Univariate Forecasts for extreme_heat and extreme_cold

In this part we need to evaluate univariate forecasts of the temperatures for the multiple linear regression. We will thus need to use the predicted values : - of min_temp_val and heat_days_val from February 2017 to July 2017 - of extreme_heat and extreme_cold from March 2017 to July 2017. Here, we will only predict extreme_heat and extreme_cold. In another notebook, we predicted min_temp_val and heat_days_val.

Load the data

```
library(readr)
google <- na.omit(read_csv("data/google.csv", col_types = cols(date = col_date(format = "%Y-%m-%d"))))
temp <- na.omit(read_csv("data/temp.csv", col_types = cols(date = col_date(format = "%Y-%m-%d"))))
rmse <- function(l, r) {
  sqrt(sum((l - r)^2)/NROW(l))
}
exp_rmse <- function(l, r) {
  rmse(exp(l), exp(r))
}
```

Variable creations :

```
# extreme_heat will be 1 when heatwave > mean(heatwave)
extreme_heat <- as.numeric(google$heatwave > mean(google$heatwave))
# extreme_cold will be 1 when snow_storm > mean(snow_storm) and extreme_weather > mean(extreme_weather)
extreme_cold <- as.numeric(google$snow_storm > mean(google$snow_storm) & google$extreme_weather > mean(g
```

extreme_heat

Stationarity and integration of the extreme_heat

```
library(forecast)
```

```
## Loading required package: zoo
```

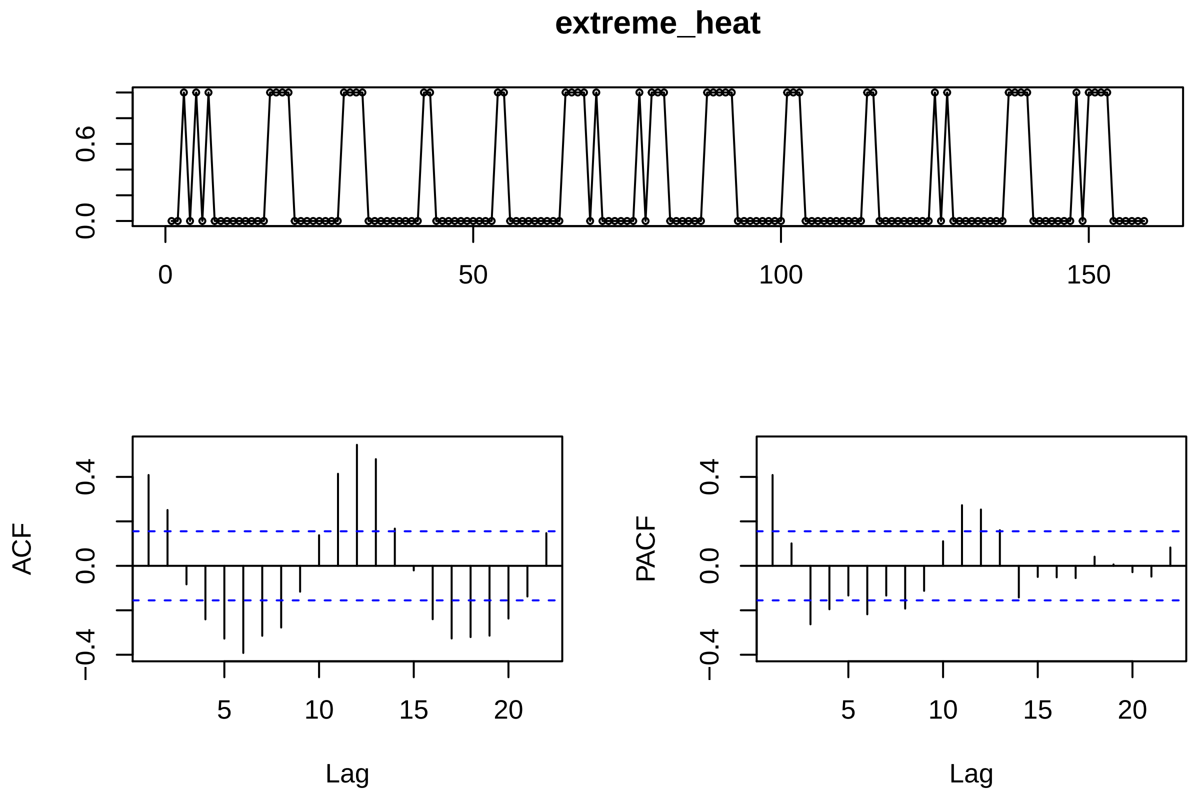
```
##
## Attaching package: 'zoo'
```

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

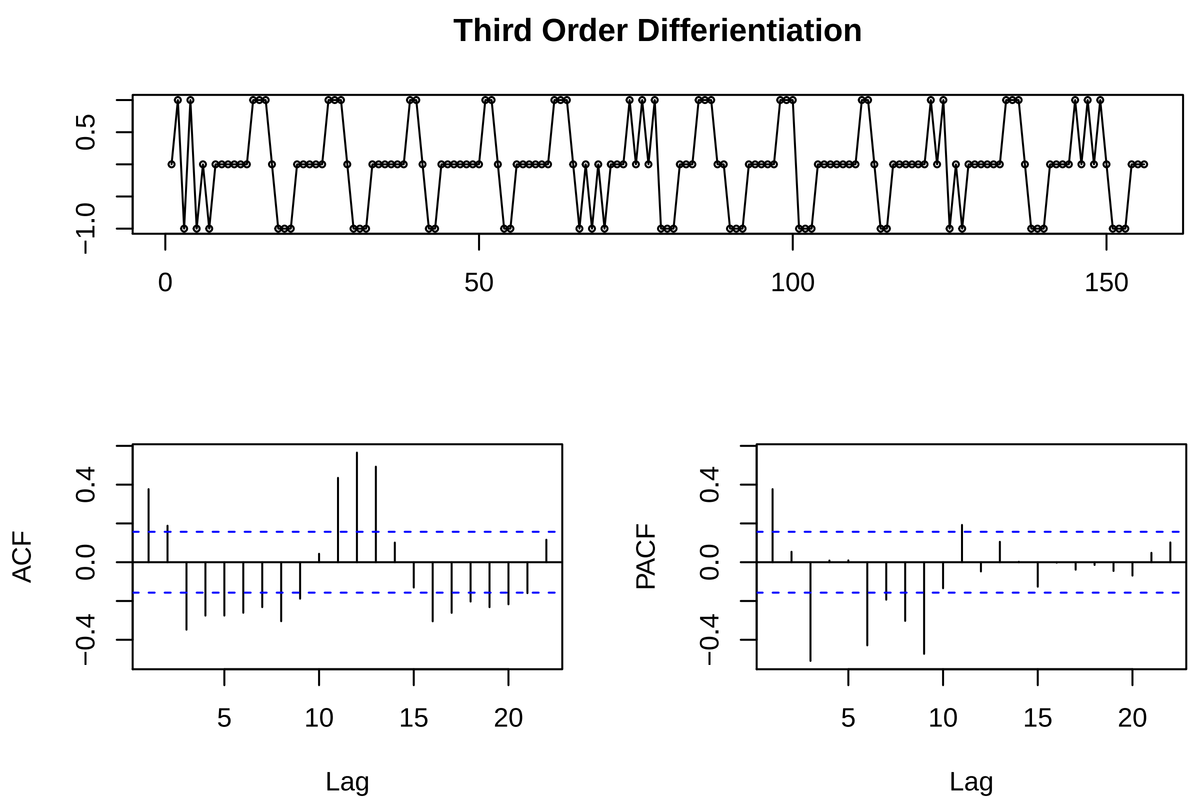
```
## Loading required package: timeDate
```

```
## This is forecast 7.3
```

