Chapter 15: Timeseries and Forecasting

Ram Gopal, Dan Philps, and Tillman Weyde

2022

Contents

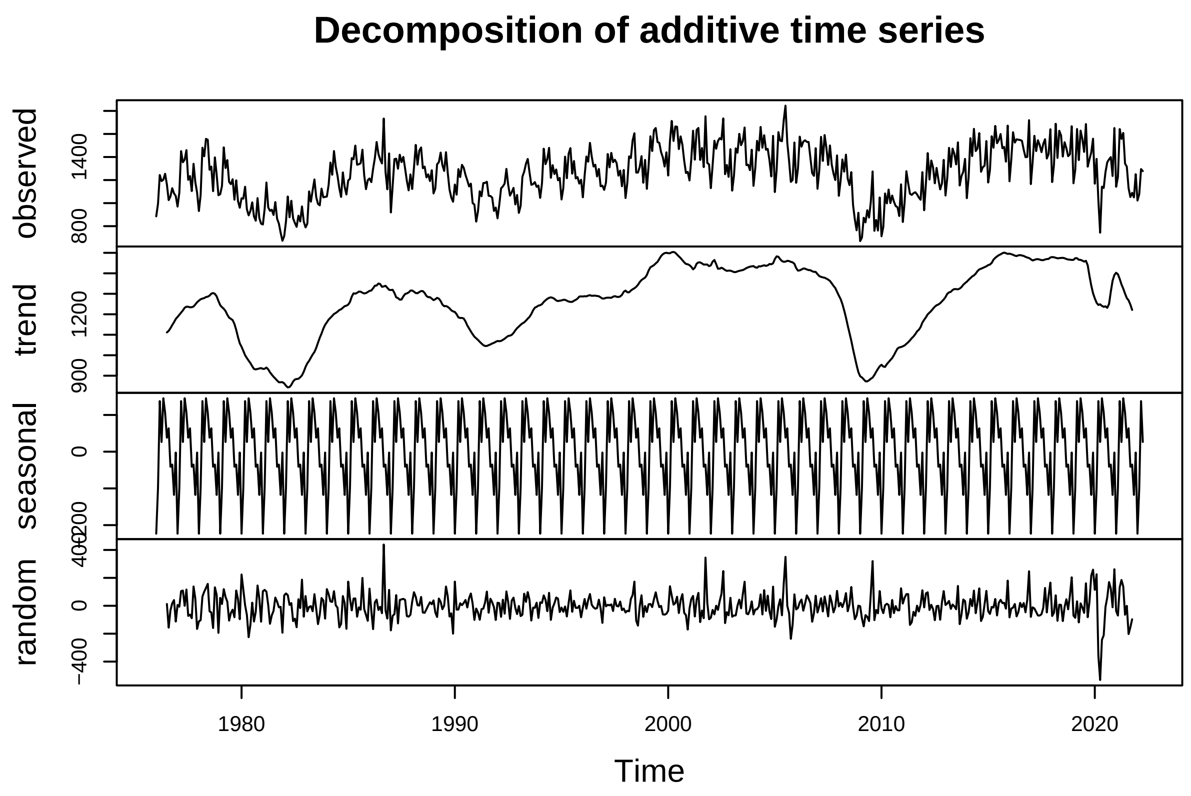
Load packages

```
library(xts)
library(forecast)
```

In this notebook, we will look at timeseries and how to forecast future events and datapoints. Timeseries, such as the changing level profits quarter by quarter, or daily prices, are a central part of business analytics. We can decompose timeseries into trend, seasonality and noise using simple tools as shown below.

