M9: Forecast Accuracy

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)
library(smooth)

#New package for M9 to assist with tables
#install.packages("kableExtra")
library(kableExtra)
```

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.csv",
  header=TRUE,
  skip=4)

#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)
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
```

```r
#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()*.

```r
ts_electricity_price <- ts(
  electricity_price_processed[,2:(nvar+1)],
  start=c(year(electricity_price_processed$Month[1]),month(electricity_price_processed$Month[1])),
  frequency=12)

#note that we are only transforming columns with electricity price, not the date columns
head(ts_electricity_price,15)
```

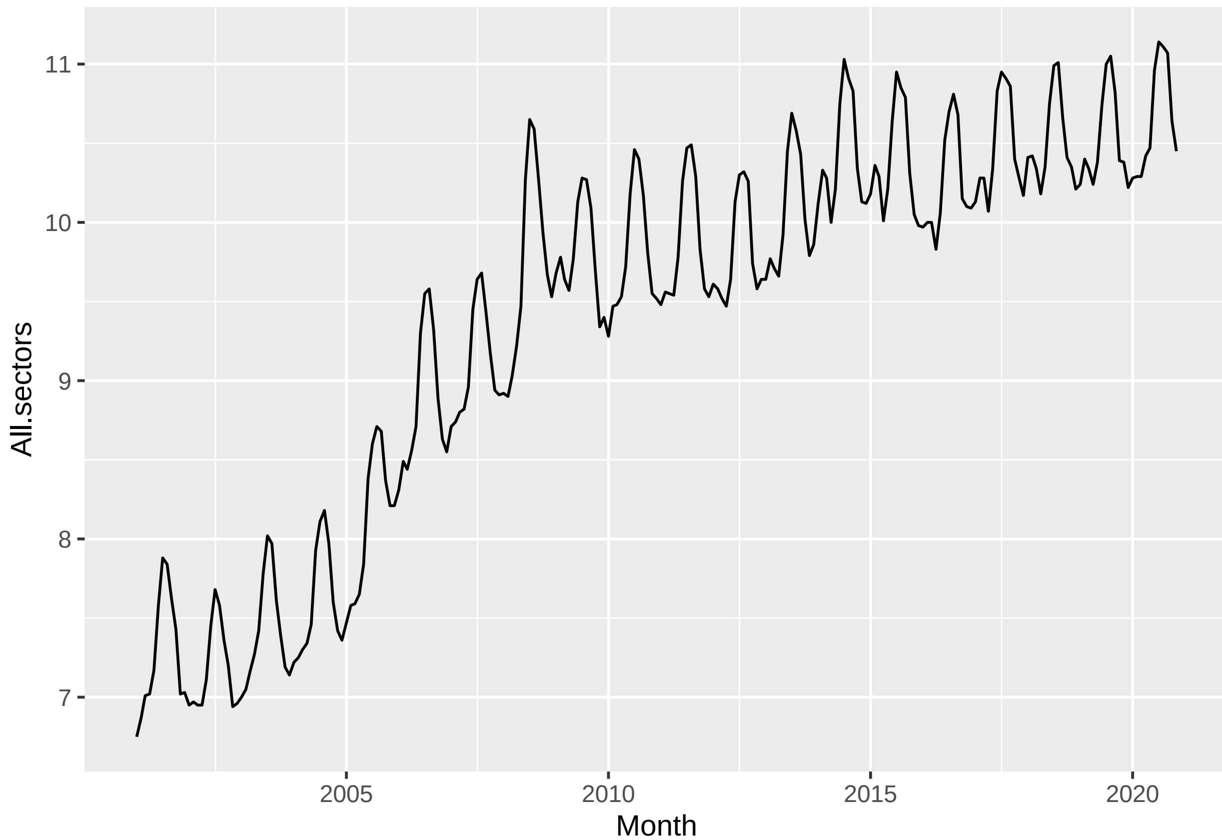
```
