Computer exercise 2: Time series

Data Analysis and Visualization / Måns Thulin

Contents

Preparations

In this computer exercise we will work with visualisations of time series data. We will also explore how to find and plot trends in data.

For this, we will use some datasets and functions from the `forecast`, `plotly`, `nlme` and `fpp2` packages. We start by loading the required packages:

```r
library(ggplot2)
library(forecast)
library(plotly)
library(nlme)
library(fpp2)
```

In case you don't already have them installed on your computer, you can install and load these packages using:

```r
install.packages(c("nlme", "plotly", "forecast", "fpp2"), dependencies = TRUE)
library(plotly)
library(forecast)
library(nlme)
library(fpp2)
```

In this exercise, we will study the following datasets:

- `a10`: total monthly anti-diabetic drug sales in Australia 1991-2008; from `fpp2`.
- `gold`: daily gold prices in the 1980's; from `fpp2`.
- `elecdaily`: daily electricity demand in Victoria, Australia in 2014; from `fpp2`.
- `msleep`: describing the sleep times of 83 mammals; from `ggplot2`.
- `Oxboys`: the height of a number of boys from Oxford, England, over time; from `nlme`.
- `writing`: sales of printing paper in France, 1963-1972; from `fpp2`.

Trends in scatterplots

Before we have a look at time series data, let's return to an example that we studied in the last computer exercise: the relationship between animal brain size and sleep times:

```r
ggplot(msleep, aes(brainwt, sleep_total)) +
      geom_point() +
      xlab("Brain weight (logarithmic scale)") +
      ylab("Total sleep time") +
      scale_x_log10()
```

There appears to be a decreasing trend in the plot. To aid the eye, we can add a smoothed line by adding a new geom, `geom_smooth`, to the figure:

```r
ggplot(msleep, aes(brainwt, sleep_total)) +
      geom_point() +
      geom_smooth() +
      xlab("Brain weight (logarithmic scale)") +
      ylab("Total sleep time") +
      scale_x_log10()
```

This technique is useful for bivariate data as well as for time series.

By default, `geom_smooth` adds a line fitted using either LOESS[1] or GAM[2], as well as the corresponding 95 % confidence interval. There are several useful arguments that can be used with `geom_smooth`. You will explore some of these in exercise below.

Exercise 1

Check the documentation for `geom_smooth`. Starting with the plot of brain weight vs. sleep time created above, do the following:

1. Decrease the degree of smoothing for the LOESS line that was fitted to the data. Which is better in this case, more or less smoothing?
2. Fit a straight line to the data instead of a non-linear LOESS line.
3. Remove the confidence interval from the plot.
4. Change the colour of the fitted line to red.

(Click here to go to the solution.)

Time series plots

The `a10` dataset contains information about the monthly anti-diabetic drug sales in Australia during the period July 1991 to June 2008. By checking its structure, we see that it is saved as a time series object:

```
library(fpp2)
str(a10)
```

`ggplot2` requires that data is saved as a data frame or tibble in order for it to be plotted. In order to plot the time series, we could first convert it to a data frame and then plot each point using `geom_points`:

```
a10_df <- data.frame(time = time(a10), sales = a10)
ggplot(a10_df, aes(time, sales)) +
      geom_point()
```

It is however usually preferable to plot time series using lines instead of points. This is done using a different geom: `geom_line`:

```
ggplot(a10_df, aes(time, sales)) +
      geom_line()
```

Having to convert the time series object to a data frame is a little awkward. Luckily, there is a way around this. `ggplot2` offers a function called `autoplot`, that automatically draws an appropriate plot for certain types of data. The `forecast` package extends this to include time series objects:

```
library(forecast)
autoplot(a10)
```

We can still add other geoms, axis labels and other things just as before. `autoplot` has simply replaced the `ggplot(data, aes()) + geom` part that would be the first two rows of the `ggplot2` figure.