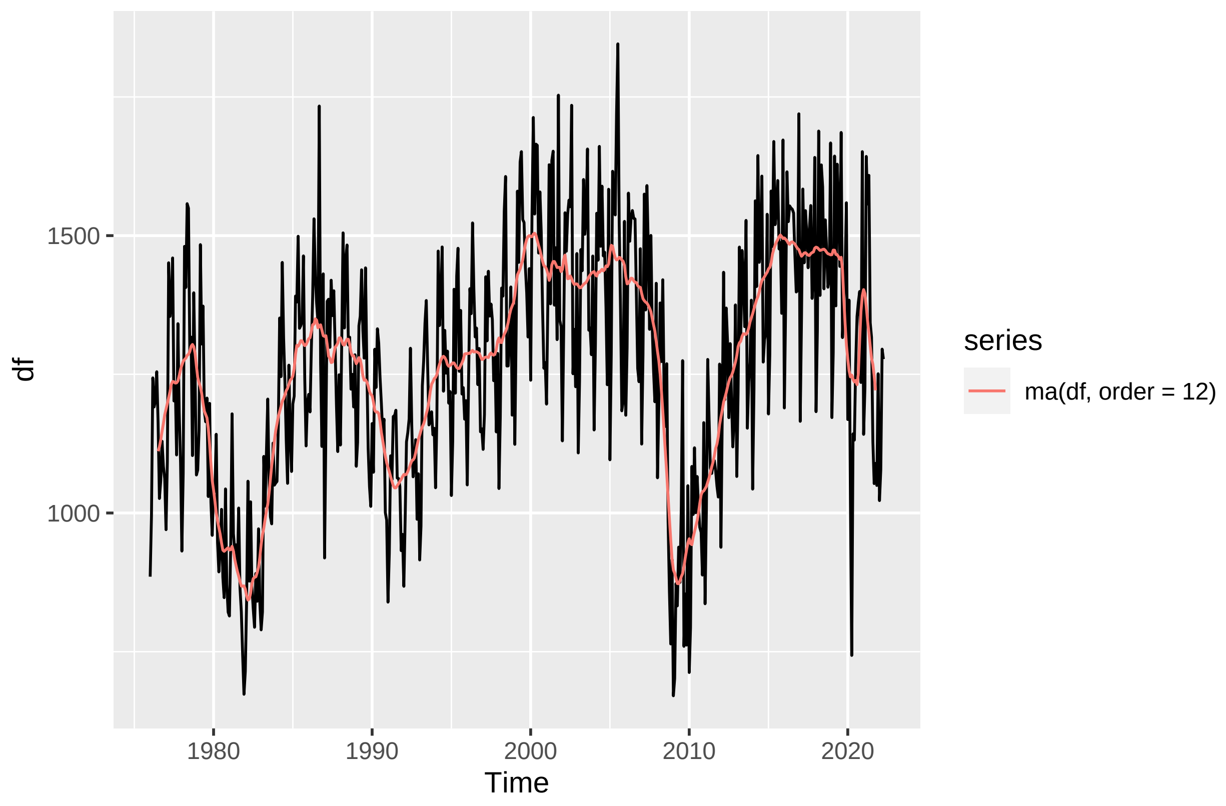
Timeseries make-up: Trend, Seasonality and Noise

```
df = read.csv("../../data/TOTALNSA.csv")
df[['DATE']] = as.Date(df[['DATE']], format='%Y-%m-%d')
df <- ts(df$TOTALNSA, start = '1976', freq=12)
plot(decompose(df))
```



# Stripping away seasonality using Moving Averages The underlying pattern in a timeseries is usually the key to forecasting it. The simplest way of extracting this underlying pattern is by using a moving average (MA). To use an MA you have to specify the number of data points (or the window) to average over, and below we simply use a 12month (backward looking) average, plotted for every point of the timeseries, which also has the effect of removing the seasonality:

```
autoplot(df) + autolayer(ma(df, order = 12), colour = TRUE)
```

