Manipulating Time Series Data in R

Harrison Brown

2022-05-19

Contents

Welcome

Welcome to the course outline for *Time Series Data in R*! This course offers methods and workflows for analyzing and interpreting time series data, an overview of when, why, and how to use time series data, and various utilities and packages in R that are beneficial to analysts.

By the end of this course, students will have the skills to:

- Interpret and understand time series plots
- Import ts data to create and manipulate `ts` objects from the `stats` package
- Understand why time series data is fundamentally different than non-ts data.
- Analyze time series data with plots
- ?Intro to Wavelet analysis?

Chapter 1

Introduction to time series data

1.1 Lesson: What is Time Series Data

- Learning Objective: Learner will be able to understand why and how TS-data differs from non-temporal data
- LO: What kinds of inferences and results can be obtained from TS-data
- LO: Converting to and from time-based data formats, such as `numeric`, `Date`, and `POSIXct` classes
  - Functions: `as.Date()`, `lubridate::`, etc.

1.2 Lesson: How to Interpret Time Series Data

- LO: Learner will understand how to interpret attributes of a basic time series plot
- LO: "Signal and Noise" in the context of TS data
- Introduction to Stationarity: Most real-world data are not stationary and require additional steps to work with

1.3 Lesson: Components of Time Series Data

Chapter 2

Creating and Manipulating Time Series

2.1 `ts` Class

2.2 Missing Values

```
d <- c('2001-01-01', '2001-01-02', '2001-01-04', '2001-01-05')
d <- as.Date(d)
date_range <- seq(min(d), max(d), by = 1)
date_range[!date_range %in% d]
```

```
## [1] "2001-01-03"
```

Chapter 3

Rolling and Expanding Windows

3.1 Rolling Window

- Moving lower and upper bound

3.1.1 Data

3.1.2 Calculating a Rolling Window

Chapter 4

Introduction to Forecasting in R

4.1 Methods for Forecasting

4.1.1 Exponential Smoothing

Chapter 5

Capstone Exercise

The final exercise for this course involves performing a time series analysis on real-world data: Carbon Dioxide concentration at the Mauna Loa Observatory, from early 1959 to Present. You'll go through the process of imputing missing values, testing for stationarity, decomposing the time series, and adjusting for seasonality. The goal for this exercise is a plot showing the seasonal-adjusted time series, which shows the "overall trend" in the data over the last few decades.

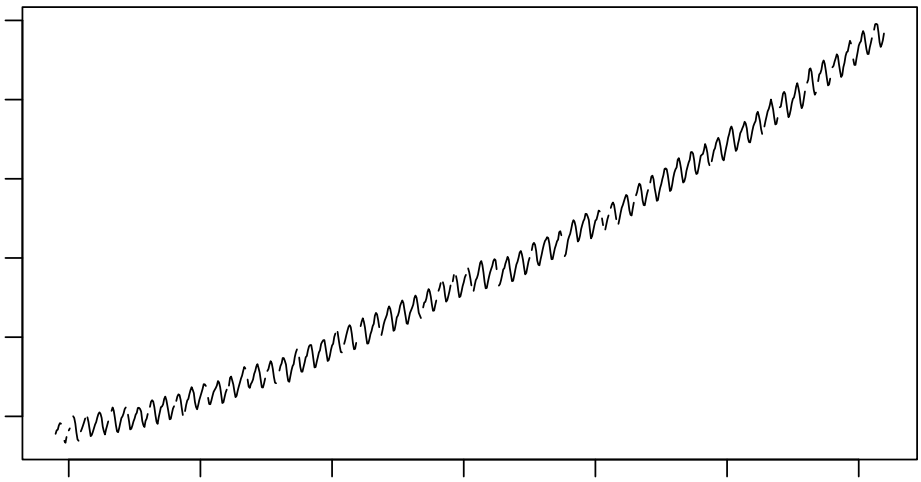
5.1 Importing the Data

```
# The following libraries are included for you

library(tidyverse)
library(zoo)
library(forecast)
library(tseries)
# Sample data from the Mauna Loa Observatory
# https://gml.noaa.gov/webdata/ccgg/trends/co2/co2_mm_mlo.csv

# Data is already pre-processed as a `ts` object. It contains missing values, so
# we'll need to impute those!
co2 <- readRDS("data/missing.Rds")
```

5.2 Visual Checks

1. Plot your co2 data and see if the data is stationary, non-stationary (time-dependent), and seasonal or non-seasonal. Use the Augmented Dickey-Fuller test to determine stationarity.

```
plot.ts(co2)
```



```
adf.test(co2)
```

```
## Error in adf.test(co2): NAs in x
```

Looks like we have some missing values! Let's try and impute those to fill in the gaps:

2. Impute missing values with the `na.approx()` function from `zoo`:

```
co2 <- co2 %>%
  zoo::na.approx()

adf.test(co2)
```

```
## Warning in adf.test(co2): p-value greater than printed p-value
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  co2
## Dickey-Fuller = -0.12404, Lag order = 9, p-value = 0.99
## alternative hypothesis: stationary
```

Based on the statistic and p-value, the data is non-stationary.

