Regressors of Diwali on Industrial Production of India

Diwali is the most important festival of India and the timing of its distorts the monthly time-series of industrial production heavily. Generally Diwali is celebrated in the month of October according to Gregorian Calendar but that is not fixed and depending on which month it is celebrated the industrial production index also changes. The standard software packages for seasonal adjustment, `X-12-ARIMA` and `X-13-ARIMA-SEATS` (developed by the U.S. Census Bureau) or Tramo Seats (developed by the Bank of Spain) have a built-in adjustment procedure for Easter holiday, but not for Diwali. However, all packages allow for the inclusion of user defined variables, and the Chinese New Year can be modeled as such. `seasonal` (Sax and Eddelbuettel 2018) is an interface to X-13ARIMA-SEATS.

```r
rm(list = ls())
library(seasonal)
```

```
## Warning: package 'seasonal' was built under R version 4.2.3
```

```r
library(VedicDateTime)
library(insol)
library(forecast)
```

```
## Warning: package 'forecast' was built under R version 4.2.3
```

```
## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo
```

```r
library(zoo)
```

```
## Warning: package 'zoo' was built under R version 4.2.3
```

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

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

```r
data(seasonal)
data(holiday)
```

Considering Industrial Production of India after 2000

```r
industrial_prod <- window(iip, start = 2000)
```

```
## Warning in window.default(x, ...): 'start' value not changed
```

```r
industrial_prod
```

```
##           Jan      Feb      Mar      Apr      May      Jun      Jul      Aug
## 2005                               99.0838 103.0900 103.9666 102.4425 104.1004
## 2006 118.4664 112.4255 126.7159 108.8396 114.8274 114.2126 117.6044 114.2655
## 2007 134.8564 127.8147 144.8739 128.2021 136.8587 136.7408 136.6459 134.5998
## 2008 152.5200 149.3196 161.8845 142.3296 146.7452 148.3757 144.2976 141.8657
```

```
## 2009 144.3709 138.5057 153.5340 139.5905 144.2723 145.7417 146.7192 149.4225
## 2010 163.6191 157.5185 176.4743 157.8465 156.5436 156.5543 161.3000 156.1000
## 2011 175.9000 168.0000 193.1000 166.2000 166.2000 171.4000 167.2000 161.4000
## 2012 177.6000 175.2000 187.6000 164.1000 170.3000 168.0000 167.1000 164.7000
## 2013 182.0000 176.2000 194.2000 166.5000 166.0000 164.9000 171.4000 165.4000
## 2014 184.0000 172.7000 193.3000 172.7000 175.3000 172.0000 173.0000 166.2000
##           Sep      Oct      Nov      Dec
## 2005 104.4108 107.3272 104.6366 116.8191
## 2006 118.1773 117.6792 125.5368 132.7696
## 2007 133.9824 140.7213 137.9242 150.7315
## 2008 148.5866 146.1718 139.6522 148.2840
## 2009 151.0090 149.6481 148.4985 162.3779
## 2010 160.3000 166.6000 158.0000 175.6000
## 2011 164.3000 158.3000 167.5000 180.3000
## 2012 163.1000 171.6000 165.8000 179.3000
## 2013 167.5000 169.6000 163.6000 179.5000
## 2014 172.2000 162.5000 169.8000
```

Convert Gregorian dates of Diwali holidays to Vedic calendar dates

```
get_vedic_date<- function(julian_day, place) {

masa_num <- VedicDateTime::masa(julian_day, place)
vikram_samvatsara = VedicDateTime::elapsed_year(julian_day, masa_num)[5]
tithi_ = tithi(julian_day, place)[1]
masa_ = masa(julian_day, place)[1]
vedic_dates = paste(as.character(vikram_samvatsara),"-",as.character(masa_),"-",as.character(tithi_), se
return(vedic_dates)
}
```