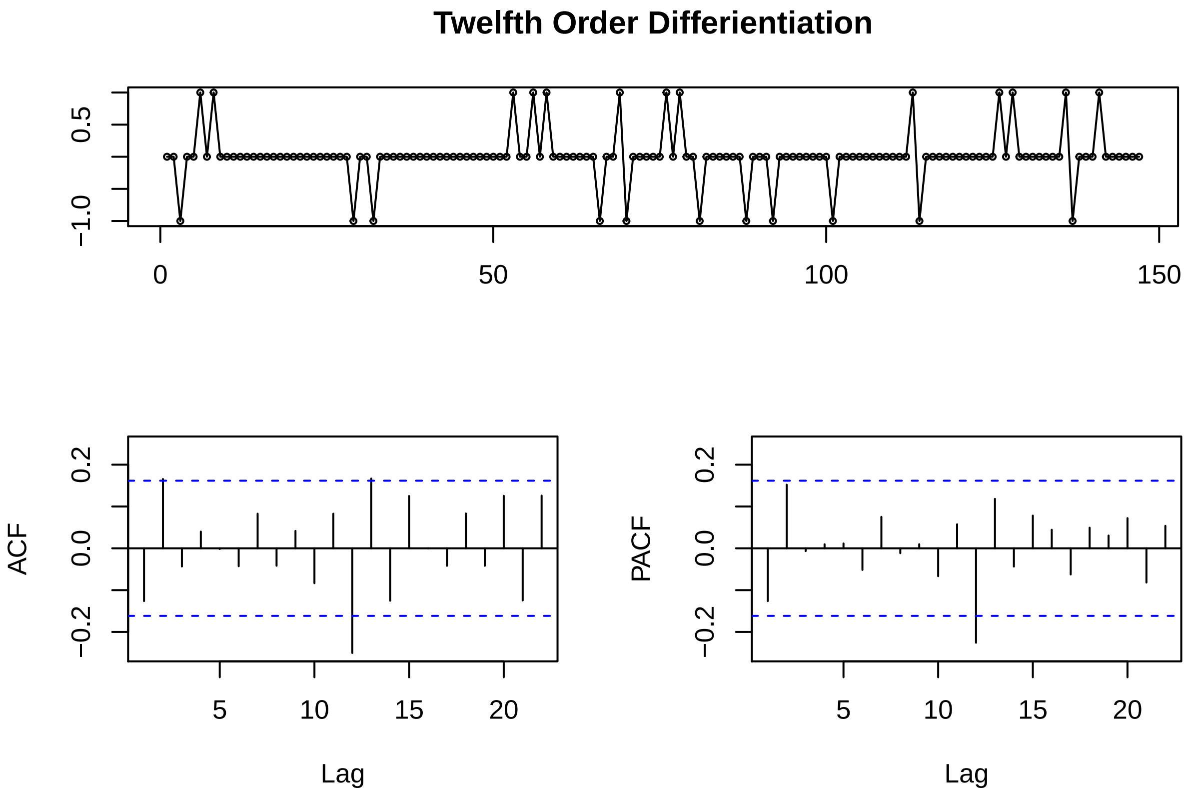
```
tsdisplay(extreme_heat,main="extreme_heat")
```




```r
tsdisplay(diff(extreme_heat, 3), main="Third Order Differientiation")
```



```
tsdisplay(diff(extreme_heat, 12), main="Twelfth Order Differientiation")
```



As we can see, we need to do a 12th order differenciation in order to remove the seasonality. But we almost obtain a white noise !

Models evaluation

Let us convert data to a time series

```
sextreme_heat = ts(extreme_heat, start = c(2004, 1), frequency=12)
smin_temp_val = ts(temp$min_temp_val, start = c(1895, 1), frequency=12)
train = window(smin_temp_val, start = c(2004, 1), end = c(2016, 2))
test = window(smin_temp_val, start = c(2016,3), end = c(2017, 2))
```

Simple Snaive

We first start with a simple seasonal naive process (ie. repetition of the last temporality).

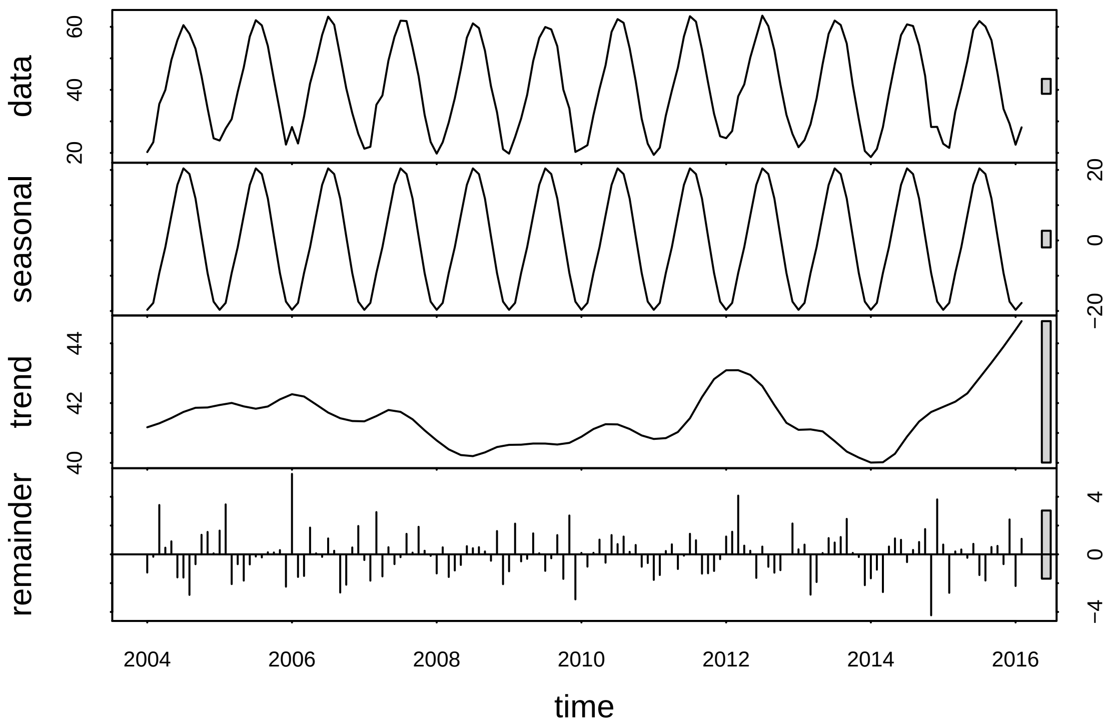
```
snaive = snaive(train, h = 12)
accuracy(snaive, test)
```

```
##                     ME     RMSE      MAE        MPE     MAPE      MASE
## Training set 0.1489552 2.631078 1.988806 0.02203069 6.184159 1.0000000
## Test set     0.2100000 2.221640 1.606667 0.09808236 5.155357 0.8078549
##                    ACF1 Theil's U
## Training set  0.2007327        NA
## Test set     -0.3509174 0.3442317
```

Time Series Decomposition with min_temp_val

Now for Time Series decomposition.

```
decomp = stl(train, s.window="periodic")
plot(decomp)
```



As previously, our first attempt will just forecast the time series by removing seasonality, and then using the last observation to which we add back the seasonality as the next forecast value.

```
stlfk_naive = forecast(decomp, method="naive", h=12)
summary(stlfk_naive)
```

```
##
## Forecast method: STL +  Random walk
##
## Model Information:
## $drift
## [1] 0
##
## $drift.se
## [1] 0
##
## $sd
## [1] 2.268991
##
## $call
## rwf(y = x, h = h, drift = FALSE, level = level)
##
##
## Error measures:
##                         ME      RMSE      MAE        MPE      MAPE      MASE
```

```
## Training set 0.04064998 2.261518 1.675095 -0.1862235 5.137839 0.8422618
##                    ACF1
## Training set -0.5119057
##
## Forecasts:
##          Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
## Mar 2016       36.59956 33.70130 39.49781 32.16706 41.03205
## Apr 2016       43.91371 41.01546 46.81196 39.48121 48.34620
## May 2016       52.85453 49.95628 55.75278 48.42204 57.28702
## Jun 2016       61.57856 58.68030 64.47681 57.14606 66.01105
## Jul 2016       66.25925 63.36100 69.15750 61.82675 70.69174
## Aug 2016       64.64313 61.74488 67.54138 60.21063 69.07562
## Sep 2016       57.69867 54.80042 60.59692 53.26618 62.13117
## Oct 2016       46.98234 44.08409 49.88059 42.54984 51.41483
## Nov 2016       36.57017 33.67192 39.46843 32.13768 41.00267
## Dec 2016       28.47479 25.57654 31.37304 24.04229 32.90728
## Jan 2017       26.15425 23.25600 29.05250 21.72175 30.58674
## Feb 2017       28.11000 25.21175 31.00825 23.67751 32.54249
```

```r
accuracy(stlfk_naive, test)
```

```
##                       ME     RMSE      MAE        MPE     MAPE      MASE
## Training set  0.04064998 2.261518 1.675095 -0.1862235 5.137839 0.8422618
## Test set     -2.37657891 3.067990 2.724912 -5.7553539 6.908775 1.3701247
##                     ACF1 Theil's U
## Training set -0.51190568        NA
## Test set      0.02845487 0.4273305
```

Let's try with exponential smoothing to forecast the seasonally-adjusted series.

```r
stlfk_ets = forecast(decomp, method="ets", h=12)
summary(stlfk_ets)
```

```
##
## Forecast method: STL +  ETS(M,N,N)
##
## Model Information:
## ETS(M,N,N)
##
## Call:
##  ets(y = x, model = etsmodel, allow.multiplicative.trend = allow.multiplicative.trend)
##
##   Smoothing parameters:
##     alpha = 0.2071
##
##   Initial states:
##     l = 41.5325
##
##   sigma:  0.0428
##
##      AIC     AICc      BIC
## 900.6789 900.8479 909.6297
##
## Error measures:
##                      ME     RMSE      MAE       MPE     MAPE      MASE
## Training set 0.07537092 1.775772 1.393616 -0.145014 4.219744 0.7007299
```

```
##                       ACF1
## Training set 0.0005204832
##
## Forecasts:
##          Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
## Mar 2016       34.59161 32.18814 36.99508 30.91583 38.26739
## Apr 2016       41.90576 39.45118 44.36035 38.15181 45.65972
## May 2016       50.84658 48.34193 53.35124 47.01604 54.67713
## Jun 2016       59.57061 57.01686 62.12437 55.66498 63.47624
## Jul 2016       64.25130 61.64937 66.85323 60.27199 68.23061
## Aug 2016       62.63518 59.98595 65.28442 58.58353 66.68684
## Sep 2016       55.69073 52.99502 58.38644 51.56799 59.81346
## Oct 2016       44.97439 42.23299 47.71580 40.78178 49.16701
## Nov 2016       34.56223 31.77588 37.34858 30.30087 38.82358
## Dec 2016       26.46684 23.63626 29.29743 22.13784 30.79585
## Jan 2017       24.14630 21.27216 27.02045 19.75068 28.54193
## Feb 2017       26.10205 23.18500 29.01911 21.64080 30.56331
```

```r
accuracy(stlfk_ets, test)
```

```
##                       ME     RMSE      MAE        MPE     MAPE      MASE
## Training set  0.07537092 1.775772 1.393616 -0.1450140 4.219744 0.7007299
## Test set     -0.36863382 1.974925 1.632535 -0.6064302 4.473935 0.8208621
##                       ACF1 Theil's U
## Training set 0.0005204832        NA
## Test set     0.0284548714 0.3465735
```

