Time Series in R - BernR Meetup

Isabel Martinez

9/9/2019

Overview

Time series as data type in base R

Seasonal adjustment made easy

Visualizing data with ts.plot

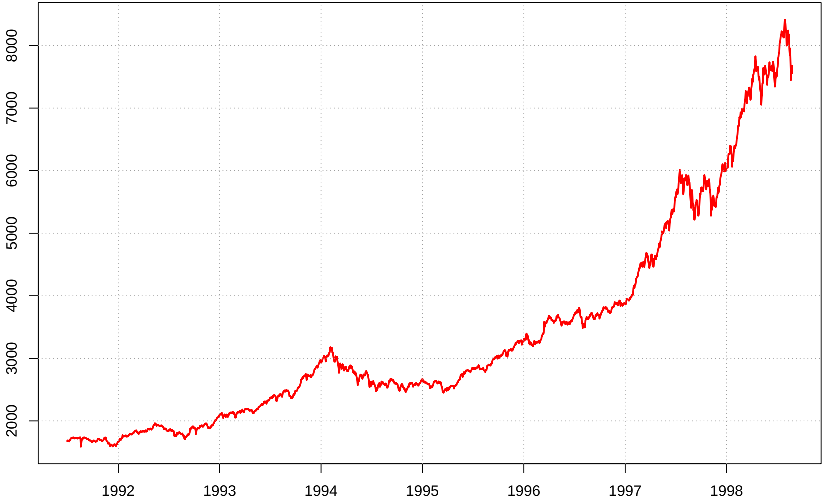
tsbox

Time series

Many data have a time dimension which naturally structures the data. In econ, typical examples include:

- annual profits of a company
- quarterly sales
- monhtly imports and exports
- daily stock prices or exchange rates

Example: Swiss Stock Exchange Index SMI



Time series objects in R

- Can define data as time series; very handy, e.g., to:
  - compute growth rates
  - work with lagged values
  - seasonally adjust the data
- Base R knows the object class ts
- Just have to tell R the start date and the frequency of our data

Time series objects in R

- Can define data as time series; very handy, e.g., to:
  - compute growth rates
  - work with lagged values
  - seasonally adjust the data
- Base R knows the object class ts
- Just have to tell R the start date and the frequency of our data

- Many popular packages for time series exist, e.g., zoo and xts
- They all come with their own syntayx...
- ... and object classes
- Great source for confusion if you're just starting out!

Defining a ts-object: annual series

```r
values <- round(runif(24, min = 1, max = 50))

# annual series
ts(values, frequency = 1, start = 1959)
```

```
## Time Series:
## Start = 1959
## End = 1982
## Frequency = 1
##  [1] 10 29 24 29 28 27 24 42 32 22 34 30  3 42 37 24 28
## [24] 25
```

Defining a ts-object: quarterly series

- Definition of frequency: observations per year
- Quarterly data: 4 observations per year → frequency = 4

```
# quarterly series
ts(values, frequency = 4, start = c(1959, 2))
```

```
##      Qtr1 Qtr2 Qtr3 Qtr4
## 1959        10   29   24
## 1960   29   28   27   24
## 1961   42   32   22   34
## 1962   30    3   42   37
## 1963   24   28   43    8
## 1964   21   24   41   17
## 1965   25
```

Defining a ts-object: monthly series

```r
# monthly series
ts(values, frequency = 12, start = c(1959, 1))
```

```
##      Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
## 1959  10  29  24  29  28  27  24  42  32  22  34  30
## 1960   3  42  37  24  28  43   8  21  24  41  17  25
```

Defining a ts-object: decennial series

```r
values <- round(runif(12, min = 1, max = 50))

# decennial series
ts(values, frequency = 0.1, start = 1860)
```

```
## Time Series:
## Start = 1860
## End = 1970
## Frequency = 0.1
##  [1]  2 36 27 46  2 26 13 27 36 17 43 29
```

Useful functions with ts objects

```r
# what is the time frame of a given ts?
time(AirPassengers)

# what is the frequency of a given ts?
frequency(AirPassengers)
```

