Seasonal adjustment of daily time series in R: An overview

Automated data processing and the Internet have brought an enormous increase in data that is processed on a high frequency, e.g., at a daily, hourly or even higher frequency. While some higher frequency series have been used in the past (e.g., Fisher 1923, cited by Ladiray 2018) these series are much more abundant now. X-13ARIMA-SEATS offers a well tested and time proven way of adjusting monthly, quarterly (or bi-annual) series, but it cannot deal with data at a higher frequency.

This [article/chaper/post] discusses how to perform seasonal adjustment on a higher frequency. We focus on daily data, as this is the most common use case, but will briefly discuss some challenges involving weekly or intra-day adjustments.

Despite the large interest, there is not much consensus on the appropriate adjustment method for daily series. Adjusting daily series often involves a substantial amount of trial and error, subjective judgment and exploration. This [article/chapter/post] gives an overview of the tools that are currently (2021) available in R.

Contribution of this [article/chaper/post]

- Discuss literature on daily seasonal adjustment
- Overview of available methods in R (including examples)
- OOS forecast evaluation of available methods in R
- Discuss specific problems

Problems

Daily seasonal adjustment comes with a few challenges that are not present in lower frequency data. Let's focus on daily traffic casualties.

First, daily data comes at multiple periodicities: There is an annual periodicity, such as the effect of weather condiditions or holiday patterns. Then there is a weekly periodicity. Casualities may be higher during the weekdays, due to increased work traffic. For some series, there may be also a monthly periodicity. If people are get their salaries by the end of the month, they may be more likely to perform certain investments.

Second, many daily data series are available for a few years only. Whily, e.g., the SEATS adjustment requires a minimal series length of XX years, many daily series are shorter.

Third, higher frequency series are generally more volatile and prone to outliers.

Fourth, the effect of individual holiday is challenging to estimate. Often, economic effects of holidays may occur before or after a holiday, thus lagging or leading them is crucial.

Parametric versus Non-parametric Models

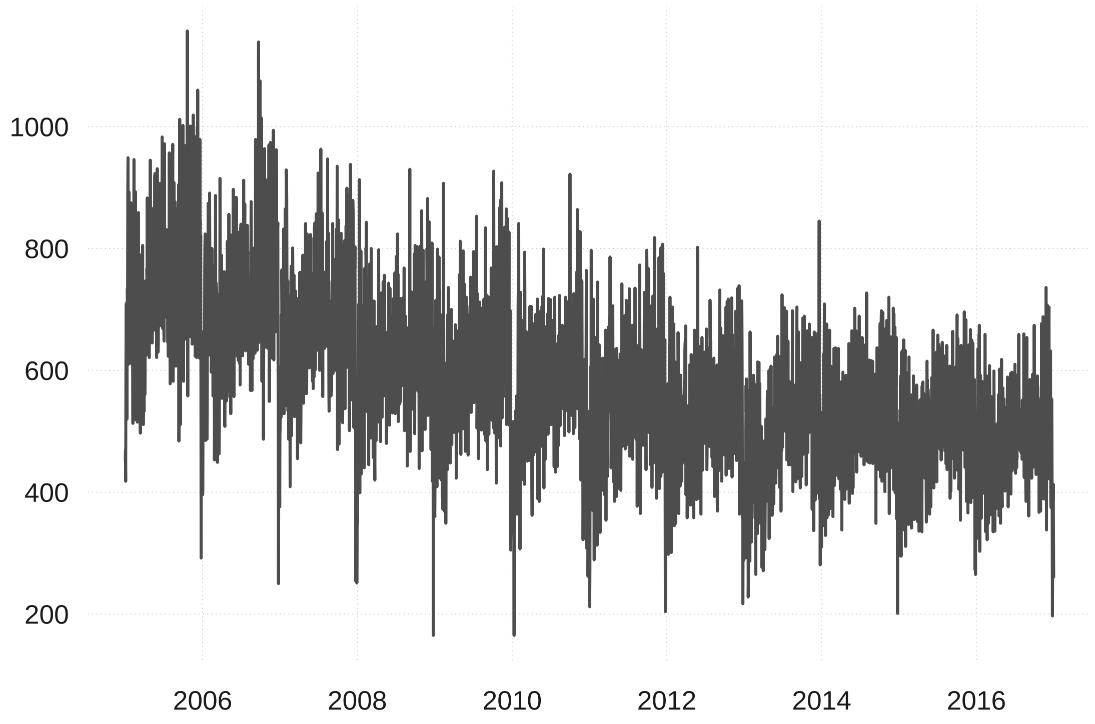
Various attempts to seasonly adjust data can be broadly distinguished into parametric and non-parametric approaches. Non-parametric approaches seem to be the more obvious candidates to use with the irregular structure of daily data. Parametric models require the time units to be regularly spaced. Non-parametric estimation is also what is used in the X-11 method of X-13.