As a last attempt, we can also use arima on the seasonally-adjusted data.

```r
stlfk_arima = forecast(decomp, method="arima", h=12)
summary(stlfk_arima)
```

```
##
## Forecast method: STL +  ARIMA(3,0,0) with non-zero mean
##
## Model Information:
## Series: x
## ARIMA(3,0,0) with non-zero mean
##
## Coefficients:
##          ar1     ar2     ar3  intercept
##       0.1538  0.1968  0.1561    41.5203
## s.e.  0.0827  0.0824  0.0858     0.2859
##
## sigma^2 estimated as 3.055:  log likelihood=-286.79
## AIC=583.58   AICc=584.01   BIC=598.5
##
## Error measures:
##                       ME     RMSE      MAE        MPE     MAPE      MASE
## Training set 0.004968654 1.723831 1.340291 -0.3275602 4.107519 0.6739173
##                     ACF1
## Training set 0.005010114
##
## Forecasts:
##          Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
## Mar 2016       33.89807 31.65799 36.13815 30.47217 37.32397
```

```
## Apr 2016       40.82061 38.55420 43.08702 37.35444 44.28678
## May 2016       49.72612 47.40656 52.04569 46.17866 53.27359
## Jun 2016       57.94592 55.57446 60.31738 54.31908 61.57275
## Jul 2016       62.48099 60.09871 64.86327 58.83760 66.12437
## Aug 2016       60.73775 58.34632 63.12919 57.08037 64.39513
## Sep 2016       53.66638 51.27000 56.06275 50.00144 57.33132
## Oct 2016       42.88278 40.48431 45.28125 39.21464 46.55092
## Nov 2016       32.41546 30.01571 34.81520 28.74536 36.08555
## Dec 2016       24.27854 21.87815 26.67893 20.60746 27.94963
## Jan 2017       21.93026 19.52954 24.33098 18.25868 25.60184
## Feb 2017       23.86496 21.46406 26.26586 20.19311 27.53682
```

```r
accuracy(stlfk_arima, test)
```

```
##                       ME     RMSE      MAE        MPE     MAPE      MASE
## Training set 0.004968654 1.723831 1.340291 -0.3275602 4.107519 0.6739173
## Test set     1.389346252 2.530931 1.830226  4.0391596 5.337598 0.9202636
##                     ACF1 Theil's U
## Training set 0.005010114        NA
## Test set     0.147787493 0.4925082
```

R picked an ARIMA(3,0,0).

The best model for now when we consider RMSE on the Test set is the STL + ETS(M,N,N).

Seasonal ARIMA

Auto Arima on the log data

We first look at the auto.arima output.

```r
autofit = auto.arima(train, seasonal=TRUE)
print(autofit)
```

```
## Series: train
## ARIMA(3,0,0)(2,1,2)[12]
##
## Coefficients:
##          ar1     ar2     ar3     sar1     sar2     sma1    sma2
##       0.1719  0.1656  0.1365  -0.0888  -0.3363  -0.7702  0.0269
## s.e.  0.0901  0.0885  0.0904   0.3322   0.1174   0.3381  0.3359
##
## sigma^2 estimated as 3.511:  log likelihood=-279.89
## AIC=575.77   AICc=576.92   BIC=598.95
```

```r
autofk = forecast(autofit,h=12)
accuracy(autofk,test)
```

```
##                      ME     RMSE      MAE        MPE     MAPE      MASE
## Training set 0.02325838 1.747624 1.323112 -0.4383637 4.085590 0.6652793
## Test set     1.49037835 2.498759 1.808324  4.4759926 5.210906 0.9092510
##                    ACF1 Theil's U
## Training set 0.00680514        NA
## Test set     0.28742312 0.4894413
```

Let us now try with our own seasonal ARIMA. Based on the ACF PACF of the annual differentiation, we try with an ARIMA(3,0,0)(2,1,2).

Custom Seasonal Arima on the log data

```r
custfit = Arima(train, order=c(3,0,0), seasonal=c(2,1,2))
print(custfit)
```

```
## Series: train
## ARIMA(3,0,0)(2,1,2)[12]
##
## Coefficients:
##          ar1     ar2     ar3     sar1     sar2     sma1    sma2
##       0.1719  0.1656  0.1365  -0.0888  -0.3363  -0.7702  0.0269
## s.e.  0.0901  0.0885  0.0904   0.3322   0.1174   0.3381  0.3359
##
## sigma^2 estimated as 3.511:  log likelihood=-279.89
## AIC=575.77   AICc=576.92   BIC=598.95
```

```r
custfk = forecast(custfit,h=12)
accuracy(custfk,test)
```

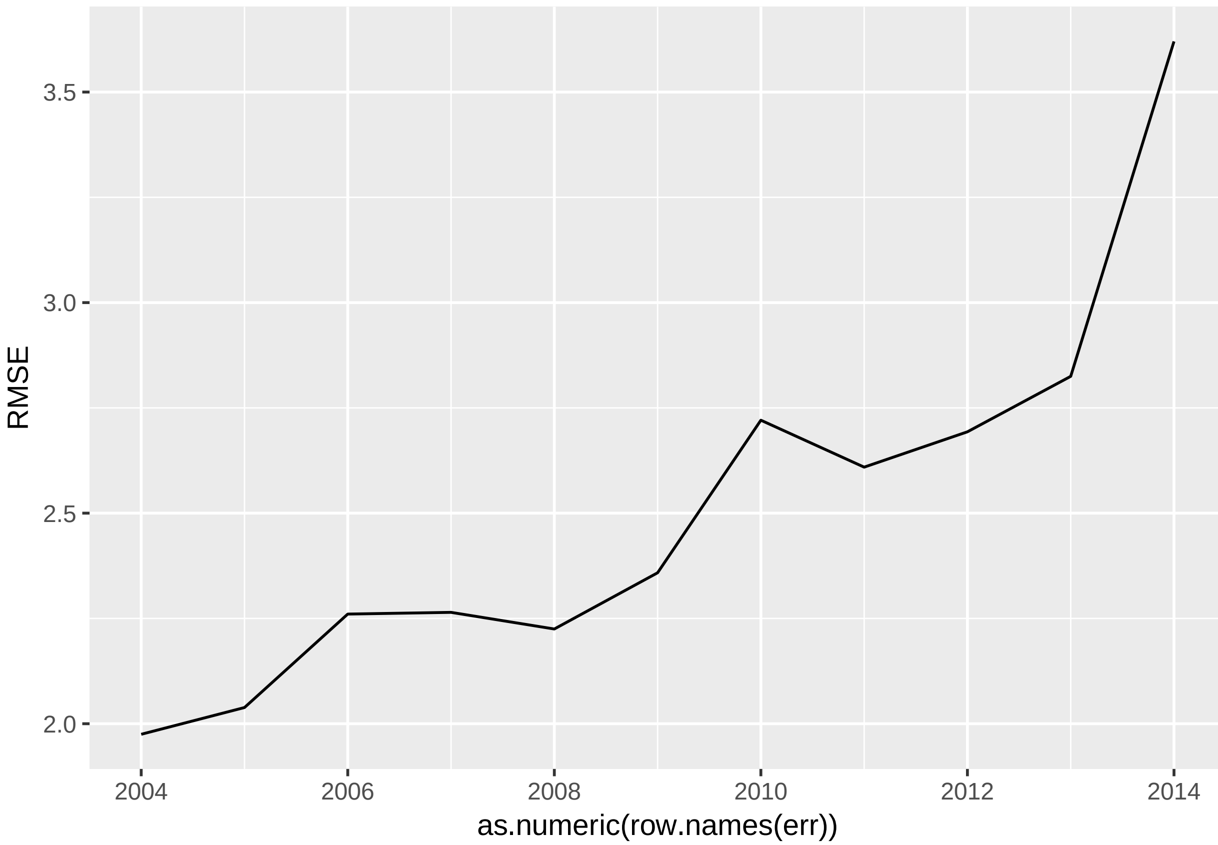
```
##                      ME     RMSE      MAE        MPE     MAPE      MASE
## Training set 0.02325838 1.747624 1.323112 -0.4383637 4.085590 0.6652793
## Test set     1.49037835 2.498759 1.808324  4.4759926 5.210906 0.9092510
##                    ACF1 Theil's U
## Training set 0.00680514        NA
## Test set     0.28742312 0.4894413
```

Although the statistical tests and criterion seem better here, the error on train and test sets are higher, so we keep the STL + ETS(M,N,N).