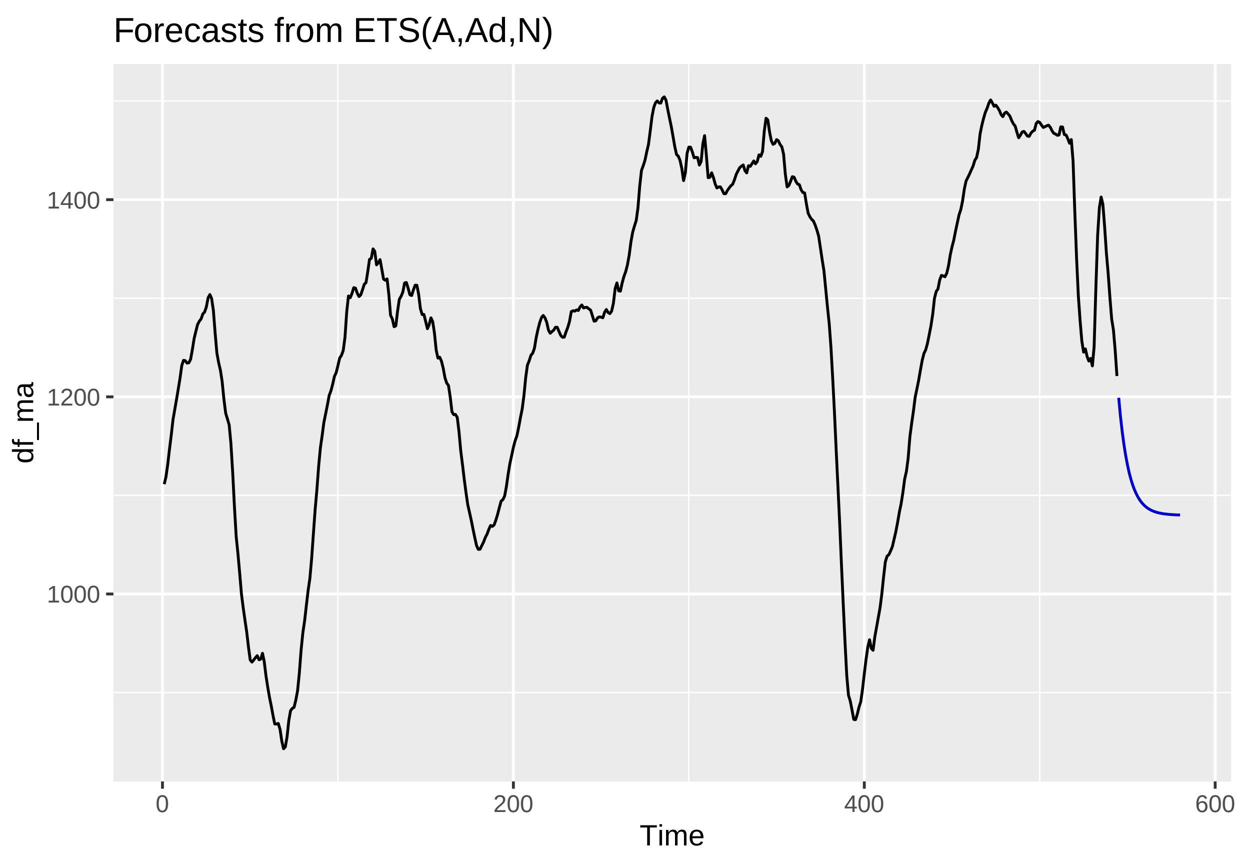


Simple forecasting: Extrapolation

The simplest form of forecasting is to assume the same value continues to occur in the future. Next most simple is straight line (or linear) forecast, extending a straight line fitted to the most recent datapoints into the future.

- parameter h controls how far to extrapolate

```
df_ma <- ma(df, order = 12)
df_ma <- ts(na.exclude(df_ma))
myforecast <- forecast(df_ma, level = c(0), h=36)
autoplot(myforecast)
```

