M8: Forecasting in R: Simple Averaging Models & ARIMA

Luana Lima

03/12/2021

Setting R code chunk options

First R code chunk is used for setting the options for all R code chunks. The choice echo=TRUE means both code and output will appear on report, include = FALSE neither code nor output is printed.

Loading packages and initializing

Second R code chunk is for loading packages. By setting message = FALSE, the code will appear but not the output.

```
library(lubridate)
library(ggplot2)
library(forecast)
library(Kendall)
library(tseries)
library(outliers)
library(tidyverse)

#install.packages("smooth")
library(smooth)
```

Importing data

For this module we will continue to work with the electricity retail price in US dataset from the U.S. Energy Information Administration. You may download [here][https://www.eia.gov/electricity/data/browser/#/topic/7?agg=2,0,1&geo=g&freq=M%2013:41:41%20GMT-0500%20(EST)]. But this week we will work with the all.sectors column instead of residential price.

```
#Importing time series data from text file#
electricity_price <- read.csv(file="./Data/Average_retail_price_of_electricity_United_States_monthly.cs

#Inspect data
head(electricity_price)
```

```
##      Month all.sectors.cents.per.kilowatthour
## 1 Nov 2020                              10.45
## 2 Oct 2020                              10.64
## 3 Sep 2020                              11.07
## 4 Aug 2020                              11.11
## 5 Jul 2020                              11.14
## 6 Jun 2020                              10.96
##   residential.cents.per.kilowatthour commercial.cents.per.kilowatthour
## 1                              13.35                             10.59
## 2                              13.60                             10.73
## 3                              13.55                              11.07
## 4                              13.31                              10.95
## 5                              13.26                              10.90
## 6                              13.28                              10.95
##   industrial.cents.per.kilowatthour
## 1                              6.48
## 2                              6.72
## 3                              7.01
## 4                              7.09
## 5                              7.17
## 6                              6.94
```

```
nvar <- ncol(electricity_price) - 1
nobs <- nrow(electricity_price)

#Preparing the data - create date object and rename columns
electricity_price_processed <-
  electricity_price %>%
  mutate( Month = my(Month) ) %>%
  rename( All.sectors = all.sectors.cents.per.kilowatthour ) %>%
  rename( Residential = residential.cents.per.kilowatthour ) %>%
  rename( Commercial = commercial.cents.per.kilowatthour ) %>%
  rename( Industrial = industrial.cents.per.kilowatthour ) %>%
  arrange( Month )

head(electricity_price_processed)
```

```
##        Month All.sectors Residential Commercial Industrial
## 1 2001-01-01        6.75        7.73       7.25       4.73
## 2 2001-02-01        6.87        8.04       7.51       4.80
## 3 2001-03-01        7.01        8.32       7.70       4.86
## 4 2001-04-01        7.02        8.46       7.73       4.87
## 5 2001-05-01        7.17        8.83       7.77       5.00
## 6 2001-06-01        7.58        9.07       8.13       5.23
```

```
summary(electricity_price_processed)
```

```
##      Month             All.sectors      Residential      Commercial
##  Min.   :2001-01-01   Min.   : 6.750   Min.   : 7.73   Min.   : 7.250
##  1st Qu.:2005-12-16   1st Qu.: 8.520   1st Qu.: 9.82   1st Qu.: 9.070
##  Median :2010-12-01   Median : 9.720   Median :11.77   Median :10.080
##  Mean   :2010-11-30   Mean   : 9.381   Mean   :11.23   Mean   : 9.746
##  3rd Qu.:2015-11-16   3rd Qu.:10.305   3rd Qu.:12.64   3rd Qu.:10.540
##  Max.   :2020-11-01   Max.   :11.140   Max.   :13.60   Max.   :11.170
##    Industrial
##  Min.   :4.71
##  1st Qu.:5.99
##  Median :6.58
##  Mean   :6.37
##  3rd Qu.:6.89
##  Max.   :7.72
```

```
#No NAs so we don't need to worry about missing values
```

Transforming data into time series object

Many of the functions we will use require a time series object. You can transform your data in a time series using the function *ts()*.

```
ts_electricity_price <- ts(electricity_price_processed[,2:(nvar+1)],
                           start=c(year(electricity_price_processed$Month[1]),month(electricity_price_p
                           frequency=12)
#note that we are only transforming columns with electricity price, not the date columns
head(ts_electricity_price,15)
```

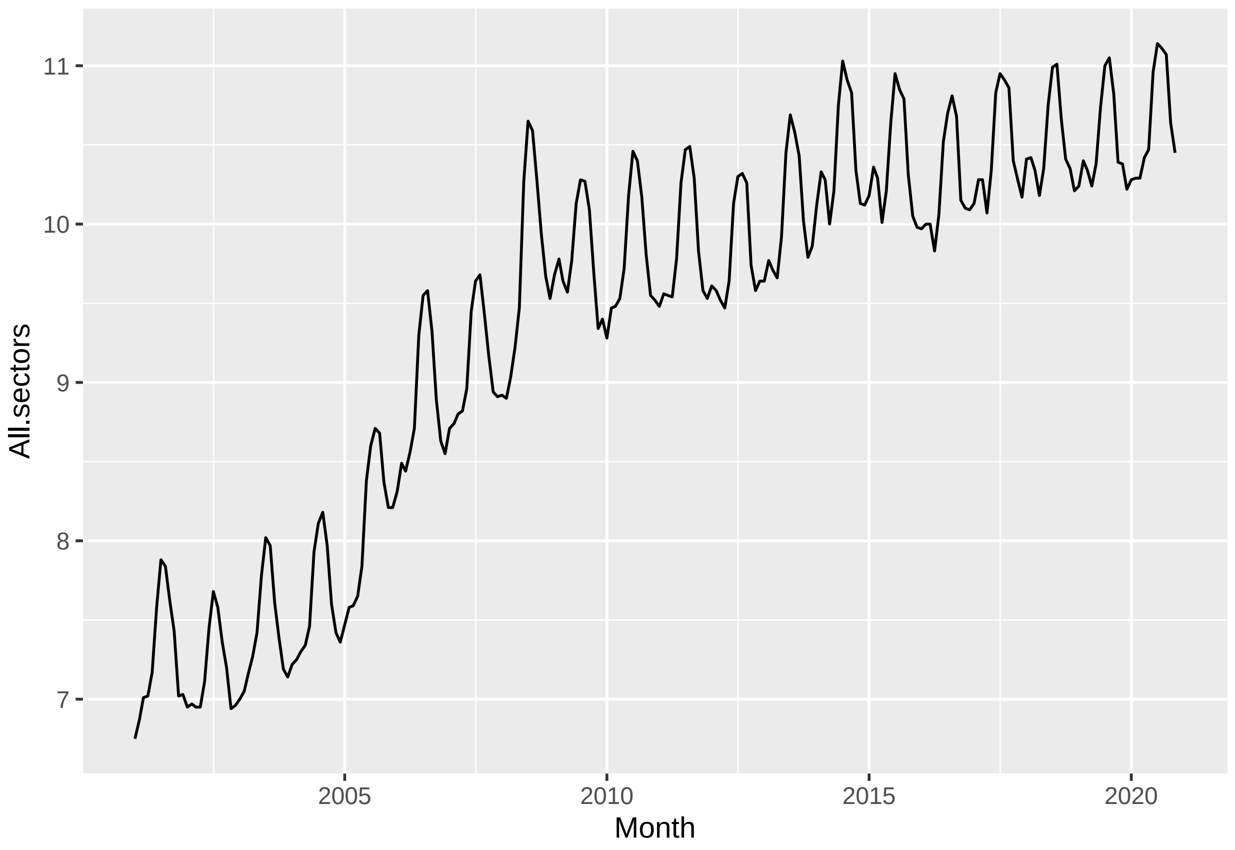
```
##          All.sectors Residential Commercial Industrial
## Jan 2001        6.75        7.73       7.25       4.73
## Feb 2001        6.87        8.04       7.51       4.80
## Mar 2001        7.01        8.32       7.70       4.86
## Apr 2001        7.02        8.46       7.73       4.87
## May 2001        7.17        8.83       7.77       5.00
## Jun 2001        7.58        9.07       8.13       5.23
## Jul 2001        7.88        9.03       8.41       5.57
## Aug 2001        7.84        9.01       8.35       5.50
## Sep 2001        7.62        8.92       8.22       5.31
## Oct 2001        7.43        8.84       8.27       5.07
## Nov 2001        7.02        8.47       7.73       4.78
## Dec 2001        7.03        8.29       7.66       4.78
## Jan 2002        6.95        8.07       7.49       4.73
## Feb 2002        6.97        8.19       7.68       4.76
## Mar 2002        6.95        8.17       7.72       4.73
```

```
tail(ts_electricity_price,15)
```