Rolling Window Evaluation

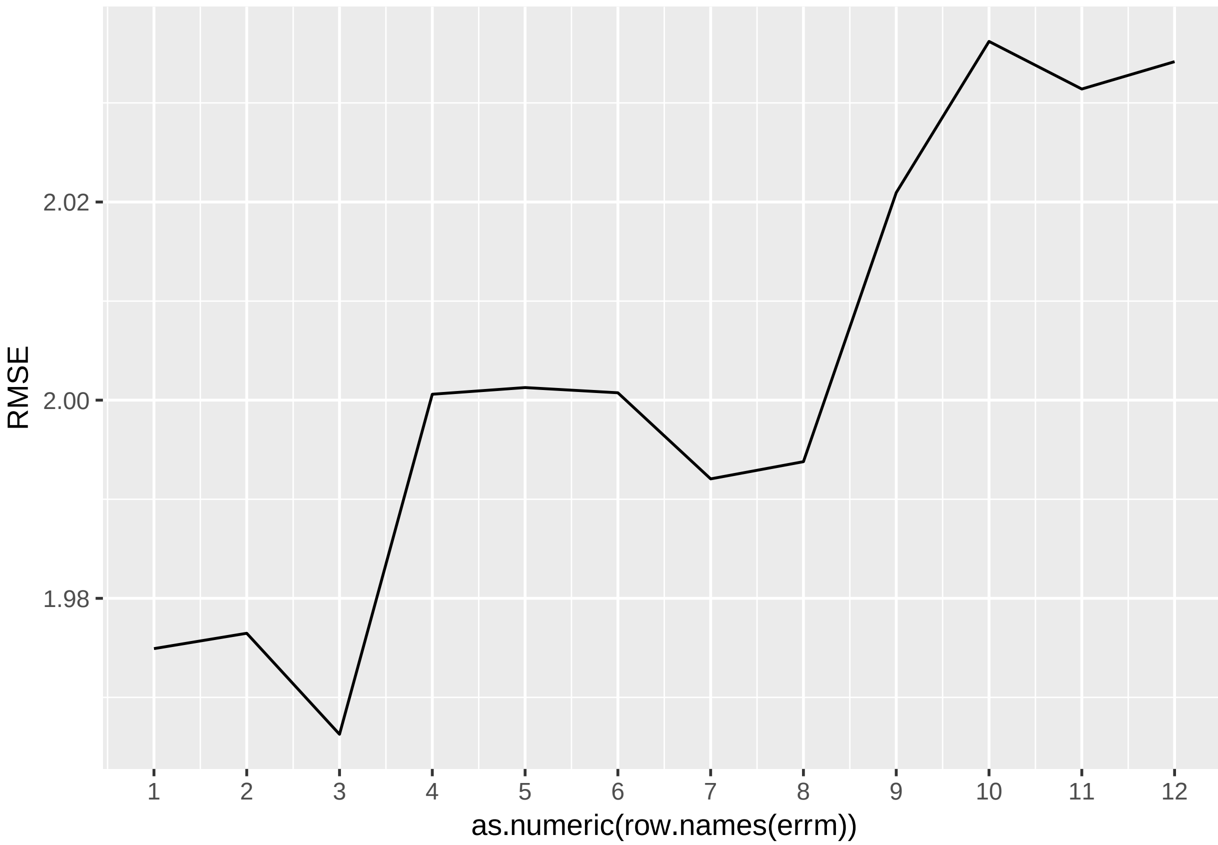
To try more model, we would like to setup a rolling window. Let us find the optimal size of that window using the STL + ETS(M,N,N) we just fitted.

```r
library(ggplot2)
err = data.frame("RMSE" = rep(0, 2014-2004+1), row.names = seq(2004, 2014))
for (y in 2004:2014){
    model = stl(window(train, start=c(y,1)), s.window="periodic")
    rmse = accuracy(forecast(model, method="ets", h=12),test)[2,"RMSE"]
    err[as.character(y),] = rmse
}
ggplot(err)+geom_line(aes(x=as.numeric(row.names(err)), y=RMSE))
```



So we should take the entire historical data ! Let's just check with the months in 2014 :

```
errm = data.frame("RMSE" = rep(0, 12), row.names = seq(1, 12))
for (m in 1:12){
    model = stl(window(train, start=c(2004,m)), s.window="periodic")
    rmse = accuracy(forecast(model, method="ets", h=12),test)[2,"RMSE"]
    errm[as.character(m),] = rmse
}
ggplot(errm)+geom_line(aes(x=as.numeric(row.names(errm)), y=RMSE))+scale_x_continuous(breaks=seq(1,12))
```

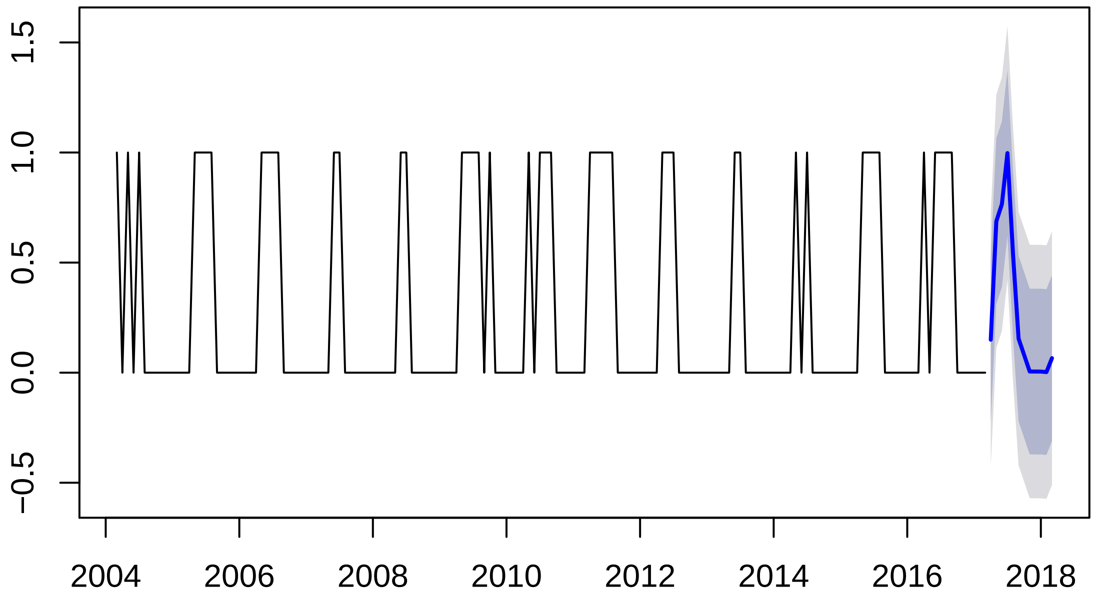


We actually should start at 2004-03 (which makes sense since we predict starting mars). We will thus use all the dataset except the 2 first months. So no need for a rolling window.

Creation of the needed forecast : from March 2017 to March 2018 (we will only use from March 2017 to July 2017).

```
train_final = window(sextreme_heat, start=c(2004,3), end=c(2017,3))
model = stl(train_final, s.window="periodic")
yearly_forecast = forecast(model, method="ets", h=12)
plot(yearly_forecast)
```

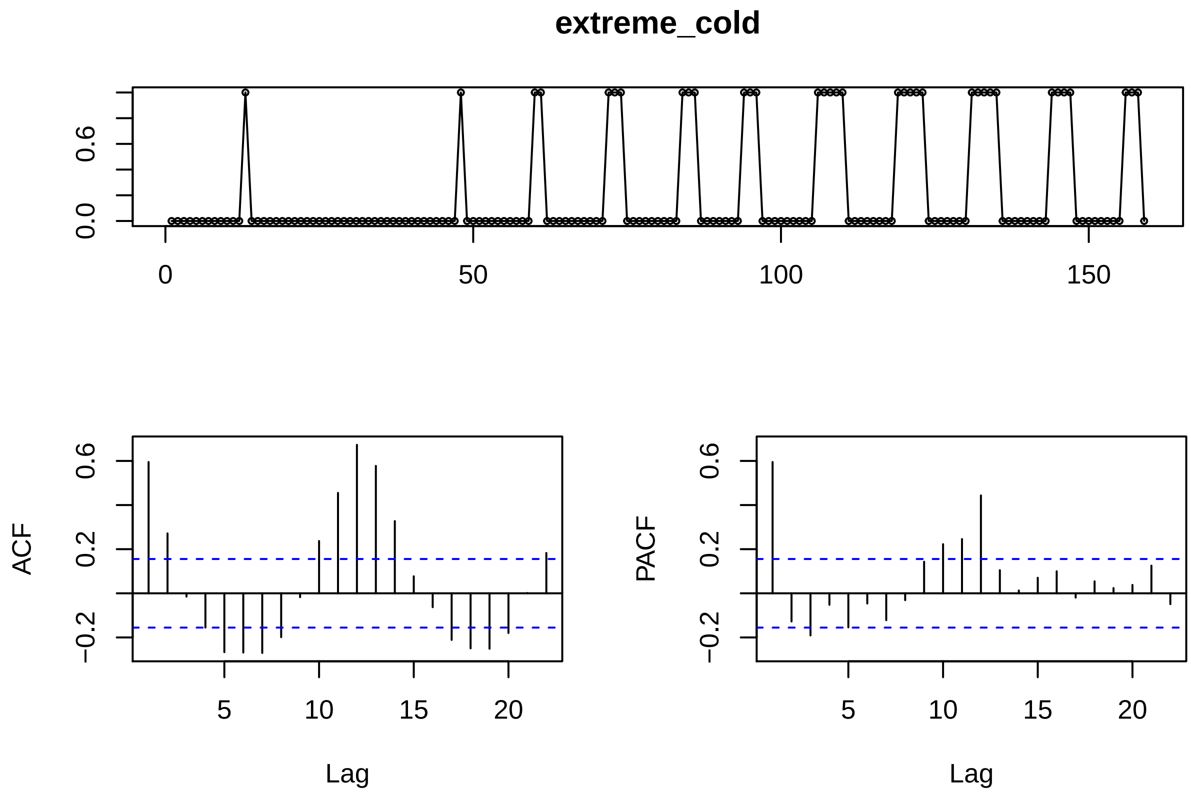
Forecasts from STL + ETS(A,N,N)