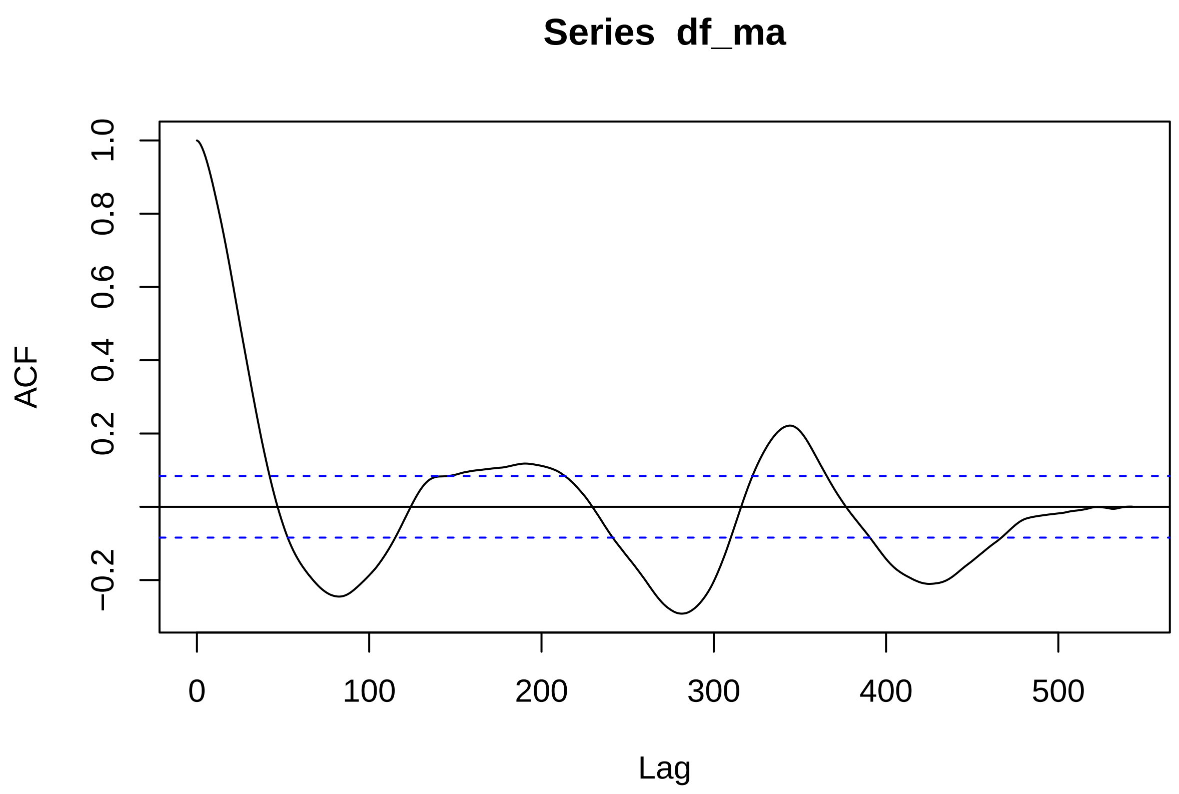


Time dependence (Autocorrelation)

In many timeseries, the value tomorrow will be dependent on the value we see today (and perhaps values in the past too). This time dependency (or temporal dependency) can be represented by autocorrelation (or self-correlation), which is exploited by autoregressive (AR) models, that simply weight past data points to provide a forecast for future data points.

- The function acf computes estimates of autocorrelation.

```
acf_obs = acf(df_ma, lag.max = length(df_ma), plot = FALSE)
plot(acf_obs, type = "l")
```



Forecasting using Autoregression (AR)

- You may want to have students write a loop to try different levels of lags (order.max parameter) and assess performance.

```
ar(df_ma,order.max = 1)
```