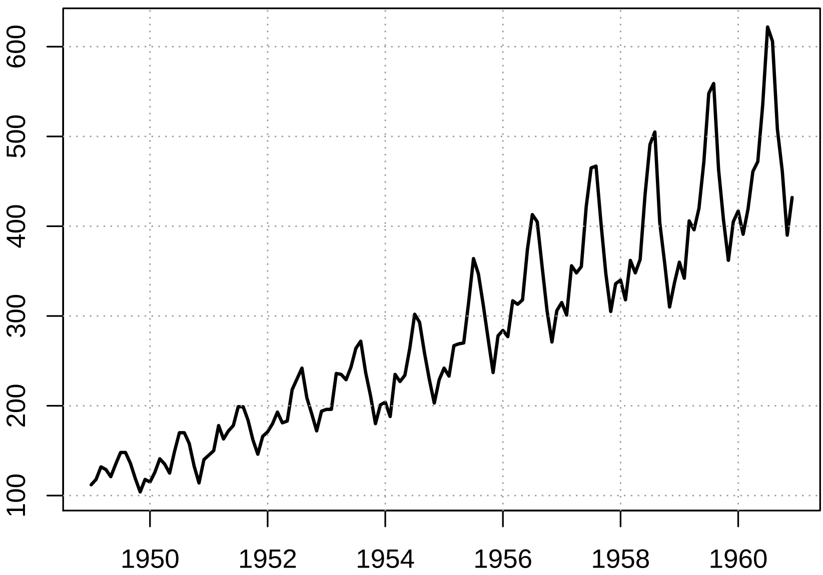
```r
# select a certain time window
window(AirPassengers,
       start = c(1949, 10),
       end = c(1951, 7))
```

```
##      Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
## 1949                                     119 104 118
## 1950 115 126 141 135 125 149 170 170 158 133 114 140
## 1951 145 150 178 163 172 178 199
```

Dealing with seasonality

- TS often have systematic seasonal patterns
- These patterns add noise to our measurement
- We are often more interested in what the trend value of a variable is, rather than its actual value in a given month
- Seasonal patterns make it hard to compare values of a variable at two different points in time

Example: Monthly totals of international airline passengers

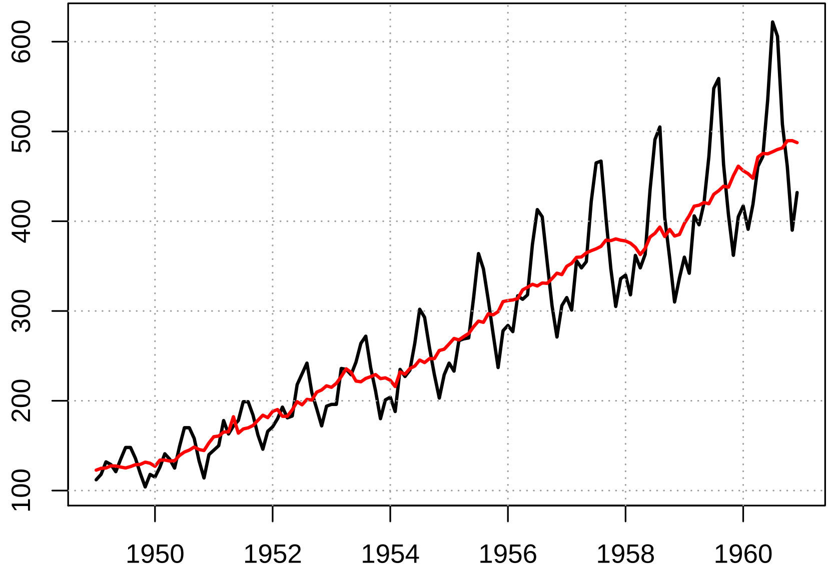


Seasonal adjutment made easy with “seasonal”