R Packages

As mentioned before, there is no accepted consensus on how to perform daily seasonal adjustment. In the following, we discuss various possibilities to adjust series in R. We focus on a single time series, and describe the concrete steps that are required to perform an adjustment.

```r
library(dailyadj)
library(tsbox)
```

```
x <- casualties

ts_plot(x)
```



ARIMA + Month, Weekday / Holiday Dummies

Let's start with a simple model.

Perhaps because of their simplicity, these kind of adjustments are frequently found in the literature. E.g., timmermans 18, lengwiler, (forthcoming, ask Ronald)

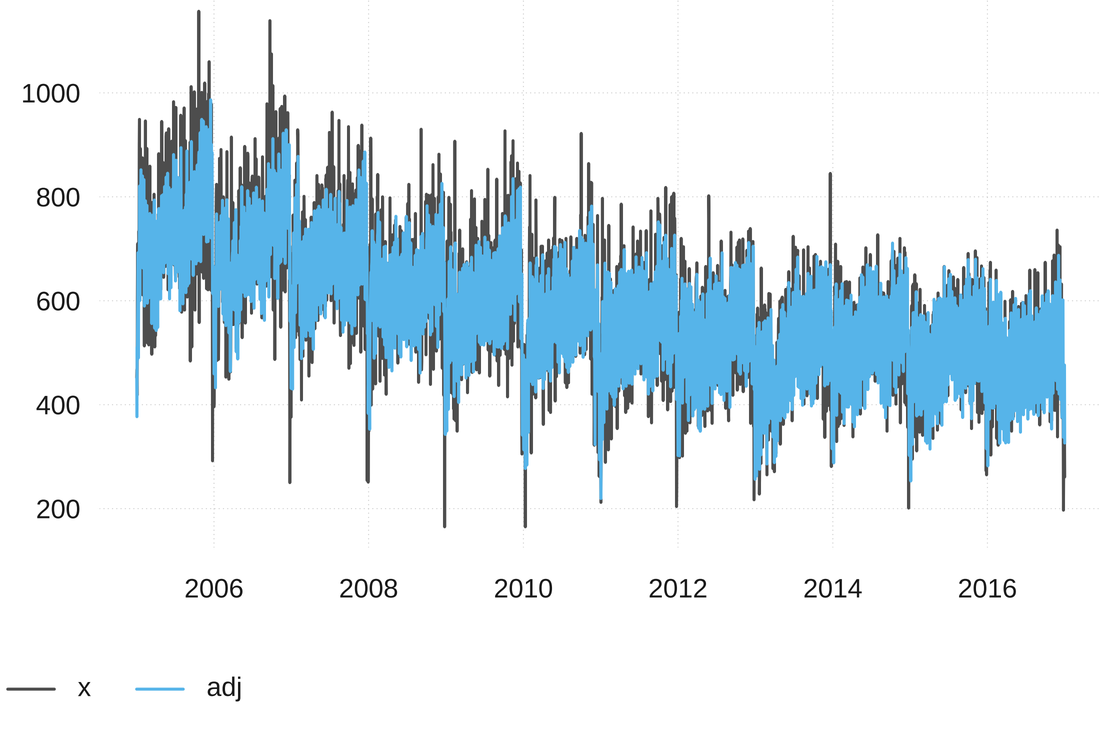
We start by constructing a dummy variables with weekday and monthly effects:

```
dums <-
  x %>%
  mutate(wday = lubridate::wday(time, label = TRUE)) %>%
  mutate(month = lubridate::month(time, label = TRUE)) %>%
  select(time, wday, month) %>%
  fastDummies::dummy_cols("wday", remove_selected_columns = TRUE) %>%
  fastDummies::dummy_cols("month", remove_selected_columns = TRUE) %>%
  select(-wday_Mon, -month_Jan)

dums
```

```
## # A tibble: 4,383 x 18
##    time       wday_Sun wday_Tue wday_Wed wday_Thu wday_Fri wday_Sat month_Feb
##    <date>        <int>    <int>    <int>    <int>    <int>    <int>     <int>
##  1 2005-01-01        0        0        0        0        0        1         0
##   2 2005-01-02         1         0         0         0         0         0          0
##   3 2005-01-03         0         0         0         0         0         0          0
##   4 2005-01-04         0         1         0         0         0         0          0
##   5 2005-01-05         0         0         1         0         0         0          0
##   6 2005-01-06         0         0         0         1         0         0          0
##   7 2005-01-07         0         0         0         0         1         0          0
##   8 2005-01-08         0         0         0         0         0         1          0
##   9 2005-01-09         1         0         0         0         0         0          0
##  10 2005-01-10         0         0         0         0         0         0          0
## # ... with 4,373 more rows, and 10 more variables: month_Mar <int>,
## #   month_Apr <int>, month_May <int>, month_Jun <int>, month_Jul <int>,
## #   month_Aug <int>, month_Sep <int>, month_Oct <int>, month_Nov <int>,
## #   month_Dec <int>
```