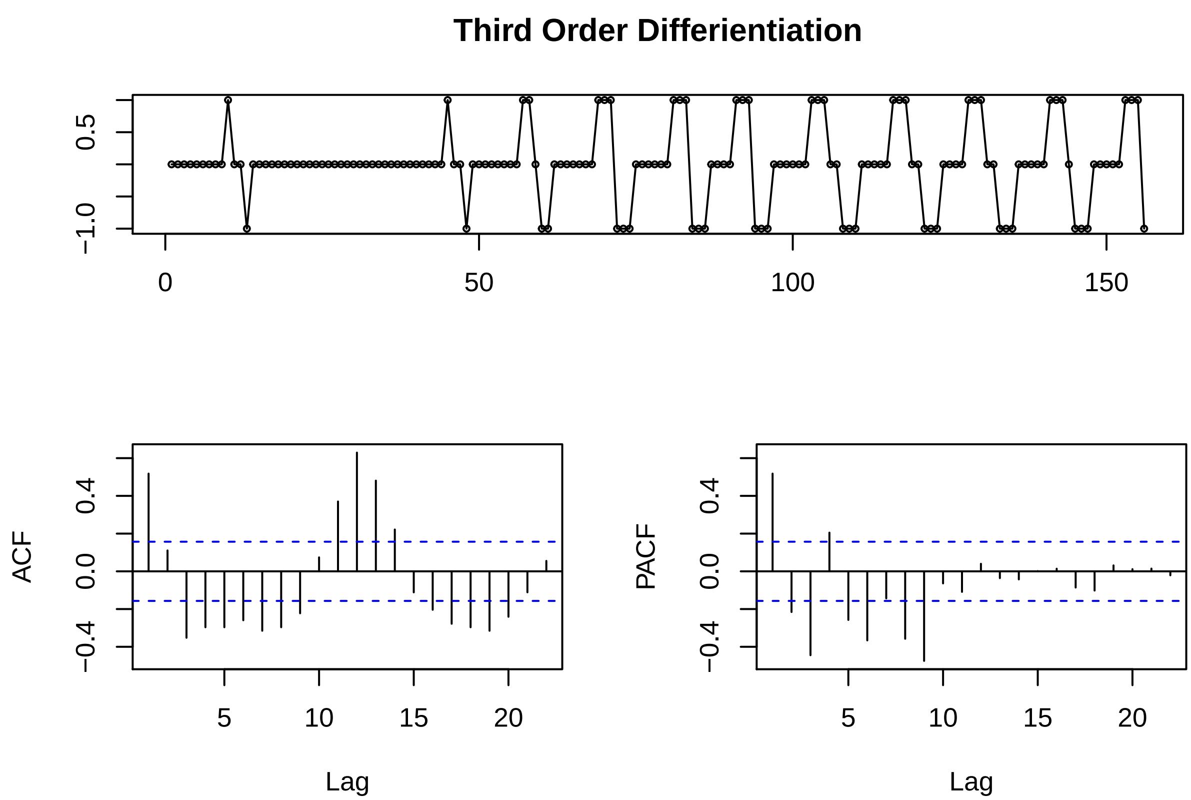
extreme_cold

Stationarity and integration of the extreme_cold

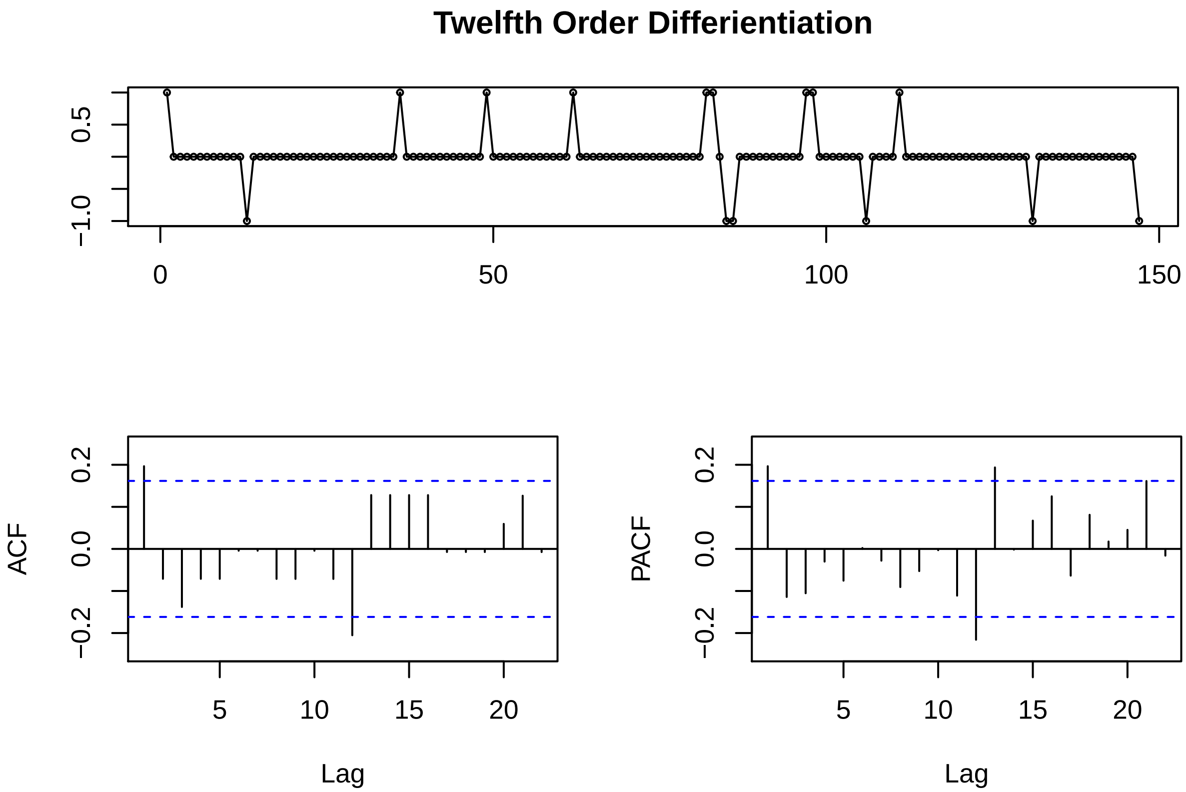
```
library(forecast)
tsdisplay(extreme_cold,main="extreme_cold")
```




```r
tsdisplay(diff(extreme_cold, 3), main="Third Order Differientiation")
```



```r
tsdisplay(diff(extreme_cold, 12), main="Twelfth Order Differientiation")
```



As previously, we need to do a 12th order differenciation in order to remove the seasonality. We notice that extreme_cold also ressembles a white noise.

Models evaluation

Let us convert data to a time series

```r
sextreme_cold = ts(extreme_cold, start = c(2004, 1), frequency=12)
train = window(sextreme_cold, start = c(2004, 1), end = c(2016, 2))
test = window(sextreme_cold, start = c(2016,3), end = c(2017, 2))
```

Simple Snaive

We first start with a simple seasonal naive process (ie. repetition of the last temporality).

```r
snaive = snaive(train, h = 12)
accuracy(snaive, test)
```