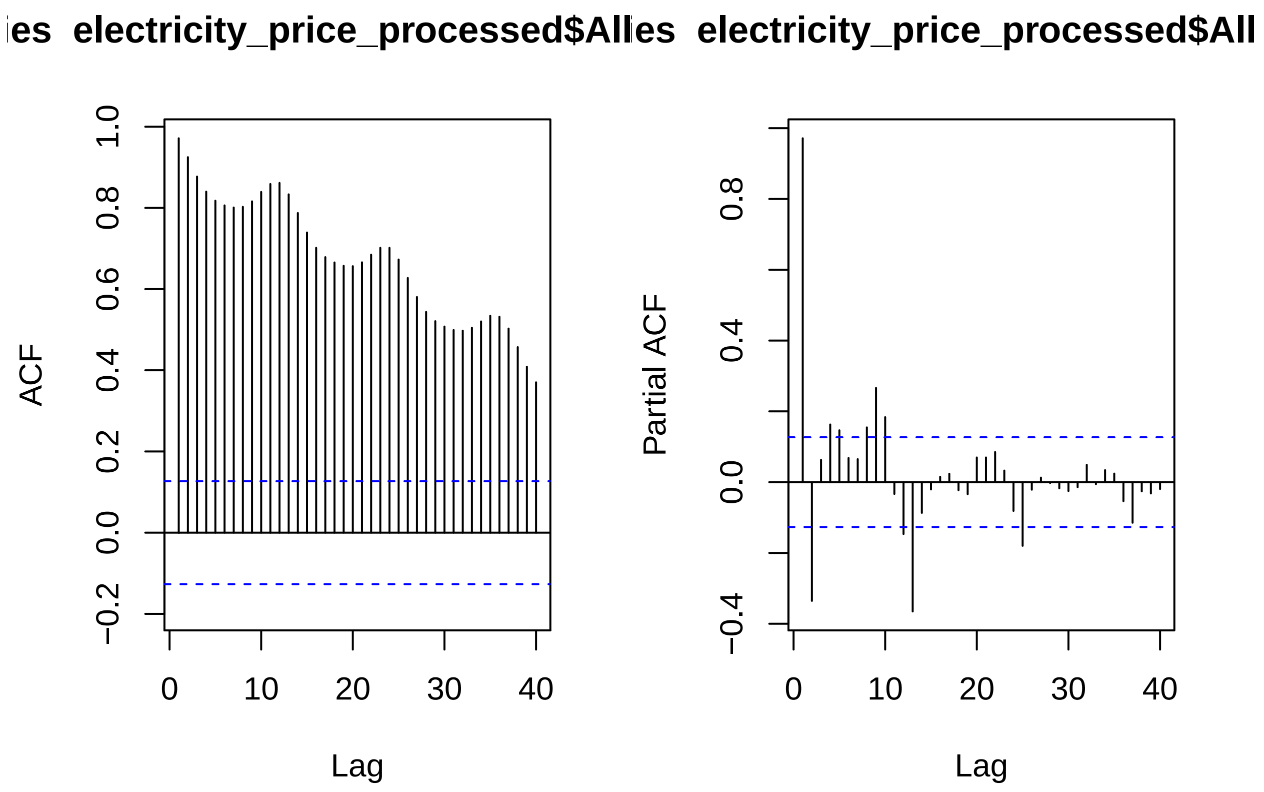
```
##          All.sectors Residential Commercial Industrial
## Sep 2019       10.82       13.16      10.96       7.06
## Oct 2019       10.39       12.81      10.74       6.84
## Nov 2019       10.38       13.03      10.57       6.72
## Dec 2019       10.22       12.68      10.32       6.38
## Jan 2020       10.28       12.79      10.24       6.33
## Feb 2020       10.29       12.85      10.36       6.41
## Mar 2020       10.29       13.09      10.41       6.38
## Apr 2020       10.42       13.28      10.42       6.40
## May 2020       10.47       13.15      10.46       6.53
## Jun 2020       10.96       13.28      10.95       6.94
## Jul 2020       11.14       13.26      10.90       7.17
## Aug 2020       11.11       13.31      10.95       7.09
## Sep 2020       11.07       13.55      11.07       7.01
## Oct 2020       10.64       13.60      10.73       6.72
## Nov 2020       10.45       13.35      10.59       6.48
```

Initial Plots

```
TS_Plot <-
  ggplot(electricity_price_processed, aes(x=Month, y=All.sectors)) +
      geom_line()
plot(TS_Plot)
```



```r
#ACF and PACF plots
par(mfrow=c(1,2))
ACF_Plot <- Acf(electricity_price_processed$All.sectors, lag = 40, plot = TRUE)
PACF_Plot <- Pacf(electricity_price_processed$All.sectors, lag = 40)
```