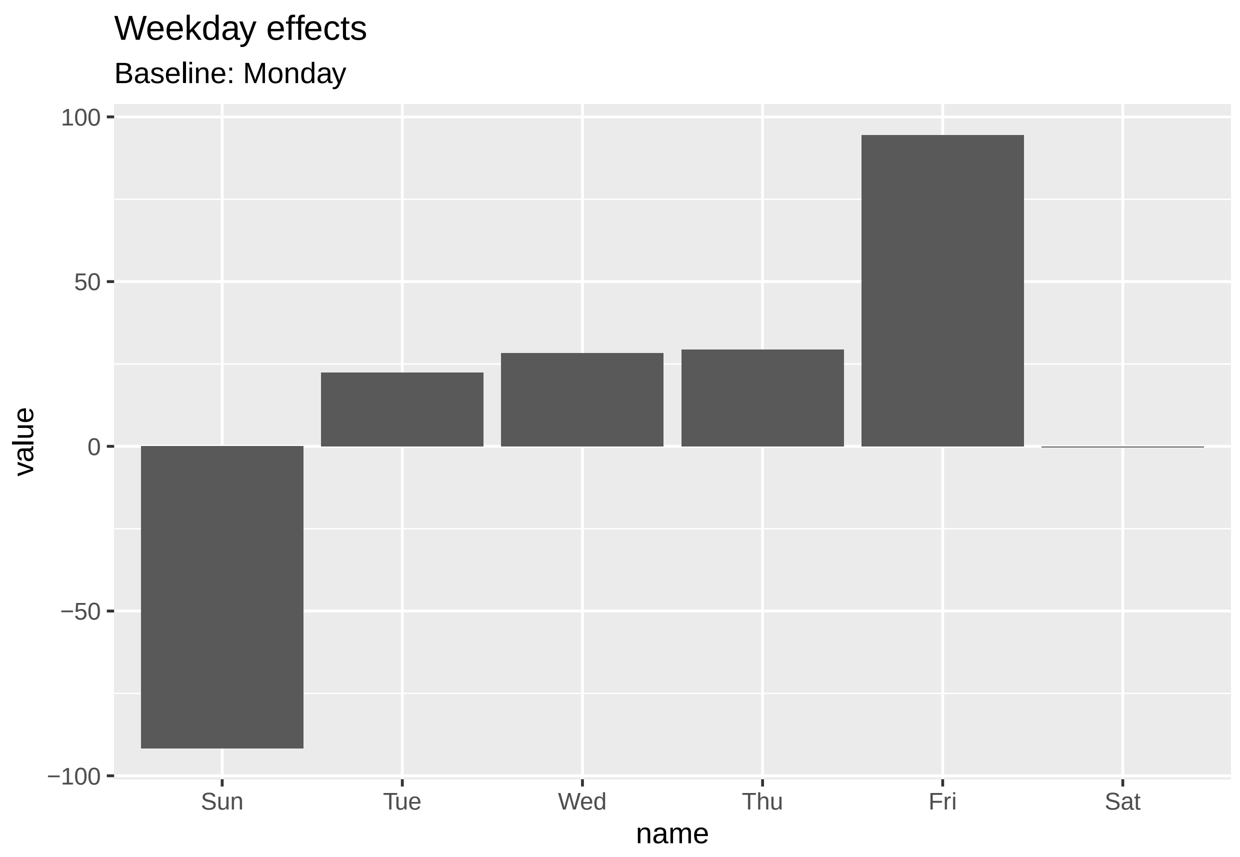
These variables can be used as exogenous variables in a ARIMA model. We use `forecat::auto.arima()` to determine the ARMA order. Note that we do not want to use the seasonal part of the model, since we use dummies for this purpose.

```r
fit <- auto.arima(x$value, seasonal = FALSE, xreg = as.matrix(dums[, -1]))

adj <- x
adj$value <- as.numeric(fit$fitted)

ts_plot(x, adj)
```