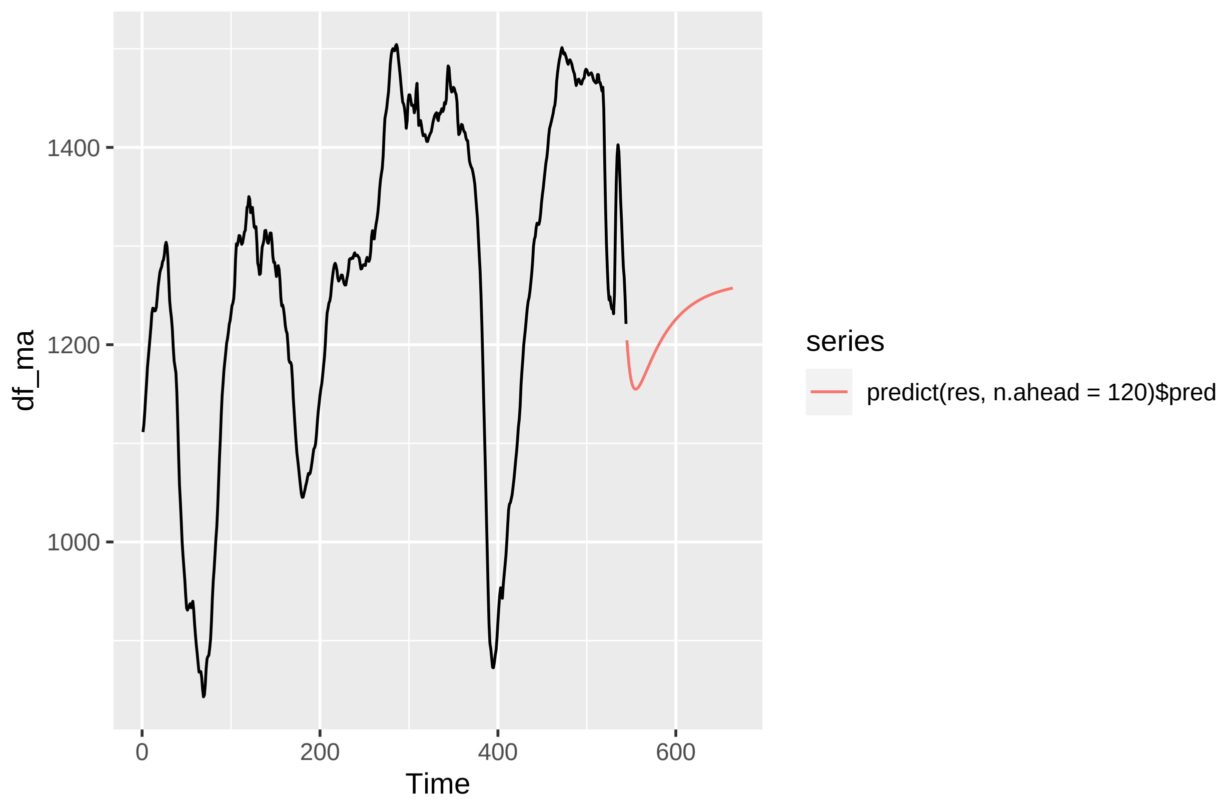
```
##
## Call:
## ar(x = df_ma, order.max = 1)
##
## Coefficients:
##     1
## 0.997
##
## Order selected 1  sigma^2 estimated as  187
```

- The following selects the best value of lag and provides a 120 period forecast.The number of lags is selected based on AIC measure.

```
res = ar(df_ma,aic=T,order.max = 12)
res
```

```
##
## Call:
## ar(x = df_ma, aic = T, order.max = 12)
##
## Coefficients:
##       1        2        3        4        5
##   1.511   -0.464   0.030   -0.020   -0.063
##
## Order selected 5  sigma^2 estimated as  115
```

```
pred_ar <- predict(res, n.ahead = 120)
autoplot(df_ma) +  autolayer(predict(res, n.ahead = 120)$pred)
```



Detrending

In many cases the trend is a distraction and, in some, downright misleading for timeseries forecasting models. To remove the distraction of the trend we need to detrend our series. The simplest detrending approach is differencing, where we subtract each observation from the observation before (e.g., use this on vehicle sales), or divide each by the observation before (e.g., do for prices, where the change is the price return).

```
# Detrending
s1 = df[1:length(df)-1]
s2 = df[2:length(df)]
```