Converted Diwali dates to vedic datetime compatible date-times

```
date<- seasonal::diwali
date <- as.POSIXct.Date(date)
julianday <- insol::JD(date)


place <- c(15.34, 75.13, +5.5) #Latitude, Longitude and timezone of the location
diwali_vedic_calendar = c()

for (i in 1:length(julianday))
{
  diwali_vedic_calendar = c(diwali_vedic_calendar, get_vedic_date(julianday[i], place))
}
diwali_vedic_calendar
```

```
##   [1] "1957-7-30" "1958-7-30" "1959-7-30" "1960-7-30" "1961-7-30" "1962-7-30"
##   [7] "1963-7-29" "1964-7-30" "1965-7-30" "1966-7-30" "1967-7-30" "1968-7-30"
##  [13] "1969-7-30" "1970-7-30" "1971-7-30" "1972-7-30" "1973-7-30" "1974-7-30"
##  [19] "1975-7-30" "1976-7-30" "1977-7-30" "1978-7-29" "1979-7-30" "1980-7-30"
##  [25] "1981-7-30" "1982-7-30" "1983-7-30" "1984-7-30" "1985-7-30" "1986-7-30"
##  [31] "1987-7-30" "1988-7-29" "1989-7-30" "1990-7-30" "1991-7-30" "1992-7-30"
##  [37] "1993-7-30" "1994-7-30" "1995-7-30" "1996-7-30" "1997-7-29" "1998-7-30"
##  [43] "1999-7-30" "2000-7-30" "2001-7-30" "2002-7-30" "2003-7-30" "2004-7-30"
```

```
##  [49] "2005-7-30" "2006-7-29" "2007-7-29" "2008-7-30" "2009-7-30" "2010-7-30"
##  [55] "2011-7-30" "2012-7-30" "2013-7-30" "2014-7-30" "2015-7-30" "2016-7-29"
##  [61] "2017-7-30" "2018-7-30" "2019-7-30" "2020-7-30" "2021-7-30" "2022-7-30"
##  [67] "2023-7-30" "2024-7-30" "2025-7-30" "2026-7-30" "2027-7-30" "2028-7-30"
##  [73] "2029-7-30" "2030-7-30" "2031-8-1"  "2032-7-30" "2033-7-30" "2034-7-30"
##  [79] "2035-7-30" "2036-7-30" "2037-7-30" "2038-7-30" "2039-7-30" "2040-7-30"
##  [85] "2041-7-30" "2042-7-30" "2043-7-29" "2044-7-30" "2045-7-30" "2046-7-30"
##  [91] "2047-7-30" "2048-7-30" "2049-7-30" "2050-7-30" "2051-7-30" "2052-7-29"
##  [97] "2053-7-29" "2054-7-29" "2055-7-29" "2056-7-29" "2057-7-29" "2058-7-29"
## [103] "2059-7-30" "2060-7-30" "2061-7-30" "2062-7-29" "2063-7-29" "2064-7-30"
## [109] "2065-7-30" "2066-7-29" "2067-7-29" "2068-7-30" "2069-7-29" "2070-7-29"
## [115] "2071-7-29" "2072-7-30" "2073-7-30" "2074-7-30" "2075-7-30" "2076-7-29"
## [121] "2077-7-29" "2078-7-30" "2079-7-29" "2080-7-29" "2081-7-30" "2082-7-30"
## [127] "2083-7-29" "2084-7-30" "2085-7-29" "2086-7-29" "2087-7-30"
```

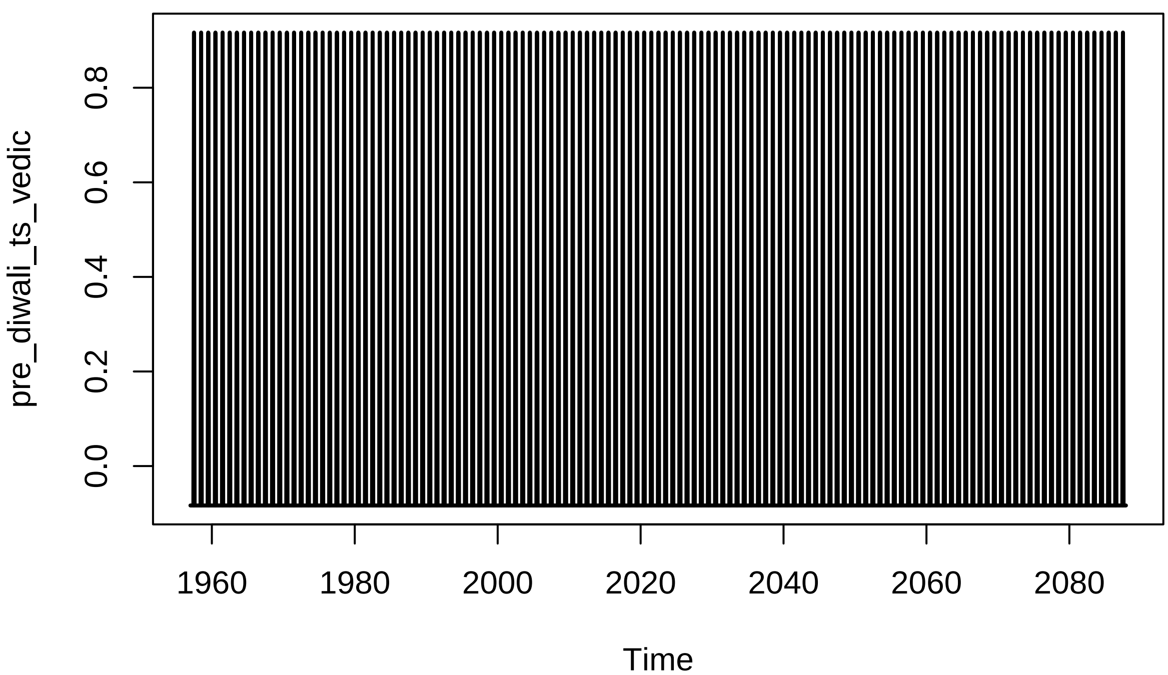
Generate time-series based on ‘genhol()‘ function using dates of Diwali as input