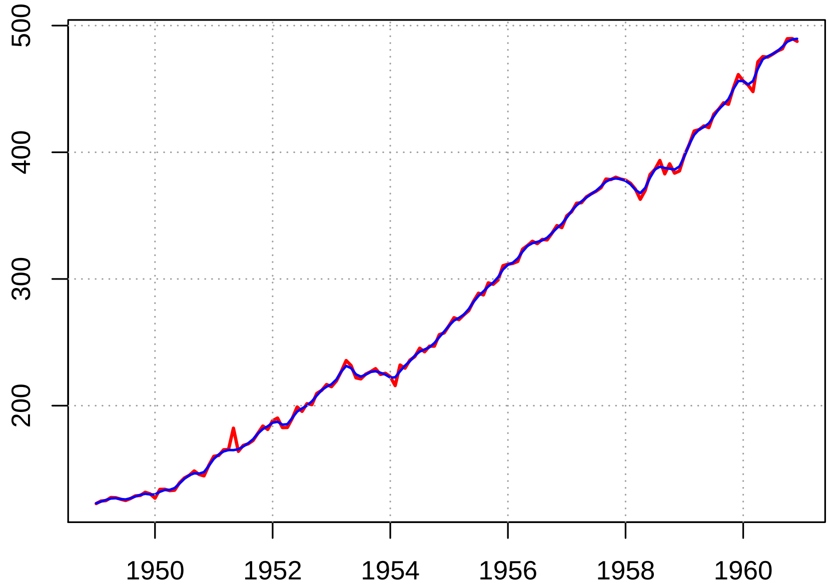
- Great package by Christoph Sax for seasonal adjustment
- Method: X-13ARIMA-SEATS (US-Census Bureau)
- Objects must be of ts class
- Quick and easy, no need to set model parameters (but you can!)

```r
library(seasonal)
??seas

# obtain seasonally adjusted series
AirPassengers_sa <- final(seas(AirPassengers))

# obtain trend series
AirPassengers_t <- trend(seas(AirPassengers))
```

Example: Monthly totals of international airline passengers



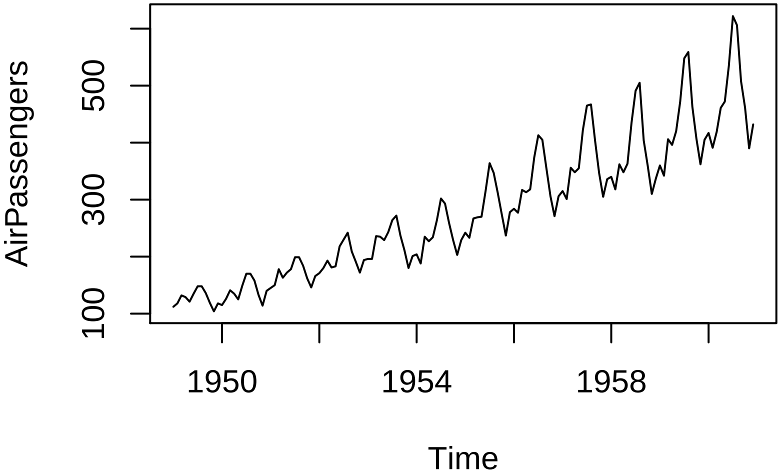
Example: Monthly totals of international airline passengers



Plotting ts objects: ts.plot (base R)