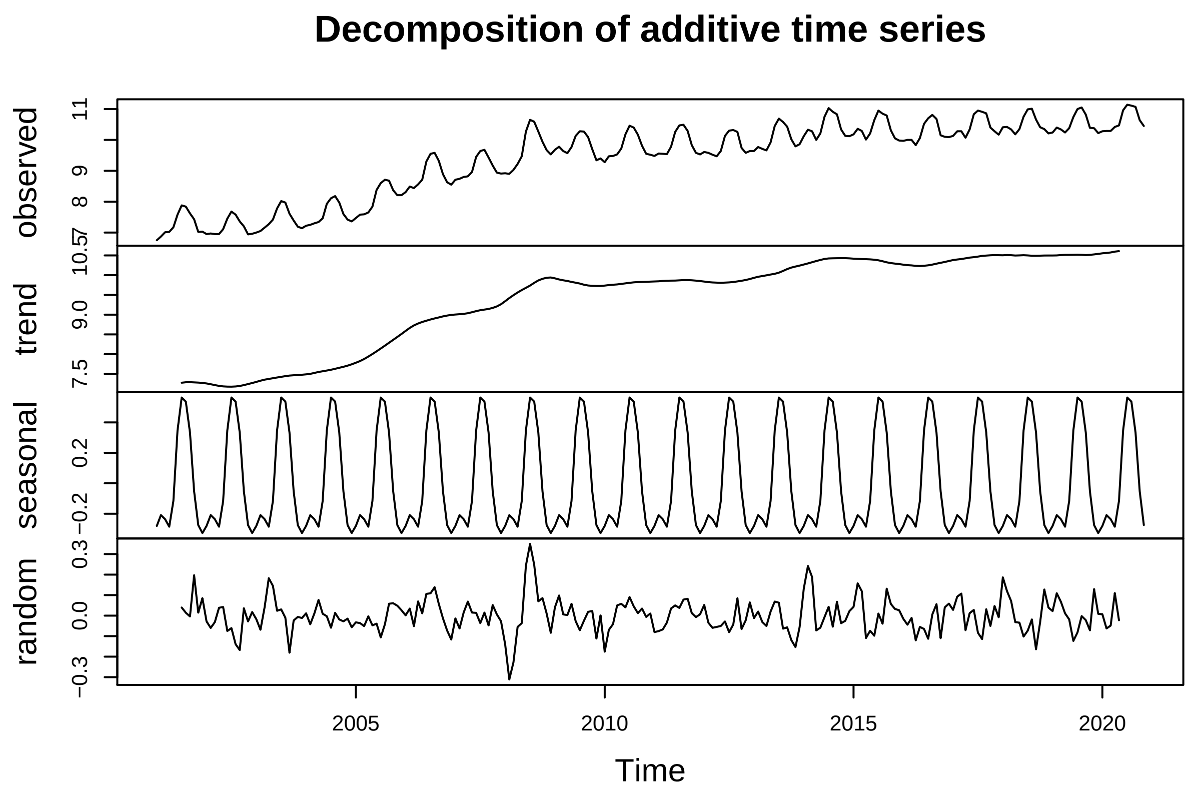


```r
par(mfrow=c(1,1))
```

Decomposing the time series

The plots from the previous section show the data has a seasonal component. Since we are working with non-seasonal ARIMA, we need to decompose the series and eliminate the seasonality.

```r
#Using R decompose function
decompose_allsectors_price <- decompose(ts_electricity_price[,"All.sectors"],"additive")
plot(decompose_allsectors_price)
```



```
#The ACF plot show a slow decay which is a sign of non-stationarity.

#Creating non-seasonal residential price time series because some models can't handle seasonality
deseasonal_allsectors_price <- seasadj(decompose_allsectors_price)
```

Forecasting with Naive methods

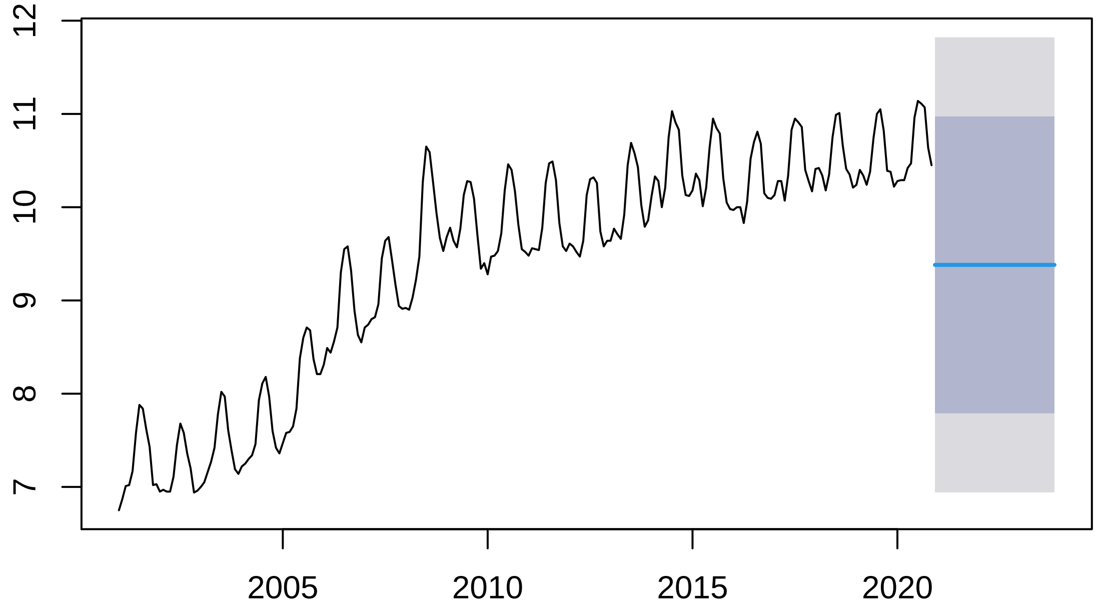
We will go over three naive methods. (a) Forecast using the arithmetic mean with function meanf() from package `forecast`. (b) Forecast using the naive method with function naive() from package `forecast`. (c) Forecast using the seasonal naive method with function snaive() from package `forecast`. You need to specify frequency when defining the ts object to be able to use this function.

These three function take a few main arguments:

y ts object containing data needed to be forecasted **h** forecasting horizon, i.e., how many steps ahead do you want to forecast **level** Confidence level for prediction intervals

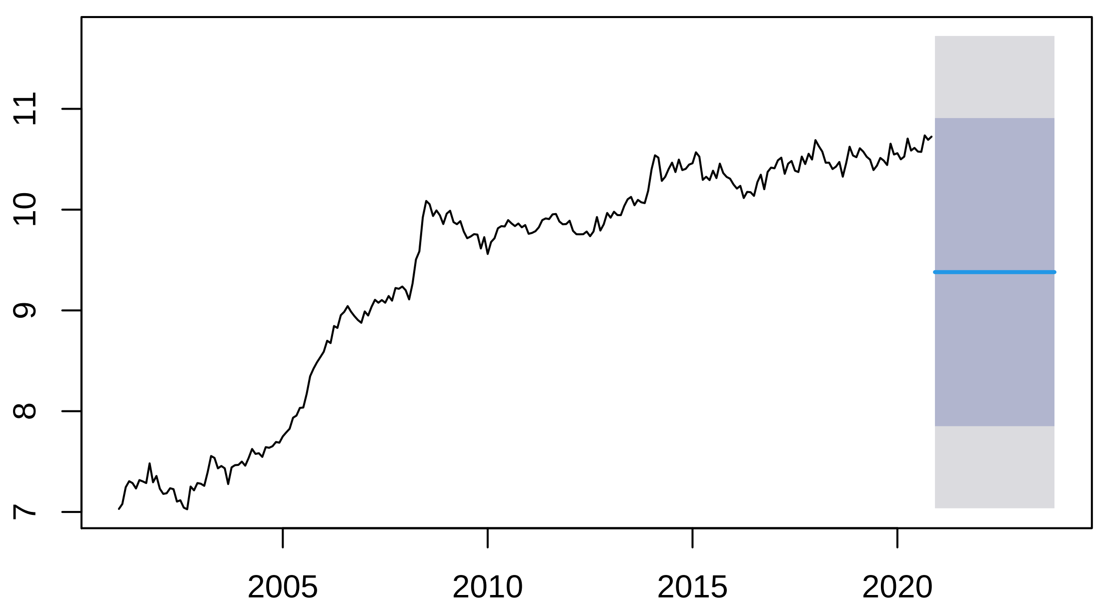
```
#Model 1: Arithmetic mean on original data
MEAN_seas <- meanf(y = ts_electricity_price[,"All.sectors"], h = 36)
plot(MEAN_seas)
```