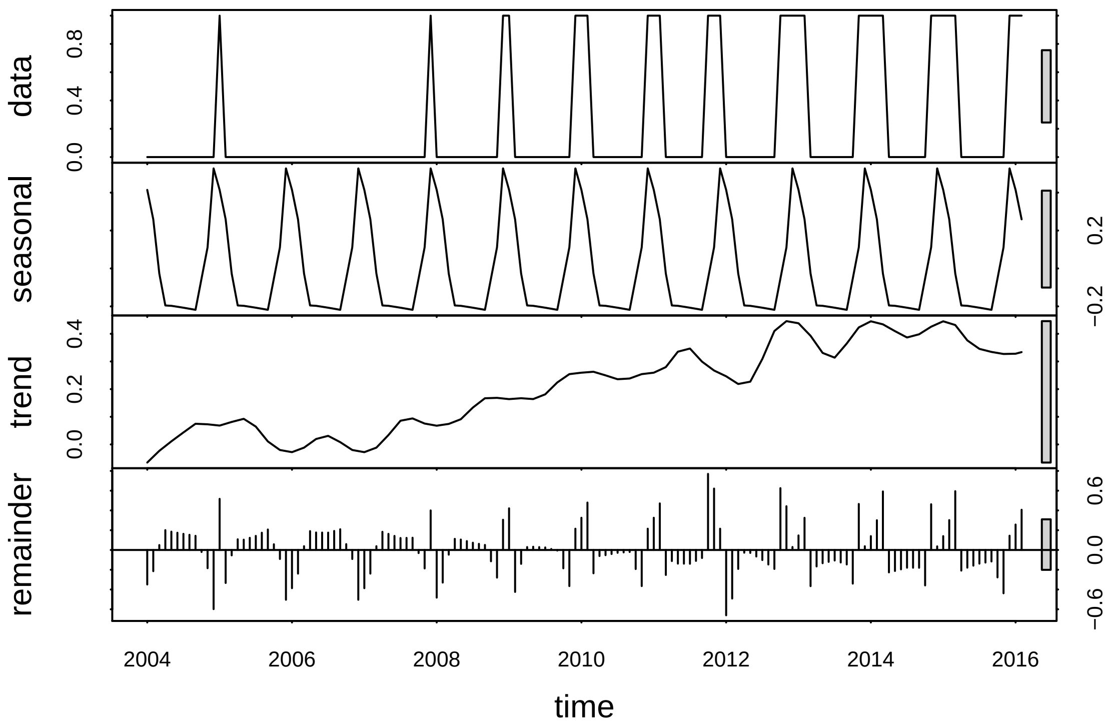
```
##                      ME    RMSE       MAE  MPE MAPE MASE      ACF1
## Training set 0.02985075 0.32323 0.1044776 -Inf  Inf    1 0.2096132
## Test set     0.00000000 0.00000 0.0000000    0    0    0       NaN
##              Theil's U
## Training set        NA
## Test set           NaN
```

We have an exact forecast !!

Time Series Decomposition with min_temp_val

Now for Time Series decomposition.

```r
decomp = stl(train, s.window="periodic")
plot(decomp)
```



As previously, our first attempt will just forecast the time series by removing seasonality, and then using the last observation to which we add back the seasonality as the next forecast value.

```r
stlfk_naive = forecast(decomp, method="naive", h=12)
summary(stlfk_naive)
```

```
##
## Forecast method: STL +  Random walk
##
## Model Information:
## $drift
## [1] 0
##
## $drift.se
## [1] 0
##
## $sd
## [1] 0.3205268
##
## $call
## rwf(y = x, h = h, drift = FALSE, level = level)
##
##
## Error measures:
##                        ME      RMSE       MAE MPE MAPE      MASE        ACF1
```

```
## Training set 0.007968842 0.319519 0.1938784 NaN  Inf 1.855694 -0.2656563
##
## Forecasts:
##          Point Forecast     Lo 80     Hi 80       Lo 95    Hi 95
## Mar 2016      0.7142187 0.3047386 1.1236987  0.08797292 1.340464
## Apr 2016      0.5457466 0.1362665 0.9552267 -0.08049914 1.171992
## May 2016      0.5439409 0.1344608 0.9534210 -0.08230482 1.170187
## Jun 2016      0.5389395 0.1294594 0.9484196 -0.08730624 1.165185
## Jul 2016      0.5339381 0.1244580 0.9434182 -0.09230766 1.160184
## Aug 2016      0.5283989 0.1189188 0.9378790 -0.09784682 1.154645
## Sep 2016      0.5228598 0.1133797 0.9323398 -0.10338598 1.149106
## Oct 2016      0.6875793 0.2780992 1.0970594  0.06133356 1.313825
## Nov 2016      0.8522986 0.4428185 1.2617787  0.22605285 1.478544
## Dec 2016      1.2688871 0.8594070 1.6783672  0.64264134 1.895133
## Jan 2017      1.1554821 0.7460020 1.5649622  0.52923636 1.781728
## Feb 2017      1.0000000 0.5905199 1.4094801  0.37375426 1.626246
```

```
accuracy(stlfk_naive, test)
```

```
##                        ME      RMSE       MAE  MPE MAPE     MASE
## Training set  0.007968842 0.3195190 0.1938784  NaN  Inf 1.855694
## Test set     -0.407690793 0.5081969 0.4553210 -Inf  Inf 4.358073
##                    ACF1 Theil's U
## Training set -0.2656563        NA
## Test set      0.1986717       NaN
```

Let's try with exponential smoothing to forecast the seasonally-adjusted series.

```
stlfk_ets = forecast(decomp, method="ets", h=12)
summary(stlfk_ets)
```

```
##
## Forecast method: STL +  ETS(A,N,N)
##
## Model Information:
## ETS(A,N,N)
##
## Call:
##  ets(y = x, model = etsmodel, allow.multiplicative.trend = allow.multiplicative.trend)
##
##   Smoothing parameters:
##     alpha = 0.4696
##
##   Initial states:
##     l = -0.2263
##
##   sigma:  0.2988
##
##      AIC     AICc      BIC
## 380.8928 381.0618 389.8436
##
## Error measures:
##                      ME     RMSE       MAE MPE MAPE    MASE      ACF1
## Training set 0.01146764 0.298817 0.2126976 NaN  Inf 2.03582 0.1313604
##
## Forecasts:
```

```
##          Point Forecast       Lo 80     Hi 80       Lo 95    Hi 95
## Mar 2016      0.5330803  0.15013091 0.9160298 -0.05259028 1.118751
## Apr 2016      0.3646083 -0.05846263 0.7876792 -0.28242284 1.011639
## May 2016      0.3628026 -0.09690135 0.8225065 -0.34025392 1.065859
## Jun 2016      0.3578012 -0.13582463 0.8514270 -0.39713435 1.112737
## Jul 2016      0.3527998 -0.17256217 0.8781617 -0.45067198 1.156272
## Aug 2016      0.3472606 -0.20802661 0.9025478 -0.50197790 1.196499
## Sep 2016      0.3417214 -0.24195878 0.9254017 -0.55094043 1.234383
## Oct 2016      0.5064410 -0.10431374 1.1171957 -0.42762776 1.440510
## Nov 2016      0.6711603  0.03448135 1.3078392 -0.30255611 1.644877
## Dec 2016      1.0877488  0.42616069 1.7493368  0.07593712 2.099560
## Jan 2017      0.9743438  0.28875098 1.6599366 -0.07417993 2.022868
## Feb 2017      0.8188617  0.11007665 1.5276467 -0.26513148 1.902855
```

```r
accuracy(stlfk_ets, test)
```

```
##                       ME      RMSE       MAE  MPE MAPE     MASE      ACF1
## Training set  0.01146764 0.2988170 0.2126976  NaN  Inf 2.035820 0.1313604
## Test set     -0.22655248 0.3786533 0.3388382 -Inf  Inf 3.243165 0.1986717
##              Theil's U
## Training set        NA
## Test set           NaN
```

As a last attempt, we can also use arima on the seasonally-adjusted data.

```r
stlfk_arima = forecast(decomp, method="arima", h=12)
summary(stlfk_arima)
```

```
##
## Forecast method: STL +  ARIMA(1,1,1)
##
## Model Information:
## Series: x
## ARIMA(1,1,1)
##
## Coefficients:
##          ar1      ma1
##       0.3874  -0.9537
## s.e.  0.0816   0.0201
##
## sigma^2 estimated as 0.07557:  log likelihood=-18.31
## AIC=42.63   AICc=42.8   BIC=51.56
##
## Error measures:
##                      ME      RMSE       MAE MPE MAPE    MASE        ACF1
## Training set 0.04008413 0.2720568 0.1965266 NaN  Inf 1.88104 0.004658666
##
## Forecasts:
##          Point Forecast       Lo 80     Hi 80       Lo 95     Hi 95
## Mar 2016      0.4888274  0.13653439 0.8411205 -0.04995828 1.0276132
## Apr 2016      0.2330489 -0.15093857 0.6170363 -0.35420924 0.8203070
## May 2016      0.1974245 -0.19390943 0.5887585 -0.40106912 0.7959182
## Jun 2016      0.1793233 -0.21465070 0.5732973 -0.42320793 0.7818545
## Jul 2016      0.1692476 -0.22618528 0.5646804 -0.43551479 0.7740100
## Aug 2016      0.1617429 -0.23478466 0.5582704 -0.44469365 0.7681794
## Sep 2016      0.1554423 -0.24205178 0.5529365 -0.45247245 0.7633571
```

```
## Oct 2016      0.3198670 -0.07854511 0.7182790 -0.28945171 0.9291856
## Nov 2016      0.4844720  0.08516169 0.8837823 -0.12622042 1.0951644
## Dec 2016      0.9010162  0.50081644 1.3012160  0.28896346 1.5130690
## Jan 2017      0.7875941  0.38650942 1.1886788  0.17418800 1.4010003
## Feb 2017      0.6321054  0.23013872 1.0340721  0.01735042 1.2468604
```

```r
accuracy(stlfk_arima, test)
```

```
##                      ME      RMSE       MAE  MPE MAPE     MASE
## Training set  0.04008413 0.2720568 0.1965266  NaN  Inf 1.881040
## Test set     -0.05917597 0.2876628 0.2575854 -Inf  Inf 2.465461
##                     ACF1 Theil's U
## Training set 0.004658666        NA
## Test set     0.213998627       NaN
```

R picked an ARIMA(1,1,1).