[1]LOESS, LOcally Estimated Scatterplot Smoothing, is a non-parametric regression method that fits a polynomial to local areas of the data. (Click here to visit the Wikipedia page on LOESS)

[2]GAM, Generalised Additive Model, is a generalised linear model where the response variable is a linear function of smooth functions of the predictors. (Click here to visit the Wikipedia page on GAM)

Exercise 2

Using the `autoplot(a10)` figure, do the following:

1. Add a smoothed line describing the trend in the data. Make sure that it is smooth enough not to capture the seasonality in the data.
2. Change the label of the x-axis to "Year" and the label of the y-axis to "Sales ($ million)".
3. Check the documentation for the `ggtitle` function. What does it do? Use it with the figure.
4. Change the colour of the time series line to red.

(Click here to go to the solution.)

Annotations and reference lines

We sometimes wish to add text or symbols to plots, for instance to highlight interesting observations. Consider the following time series plot of daily morning gold prices:

```r
library(forecast); library(fpp2)
autoplot(gold)
```

There is a sharp spike a few weeks before day 800, which is due to an incorrect value in the data series. We'd like to add a note about that to the plot. First, we wish to find out on which day the spike appears. This can be done by checking the data manually, or using some code:

```r
spike_date <- which.max(gold)
```

To add a circle around that point, we add a call to `annotate` to the plot:

```r
autoplot(gold) +
      annotate(geom = "point", x = spike_date, y = gold[spike_date],
               size = 5, shape = 21, colour = "red", fill = "transparent")
```

Exercise 3

Using the figure created above and the documentation for `annotate`, do the following:

1. Add the text "Incorrect value" next to the circle.
2. Create a second plot where the incorrect value has been removed.
3. Read the documentation for the geom `geom_hline`. Use it to add a red reference line to the plot, at $y = 400$.

(Click here to go to the solution.)

Longitudinal data

Autoplot

Multiple time series with identical time points, known as longitudinal data or panel data, commonly occur in applications. One example of this is given by the `elecdaily` time series, which contains information about electricity demand in Victoria, Australia during 2014. As with a single time series, we can plot these data using `autoplot`:

```r
library(forecast); library(fpp2)
autoplot(elecdaily)
```

In this case, it is probably a good idea to facet the data, i.e. to plot each series in a separate figure:

```r
autoplot(elecdaily, facets = TRUE)
```

Exercise 4

Make the following changes to the plot you created above:

1. Remove the `WorkDay` variable from the plot (it describes whether or not a given date is a work day, and while it is useful for modelling purposes, we do not wish to include it in our figure).
2. Add smoothed trend lines to the time series plots.
3. You'll notice that the x-axis shows week numbers with decimals, rather than dates (the dates in the `elecdaily` time series object are formatted as weeks with decimal numbers). Make a time series plot of the `Demand` variable with dates (2014-01-01 to 2014-12-31) along the x-axis (your solution is likely to rely on standard R techniques rather than `autoplot`).

(Click here to go to the solution.)

Path plots

Another option for plotting multiple time series is path plots. A path plot is a scatterplot where the points are connected with lines in the order that they appear in the data (which, for time series data, should correspond to time). The lines and points can be coloured to represent time.

To make a path plot of Temperature versus Demand for the `elecdaily` data, we first convert the time series object to a data frame and create a scatterplot:

```r
library(fpp2)
ggplot(as.data.frame(elecdaily), aes(Temperature, Demand)) +
      geom_point()
```

Next, we connect the points by lines using the `geom_path` geom:

```r
ggplot(as.data.frame(elecdaily), aes(Temperature, Demand)) +
      geom_point() +
      geom_path()
```

The resulting figure is quite messy. Using colour to indicate the passing of time helps a little. For this, we need to add the dates to the data frame:

```r
elecdaily2 <- as.data.frame(elecdaily)
elecdaily2$Date <- seq.Date(as.Date("2014-01-01"), as.Date("2014-12-31"), by = "day")

ggplot(elecdaily2, aes(Temperature, Demand, colour = Date)) +
      geom_point() +
      geom_path()
```

It becomes clear from the plot that temperatures were the highest at the beginning of the year, and lower in the winter months (July-August).

Exercise 5

Make the following changes to the plot you created above:

1. Decrease the size of the points.
2. Add text annotations showing the dates of the highest and lowest temperatures, by the corresponding points in the figure.

(Click here to go to the solution.)