Forecasts from Mean

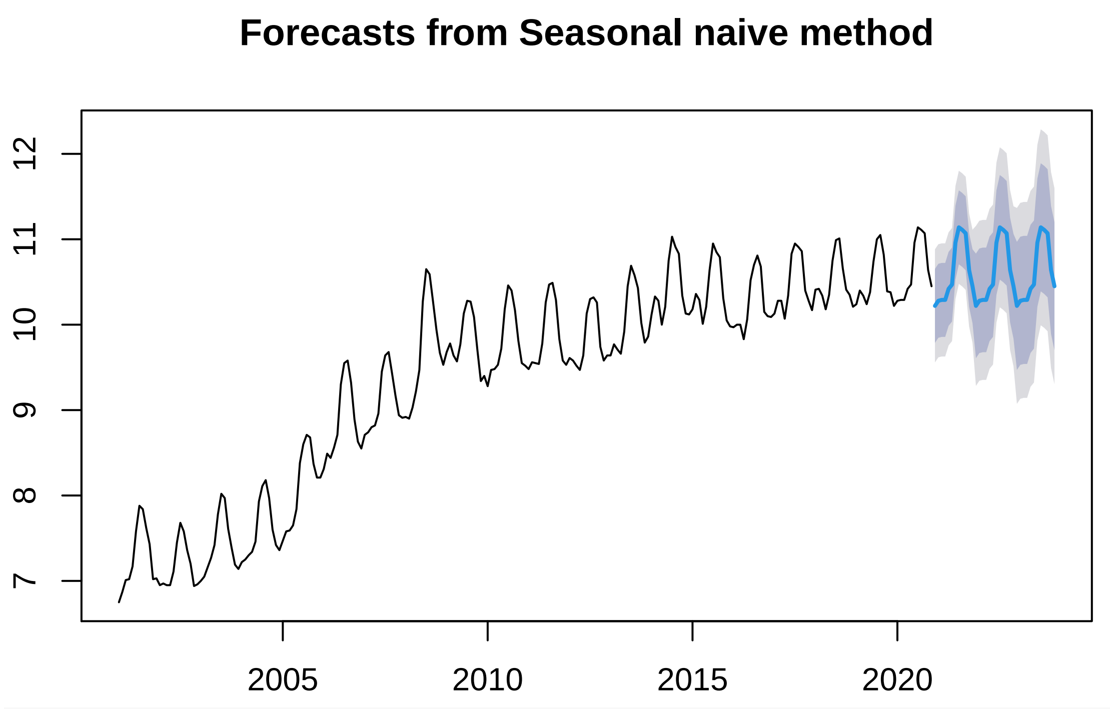


```
#Model 2: Arithmetic mean on deseas data
MEAN_deseas <- meanf(deseasonal_allsectors_price, h=36)
plot(MEAN_deseas)
```

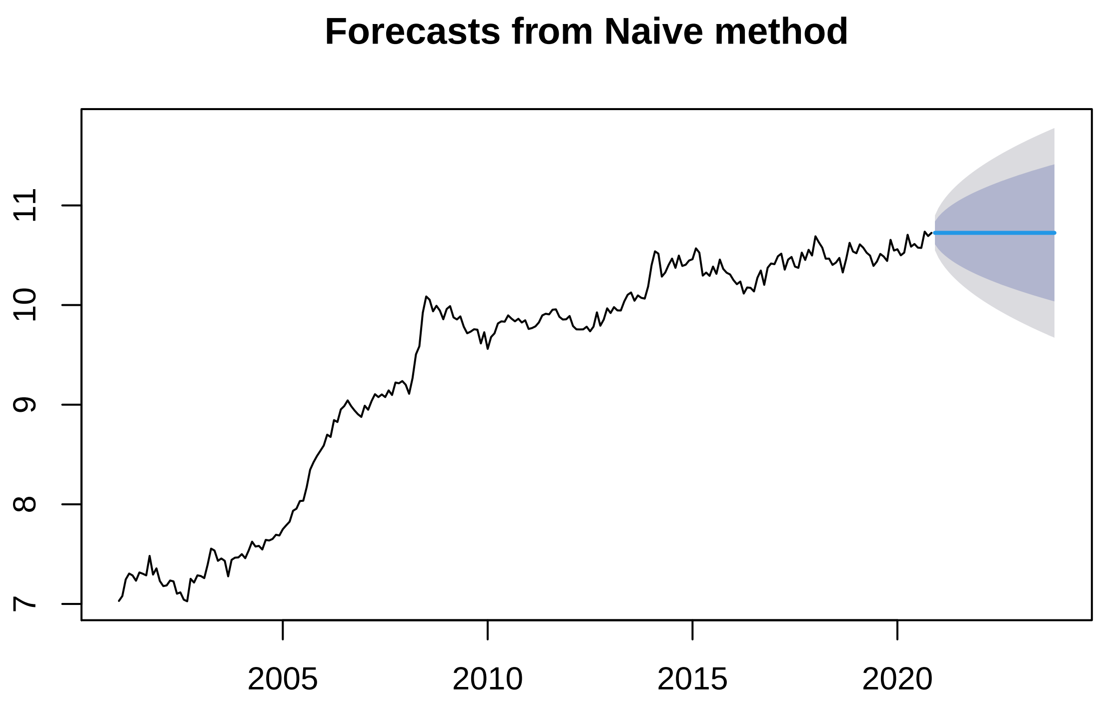
Forecasts from Mean



```
#Model 3: Seasonal naive on original data
SNAIVE_seas <- snaive(ts_electricity_price[,"All.sectors"], h=36)
plot(SNAIVE_seas)
```



```
#Model 4: Naive on deseas data
NAIVE_deseas <- naive(deseasonal_allsectors_price, h=36)
plot(NAIVE_deseas)
```



Forecasting with Simple Averaging Models

The simple averaging techniques we will explore are the moving average and the exponential smoothing. The simple moving average can be obtained with function sma() from package `smooth`. The main arguments function sma() will take are:

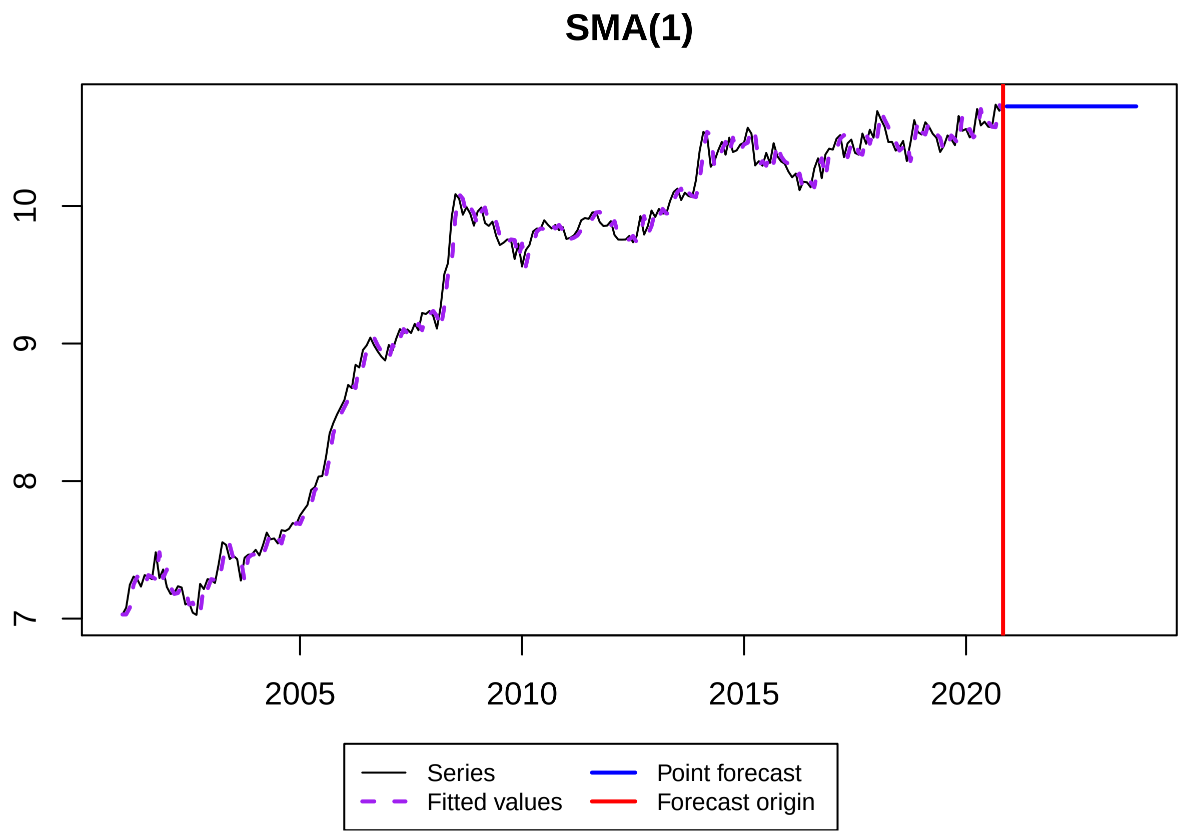
y ts object containing data needed to be forecasted **order** order of simple moving average. If you don't specify, R will find the best order for you **h** forecasting horizon, i.e., how many steps ahead do you want to forecast **holdout** If TRUE, the last **h** observations are taken from the end of the data. One reason to holdout some data is to check model performance. We will talk about that on M9. **silent** if FALSE, no output will be generated **level** Confidence level for prediction intervals

The simple exponential smoothing can be obtained with the ses() function also from package `forecast`. The main arguments function ses() will take are:

y ts object containing data needed to be forecasted **alpha** the smoothing constant. If you don't specify, R will find the best alpha for you. **h** forecasting horizon, i.e., how many steps ahead do you want to forecast **holdout** If TRUE, the last **h** observations are taken from the end of the data. One reason to holdout some data is to check model performance. We will talk about that on M9. **silent** if FALSE, no output will be generated **level** Confidence level for prediction intervals

We will try these models on the original dataset and the deaseason electricity price.