- ts.plot (base R) quickly plots ts objects

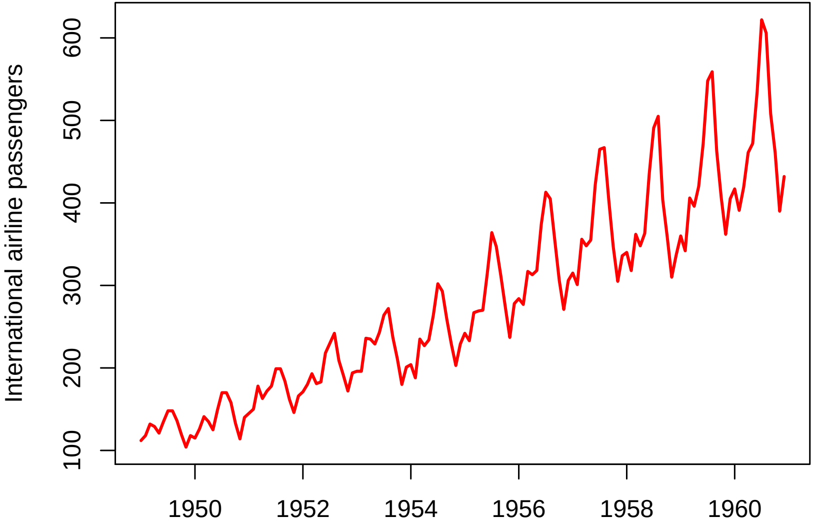
```
ts.plot(AirPassengers)
```



Modifying basic aspects of the graph

```r
ts.plot(AirPassengers,
        ylab = "International airline passengers",
        xlab = "",
        lwd = 2,
        col = "red")
```

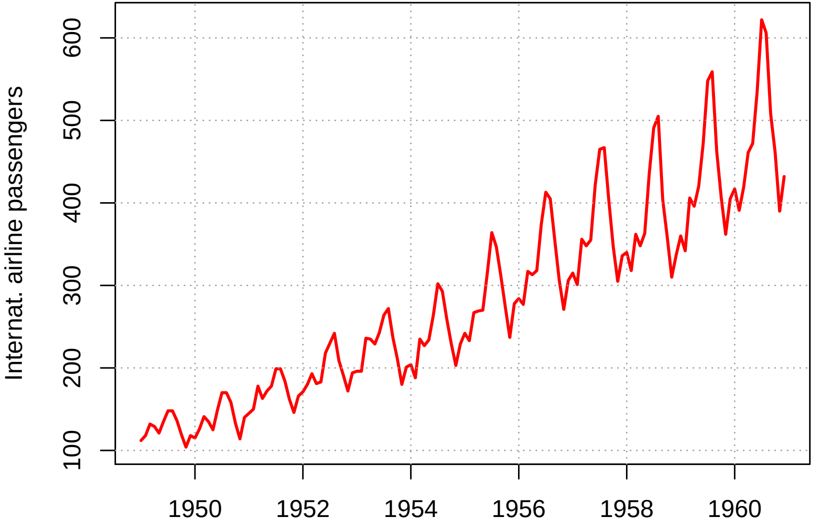
Result: Modifying basic aspects of the graph



Adding a grid

```r
ts.plot(AirPassengers,
        ylab = "Internat. airline passengers",
        xlab = "",
        lwd = 2,
        col = "red")

grid(NULL, NULL, lwd = 1, col = "gray61")
```