Generate time-series based on Vedic calendar dates of Diwali

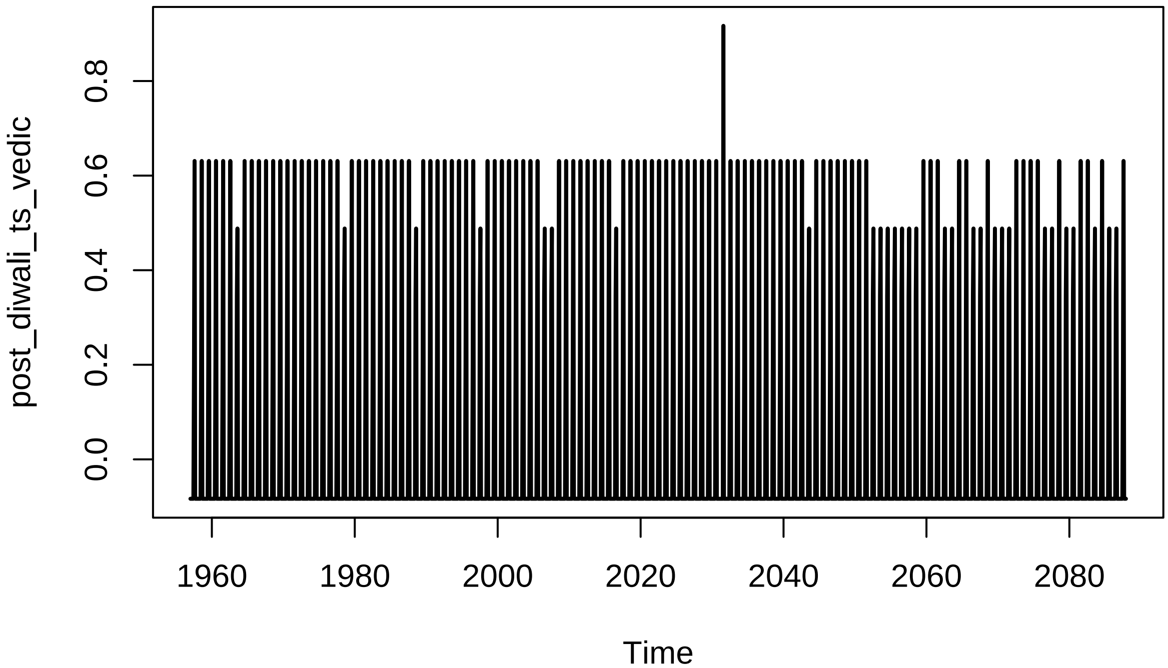
```
pre_diwali_ts_vedic <- genhol(as.Date(diwali_vedic_calendar), start = -6, end = -1, center = "mean")
post_diwali_ts_vedic <- genhol(as.Date(diwali_vedic_calendar), start = 0, end = 6, center = "mean")
```

`pre_diwali_ts` and `post_diwali_ts` both are of time-series class object and represent 2 time-series to include pre and post festival for better seasonal adjustment.

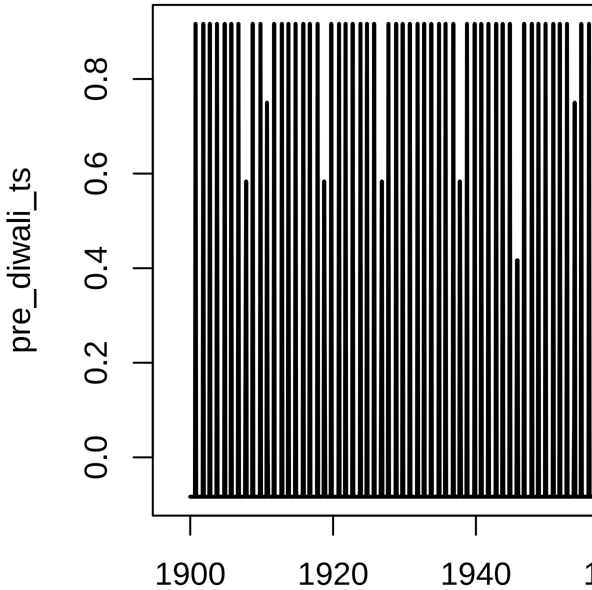
```
ts.plot(pre_diwali_ts_vedic,lwd = c(2, 1))
```



```r
ts.plot(post_diwali_ts_vedic,lwd = c(2, 1))
```

