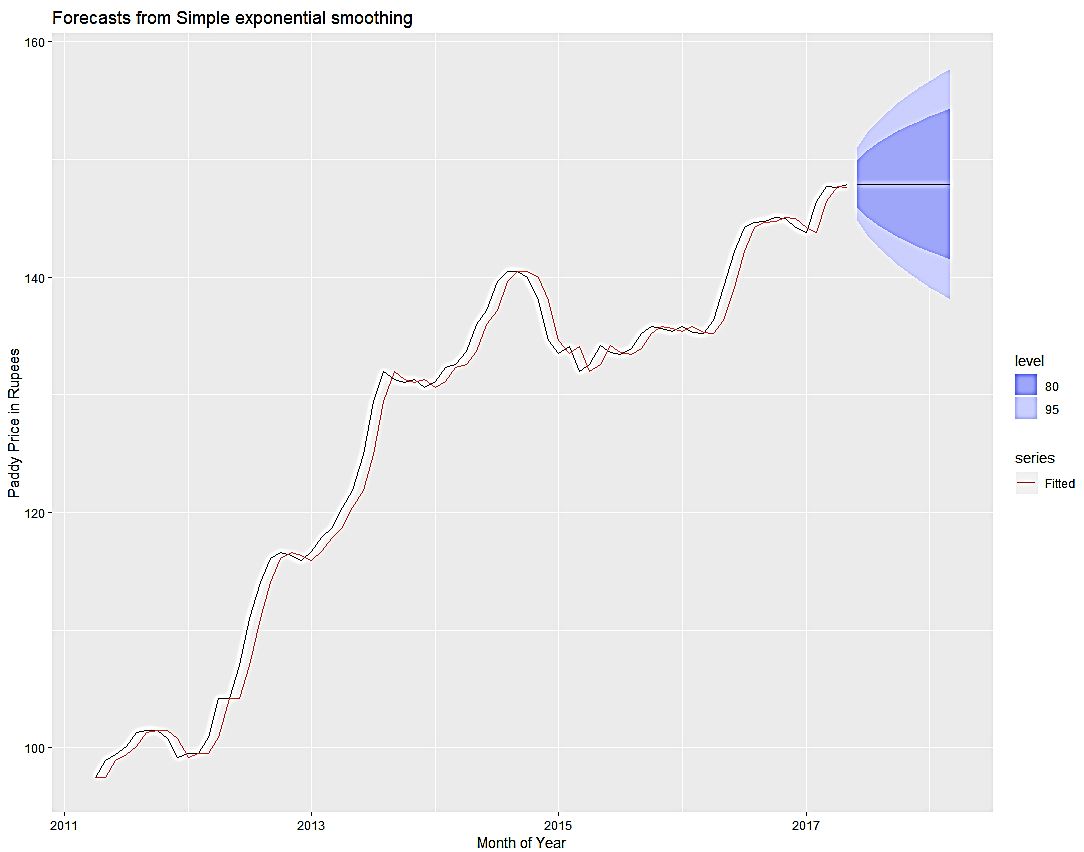
p=forecast::autoplot(sests) + forecast::autolayer(sests$fitted, series="Fitted")

p$labels$x="Month of Year"

p$labels$y="Paddy Price in Rupees"

p #to run the plot again.

#Note the plotted "one-step-ahead" fitted values alongside the data : https://stackoverflow.com/questions/41237309/one-step-ahead-out-of-sample-forecast-from-only-one-value-received-at-a-time-i

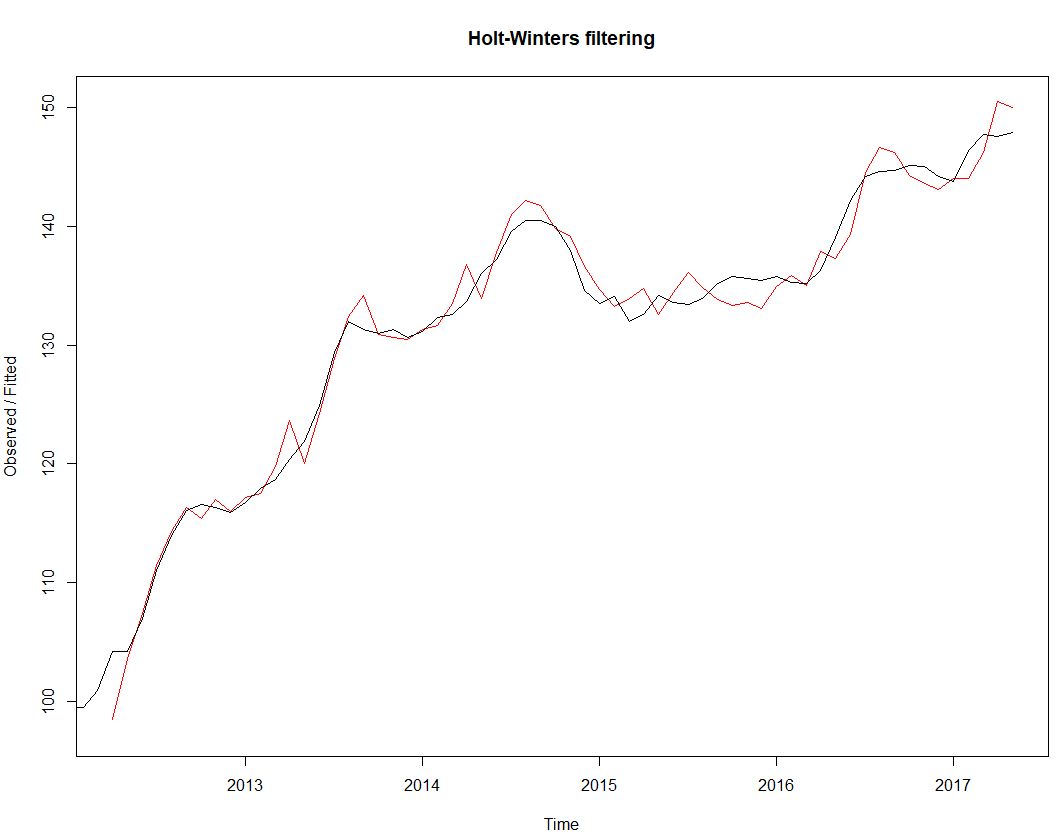


#---------------Triple ES with HoltWinters()-----------------------

#Paddy has both Trend and Seasonality so we will use HW for triple ES (alpha, beta and gamma)

hw\_triple=HoltWinters(ts)

plot(hw\_triple)



hw\_triple

# Holt-Winters exponential smoothing with trend and additive seasonal component.

#

# Call:

# HoltWinters(x = ts)

#

# Smoothing parameters:

# alpha: 0.8116069

# beta : 0.1312034

# gamma: 1

#

# Coefficients:

# [,1]

# a 148.7322634

# b 0.5899210

# s1 -0.1607628

# s2 1.3013015

# s3 2.6064806

# s4 3.3684228

# s5 2.7066913

# s6 1.0163001

# s7 -1.2737798

# s8 -2.2373703

# s9 -2.2835067

# s10 -3.1474808

# s11 -2.0298391

# s12 -0.8322634

# The value of alpha is high indicating that the estimate of the level at the current time point is based upon recent observations and few observations in the more distant past.

# The value of beta is nearly 0.00, indicating that the estimate of the slope b of the trend component is not updated over the time series, and instead is set equal to its initial value. This makes good intuitive sense, as the level changes quite a bit over the time series, but the slope b of the trend component remains roughly the same.

# In contrast, the value of gamma is high, indicating that the estimate of the seasonal component at the current time point is just based upon very recent observations.

hw\_triple$SSE #sum of squared errors

#[1] 173.1214

# by default HoltWinters() just makes forecasts for the time period covered by the original data, which is 1813-1912 for the rainfall time series. We can make forecasts for further time points by using the “forecast.HoltWinters()” function in the R “forecast” package.