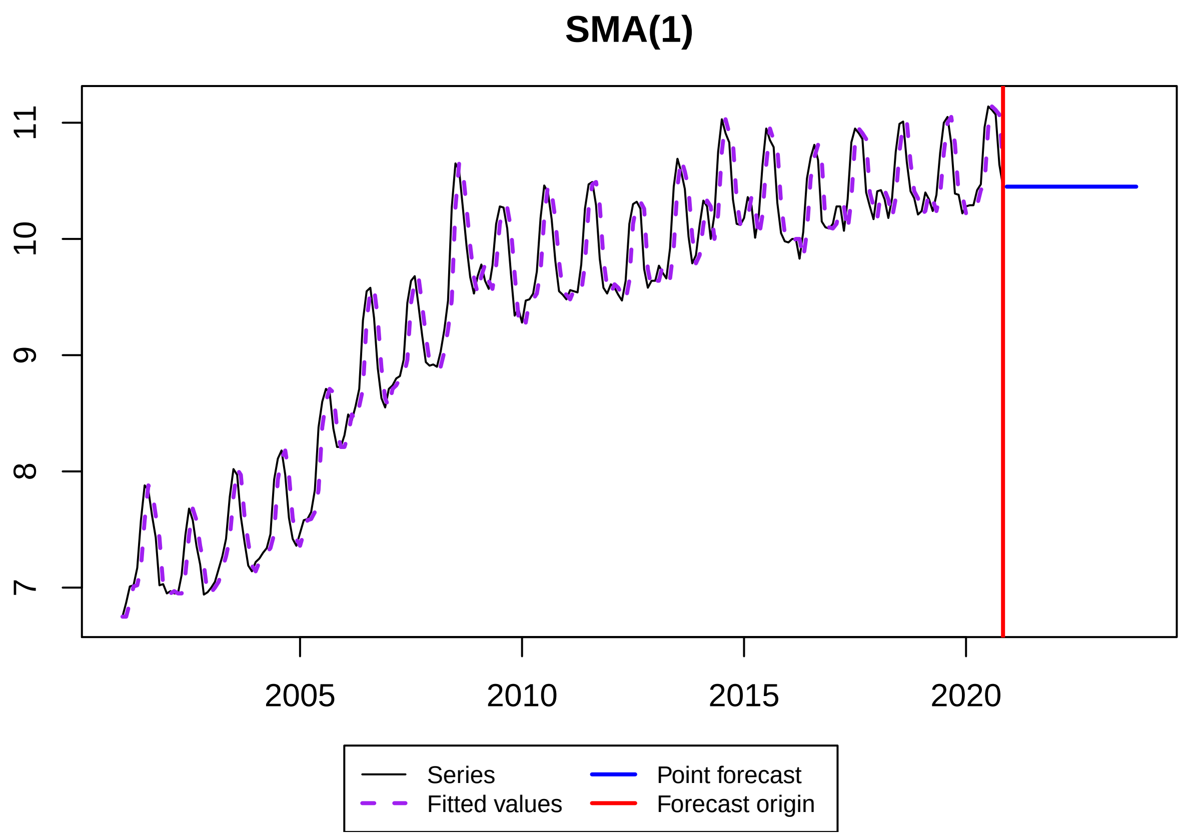
```
# Model 5: Simple moving average on deseasonal data
SMA_deseas_fit <- sma(y = deseasonal_allsectors_price, h = 36, holdout = FALSE, silent = FALSE) #order=
```



```
summary(SMA_deseas_fit)
```

```
## Time elapsed: 8.71 seconds
## Model estimated: SMA(1)
## Initial values were produced using backcasting.
##
## Loss function type: MSE; Loss function value: 0.008
## Error standard deviation: 0.0897
## Sample size: 239
## Number of estimated parameters: 2
## Number of degrees of freedom: 237
## Information criteria:
##       AIC      AICc       BIC      BICc
## -472.6098 -472.5589 -465.6569 -465.5176
```

```
# Model 6: Simple moving average on original data
SMA_seas_fit <- sma( y = ts_electricity_price[,"All.sectors"], h = 36, holdout = FALSE, silent = FALSE)
```



```
summary(SMA_seas_fit)
```

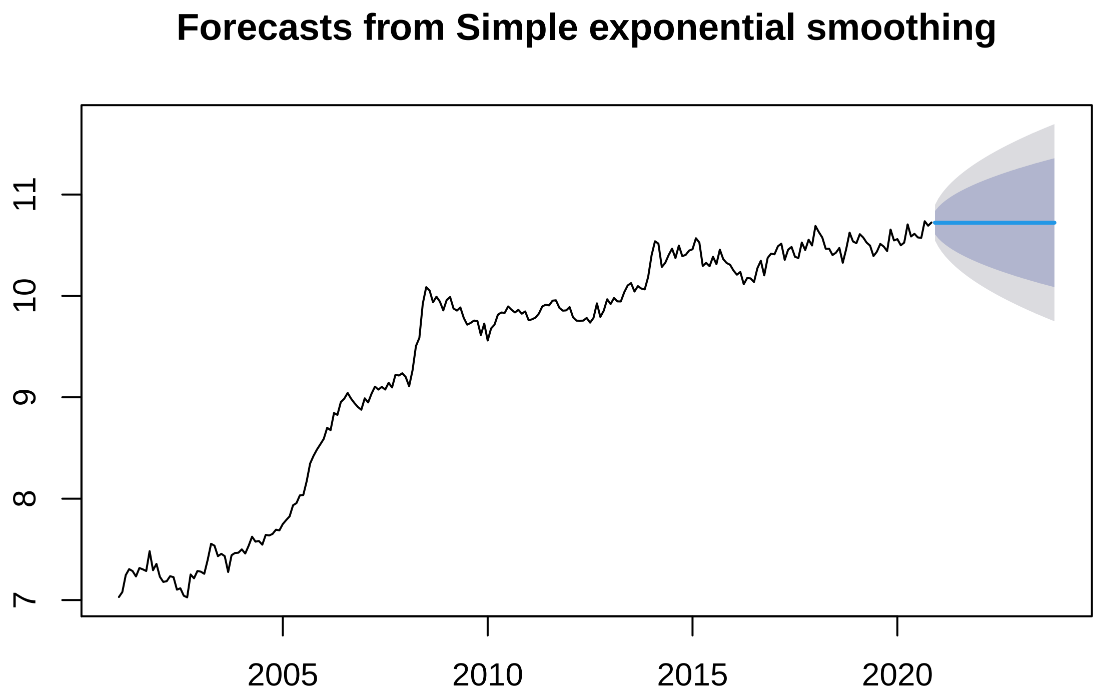
```
## Time elapsed: 6.53 seconds
## Model estimated: SMA(1)
## Initial values were produced using backcasting.
##
## Loss function type: MSE; Loss function value: 0.053
## Error standard deviation: 0.2311
## Sample size: 239
## Number of estimated parameters: 2
## Number of degrees of freedom: 237
## Information criteria:
##      AIC     AICc      BIC     BICc
## -19.9823 -19.9315 -13.0294 -12.8902
```

```
# Model 7:  Simple exponential smoothing on deseasonal data
SES_deseas_fit=ses( y = deseasonal_allsectors_price, h = 36, holdout = FALSE, silent = FALSE)  #alpha i
summary(SES_deseas_fit)
```