The best model for now the snaive forecast ! But this might not be the case for all test sets.

Seasonal ARIMA

Auto Arima on the log data

We first look at the auto.arima output.

```r
autofit = auto.arima(train, seasonal=TRUE)
print(autofit)
```

```
## Series: train
## ARIMA(4,1,1)(1,0,0)[12]
##
## Coefficients:
##          ar1      ar2      ar3      ar4      ma1    sar1
##       0.3935  -0.0489  -0.1009  -0.0595  -0.9687  0.5274
## s.e.  0.0932   0.0903   0.0904   0.0883   0.0179  0.0835
##
## sigma^2 estimated as 0.08026:  log likelihood=-22.6
## AIC=59.2   AICc=60.01   BIC=80.03
```

```r
autofk = forecast(autofit,h=12)
accuracy(autofk,test)
```

```
##                        ME      RMSE       MAE  MPE MAPE     MASE
## Training set  0.029919767 0.2764187 0.1607468  NaN  Inf 1.538577
## Test set     -0.009232543 0.2053219 0.1934145 -Inf  Inf 1.851254
##                     ACF1 Theil's U
## Training set -0.03426954        NA
## Test set      0.52015607       NaN
```

Let us now try with our own seasonal ARIMA. Based on the ACF PACF of the annual differentiation, we try with an ARIMA(4,1,1)(1,0,0).

Custom Seasonal Arima on the log data

```r
custfit = Arima(train, order=c(4,1,1), seasonal=c(1,0,0))
print(custfit)
```

```
## Series: train
## ARIMA(4,1,1)(1,0,0)[12]
##
## Coefficients:
##          ar1      ar2      ar3      ar4      ma1    sar1
##       0.3935  -0.0489  -0.1009  -0.0595  -0.9687  0.5274
## s.e.  0.0932   0.0903   0.0904   0.0883   0.0179  0.0835
##
## sigma^2 estimated as 0.08026:  log likelihood=-22.6
## AIC=59.2   AICc=60.01   BIC=80.03
```

```r
custfk = forecast(custfit,h=12)
accuracy(custfk,test)
```

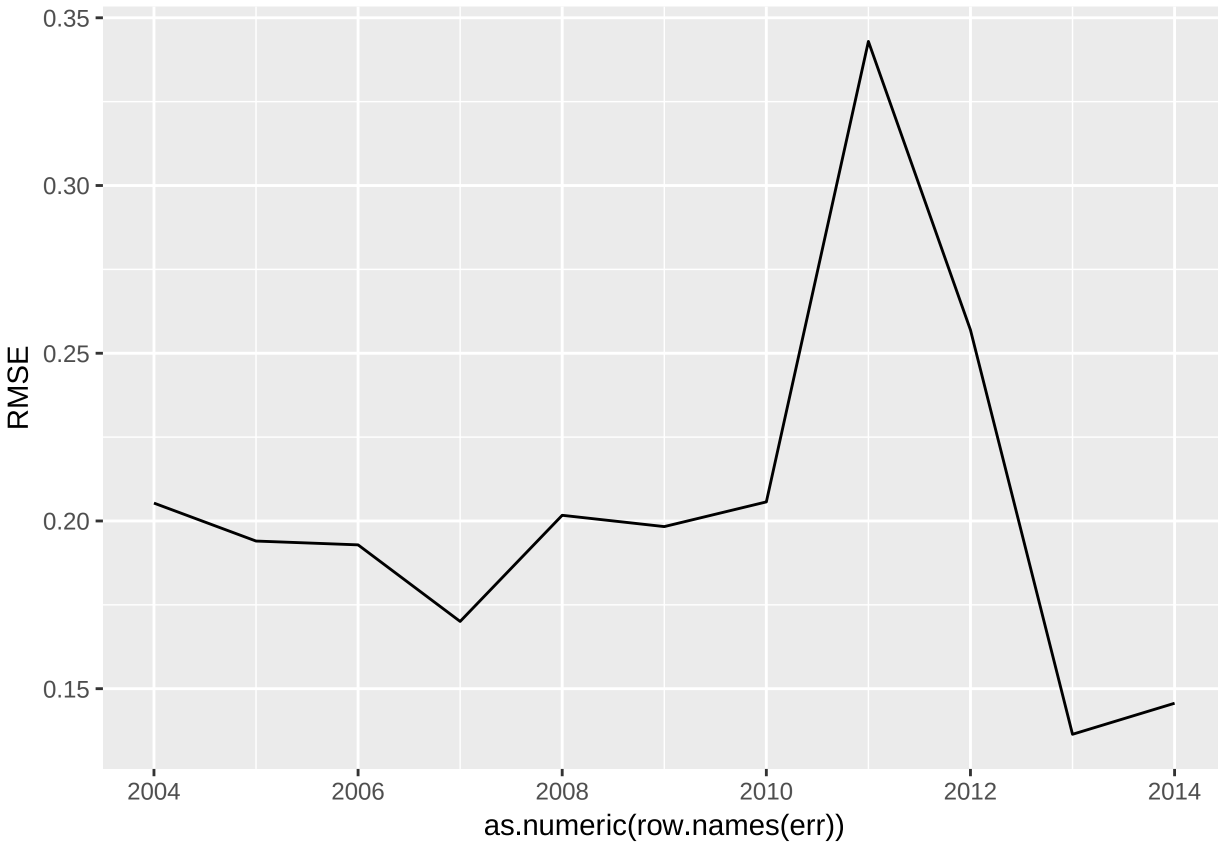
```
##                        ME      RMSE       MAE  MPE MAPE     MASE
## Training set  0.029919767 0.2764187 0.1607468  NaN  Inf 1.538577
## Test set     -0.009232543 0.2053219 0.1934145 -Inf  Inf 1.851254
##                     ACF1 Theil's U
## Training set -0.03426954        NA
## Test set      0.52015607       NaN
```

Here, we will compare snaive and the seasonal ARIMA(4,1,1)(1,0,0).

Rolling Window Evaluation

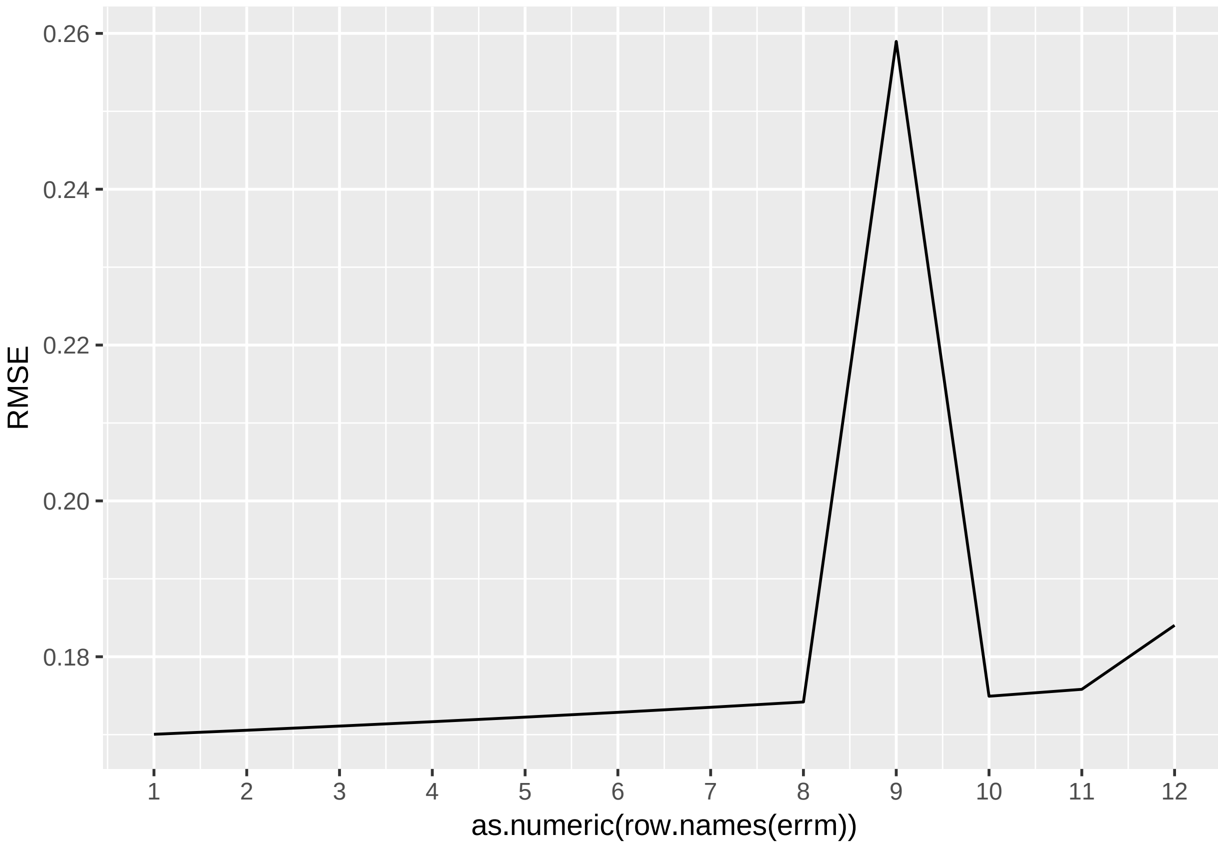
First, we would like to setup a rolling window. There is no need to do so for snaive, as it only takes into account the last year. Let us find the optimal size of that window using the seasonal ARIMA(4,1,1)(1,0,0).

```r
library(ggplot2)
err = data.frame("RMSE" = rep(0, 2014-2004+1), row.names = seq(2004, 2014))
for (y in 2004:2014){
    model = Arima(window(train, start=c(y,1)), order=c(4,1,1), seasonal=c(1,0,0))
    rmse = accuracy(forecast(model,h=12),test)[2,"RMSE"]
    err[as.character(y),] = rmse
}
ggplot(err)+geom_line(aes(x=as.numeric(row.names(err)), y=RMSE))
```



One year data is the best length ! But this is not enough for the seasonal ARIMA model selected. Let us thus take a closer look at the year 2007.