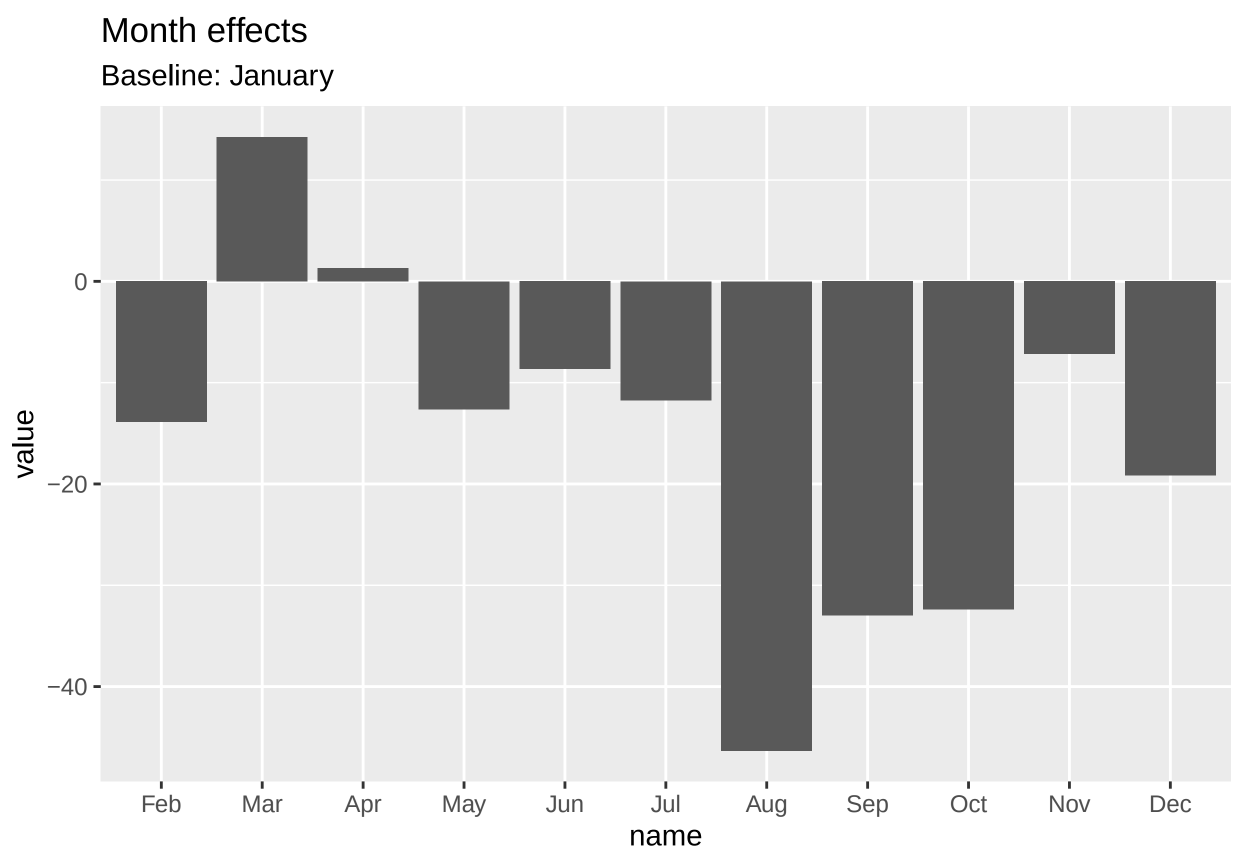


The nice think about the dummy model is that its seasonal effects are very easy to interprete. By construction, they are constant over time, and can be visualized as follows:

```r
enframe(coef(fit)) %>%
  filter(grepl("wday", name)) %>%
  mutate(name = gsub("wday_", "", name)) %>%
  mutate(name = factor(name, levels = unique(name))) %>%
  ggplot(aes(x = name, y = value)) +
    geom_col() +
    ggtitle("Weekday effects", subtitle = "Baseline: Monday")
```



```r
enframe(coef(fit)) %>%
  filter(grepl("month", name)) %>%
  mutate(name = gsub("month_", "", name)) %>%
  mutate(name = factor(name, levels = unique(name))) %>%
  ggplot(aes(x = name, y = value)) +
    geom_col() +
    ggtitle("Month effects", subtitle = "Baseline: January")
```