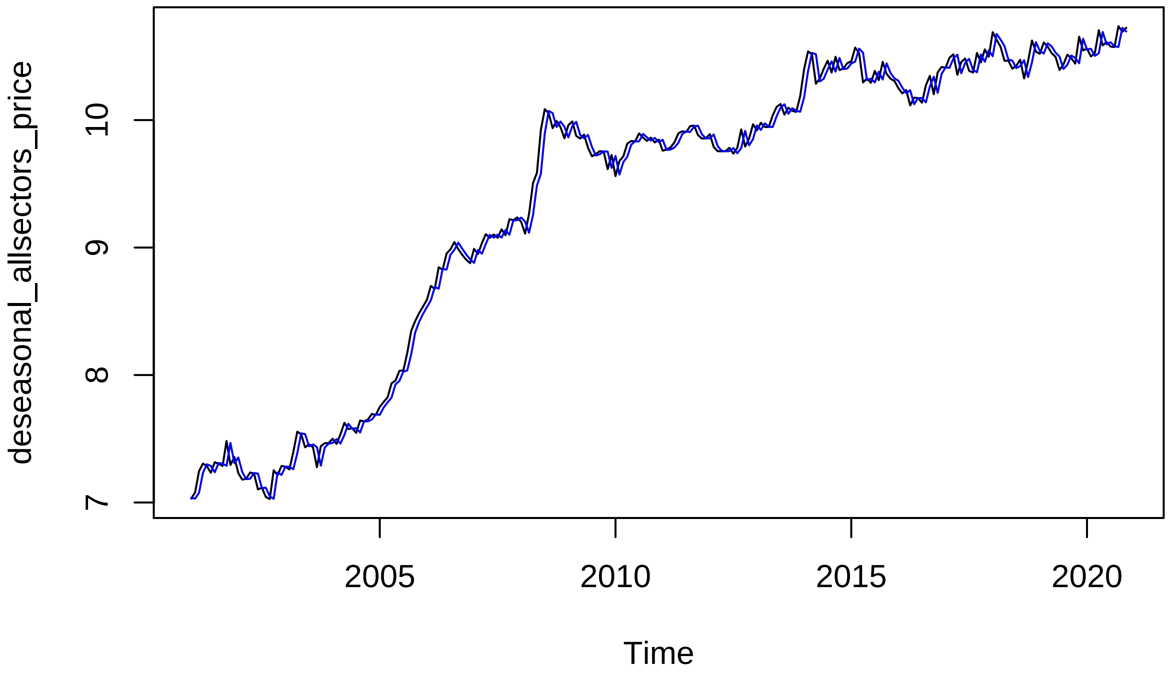
```
##
## Forecast method: Simple exponential smoothing
##
```

```
## Model Information:
## Simple exponential smoothing
##
## Call:
##  ses(y = deseasonal_allsectors_price, h = 36, holdout = FALSE,
##
##  Call:
##      silent = FALSE)
##
##   Smoothing parameters:
##     alpha = 0.923
##
##   Initial states:
##     l = 7.0347
##
##   sigma:  0.0894
##
##      AIC     AICc      BIC
## 158.4380 158.5401 168.8674
##
## Error measures:
##                      ME       RMSE        MAE      MPE      MAPE      MASE
## Training set 0.01671655 0.08898176 0.06787647 0.186005 0.7312023 0.2746517
##                  ACF1
## Training set -0.04116
##
## Forecasts:
##          Point Forecast    Lo 80    Hi 80     Lo 95    Hi 95
## Dec 2020       10.72227 10.60776 10.83679 10.547137 10.89741
## Jan 2021       10.72227 10.56644 10.87811 10.483941 10.96060
## Feb 2021       10.72227 10.53397 10.91057 10.434295 11.01025
## Mar 2021       10.72227 10.50634 10.93821 10.392029 11.05252
## Apr 2021       10.72227 10.48186 10.96269 10.354590 11.08996
## May 2021       10.72227 10.45965 10.98490 10.320626 11.12392
## Jun 2021       10.72227 10.43918 11.00537 10.289318 11.15523
## Jul 2021       10.72227 10.42009 11.02445 10.260127 11.18442
## Aug 2021       10.72227 10.40214 11.04241 10.232672 11.21187
## Sep 2021       10.72227 10.38514 11.05940 10.206678 11.23787
## Oct 2021       10.72227 10.36896 11.07558 10.181932 11.26261
## Nov 2021       10.72227 10.35349 11.09105 10.158272 11.28627
## Dec 2021       10.72227 10.33864 11.10590 10.135564 11.30898
## Jan 2022       10.72227 10.32435 11.12019 10.113704 11.33084
## Feb 2022       10.72227 10.31055 11.13399 10.092602 11.35194
## Mar 2022       10.72227 10.29720 11.14734 10.072184 11.37236
## Apr 2022       10.72227 10.28426 11.16029 10.052388 11.39216
## May 2022       10.72227 10.27169 11.17286 10.033161 11.41138
## Jun 2022       10.72227 10.25946 11.18509 10.014456 11.43009
## Jul 2022       10.72227 10.24754 11.19700  9.996233 11.44831
## Aug 2022       10.72227 10.23592 11.20863  9.978456 11.46609
## Sep 2022       10.72227 10.22456 11.21998  9.961094 11.48345
## Oct 2022       10.72227 10.21347 11.23108  9.944120 11.50043
## Nov 2022       10.72227 10.20260 11.24194  9.927508 11.51704
## Dec 2022       10.72227 10.19196 11.25258  9.911236 11.53331
## Jan 2023       10.72227 10.18153 11.26301  9.895284 11.54926
## Feb 2023       10.72227 10.17130 11.27324  9.879634 11.56491
## Mar 2023       10.72227 10.16125 11.28329  9.864269 11.58028
## Apr 2023       10.72227 10.15138 11.29316  9.849175 11.59537
## May 2023       10.72227 10.14168 11.30286  9.834338 11.61021
## Jun 2023       10.72227 10.13214 11.31240  9.819744 11.62480
## Jul 2023       10.72227 10.12275 11.32180  9.805382 11.63916
## Aug 2023       10.72227 10.11350 11.33104  9.791242 11.65330
## Sep 2023       10.72227 10.10440 11.34015  9.777314 11.66723
## Oct 2023       10.72227 10.09542 11.34912  9.763588 11.68096
## Nov 2023       10.72227 10.08657 11.35797  9.750056 11.69449
```

```
plot(SES_deseas_fit)
```



```
#Best alpha = 0.923
plot(deseasonal_allsectors_price,col="black")
lines(SES_deseas_fit$fitted, col="blue")
```