##          All.sectors Residential Commercial Industrial
## Jan 2001        6.75        7.73       7.25       4.73
## Feb 2001        6.87        8.04       7.51       4.80
## Mar 2001        7.01        8.32       7.70       4.86
## Apr 2001        7.02        8.46       7.73       4.87
## May 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
```

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

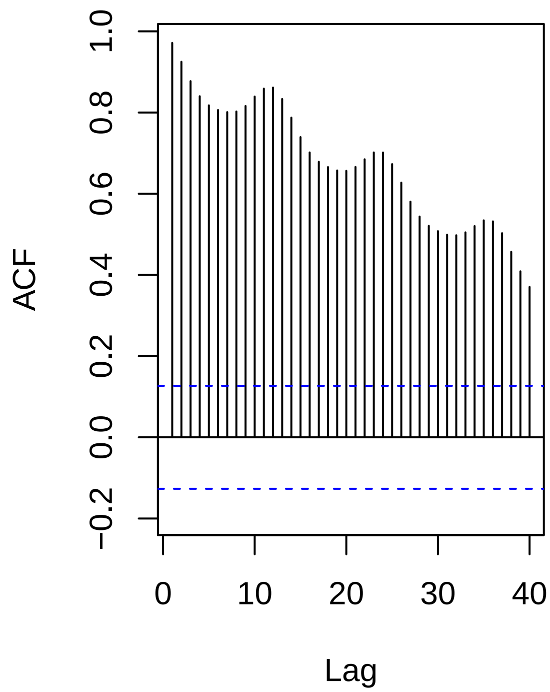
```
##          All.sectors Residential Commercial Industrial
## Sep 2019       10.82       13.16      10.96       7.06
## Oct 2019       10.39       12.81      10.74       6.84
## Nov 2019       10.38       13.03      10.57       6.72
## Dec 2019       10.22       12.68      10.32       6.38
## Jan 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
```

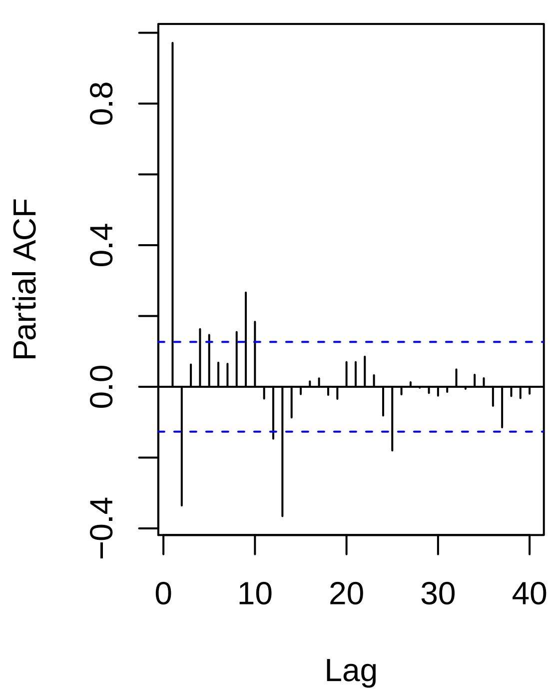
Initial Plots

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




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



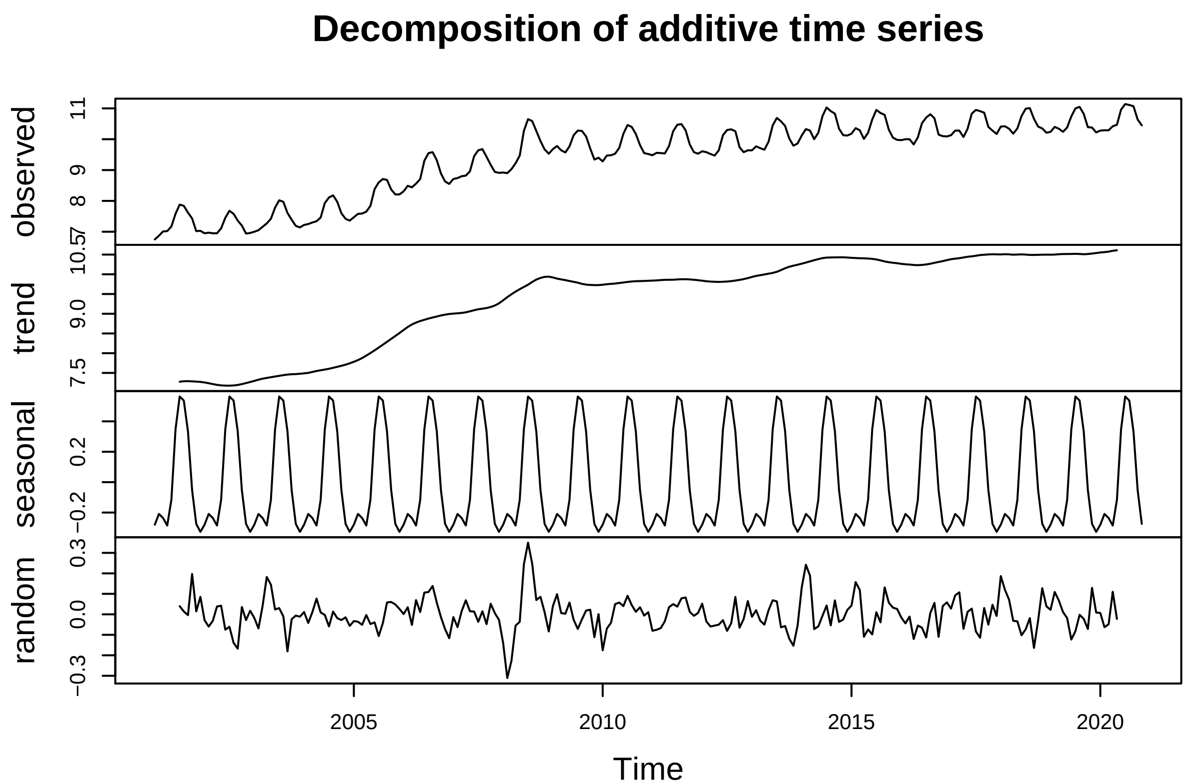