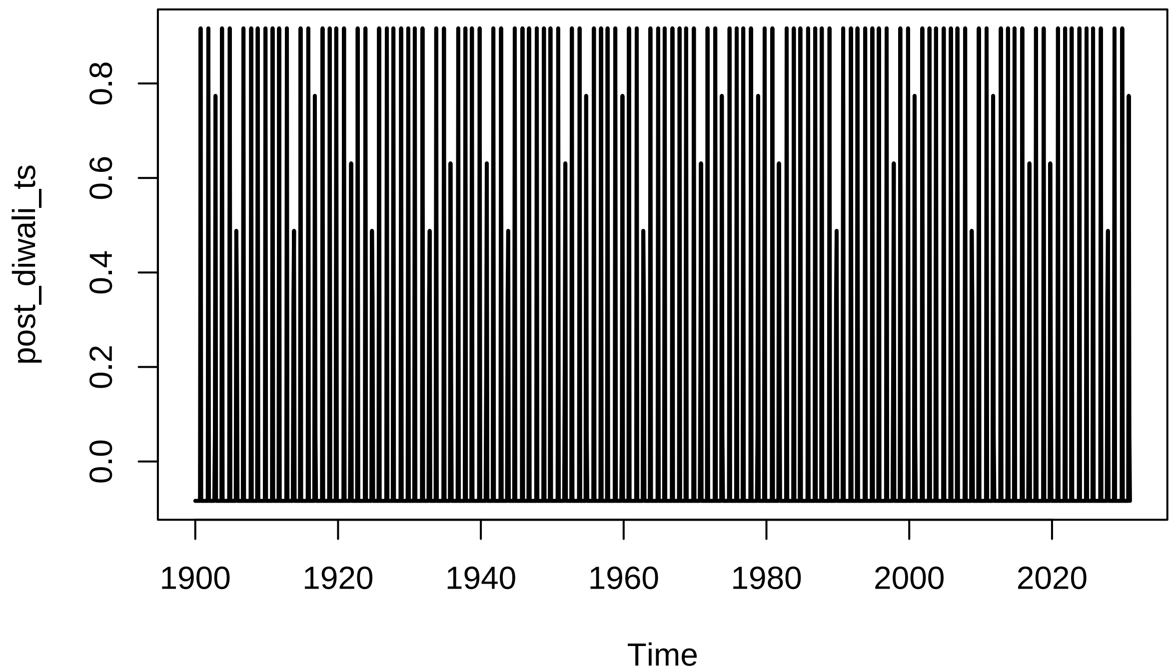



```r
pre_diwali_ts<- genhol(seasonal::diwali, start = -6, end = -1, center = "mean")
post_diwali_ts<- genhol(seasonal::diwali, start = 0, end = 6, center = "mean")
ts.plot(pre_diwali_ts,lwd = c(2, 1))
```



Generate time-series with Gregorian calendar system for Diwali

```
ts.plot(post_diwali_ts,lwd = c(2, 1))
```



Including user defined regressors

Converting India Industrial Output time-series object to Vedic Calendar system

Converting ‘seasonal::iip‘ ts object to Vedic calendar based time-series object to model seasonality

```
iip_data <- data.frame(Y=as.matrix(seasonal::iip), date=as.Date(zoo::as.yearmon(time(seasonal::iip))))
date<- iip_data$date
date <- as.POSIXct.Date(date)
julianday_iip <- insol::JD(date)
place <- c(15.34, 75.13, +5.5) #Latitude, Longitude and timezone of the location
iip_vedic_calendar = c()
for (i in 1:length(julianday_iip))
{
iip_vedic_calendar = c(iip_vedic_calendar, get_vedic_date(julianday_iip[i], place))
}
iip_data$date <- iip_vedic_calendar
iip_vedic_data_ts <- ts(iip_data$Y,start=c(2061,2))
iip_vedic_data_ts
```

```
## Time Series:
## Start = 2062
## End = 2177
## Frequency = 1
##   [1]  99.0838 103.0900 103.9666 102.4425 104.1004 104.4108 107.3272 104.6366
##   [9] 116.8191 118.4664 112.4255 126.7159 108.8396 114.8274 114.2126 117.6044
```

```
##  [17] 114.2655 118.1773 117.6792 125.5368 132.7696 134.8564 127.8147 144.8739
##  [25] 128.2021 136.8587 136.7408 136.6459 134.5998 133.9824 140.7213 137.9242
##  [33] 150.7315 152.5200 149.3196 161.8845 142.3296 146.7452 148.3757 144.2976
##  [41] 141.8657 148.5866 146.1718 139.6522 148.2840 144.3709 138.5057 153.5340
##  [49] 139.5905 144.2723 145.7417 146.7192 149.4225 151.0090 149.6481 148.4985
##  [57] 162.3779 163.6191 157.5185 176.4743 157.8465 156.5436 156.5543 161.3000
##  [65] 156.1000 160.3000 166.6000 158.0000 175.6000 175.9000 168.0000 193.1000
##  [73] 166.2000 166.2000 171.4000 167.2000 161.4000 164.3000 158.3000 167.5000
##  [81] 180.3000 177.6000 175.2000 187.6000 164.1000 170.3000 168.0000 167.1000
##  [89] 164.7000 163.1000 171.6000 165.8000 179.3000 182.0000 176.2000 194.2000
##  [97] 166.5000 166.0000 164.9000 171.4000 165.4000 167.5000 169.6000 163.6000
## [105] 179.5000 184.0000 172.7000 193.3000 172.7000 175.3000 172.0000 173.0000
## [113] 166.2000 172.2000 162.5000 169.8000
```