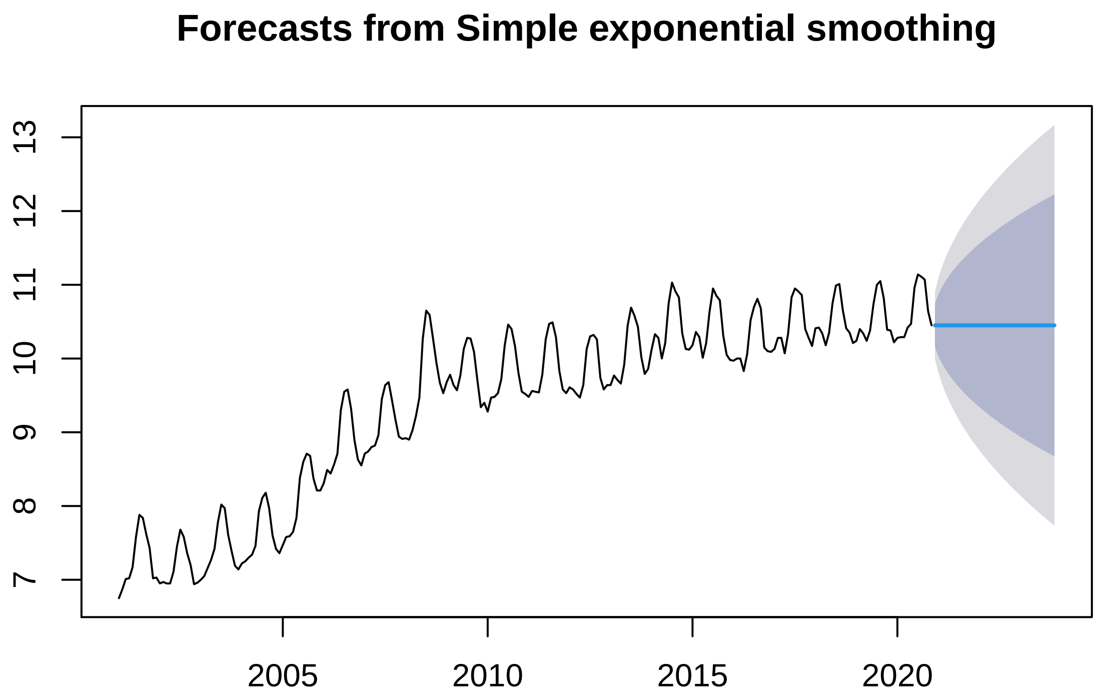


```r
# Model 8:  Simple exponential smoothing on original data
SES_seas_fit <- ses(y = ts_electricity_price[,"All.sectors"], h = 36, holdout = FALSE, silent = FALSE)
summary(SES_seas_fit)
```

```
##
## Forecast method: Simple exponential smoothing
##
## Model Information:
## Simple exponential smoothing
##
## Call:
##  ses(y = ts_electricity_price[, "All.sectors"], h = 36, holdout = FALSE,
##
##  Call:
##      silent = FALSE)
##
##   Smoothing parameters:
##     alpha = 0.9999
##
##   Initial states:
##     l = 6.7508
##
##   sigma:  0.2311
##
##      AIC     AICc      BIC
## 612.6650 612.7672 623.0944
##
## Error measures:
##                      ME      RMSE       MAE       MPE    MAPE      MASE
## Training set 0.01547949 0.2301422 0.1759971 0.1530334 1.87496 0.7121451
##                   ACF1
## Training set 0.5244896
##
```

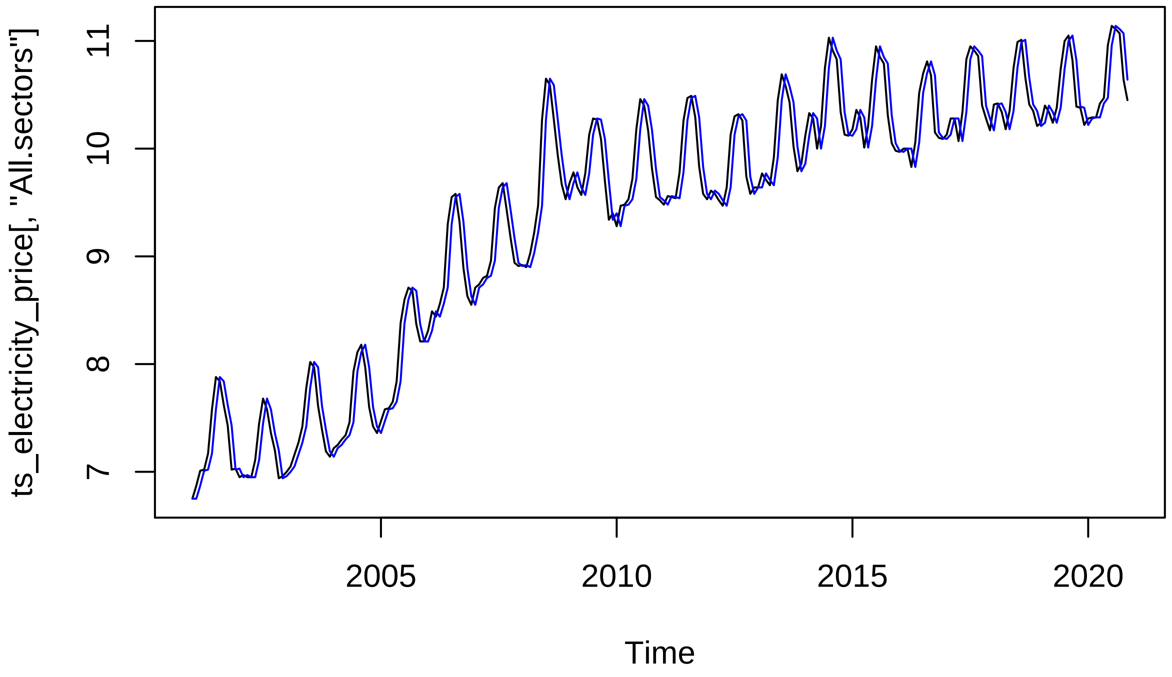
```
## Forecasts:
##          Point Forecast     Lo 80    Hi 80    Lo 95    Hi 95
## Dec 2020       10.45002 10.153838 10.74620 9.997049 10.90299
## Jan 2021       10.45002 10.031177 10.86886 9.809455 11.09058
## Feb 2021       10.45002  9.937053 10.96299 9.665505 11.23453
## Mar 2021       10.45002  9.857702 11.04234 9.544148 11.35589
## Apr 2021       10.45002  9.787791 11.11225 9.437229 11.46281
## May 2021       10.45002  9.724587 11.17545 9.340567 11.55947
## Jun 2021       10.45002  9.666465 11.23357 9.251677 11.64836
## Jul 2021       10.45002  9.612366 11.28767 9.168940 11.73110
## Aug 2021       10.45002  9.561555 11.33848 9.091231 11.80881
## Sep 2021       10.45002  9.513497 11.38654 9.017732 11.88231
## Oct 2021       10.45002  9.467787 11.43225 8.947825 11.95221
## Nov 2021       10.45002  9.424112 11.47593 8.881030 12.01901
## Dec 2021       10.45002  9.382222 11.51782 8.816965 12.08307
## Jan 2022       10.45002  9.341914 11.55812 8.755319 12.14472
## Feb 2022       10.45002  9.303022 11.59702 8.695839 12.20420
## Mar 2022       10.45002  9.265406 11.63463 8.638310 12.26173
## Apr 2022       10.45002  9.228949 11.67109 8.582553 12.31748
## May 2022       10.45002  9.193549 11.70649 8.528413 12.37162
## Jun 2022       10.45002  9.159119 11.74092 8.475757 12.42428
## Jul 2022       10.45002  9.125584 11.77445 8.424470 12.47557
## Aug 2022       10.45002  9.092877 11.80716 8.374449 12.52559
## Sep 2022       10.45002  9.060940 11.83910 8.325606 12.57443
## Oct 2022       10.45002  9.029721 11.87032 8.277861 12.62218
## Nov 2022       10.45002  8.999174 11.90086 8.231143 12.66890
## Dec 2022       10.45002  8.969257 11.93078 8.185388 12.71465
## Jan 2023       10.45002  8.939932 11.96011 8.140540 12.75950
## Feb 2023       10.45002  8.911166 11.98887 8.096547 12.80349
## Mar 2023       10.45002  8.882928 12.01711 8.053360 12.84668
## Apr 2023       10.45002  8.855190 12.04485 8.010939 12.88910
## May 2023       10.45002  8.827926 12.07211 7.969242 12.93080
## Jun 2023       10.45002  8.801113 12.09892 7.928235 12.97180
## Jul 2023       10.45002  8.774729 12.12531 7.887884 13.01215
## Aug 2023       10.45002  8.748754 12.15128 7.848159 13.05188
## Sep 2023       10.45002  8.723170 12.17687 7.809031 13.09101
## Oct 2023       10.45002  8.697959 12.20208 7.770475 13.12956
## Nov 2023       10.45002  8.673106 12.22693 7.732466 13.16757
```

```
plot(SES_seas_fit)
```



```
#Best alpha = 0.9999

plot(ts_electricity_price[,"All.sectors"],col="black")
lines(SES_seas_fit$fitted, col="blue")
```