```
par(mfrow=c(1,1))
```

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.

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



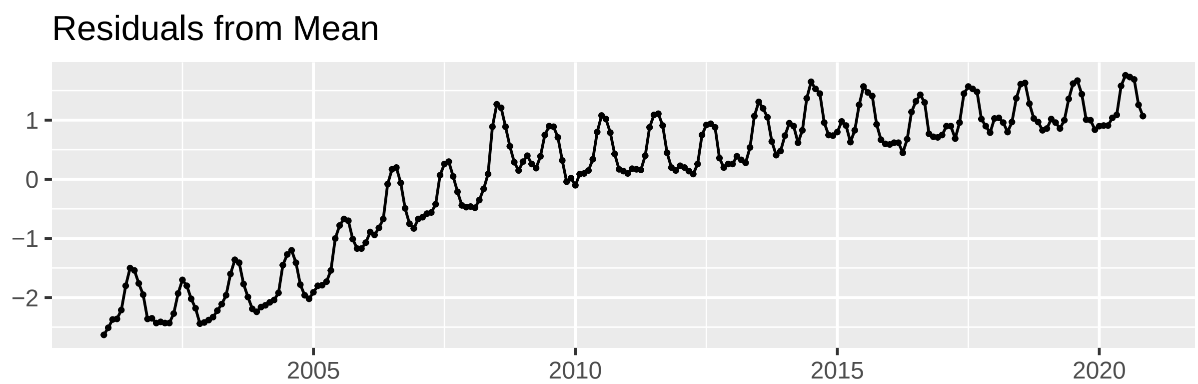