#To make forecasts for future times not included in the original time series, we use the “forecast.HoltWinters()” function in the “forecast” package.

#now after ensuring the fit, we can now predict/forecast for the future

install.packages("forecast")

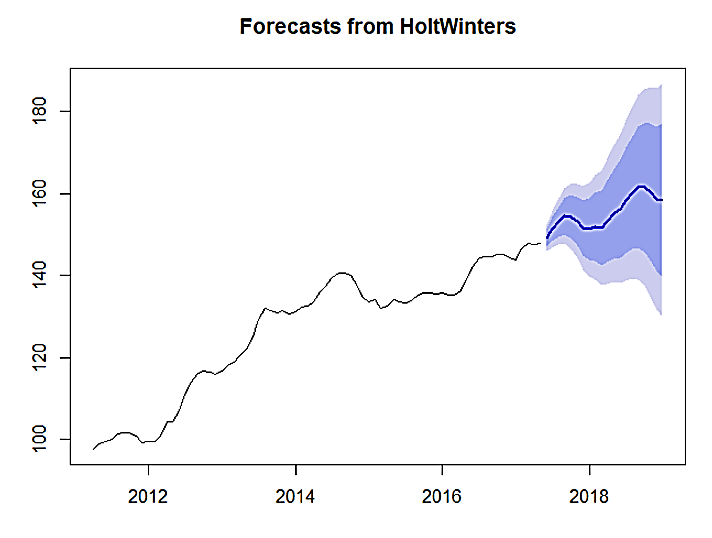
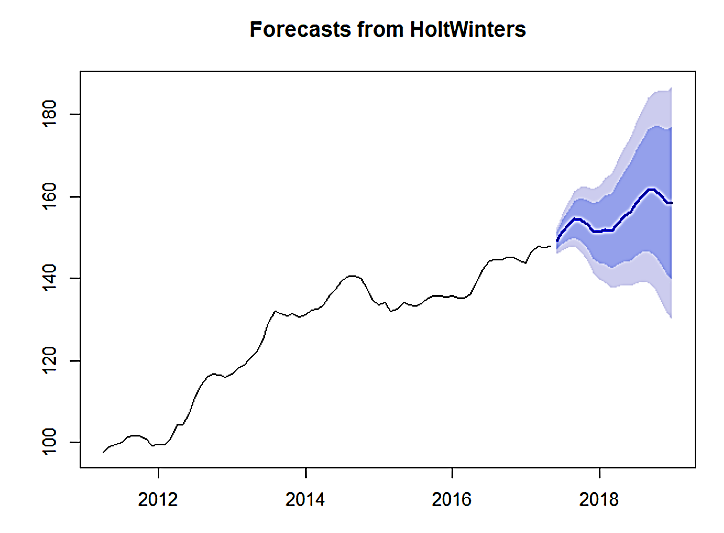
library(forecast)

#use existing fitted HW model for forecasting:

plot(forecast::forecast(hw\_triple,h=20))

#or

plot(forecast:::forecast.HoltWinters(hw\_triple,h=20)) #note 3 colons



#The forecasts are shown as a blue line, with the 80% prediction intervals as a dark shaded area, and the 95% prediction intervals as a light shaded area

#or to change colors - Orange 95% , yellow 80%

#95% confidence interval is yt±1.96σ

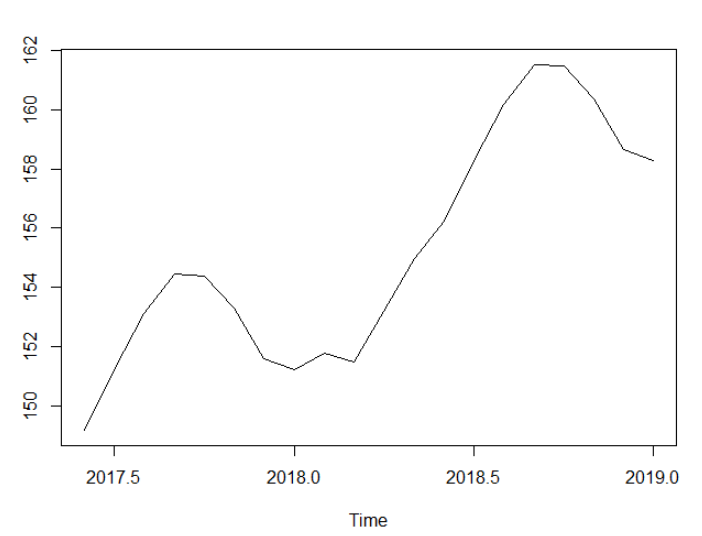
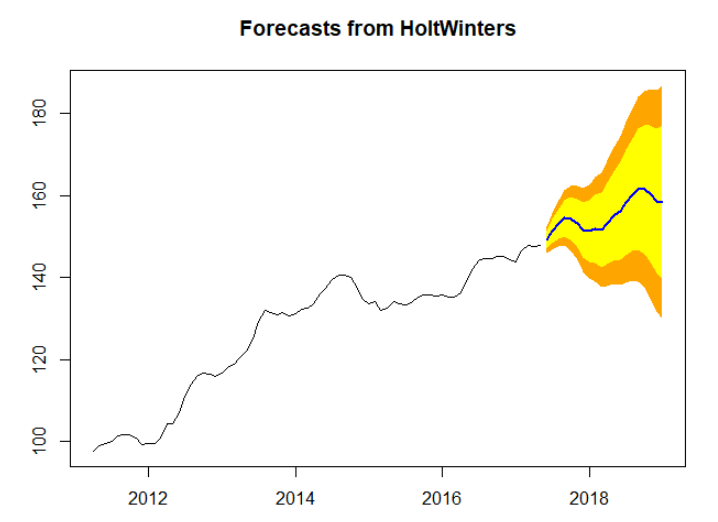
#80% confidence interval is yt±1.28σ

#Where yt is a forecast value and σ "is an estimate of the standard deviation of the forecast distribution"

plot(forecast:::forecast.HoltWinters(hw\_triple,h=20,level = c(95,80)), shadecols = c("orange","yellow")) #note 3 colons

#get the values

plot(stats:::predict.HoltWinters(hw\_triple,n.ahead = 20)) #note 3 colons



#or

myfore=forecast:::forecast.HoltWinters(hw\_triple,h=20,level = c(95,80))

myfore

# Point Forecast Lo 95 Hi 95 Lo 80 Hi 80

# Jun 2017 149.1614 145.8636 152.4592 147.0051 151.3177

# Jul 2017 151.2134 146.7365 155.6903 148.2861 154.1407

# Aug 2017 153.1085 147.4996 158.7174 149.4411 156.7759

# Sep 2017 154.4604 147.7245 161.1963 150.0560 158.8647

# Oct 2017 154.3886 146.5127 162.2644 149.2388 159.5383

……………………..