Forecasting with ARIMA

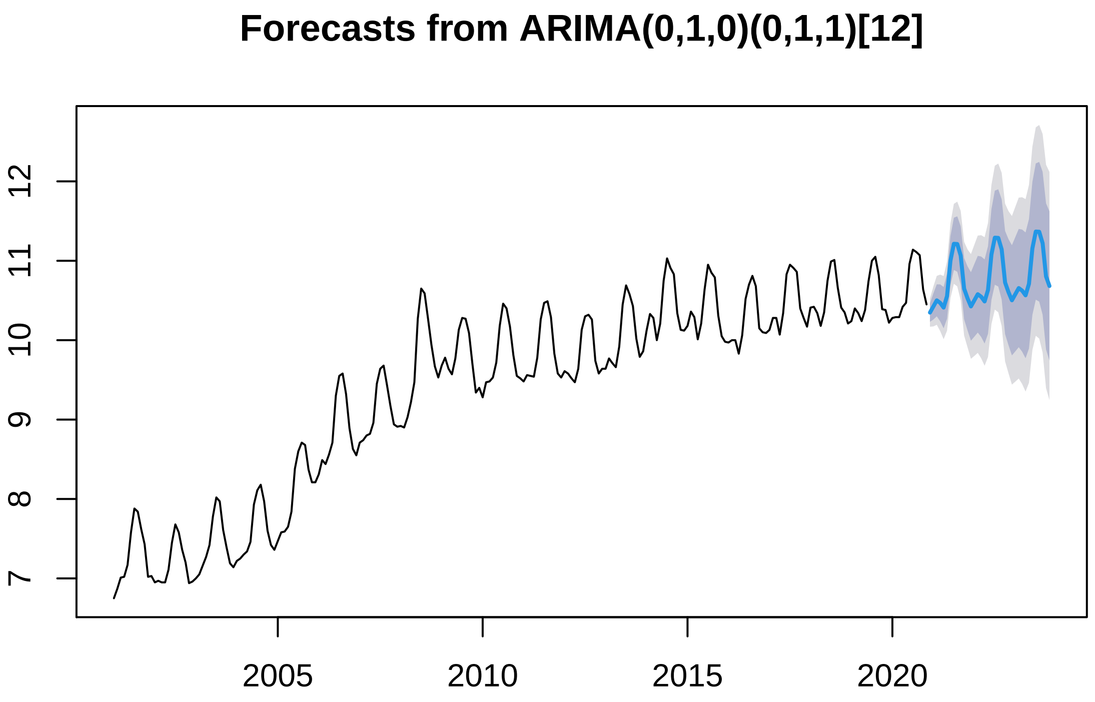
When forecasting with ARIMA models, first you will need to fit the model with the auto.arima() function we studied on M6 and M7. Then you will need to use the forecast() function from package `forecast`. forecast() is a generic function for forecasting from time series or time series models. It takes two main arguments:

object a time series or time series model for which forecasts are required; **h** forecasting horizon, i.e., how many steps ahead do you want to forecast.

```
# Model 9:  SARIMA on original data
SARIMA_autofit <- auto.arima(ts_electricity_price[,"All.sectors"])
print(SARIMA_autofit)
```

```
## Series: ts_electricity_price[, "All.sectors"]
## ARIMA(0,1,0)(0,1,1)[12]
##
## Coefficients:
##          sma1
##       -0.6796
## s.e.   0.0647
##
## sigma^2 estimated as 0.008235:  log likelihood=218.47
## AIC=-432.93   AICc=-432.88   BIC=-426.09
```

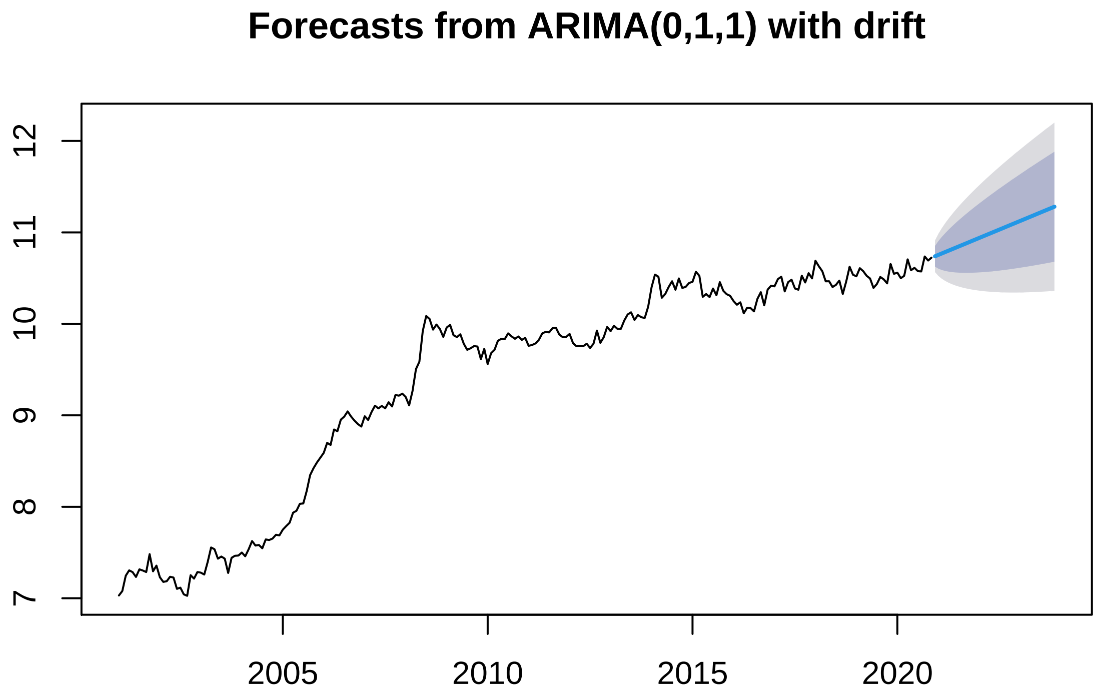
```
SARIMA_forecast <- forecast(object = SARIMA_autofit, h = 36)
plot(SARIMA_forecast)
```



```
# Model 10:  ARIMA on deseasonal data
ARIMA_autofit <- auto.arima(deseasonal_allsectors_price, max.D = 0, max.P = 0, max.Q = 0)
print(ARIMA_autofit)
```

```
## Series: deseasonal_allsectors_price
## ARIMA(0,1,1) with drift
##
## Coefficients:
##           ma1   drift
##       -0.1133  0.0155
## s.e.   0.0622  0.0050
##
## sigma^2 estimated as 0.007723:  log likelihood=242.05
## AIC=-478.11   AICc=-478.01   BIC=-467.69
```

```
ARIMA_forecast <- forecast(object = ARIMA_autofit, h = 36)
plot(ARIMA_forecast)
```


Forecasts from ARIMA(0,1,1) with drift