```
#The ACF plot show a slow decay which is a sign of non-stationarity.
#Creating non-seasonal residential price time series because some models can't handle seasonality
deseasonal_allsectors_price <- seasadj(decompose_allsectors_price)
```

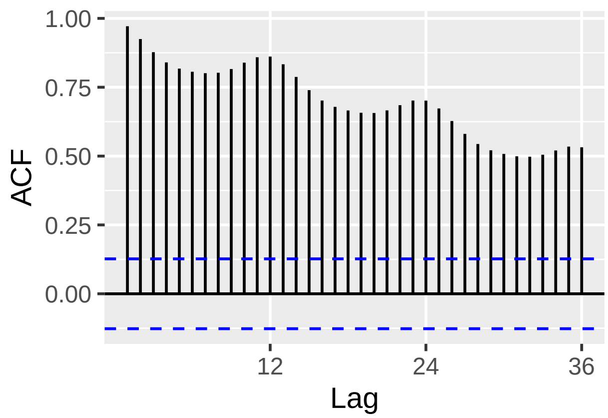
Fitting Models to the original (seasonal) series

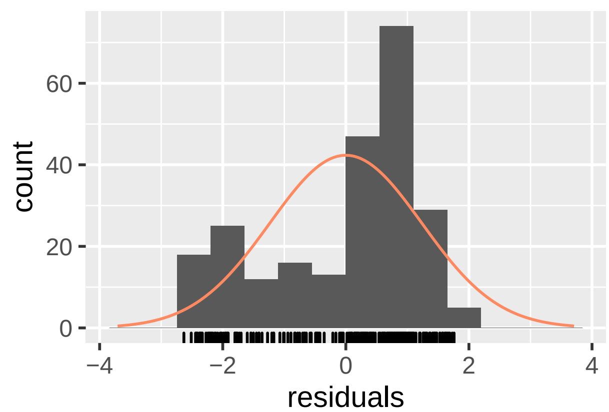
On M8 we tried several models for both seasonal and deseasonal electricity price series. This week the goal is to chekc accuracy of those models. Let's start by looking at residual plots and AIC to check how the models represent the historical prices.

Model 1: Arithmetic mean

```
MEAN_seas <- meanf(y = ts_electricity_price[,"All.sectors"], h = 12)
checkresiduals(MEAN_seas)
```





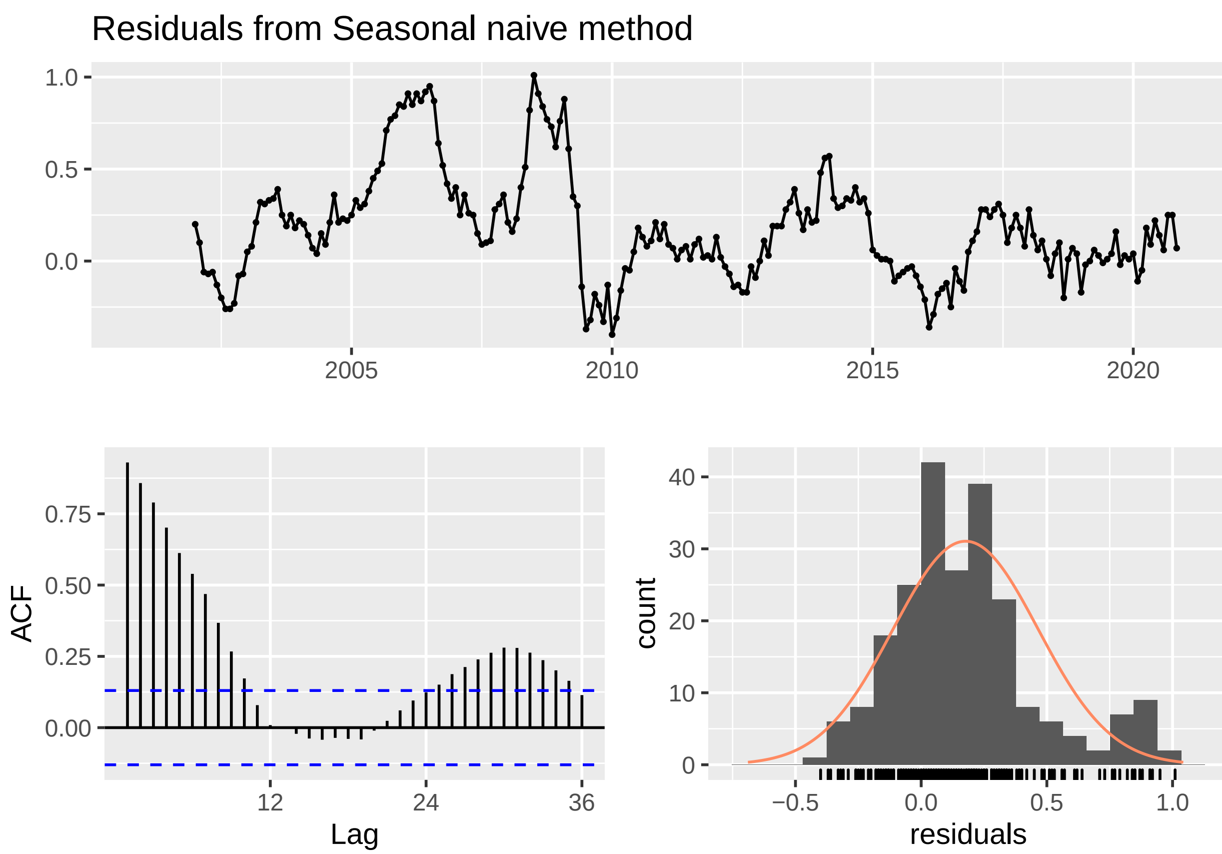