# Jun 2018 156.2405 138.0039 174.4771 144.3162 168.1648

# Jul 2018 158.2925 138.6706 177.9143 145.4624 171.1225

# Aug 2018 160.1876 139.1479 181.2273 146.4304 173.9447

# Sep 2018 161.5394 139.0499 184.0289 146.8343 176.2445

# Oct 2018 161.4676 137.4970 185.4382 145.7941 177.1411

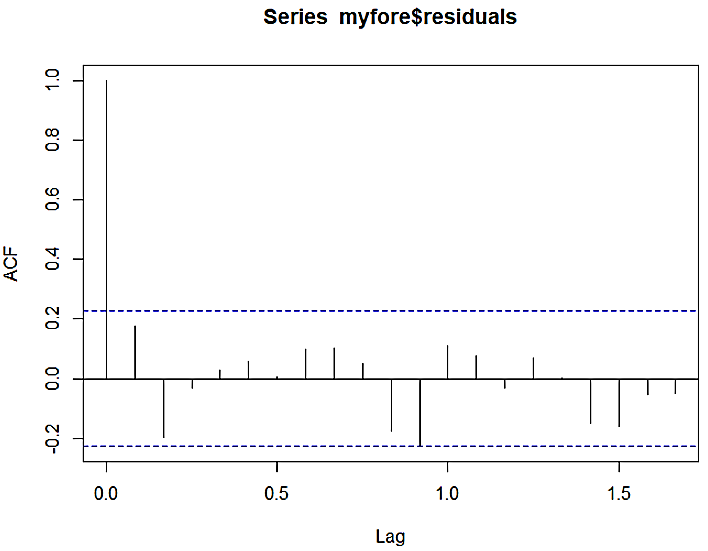
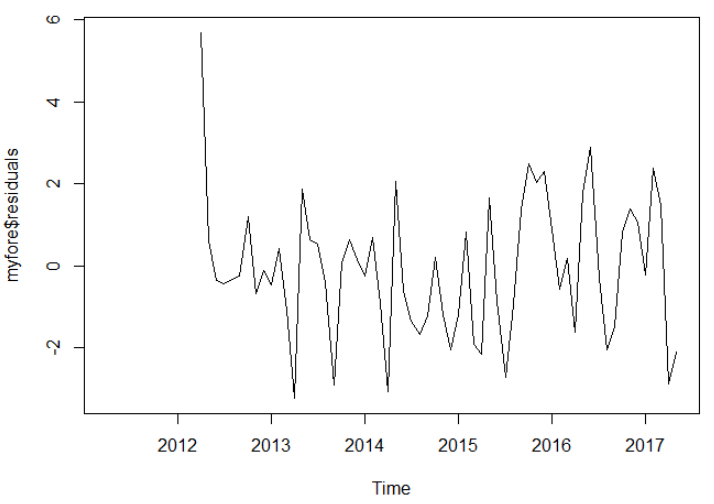
# Nov 2018 160.3671 134.8848 185.8495 143.7051 177.0291

# Dec 2018 158.6670 131.6428 185.6912 140.9968 176.3371

# Jan 2019 158.2933 129.6978 186.8889 139.5957 176.9909

### --------------------- Validations --------------------------------

plot(myfore$residuals)



acf(myfore$residuals, lag.max =20,na.action = na.pass) #without na.action it will fail

#plot looks good, notice the lag, do a lag\*12 to get actual lag number.

Box.test(myfore$residuals, lag=20, type="Ljung-Box") #seems good

#If the p value is greater than 0.05 then the residuals are independent which we want for the model to be correct.

# Box-Ljung test

# data: myfore$residuals

# X-squared = 19.385, df = 20, p-value = 0.4969

#------ES using automated ETS method ---------------

#An alternative to estimating the parameters by minimising the sum of squared errors is to maximise the “likelihood”. The likelihood is the probability of the data arising from the specified model. With the ets() function, the default estimation method is maximum likelihood rather than minimum sum of squares.

myets=forecast::ets(ts)

# summary(myets)

# ETS(M,Ad,N)

# Call:

# forecast::ets(y = ts)

# Smoothing parameters:

# alpha = 0.9999

# beta = 0.8123

# phi = 0.8

# Initial states:

# l = 96.1838

# b = 1.2645

# sigma: 0.0102

# AIC AICc BIC

# 362.7057 363.9594 376.5301

# Training set error measures:

# ME RMSE MAE MPE MAPE MASE ACF1

# Training set 0.1517182 1.221302 0.9751211 0.129346 0.7799746 0.09961692 0.03036488

round(forecast::accuracy(myets),2)

# ME RMSE MAE MPE MAPE MASE ACF1

# Training set 0.15 1.22 0.98 0.13 0.78 0.1 0.03

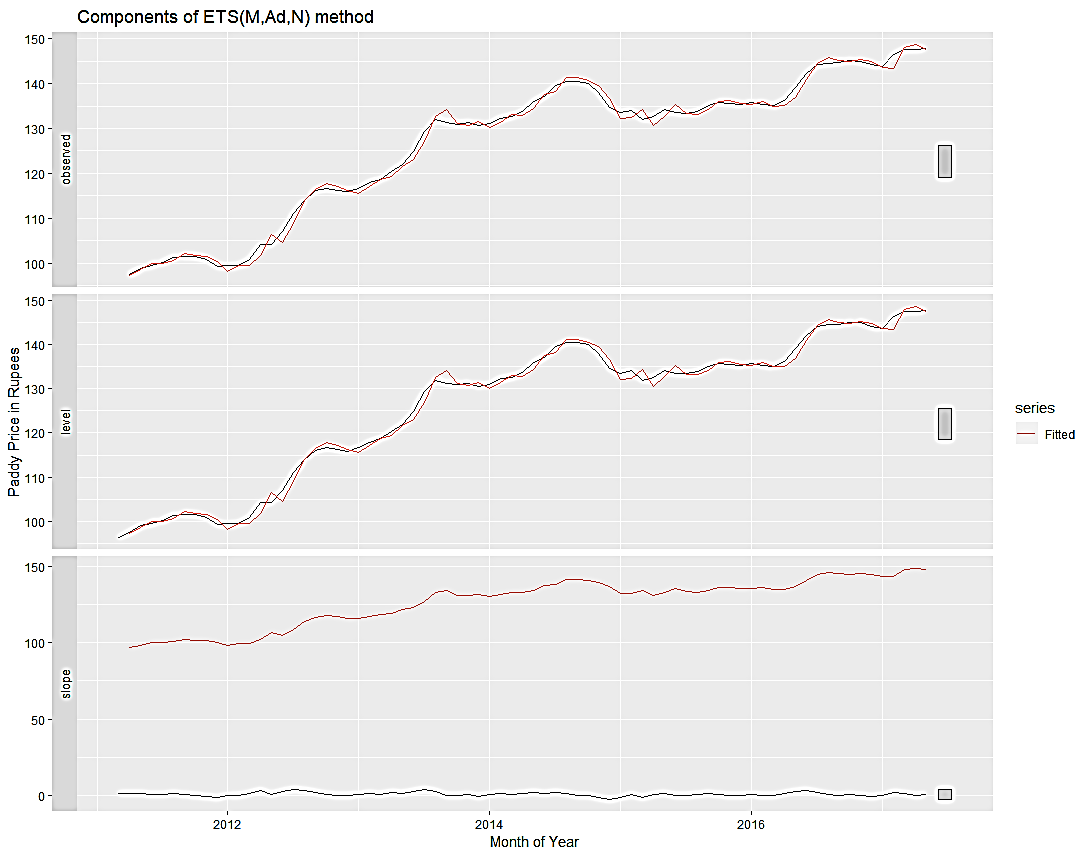
#no forecasts will be given . ets() is used only to get a good model parameters.

p=forecast::autoplot(myets) + forecast::autolayer(myets$fitted, series="Fitted")

