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`
- `lubridate` (so far, only for the `lubridate::year()` function (Lesson 3.3))

Functions are written in `code text` below each **Learning Objective**

Chapter 1: Introduction to Time Series Data

Lesson 1.1: *What is Time Series Data*

- **Learning Objective:** Learner will be able to understand the foundations of time series data: rather than just analyzing a variable at different points in time, time series analysis studies *how* that variable changes with time.

Lesson 1.2: *Interpreting a Time Series*

- **Learning Objective:** Learner will be able to interpret a time series graph, understanding the x- and y-axes and identifying trends and periods at an introductory level.

Lesson 1.3: *Temporal data classes in R*

- **Learning Objective:** Learner will be introduced to different formats for temporal data in R, such as the `Date`, `numeric`, and `character` classes:
- e.g.: 2022-01-30, `19022`, and "2022-01-30" share the same information, but in different formats
- **Learning Objective:** Learners will be able to check classes of data stored as vectors or as columns in a dataframe or tibble; as some formats appear identical, it's important to understand the class of the data you're working with:
  - `class()`

Lesson 1.4: *Converting between data classes*

- **Learning Objective:** Learners will be able to convert between classes in R, such as converting a `character` vector/column to a `Date` vector/column:
  - `as.Date()`
  - `as.numeric()`

- `as.character()`

Chapter 2: Time Series objects in R

Lesson 2.1: *How does R store Time Series Data?*

- **Learning Objective:** Learners will be introduced to `ts` objects in R, and how they differ from objects like vectors or data frames in how they store data, and in how that data is displayed by R:
  - `print()` (specifically, `print.ts()`)
  - `plot()` (specifically, `plot.ts()`)
- **Learning Objective:** Learners will be able to retrieve the temporal attributes (start and end points, as well as frequency) of a time series object:
  - `start()`
  - `end()`
  - `frequency()`

Lesson 2.2: *Create a Time Series object in Base R*

- **Learning Objective:** Learners will convert a vector of observations with a known start time and frequency (e.g., monthly data starting in the year 2004) into a `ts` object:
  - `ts()`
  - `as.ts()`

Lesson 2.3: *Using the Zoo Package to store time series data*

- **Learning Objective:** Learners will be introduced to the `zoo` object from the `zoo` package, and why is it different from base `ts`:
  - Zoo can use irregular time intervals, more robust, etc.
- **Learning Objective:** Learners will be able to convert 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*

- **Learning Objective:** Learners can extract "core data" and time data from a `ts` or `zoo` object:
  - `time()`
  - `zoo::coredata()`

Chapter 3: Subsetting, Extracting, and Resampling

Lesson 3.1: *Subsetting a window of observations*

- **Learning Objective:** Learners will be able to extract a window of observations between a set of given points in time:
  - `window()`

  - `as.Date()`
  - `zoo::as.yearmon()`
- **Learning Objective:** Learners will use the `'['` operator with `as.Date()` to extract an observation from a specific time:
  - `'['`
  - `as.Date()`
  - `zoo::as.yearmon()`

Lesson 3.2: *Retrieving observations by index*

- **Learning Objective:** Learners will use the standard R functions to extract one or more observations by numerical index:
  - `'['`
  - `head()`
  - `tail()`
  - e.g.: `data[1:20]` retrieves observations 1 through 20, as does `head(data, n = 20)`

Lesson 3.3: *Resampling observations*

- **Learning Objective:** Learner will be able to re-sample observations to any interval of time (yearly, monthly, quarterly, etc.):
  - `aggregate()` (specifically, `aggregate.zoo()` for `zoo` objects)
  - `lubridate::year()`
  - `zoo::yearqtr()`
  - `zoo::yearmon()`
  - e.g.: `aggregate(data, by = lubridate::year, FUN = sum)` finds sums of observations within each year.

Lesson 3.4: *Imputing Missing Values*

- **Learning Objective:** Learners will use the `zoo` package to impute missing values with either linear interpolation or cubic spline interpolation:
  - `zoo::na.approx()` and `zoo::na.spline()`, respectively

Chapter 4: Rolling and Expanding Windows

Lesson 4.1: *What are windows?*

- **Learning Objective:** Learners will understand the utility of rolling and expanding windows: finding moving averages, cumulative sums, etc.

Lesson 4.2: *Calculating a Rolling Window*

- **Learning Objective:** Learners will be able to perform a rolling window operation on a time series, creating a moving average (or moving sum) of an arbitrary 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*

- **Learning Objective:** Learners will be able to create an expanding window – a rolling window where the "start" is fixed and the "end" moves:
  - `cumsum()`
  - `seq_along()`
  - `cumsum(data) / seq_along(data)` gives a rolling mean, which exists in `dplyr::cummean()` but not base R.

Lesson 4.4: *Plotting windows alongside Data*

- **Learning Objective:** Learners 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()`