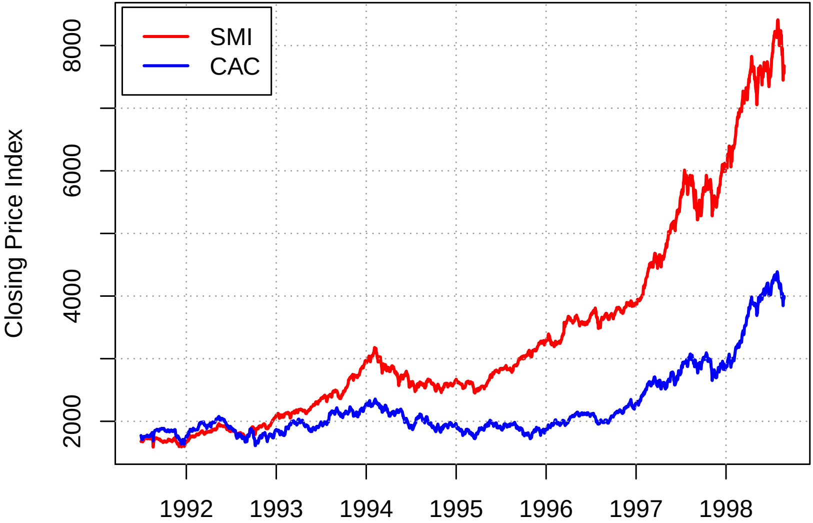
Result: Adding a grid



Multiple time series and legends

```
ts.plot(EuStockMarkets[, "SMI"],
             EuStockMarkets[, "CAC"],
        ylab = "Closing Price Index",
        xlab = "",
        lwd = 2,
        col = c("red", "blue"))

grid(NULL, NULL, lwd = 1, col = "gray61")

legend("topleft", inset=c(0.01, 0.01), ncol = 1,
       legend = c ("SMI", "CAC"),
       col = c("red", "blue"),
       lwd = 2, bg = "white" )
```

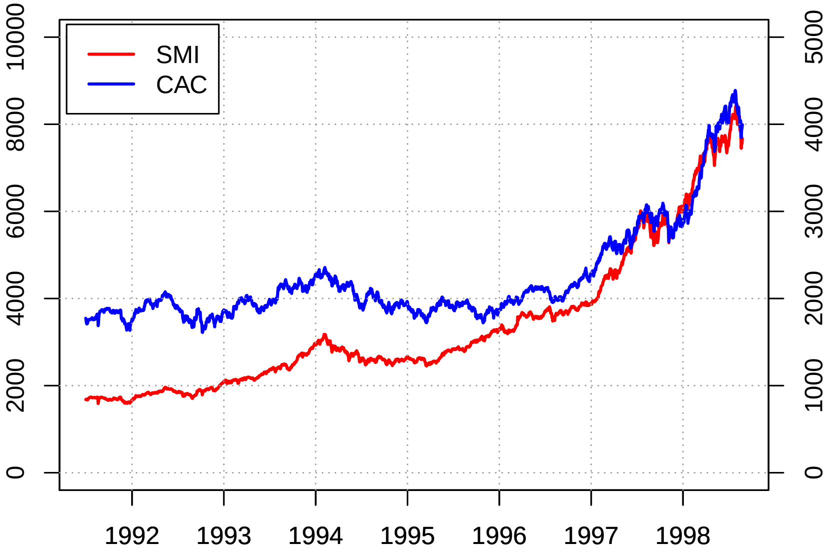
Result: Multiple time series and legends



Second axis

```r
# we overlay 2 plots
# 1st plot
ts.plot(EuStockMarkets[, "SMI"],
        ylab = "SMI",  xlab = "", ylim = c(0, 10000),
        lwd = 2, col = "red")
# 2nd plot
par(new=T)  ##new graphic parameters
ts.plot(EuStockMarkets[, "CAC"],
        ylab = "", xlab = "", ylim = c(0, 5000),
        lwd = 2, col = "blue",
        gpars = list( yaxt="n"))  ##suppress 1st axis
# add 2nd axis
axis(side = 4) ##1st axis has side = 2
grid(NULL, NULL, lwd = 1, col = "gray61")
legend("topleft", inset=c(0.01, 0.01), ncol = 1,
       legend = c ("SMI", "CAC"),
       col = c("red", "blue"), lwd = 2, bg = "white" )
```