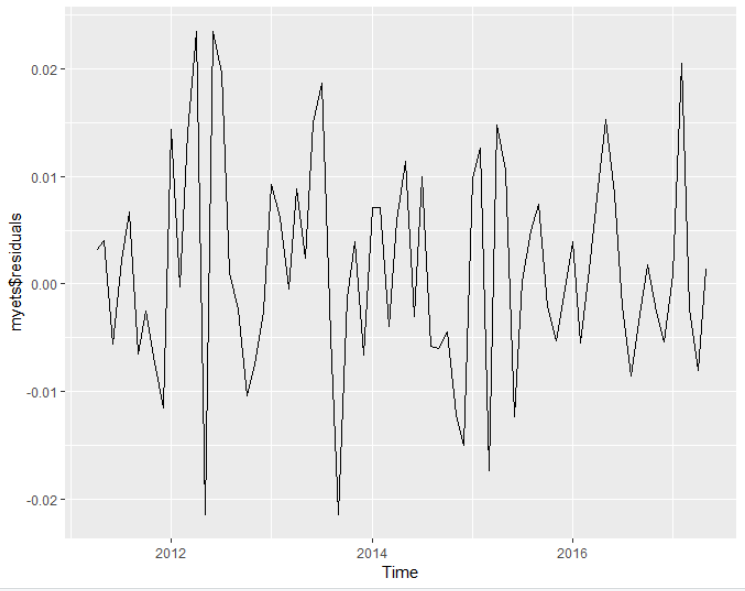
p$labels$x="Month of Year"

p$labels$y="Paddy Price in Rupees"

p #to run the plot again.



forecast::autoplot(myets$residuals)



myforeets=forecast:::forecast(myets, h=20)

plot(myforeets$residuals) #seems fine , same as above plot

Box.test(myforeets$residuals, lag=20, type="Ljung-Box") #seems fine , just escaped

# Box-Ljung test

#

# data: myforeets$residuals

# X-squared = 29.798, df = 20, p-value = 0.07319

#------------------------ ARIMA ---------------------------

# Exponential smoothing methods are useful for making forecasts, and make no assumptions about the correlations between successive values of the time series. However, if you want to make prediction intervals for forecasts made using exponential smoothing methods, the prediction intervals require that the forecast errors are uncorrelated and are normally distributed with mean zero and constant variance.

#

# While exponential smoothing methods do not make any assumptions about correlations between successive values of the time series, in some cases you can make a better predictive model by taking correlations in the data into account. Autoregressive Integrated Moving Average (ARIMA) models include an explicit statistical model for the irregular component of a time series, that allows for non-zero autocorrelations in the irregular component.

#While exponential smoothing models are based on a description of the trend and seasonality in the data, ARIMA models aim to describe the autocorrelations in the data.

install.packages("aTSA")

library(aTSA)

adf.test(ts) #here k=7 by default.

# Augmented Dickey-Fuller Test

# alternative: stationary

#

# Type 1: no drift no trend

# lag ADF p.value

# [1,] 0 3.90 0.990

# [2,] 1 1.71 0.977

# [3,] 2 1.96 0.986

# [4,] 3 1.58 0.970

# Type 2: with drift no trend

# lag ADF p.value

# [1,] 0 -1.52 0.513

# [2,] 1 -1.24 0.610

# [3,] 2 -1.35 0.571

# [4,] 3 -1.31 0.585

# Type 3: with drift and trend

# lag ADF p.value

# [1,] 0 -1.11 0.915

# [2,] 1 -1.95 0.591

# [3,] 2 -1.76 0.667

# [4,] 3 -2.01 0.563

# ----

# Note: in fact, p.value = 0.01 means p.value <= 0.01

#H0= not stationary (opposite in ADF)

#Ha= stationary

#result means not stationary

#read this: https://stats.stackexchange.com/questions/57807/confusion-with-augmented-dickey-fuller-test

> adf.test(ts, k=12)

# Augmented Dickey-Fuller Test

# data: ts

# Dickey-Fuller = -2.9141, Lag order = 12, p-value = 0.2028

# alternative hypothesis: stationary

> adf.test(diff(ts,differences = 1), k=12)

# Augmented Dickey-Fuller Test

# data: diff(ts, differences = 1)

# Dickey-Fuller = -2.0901, Lag order = 12, p-value = 0.5387

# alternative hypothesis: stationary

> adf.test(diff(ts,differences = 1), k=7)

# Augmented Dickey-Fuller Test

# data: diff(ts, differences = 1)

# Dickey-Fuller = -3.5275, Lag order = 7, p-value = 0.04575

# alternative hypothesis: stationary

> adf.test(ts, k=7)

# Augmented Dickey-Fuller Test

# data: ts

# Dickey-Fuller = -1.5066, Lag order = 7, p-value = 0.7766

# alternative hypothesis: stationary

#

adf.test(diff(ts,differences = 1))