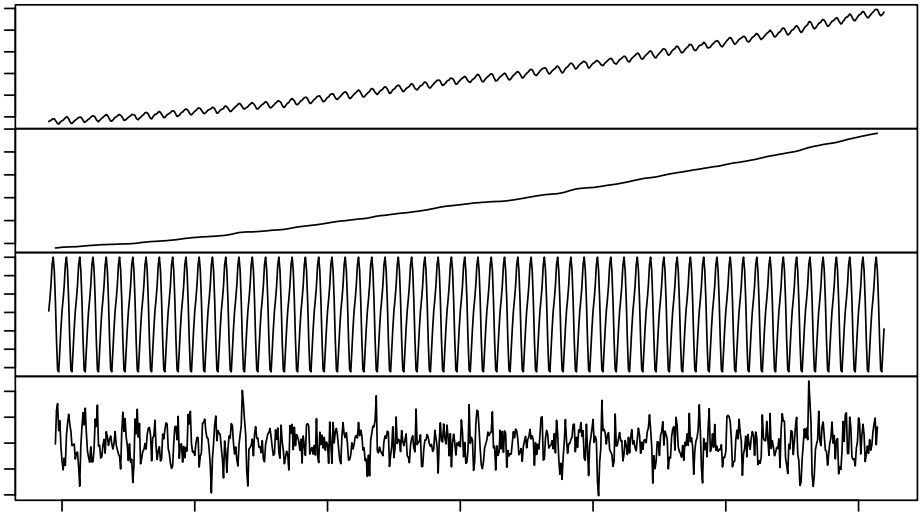
5.3 Decomposing the Time Series

We're interested in the seasonal, remainder, and specifically, trend components. We want to know what the data looks like when adjusted for seasonality. To do

this, we need to decompose our time series into its ETS components. Then, we can remove the seasonal component.

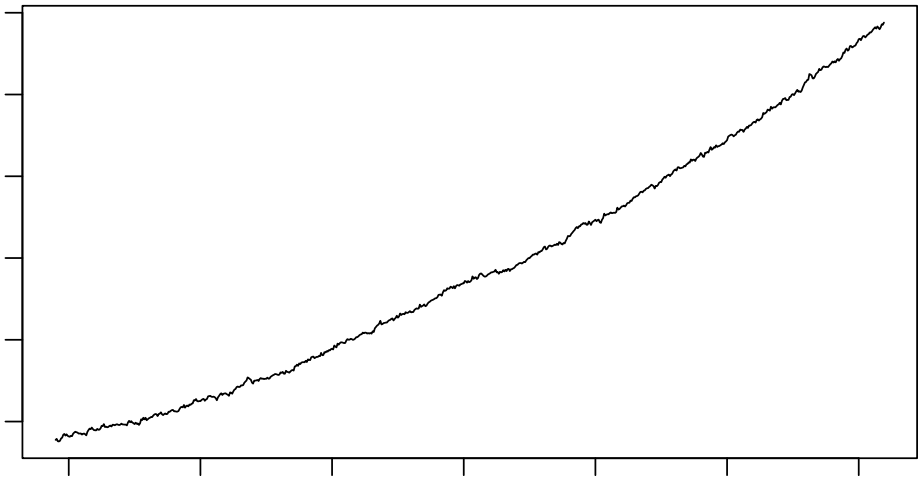
3. Decompose the time series and plot the resulting `decomposed.ts` object:

```
co2_decomp <- co2 %>%
  decompose()

plot(co2_decomp)
```



4. Then, adjust for seasonality, and plot the results. Label the y-axis as "CO2 Concentration", and title the plot "Seasonally-adjusted CO2 Concentration":

```
co2_decomp %>%
  seasadj() %>%

  # If we want to include a rolling window, it's really easy to do with
  # zoo::rollapplyr()

  #rollapplyr(FUN = mean, width = 12) %>%
  plot(ylab = "CO2 Concentration", main = "Seasonally-adjusted CO2 Concentration")
```



Voila! We now have the overall trend, plus remainder component, of CO2 concentrations. By removing the seasonality of data, we can better assess the year-to-year differences in our data, and can reduce the effect of intra- and inter-year cycles present in the data.

5.4 Example Code

This is the code given in the exercise itself

Step 1:

```
plot(___)

___.test(co2)
```

Step 2:

```
co2_impute <- co2 %>%
  __.___()

adf.test(co2_impute)
```

Step 3:

```
co2_decomp <- co2_impute %>%
  ___()

plot(___)
```

Step 4:

```
co2_decomp %>%
  ____() %>%
```

```
  plot(ylab = "___", main = "___")
```

Chapter 6

Course Outline: Manipulating Time Series Data in R

This course will introduce learners to working with time series data in R. Learners will explore how to store and format data in date and time objects as well as how to manipulate time series datasets through subsetting, indexing, and extraction. Examples of time series data across a variety of fields in business and science should be discussed. The course will cover summarization, frequency, missing data, resampling, and comparison techniques as well as window functions for both rolling and expanding windows.