```
##
##  Ljung-Box test
##
## data:  Residuals from Mean
## Q* = 3730.9, df = 23, p-value < 2.2e-16
##
## Model df: 1.   Total lags used: 24
```

Note a clear trend on residuals series, showing that the mean is not a good to model the trend component. And aside from trend the seasonal component is also not being modeled.

Model 2: Seasonal naive

```
SNAIVE_seas <- snaive(ts_electricity_price[,"All.sectors"], h=12)
checkresiduals(SNAIVE_seas)
```

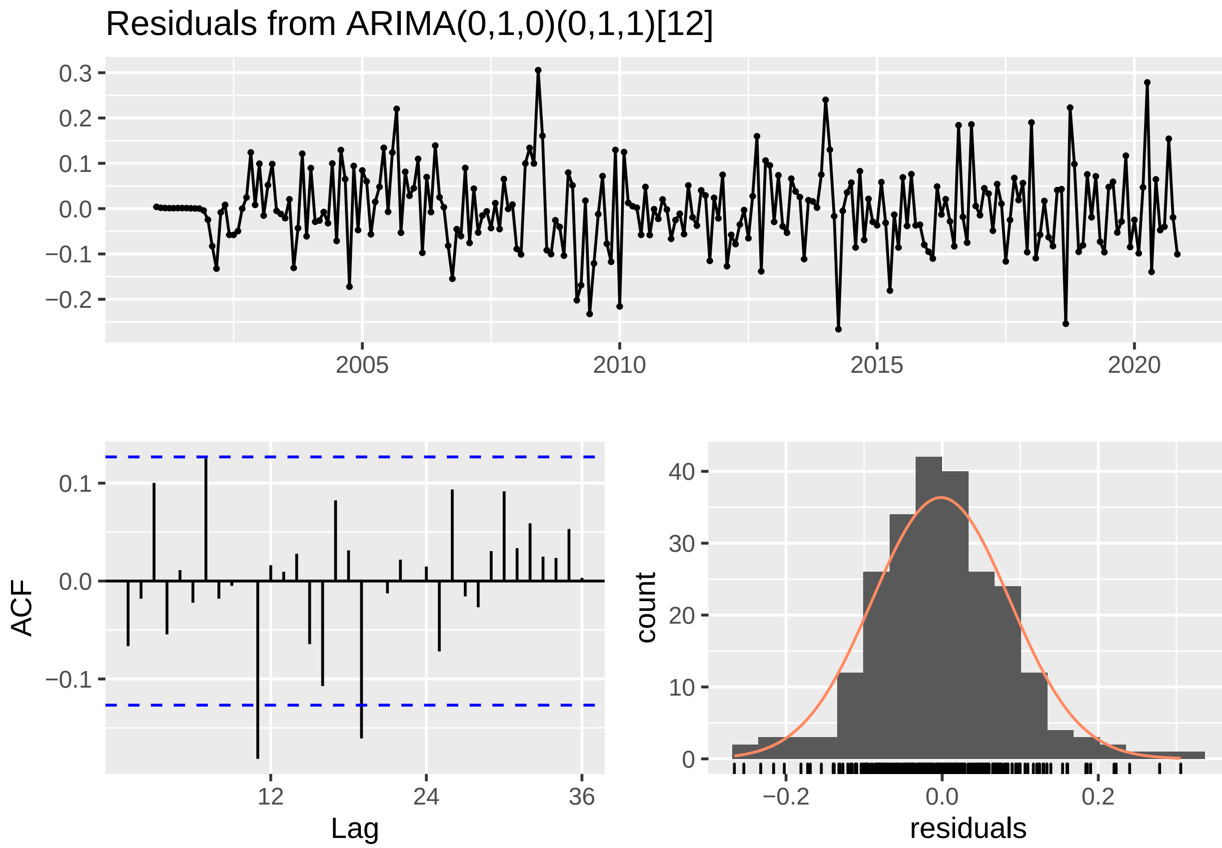