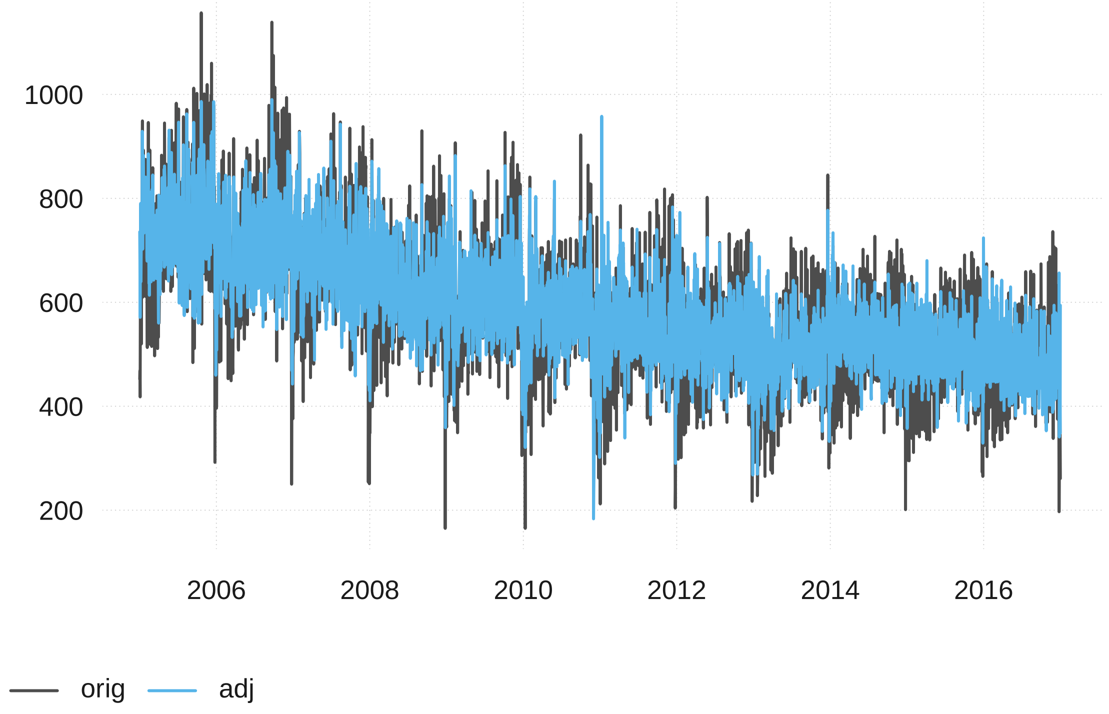


We see that, on average, transcations are lower on Sunday and peak on Friday. We also see that, on average, transactions are sligthly lower in early autumn.

STL

This package contains a very simple implementation of STL that works in many circumstances.

```
seas_daily(x) %>%
  ts_pick("orig", "adj") %>%
  ts_plot()
```