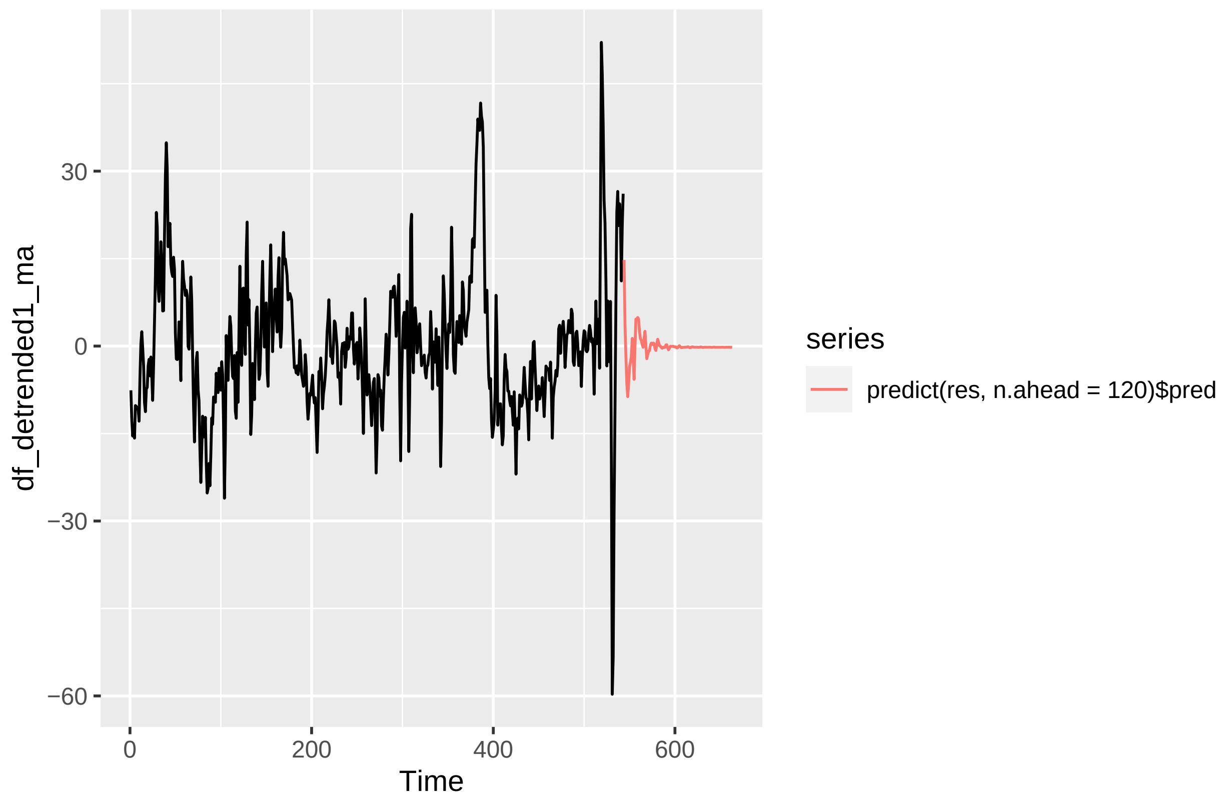
```r
df_detrended1 <- ts(s1-s2,frequency = 12)
df_detrended2 <- ts(s1/s2,frequency = 12)
```

- Ask students to create a detrended AR model.

```r
df_detrended1_ma <- ma(df_detrended1, order = 12)
df_detrended1_ma <- ts(na.exclude(df_detrended1_ma))
res = ar(df_detrended1_ma,aic=T,order.max = 20)
res
```

```
##
## Call:
## ar(x = df_detrended1_ma, aic = T, order.max = 20)
##
## Coefficients:
##       1        2        3        4        5        6        7        8        9       10
##   1.322   -0.975    0.933   -0.763    0.609   -0.414    0.303   -0.263    0.309   -0.298
##      11       12       13       14
##   0.195   -0.420    0.432   -0.081
##
## Order selected 14  sigma^2 estimated as  26.3
```

```r
pred_ar <- predict(res, n.ahead = 120)
autoplot(df_detrended1_ma) +  autolayer(predict(res, n.ahead = 120)$pred)
```



ARIMA

ARIMA model specification: (p,d,q) are the AR order, the degree of differencing, and the MA order.