```
##
##  Ljung-Box test
##
## data:  Residuals from Seasonal naive method
## Q* = 903.69, df = 24, p-value < 2.2e-16
##
## Model df: 0.   Total lags used: 24
```

The residuals for the seasonal naive don't seem to have a strong trend. Because it repeats the observations that happen in a previous seasonal lag (in this case one year ago), the seasonal naive is able to model the trend and seasonal component. But the residuals series show a strong autoregressive component which is also not desired.

Model 3: SARIMA

```
SARIMA_autofit <- auto.arima(ts_electricity_price[,"All.sectors"])
checkresiduals(SARIMA_autofit)
```




```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(0,1,0)(0,1,1)[12]
## Q* = 30.131, df = 23, p-value = 0.1457
##
## Model df: 1.   Total lags used: 24
```

This is by far the best fit. Notice the residual series seems to be random and ACF shows no significant self correlation.

Model Performance for forecasting 12 steps ahead

We are done with backward-looking assessment. SARIMA seems to be a good fit for our data. In a real world, you wouldn't even move further with the arithmetic mean or the seasonal naive method. Since they fail the backward-looking assessment, it's known that they will lead to poor forecast. But just as an exercise we will also perform a forward-looking assessment for all three model.

Function accuracy() from package `forecast`

The function accuracy() will return performance measures. It takes the main arguments:

object object of class forecast, or numerical values containing forecasts. **x** numerical vector containing observed values (optional).

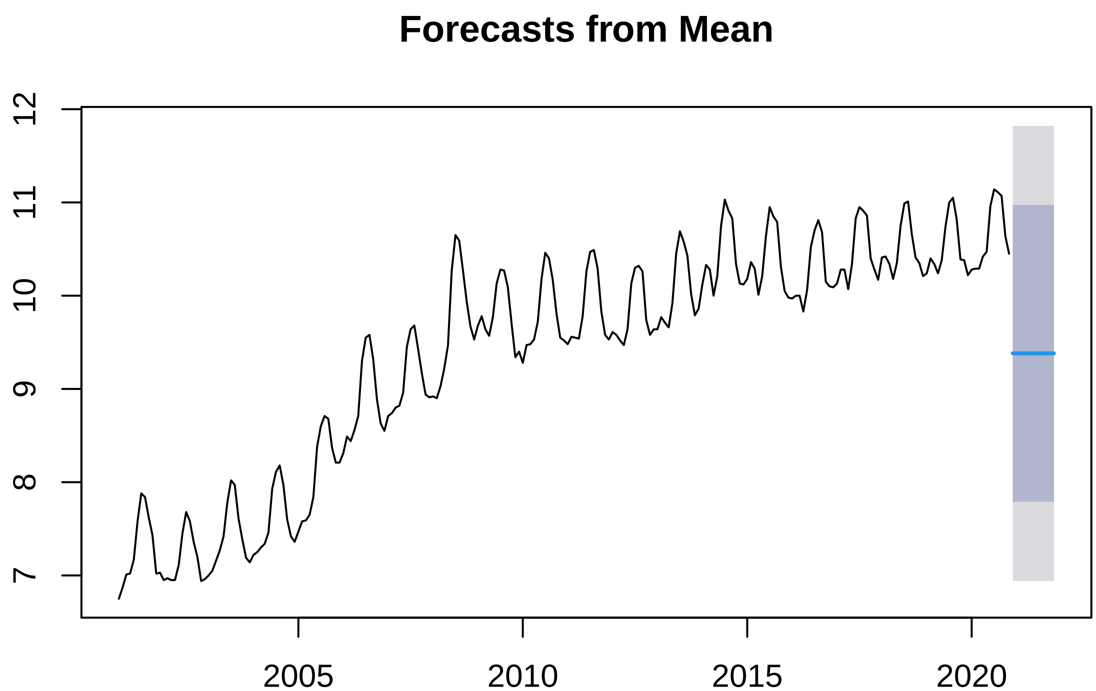
If **x** is not provided the function will return performance measures for trainign set, i.e., based on historical data it will compare observed and fitted values.

The measures calculated are:

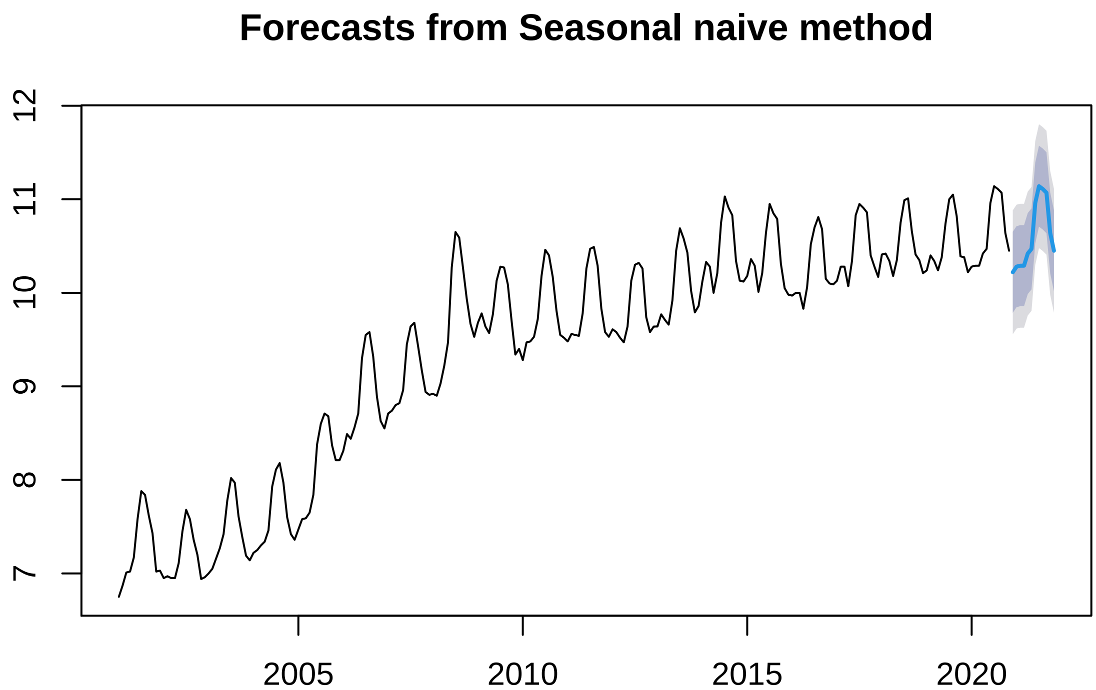
ME: Mean Error **RMSE:** Root Mean Squared Error **MAE:** Mean Absolute Error **MPE:** Mean Percentage Error **MAPE:** Mean Absolute Percentage Error **MASE:** Mean Absolute Scaled Error **ACF1:** Autocorrelation of errors at lag 1

Checking accuracy of the three models

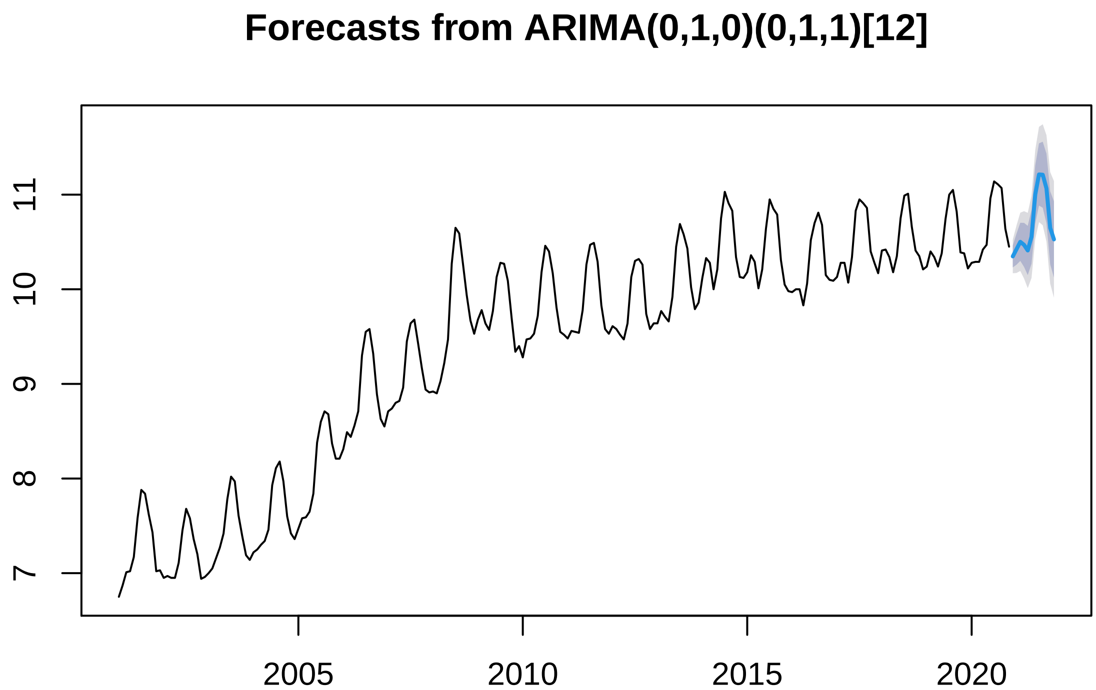
```
#Model 1: Arithmetic mean
MEAN_scores <- accuracy(MEAN_seas)  #store the performance metrics
plot(MEAN_seas)  #plot forecasts
```