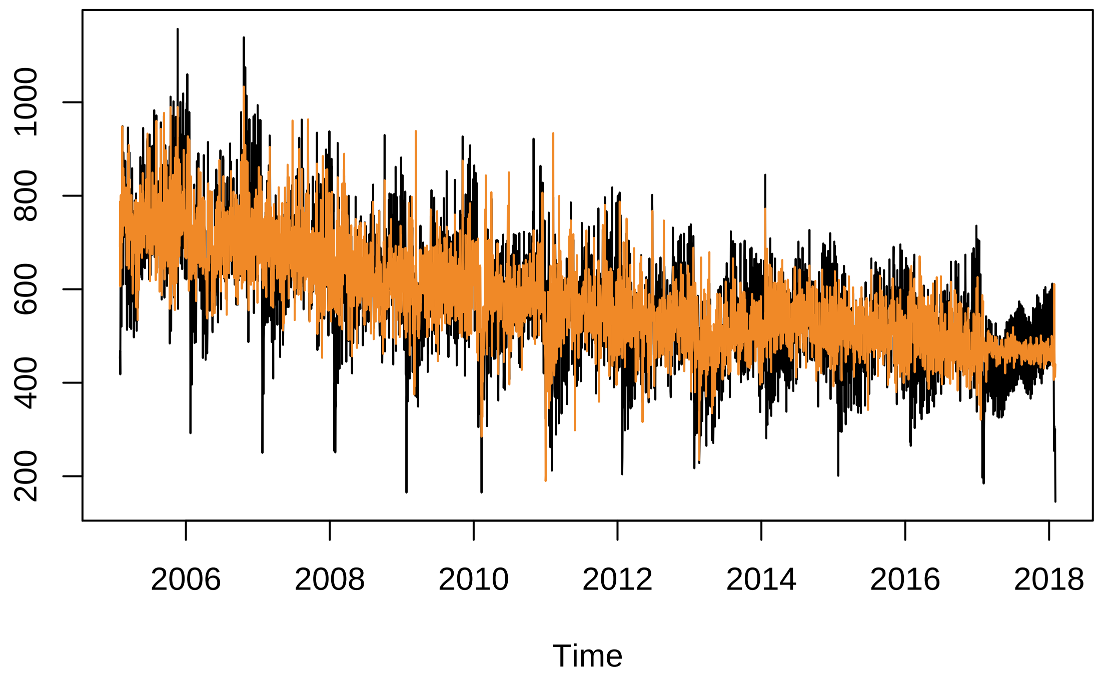
dsa

Similarly in spirit, the dsa packages implements a version of STL that works well in many circumstances, but is computationally slow. The following code automatlically decomposes `casulties`.

```
library(dsa)
z <- dsa::dsa(ts_xts(x))
```

```
##   |                                                                              |
```

```
plot(z, dy = FALSE)
```