The `seasonal` package allows to add user-defined regressors to remove seasonality from a time-series. Here `pre_diwali_ts` and `post_diwali_ts` are added in the main seasonal adjustment. `X-13ARIMA-SEATS` is used to adjust for the festival seasonal component.

```r
m1 <- seas(industrial_prod, xreg = cbind(pre_diwali_ts, post_diwali_ts), regression.usertype = "holiday
```

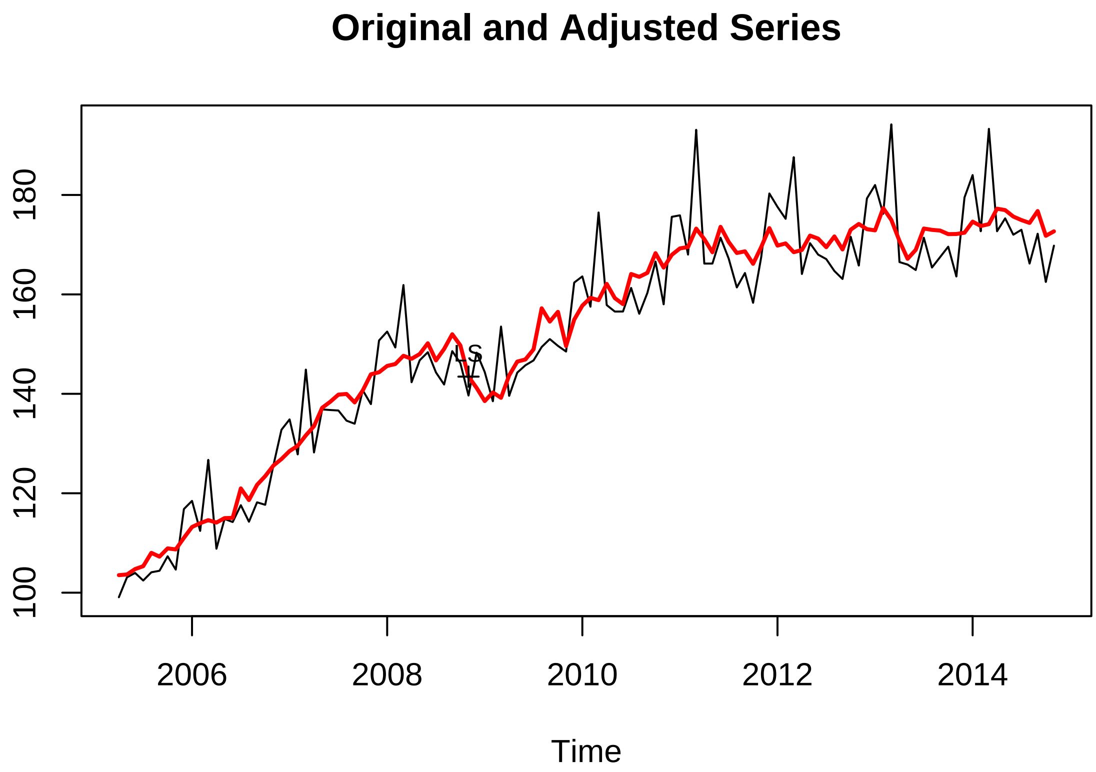
`xreg` adds the user-defined regressors and `x11` is chosen as the decomposition effect.

```r
summary(m1)
```

```
##
## Call:
## seas(x = industrial_prod, xreg = cbind(pre_diwali_ts, post_diwali_ts),
##     regression.usertype = "holiday", x11 = list())
##
## Coefficients:
##                      Estimate Std. Error z value Pr(>|z|)
## xreg1               -0.0034498  0.0091813  -0.376  0.70711
## xreg2               -0.0383834  0.0081358  -4.718 2.38e-06 ***
## Weekday              0.0012087  0.0004275   2.827  0.00469 **
## Constant            -0.0014297  0.0003227  -4.431 9.39e-06 ***
## LS2008.Nov          -0.0738919  0.0160213  -4.612 3.99e-06 ***
## MA-Nonseasonal-01    0.4328295  0.0785625   5.509 3.60e-08 ***
## MA-Seasonal-12       0.9995753  0.0785461  12.726  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## X11 adj.  ARIMA: (0 1 1)(0 1 1)  Obs.: 116  Transform: log
## AICc: 543.2, BIC: 562.7  QS (no seasonality in final):    0
## Box-Ljung (no autocorr.): 31.53   Shapiro (normality): 0.9894
```