# Augmented Dickey-Fuller Test

# alternative: stationary

# Type 1: no drift no trend

# lag ADF p.value

# [1,] 0 -4.04 0.01

# [2,] 1 -3.86 0.01

# [3,] 2 -2.89 0.01

# [4,] 3 -3.58 0.01

# Type 2: with drift no trend

# lag ADF p.value

# [1,] 0 -4.53 0.0100

# [2,] 1 -4.52 0.0100

# [3,] 2 -3.43 0.0152

# [4,] 3 -4.39 0.0100

# Type 3: with drift and trend

# lag ADF p.value

# [1,] 0 -4.54 0.0100

# [2,] 1 -4.56 0.0100

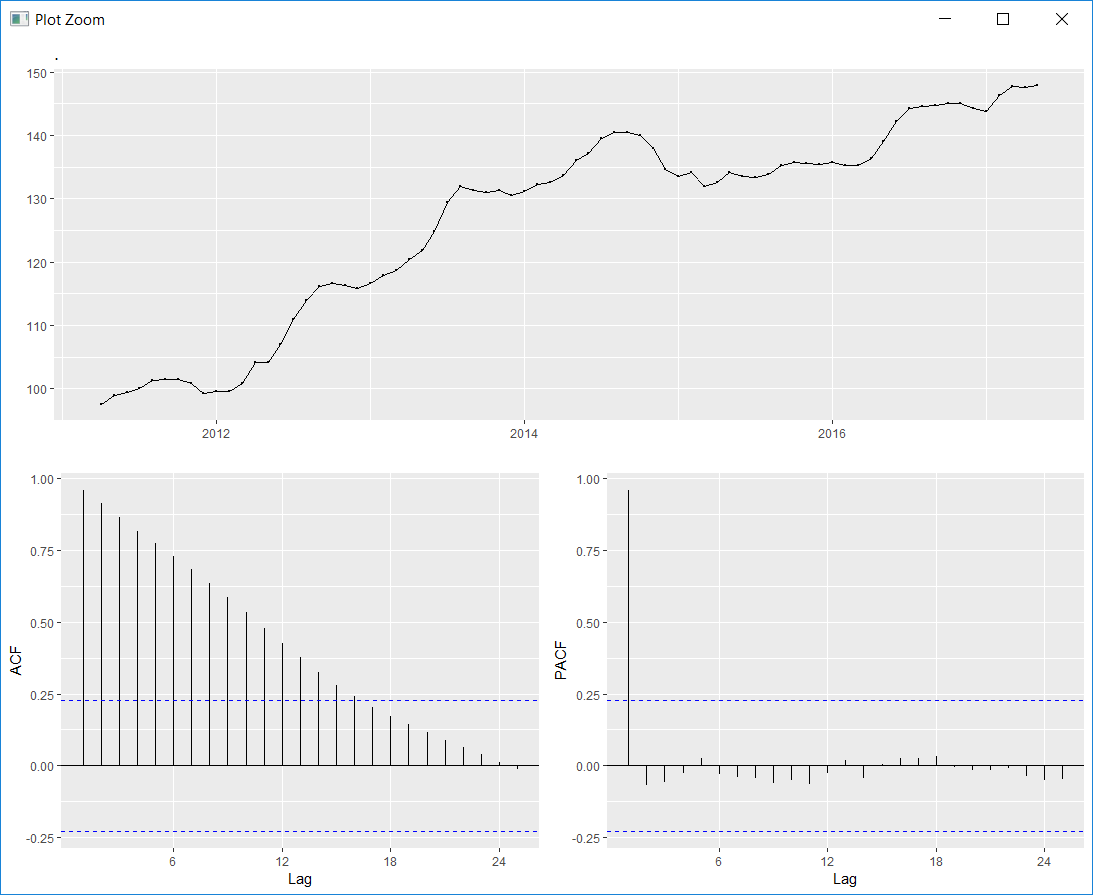
# [3,] 2 -3.47 0.0514

# [4,] 3 -4.49 0.0100

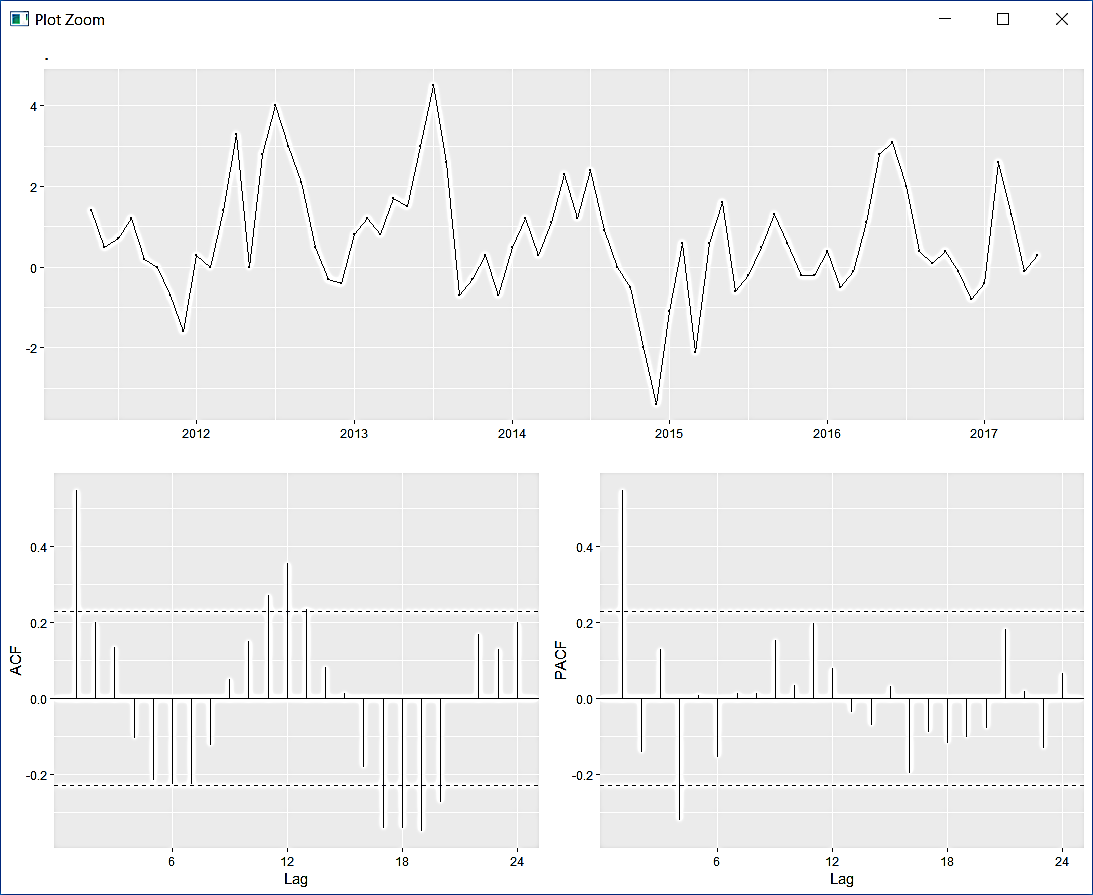
# Note: in fact, p.value = 0.01 means p.value <= 0.01

#result means reject H0 (ts is not stationary), so means ts is stationary now, let us check it now

ts%>% forecast::ggtsdisplay()

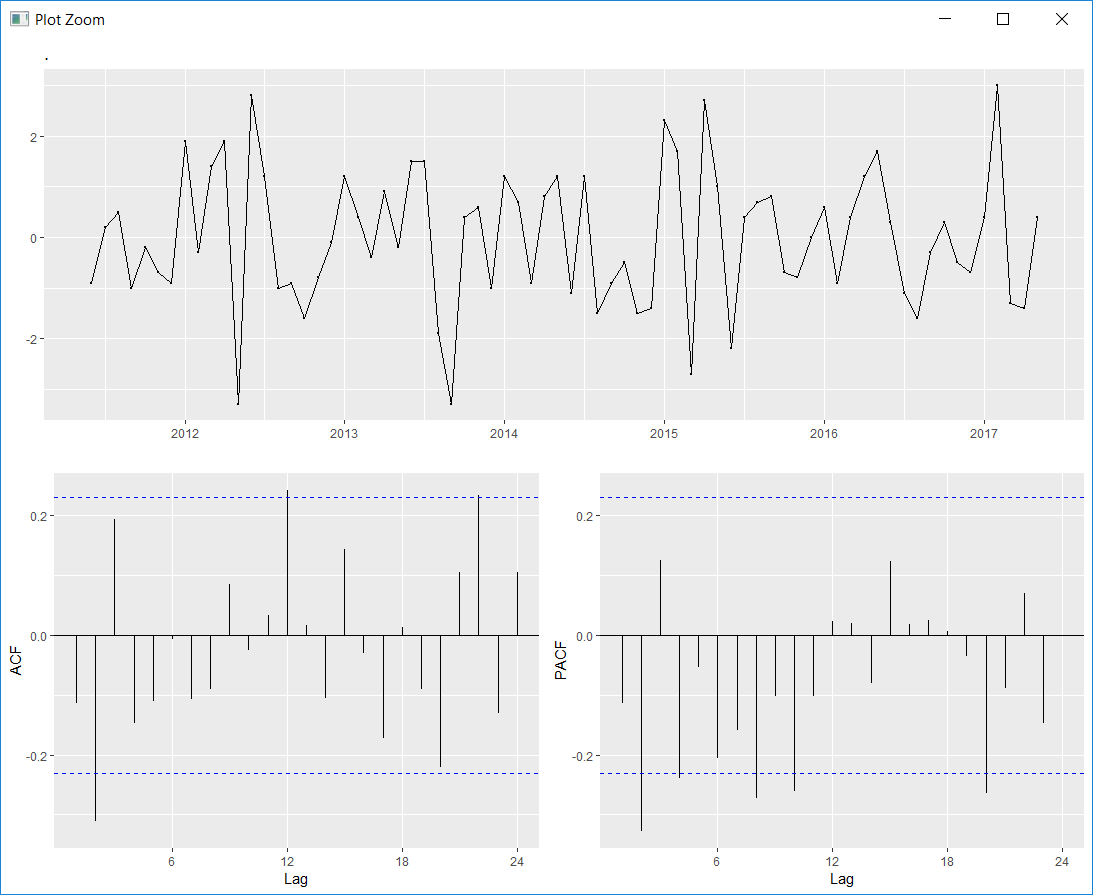


ts %>% diff(differences=1) %>% forecast::ggtsdisplay()