Packages Used:

- `base` and `stats` (default libraries, but I wanted to name them explicitly)
- `zoo`

6.1 Chapter 1: Introduction to Time Series Data

- Lesson 1.1: *What is Time Series Data*
  - LO: Learner will be able to understand the foundations of time series data: rather than just analyzing a variable at different points in time, ts analysis studies *how* that variable changes with time.
- Lesson 1.2: *Interpreting a Time Series*
  - LO: Learner will be able to interpret a time series graph, understanding the x- and y-axes, trend, identifying periods, etc. at an

introductory level.

- Lesson 1.3: *Temporal data classes in R*
  - LO: Introduction to different formats for temporal data in R, such as the `Date`, `numeric`, and `character` formats:
    - e.g.: 2022-01-30, `19022`, and "2022-01-30" share the same information, but in different formats
  - LO: Learners will be able to check classes of data stored as vectors or as columns in a dataframe or tibble.
    - `class()`
- Lesson 1.4: *Converting between data classes*
  - LO: Learners will be able to convert between classes in R, such as converting a `character` vector to a `Date` vector
    - `as.Date()`
    - `as.numeric()`
    - `as.character()`

6.2 Chapter 2: Time Series objects in R

- Lesson 2.1: *How does R store Time Series Data?*
  - LO: Learners will be introduced to `ts` objects in R, and how they differ from objects like vectors or data frames
  - LO: Retrieve the temporal attributes (start, end, and frequency) of a time series object.
    - `start()`
    - `end()`
    - `frequency()`
- Lesson 2.2: *Create a Time Series object in Base R*
  - LO: Convert a vector of observations into a `ts` object, specifying start time and frequency
    - `ts()`
- Lesson 2.3: *Using the Zoo Package to store time series data*
  - LO: What is `zoo` and why is it different from base `ts`?
    - Zoo can use irregular time intervals
  - LO: Create and coerce time series objects with the `zoo` package:
    - `zoo::zoo()`
    - `zoo::as.zoo()`
- Lesson 2.4: *Using Zoo to extract time and data vectors*
  - LO: Extract "core data" and time data from a `ts` or `zoo` object:
    - `time()`
    - `zoo::coredata()`

6.3 Chapter 3: Subsetting, Extracting, and Resampling

- Lesson 3.1: *Subsetting a window of observations*
  - LO: Learner will be able to extract a window of observations between a set of time intervals
    - `window()`
    - `as.Date()`
    - `zoo::as.yearmon()`
  - LO: Use the `'['` operator with `as.Date()` to extract a specific date's observation
    - `'['`
    - `as.Date()`
    - `zoo::as.yearmon()`
- Lesson 3.2: *Retrieving observations by index*
  - LO: Use standard R `'['` operator to extract one or more observations by numerical index
    - `'['`
    - e.g.: `data[1:20]` retrieves observations 1 through 20
- Lesson 3.3: *Resampling observations*
  - LO: Learner will be able to resample observations to any interval of time (yearly, monthly, quarterly, etc.)
    - `aggregate()`
    - e.g.: `aggregate(data, nfrequency = 12, FUN = sum)` finds sums of observations within each month.
- Lesson 3.4: *Imputing Missing Values*
  - LO: Use the `zoo` package to impute missing values with either linear interpolation or cubic spline interpolation
    - `zoo::na.approx()` and `zoo::na.spline()`, respectively

6.4 Chapter 4: Rolling and Expanding Windows

- Lesson 4.1: *What are windows?*
  - LO: Learner will understand the utility of rolling and expanding windows: finding moving averages, cumulative sums, etc.
- Lesson 4.2: *Calculating a Rolling Window*
  - LO: Learner will be able to perform a rolling window operation on a time series, creating a moving average (or moving sum) of any length
    - `zoo::rollapply()`
    - `zoo::rollapplyr()` (convenience wrapper for `zoo::rollapply(align = "right")`)
    - e.g.: `zoo::rollapplyr(daily_data, FUN = mean, width = 7)` to create a 7-day rolling average from `daily_data`

- Lesson 4.3: *Calculating an Expanding Window*
  - LO: Learner will be able to create an expanding window: a rolling window where the "start" is fixed and the "end" moves
    - `cumsum()`
    - `seq_along()`
    - e.g.: `cumsum(x) / seq_along(x)` calculates a cumulative mean
    - *this function exists in* ***`dplyr::cummean()`****, but I didn't want an entire package dependency for something so simple*
- Lesson 4.4: *Plotting windows alongside Data*
  - LO: Learner will be able to plot the rolling/expanding window alongside the original data, in order to visually assess how these operations affect the data
    - `plot()`
    - `lines()`