Spaghetti plots

In cases where we've observed multiple subjects over time, we often wish to visualise their individual time series together using so-called spaghetti plots. With `ggplot2` this is done using the `geom_line` geom. To illustrate this, we use the `Oxboys` data, showing the height of 26 boys over time.

```r
library(nlme)
ggplot(Oxboys, aes(age, height, group = Subject)) +
      geom_point() +
      geom_line()
```

The first two `aes` arguments specify the x and y-axes, and the third specifies that there should be one line per subject (i.e. per boy) rather than a single line interpolating all points. The latter would be a rather useless figure that looks like this:

```r
ggplot(Oxboys, aes(age, height)) +
      geom_point() +
      geom_line() +
      ggtitle("A terrible plot")
```

Returning to the original plot, if we wish to be able to identify which time series corresponds to which boy, we can add a colour aesthetic:

```r
ggplot(Oxboys, aes(age, height, group = Subject, colour = Subject)) +
      geom_point() +
      geom_line()
```

Note that the boys are ordered by height, rather than subject number, in the legend.

Now, imagine that we wish to add a trend line describing the general growth trend for all boys. The growth appears approximately linear, so it seems sensible to use `geom_smooth(method = "lm")` to add the trend:

```r
ggplot(Oxboys, aes(age, height, group = Subject, colour = Subject)) +
      geom_point() +
      geom_line() +
      geom_smooth(method = "lm", colour = "red", se = FALSE)
```

Unfortunately, because we have specified in the aestethics that the data should be grouped by `Subject`, `geom_smooth` produces one trend line for each boy. The "problem" is that when we specify an aesthetic in the `ggplot` call, it is used for all geoms.

Exercise 6

Figure out how to produce a spaghetti plot of the `Oxboys` data with a single red trend line based on the data from all 26 boys.

(Click here to go to the solution.)

Seasonal plots, decomposition and ACF

The `forecast` packages includes a number of useful functions when working with time series. One of them is `ggseasonplot`, which allows us to easily create a spaghetti plot of different periods of a time series with seasonality, i.e. with patterns that repeat seasonally over time. It works similar to the `autoplot` function, in that it replaces the `ggplot(data, aes) + geom` part of the code.

```r
library(forecast); library(fpp2)
ggseasonplot(a10)
```

This function is very useful when visually inspecting seasonal patterns.

The `year.labels` and `year.labels.left` arguments removes the legend in favour of putting the years at the end and beginning of the lines:

```r
ggseasonplot(a10, year.labels = TRUE, year.labels.left = TRUE)
```

As always, we can add more things to our plot if we like:

```r
ggseasonplot(a10, year.labels = TRUE, year.labels.left = TRUE) +
      ylab("Sales ($ million)") +
      ggtitle("Seasonal plot of anti-diabetic drug sales")
```

When working with seasonal time series, it is common to decompose the series into a seasonal component, a trend component and a remainder. In R, this is typically done using the `stl` function, which uses repeated LOESS smoothing to decompose the series. There is an `autoplot` function for `stl` objects:

```r
autoplot(stl(a10, s.window = 365))
```

Similarly, there is an `autoplot` function for auto-correlation functions (ACF) of time series:

```r
autoplot(acf(a10))
```

These can be manipulated in the same way as other `ggplot` objects.

Exercise 7

Investigate the `writing` dataset graphically. Make a time series plot with a smoothed trend line, a seasonal plot and a stl-decomposition plot. Add appropriate plot titles and labels to the axes.

(Click here to go to the solution.)

Interactive time series plots

The `plotly` packages can be used to create interactive time series plots. When working with `plotly`, you create a `ggplot2` object as usual, assigning it to a variable and then call the `ggplotly` function. Here is an example with the `elecdaily` data:

```r
library(forecast)
library(fpp2)
library(plotly)

myPlot <- autoplot(elecdaily[,"Demand"])

ggplotly(myPlot)
```

When you hover the mouse pointer over a point, a box appears, which displays information about that data point. Unfortunately, the date formatting isn't great in this example - dates are shown as weeks with decimal points. We fix this in the same way as before, i.e. by creating a new data frame with nicely formatted dates:

```r
# First, create a data frame with better formatted dates
elecdaily2 <- as.data.frame(elecdaily)
elecdaily2$Date <- seq.Date(as.Date("2014-01-01"), as.Date("2014-12-31"), by = "day")

# Create the plot object
myPlot <- ggplot(elecdaily2, aes(Date, Demand)) +
      geom_line()

# Create the interactive plot
ggplotly(myPlot)
```

Exercise 8

Create an interactive version time series plot of the `a10` anti-diabetic drug sales data. Make sure that the dates are correctly displayed.