```
#Model 2: Seasonal naive
SNAIVE_scores <- accuracy(SNAIVE_seas)
plot(SNAIVE_seas)
```




```
# Model 3:  SARIMA
#remember auto.arima does not call the forecast() internally so we need one more step
SARIMA_for <- forecast(SARIMA_autofit,h=12)
SARIMA_scores <- accuracy(SARIMA_for)
plot(SARIMA_for)
```



Compare performance metrics

Now we will create a data frame that combines performance metrics for all the three models. You can choose one metric to help you choose among models. For example let's say we want the model with lowest RMSE.

```
#create data frame
seas_scores <- as.data.frame(rbind(MEAN_scores, SNAIVE_scores, SARIMA_scores))
row.names(seas_scores) <- c("MEAN", "SNAIVE","SARIMA")

#choose model with lowest RMSE
best_model_index <- which.min(seas_scores[,"RMSE"])
cat("The best model by RMSE is:", row.names(seas_scores[best_model_index,]))
```

```
## The best model by RMSE is: SARIMA
```

SARIMA was the best fit for the seasonal data. If you want generate a table to compare model accuracy and help visualize the results here is a suggestion on how to include a table on your Rmd report. You can use the `kable_styling(latex_options="striped")` to highlight the model that leads to minimum RMSE.

```
kbl(seas_scores,
      caption = "Forecast Accuracy for Seasonal Data",
      digits = array(5,ncol(seas_scores))) %>%
  kable_styling(full_width = FALSE, position = "center") %>%
  #highlight model with lowest RMSE
  kable_styling(latex_options="striped", stripe_index = which.min(seas_scores[,"RMSE"]))
```

Table 1: Forecast Accuracy for Seasonal Data

	ME	RMSE	MAE	MPE	MAPE	MASE	ACF1
MEAN	0.00000	1.23333	1.04058	-1.96778	11.94872	4.21055	0.97150
SNAIVE	0.17674	0.33787	0.24714	1.93341	2.68261	1.00000	0.93001
SARIMA	-0.00150	0.08805	0.06652	-0.01304	0.70553	0.26917	-0.06648