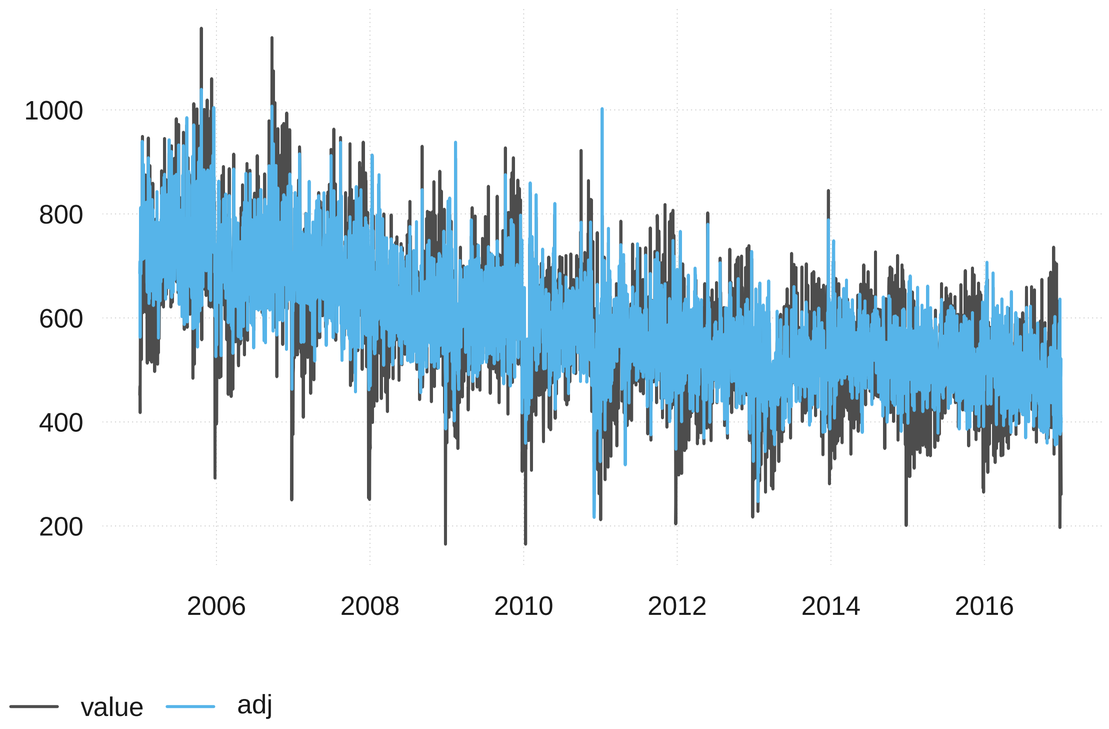


prophet

```r
library(prophet)
```

```
## Loading required package: Rcpp

## Loading required package: rlang

##
## Attaching package: 'rlang'

## The following objects are masked from 'package:purrr':
##
##     %@%, as_function, flatten, flatten_chr, flatten_dbl, flatten_int,
##     flatten_lgl, flatten_raw, invoke, list_along, modify, prepend,
##     splice
```

```r
df <- rename(x, ds = time, y = value)
m <-
  prophet(daily.seasonality = FALSE) %>%
  # including swiss holidays, which seems to have no effect
  add_country_holidays(country_name = 'CH') %>%
  fit.prophet(df)

# not strictly needed, but will include forecast too
future <- make_future_dataframe(m, periods = 31)
forecast <- as_tibble(predict(m, future))
forecast %>%
  transmute(
    time = as.Date(ds),
    additive_terms,
    yhat
  ) %>%
  left_join(x, by = "time") %>%
  mutate(adj = value - additive_terms) %>%
  select(time, value, adj) %>%
  ts_long() %>%
  ts_plot()
```



stlf

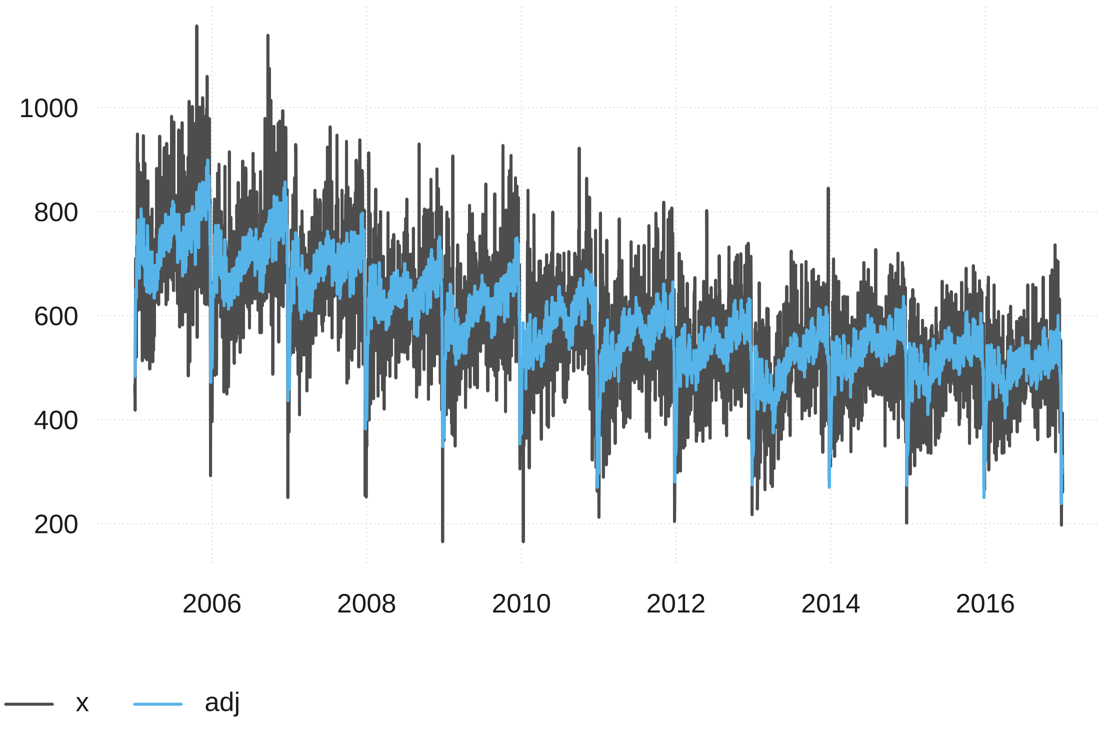
Some models require the data to be equispaced. I.e., each low frequency period must include the same number of high frequency periods. `ts_ts` from the tsbox package offers an easy way to convert daily data into regular `"ts"` objects with a frequency of 365.2425. Thus, days are slightly offset in each year.

```
x_ts <- ts_ts(x)
head(x_ts)
```

```
## Time Series:
## Start = 2005
## End = 2005.01368953503
## Frequency = 365.2425
## [1] 452 468 418 599 686 710
```

[Probably don't show everything. This one is useless]