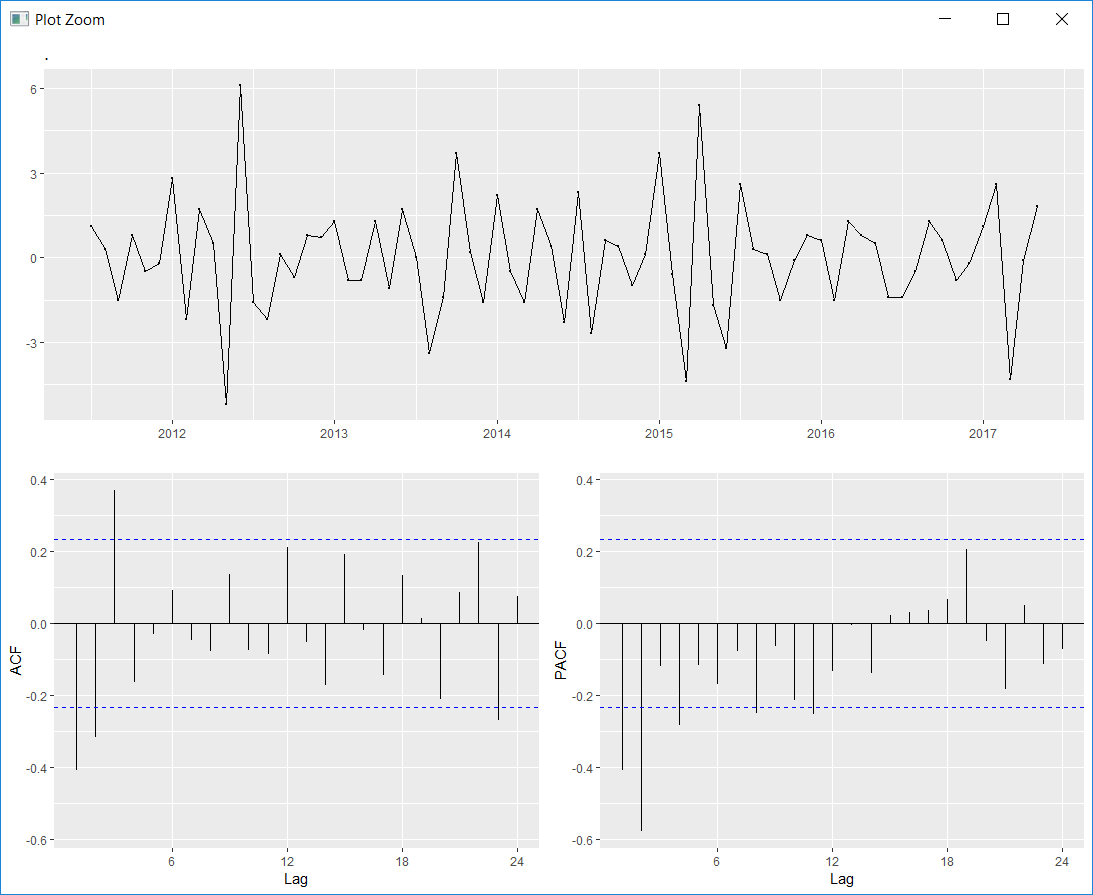


#better now

ts %>% diff(differences=2) %>% forecast::ggtsdisplay()



ts %>% diff(differences=3) %>% forecast::ggtsdisplay()



#Note: ADF only tests for unit root stationary, this could be trend stationary. So you should use the KPSS test, see stats.stackexchange.com/questions/30569/… In general, there is a difference, between DS (difference-stationary) and TS (trend stationary) models. KPSS is the better test to distinguish between those models

#use KPSS Unit Root test also to check if d>0 is needed - Kwiatkowski-Phillips-Schmidt-Shin (KPSS) test

#unit root = also called a unit root process or a difference stationary process

#here Trend stationary means "stationary around the trend", i.e. the trend needn't be stationary, but the de-trended data is. Level stationary means that the data is like white noise.

#The null hypothesis for the test is that the data is stationary.

#The alternate hypothesis for the test is that the data is not stationary.

#https://stats.stackexchange.com/questions/30569/what-is-the-difference-between-a-stationary-test-and-a-unit-root-test

# The Concepts and examples of Unit-root tests and stationarity tests

# ----------------------------------------------------------------------------------------

# Concept of Unit-root tests:

# Null hypothesis: Unit-root

# Alternative hypothesis: Process has root outside the unit circle, which is usually equivalent to stationarity or trend stationarity

#

# Concept of Stationarity tests

#

# Null hypothesis: (Trend) Stationarity

#

# Alternative hypothesis: There is a unit root.

#

# There are many different Unit-root tests and many Stationarity tests.

#

# Some Unit root tests:

#

# Dickey-Fuller test

# Augmented Dickey Fuller test

# Phillipps-Perron test

# Zivot-Andrews test

# ADF-GLS test

# The most simple test is the DF-test. The ADF and the PP test are similar to the Dickey-Fuller test, but they correct for lags. The ADF does so by including them the PP test does so by adjusting the test statistics.

#

# Some Stationarity tests:

#

# KPSS

#

# Leybourne-McCabe

#

# In practice KPSS test is used far more often. The main difference of both tests is that KPSS is a non-parametric test and Leybourne-McCabe is a parametric test.

#

# How unit-root test and stationarity-test complement each other

# If you have a time series data set how it usually appears in econometric time series I propose you should apply both a Unit root test: (Augmented) Dickey Fuller or Phillips-Perron depending on the structure of the underlying data and a KPSS test.

#

# Case 1 Unit root test: you can’t reject H0; KPSS test: reject H0. Both imply that series has unit root.

# Case 2 Unit root test: Reject H0. KPSS test: don't reject H0. Both imply that series is stationary. – our case

# Case 3 If we can’t reject both test: data give not enough observations.

# Case 4 Reject unit root, reject stationarity: both hypotheses are component hypotheses – heteroskedasticity in a series may make a big difference; if there is structural break it will affect inference.

#

# Power problem: if there is small random walk component (small variance σ2μ), we can’t reject unit root and can’t reject stationarity.

#

# Economics: if the series is highly persistent we can’t reject H0 (unit root) – highly persistent may be even without unit root, but it also means we shouldn’t treat/take data in levels. Whether a time series is "highly persistent" can be measured with the p-value of a unit-root test. For a more detailed discussion what "persistence" means in time-series see: Persistence in time series

#

# General rule about statistical testing You cannot proove a null hypothesis, you can only affirm it. However, if you reject a null hypothesis then you can be very sure that the null hypothesis is really not true. Thus alternative hypothesis is always a stronger hypothesis than the null hypothesis.

#

install.packages("tseries")

library(tseries)

kpss.test(ts)#reject null hypothesis

# KPSS Test for Level Stationarity

#

# data: ts

# KPSS Level = 3.3243, Truncation lag parameter = 1, p-value = 0.01

kpss.test(diff(ts,differences = 1)) #accept null hypothesis

# KPSS Test for Level Stationarity

#

# data: diff(ts, differences = 1)

# KPSS Level = 0.21802, Truncation lag parameter = 1, p-value = 0.1

kpss.test(diff(ts,differences = 1,null=c("Level", "Trend")))

# KPSS Test for Level Stationarity

#

# data: diff(ts, differences = 1, null = c("Level", "Trend"))

# KPSS Level = 0.21802, Truncation lag parameter = 1, p-value = 0.1

#

kpss.test(diff(ts,differences = 1,null="Trend"))

# KPSS Test for Level Stationarity

#

# data: diff(ts, differences = 1, null = "Trend")

# KPSS Level = 0.21802, Truncation lag parameter = 1, p-value = 0.1

acf(diffts, lag.max = 20,na.action = na.pass, plot=FALSE) #to get the values

#

# Autocorrelations of series ‘diffts’, by lag

#

# 0.0000 0.0833 0.1667 0.2500 0.3333 0.4167 0.5000 0.5833 0.6667 0.7500 0.8333 0.9167 1.0000 1.0833 1.1667

# 1.000 -0.407 -0.315 0.370 -0.164 -0.030 0.092 -0.046 -0.077 0.136 -0.075 -0.086 0.213 -0.051 -0.173

# 1.2500 1.3333 1.4167 1.5000 1.5833 1.6667

# 0.192 -0.018 -0.144 0.134 0.016 -0.211

#d= 1

#p = 3?