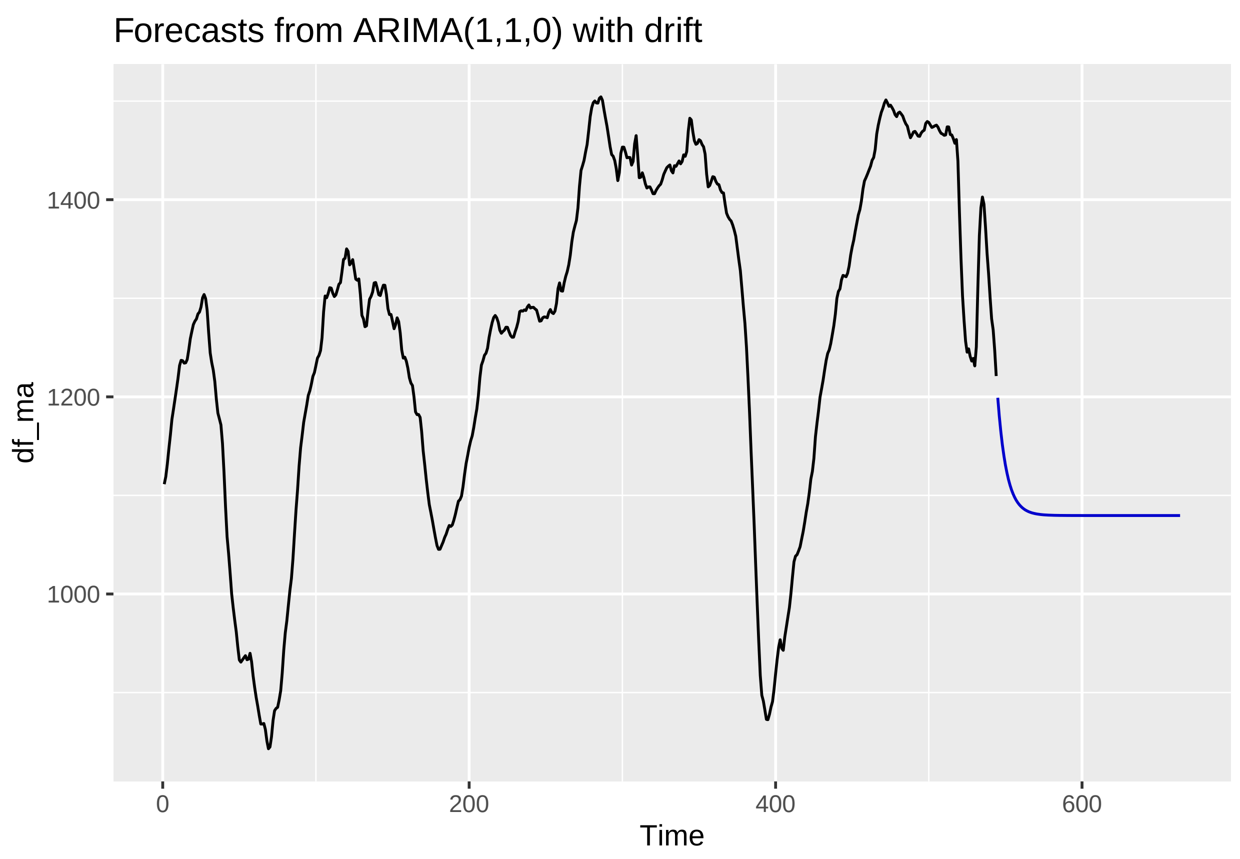
```
arima(df_ma, order = c(1,1,2))
```

```
##
## Call:
## arima(x = df_ma, order = c(1, 1, 2))
##
## Coefficients:
##          ar1    ma1     ma2
##        0.841  0.618  -0.382
## s.e.  0.038  0.069   0.069
##
## sigma^2 estimated as 25.4:  log likelihood = -1652,  aic = 3312
```

```
mod = auto.arima(df_ma,ic = "aic")
print(mod)
```

```
## Series: df_ma
## ARIMA(1,1,0) with drift
##
## Coefficients:
##          ar1  drift
##        0.844  0.000
## s.e.  0.023  1.745
##
## sigma^2 = 41.1:  log likelihood = -1779
## AIC=3564   AICc=3564   BIC=3576
```

```
autoplot(forecast(mod, h = 120,level = 0))
```



```
accuracy(mod)
```

```
##                      ME  RMSE   MAE      MPE   MAPE   MASE   ACF1
## Training set -0.01357 6.391 4.547 0.005531 0.3665 0.5355 0.2366
```

- Have students build an ARIMA model on the df original dataset.