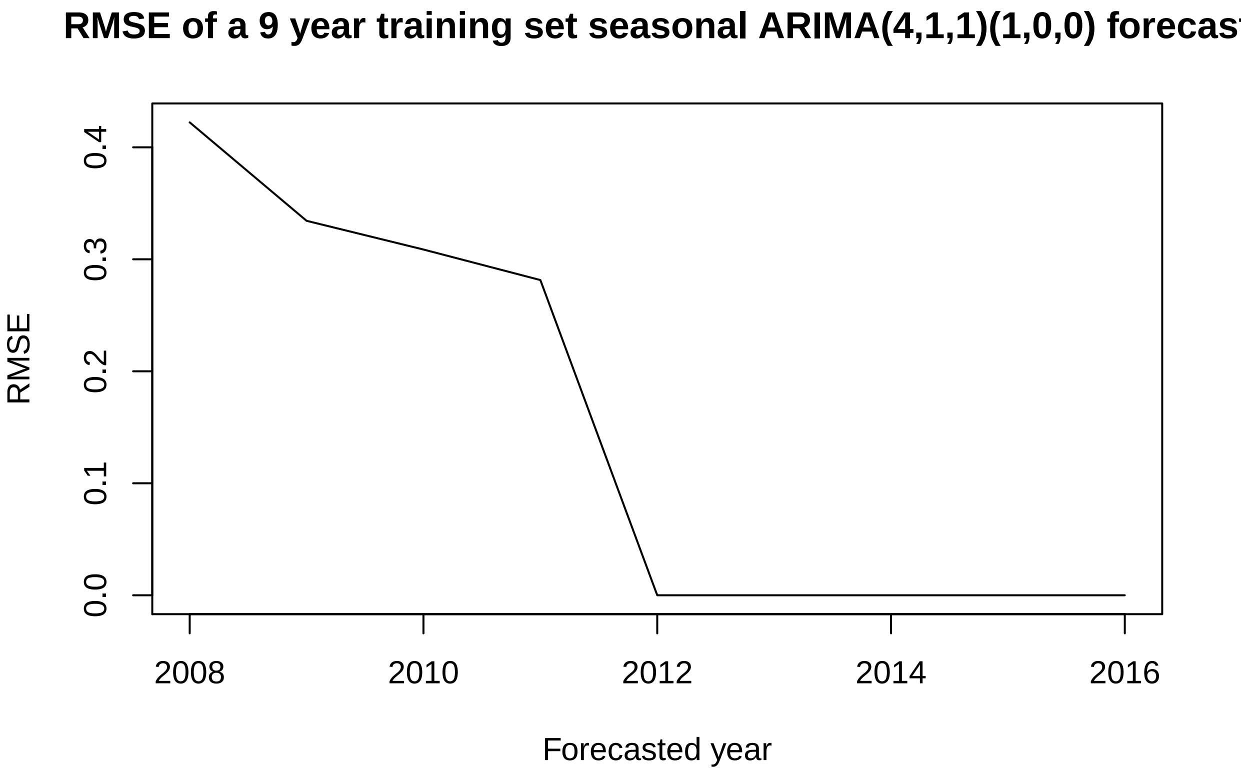
```
errm = data.frame("RMSE" = rep(0, 12), row.names = seq(1, 12))
for (m in 1:12){
    model = Arima(window(train, start=c(2007,m)), order=c(4,1,1), seasonal=c(1,0,0))
    rmse = accuracy(forecast(model,h=12),test)[2,"RMSE"]
    errm[as.character(m),] = rmse
}
ggplot(errm)+geom_line(aes(x=as.numeric(row.names(errm)), y=RMSE))+scale_x_continuous(breaks=seq(1,12))
```



We will still use a rolling window of 9 full years.

```
rollwitme = function(lgas){
    RMSE = rep(0,9)
    for (y in 2004:2007){
        tr = window(lgas, start=c(y,3), end=c(y+8,2))
        te = window(lgas, start=c(y+8,3), end=c(y+9,2))
        model = Arima(tr, order=c(4,1,1), seasonal=c(1,0,0))
        RMSE[y+1-2004] = accuracy(forecast(model,h=12),te)[2,"RMSE"]
    }
    return(RMSE)
}
```

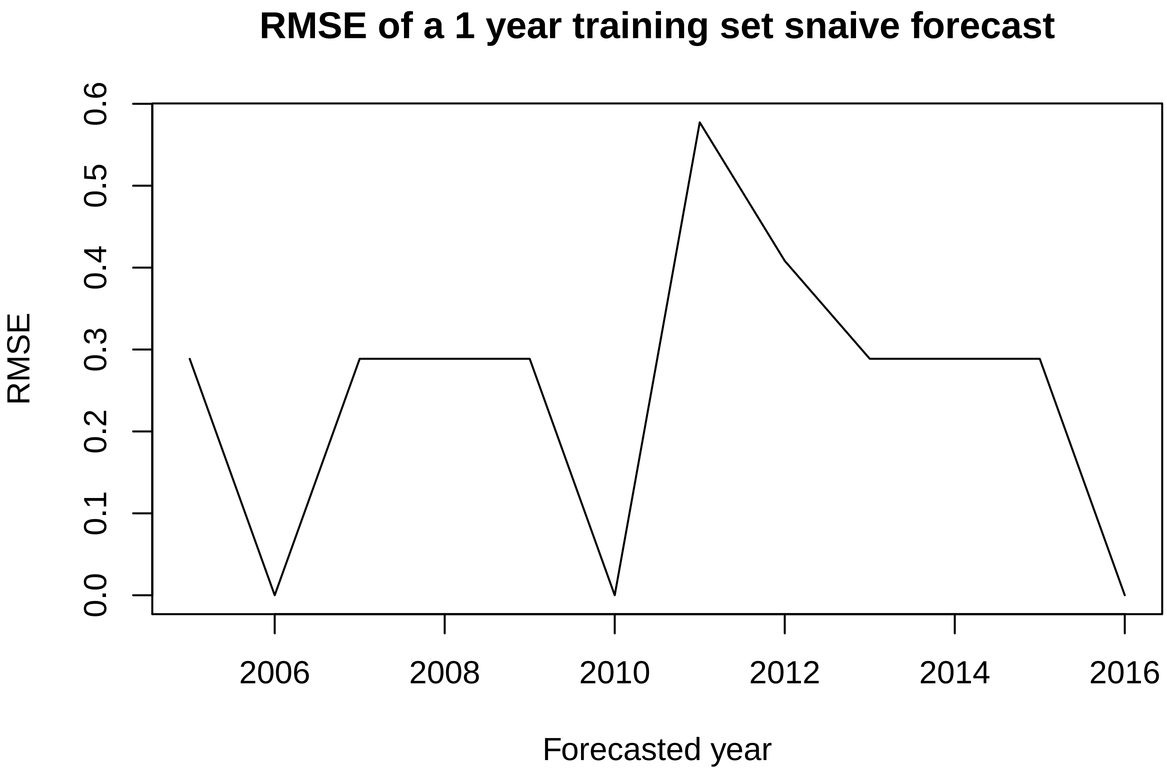
```
plot(2008:2016, rollwitme(sextreme_cold), type = 'l', main = "RMSE of a 9 year training set seasonal ARI
```



As for snaive :

```
rollwitme = function(lgas){
    RMSE = rep(0,12)
    for (y in 2004:2015){
        tr = window(lgas, start=c(y,3), end=c(y+1,2))
        te = window(lgas, start=c(y+1,3), end=c(y+2,2))
        pred = snaive(tr, h = 12)
        RMSE[y+1-2004] = accuracy(pred,te)[2,"RMSE"]
    }
    return(RMSE)
}
```

```
plot(2005:2016, rollwitme(sextreme_cold), type = 'l', main = "RMSE of a 1 year training set snaive fore
```



Clearly it's better to use the seasonal ARIMA !

Creation of the needed forecast : from March 2017 to March 2018 (we will only use from March 2017 to July 2017).

```
train_final = window(sextreme_cold, start=c(2004,3), end=c(2017,3))
model = Arima(train_final, order=c(4,1,1), seasonal=c(1,0,0))
yearly_forecast = forecast(model,h=12)
plot(yearly_forecast)
```