(Click here to go to the solution.)

Preparations for the next computer exercise

In the next computer exercise, we will use some datasets and functions from the `dplyr`, `nycflights13` and `ggridges` packages. Ahead of the next computer exercise, you should therefore install them:

```r
install.packages(c("dplyr", "ggridges", "nycflights13"), dependencies = TRUE)
```

Solutions

Exercise 1

To decrease the smoothness of the line, we use the `span` argument in `geom_smooth`. The default is `geom_smooth(span = 0.75)`. Decreasing this values yields a very different fit:

```r
ggplot(msleep, aes(brainwt, sleep_total)) +
      geom_point() +
      geom_smooth(span = 0.25) +
      xlab("Brain weight (logarithmic scale)") +
      ylab("Total sleep time") +
      scale_x_log10()
```

More smoothing is probably preferable in this case. The relationship appears to be fairly weak, and appears to be roughly linear.

We can use the `method` argument in `geom_smooth` to fit a straight line using `lm` instead of LOESS:

```r
ggplot(msleep, aes(brainwt, sleep_total)) +
      geom_point() +
      geom_smooth(method = "lm") +
      xlab("Brain weight (logarithmic scale)") +
      ylab("Total sleep time") +
      scale_x_log10()
```

To remove the confidence interval from the plot, we set `se = FALSE` in `geom_smooth`:

```r
ggplot(msleep, aes(brainwt, sleep_total)) +
      geom_point() +
      geom_smooth(method = "lm", se = FALSE) +
      xlab("Brain weight (logarithmic scale)") +
      ylab("Total sleep time") +
      scale_x_log10()
```

Finally, we can change the colour of the smoothing line using the `colour` argument:

```r
ggplot(msleep, aes(brainwt, sleep_total)) +
      geom_point() +
      geom_smooth(method = "lm", se = FALSE, colour = "red") +
      xlab("Brain weight (logarithmic scale)") +
      ylab("Total sleep time") +
      scale_x_log10()
```

(Click here to return to the exercise.)

Exercise 2

Adding the `geom_smooth` geom with the default settings produces a trend line that does not capture seasonality:

```r
autoplot(a10) +
      geom_smooth()
```

We can change the axes labels using `xlab` and `ylab`:

```r
autoplot(a10) +
      geom_smooth() +
      xlab("Year") +
      ylab("Sales ($ million)")
```

`ggtitle` adds a title to the figure:

```r
autoplot(a10) +
      geom_smooth() +
      xlab("Year") +
      ylab("Sales ($ million)") +
      ggtitle("Anti-diabetic drug sales in Australia")
```

The `colour` argument can be passed to `autoplot` to change the colour of the time series line:

```r
autoplot(a10, colour = "red") +
      geom_smooth() +
      xlab("Year") +
      ylab("Sales ($ million)") +
      ggtitle("Anti-diabetic drug sales in Australia")
```

(Click here to return to the exercise.)

Exercise 3

The text can be added by using `annotate(geom = "text", ...)`. In order not to draw the text on top of the circle, you can shift the x-value of the text (the appropriate shift depends on the size of your plot window):

```r
autoplot(gold) +
      annotate(geom = "point", x = spike_date, y = gold[spike_date],
               size = 5, shape = 21, colour = "red", fill = "transparent") +
      annotate(geom = "text", x = spike_date - 100, y = gold[spike_date],
               label = "Incorrect value!")
```

We can remove the erroneous value by replacing it with `NA` in the time series:

```r
gold[spike_date] <- NA
autoplot(gold)
```

Finally, we can add a reference line using `geom_hline`:

```r
autoplot(gold) +
      geom_hline(yintercept = 400, colour = "red")
```

(Click here to return to the exercise.)