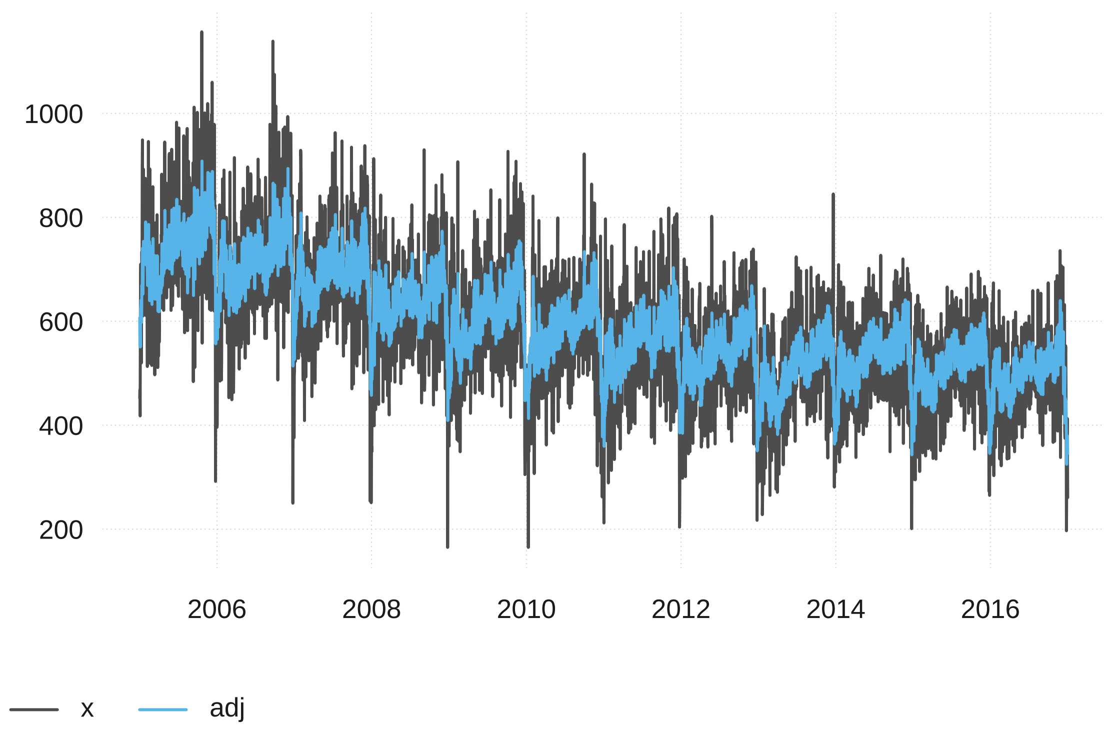
```r
m <- stlf(x_ts)
fct <- ts_tbl(m$mean)
adj <- m$fitted
ts_plot(x, adj)
```



TBATS

Other models require the data to be equispaced and without missing values. Imputation offers an easy way out.

```r
fit <- tbats(imputeTS::na_interpolation(x_ts))
adj <- fit$fitted
ts_plot(x, adj)
```



Evaluation

Which method should you choose? [discuss criterions]

Out-of-sample forecast

As in (Timmermans 18), the following produces OOS forecasts for the twelve last months for UK traffic deaths. We apply `dailyseas::eval_oos()` to perform an OOS forecast evaluation for all models.

The full OOS results can be found here

[I think the detailed OOS results are quite interesting, as they show you what a method 'gets' and what it does not.]

Here is the table for two series, showing the mean percentage deviation:

model	casualties	transact
dummy	0.1564534	0.1554219
daily	0.1101139	0.1105550
dsa	0.0976475	0.1016794
prophet	0.1049195	0.1165156
harmon	0.1234707	0.1542884
tbats	0.1314058	0.1306881
naive	0.1402331	0.1710234

[The framework is now quite clean, so it is easy to include new methods and series, or show other statistics] prophet, dsa, and my stl work realatively good, the other methods are not working very well.

Other Criteria

What other criteria could we look at?

- Variance?
- Tests for remaining seasonality?
- ???

Computation time

```
stl (my one): 0.639
seas_dummy:   2.319
prophet:     10.181
dsa:        145.553
```

Other challenges

[Discuss some other challenges of daily seas adj]

Short series

Many daily time series are short. What does it mean with respect to the discussion above. If I have only 3 years of data, which methods still work?

Calendar effects

[Will try to improve my method on that, prophet and dsa do somehting about it. Could provide an example, OOS comparison]

Cross Seasonal Effects

If month effects (e.g., salary payment at the end of a month) collide with week effects (e.g., weekend), we can get some patterns that are very hard to model. How relevant is the problem? What should you be done about it?

Series specific effects

Transaction series have an end of months effect (Timmermans 18), but we don't find it in other data. How to deal with such things?

Weekly seasonality

One solution would be to disaggregate, perform daily seasonal adjustment and aggregate again. Could provide an example.

Compare to weekly methods?

Conclusions