#q= 2 ?

#https://otexts.org/fpp2/arima-r.html

forecast:::auto.arima(ts,stepwise=FALSE , approximation=FALSE,seasonal=FALSE)

#stepwise=FALSE and approximation=FALSE to make it work harder

#seasonal=FALSE for not looking for seasonal arima models

# Series: ts

# ARIMA(0,1,1) with drift

#

# Coefficients:

# ma1 drift

# 0.7540 0.6930

# s.e. 0.0956 0.2295

#

# sigma^2 estimated as 1.3: log likelihood=-112.55

# AIC=231.1 AICc=231.44 BIC=237.97

#If we use the “aic” criterion, which penalises the number of parameters , but it remains the same in our case.

fit=forecast:::auto.arima(ts,ic = "aic", seasonal = TRUE)

# Series: ts

# ARIMA(0,1,1)(0,1,1)[12]

#

# Coefficients:

# ma1 sma1

# 0.6212 -0.8816

# s.e. 0.1234 0.5684

#

# sigma^2 estimated as 1.148: log likelihood=-97.49

# AIC=200.99 AICc=201.41 BIC=207.32

#If we combine differencing with autoregression and a moving average model, we obtain a non-seasonal ARIMA model.

arima(ts, order=c(0,1,1))

# Call:

# arima(x = ts, order = c(0, 1, 1))

#

# Coefficients:

# ma1

# 0.7925

# s.e. 0.0838

#

# sigma^2 estimated as 1.415: log likelihood = -116.75, aic = 237.5

library(dplyr)

#-------------------SEASONAL ARIMA ---------------------

#https://otexts.org/fpp2/seasonal-arima.html

fit=forecast:::auto.arima(ts,ic = "aic")

# Series: ts

# ARIMA(0,1,1)(0,1,1)[12]

#

# Coefficients:

# ma1 sma1

# 0.6212 -0.8816

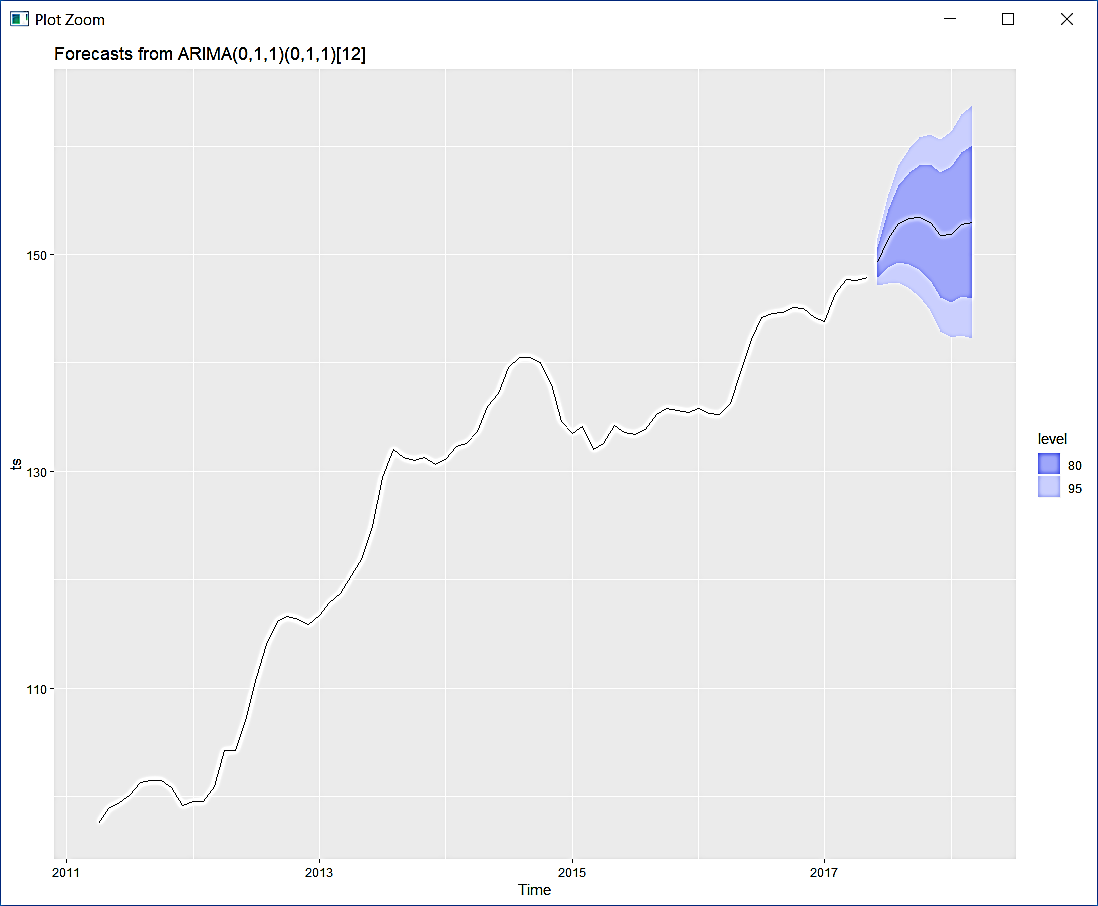
# s.e. 0.1234 0.5684

#

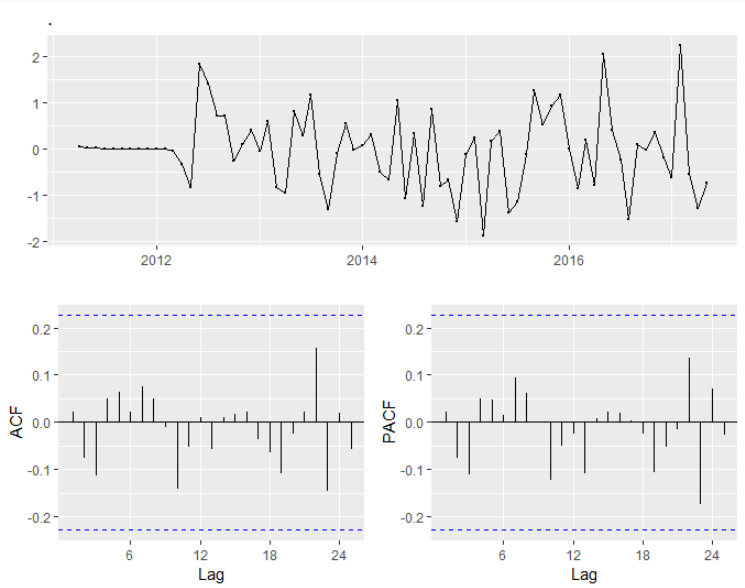
# sigma^2 estimated as 1.148: log likelihood=-97.49

# AIC=200.99 AICc=201.41 BIC=207.32

fit %>% forecast::forecast(h=10) %>% forecast::autoplot(include=80)



ts %>% forecast::Arima(order=c(2,1,2), seasonal=c(2,1,2)) %>% residuals() %>% forecast::ggtsdisplay()



forecast::Arima(ts,order=c(2,1,2), seasonal=c(2,1,2))

# Series: ts

# ARIMA(2,1,2)(2,1,2)[12]

# Coefficients:

# ar1 ar2 ma1 ma2 sar1 sar2 sma1 sma2

# 0.4560 0.4053 0.0666 -0.6640 0.4198 -0.3011 -1.4517 0.5218

# s.e. 0.2475 0.1941 0.2134 0.1668 0.9689 0.3366 3.2013 1.6358

# sigma^2 estimated as 0.9392: log likelihood=-94.53

# AIC=207.07 AICc=210.6 BIC=226.06

forecast::Arima(ts,order=c(2,1,2), seasonal=c(1,1,2))