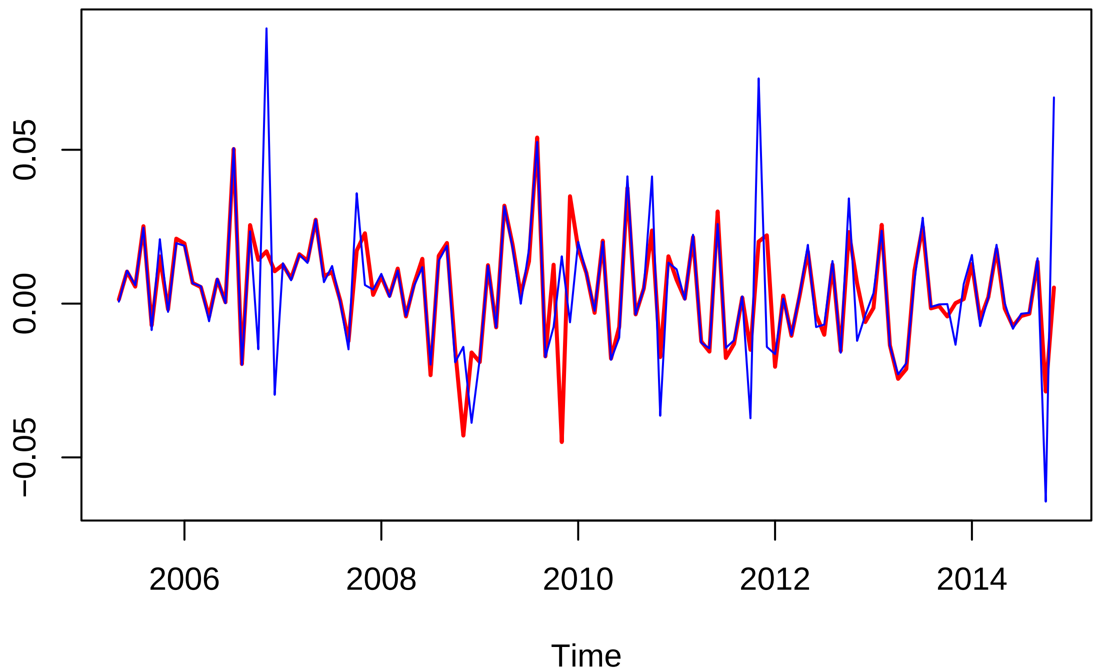
The seasonal co-efficient shows minor decline during pre and post Diwali season across the time-series. In the below unadjusted v adjusted seasonal plot it can be observed that seasonal adjustment based on Diwali season removes distortion from the time-series.

```r
plot(m1)
lines(iip_vedic_data_ts, col="red")
```