Exercise 4

We can specify which variables to include in the plot as follows:

```r
autoplot(elecdaily[, c("Demand", "Temperature")], facets = TRUE)
```

This produces a terrible-looking label for the y-axis, which we can remove by setting the y-label to `NULL`:

```r
autoplot(elecdaily[, c("Demand", "Temperature")], facets = TRUE) +
      ylab(NULL)
```

As before, we can add smoothers using `geom_smooth`:

```r
autoplot(elecdaily[, c("Demand", "Temperature")], facets = TRUE) +
      geom_smooth() +
      ylab(NULL)
```

The x-axis of the data can be changed in multiple ways. A simple approach is the following:

```r
# Create a new data frame with the correct dates and the demand data:
dates <- seq.Date(as.Date("2014-01-01"), as.Date("2014-12-31"), by = "day")
elecdaily2 <- data.frame(dates = dates, demand = elecdaily[,1])

ggplot(elecdaily2, aes(dates, demand))
      + geom_line()
```

A more elegant approach relies on the `xts` package for time series:

```r
library(xts)
# Convert time series to an xts object:
dates <- seq.Date(as.Date("2014-01-01"), as.Date("2014-12-31"), by = "day")
elecdaily3 <- xts(elecdaily, order.by = dates)

autoplot(elecdaily3[,"Demand"])
```

(Click here to return to the exercise.)

Exercise 5

We set the size of the points using `geom_point(size)`:

```r
ggplot(elecdaily2, aes(Temperature, Demand, colour = Date)) +
      geom_point(size = 0.5) +
      geom_path()
```

To add annotations, we use `annotate` and some code to find the dates of the lowest and highest temperatures:

```r
# Lowest temperature
lowest <- which.min(elecdaily2$Temperature)

# Highest temperature
highest <- which.max(elecdaily2$Temperature)

# We shift the y-values of the text so that it appears above the points
ggplot(elecdaily2, aes(Temperature, Demand, colour = Date)) +
      geom_point(size = 0.5) +
      geom_path() +
      annotate(geom = "text", x = elecdaily2$Temperature[lowest],
               y = elecdaily2$Demand[lowest] + 4,
               label = elecdaily2$Date[lowest]) +
      annotate(geom = "text", x = elecdaily2$Temperature[highest],
               y = elecdaily2$Demand[highest] + 4,
               label = elecdaily2$Date[highest])
```

(Click here to return to the exercise.)

Exercise 6

We can specify aes(group) for a particular geom as follows:

```
ggplot(Oxboys, aes(age, height, colour = Subject)) +
      geom_point() +
      geom_line(aes(group = Subject)) +
      geom_smooth(method = "lm", colour = "red", se = FALSE)
```

Subject is now used for grouping the points used to draw the lines (i.e. for geom_line), but not for geom_smooth, which now uses all the points to create a trend line showing the average height of the boys over time.

(Click here to return to the exercise.)

Exercise 7

Code for producing the three plots is given below:

```
# Time series plot
autoplot(writing) +
      geom_smooth() +
      ylab("Sales (francs)") +
      ggtitle("Sales of printing and writing paper")

# Seasonal plot
ggseasonplot(writing, year.labels = TRUE, year.labels.left = TRUE) +
      ylab("Sales (francs)") +
      ggtitle("Seasonal plot of sales of printing and writing paper")
# There is a huge dip in sales in August, when many French offices are closed
# due to holidays.

# stl-decomposition
autoplot(stl(writing, s.window = 365)) +
      ggtitle("Seasonal decomposition of paper sales time series")
```

(Click here to return to the exercise.)

Exercise 8

```
# First, create a data frame with better formatted dates
a102 <- as.data.frame(a10)
a102$Date <- seq.Date(as.Date("1991-07-01"), as.Date("2008-06-01"), by = "month")

# Create the plot object
myPlot <- ggplot(a102, aes(Date, x)) +
      geom_line() +
      xlab("Sales")

# Create the interactive plot
ggplotly(myPlot)
```

(Click here to return to the exercise.)