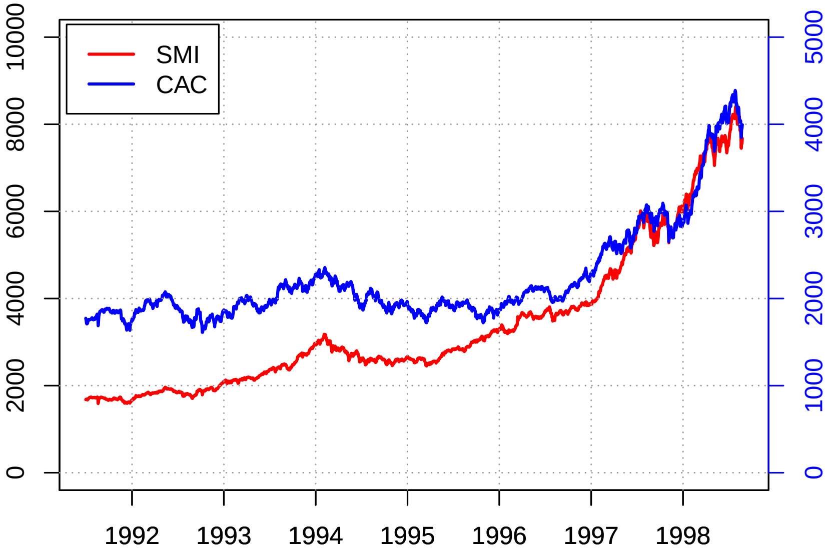
Result: Second axis



Multiple axes with colors

```
ts.plot(EuStockMarkets[, "SMI"],
        ylab = "",  xlab = "", ylim = c(0, 10000),
        lwd = 2, col = "red")
par(new=T)
ts.plot(EuStockMarkets[, "CAC"],
        ylab = "", xlab = "", ylim = c(0, 5000),
        lwd = 2, col = "blue",
        gpars = list( yaxt="n"))
# add 2nd axis and use same color as data it represents
axis(side = 4,
     col="blue", col.ticks="blue", col.axis="blue")
grid(NULL, NULL, lwd = 1, col = "gray61")
legend("topleft", inset=c(0.01, 0.01), ncol = 1,
       legend = c ("SMI", "CAC"),
       col = c("red", "blue"), lwd = 2, bg = "white" )
```

Result: Multiple axes with colors



Time series of the world unite! The tsbox package

- Many time series packages exist, all have own object class, e.g., zoo, xts, tsibble, timeSeries, . . .

```r
library(tsbox) ## (by Christoph Sax)
```

- provides a set of tools that are agnostic towards existing standards
- even handles time series as plain data frames, allowing for dplyr or data.table workflow

- tsbox does many more things, especially:
  - Convert everything into everything
  - Change frequency with *ts_frequency*
  - Create indices with *ts_index*
  - Forecasting with *ts_forecast*
  - Seasonal adjustment with *ts_seas*

Plotting with tsbox: ts_plot and ts_ggplot

- ts.plot can only deal with ts objects
- ts.plot cannot plot different frequencies

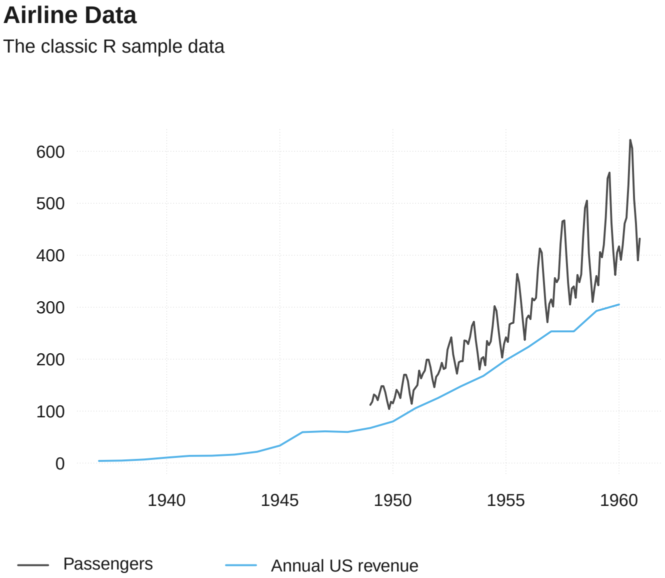
- ts_plot
  - plots different frequencies
  - fast and simple
  - for all time series classes, not only ts
  - limited customizability

- ts_ggplot
  - same syntax and similar plots as ts_plot
  - + ggplot2 graphic system
  - can be highly customized

Example: ts_plot

```r
library(tsbox)
ts_plot( Passengers = AirPassengers,
         "Annual US revenue" = airmiles/100,
  title = "Airline Data",
  subtitle = "The classic R sample data"
)
```

Example: ts_plot


Airline Data
The classic R sample data
Passengers
Annual US revenue