Comparing the series

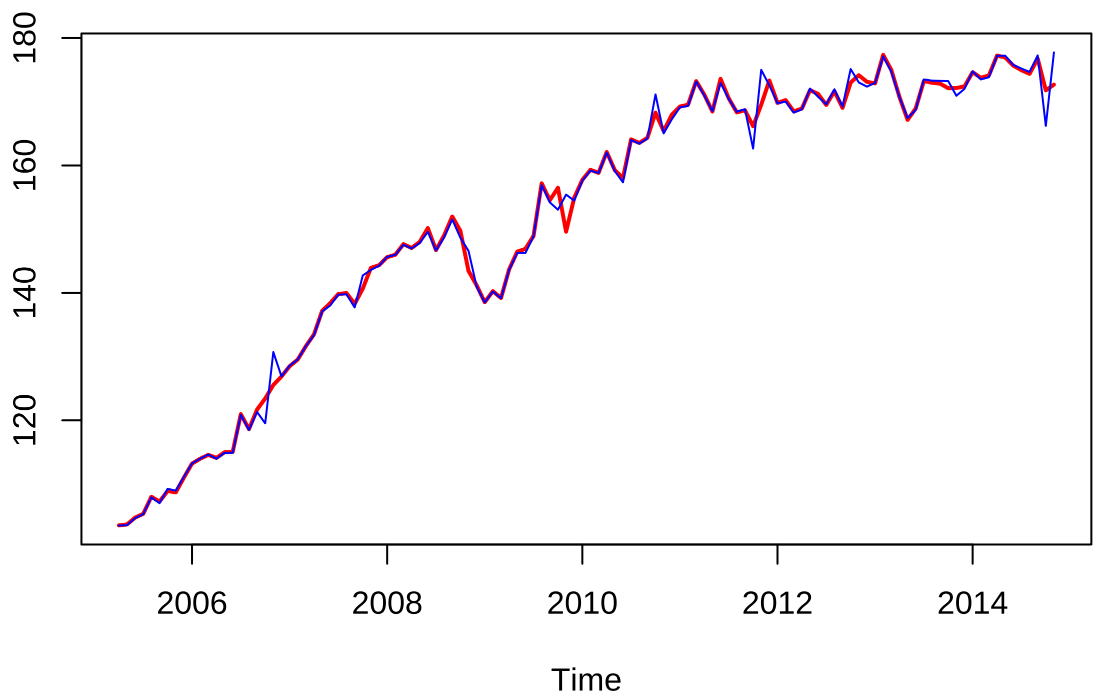
```
m2 <- seas(x = iip, x11 = list(), regression.variables = c("td1coef", "ls2008.Nov"),
           arima.model = "(0 1 1)(0 1 1)", regression.aictest = NULL, outlier = NULL,
           transform.function = "log")
ts.plot(diff(log(cbind(final(m1), final(m2)))), col = c("red", "blue"),
        lwd = c(2, 1))
```



In the above chart, non-adjusted(blue) vs adjusted(red) seasonal plot clearly shows the amount of distortion present in the series.

The below chart also indicated a level of distortion present for industrial output due to Diwali festival.

```
ts.plot(final(m1), final(m2), col = c("red", "blue"), lwd = c(2, 1))
```



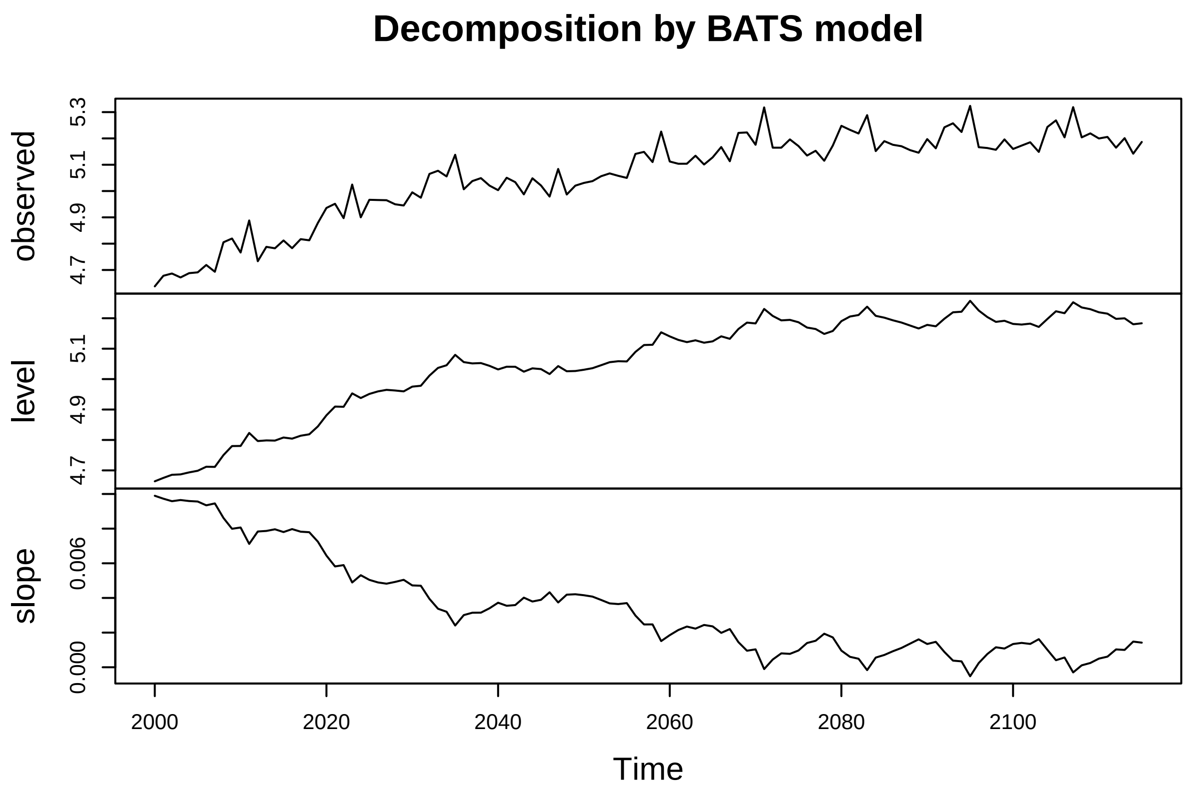
Multi-seasonality decomposition for India industrial output