# Series: ts

# ARIMA(2,1,2)(1,1,2)[12]

# Coefficients:

# ar1 ar2 ma1 ma2 sar1 sma1 sma2

# 0.4837 0.3603 0.0694 -0.6300 -0.4027 -0.4874 -0.5121

# s.e. 0.2514 0.1894 0.2181 0.1491 0.6726 0.6991 0.6482

# sigma^2 estimated as 1.069: log likelihood=-95.22

# AIC=206.44 AICc=209.21 BIC=223.33

forecast::Arima(ts,order=c(2,1,2), seasonal=c(1,1,1))

# Series: ts

# ARIMA(2,1,2)(1,1,1)[12]

#

# Coefficients:

# ar1 ar2 ma1 ma2 sar1 sma1

# 0.4993 0.3521 0.0604 -0.6292 0.0779 -0.9997

# s.e. 0.2425 0.1868 0.2086 0.1434 0.1644 0.3122

#

# sigma^2 estimated as 1.06: log likelihood=-95.42

# AIC=204.84 AICc=206.95 BIC=219.61

forecast::Arima(ts,order=c(1,1,1), seasonal=c(1,1,1))

# Series: ts

# ARIMA(1,1,1)(1,1,1)[12]

# Coefficients:

# ar1 ma1 sar1 sma1

# -0.1766 0.7446 0.1133 -0.9999

# s.e. 0.2238 0.1720 0.1638 0.3278

# sigma^2 estimated as 1.084: log likelihood=-96.89

# AIC=203.79 AICc=204.88 BIC=214.34

forecast::Arima(ts,order=c(1,1,1), seasonal=c(1,1,0))

# Series: ts

# ARIMA(1,1,1)(1,1,0)[12]

# Coefficients:

# ar1 ma1 sar1

# -0.0324 0.6848 -0.3725

# s.e. 0.2287 0.1888 0.1325

# sigma^2 estimated as 1.657: log likelihood=-101.57

# AIC=211.15 AICc=211.86 BIC=219.59

forecast::Arima(ts,order=c(1,1,1), seasonal=c(0,1,1))

# Series: ts

# ARIMA(1,1,1)(0,1,1)[12]

# Coefficients:

# ar1 ma1 sma1

# -0.1872 0.7534 -0.9994

# s.e. 0.2088 0.1553 0.6962

# sigma^2 estimated as 1.036: log likelihood=-97.15

# AIC=202.29 AICc=203 BIC=210.73

forecast::Arima(ts,order=c(0,1,1), seasonal=c(0,1,1))

# Series: ts

# ARIMA(0,1,1)(0,1,1)[12]

# Coefficients:

# ma1 sma1

# 0.6212 -0.8816

# s.e. 0.1234 0.5684

# sigma^2 estimated as 1.148: log likelihood=-97.49

# AIC=200.99 AICc=201.41 BIC=207.32

forecast::Arima(ts,order=c(0,1,1), seasonal=c(0,1,0))

# Series: ts

# ARIMA(0,1,1)(0,1,0)[12]

# Coefficients:

# ma1

# 0.6196

# s.e. 0.1127

# sigma^2 estimated as 1.847: log likelihood=-104.98

# AIC=213.96 AICc=214.17 BIC=218.18

forecast::Arima(ts,order=c(0,1,0), seasonal=c(0,1,1))

# Series: ts

# ARIMA(0,1,0)(0,1,1)[12]

# Coefficients:

# sma1

# -0.9999

# s.e. 0.3216

# sigma^2 estimated as 1.342: log likelihood=-105.81

# AIC=215.62 AICc=215.83 BIC=219.84

forecast::Arima(ts,order=c(0,1,1), seasonal=c(0,1,1))

# Series: ts

# ARIMA(0,1,1)(0,1,1)[12]

# Coefficients:

# ma1 sma1

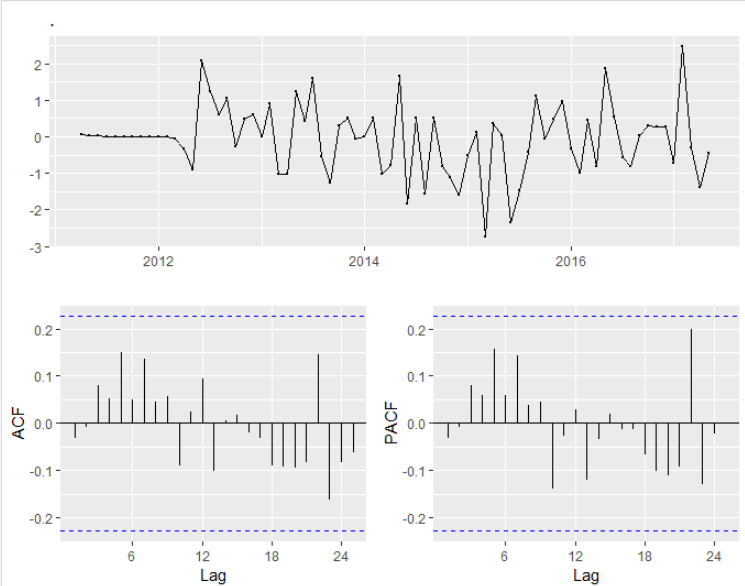
# 0.6212 -0.8816

# s.e. 0.1234 0.5684

# sigma^2 estimated as 1.148: log likelihood=-97.49

# AIC=200.99 AICc=201.41 BIC=207.32

ts %>% forecast::Arima(order=c(0,1,1), seasonal=c(0,1,1)) %>% residuals() %>% forecast::ggtsdisplay()



fit=forecast::Arima(ts,order=c(0,1,1),seasonal=c(0,1,1))

# Point Forecast Lo 80 Hi 80 Lo 95 Hi 95

# Jun 2017 149.2747 147.8573 150.6921 147.1070 151.4424

# Jul 2017 151.4623 148.7646 154.1601 147.3365 155.5882

# Aug 2017 152.8382 149.2961 156.3804 147.4210 158.2555

# Sep 2017 153.3309 149.1101 157.5518 146.8757 159.7862

# Oct 2017 153.4593 148.6546 158.2640 146.1112 160.8074

# Nov 2017 152.9663 147.6415 158.2912 144.8227 161.1100

# Dec 2017 151.7949 145.9964 157.5934 142.9268 160.6630

# Jan 2018 151.8556 145.6192 158.0919 142.3179 161.3932

# Feb 2018 152.7381 146.0928 159.3835 142.5749 162.9013

# Mar 2018 153.0017 145.9711 160.0324 142.2493 163.7542

# Apr 2018 154.2333 146.8370 161.6295 142.9216 165.5449

# May 2018 155.6537 147.9148 163.3925 143.8181 167.4893

# Jun 2018 157.2734 149.1342 165.4127 144.8255 169.7213

# Jul 2018 159.4611 150.8946 168.0275 146.3598 172.5623

# Aug 2018 160.8370 151.8636 169.8103 147.1134 174.5605

# Sep 2018 161.3297 151.9671 170.6922 147.0109 175.6485

# Oct 2018 161.4580 151.7218 171.1943 146.5678 176.3483

# Nov 2018 160.9651 150.8690 171.0612 145.5244 176.4057

# Dec 2018 159.7936 149.3501 170.2372 143.8216 175.7657

# Jan 2019 159.8543 149.0745 170.6341 143.3680 176.3406

# Feb 2019 160.7369 149.6310 171.8428 143.7519 177.7219

# Mar 2019 161.0005 149.5778 172.4232 143.5310 178.4700

# Apr 2019 162.2320 150.5034 173.9606 144.2947 180.1693

# May 2019 163.6524 151.6339 175.6709 145.2717 182.0331

fit %>% forecast::forecast(h=10) %>% forecast::autoplot(include=80)

forecast::forecast(fit)