(**forecast?**) library was used to identify multiple seasonality present in the data-set and understand the difference between them.

Seasonality decomposition for Industrial Output for Gregorian Calendar

The start of the series is from year 2000 according to Gregoriann calendar system

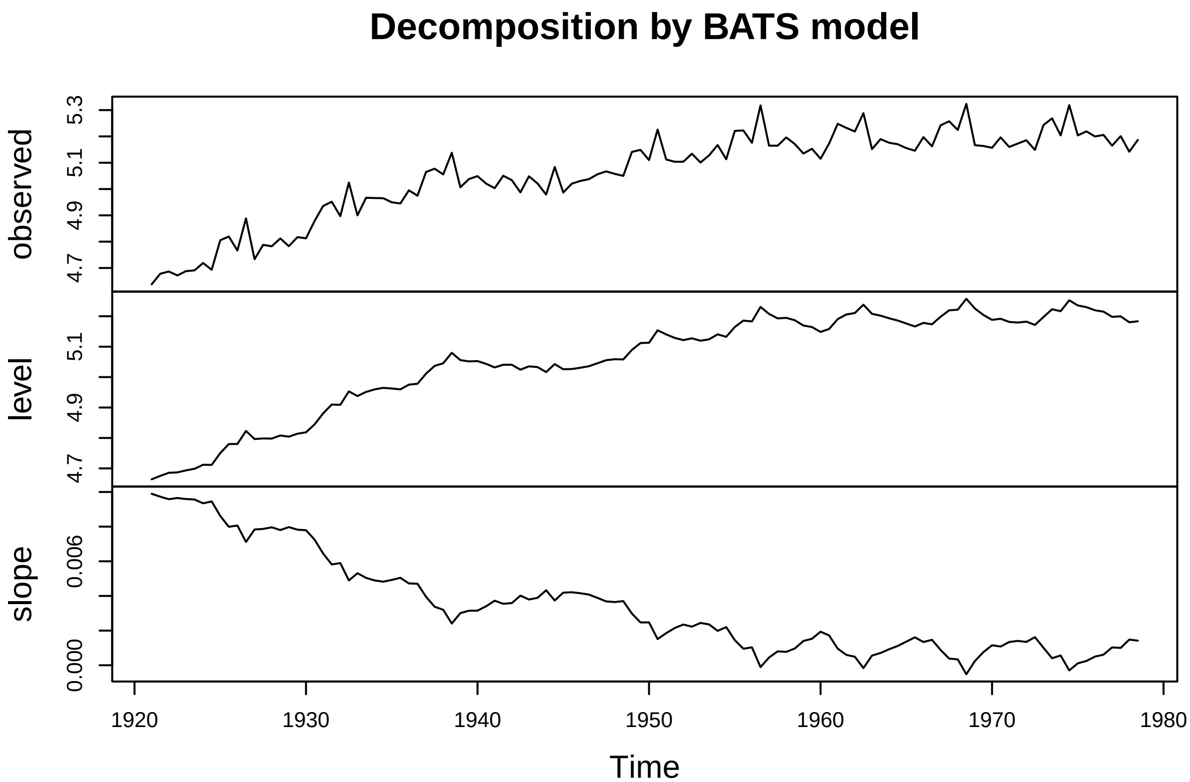
```
iip_tbats <- forecast::msts(industrial_prod, start=2000, seasonal.periods = c(1))
fit <- forecast::tbats(iip_tbats)
plot(fit)
```



Seasonality decomposition for Industrial Output for Vedic Calendar

The start of the series is 1921, according to Vedic calendar system which is equivalent to 2000

```
iip_tbats_vedic <- forecast::msts(iip_vedic_data_ts, start=1921, seasonal.periods = c(2))
fit_vedic <- forecast::tbats(iip_tbats_vedic)
plot(fit_vedic)
```



Sax, Christoph, and Dirk Eddelbuettel. 2018. "Seasonal Adjustment by {x-13ARIMA-SEATS} in {r}" 87. https://doi.org/10.18637/jss.v087.i11.